Machine Learning and Scientific Computing

Thesis by
Nikola Borislavov Kovachki

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

Caltech

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2022
Defended May 3, 2022

ACKNOWLEDGEMENTS

To Jiaxin Li for being my source of inspiration and drive, and for her unfaltering support even during the hardest times. To my parents Magdalina Zulamska-Kovachka and Borislav Kovachki for always believing in me, allowing and pushing me to pursue my interests, and for being by my side whenever I need them. To my mentor Andrew M. Stuart for patiently teaching me everything I know about math, allowing me the freedom to explore, and for his incredible kindness. To my numerous collaborators for giving me the chance to learn from them. This thesis would not have been possible without any of you. Thank you!

ABSTRACT

The remarkable success of machine learning methods for tacking problems in computer vision and natural language processing has made them auspicious tools for applications to scientific computing tasks. The present work advances both machine learning techniques by using ideas from numerical analysis, inverse problems, and data assimilation and introduces new machine learning based tools for accurate and computationally efficient scientific computing. Chapters 2 and 3 introduce new methods and analyze existing methods for the optimization of deep neural networks. Chapters 4 and 5 formulate approximation architectures acting between infinite dimensional functions spaces for applications to parametric PDE problems. Chapter 6 demonstrates how to re-formulate GAN(s) so they can condition on continuous data and exhibits applications to Bayesian inverse problems. In Chapter 7, we present a novel regression-clustering method and apply it to the problem of predicting molecular activation energies.

PUBLISHED CONTENT AND CONTRIBUTIONS

Bhattacharya, Kaushik et al. (2021). "Model Reduction And Neural Networks For Parametric PDEs". In: *The SMAI journal of computational mathematics* 7, pp. 121–157. URL: https://smai-jcm.centre-mersenne.org/articles/10.5802/smai-jcm.74/.
N.B.K. participated in the conception of the project, proved all theoretical results expect for the locally Lipschitz case, generated data for all numerical problems, coded and ran all numerical experiments, and wrote the majority of the manuscript.

Kovachki, Nikola B, Zongyi Li, et al. (2021). "Neural Operator: Learning Maps Between Function Spaces". In: *CoRR* abs/2108.08481. URL: https://arxiv.org/abs/2108.08481.
N.B.K. participated in the conception of the project, conceptualized and proved all theoretical results, generated data for all numerical problems except the advection equation, wrote code for the GNO, LNO, and FNO models, ran some numerical experiments for the Burgers' equation, Darcy flow, and the Navier-Stokes equation and all experiments concerning the Bayesian inverse problem, and wrote the majority of the manuscript excluding the numerical results section.

Kovachki, Nikola B and Andrew M Stuart (2021). "Continuous Time Analysis of Momentum Methods". In: *Journal of Machine Learning Research* 22.17, pp. 1–40. URL: https://www.jmlr.org/papers/v22/19-466.html.
N.B.K. participated in the conception of the project, proved all theoretical results, coded and ran all numerical experiments, and wrote the majority of the manuscript.

Kovachki, Nikola B, Ricardo Baptista, et al. (2020). "Conditional Sampling with Monotone GANs". In: *CoRR* abs/2006.06755. URL: https://arxiv.org/abs/2006.06755.
N.B.K. participated in the conception of the project, conceptualized MGAN(s) and their training algorithm, coded and ran all numerical experiments except the section "Block triangular versus triangular maps," and participated in writing and editing the manuscript.

Cheng, Lixue et al. (2019). "Regression Clustering for Improved Accuracy and Training Costs with Molecular-Orbital-Based Machine Learning". In: *Journal of Chemical Theory and Computation* 15.12, pp. 6668–6677. URL: https://pubs.acs.org/doi/abs/10.1021/acs.jctc.9b00884.
N.B.K. participated in the conception of the project, conceptualized and implemented the regression-clustering algorithm, coded and ran numerical experiments on the QM7b-T dataset, and wrote the "Regression clustering with a greedy algorithm" section of the manuscript.

Kovachki, Nikola B and Andrew M Stuart (2019). "Ensemble Kalman Inversion: a Derivative-free Technique for Machine Learning Tasks". In: *Inverse Problems* 35.9, p. 095005. URL: https://iopscience.iop.org/article/10.

`1088/1361-6420/ab1c3a`.
N.B.K. participated in the conception of the project, conceptualized and coded all the proposed parts of the algorithm, collected the data and ran all numerical experiments, and wrote the majority of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

With the advent of computers, the fields of machine learning and scientific computing have developed separately but in parallel over the last eight decades. While seemingly different the two fields have a common unifying goal: model, understand, and harness natural and worldly phenomena through computational tools. While scientific computing has traditionally focused on solving and understanding physics-based models, machine learning generally operates in areas where there are no known good models and rather tries to build them through data. Due to this difference, the mathematical theories that underpin the two fields have developed distinctly and resulted in the conception of diverse sets of numerical methods. Scientific computing is based on numerical analysis where one studies model approximations through their convergence to a continuum physical model. On the other hand, machine learning is based on statistical learning theory where one studies model approximations through their convergence to an unknown statistical model defined via the observed data. By re-interpreting the problems in each field through the mathematical lens of the other, it becomes apparent that many similar and mutually useful ideas have emerged, opening the door for new theoretical analysis and numerical techniques that hold the potential for revolutionizing both fields. The present work takes a step in this direction by introducing ideas from numerical analysis to study and improve machine learning methods while also applying approximation and optimization techniques from machine learning to tackle computationally difficult physical problems.

In Chapter 2, we re-cast the problem of learning from data as a general inverse problem more commonly studied in the context of inferring unknown model parameters through few physical observations. This allows us to apply the successful Ensemble Kalman Inversion algorithm to the problem of training deep neural networks as well as to other semi-supervised learning problems. We propose several modifications to the algorithm inspired by known optimization techniques, resulting in a fully-parallelizable and derivative-free method for solving general learning problems. Continuing via the lens of optimization, in Chapter 3, we study popular momentum-based methods as applied in deep learning through a continuum interpretation. We show that, to first order, these methods converge to a re-scaled gradient flow and all momentum-like effects emerge at the scale of the learning rate. This gives a unified view of momentum methods and a precise characterization of their deviation from the standard gradient decent.

In Chapters 4 and 5, we focus on developing architectures which approximate maps between two functions space, i.e. operators. In particular, we exhibit methods which can consistently operate on functions discretized arbitrarily while keeping fixed their number of parameters. This allows for the design of architectures that are provably consistent with an underlying physical model. While our primary application for this endeavor is demonstrating computationally fast and accurate methods for solving parametric PDE(s), its usability can straightforwardly be extended to video, audio, and image data. Chapter 4 introduces the concept of operator learning and introduces a new approximation method inspired by classical model reduction techniques. Universal approximation theorems for the architecture are proved when mapping between infinite-dimensional Hilbert spaces. Chapter 5 describers a natural way of lifting standard neural networks to operators. Universal approximation theorems for the architecture are proved when mapping between infinite-dimensional Banach spaces. Our work conceptualizes a new type of universal approximation theorem in which error is measured with respect to a Bochner norm, closer to the statistical learning setting, as opposed to a uniform error over compact sets as is classically done in approximation theory. For the problems considered, while being accurate, the methods are also significantly faster than traditional approach for solving PDE(s). Our methodology bridges the gap between machine learning and the approximation of continuum physical models.

In the last two chapters, we focus on developing methods for approximating problems where the underlying model is polluted by noise or is intractable to compute. We work in a finite-dimensional setting, leaving extensions to infinite dimensions to future work. Chapter 6 introduces the concept of a Monotone Generate Adversarial Network (MGAN) which is a method that approximates the pushforward map from a known reference probability measure to a joint data measure of inputs and outputs. The architecture is designed so that it is trivial to extract a pushforward map to any conditional of the joint measure, realizing a GAN that is able to condition on continuous data. Our method gives a computationally cheap way of sampling the posterior distribution to Bayesian inverse problems or quantifying the uncertainty in noisy forward problems. In Chapter 7, we tackle the problem of cheaply approximating the ground-state activation energies of molecules from features based on the Hartree–Fock theory. For molecules with many electrons, accurate physical approximations become computationally intractable, making data-driven methods an indispensable tool. We formulate a novel regression-clustering approach based on the observation that the data exhibits a piece-wise linear structure. Combined with Gaussian Process regression, our method achieves state-of-the-art results on standard benchmarks containing molecules with up to thirteen heavy atoms.

Due to the broad scope of the present work, targeting researchers in applied mathematics, statistics, physics, chemistry, and engineering, each chapter sets forth separate notation outlined therein.

Chapter 2

ENSEMBLE KALMAN INVERSION FOR MACHINE LEARNING

2.1 Introduction

The Setting

The field of machine learning has seen enormous advances over the last decade. These advances have been driven by two key elements: (i) the introduction of flexible architectures which have the expressive power needed to efficiently represent the input-output maps encountered in practice; (ii) the development of smart optimization tools which train the free parameters in these input-output maps to match data. The text (Goodfellow, Bengio, and Courville, 2016) overviews the start-of-the-art.

While there is little work in the field of derivative-free, paralellizable methods for machine learning tasks, such advancements are greatly needed. Variants on the Robbins-Monro algorithm (Robbins and Monro, 1951), such as stochastic gradient descent (SGD), have become state-of-the-art for practitioners in machine learning (Goodfellow, Bengio, and Courville, 2016) and an attendant theory (Dieuleveut, Bach, et al., 2016; Bach and Moulines, 2013; Schmidt, Le Roux, and Bach, 2017; Lee et al., 2016; Jordan, 2017) is emerging. However the approach faces many challenges and limitations (Glorot and Bengio, 2010; Pascanu, Mikolov, and Bengio, 2013; Taylor et al., 2016). New directions are needed to overcome them, especially for parallelization, as attempts to parallelize SGD have seen limited success (Zhang, Choromanska, and LeCun, 2015).

A step in the direction of a derivative-free, parallelizable algorithm for the training of neural networks was attempted in (Carreira-Perpinan and W. Wang, 2014) by use of the the method of auxiliary coordinates (MAC). Another approach using the alternating direction method of multipliers (ADMM) and a Bregman iteration is attempted in (Taylor et al., 2016). Both methods seem successful but are only demonstrated on supervised learning tasks with shallow, dense neural networks that have relatively few parameters. In reinforcement learning, genetic algorithm have seen some success (see (Such et al., 2017) and references therein), but it is not clear how to deploy them outside of that domain.

To simultaneously address the issues of parallelizable and derivative-free optimization, we demonstrate in this paper the potential for using ensemble Kalman methods to undertake machine learning tasks. Optimizing neural networks via Kalman filtering has been attempted before (see (Haykin, 2001) and references therein), but most have been through the use of extended or unscented Kalman Filters. Such methods are plagued by inescapable computational and memory constraints and hence their application has been restricted to small parameter models. A contemporaneous paper (Haber, Lucka, and Ruthotto, 2018) has introduced a variant on the ensemble Kalman filter, and applied it to the training of neural networks; our paper works with a more standard implementations of ensemble Kalman methods for filtering and inversion (K. Law, Andrew Stuart, and Zygalakis, 2015; M. A. Iglesias, K. J. H. Law, and Andrew M Stuart, 2013) and demonstrates potential for these methods within a wide range of machine learning tasks, when suitably enhanced by ideas that have become routine in the successful implementation of SGD and its variants.

Our Contribution

The goal of this work is two-fold:

- First we show that many of the common tasks considered in machine learning can be formulated in the unified framework of Bayesian inverse problems. The advantage of this point of view is that it allows for the transfer of theory and algorithms developed for inverse problems to the field of machine learning, in a manner accessible to the inverse problems community. To this end we give a precise, mathematical description of the most common approximation architecture in machine learning, the neural network (and its variants); we use the language of dynamical systems, and avoid references to the neurobiological language and notation more common-place in the applied machine learning literature. We do not pursue uncertainty quantification in this paper, but the framework in which we work will allow for this in the future. Work taking the deterministic viewpoint of inverse problems has already been pursued in (De Vito et al., 2005).

- Secondly, adopting the inverse problem point of view, we show that variants of ensemble Kalman methods (EKI, EnKF) can be just as effective at solving most machine learning tasks as the plethora of gradient-based methods that are widespread in the field. We borrow some ideas from machine learning and optimization to modify these ensemble methods, in order to enhance their performance. In short we develop algorithms which do not require backpropagation, but instead use an ensemble based approach to leverage sensitivities to perform parameter estimation; the method is inherently parallelizable.

Our belief is that by formulating machine learning tasks as inverse problems, and by demonstrating the potential for methodologies to be transferred from the field of inverse problems to machine learning, we will open up new ways of thinking about machine learning which may ultimately lead to deeper understanding of the optimization tasks at the heart of the field, and to improved methodology for addressing those tasks. To substantiate the second assertion we give examples of the competitive application of ensemble methods to supervised, semi-supervised, and online learning problems with deep dense, convolutional, and recurrent neural networks. To the best of our knowledge, this is the first paper to successfully apply ensemble Kalman methods to such a range of relatively large scale machine learning tasks. Furthermore we explicitly demonstrate that the method may be used on architectures for which backpropagation is not possible, for example in training a dense neural network with Heaviside activations on FashionMNIST. Whilst we do not attempt parallelization, ensemble methods are easily parallelizable and we give references to relevant literature. Our work leaves many open questions and future research directions for the inverse problems community.

Notation and Overview

We adopt the notation $\mathbb{R}$ for the real axis, $\mathbb{R}_+$ the subset of non-negative reals, and $\mathbb{N} = \{0, 1, 2, \dots\}$ for the set of natural numbers. For any set A, we use A^n to denote its n-fold Cartesian product for any $n \in \mathbb{N} \setminus \{0\}$. For any function $f : A \to B$, we use $\text{Im}(f) = \{y \in B : y = f(x), \text{ for, for}, x \in A\}$ to denote its image. For any subset $V \subseteq \mathcal{X}$ of a linear space $\mathcal{X}$, we let $\dim V$ denote the dimension of the smallest subspace containing V. For any Hilbert space $\mathcal{H}$, we adopt the notation $\|\cdot\|_{\mathcal{H}}$ and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ to be its associated norm and inner-product respectively. Furthermore for any symmetric, positive-definite operator $C : \mathcal{D}(C) \subset \mathcal{H} \to \mathcal{H}$, we use the notation $\|\cdot\|_C = \|C^{-\frac{1}{2}} \cdot\|_{\mathcal{H}}$ and $\langle \cdot, \cdot \rangle_C = \langle C^{-\frac{1}{2}} \cdot, C^{-\frac{1}{2}} \cdot, \rangle_{\mathcal{H}}$. For any two topological spaces $\mathcal{X}, \mathcal{Y}$, we let $C(\mathcal{X}, \mathcal{Y})$ denote the set of continuous functions from $\mathcal{X}$ to $\mathcal{Y}$. We define

$$\mathbb{P}^m = \{y \in \mathbb{R}^m \mid \|y\|_1 = 1, y_1, \dots, y_m \geq 0\}$$

the set of m-dimensional probability vectors, and the subset

$$\mathbb{P}^m_0 = \{y \in \mathbb{R}^m \mid \|y\|_1 = 1, y_1, \dots, y_m > 0\}.$$

Section 2.2 delineates the learning problem, starting from the classical, optimization-based framework, and shows how it can be formulated as a Bayesian inverse problem. Section 2.3 gives a brief overview of modern neural network architectures as dynamical systems. Section 2.4 outlines the state-of-the-art algorithms for fitting neural network models, as well as the EKI method and our proposed modifications of it. Section 2.5 presents our numerical experiments, comparing and contrasting EKI methods with the state-of-the-art. Section 2.6 gives some concluding remarks and possible future directions for this line of work.

2.2 Problem Formulation

Subsection 2.2 overviews the standard formulation of machine learning problems with subsections 2.2, 2.2, and 2.2 presenting supervised, semi-supervised, and online learning respectively. Subsection 2.2 sets forth the Bayesian inverse problem interpretation of these tasks and gives examples for each of the previously presented problems.

Classical Framework

The problem of supervised learning is usually formulated as minimizing an expected cost over some space of mappings relating the data (Goodfellow, Bengio, and Courville, 2016; Vapnik, 1995; Murphy, 2012). More precisely, let $\mathcal{X}$, $\mathcal{Y}$ be separable Hilbert spaces and let $\mathbb{P}(x, y)$ be a probability measure on the product space $\mathcal{X} \times \mathcal{Y}$. Let $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a positive-definite function and define $\mathcal{F}$ to be the set of mappings $\{\mathcal{G} : \mathcal{X} \to \mathcal{Y}\}$ on which the composition $\mathcal{L}(\mathcal{G}(\cdot), \cdot)$ is $\mathbb{P}$-measurable for all $\mathcal{G}$ in $\mathcal{F}$. Then we seek to minimize the functional

$$Q(\mathcal{G}) = \int_{\mathcal{X}\times\mathcal{Y}} \mathcal{L}(\mathcal{G}(x), y)\; d\mathbb{P}(x, y) \tag{2.1}$$

across all mappings in $\mathcal{F}$. This minimization may not be well defined as there could be infimizing sequences not converging in $\mathcal{F}$. Thus further constraints (regularization) are needed to obtain an unambiguous optimization problem. These are generally introduced by working with parametric forms of $\mathcal{G}$. Additional explicit regularization is also often added to parameterized versions of (2.1).

Usually $\mathcal{L}$ is called the loss or cost function and acts as a metric-like function on $\mathcal{Y}$; however it is useful in applications to relax the strict properties of a metric, and we, in particular, do not require $\mathcal{L}$ to be symmetric or subadditive. With this interpretation of $\mathcal{L}$ as a cost, we are seeking a mapping $\mathcal{G}$ with lowest cost, on average with respect to $\mathbb{P}$. There are numerous choices for $\mathcal{L}$ used in applications (Goodfellow, Bengio, and Courville, 2016); some of the most common include the squared-error loss $\mathcal{L}(y', y) = \|y - y'\|_{\mathcal{Y}}^2$ used for regression tasks, and the cross-entropy loss $\mathcal{L}(y', y) = -\langle y, \log y' \rangle_{\mathcal{Y}}$ used for classification tasks. In both these cases we often have $\mathcal{Y} = \mathbb{R}^K$, and, for classification, we may restrict the class of mappings to those taking values in $\mathbb{P}_0^K$.

Most of our focus will be on *parametric* learning where we approximate $\mathcal{F}$ by a parametric family of models $\{\mathcal{G}(u|\cdot) : \mathcal{X} \to \mathcal{Y}\}$ where $u \in \mathcal{U}$ is the parameter and $\mathcal{U}$ is a separable Hilbert space. This allows us to work with a computable class of functions and perform the minimization directly over $\mathcal{U}$. Much of the early work in machine learning focuses on model classes which make the associated minimization problem convex (Boser, Guyon, and Vapnik, 1992; Hoerl and Kennard, 1970; Murphy, 2012), but the recent empirical success of neural networks has driven research away from this direction (LeCun, Bengio, and Geoffrey E. Hinton, 2015; Goodfellow, Bengio, and Courville, 2016). In Section 2.3, we give a brief overview of the model space of neural networks.

While the formulation presented in (2.1) is very general, it is not directly transferable to practical applications as, typically, we have no direct access to $\mathbb{P}(x, y)$. How we choose to address this issue depends on the information known to us, usually in the form of a data set, and defines the type of learning. Typically information about $\mathbb{P}$ is accessible only through our sample data. The next three subsections describe particular structures of such sample data sets which arise in applications, and the minimization tasks which are constructed from them to determine the parameter u.

Supervised Learning

Suppose that we have a dataset $\{(x_j, y_j)\}_{j=1}^N$ assumed to be i.i.d. samples from $\mathbb{P}(x, y)$. We can thus replace the integral (2.1) with its Monte Carlo approximation, and add a regularization term, to obtain the following minimization problem:

$$\arg\min_{u \in \mathcal{U}} \Phi_{\mathrm{s}}(u; \mathsf{x}, \mathsf{y}), \tag{2.2}$$

$$\Phi_{\mathrm{s}}(u; \mathsf{x}, \mathsf{y}) = \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}(\mathcal{G}(u|x_j), y_j) + R(u). \tag{2.3}$$

Here $R : \mathcal{U} \to \mathbb{R}$ is a regularizer on the parameters designed to prevent overfitting or address possible ill-posedness. We use the notation $\mathsf{x} = [x_1, \dots, x_N] \in \mathcal{X}^N$, and analogously y, for concatenation of the data in the input and output spaces $\mathcal{X}, \mathcal{Y}$ respectively.

A common choice of regularizer is $R(u) = \lambda \|u\|_{\mathcal{U}}^2$ where $\lambda \in \mathbb{R}$ is a tunable parameter. This choice is often called *weight decay* in the machine learning literature; the regularizer promotes parameters with smaller norm, thus inducing decay when represented in specific bases. Other choices, such as sparsity promoting norms, are also employed; carefully selected choices of the norm can induce desired behavior in the parameters (E. J. Candes and Tao, 2006; Vogel, 2002). We note also that Monte Carlo approximation is itself a form of regularization of the minimization task (2.1).

This formulation is known as *supervised* learning. Supervised learning is perhaps the most common type of machine learning with numerous applications including image/video classification, object detection, and natural language processing (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012; Manning and Schütze, 1999; Sutskever, Vinyals, and Le, 2014).

Semi-Supervised Learning

Suppose now that we only observe a small portion of the data y in the image space; specifically we assume that we have access to data $\{x_j\}_{j \in Z}$, $\{y_j\}_{j \in Z'}$ where $x_j \in \mathcal{X}, y_j \in \mathcal{Y}$, $Z = \{1, \dots, N\}$ and where $Z' \subset Z$ with $|Z'| \ll |Z|$. Clearly this can be turned into supervised learning by ignoring all data indexed by $Z \setminus Z'$, but we would like to take advantage of all the information known to us. Often the data in $\mathcal{X}$ is known as unlabeled data, and the data in $\mathcal{Y}$ as labeled data; in particular the labeled data is often in the form of categories. We use the terms labeled and unlabeled in general, regardless or whether the data in $\mathcal{Y}$ is categorical; however some of our illustrative discussion below will focus on the binary classification problem. The objective is to assign a label y_j to every $j \in Z$. This problem is known as *semi-supervised* learning.

One approach to the problem is to seek to minimize

$$\arg\min_{u \in \mathcal{U}} \Phi_{\text{ss}}(u; \mathsf{x}, \mathsf{y}) \tag{2.4}$$

$$\Phi_{\text{ss}}(u; \mathsf{x}, \mathsf{y}) = \frac{1}{|Z'|} \sum_{j \in Z'} \mathcal{L}(\mathcal{G}(u|x_j), y_j) + R(u; \mathsf{x}) \tag{2.5}$$

where the regularizer $R(u; \mathsf{x})$ may use the unlabeled data in $Z \backslash Z'$, but the loss term involves only labeled data in Z'.

There are a variety of ways in which one can construct the regularizer $R(u; \mathsf{x})$ including graph-based and low-density separation methods (A. L. Bertozzi et al., 2017; A. Bertozzi and Flenner, 2012). In this work, we will study a nonparametric graph approach where we think of Z as indexing the nodes on a graph. To illustrate ideas we consider the case of binary outputs, take $\mathcal{Y} = \mathbb{R}$ and restrict attention to mappings $\mathcal{G}(u|\cdot)$ which take values in $\{-1, 1\}$; we sometimes abuse notation and simply take $\mathcal{Y} = \{-1, 1\}$, so that $\mathcal{Y}$ is no longer a Hilbert space. We assume that $\mathcal{U}$ comprises real-valued functions on the nodes Z of the graph, equivalently vectors in $\mathbb{R}^N$. We specify that $\mathcal{G}(u|j) = \operatorname{sgn}(u(j))$ for all $j \in Z$, and take, for example, the probit or logistic loss function (C. E. Rasmussen and C. K. Williams, 2006; A. L. Bertozzi et al., 2017). Once we have found an optimal parameter value for $u : Z \to \mathbb{R}$, application of $\mathcal{G}$ to u will return a labeling over all nodes j in Z. In order to use all the unlabeled data we introduce edge weights which measure affinities between nodes of a graph with vertices Z, by means of a weight function on $\mathcal{X} \times \mathcal{X}$. We then compute the graph Laplacian $\mathsf{L}(\mathsf{x})$ and use it to define a regularizer in the form

$$R(u; \mathsf{x}) = \langle u, (\mathsf{L}(\mathsf{x}) + \tau^2 I)^{\alpha} u \rangle_{\mathbb{R}^N}.$$

Here I is the identity operator, and $\tau, \alpha \in \mathbb{R}$ with $\alpha > 0$ are tunable parameters. Further details of this method are in the following section. Applications of semi-supervised learning can include any situation where data in the image space $\mathcal{Y}$ is hard to come by, for example because it requires expert human labeling; a specific example is medical imaging (Litjens et al., 2017).

Online Learning

Our third and final class of learning problems concerns situations where samples of data are presented to us sequentially and we aim to refine our choice of parameters at each step. We thus have the supervised learning problem (2.2) and we aim to solve it sequentially as each pair of data points $\{x_j, y_j\}$ is delivered. To facilitate cheap algorithms we impose a Markovian structure in which we are allowed to use only the current data sample, as well as our previous estimate of the parameters, when searching for the new estimate. We look for a sequence $\{u_j\}_{j=1}^{\infty} \subset \mathcal{U}$ such that $u_j \to u^*$ as $j \to \infty$ where, in the perfect scenario, u^* will be a minimizer of the limiting learning problem (2.1). To make the problem Markovian, we may formulate it as the following minimization task:

$$u_j = \operatorname*{arg\,min}_{u \in \mathcal{U}} \Phi_{\mathrm{o}}(u, u_{j-1}; x_j, y_j) \tag{2.6}$$

$$\Phi_{\mathrm{o}}(u, u_{j-1}; x_j, y_j) = \mathcal{L}(\mathcal{G}(u|x_j), y_j) + R(u; u_{j-1}) \tag{2.7}$$

where R is again a regularizer that could enforce a closeness condition between consecutive parameter estimates, such as

$$R(u; u_{j-1}) = \lambda \|u - u_{j-1}\|_{\mathcal{U}}^2.$$

Furthermore this regularization need not be this explicit, but could rather be included in the method chosen to solve (2.6). For example if we use an iterative method for the minimization, we could simply start the iteration at u_{j-1}.

This formulation of supervised learning is known as *online* learning. It can be viewed as reducing computational cost as a cheaper, sequential way of estimating a solution to (2.1); or it may be necessitated by the sequential manner in which data is acquired.

Inverse Problems

The preceding discussion demonstrates that, while the goal of learning is to find a mapping which generalizes across the whole distribution of possible data, in practice, we are severely restricted by only having access to a finite data set. Namely formulations (2.2), (2.4), and (2.6) can be stated for any input-output pair data set with no reference to $\mathbb{P}(x, y)$ by simply assuming that there exists some function in our model class that will relate the two. In fact, since $\mathcal{L}$ is positive-definite, its dependence also washes out when ones takes a function approximation point of view.

To make the above precise, consider the inverse problem of finding $u \in \mathcal{U}$ such that

$$\mathsf{y} = \mathsf{G}(u|\mathsf{x}) + \eta; \tag{2.8}$$

here $\mathsf{G}(u|\mathsf{x}) = [\mathcal{G}(u|x_1), \dots, \mathcal{G}(u|x_N)]$ is a concatenation and $\eta \sim \pi$ is a $\mathcal{Y}^N$-valued random variable distributed according to a measure π that models possible noise in the data, or model error. In order to facilitate a Bayesian formulation of this inverse problem we let μ_0 denote a prior probability measure on the parameters u. Then supposing

$$\begin{aligned} -\log(\pi(\mathsf{y} - \mathsf{G}(u|\mathsf{x}))) &\propto \sum_{j=1}^{N} \mathcal{L}(\mathcal{G}(u|x_j), y_j) \\ -\log(\mu_0(u)) &\propto R(u) \end{aligned}$$

we see that (2.2) corresponds to the standard MAP estimator arising from a Bayesian formulation of (2.8). The semi-supervised learning problem (2.4) can also be viewed as a MAP estimator by restricting (2.8) to Z' and using x to build μ_0. This is the perspective we take in this work and we illustrate with an example for each type of problem.

Example 1. *Suppose that $\mathcal{Y}$ and $\mathcal{U}$ are Euclidean spaces and let $\pi = \mathcal{N}(0, \Gamma)$ and $\mu_0 = \mathcal{N}(0, \Sigma)$ be Gaussian with positive-definite covariances Γ, Σ where Γ is block-diagonal with N identical blocks Γ_0. Computing the MAP estimator of* (2.8)*, we obtain that $\mathcal{L}(y', y) = \|y - y'\|^2_{\Gamma_0}$ and $R(u) = \|u\|^2_{\Sigma}$.*

Example 2. *Suppose that $\mathcal{U} = \mathbb{R}^N$ and $\mathcal{Y} = \mathbb{R}$ with the data $y_j = \pm 1 \; \forall j \in Z'$. We will take the model class to be a single function $\mathcal{G} : \mathbb{R}^N \times Z \to \mathbb{R}$ depending only on the index of each data point and defined by $\mathcal{G}(u|j) = sgn(u_j)$. As mentioned, we think of Z as the nodes on a graph and construct the edge set $E = (e_{ij}) = \eta(x_i, x_j)$ where $\eta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is a symmetric function. This allows construction of the associated graph Laplacian $\mathsf{L}(\mathsf{x})$. We shift it to remove its null space and consider powers of this operator leading to the symmetric, positive-definite operator $C = (\mathsf{L}(\mathsf{x}) + \tau^2 I)^{-\alpha}$ from which we can define the Gaussian measure $\mu_0 = \mathcal{N}(0, C)$. For details on why this construction defines a reasonable prior we refer to (A. L. Bertozzi et al., 2017). Letting $\pi = \mathcal{N}(0, \frac{1}{\gamma^2} I)$, we restrict* (2.8) *to the inverse problem*

$$y_j = \mathcal{G}(u|j) + \eta_j \quad \forall j \in Z'.$$

With the given definitions, letting $\gamma^2 = |Z'|$, the associated MAP estimator has the form of (2.4)*, namely*

$$\frac{1}{|Z'|} \sum_{j \in Z'} |\mathcal{G}(u|j) - y_j|^2 + \langle u, C^{-1} u \rangle_{\mathbb{R}^N}.$$

The infimum for this functional is not achieved (M. Iglesias, Lu, and A.M. Stuart, 2016), but the ensemble based methods we employ to solve the problem implicitly apply a further regularization which circumvents this issue.

Example 3. *Lastly we turn to the online learning problem (2.6). We assume that there is some unobserved, fixed in time parameter of our model that will perfectly match the observed data up to a noise term. Our goal is to estimate this parameter* *sequentially. Namely, we consider the stochastic dynamical system,*

$$\begin{aligned} u_{j+1} &= u_j \\ y_{j+1} &= \mathcal{G}(u_{j+1}|x_{j+1}) + \eta_{j+1} \end{aligned} \tag{2.9}$$

where the sequence $\{\eta_j\}$ are $\mathcal{Y}$-valued i.i.d. random variables that are also independent from the data. This is an instance of the classic filtering problem considered in data assimilation (K. Law, Andrew Stuart, and Zygalakis, 2015). We may view this as solving an inverse problem at each fixed time with increasingly strong prior information as time unrolls. With the appropriate assumptions on the prior and the noise model, we may again view (2.6) *as the MAP estimators of each fixed inverse problem. Thus we may consider all problems presented here in the general framework of (2.8).*

2.3 Approximation Architectures

In this section, we outline the approximation architectures that we will use to solve the three machine learning tasks outlined in the preceding section. For supervised and online learning these amount to specifying the dependence of $\mathcal{G}$ on u; for semi-supervised learning this corresponds to determining a basis in which to seek the parameter u. We do not give further details for the semi-supervised case as our numerics fit in the context of Example 2, but we refer the reader to (A. L. Bertozzi et al., 2017) for a detailed discussion.

Subsection 2.3 details feed-forward neural networks with subsections 2.3 and 2.3 showing the parameterizations of dense and convolutional networks respectively.

Feed-Forward Neural Networks

Feed-forward neural networks are a parametric model class defined as discrete time, nonautonomous, semi-dynamical systems of an unusual type. Each map in the composition takes a specific parametrization and can change the dimension of its input while the whole system is computed only up to a fixed time horizon. To make this precise, we will assume $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}^m$ and define a neural network with $n \in \mathbb{N}$ hidden layers as the composition

$$\mathcal{G}(u|x) = (S \circ A \circ F_{n-1} \circ \cdots \circ F_0)(x)$$

where $d_0 = d$ and $F_j \in C(\mathbb{R}^{d_j}, \mathbb{R}^{d_{j+1}}), n = 0, \dots, n-1$ are nonlinear maps, referred to as *layers*, depending on parameters $\theta_0, \dots, \theta_{n-1}$ respectively, $A : \mathbb{R}^{d_n} \to \mathbb{R}^m$ is an affine map with parameters θ_n, and $u = [\theta_0, \dots, \theta_n]$ is a concatenation. The map $S : \mathbb{R}^m \to V \subseteq \mathbb{R}^m$ is fixed and thought of as a projection or thresholding done to move the output to the appropriate subset of data space. The choice of S is dependent on the problem at hand. If we are considering a regression task and $V = \mathbb{R}^m$ then S can simply be taken as the identity. On the other hand, if we are considering a classification task and $V = \mathbb{P}^m$, the set of probability vectors in $\mathbb{R}^m$, then S is often taken to be the softmax function defined as

$$S(w) = \frac{1}{\sum_{j=1}^{m} e^{w_j}} (e^{w_1}, \dots, e^{w_m}).$$

From this perspective, the neural network approximates a categorical distribution of the input data and the softmax arises naturally as the canonical response function of the categorical distribution (when viewed as belonging to the exponential family of distributions) (McCullagh and Nelder, 1989; Ruck et al., 1990). If we have some specific bounds for the output data, for example $V = [-1, 1]^m$ then S can be a point-wise hyperbolic tangent.

What makes this dynamic unusual is the fact that each map can change the dimension of its input unlike a standard dynamical system which operates on a fixed metric space. However, note that the sequence of dimension changes $d_1, \dots, d_n$ is simply a modeling choice that we may alter. Thus let $d_{\max} = \max\{d_0, \dots, d_n\}$ and consider the solution map $\phi : \mathbb{N}_0 \times \mathbb{N}_0 \times \mathbb{R}^{d_{\max}} \to \mathbb{R}^{d_{\max}}$ generated by the nonautonomous difference equation

$$z_{k+1} = F_k(z_k)$$

where each map $F_k \in C(\mathbb{R}^{d_{\max}}, \mathbb{R}^{d_{\max}})$ is such that $\dim \operatorname{Im}(F_k) \leq d_{k+1}$; then $\phi(n, m, x)$ is z_n given that $z_m = x$. We may then define a neural network as

$$\mathcal{G}(u|x) = S \circ A \circ \phi(n, 0, \mathcal{P}x)$$

where $\mathcal{P} : \mathbb{R}^d \to \mathbb{R}^{d_{\max}}$ is a projection operator and $A : \mathbb{R}^{d_{\max}} \to \mathbb{R}^m$ is again an affine map. While this definition is mathematically satisfying and potentially useful for theoretical analysis as there is a vast literature on nonautonomus semi-dynamical systems (Kloeden and M. Rasmussen, 2011), in practice, it is more useful to think of each map as changing the dimension of its input. This is because it allows us to work with parameterizations that explicitly enforce the constraint on the dimension of the image. We take this point of view for the rest of this section to illustrate the practical uses of neural networks.

Dense Networks

A key feature of neural networks is the specific parametrization of each map F_k. In the most basic case, each F_k is an affine map followed by a point-wise nonlinearity, in particular,

$$F_k(z_k) = \sigma(W_k z_k + b_k)$$

where $W_k \in \mathbb{R}^{d_{k+1} \times d_k}$, $b_k \in \mathbb{R}^{d_{k+1}}$ are the parameters i.e. $\theta_k = [W_k, b_k]$ and $\sigma \in C(\mathbb{R}, \mathbb{R})$ is non-constant and bounded; we extend σ to a function on $\mathbb{R}^d$ by defining it point-wise as $\sigma(u)_j = \sigma(u_j)$ for any vector $u \in \mathbb{R}^d$. This layer type is referred to as *dense*, or fully-connected, because each entry in W_k is a parameter with no global sparsity assumptions and hence we can end up with a dense matrix. A neural network with only this type of layer is called dense or fully-connected (DNN).

The nonlinearity σ, called the *activation function*, is a design choice and usually does not vary from layer to layer. Some popular choices include the sigmoid, the hyperbolic tangent, or the rectified linear unit (ReLU) defined by $\sigma(q) = \max\{0, q\}$. Note that ReLU is unbounded and hence does not satisfy the assumptions for the classical universal approximation theorem (Hornik, 1991), but it has shown tremendous numerical success when the associated inverse problem is solved via backpropagation (method of adjoints) (Nair and Geoffrey E. Hinton, 2010).

Convolutional Networks

Instead of seeking the full representation of a linear operator at each time step, we may consider looking only for the parameters associated to a pre-specified type of operator. Namely we consider

$$F_k(z_k) = \sigma(W(s_k) z_k + b_k)$$

where W can be fully specified by the parameter s_k. The most commonly considered operator is the one arising from a discrete convolution (Lecun et al., 1998). The motivation behind this choice lies in the application to natural images where we want to exploit spatial features of the data. In particular, considering objects in an image, a natural choice is to pick a transformation which is translation equivariant. Namely, translations in the image result in equivalent translations in the output. This captures the property that moving an object in an image does not change the content of that image. Convolution then becomes the natural choice as as the set of convolutions is in bijection with the set of equivariant functions.

We consider the input z_k as a function on the integers with period d_k then we may define $W(s_k)$ as the circulant matrix arising as the kernel of the discrete circular convolution with convolution operator s_k. Exact construction of the operator W is a modeling choice as one can pick exactly which blocks of z_k to compute the convolution over. Usually, even with maximally overlapping blocks, the operation is dimension reducing, but can be made dimension preserving, or even expanding, by appending zero entries to z_k. This is called *padding*. For brevity, we omit exact descriptions of such details and refer the reader to (Goodfellow, Bengio, and Courville, 2016). The parameter s_k is known as the *stencil*. Neural networks following this construction are called *convolutional* (CNN).

In practice, a CNN computes a linear combination of many convolutions at each time step, namely

$$F_k^{(j)}(z_k) = \sigma\left(\sum_{m=1}^{M_k} W(s_k^{(j,m)})z_k^{(m)} + b_k^{(j)}\right)$$

for $j = 1, 2, \dots, M_{k+1}$ where $z_k = [z_k^{(1)}, \dots, z_k^{(M_k)}]$ with each entry known as a *channel* and $M_k = 1$ if no convolutions were computed at the previous iteration. Finally we define $F_k(z_k) = [F_k^{(1)}(z_k), \dots, F_k^{(M_{k+1})}(z_k)]$. The number of channels at each time step, the integer M_{k+1}, is a design choice which, along with the choice for the size of the stencils $s_k^{(j,m)}$, the dimension of the input, and the design of W determine the dimension of the image space d_{k+1} for the map F_k.

When employing convolutions, it is standard practice to sometimes place maps which compute certain statistics from the convolution. These operations are commonly referred to as *pooling*. They are dimension reducing and are usually thought of as a way of extracting the most important information from a convolution. We refer the reader to (Nagi et al., 2011; Tsai, Hamsici, and Yang, 2015; Graham, 2014; Gulcehre et al., 2014; K. He et al., 2014) for further details.

Perhaps the most common such operation is known as max-pooling. To illustrate suppose $[F_k^{(1)}, \dots, F_k^{(M_{k+1})}]$ are the M_{k+1} channels computed as the output of a convolution (dropping the z_k dependence for notational convenience). In this context, it is helpful to change perspective slightly and view each $F_k^{(j)}$ as a matrix whose concatenation gives the vector F_k. Each of these matrices is a two-dimensional grid whose value at each point represents a linear combination of convolutions each computed at that spatial location. We define a maximum-based, block-subsampling

Convolutional Neural Network		
Map	**Type**	**Notation**
$F_0 : \mathbb{R}^{28\times28} \to \mathbb{R}^{32\times24\times24}$	Conv 32x5x5	$M_0 = 1, M_1 = 32, s_0^{(j,m)} \in \mathbb{R}^{5\times5}$ $j \in \{1,\dots,32\}, m = \{1\}$
$F_1 : \mathbb{R}^{32\times24\times24} \to \mathbb{R}^{32\times10\times10}$	Conv 32x5x5 MaxPool 2x2	$M_2 = 32, s_1^{(j,m)} \in \mathbb{R}^{5\times5}$ $j \in \{1,\dots,32\}, m \in \{1,\dots,32\}$ $H_1 = H_2 = 2$ $(\alpha = \beta = 2)$
$F_2 :: \mathbb{R}^{32\times10\times10} \to \mathbb{R}^{64\times6\times6}$	Conv 64x5x5	$M_3 = 64, s_2^{(j,m)} \in \mathbb{R}^{5\times5}$ $j \in \{1,\dots,64\}, m \in \{1,\dots,32\}$
$F_3 : \mathbb{R}^{64\times6\times6} \to \mathbb{R}^{64}$	Conv 64x5x5 MaxPool 2x2 (global)	$M_4 = 64, s_3^{(j,m)} \in \mathbb{R}^{5\times5}$ $j \in \{1,\dots,64\}, m \in \{1,\dots,64\}$ $H_1 = H_2 = 2$
$F_4 : \mathbb{R}^{64} \to \mathbb{R}^{500}$	FC-500	$W_4 \in \mathbb{R}^{500\times64}, b_4 \in \mathbb{R}^{500}$
$A : \mathbb{R}^{500} \to \mathbb{R}^{10}$	FC-10	$W_5 \in \mathbb{R}^{10\times500}, b_5 \in \mathbb{R}^{10}$
$S : \mathbb{R}^{10} \to \mathbb{R}^{10}$	Softmax	$S(w) = \frac{1}{\sum_{j=1}^{10} e^{w_j}}(e^{w_1},\dots,e^{w_{10}})$

Figure 21: A four layer convolutional neural network for classifying images in the MNIST data set. The middle column shows a description typical of the machine learning literature. The other two columns connect this jargon to the notation presented here. No padding is added and the convolutions are computed over maximally overlapping blocks (stride of one). The nonlinearity σ is the ReLU and is the same for every layer.

operation

$$(p_k^{(j)})_{il} = \max_{q\in\{1,\dots,H_1\}} \max_{v\in\{1,\dots,H_2\}} (F_k^{(j)})_{\alpha(i-1)+q,\beta(l-1)+v}$$

where the tuple $(H_1, H_2) \in \mathbb{N}^2$ is called the *pooling kernel* and the tuple $(\alpha, \beta) \in \mathbb{N}^2$ is called the *stride*, each a design choice for the operation. It is common practice to take $H_1 = H_2 = \alpha = \beta$. We then define the full layer as $F_k(z_k) = [p_k^{(1)}(z_k), \dots, p_k^{(M_{k+1})}(z_k)]$. There are other standard choices for pooling operations including average pooling, ℓ_p-pooling, fractional max-pooling, and adaptive max-pooling where each of the respective names are suggestive of the operation being performed; details may be found in (Tsai, Hamsici, and Yang, 2015; Graham, 2014; Gulcehre et al., 2014; K. He et al., 2014). Note that pooling operations are dimension reducing and are usually thought of as a way of extracting the most important information from a convolution. When one chooses the kernel (H_1, H_2) such that $F_k^{(j)} \in \mathbb{R}^{H_1\times H_2}$, the per channel output of the pooling is a scalar and the operation is called *global pooling*.

Designs of feed-forward neural networks usually employ both convolutional (with and without pooling) and dense layers. While the success of convolutional networks

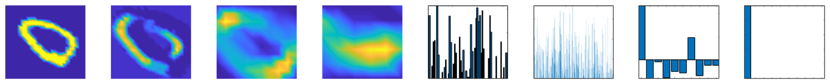


Figure 22: Output of each map from left to right of the convolutional neural network shown in Figure 21. The left most image is the input and the next three images show a single randomly selected channel from the outputs of F_0, F_1, F_2 respectively. The outputs of F_3, F_4, A, S are vectors shown respectively in the four subsequent plots. We see that with high probability the network determines that the image belongs to the first class (0) which is correct.

has mostly come from image classification or object detection tasks (Krizhevsky, Sutskever, and Geoffrey E Hinton, 2012), they can be useful for any data with spatial correlations (Binkowski, Marti, and Donnat, 2017; Goodfellow, Bengio, and Courville, 2016). To connect the complex notation presented in this section with the standard in machine learning literature, we will give an example of a deep convolutional neural network. We consider the task of classifying images of hand-written digits given in the MNIST dataset (LeCun and Cortes, 2010). These are 28×28 grayscale images of which there are $N = 60,000$ and 10 overall classes $\{0, \dots, 9\}$ hence we consider $\mathcal{X} = \mathbb{R}^{28\times 28} \cong \mathbb{R}^{784}$ and $\mathcal{Y}$ the space of probability vectors over $\mathbb{R}^{10}$. Figure 21 show a typical construction of a deep convolutional neural network for this task. The word deep is generally reserved for models with $n > 3$. Once the model has been fit, Figure 22 shows the output of each map on an example image. Starting with the digitized digit 0, the model computes its important features, through a sequence of operations involving convolutional layers, culminating in the second to last plot, the output of the affine map A. This plot shows model determining that the most likely digit is 0, but also giving substantial probability weight on the digit 6. This makes sense, as the digits 0 and 6 can look quite similar, especially when hand-written. Once the softmax is taken (because it exponentiates), the probability of the image being a 6 is essentially washed out, as shown in the last plot. This is a short-coming of the softmax as it may not accurately retain the confidence of the model's prediction. We stipulate that this may be a reason for the emergence of highly-confident adversarial examples (Szegedy et al., 2014; Goodfellow, Shlens, and Szegedy, 2015), but do not pursue suitable modifications in this work.

Recurrent Neural Networks

Recurrent neural networks are models for time-series data defined as discrete time, nonautonomous, semi-dynamical systems that are parametrized by feed-forward neural networks. To make this precise, we first define a layer of two-inputs simply as the sum of two affine maps followed by a point-wise nonlinearity, namely for $j = 1, \dots, n$ define $F_{\theta_j} : \mathbb{R}^{d_h} \times \mathbb{R}^d \to \mathbb{R}^{d_h}$ by

$$F_{\theta_j}(z, q) = \sigma(W_h^{(j)} z + b_h^{(j)} + W_x^{(j)} q + b_x^{(j)})$$

where $W_h^{(j)} \in \mathbb{R}^{d_h \times d_h}$, $W_x^{(j)} \in \mathbb{R}^{d_h \times d}$ and $b_h^{(j)}, b_x^{(j)} \in \mathbb{R}^{d_h}$; the parameters are then given by the concatenation $\theta_j = [W_h^{(j)}, W_x^{(j)}, b_h^{(j)}, b_x^{(j)}]$. The dimension d_h is a design choice that we can pick on a per-problem basis. Now define the map $F_\theta : \mathbb{R}^{d_h} \times \mathbb{R}^d \to \mathbb{R}^{d_h}$ by composing along the first component

$$F_\theta(z, q) = F_{\theta_n}(F_{\theta_{n-1}}(\dots F_{\theta_1}(z, q), \dots q), q), q)$$

where $\theta = [\theta_1, \dots, \theta_n]$ is a concatenation. Now suppose $x_0, \dots, x_{T-1} \in \mathbb{R}^d$ is an observed time series and define the dynamic

$$h_{t+1} = F_\theta(h_t, x_t)$$

up to time $t = T$. We can think of this as a nonautonomous, semi-dynamical system on $\mathbb{R}^{d_h}$ with parameter $x = [x_0, \dots, x_{T-1}]$. Let $\phi : \{0, \dots, T\} \times \mathbb{R}^{T \times d} \times \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$ be the solution map generated by this difference equation. We can finally define a recurrent neural network $\mathcal{G}(u|\cdot) : \mathbb{R}^{T \times d} \to V \subseteq \mathbb{R}^{T \times d}$ by

$$\mathcal{G}(u|x) = \begin{bmatrix} S(A_1 \circ \phi(1, x, h_0)) \\ S(A_2 \circ \phi(2, x, h_0)) \\ \vdots \\ S(A_T \circ \phi(T, x, h_0)) \end{bmatrix}$$

where $A_1, \dots, A_T$ are affine maps, S is a thresholding (such as softmax) as previously discussed, and u a concatenation of the parameters θ as well as the parameters for all of the affine maps. Usually ones takes $h_0 = 0$, but randomly generated initial conditions are also used in practice.

The construction presented here is the most basic recurrent neural network. Many others architectures such as Long Short-Term Memory (LSTM), recursive, and bi-recurrent networks have been proposed in the literature (Schmidhuber, 1992; Sepp Hochreiter and Schmidhuber, 1997; S. Hochreiter et al., 2001; Goodfellow, Bengio, and Courville, 2016), but they are all slight modifications to the above dynamic. These architectures can be used as sequence to sequence maps, or, if we only consider the output at the last time that is $S(A_T \circ \phi(T, x, h_0))$, as predicting x_T or classifying the sequence $x_0, \dots, x_{T-1}$. We refer the reader to (Sutskever, 2013) for an overview of the applications of recurrent neural networks.

2.4 Algorithms

Subsection 2.4 describes the choice of loss function. Subsection 2.4 outlines the state-of-the-art derivative based optimization, with subsection 2.4 presenting the algorithms and subsection 2.4 presenting tricks for better convergence. Subsection 2.4 defines the EKI method, with subsequent subsections presenting our various modifications.

Loss Function

Before delving into the specifics of optimization methods used, we discuss the general choice of loss function $\mathcal{L}$. While the machine learning literature contains a wide variety of loss functions that are designed for specific problems, there are two which are most commonly used and considered first when tackling any regression and classification problems respectively, and on which we focus our work in this paper. For regression tasks, the squared-error loss

$$\mathcal{L}(y', y) = \|y - y'\|_{\mathcal{Y}}^2$$

is standard and is well known to the inverse problems community; it arises from an additive Gaussian noise model. When the task at hand is classification, the standard choice of loss is the cross-entropy

$$\mathcal{L}(y', y) = -\langle y, \log y' \rangle_{\mathcal{Y}},$$

with the $\log$ computed point-wise and where we consider $\mathcal{Y} = \mathbb{R}^m$. This loss is well-defined on the space $\mathbb{P}_0^m \times \mathbb{P}^m$. It is consistent with the the projection map S of the neural network model being the softmax as $\text{Im}(S) = \mathbb{P}_0^m$. A simple Lagrange multiplier argument shows that $\mathcal{L}$ is indeed infimized over $\mathbb{P}_0^m$ by sequence $y' \to y$ and hence the loss is consistent with what we want our model output to be. [1] From a modeling perspective, the choice of softmax as the output layer has some drawbacks as it only allows us to asymptotically match the data. However it is a good choice if the cross-entropy loss is used to solve the problem; indeed, in practice, the softmax along with the cross-entropy loss has seen the best numerical results when compared to other choices of thresholding/loss pairs (Goodfellow, Bengio, and Courville, 2016).

The interpretation of the cross-entropy loss is to think of our model as approximating a categorical distribution over the input data and, to get this approximation, we want

[1]Note that the infimum is not, in general, attained in $\mathbb{P}_0^m$ as defined, because perfectly labeled data may take the form $\{y \in \mathbb{R}^m \mid \exists! j \text{ s.t. } y_j = 1, y_k = 0 \; \forall k \neq j\}$ which is in the closure of $\mathbb{P}_0^m$ but not in $\mathbb{P}_0^m$ itself.

to minimize its Shannon cross-entropy with respect to the data. Note, however, that there is no additive noise model for which this loss appears in the associated MAP estimator simply because $\mathcal{L}$ cannot be written purely as a function of the residual $y - y'$.

Gradient Based Optimization

The Iterative Technique

The current state of the art for solving optimization problems of the form (2.2), (2.4), (2.6) is based around the use of stochastic gradient descent (SGD) (Robbins and Monro, 1951; Kiefer and Wolfowitz, 1952; Rumelhart, Geoffrey E. Hinton, and R. J. Williams, 1988). We will describe these methods starting from a continuous time viewpoint, for pedagogical clarity. In particular, we think of the unknown parameter as the large time limit of a smooth function of time $u : [0, \infty) \to \mathcal{U}$. Let $\Phi(u; \mathsf{x}, \mathsf{y}) = \Phi_{\mathrm{s}}(u; \mathsf{x}, \mathsf{y})$ or $\Phi_{\mathrm{ss}}(u; \mathsf{x}, \mathsf{y})$, then gradient descent imposes the dynamic

$$\dot{u} = -\nabla\Phi(u; \mathsf{x}, \mathsf{y}), \quad u(0) = u_0 \tag{2.10}$$

which moves the parameter in the steepest descent direction with respect to the regularized loss function, and hence will converge to a local minimum for Lebesgue almost all initial data, leading to bounded trajectories (Lee et al., 2016; Andrew Stuart and Humphries, 1998).

For the practical implementations of this approach in machine learning, a number of adaptations are made. First the ODE is discretized in time, typically by a forward Euler scheme; the time-step is referred to as the *learning rate*. The time-step is often, but not always, chosen to be a decreasing function of the iteration step (Dieuleveut, Bach, et al., 2016; Robbins and Monro, 1951). Secondly, at each step of the iteration, only a subset of the data is used to approximate the full gradient. In the supervised case, for example,

$$\Phi_{\mathrm{s}}(u; \mathsf{x}, \mathsf{y}) \approx \frac{1}{N'} \sum_{j \in B_{N'}} \mathcal{L}(\mathcal{G}(u|x_j), y_j) + R(u)$$

where $B_{N'} \subset \{1, \dots, N\}$ is a random subset of cardinality N' usually with $N' \ll N$. A new $B_{N'}$ is drawn at each step of the Euler scheme without replacement until the full dataset has been exhausted. One such cycle through all of the data is called an *epoch*. The number of epochs it takes to train a model varies significantly based on the model and data at hand but is usually within the range 10 to 500. This idea, called *mini-batching*, leads to the terminology *stochastic gradient descent (SGD)*.

Recent work has suggested that adding this type of noise helps preferentially guide the gradient descent towards places in parameter space which generalize better than standard descent methods (Chaudhari, Choromanska, et al., 2016; Chaudhari and Soatto, 2017).

A third variant on basic gradient descent is the use of momentum-augmented methods utilized to accelerate convergence (Nesterov, 1983). The deep theory associated with these methods relates to a clever adaptive choice of momentum which changes with the step of the algorithm; however as used in machine lerning practice the momentum level is typically fixed. Various continuous time dynamics associated with the Nesterov momentum method can be found in (Su, Boyd, and E. Candes, 2014; Kovachki and Andrew M. Stuart, 2019) and take the form

$$\begin{aligned} &m\ddot{u} + \gamma(t)\dot{u} = -\nabla\Phi(u; \mathsf{x}, \mathsf{y}), \\ &u(0) = u_0, \quad \dot{u}(0) = 0. \end{aligned} \tag{2.11}$$

From these variants on continuous time gradient descent have come a plethora of adaptive first-order optimization methods that attempt to solve the learning problem. Some of the more popular include Adam, RMSProp, and Adagrad (Kingma and J. Ba, 2014; Duchi, Hazan, and Singer, 2011). There is no consensus on which method performs best, although some recent work has argued in favor of SGD and momentum SGD (Wilson et al., 2017).

Lastly, the online learning problem (2.6) is also commonly solved via a gradient descent method dubbed online gradient descent (OGD). The dynamic is

$$\begin{aligned} &\dot{u}_j = -\nabla\Phi_{\mathrm{o}}(u_j, u_{j-1}; x_j, y_j) \\ &u_j(0) = u_{j-1}(T) \end{aligned}$$

which can be extended to the momentum case in the obvious way. It is common that only a single step of the Euler scheme is computed. The process of letting all these ODE(s) evolve in time is called *training*.

Initialization

A major challenge for the iterative methods presented here is finding a good starting point u_0 for the dynamic; a problem usually termed *initialization*. Historically, initialization was first dealt with using a technique called *layer-wise pretraining* (G. Hinton and Salakhutdinov, 2006). In this approach the parameters are initialized randomly. Then the parameters of all but first layer are held fixed and SGD is used to find the parameters of the first layer. Then all but the parameters of the second layer are held fixed and SGD is used to find the parameters of the second layer. Repeating this for all layers yields an estimate u_0 for all the parameters, and this is then used as an initialization for SGD in a final step called *fine-tuning*. Development of new activation functions, namely the ReLU, has allowed simple random initialization (from a carefully designed prior measure) to work just as well, making layer-wise pretraining essentially obsolete. There are many proposed strategies in the literature for how one should design this prior (Glorot and Bengio, 2010; Mishkin and Matas, 2015). The main idea behind all of them is to somehow normalize the output mean and variance of each map F_k. One constructs the product probability measure

$$\mu_0 = \mu_0^{(0)} \otimes \mu_0^{(1)} \otimes \cdots \otimes \mu_0^{(n-1)} \otimes \mu_0^{(n)}$$

where each $\mu_0^{(k)}$ is usually a centered, isotropic probability measure with covariance scaling γ_k. Each such measure corresponds to the distribution of the parameters of each respective layer with $\mu_0^{(n)}$ attached to the parameters of the map A. A common strategy called Xavier initialization (Glorot and Bengio, 2010) proposes that the inverse covariance (precision) is determined by the average of the input and output dimensions of each layer:

$$\gamma_k^{-1} = \frac{1}{2}\big(d_k + d_{k+1}\big)$$

and thus

$$\gamma_k = \frac{2}{d_k + d_{k+1}}.$$

When the layer is convolutional, d_k and d_{k+1} are instead taken to be the number of input and output channels times the size of each stencil respectively. Usually each $\mu_0^{(k)}$ is then taken to be a centered Gaussian or uniform probability measure. Once this prior is constructed one initializes SGD by a single draw.

As we have seen, initialization strategies aim to normalize the output distribution of each layer. However, once SGD starts and the parameters change, this normalization is no longer in place. This issue has been called the *internal covariate shift*. To address it, normalizing parameters are introduced after the output of each layer. The most common strategy for finding these parameters is called *batch-normalization* (Ioffe and Szegedy, 2015), which, as the name implies, relies on computing a mini-batch of the data. To illustrate the idea, suppose $z_m(x_{k_1}), \ldots, z_m(x_{k_B})$ are the outputs of the map F_{m-1} at inputs $x_{k_1}, \dots, x_{k_B}$. We compute the mean and variance

$$\nu_m = \frac{1}{B}\sum_{j=1}^{B} z_m(x_{k_j}); \quad \sigma_m^2 = \frac{1}{B}\sum_{j=1}^{B} \|z_m(x_{k_j}) - \nu_m\|_2^2$$

and normalize these outputs so that the inputs to the map F_m are

$$\frac{z_m(x_{k_j}) - \nu_m}{\sqrt{\sigma_m^2 + \epsilon}}\gamma + \beta$$

where $\epsilon > 0$ is used for numerical stability while γ, β are new parameters to be estimated, and are termed the *scale* and *shift* respectively; they are found by means of the SGD optimization process. It is not necessary to introduce the new parameters γ, β but is common in practice and, with them, the operation is called *affine batch-normalization*. When an output has multiple channels, separate normalization is done per channel. During training a running mean of each ν_m, σ_m^2 is kept and the resulting values are used for the final model. A clear drawback to batch normalization is that it relies on batches of the data to be computed and hence cannot be used in the online setting. Many similar strategies have been proposed (Ulyanov, Vedaldi, and Lempitsky, 2016; L. J. Ba, Kiros, and Geoffrey E. Hinton, 2016) with no clear consensus on which works best. Recently a new activation function called SeLU (Klambauer et al., 2017) has been claimed to perform the required normalization automatically.

Ensemble Kalman Inversion

The ensemble Kalman filter (EnKF) is a method for estimating the state of a stochastic dynamical system from noisy observations (Evensen, 2003). Over the last decade the method has been systematically developed as an iterative method for solving general inverse problems; in this context, it is sometimes referred to as ensemble Kalman inversion (EKI) (M. A. Iglesias, K. J. H. Law, and Andrew M Stuart, 2013). Viewed as a sequential Monte Carlo method (Schillings and Andrew M. Stuart, 2017), it works on an ensemble of parameter estimates (particles) transforming them from the prior into the posterior. Recent work has established, however, that unless the forward operator is linear and the additive noise is Gaussian (Schillings and Andrew M. Stuart, 2017), the correct posterior is not obtained (G. Ernst, Sprungk, and Starkloff, 2015). Nevertheless there is ample numerical evidence that shows EKI works very well as a derivative-free optimization method for nonlinear least-squares problems (Kay and Sebastian, n.d.; Bergemann and Reich, 2010). In this paper, we view it purely through the lens of optimization and propose several modifications to the method that follow from adopting this perspective within the context of machine learning problems.

Consider the general inverse problem

$$\mathsf{y} = \mathsf{G}(u) + \eta$$

where $\eta \sim \pi = N(0, \Gamma)$ represent noise, and let μ_0 be a prior measure on the parameter u. Note that the supervised, semi-supervised, and online learning problems (2.8), (2.9) can be put into this general framework by adjusting the number of data points in the concatenations y, x and letting x be absorbed into the definition of G. Let $\{u^{(j)}\}_{j=1}^{J} \subset \mathcal{U}$ be an ensemble of parameter estimates which we will allow to evolve in time through interaction with one another and with the data; this ensemble may be initialized by drawing independent samples from μ_0, for example. The evolution of $u^{(j)} : [0, \infty) \to \mathcal{U}$ is described by the EKI dynamic (Schillings and Andrew M. Stuart, 2017)

$$\begin{aligned} \dot{u}^{(j)} &= -C^{\mathrm{uw}}(u)\Gamma^{-1}(\mathsf{G}(u^{(j)}) - \mathsf{y}), \\ u^{(j)}(0) &= u_0^{(j)}. \end{aligned}$$

Here

$$\bar{\mathsf{G}} = \frac{1}{J}\sum_{l=1}^{J} \mathsf{G}(u^{(l)}), \quad \bar{u} = \frac{1}{J}\sum_{l=1}^{J} u^{(l)}$$

and $C^{\mathrm{uw}}(u)$ is the empirical cross-covariance operator

$$C^{\mathrm{uw}}(u) = \frac{1}{J}\sum_{j=1}^{J}(u^{(j)} - \bar{u}) \otimes (\mathsf{G}(u^{(j)}) - \bar{\mathsf{G}}).$$

Thus

$$\begin{aligned} \dot{u}^{(j)} &= -\frac{1}{J}\sum_{k=1}^{J}\langle \mathsf{G}(u^{(k)}) - \bar{\mathsf{G}}, \mathsf{G}(u^{(j)}) - \mathsf{y}\rangle_{\Gamma}\, u^{(k)}, \\ u^{(j)}(0) &= u_0^{(j)}. \end{aligned} \tag{2.12}$$

Viewing the difference of $\mathsf{G}(u^{(k)})$ from its mean, appearing in the left entry of the inner-product, as a projected approximate derivative of G, it is possible to understand (2.12) as an approximate gradient descent.

Rigorous analysis of the long-term properties of this dynamic for a finite J are poorly understood except in the case where $\mathsf{G}(\cdot) = A\cdot$ is linear (Schillings and Andrew M.

Stuart, 2017). In the linear case, we obtain that $u^{(j)} \to u^*$ as $t \to \infty$ where u^* minimizes the functional

$$\Phi(u; \mathsf{y}) = \frac{1}{2}\|\mathsf{y} - Au\|_\Gamma^2$$

in the subspace $\mathcal{A} = \operatorname{span}\{u_0^{(j)} - \bar{u}\}_{j=1}^J$, and where $\bar{u}$ is the mean of the initial ensemble $\{u_0^{(j)}\}$. This follows from the fact that, in the linear case, we may re-write (2.12) as

$$\dot{u}^{(j)} = -C(u)\nabla_u \Phi(u^{(j)}; \mathsf{y})$$

where $C(u)$ is an empirical covariance operator

$$C(u) = \frac{1}{J}\sum_{j=1}^{J}(u^{(j)} - \bar{u}) \otimes (u^{(j)} - \bar{u}).$$

Hence each particle performs a gradient descent with respect to Φ and $C(u)$ projects into the subspace $\mathcal{A}$.

To understand the nonlinear setting we use linearization. Note from (2.12) that

$$\begin{aligned}
\dot{u}^{(j)} &= -\frac{1}{J}\sum_{k=1}^{J}\langle \mathsf{G}(u^{(k)}) - \frac{1}{J}\sum_{l=1}^{J}\mathsf{G}(u^{(l)}), \mathsf{G}(u^{(j)}) - \mathsf{y}\rangle_\Gamma \, u^{(k)} \\
&= -\frac{1}{J}\sum_{k=1}^{J}\langle \mathsf{G}(u^{(k)}) - \frac{1}{J}\sum_{l=1}^{J}\mathsf{G}(u^{(l)}), \mathsf{G}(u^{(j)}) - \mathsf{y}\rangle_\Gamma \, (u^{(k)} - \bar{u}) \\
&= -\frac{1}{J^2}\sum_{k=1}^{J}\sum_{l=1}^{J}\langle \mathsf{G}(u^{(k)}) - \mathsf{G}(u^{(l)}), \mathsf{G}(u^{(j)}) - \mathsf{y}\rangle_\Gamma \, (u^{(k)} - \bar{u}).
\end{aligned}$$

Now we linearize on the assumption that the particles are close to one another, so that

$$\begin{aligned}
\mathsf{G}(u^{(k)}) &= \mathsf{G}(u^{(j)} + u^{(k)} - u^{(j)}) \approx \mathsf{G}(u^{(j)}) + D\mathsf{G}(u^{(j)})(u^{(k)} - u^{(j)}) \\
\mathsf{G}(u^{(l)}) &= \mathsf{G}(u^{(j)} + u^{(l)} - u^{(j)}) \approx \mathsf{G}(u^{(j)}) + D\mathsf{G}(u^{(j)})(u^{(l)} - u^{(j)}).
\end{aligned}$$

Here $D\mathsf{G}$ is the Fréchet derivative of G. With this approximation, we obtain

$$\begin{aligned}
\dot{u}^{(j)} &\approx -\frac{1}{J^2}\sum_{k=1}^{J}\sum_{l=1}^{J}\langle D\mathsf{G}^*(u^{(j)})(\mathsf{G}(u^{(j)}) - \mathsf{y}), u^{(k)} - u^{(l)}\rangle_\Gamma (u^{(k)} - \bar{u}) \\
&= -\frac{1}{J}\sum_{k=1}^{J}\langle D\mathsf{G}^*(u^{(j)})(\mathsf{G}(u^{(j)}) - \mathsf{y}), u^{(k)} - \bar{u}\rangle_\Gamma (u^{(k)} - \bar{u}) \\
&= -C(u)\nabla_u \Phi(u^{(j)}, \mathsf{y})
\end{aligned}$$

where

$$\Phi(u;\mathsf{y}) = \frac{1}{2}\|\mathsf{y} - \mathsf{G}(u)\|_\Gamma^2.$$

This is again just gradient descent with a projection onto the subspace $\mathcal{A}$. These arguments also motivate the interesting variants on EKI proposed in (Haber, Lucka, and Ruthotto, 2018); indeed the paper (Haber, Lucka, and Ruthotto, 2018) inspired the organization of the linearization calculations above.

In summary, the EKI is a methodology which behaves like gradient descent, but achieves this without computing gradients. Instead it uses an ensemble and is hence inherently parallelizable. In the context of machine learning this opens up the possibility of avoiding explicit backpropagation, and doing so in a manner which is well-adapted to emerging computer architectures.

Cross-Entropy Loss

The previous considerations demonstrate that EKI as typically used is closely related to minimizing an ℓ_2 loss function via gradient descent. Here we propose a simple modification to the method, allowing it to minimize any loss function instead of only the squared-error; our primary motivation is the case of cross-entropy loss.

Let $\mathcal{L}(\mathsf{y}', \mathsf{y})$ be any loss function, this may, for example, be the cross entropy

$$\mathcal{L}(\mathsf{y}', \mathsf{y}) = -\frac{1}{N}\langle \mathsf{y}, \log \mathsf{y}' \rangle_{\mathcal{Y}^N}.$$

Now consider the dynamic

$$\begin{aligned} \dot{u}^{(j)} &= -C^{\mathrm{uw}}(u)\nabla_{\mathsf{y}'}\mathcal{L}(\mathsf{G}(u^{(j)}), \mathsf{y}) \\ &= -\frac{1}{J}\sum_{k=1}^{J}\langle \mathsf{G}(u^{(k)}) - \bar{\mathsf{G}}, \nabla_{\mathsf{y}'}\mathcal{L}(\mathsf{G}(u^{(j)}), \mathsf{y})\rangle\, u^{(k)}. \end{aligned} \tag{2.13}$$

If $\mathcal{L}(\mathsf{y}', \mathsf{y}) = \frac{1}{2}\|\mathsf{y} - \mathsf{y}'\|_\Gamma^2$ then $\nabla_{\mathsf{y}'}\mathcal{L}(\mathsf{G}(u^{(j)}), \mathsf{y}) = \Gamma^{-1}(\mathsf{G}(u^{(j)}) - \mathsf{y})$ recovering the original dynamic. Note that since we've defined the loss through the auxiliary variable y' which is meant to stand-in for the output of our model, the method remains derivative-free with respect to the model parameter u, but does not allow for adding regularization directly into the loss. However regularization could be added directly into the dynamic; we leave such considerations for future work.

An interpretation of the original method is that it aims to make the norm of the residual $\mathsf{y} - \mathsf{G}(u^{(j)})$ small. Our modified version replaces this residual with $\nabla_{\mathsf{y}'}\mathcal{L}(\mathsf{G}(u^{(j)}), \mathsf{y})$, but when $\mathcal{L}$ is the cross entropy this is in fact the same (in the ℓ_1 sense). We make this precise in the following proposition.

Theorem 4. *Let* $\mathsf{G} : \mathcal{U} \to (\mathbb{P}_0^m)^N$ *and suppose* $\mathsf{y} = [e_{k_1}, \dots, e_{k_N}]^T$ *where* e_{k_j} *is the* k_j*-th standard basis vector of* $\mathbb{R}^m$*. Then* $u^* \in \mathcal{U}$ *is a solution to*

$$\arg\min_{u \in \mathcal{U}} \|\mathsf{y} - \mathsf{G}(u)\|_{\ell_1}$$

if and only if u^* *is a solution to*

$$\arg\min_{u \in \mathcal{U}} \|\nabla_{\mathsf{y}'} \mathcal{L}(\mathsf{G}(u), \mathsf{y})\|_{\ell_1}$$

where $\mathcal{L}(\mathsf{y}', \mathsf{y}) = -\langle \mathsf{y}, \log \mathsf{y}' \rangle_{\ell_2}$ *is the cross-entropy loss.*

Proof. Without loss of generality, we may assume $N = 1$ and thus let $\mathsf{y} = e_k$ be the k-th standard basis vector of $\mathbb{R}^m$. Suppose that u^* is a solution to $\arg\min_{u \in \mathcal{U}} \|\mathsf{y} - \mathsf{G}(u)\|_{\ell_1}$. Then for any $u \in \mathcal{U}$, we have

$$\sum_{j \neq k} \mathsf{G}(u^*)_j + (1 - \mathsf{G}(u^*)_k) \leq \sum_{j \neq k} \mathsf{G}(u)_j + (1 - \mathsf{G}(u)_k).$$

Adding $0 = \mathsf{G}(u^*)_k - \mathsf{G}(u^*)_k$ to the l.h.s. and $0 = \mathsf{G}(u)_k - \mathsf{G}(u)_k$ to the r.h.s. and noting that $\|\mathsf{G}(u)\|_{\ell_1} = 1$ for all $u \in \mathcal{U}$ since $\text{Im}(\mathsf{G}) = \mathbb{P}_0^m$ we obtain

$$2(1 - \mathsf{G}(u^*)_k) \leq 2(1 - \mathsf{G}(u)_k)$$

which implies

$$\frac{1}{\mathsf{G}(u^*)_k} \leq \frac{1}{\mathsf{G}(u)_k}$$

as required since $\|\nabla_{\mathsf{y}'} \mathcal{L}(\mathsf{G}(u), \mathsf{y})\|_{\ell_1} = 1/\mathsf{G}(u)_k$. The other direction follows similarly. $\square$

Momentum

Continuing in the spirit of optimization, we may also add Nesterov momentum to the EKI method. This is a simple modification to the dynamic (2.13),

$$\begin{aligned} &\ddot{u}^{(j)} + \frac{3}{t}\dot{u}^{(j)} = -C^{\text{uw}}(u)\nabla_{\mathsf{y}'}\mathcal{L}(\mathsf{G}(u^{(j)}); \mathsf{y}) \\ &u^{(j)}(0) = u_0^{(j)}, \quad \dot{u}^{(j)}(0) = 0. \end{aligned} \tag{2.14}$$

While we present momentum EKI in this form, in practice, we follow the standard in machine learning by fixing a momentum factor $\lambda \in (0, 1)$ and discretizing (2.13) using the method shown in subsection 2.4. In standard stochastic gradient decent, it has been observed that this discretization converges more quickly and possibly to a better local minima than the forward Euler discretization (Sutskever, Martens, et al., 2013). Numerically, we discover a similar speed up for EKI. However, the memory cost doubles as we need to keep track of an ensemble of positions and momenta. Some experiments in the next section demonstrate the speed-up effect. We leave analysis and possible applications to other inverse problems of the momentum method as presented in (2.14) for future work.

Discrete Scheme

Finally we present our modified EKI method in the implementable, discrete time setting and discuss some variants on this basic scheme which are particularly useful for machine learning problems. In implementation, it is useful to consider the concatenation of particles $\mathsf{u} = [u^{(1)}, \dots, u^{(J)}]$ which may be viewed as a function $\mathsf{u} : [0, \infty) \to \mathcal{U}^J$. Then (2.13) becomes

$$\dot{\mathsf{u}} = -D(\mathsf{u})\mathsf{u}$$

where for each fixed u the operator $D(\mathsf{u}) : \mathcal{U}^J \to \mathcal{U}^J$ is a linear operator. Suppose $\mathcal{U} = \mathbb{R}^P$ then we may exploit symmetry and represent $D(\mathsf{u})$ by a $J \times J$ matrix instead of a $JP \times JP$ matrix. To this end, suppose the ensemble members are stacked row-wise that is $\mathsf{u} \in \mathbb{R}^{J \times P}$ then $D(\mathsf{u})$ has the simple representation

$$(D(\mathsf{u}))_{kj} = \langle \mathsf{G}(u^{(k)}) - \bar{\mathsf{G}}, \nabla_{\mathsf{y}'} \mathcal{L}(\mathsf{G}(u^{(j)}), \mathsf{y}) \rangle$$

which is readily verified by (2.13). We then discretize via an adaptive forward Euler scheme to obtain

$$\mathsf{u}_{k+1} = \mathsf{u}_k - h_k D(\mathsf{u}_k)\mathsf{u}_k.$$

Choosing the correct time-step has an immense impact on practical performance. We have found that the choice

$$h_k = \frac{h_0}{\|D(\mathsf{u}_k)\|_F + \epsilon},$$

where $\| \cdot \|_F$ denotes the Frobenius norm, works well in practice (Dunlop, 2017). We aim to make h_0 as large as possible without loosing stability of the dynamic. The intuition behind this choice has to do with the fact that that $D(\mathsf{u})$ measures how close the propagated particles are to each other (left part of the inner-product) and how close they are to the data (right part of the inner-product). When either or both of these are small, we may take larger steps, and still retain numerical stability, by choosing h_k inversely proportional to $\|D(\mathsf{u})\|_F$; the parameter ϵ is added to avoid floating point issues when $\|D(\mathsf{u})\|_F$ is near machine precision. As $k \to \infty$, we typically match the data with increasing accuracy and, simultaneously, the propagated particles achieve consensus and collapse on one another; as a consequence $\|D(\mathsf{u}_k)\|_F \to 0$ which means we take larger and larger steps. Note that this is in contrast to the Robbins-Monro implementation of stochastic gradient descent where the sequence of time-steps are chosen to decay monotonically to zero.

Similarly, the momentum discretization of (2.13) is

$$\begin{aligned} \mathsf{u}_{k+1} &= \mathsf{v}_k - h_k D(\mathsf{v}_k)\mathsf{v}_k \\ \mathsf{v}_{k+1} &= \mathsf{u}_{k+1} + \lambda(\mathsf{u}_{k+1} - \mathsf{u}_k) \end{aligned}$$

with $\lambda \in (0,1)$ fixed, $\mathsf{u}_0 = \mathsf{v}_0$ where $h_k = h_0/(\|D(\mathsf{v}_k)\|_F + \epsilon)$ as before and v represent the particle momenta.

We now present a list of numerically successful heuristics that we employ when solving practical problems.

(I) **Initialization**: To construct the initial ensemble, we draw an i.i.d. sequence $\{u_0^{(j)}\}_{j=1}^J$ with $u_0^{(1)} \sim \mu_0$ where μ_0 is selected according to the construction discussed for initialization of the neural network model in the section outlining SGD.

(II) **Mini-batching**: We borrow from SGD the idea of mini-batching where we use only a subset of the data to compute each step of the discretized scheme, picking randomly without replacement. As in the classical SGD context, we call a cycle through the full dataset an epoch.

(III) **Prediction**: In principle, any one of the particles $u^{(j)}$ can be used as the parameters of the trained model. However, as analysis of Figure 27 below shows, the spread in their performance is quite small; furthermore even though the system is nonlinear, the mean particle $\bar{u}$ achieves an equally good performance as the individual particles. Thus, for computational simplicity, we choose to use the mean particle as our final parameter estimate. This choice further motivates one of the ways in which we randomize.

(IV) **Randomization**: The EKI property that all particles remain in the subspace spanned by the initial ensemble is not desirable when $J \ll \dim \mathcal{U}$. We break this property by introducing noise into the system. We have found two numerically successful ways of accomplishing this.

i. At each step of the discrete scheme, add noise to each particle,

$$u_k^{(j)} \mapsto u_k^{(j)} + \eta_k^{(j)}$$

where $\{\eta_k^{(j)}\}_{j=1}^J$ is an i.i.d. sequence with $\eta_k^{(1)} \sim \mu_k$. We define μ_k to be a scaled version of μ_0 by scaling its covariance operator namely $C_k = \sqrt{h_k} C_0$, where h_k is the time step as previously defined. Note that as the particles start to collapse, h_k increases, hence we add more noise to counteract this. In the momentum case, we perform the same mapping but on the particle momenta instead

$$v_k^{(j)} \mapsto v_k^{(j)} + \eta_k^{(j)}.$$

ii. At the end of each epoch, randomize the particles around their mean,

$$u_{kT}^{(j)} \mapsto \bar{u}_{kT} + \eta_{kT}^{(j)}$$

where T is the number of steps needed to complete a cycle through the entire dataset and $\{\eta_{kT}^{(j)}\}_{j=1}^J$ is an i.i.d. sequence with $\eta_{kT}^{(1)} \sim \mu_0$. Note that because this randomization is only done after a full epoch, it is not clear how the noise should be scaled and thus we simply use the prior. This may not be the optimal thing to do, but we have found great numerical success with this strategy. Figure 28 shows the spread of the ratio of the parameters to the the noise $\|\bar{u}_{kT}\|/\|\eta_{kT}^{(j)}\|$. We see that relatively less noise is added as training continues. It may be possible to achieve better results by increasing the noise with time as to combat collapse. However, we do not perform such experiments. Furthermore we have found that this does not work well in the momentum case; hence all randomization for the momentum scheme is done according to the first point.

(V) **Expanding Ensemble**: Numerical experiments show that using a small number of particles tends to have very good initial performance (one to two epochs) that quickly saturates. On the other hand, using a large number of particles does not do well to begin with but greatly outperforms small particle ensembles in the long run. Thus we use the idea of an expanding ensemble where we gradually add in new particles. This is done in the context of point (ii.) of the randomization section. Namely, at the end of an epoch, we compute the ensemble mean and create a new larger ensemble by randomizing around it.

Lastly we mention that, in many inverse problem applications, it is good practice to randomize the data for each particle at each step (K. Law, Andrew Stuart, and Zygalakis, 2015) namely map

$$\mathsf{y} \mapsto \mathsf{y} + \xi_k^{(j)}$$

where $\{\xi_k^{(j)}\}_{j=1}^J$ is an i.i.d. sequence with $\xi_k^{(1)} \sim \pi$. However we have found that this does not work well for classification problems. This may be because the given classifications are correct and there is no actual noise in the data. Such noise may thus be biased in the classification setting. We have not experimented in the case where the labels are noisy and leave this for future work.

2.5 Numerical Experiments

In the following set of experiments, we demonstrate the wide applicability of EKI on several machine learning tasks. All forward models we consider are some type of neural network, except for the semi-supervised learning case where we consider the construction in Example 2. While, for the sake of brevity, we do not give details on recurrent neural networks in this work, we refer the reader to (Goodfellow, Bengio, and Courville, 2016) for details. We benchmark EKI against SGD and momentum SGD and do not consider any other first-order adaptive methods. Recent work has shown that their value is only marginal and the solutions they find may not generalize as well (Wilson et al., 2017). Furthermore we do not employ batch normalization as it is not clear how it should be incorporated with EKI methods. However, when batch normalization is necessary, we instead use the SELU nonlinearity (Klambauer et al., 2017), finding the performance to be essentially identical to batch normalization on problems where we have been able to compare.

The next five subsections are organized as follows. Subsection 2.5 contains the conclusions drawn from the experiments. In subsection 2.5, we describe the six data sets used in all of our experiments as well as the metrics used to evaluate the methods. Subsection 2.5 gives implementation details and assigns methods using different techniques their own name. In subsections 2.5, 2.5, and 2.5 we show the supervised, semi-supervised, and online learning experiments respectively. Since most of our experiments are supervised, we split subsection 2.5 based on the type of model used namely dense neural networks, convolutional neural networks, and recurrent neural networks respectively.

Summary Of Numerical Results

The following are the primary conclusion of our numerical experiments:

- On supervised classification problems with a feed-forward neural network, EKI performs just as well as SGD even when the number of unknown parameters is up to two order of magnitude larger than the ensemble size. Furthermore EKI seems more numerically stable than SGD, as seen in the smaller amount of oscillation in the test accuracy, and requires less hyper-parameter tuning. In fact, the only parameter we vary in our experiments is the number of ensemble members, and we do this simply to demonstrate its effect. However due to the large number of forward passes required at each EKI iteration, we have found the method to be significantly slower than SGD. Due to the very efficient implementations of backpropagation a single backward-pass is comparable in wall-clock time to single forward-pass, and EKI requires J forward-passes at each iteration. This issue can be mitigated if each of the forward computations is parallelized across multiple processing units, as it often is in many industrial applications (Houtekamer, B. He, and L. Mitchell, 2014; Nino-Ruiz and Sandu, 2015; Niño, Sandu, and Deng, 2016). We leave such computational considerations for future work, as our current goal is simply to establish proof of concept. Our experiments are conducted on neural networks of up to half a million parameters which is very small compared to modern deep learning architectures. We choose such small networks to allow for rapid parallel prototyping, but note that the algorithms are inherently parallelizable and should scale well. Furthermore, as our CNN experiments point out, EKI can handle relatively deep, narrow architectures which are considered difficult to train in the machine learning literature (Romero et al., 2015; Mishkin and Matas, 2015). In addition, recent work such as (Allen-Zhu, Li, and Song, 2018) suggests that huge over-parameterization makes the associated optimization problem easier, indicating promise for EKI when used for networks with tens of millions of parameters. These experiments can be found in the first two subsections of section 2.5.

- On supervised classification problems with a recurrent neural network, EKI significantly outperforms SGD on the small problems we consider. This is likely due to the steep barriers that occur on the loss surface of recurrent networks (Pascanu, Mikolov, and Bengio, 2013; Bengio et al., 2013) which EKI may be able to avoid due to its noisy Jacobian estimates. These experiments can be found in the last subsection of section 2.5.

- On the semi-supervised learning problem we consider, EKI does not perform as well as state of the art (MCMC) (A. L. Bertozzi et al., 2017), but performs better than the naive solution. However, even with a large number of ensemble members, EKI is much faster and computationally cheaper than MCMC, allowing applications to large scale problems. These experiments can be found in section 2.5.

- On online regression problems tackled with a recurrent neural network, EKI converges significantly faster and to a better solution than SGD with $\mathcal{O}(1)$ ensemble members. While the problems we consider are only simple, univariate time-series, the results demonstrate great promise for harder problems. It has long been known that recurrent neural networks are very hard to optimize with gradient-based techniques (Pascanu, Mikolov, and Bengio, 2013), so we are very hopeful that EKI can improve on current state of the art. Again, we leave such domain specific applications to future work. These experiments can be found in section 2.5.

Data Sets

We consider four data sets where the problem at hand is classification and two data sets where it is regression. For classification, three of the data sets are comprised of images and the third of voting histories. Our goal is to classify the image based on its content or classify the voting record based on party affiliation. For regression, both datasets are univariate time-series and our goal is to predict an unobserved part of the series. Figure 23 shows samples from each of the data sets.

As outlined in section 2.2, the goal of learning is to find a model which generalizes well to unobserved data. Thus, to evaluate this criterion, we split all data sets into a training and a testing portion. The training portion is used when we let our ODE(s) evolve in time as described in section 2.4. The testing portion is used only to evaluate the model. In other contexts, the training set is further split to create a validation set, but, since we perform no hyper-parameter tuning, we omit this step. For classification, the metric we use is called *test accuracy*. This is the total number of correctly classified examples divided by the total number of examples in the test set. For regression, the metric we use is called *test error*. This is the average (across the test set) squared ℓ_2-norm of the difference between the true value and our prediction.

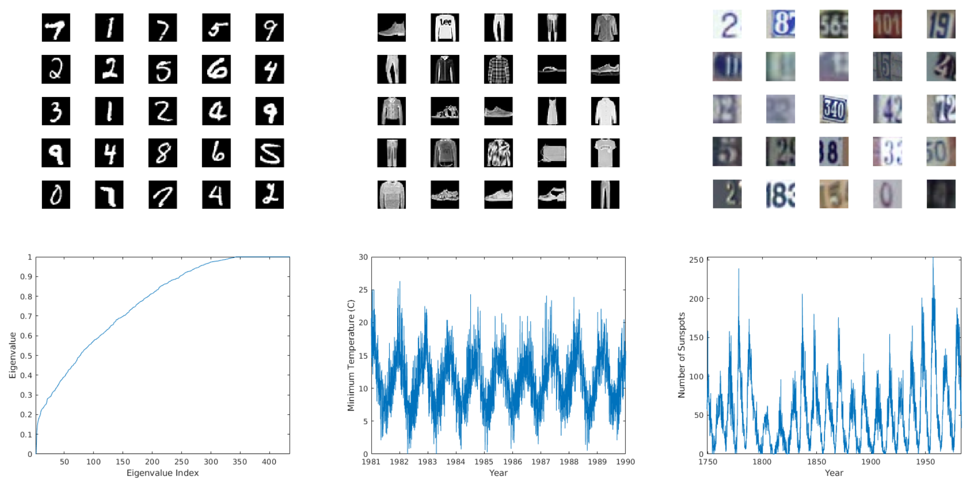


Figure 23: The six data sets used in numerical experiments. The first row shows 25 samples from MNIST, FashionMNIST, and SVHN respectively. The second row shows the spectrum of graph Lalpacian for the Voting Records data set, the full time-series for the daily minimum temperatures in Melbourne, and the monthly number of sunspots from Zürich respectively.

Classification

The first data set we consider is MNIST (LeCun and Cortes, 2010). It contains 70,000 images of hand-written digits. All examples are 28×28 grayscale images and each is given a classification in $\{0, \dots, 9\}$ depending on what digit appears in the image. Thus we consider $\mathcal{X} = \mathbb{R}^{28\times 28} \cong \mathbb{R}^{784}$ and $\mathcal{Y} = \mathbb{P}^{10}$. Each of the labels y_j is a standard basis vector of $\mathbb{R}^{10}$ with the position of the 1 indicating the digit. We use 60,000 of the images for training and 10,000 for testing. Since grayscale values range from 0 to 255, all images are fist normalized to the range $[0, 1]$ by point-wise dividing by 255. Treating all training images as a sequence of $60000 \cdot 784$ numbers, their mean and standard deviation are computed. Each image (including the test set) is then again normalized via point-wise subtraction by the mean and point-wise division by the standard deviation. This data normalization technique is standard in machine learning.

The second image data set we consider is FashionMNIST (Xiao, Rasul, and Vollgraf, 2017). It contains 70,000 images of different types of clothing items. All examples are 28×28 grayscale images and each is given a classification in $\{0, \dots, 9\}$ depending on the type of clothing item pictured. We treat it in the exact same way that we treat MNIST.

The third image data set we consider is SVHN (Netzer et al., 2011). It contains

99,289 natural images of cropped house numbers taken from Google Street View. All examples are 32×32 RGB images and each is given a classification in $\{0, \ldots, 9\}$ depending on what digit appears in the image. Thus we consider $\mathcal{X} = \mathbb{R}^{3\times 32\times 32} \cong \mathbb{R}^{3072}$ and $\mathcal{Y} = \mathbb{P}^{10}$ with the labels again being basis vectors of $\mathbb{R}^{10}$. We use 73,257 of the images for training and 26,032 for testing. All values are first normalized to be in the range $[0, 1]$. We then perform the same normalization as in MNIST, but this time per channel. That is, for all training images, we treat each color channel as a sequence of $73257 \cdot 1024$ numbers, compute the mean and standard deviation and then normalize each channel as before.

The last data set for classification we consider contains the voting record of the 435 U.S. House of Representatives members; see (A. Bertozzi and Flenner, 2012) and references therein. The votes were recorded in 1984 from the 98th United States Congress, 2nd session. Each record is tied to a particular representative and is a vector in $\mathcal{X} = \mathbb{R}^{16}$ with each entry being $+1$, -1, or 0 indicating a vote for, against, or abstain respectively. The labels live in $\mathcal{Y} = \mathbb{R}$ and are $+1$ or -1 indicating Democrat or Republican respectively. We use this data set only for semi-supervised learning and thus pick the amount of observed labels $|Z'| = 5$ with 2 Republicans and 3 Democrats. No normalization is performed. When computing the test accuracy, we do so over the entire data sets—namely we do not remove the 5 observed records.

Regression

The first data set we consider for regression is a time series of the daily minimum temperatures (in Celsius) in the Melbourne, Australia from January 1st 1981 to December 31st 1990 (Gil-Alana, 2006). It contains 3650 total observations of which we use the first 3001 for training (up to March 22nd 1989) and the rest for testing. We consider $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ by letting (in the training set) the data be the first 3000 observations and the labels be the 2nd to 3001st observations i.e. a one-step-ahead split. The same is done for the testing set. The minimum and maximum values x_{min}, x_{max} over the training set are computed and all data is transformed via

$$x_j \mapsto \frac{x_j - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}.$$

This ensures the training set is in the range $[0, 1]$ and the testing set will also be close to that range.

The second data set for regression is a time series containing the number of observed sunspots from Zürich, Switzerland during each month from January 1749 to December 1983 (Andrews and Herzberg, 1985). It contains 2820 observations of which we use the first 2301 for training (up to September 1915) and the rest for testing. The data is treated in exactly the same way as the temperatures data set.

Implementation Details

Having outlined many different strategies for performing EKI , we give methods using different techniques their own name so they are easily distinguishable. We refer to the techniques listed in section 2.4. All methods are initialized in the same way (with the prior constructed based on the model) and all use mini-batching. We refer to the forward Euler discretization of equation (2.13) as EKI and the momentum discretization, presented in section 2.4, of equation (2.13) as MEKI. When randomizing around the mean at the end of each epoch, we refer to the method as EKI(R). When randomizing the momenta at each step, we refer to the method as MEKI(R). Similarly, we call momentum SGD, MSGD. All methods use the time step described in section 2.4 with hyper-parameters $h_0 = 2$ and $\epsilon = 0.5$ fixed. For any classification problem (except the Voting Records data set), all methods use the cross-entropy loss whose gradient is implemented with a slight correction for numerical stability. Namely, in the case of a single data point, we implement

$$(\nabla_{y'} \mathcal{L}(\mathcal{G}(u), y))_k = -\frac{y_k}{(\mathcal{G}(u))_k + \delta}$$

where the constant $\delta := 0.005$ is fixed for all our numerical experiments. Otherwise the mean squared-error loss is used. All implementations are done using the PyTorch framework (Paszke et al., 2017) on a single machine with an NVIDIA GTX 1080 Ti GPU.

Supervised Learning

Dense Neural Networks

In this section, we benchmark all of our proposed methods on the MNIST problem using four dense neural networks of increasing complexity. The four network architectures are outlined in Figure 24. This will allow us to compare the methods and pick a front runner for later experiments. Furthermore we address scalability by finding the minimum number of ensemble members required to reach a certain accuracy on a dataset (standard set by SGD) for networks with increasing number of parameters. These experiments are performed with a two-layer dense network on MNIST and FashionMNIST. Lastly, we train a dense network with Heaviside activations (impossible with SGD) on FashionMNIST and compare to the equivalent

Dense Neural Networks		
Name	Architecture	Parameters
DNN 1	784-10	7,850
DNN 2	784-100-10	79,510
DNN 3	784-300-100-10	266,610
DNN 4	784-500-300-100-10	573,910

Figure 24: Architectures of the four dense neural networks considered. All networks use a softmax thresholding and a ReLU nonlinearity.

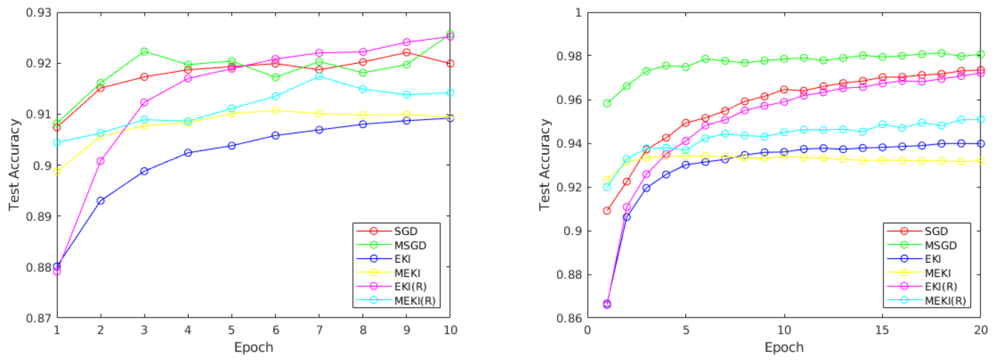
	DNN 1	DNN 2	DNN 3	DNN 4
SGD	0.9199	0.9735	0.9798	0.9818
MSGD	0.9257	**0.9807**	**0.9830**	**0.9840**
EKI	$\bar{u}$ 0.9092 $u^{(j^*)}$ 0.9114	$\bar{u}$ 0.9398 $u^{(j^*)}$ 0.9416	$\bar{u}$ 0.9424 $u^{(j^*)}$ 0.9432	$\bar{u}$ 0.9404 $u^{(j^*)}$ 0.9418
MEKI	$\bar{u}$ 0.9094 $u^{(j^*)}$ 0.9107	$\bar{u}$ 0.9320 $u^{(j^*)}$ 0.9332	n/a	n/a
EKI(R)	$\bar{u}$ 0.9252 $u^{(j^*)}$ **0.9260**	$\bar{u}$ 0.9721 $u^{(j^*)}$ 0.9695	$\bar{u}$ 0.9738 $u^{(j^*)}$ 0.9716	$\bar{u}$ 0.9741 $u^{(j^*)}$ 0.9691
MEKI(R)	$\bar{u}$ 0.9142 $u^{(j^*)}$ 0.9162	$\bar{u}$ 0.9509 $u^{(j^*)}$ 0.9511	n/a	n/a

Figure 25: Final test accuracies of six training methods on four dense neural networks, solving the MNIST classification problem. Each bold number is the maximum across the column. For each EKI method we report the accuracy of the mean particle $\bar{u}$ and of the best performing particle in the ensemble $u^{(j^*)}$.

ReLU network.

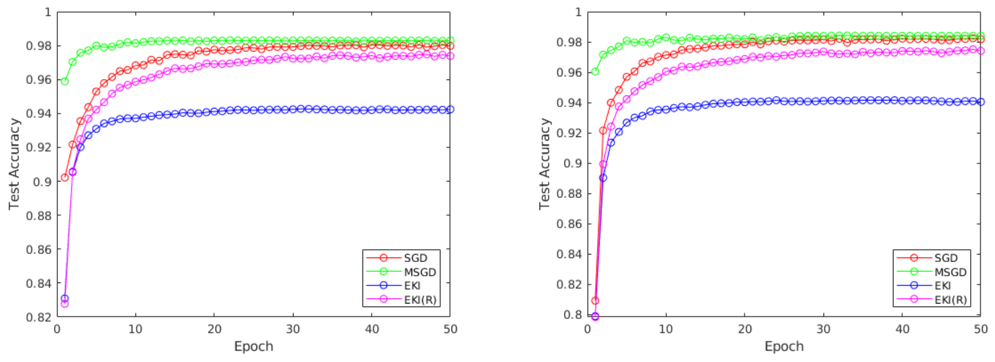
We fix the ensemble size of all methods to $J = 2000$ and the batch size to 600. SGD uses a learning rate of 0.1 and all momentum methods use the constant $\lambda = 0.9$. Figure 25 shows the final test accuracies for all methods while Figure 26 shows the accuracies at the end of each epoch. Due to memory constrains, we do not implement MEKI for DNN-(3,4). In general momentum SGD performs best, but EKI(R) trails closely. The momentum EKI methods have good initial performance but saturate. We make this clearer in a later experiment. Overall, we see that for networks with a relatively small number of parameters all EKI methods are comparable to SGD. However with a large number of parameters, randomization is needed. This effect is particularly dominant when the ensemble size is relatively small; as we later show, larger ensemble sizes can perform significantly better.

Figure 27 shows the test accuracies for each of the particles when using EKI on DNN-(1,2). We see that, the mean particle achieves roughly the average of the



(a) DNN 1

(b) DNN 2



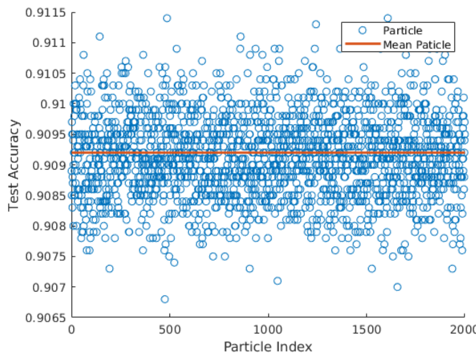
(c) DNN 3

(d) DNN 4

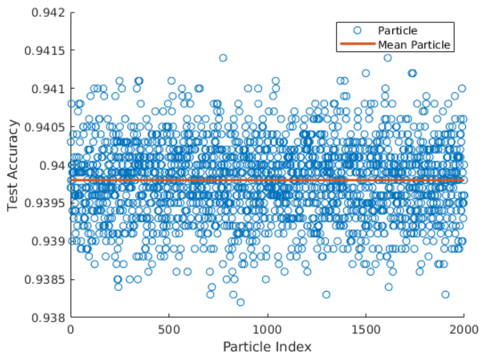
Figure 26: Test accuracies per epoch of six training methods on four dense neural networks, solving the MNIST classification problem. For each EKI method the accuracy of the mean particle $\bar{u}$ is shown.

spread, as previously discussed. Our choice to use it as the final parameter estimate is simply for convenience. One may use all the particles in a carefully weighted scheme as an ensemble of networks and possibly achieve better results. Having many parameter estimates may also be advantageous when trying to avoid adversarial examples (Goodfellow, Shlens, and Szegedy, 2015). We leave these considerations to future work.

To better illustrate the effect of the ensemble size, we compare all EKI methods on DNN 2 with an ensemble size of $J = 6000$. The accuracies are shown in Figure 29. We again observe that the momentum methods perform very well initially, but fall off with more training. This effect could be related to the specific time discretization method we use, but needs to be studied further theoretically and we leave this for future work. Note that with a larger ensemble, EKI is now comparable to SGD pointing out that remaining in the subspace spanned by the initial ensemble is a bottle neck for this method. On the other hand, when we randomize, the ensemble

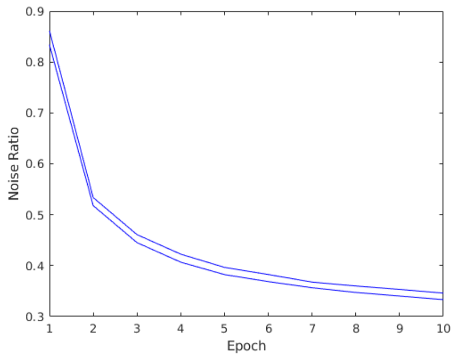


(a) DNN 1

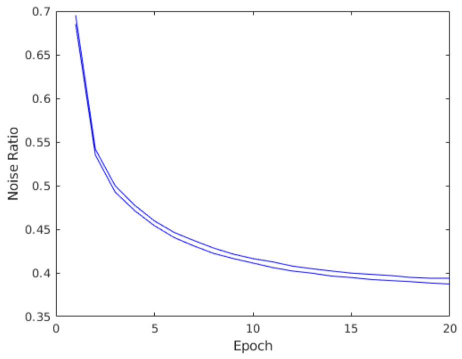


(b) DNN 2

Figure 27: Particle accuracies of EKI on DNN-(1,2) compared to the accuracy of the mean particle $\bar{u}$.



(a) DNN 1

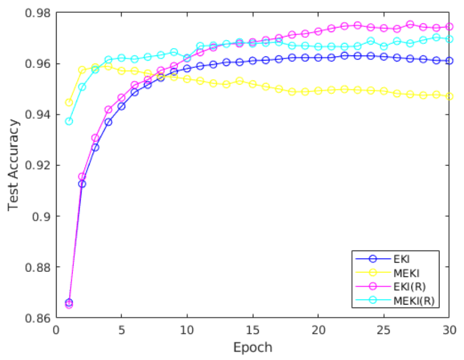


(b) DNN 2

Figure 28: Spread of the noise ratio for EKI(R) on DNN-(1,2). At the end of every epoch, when the noise is added, the upper bound is computed as $\|\bar{u}\|_2 / \max_j \|\eta^{(j)}\|_2$. The lower bound is computed analogously.

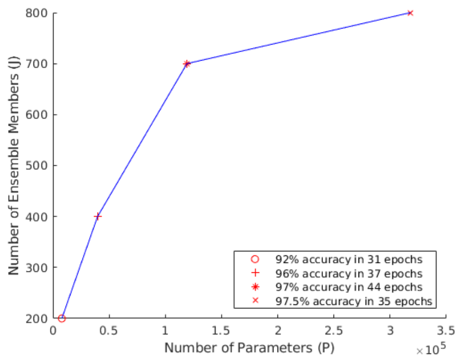
size is no longer so relevant. EKI(R) performs almost identically with 2,000 and with 6,000 ensemble members. Finding it to be the best method for these tasks, all experiments hereafter, unless stated otherwise, use EKI(R).

To address scalability, we fix a network architecture (two-layer ReLU network) and find the smallest number of ensemble members J needed for EKI(R) to reach the maximum accuracy that can be achieved by this network given a fixed computational budget of 50 epochs. This accuracy is approximated via SGD on the same computational budget. We then increase the number of parameters in the network (keeping the architecture fixed) and repeat this experiment. The results are summarized in Figure 210 for MNIST and FashionMNIST. We see that relatively few ensemble members are needed to reach the desired accuracy and the scaling with the number

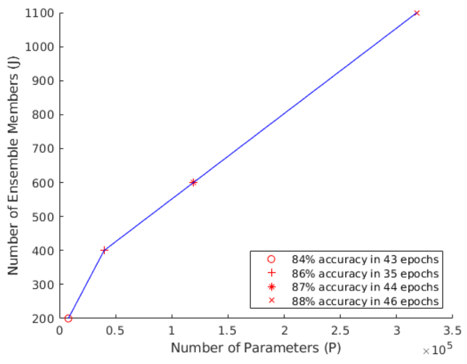


	First	Final
EKI	0.8661	0.9611
MEKI	**0.9447**	0.9471
EKI(R)	0.8652	**0.9745**
MEKI(R)	0.9373	0.9696

Figure 29: Comparison of the test accuracies of four EKI methods on DNN 2 with ensemble size $J = 6000$.



(a) MNIST



(b) FashionMNIST

Figure 210: Minimum number of ensemble members needed to reach network capacity for a two-layer ReLU network within 50 epochs on MNIST and FashionMNIST. We increase the number of parameters in the first hidden layer and show the minimum J as a function of P.

of parameters seems linear or sublinear. This is a promising indication that the EKI methodology can be scaled up to industrial-sized neural networks.

Lastly, since our method is derivative-free, we train a 784-100-10 network on FashionMNIST with Heaviside activation functions. This is impossible with gradient-based methods since the derivative of the Heaviside function is zero almost everywhere. This architecture is akin to Rosenblatt's original multi-layered perceptron (Rosenblatt, 1958). We achieve 85% test set accuracy on this task which is comparable to the 87% achieved by the equivalent ReLU network. While our experiment is very simple, it demonstrates the possibility of being able to train non-differentiable neural networks which opens new avenues for network design that we hope will be explored in the future.

Convolutional Neural Networks

For our experiments with CNN(s), we employ both MNIST and SVHN. Since MNIST is a fairly easy data set, we can use a simple architecture and still achieve almost perfect accuracy. We name the model CNN-MNIST and its specifics are given in the first column of Figure 211. SGD uses a learning rate of 0.05 while momentum SGD uses 0.01 and a momentum factor of 0.9. EKI(R) has a fixed ensemble size of $J = 2000$. Figure 212 shows the results of training. We note that since CNN-MNIST uses ReLU and no batch normalization, SGD struggles to find a good descent direction in the first few epochs. EKI(R), on the other hand, does not have this issue and exhibits a smooth test accuracy curve that is consistent with all other experiments. In only 30 epochs, we are able to achieve almost perfect classification with EKI(R) slightly outperforming the SGD-based methods.

Recent work suggests that the effectiveness of batch normalization does not come from dealing with the internal covariate shift, but, in fact, comes from smoothing the loss surface (Santurkar et al., 2018). The noisy gradient estimates in EKI can be interpreted as doing the same thing and is perhaps the reason we see smoother test accuracy curves. The contemporaneous work of Haber et al (Haber, Lucka, and Ruthotto, 2018) further supports this point of view.

Next we experiment on the SVHN data set with three CNN(s) of increasing complexity. The architectures we use are inspired by those in (Mishkin and Matas, 2015), and are referred to as Fit-Nets because each layer is shallow (has a relatively small number of parameters), but the whole architecture is deep, reaching up to sixteen layers. The details for the models dubbed CNN-(1,2,3) are given in Figure 211. Such models are known to be difficult to train; for this reason, the papers (Romero et al., 2015; Mishkin and Matas, 2015) present special initialization strategies to deal with the model complexities. We find that when using the SELU nonlinearity and no batch normalization, simple Xavier initialization works just as well. The results of training are presented in Figure 212. We benchmark only against momentum SGD as all previous experiments show it performs better than vanilla SGD. The method uses a learning rate of 0.01 and a momentum factor of 0.9. EKI(R) starts with $J = 200$ ensemble members and expands by 200 at end the of each epoch until reaching a final ensemble of $J = 5000$. For CNN-3, memory constraints allowed us to only expand up a final size of $J = 2800$. All methods use a batch size of 500. We see that, in all three cases, EKI(R) and momentum SGD perform almost identically with EKI(R) slightly outperforming on CNN-(1,2), but falling off on

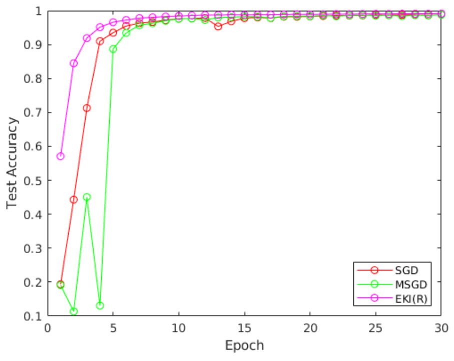
Convolutional Neural Networks			
CNN-MNIST	CNN-1	CNN-2	CNN-3
Conv 16x3x3	Conv 16x3x3	Conv 16x3x3	Conv 16x3x3
Conv 16x3x3	Conv 16x3x3	Conv 16x3x3	Conv 16x3x3
		Conv 16x3x3	Conv 32x3x3
			Conv 32x3x3
			Conv 32x3x3
MaxPool 4x4 $(s = 2)$	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2
Conv 16x3x3	Conv 16x3x3	Conv 32x3x3	Conv 48x3x3
Conv 16x3x3	Conv 16x3x3	Conv 32x3x3	Conv 48x3x3
		Conv 32x3x3	Conv 48x3x3
			Conv 48x3x3
			Conv 48x3x3
MaxPool 4x4 $(s = 2)$	MaxPool 2x2	MaxPool 2x2	MaxPool 2x2
Conv 12x3x3	Conv 32x3x3	Conv 48x3x3	Conv 64x3x3
Conv 12x3x3	Conv 32x3x3	Conv 48x3x3	Conv 64x3x3
		Conv 64x3x3	Conv 96x3x3
			Conv 96x3x3
			Conv 96x3x3
MaxPool 2x2	MaxPool 8x8	MaxPool 8x8	MaxPool 8x8
FC-10	FC-500	FC-500	FC-500
	FC-10	FC-10	FC-10

Figure 211: Architectures of four Convolutional Neural Networks with 6, 7, 10, and 16 layers respectively from left to right. All convolutions use a padding of 1, making them dimension preserving since all kernel sizes are 3x3. CNN-MNIST is evaluated on the MNIST dataset and uses the ReLU nonlinearity. CNN-(1,2,3) are evaluated on the SVHN dataset and use the SELU nonlinearity. The convention $s = 2$ refers to the stride of the max-pooling operation namely $\alpha = \beta = 2$. All networks use a softmax thresholding.

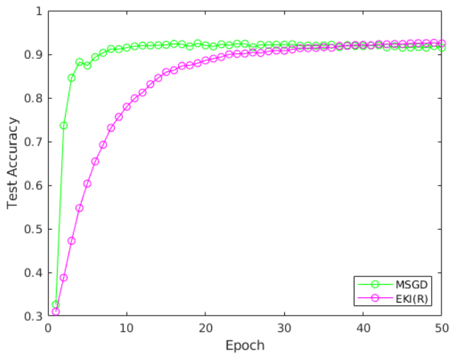
CNN-3. This is likely due to the fact that CNN-3 has a large number of parameters and we were not able to provide a large enough ensemble size. This issue can be dealt with via parallelization by splitting the ensemble among the memory banks of separate processing units. We leave this consideration to future work.

Recurrent Neural Networks

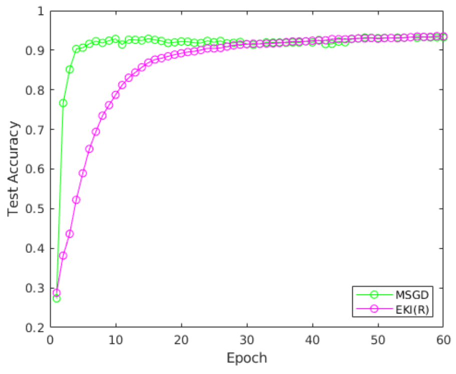
For the classification task using a recurrent neural network, we return to the MNIST data set. Since recurrent networks work on time series data, we split each image along its rows, making a 28-dimensional time sequence with 28 entries, considering time going down from the top to the bottom of the image. More complex strategies



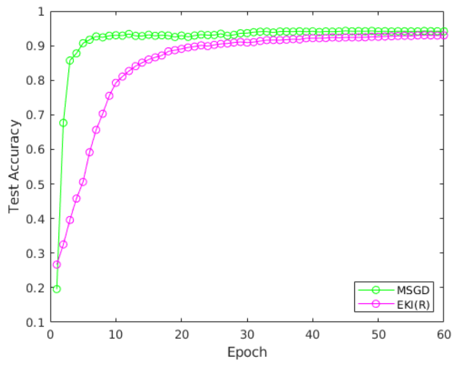
(a) CNN-MNIST



(b) CNN-1



(c) CNN-2



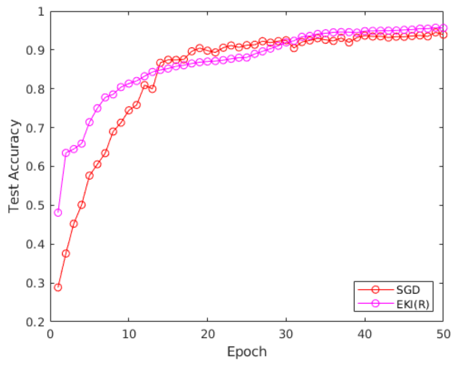
(d) CNN-3

	CNN-MNIST		CNN-1		CNN-2		CNN-3	
	First	Final	First	Final	First	Final	First	Final
SGD	0.1936	0.9878	n/a	n/a	n/a	n/a	n/a	n/a
MSGD	0.1911	0.9880	**0.3263**	0.9150	0.2734	0.9324	0.1959	**0.9414**
EKI(R)	**0.5708**	**0.9912**	0.3100	**0.9249**	**0.2874**	**0.9353**	**0.2668**	0.9299

Figure 212: Comparison of the test accuracies of SGD and EKI(R) on four convolutional neural networks. SGD(M) refers to momentum SGD. CNN-MNIST is trained on the MNIST data set, while CNN-(1,2,3) are trained on the SVHN data set.

have been explored in (J. Wang et al., 2016). We use a two-layer recurrent network with 32 hidden units and a $\tanh$ nonlinearity. Softmax thresholding is applied and the initial hidden state is always taken to be 0.

We train with a batch size of 600 and SGD uses a learning rate of 0.05. EKI(R) starts with an ensemble size of $J = 1000$ and expands by 1,000 at the end of every epoch until $J = 4000$ is reached. Figure 213 shows the result of training. EKI(R) performs significantly better than SGD and appears more reliable, overall, for this task.



	First	Final
SGD	0.2825	0.9391
EKI(R)	**0.4810**	**0.9566**

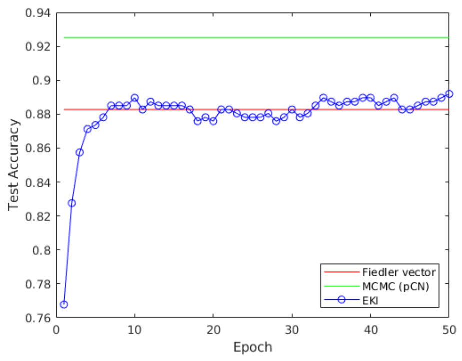
Figure 213: Comparison of the test accuracies of EKI(R) and SGD on the MNIST data set with a two layer recurrent neural network.

Semi-supervised Learning

We proceed as in the construction of Example 2, using the Voting Records data set. For the affinity measure we pick (Zelnik-manor and Perona, 2005; A. L. Bertozzi et al., 2017)

$$\eta(x, y) = \exp\left(-\frac{\|x - y\|_2^2}{2(1.25)^2}\right)$$

and construct the graph Laplacian $\mathsf{L}(\mathsf{x})$. Its spectrum is shown in Figure 23. Further we let $\tau = 0$ and $\alpha = 1$, hence the prior covariance $C = (\mathsf{L}(\mathsf{x}))^{-1}$ is defined only on the subspace orthogonal to the first eigenvector of $\mathsf{L}(\mathsf{x})$. The most naive clustering algorithm that uses a graph Laplacian simply thresholds the eigenvector of $\mathsf{L}(\mathsf{x})$ that corresponds to the smalled non-zero eigenvalue (called the Fiedler vector) (Luxburg, 2007). Its accuracy is shown in Figure 214. We found the best performing EKI method for this problem to simply be the vanilla version of the method, i.e. no randomization or momentum. We use $J = 1000$ ensemble members drawn from the prior and the mean squared-error loss. Its performance is only slightly better than the Fiedler vector as the particles quickly collapse to something close to the Fiedler vector. This is likely due to the fact that the initial ensemble is an i.i.d. sequence drawn from the prior hence EKI converges to a solution in the subspace orthogonal to the first eigenvector of $\mathsf{L}(\mathsf{x})$ which is close to the Fiedler vector, especially if the weights and other attendant hyper-parameters have been chosen so that the Fielder vector already classifies the labeled nodes correctly. On the other hand, the MCMC method detailed in (A. L. Bertozzi et al., 2017) can explore outside of this subspace and achieve much better results. We note, however, that EKI is significantly cheaper and faster than MCMC and thus could be applied to much larger problems where MCMC is not computationally feasible.



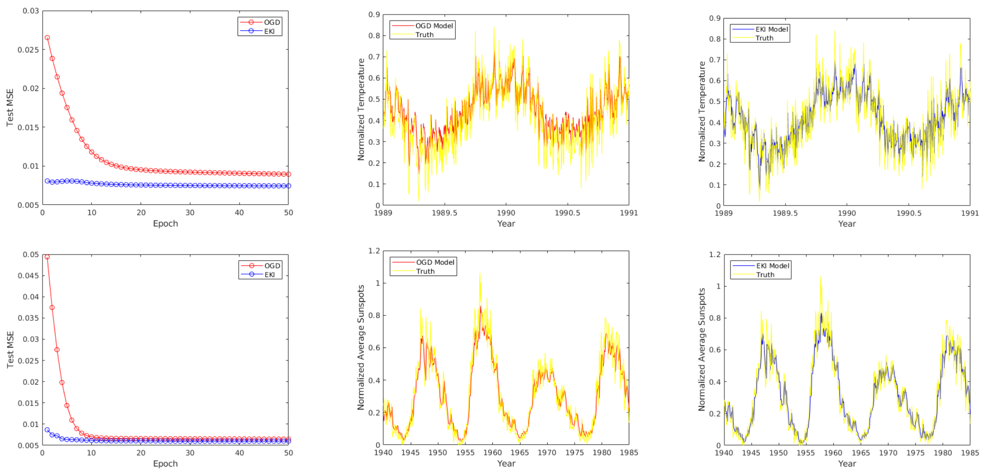
	Accuracy
Fiedler vector	0.8828
MCMC (pCN)	**0.9252**
EKI	0.8920

Figure 214: Comparison of the test accuracies of two semi-supervised learning algorithms to EKI on the Voting Records data set.

Online Learning

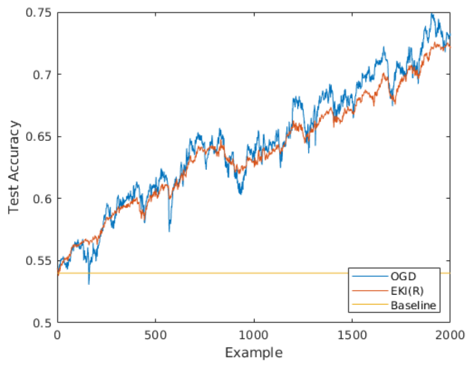
We now consider two online learning problems using a recurrent neural network. We employ two univariate time-series data sets: minimum daily temperatures in Melbourne, and the monthly number of sunspots observed from Zürich. For both, we use a single layer recurrent network with 32 hidden units and the $\tanh$ nonlinearity. The output is not thresholded i.e. $S = id$. At the initial time, we set the hidden state to 0 then use the hidden state computed in the previous step to initialize for the current step. This is an online problem as our algorithm only sees one data-label pair at a time. For OGD, we use a learning rate of 0.001 while, for EKI, we use $J = 12$ ensemble members. Figure 215 shows the results of training as well as how well each of the trained model fits the test data. Notice that EKI converges much more quickly and to a slightly better solution than OGD in both cases. Furthermore, the model learned by EKI is able to better capture small scale oscillations. These are very promising results for the application of EKI to harder RNN problems.

Finally we consider an online version of the classification problem. We train a CNN (using the architecture of CNN-MNIST given in Figure 211) on MNIST and FashionMNIST for using only 10 training examples from each data set. The performance of this network on the test set is taken as a baseline. We then feed in 2,000 examples from the training set sequentially one at a time (equivalent to a batch size of one) and update our model with OGD and EKI ($J = 2000$), comparing their performance in Figure 216. We see that the two methods perform similarly, with EKI(R) trailing slightly behind in accuracy but having an overall lower variance. These results also confirm our intuition that EKI may be viewed as approximating gradient decent.

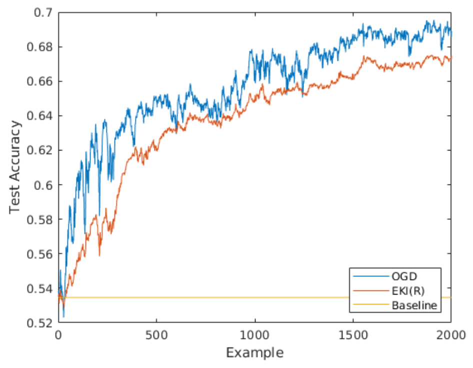


	Melbourne Temperatures		Zürich Sunspots	
	First	Final	First	Final
OGD	2.653×10^{-2}	8.954×10^{-3}	4.939×10^{-2}	6.480×10^{-3}
EKI	$\mathbf{8.086 \times 10^{-3}}$	$\mathbf{7.448 \times 10^{-3}}$	$\mathbf{8.671 \times 10^{-3}}$	$\mathbf{6.006 \times 10^{-3}}$

Figure 215: Comparison of OGD and EKI on two online learning tasks with a recurrent neural network. The top row shows the minimum daily temperatures in Melbourne data set, while the bottom shows the number of sunspots observed each month from Zürich data set.



(a) MNIST



(b) FashionMNIST

Figure 216: Comparison of OGD and EKI on the online classification task with MNIST and FashionMNIST.

2.6 Conclusion and Future Directions

We have demonstrated that many machine learning problems can easily fit into the unified framework of Bayesian inverse problems. Within this framework, we apply ensemble Kalman inversion methods, for which we suggest suitable modifications, to tackle machine learning tasks. Our numerical experiments suggest a wide applicability and competitiveness against the state-of-the-art for our schemes. The following directions for future research arise naturally from our work:

- Theoretical analysis of the momentum and general loss EKI methods as well as their possible application to physical inverse problems.
- GPU parallelization of EKI methods and its application to large scale machine learning tasks.
- Application of EKI methods to more difficult recurrent neural network problems as well as problems in reinforcement learning.
- Use of the entire ensemble of particle estimates to improve accuracy, perform dimension reduction, and possibly combat adversarial examples.
- The development of Bayesian techniques to quantify uncertainty in trained neural networks.
- The use of ensemble methods in other momentum based algorithms such as Hamiltonian Monte Carlo.

References

Allen-Zhu, Zeyuan, Yuanzhi Li, and Zhao Song (2018). "A Convergence Theory for Deep Learning via Over-Parameterization". In: *CoRR* abs/1811.03962. arXiv: `1811.03962`. URL: `http://arxiv.org/abs/1811.03962`.

Andrews, D.F. and A.M. Herzberg (1985). *Data: a collection of problems from many fields for the student and research worker*. Springer series in statistics. Springer. ISBN: 9783540961253. URL: `https://books.google.com/books?id=dCvvAAAAMAAJ`.

Ba, Lei Jimmy, Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization". In: *CoRR* abs/1607.06450. arXiv: `1607.06450`. URL: `http://arxiv.org/abs/1607.06450`.

Bach, Francis and Eric Moulines (2013). "Non-strongly-convex smooth stochastic approximation with convergence rate O (1/n)". In: *Advances in neural information processing systems*, pp. 773–781.

Bengio, Yoshua et al. (2013). "Advances in optimizing recurrent networks". In: *In Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*.

Bergemann, Kay and Sebastian Reich (July 2010). "A mollified Ensemble Kalman filter". In: 136.

Bertozzi, A. and A. Flenner (2012). “Diffuse Interface Models on Graphs for Classification of High Dimensional Data”. In: *Multiscale Modeling & Simulation* 10.3, pp. 1090–1118. DOI: 10.1137/11083109X. eprint: https://doi.org/10.1137/11083109X. URL: https://doi.org/10.1137/11083109X.

Bertozzi, Andrea L. et al. (2017). “Uncertainty Quantification in the Classification of High Dimensional Data”. In: *CoRR* abs/1703.08816. arXiv: 1703.08816. URL: http://arxiv.org/abs/1703.08816.

Binkowski, Mikolaj, Gautier Marti, and Philippe Donnat (2017). “Autoregressive Convolutional Neural Networks for Asynchronous Time Series”. In: *CoRR* abs/1703.04122. arXiv: 1703.04122. URL: http://arxiv.org/abs/1703.04122.

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik (1992). “A Training Algorithm for Optimal Margin Classifiers”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT ’92. Pittsburgh, Pennsylvania, USA: ACM, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

Candes, E. J. and T. Tao (Dec. 2006). “Near-Optimal Signal Recovery From Random Projections: Universal Encoding Strategies?” In: *IEEE Trans. Inf. Theor.* 52.12, pp. 5406–5425. ISSN: 0018-9448. DOI: 10.1109/TIT.2006.885507. URL: https://doi.org/10.1109/TIT.2006.885507.

Carreira-Perpinan, Miguel and Weiran Wang (2014). “Distributed optimization of deeply nested systems”. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Ed. by Samuel Kaski and Jukka Corander. Vol. 33. Proceedings of Machine Learning Research. Reykjavik, Iceland: PMLR, pp. 10–19.

Chaudhari, Pratik, Anna Choromanska, et al. (2016). “Entropy-SGD: Biasing Gradient Descent Into Wide Valleys”. In: *CoRR* abs/1611.01838. arXiv: 1611.01838. URL: http://arxiv.org/abs/1611.01838.

Chaudhari, Pratik and Stefano Soatto (2017). “Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks”. In: *CoRR* abs/1710.11029. arXiv: 1710.11029. URL: http://arxiv.org/abs/1710.11029.

De Vito, Ernesto et al. (2005). “Learning from examples as an inverse problem”. English. In: *Journal of Machine Learning Research* 6. ISSN: 1532-4435.

Dieuleveut, Aymeric, Francis Bach, et al. (2016). “Nonparametric stochastic approximation with large step-sizes”. In: *The Annals of Statistics* 44.4, pp. 1363–1399.

Duchi, John, Elad Hazan, and Yoram Singer (July 2011). “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *J. Mach. Learn. Res.* 12, pp. 2121–2159. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2021068.

Dunlop, Matt (2017). Private Communication.

Evensen, Geir (Nov. 2003). “The Ensemble Kalman Filter: theoretical formulation and practical implementation”. In: *Ocean Dynamics* 53.4, pp. 343–367. ISSN: 1616-7228. DOI: 10.1007/s10236-003-0036-9. URL: https://doi.org/10.1007/s10236-003-0036-9.

G. Ernst, Oliver, Björn Sprungk, and Hans-Jörg Starkloff (Apr. 2015). “Analysis of the Ensemble and Polynomial Chaos Kalman Filters in Bayesian Inverse Problems”. In: 3.

Gil-Alana, L.A. (2006). “Long memory behaviour in the daily maximum and minimum temperatures in Melbourne, Australia”. In: *Meteorologocal Applications* 11.4, pp. 319–328. eprint: https://doi.org/10.1017/S1350482704001422. URL: https://doi.org/10.1017/S1350482704001422.

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10). Society for Artificial Intelligence and Statistics*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Goodfellow, Ian, Jonathon Shlens, and Christian Szegedy (2015). “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations*. URL: http://arxiv.org/abs/1412.6572.

Graham, Benjamin (2014). “Fractional Max-Pooling”. In: *CoRR* abs/1412.6071. arXiv: 1412.6071. URL: http://arxiv.org/abs/1412.6071.

Gulcehre, Caglar et al. (2014). “Learned-Norm Pooling for Deep Feedforward and Recurrent Neural Networks”. In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 8724*. ECML PKDD 2014. Nancy, France: Springer-Verlag, pp. 530–546. ISBN: 978-3-662-44847-2. DOI: 10.1007/978-3-662-44848-9_34. URL: https://doi.org/10.1007/978-3-662-44848-9_34.

Haber, Eldad, Felix Lucka, and Lars Ruthotto (2018). “Never look back - A modified EnKF method and its application to the training of neural networks without back propagation”. In: *CoRR* abs/1805.08034. arXiv: 1805.08034. URL: https://arxiv.org/abs/1805.08034.

Haykin, Simon S. (2001). *Kalman Filtering and Neural Networks*. New York, NY, USA: John Wiley & Sons, Inc. ISBN: 0471369985.

He, Kaiming et al. (2014). “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition”. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, pp. 346–361. ISBN: 978-3-319-10578-9.

Hinton, Geoffrey and Ruslan Salakhutdinov (2006). “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786, pp. 504–507.

Hochreiter, S. et al. (2001). “Gradient flow in recurrent nets: the difficulty of learning long-term dependencies”. In: *A Field Guide to Dynamical Recurrent Neural Networks*. Ed. by Kremer and Kolen. IEEE Press.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). “Long Short-term Memory”. In: *Neural Comput.* 9.9, pp. 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. URL: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

Hoerl, Arthur E. and Robert W. Kennard (1970). “Ridge Regression: Biased Estimation for Nonorthogonal Problems”. In: *Technometrics* 12.1, pp. 55–67. ISSN: 00401706. URL: `http://www.jstor.org/stable/1267351`.

Hornik, Kurt (Mar. 1991). “Approximation Capabilities of Multilayer Feedforward Networks”. In: *Neural Netw.* 4.2, pp. 251–257. ISSN: 0893-6080. DOI: `10.1016/0893-6080(91)90009-T`. URL: `http://dx.doi.org/10.1016/0893-6080(91)90009-T`.

Houtekamer, Pieter, Bin He, and Herschel L. Mitchell (Feb. 2014). “Parallel Implementation of an Ensemble Kalman Filter”. In: 142.

Iglesias, M., Y. Lu, and A.M. Stuart (2016). “A Bayesian level set method for geometric inverse problems”. In: *Interfaces and Free Boundaries* 18 (2), pp. 181–217. DOI: `10.4171/IFB/362`.

Iglesias, Marco A, Kody J H Law, and Andrew M Stuart (2013). “Ensemble Kalman methods for inverse problems”. In: *Inverse Problems* 29.4, p. 045001. URL: `http://stacks.iop.org/0266-5611/29/i=4/a=045001`.

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France: JMLR.org, pp. 448–456. URL: `http://dl.acm.org/citation.cfm?id=3045118.3045167`.

Jordan, Michael (2017). “On Gradient-Based Optimization: Accelerated, Distributed, Asynchronous and Stochastic”. In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 45. 1. ACM, pp. 58–58.

Kay, Bergemann and Reich Sebastian (n.d.). “A localization technique for ensemble Kalman filters”. In: *Quarterly Journal of the Royal Meteorological Society* 136.648 (), pp. 701–707. DOI: `10.1002/qj.591`. eprint: `https://rmets.onlinelibrary.wiley.com/doi/pdf/10.1002/qj.591`.

URL: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.591.

Kiefer, J. and J. Wolfowitz (Sept. 1952). “Stochastic Estimation of the Maximum of a Regression Function”. In: *Ann. Math. Statist.* 23.3, pp. 462–466. DOI: 10.1214/aoms/1177729392. URL: https://doi.org/10.1214/aoms/1177729392.

Kingma, Diederik P. and Jimmy Ba (2014). “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

Klambauer, Günter et al. (2017). “Self-Normalizing Neural Networks”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 971–980. URL: http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf.

Kloeden, Peter and Martin Rasmussen (Aug. 2011). *Nonautonomous Dynamical Systems*. Vol. 176. ISBN: 978-0-8218-6871-3.

Kovachki, Nikola B. and Andrew M. Stuart (2019). “Analysis Of Momentum Methods”. In: *Preprint*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

Law, Kody, Andrew Stuart, and Konstantinos Zygalakis (2015). *Data assimilation: A mathematical introduction*. Vol. 62. Texts in Applied Mathematics. Springer, Cham. ISBN: 978-3-319-20324-9; 978-3-319-20325-6. DOI: 10.1007/978-3-319-20325-6.

LeCun, Yann, Yoshua Bengio, and Geoffrey E. Hinton (2015). “Deep learning”. In: *Nature* 521.7553, pp. 436–444. DOI: 10.1038/nature14539. URL: https://doi.org/10.1038/nature14539.

LeCun, Yann and Corinna Cortes (2010). “MNIST handwritten digit database”. In: URL: http://yann.lecun.com/exdb/mnist/.

Lecun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*, pp. 2278–2324.

Lee, Jason D et al. (2016). “Gradient descent only converges to minimizers”. In: *Conference on Learning Theory*, pp. 1246–1257.

Litjens, Geert J. S. et al. (2017). “A Survey on Deep Learning in Medical Image Analysis”. In: *CoRR* abs/1702.05747. arXiv: 1702.05747. URL: http://arxiv.org/abs/1702.05747.

Luxburg, Ulrike von (Dec. 2007). "A tutorial on spectral clustering". In: *Statistics and Computing* 17.4, pp. 395–416. ISSN: 1573-1375. DOI: 10.1007/s11222-007-9033-z. URL: https://doi.org/10.1007/s11222-007-9033-z.

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA, USA: MIT Press. ISBN: 0-262-13360-1.

McCullagh, P. and J.A. Nelder (1989). *Generalized Linear Models, Second Edition*. Chapman and Hall/CRC Monographs on Statistics and Applied Probability Series. Chapman & Hall. ISBN: 9780412317606.

Mishkin, Dmytro and Jiri Matas (2015). "All you need is a good init". In: *CoRR* abs/1511.06422. arXiv: 1511.06422. URL: http://arxiv.org/abs/1511.06422.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*. The MIT Press. ISBN: 0262018020, 9780262018029.

Nagi, Jawad et al. (2011). *Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition*.

Nair, Vinod and Geoffrey E. Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, pp. 807–814. ISBN: 978-1-60558-907-7. URL: http://dl.acm.org/citation.cfm?id=3104322.3104425.

Nesterov, Yurii (1983). "A method of solving a convex programming problem with convergence rate O(1/k2)". In: *Soviet Mathematics Doklady* 27.2, pp. 372–376.

Netzer, Yuval et al. (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In: URL: https://www-cs.stanford.edu/~twangcat/papers/nips2011_housenumbers.pdf.

Nino-Ruiz, Elias D. and Adrian Sandu (2015). "An Efficient Parallel Implementation of the Ensemble Kalman Filter Based on Shrinkage Covariance Matrix Estimation". In: *Proceedings of the 2015 IEEE 22Nd International Conference on High Performance Computing Workshops (HiPCW)*. HIPCW '15. Washington, DC, USA: IEEE Computer Society, pp. 54–54. ISBN: 978-1-4673-8717-0. DOI: 10.1109/HiPCW.2015.17. URL: http://dx.doi.org/10.1109/HiPCW.2015.17.

Niño, Elias D., Adrian Sandu, and Xinwei Deng (2016). "A Parallel Implementation of the Ensemble Kalman Filter Based on Modified Cholesky Decomposition". In: *CoRR* abs/1606.00807. arXiv: 1606.00807. URL: http://arxiv.org/abs/1606.00807.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). “On the Difficulty of Training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, pp. III-1310–III-1318. URL: `http://dl.acm.org/citation.cfm?id=3042817.3043083`.

Paszke, Adam et al. (2017). “Automatic differentiation in PyTorch”. In: *NIPS-W*.

Rasmussen, Carl Edward and Christopher KI Williams (2006). *Gaussian process for machine learning*. MIT press.

Robbins, Herbert and Sutton Monro (Sept. 1951). “A Stochastic Approximation Method”. In: *Ann. Math. Statist.* 22.3, pp. 400–407. DOI: `10.1214/aoms/1177729586`. URL: `https://doi.org/10.1214/aoms/1177729586`.

Romero, Adriana et al. (2015). “FitNets: Hints for Thin Deep Nets”. In: *In Proceedings of ICLR*.

Rosenblatt, Frank (1958). “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. In: *Psychological Review* 65.6, pp. 386–408.

Ruck, D. W. et al. (Dec. 1990). “The multilayer perceptron as an approximation to a Bayes optimal discriminant function”. In: *IEEE Transactions on Neural Networks* 1.4, pp. 296–298. ISSN: 1045-9227. DOI: `10.1109/72.80266`.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1988). “Neurocomputing: Foundations of Research”. In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6. URL: `http://dl.acm.org/citation.cfm?id=65669.104451`.

Santurkar, Shibani et al. (2018). “How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift)”. In: *CoRR* abs/1805.11604. arXiv: `1805.11604`. URL: `https://arxiv.org/abs/1805.11604`.

Schillings, Claudia and Andrew M. Stuart (2017). “Analysis of the Ensemble Kalman Filter for Inverse Problems”. In: *SIAM J. Numerical Analysis* 55, pp. 1264–1290.

Schmidhuber, Jürgen (Mar. 1992). “Learning Complex, Extended Sequences Using the Principle of History Compression”. In: *Neural Comput.* 4.2, pp. 234–242. ISSN: 0899-7667. DOI: `10.1162/neco.1992.4.2.234`. URL: `http://dx.doi.org/10.1162/neco.1992.4.2.234`.

Schmidt, Mark, Nicolas Le Roux, and Francis Bach (2017). “Minimizing finite sums with the stochastic average gradient”. In: *Mathematical Programming* 162.1-2, pp. 83–112.

Stuart, Andrew and Anthony R Humphries (1998). *Dynamical systems and numerical analysis*. Vol. 2. Cambridge University Press.

Su, Weijie, Stephen Boyd, and Emmanuel Candes (2014). “A Differential Equation for Modeling Nesterov’s Accelerated Gradient Method: Theory and Insights”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2510–2518. URL: http://papers.nips.cc/paper/5322-a-differential-equation-for-modeling-nesterovs-accelerated-gradient-method-theory-and-insights.pdf.

Such, Felipe Petroski et al. (2017). “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning”. In: *CoRR* abs/1712.06567. arXiv: 1712.06567. URL: http://arxiv.org/abs/1712.06567.

Sutskever, Ilya (2013). “Training Recurrent Neural Networks”. AAINS22066. PhD thesis. Toronto, Ont., Canada, Canada. ISBN: 978-0-499-22066-0.

Sutskever, Ilya, James Martens, et al. (2013). “On the Importance of Initialization and Momentum in Deep Learning”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, pp. III-1139–III-1147. URL: http://dl.acm.org/citation.cfm?id=3042817.3043064.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). “Sequence to Sequence Learning with Neural Networks”. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’14. Montreal, Canada: MIT Press, pp. 3104–3112. URL: http://dl.acm.org/citation.cfm?id=2969033.2969173.

Szegedy, Christian et al. (2014). “Intriguing properties of neural networks”. In: *International Conference on Learning Representations*. URL: http://arxiv.org/abs/1312.6199.

Taylor, Gavin et al. (2016). “Training Neural Networks Without Gradients: A Scalable ADMM Approach”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. New York, NY, USA: JMLR.org, pp. 2722–2731. URL: http://dl.acm.org/citation.cfm?id=3045390.3045677.

Tsai, Y. H., O. C. Hamsici, and M. H. Yang (June 2015). “Adaptive region pooling for object detection”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 731–739. DOI: 10.1109/CVPR.2015.7298673.

Ulyanov, Dmitry, Andrea Vedaldi, and Victor S. Lempitsky (2016). “Instance Normalization: The Missing Ingredient for Fast Stylization”. In: *CoRR* abs/1607.08022. arXiv: 1607.08022. URL: http://arxiv.org/abs/1607.08022.

Vapnik, Vladimir N. (1995). *The Nature of Statistical Learning Theory*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0-387-94559-8.

Vogel, Curtis R. (2002). *Computational Methods for Inverse Problems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics. ISBN: 0898715075.

Wang, J. et al. (June 2016). "CNN-RNN: A Unified Framework for Multi-label Image Classification". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2285–2294. DOI: `10.1109/CVPR.2016.251`.

Wilson, Ashia C et al. (2017). "The Marginal Value of Adaptive Gradient Methods in Machine Learning". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 4148–4158. URL: `http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf`.

Xiao, Han, Kashif Rasul, and Roland Vollgraf (2017). "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *CoRR* abs/1708.07747. arXiv: `1708.07747`. URL: `http://arxiv.org/abs/1708.07747`.

Zelnik-manor, Lihi and Pietro Perona (2005). "Self-Tuning Spectral Clustering". In: *Advances in Neural Information Processing Systems 17*. Ed. by L. K. Saul, Y. Weiss, and L. Bottou. MIT Press, pp. 1601–1608. URL: `http://papers.nips.cc/paper/2619-self-tuning-spectral-clustering.pdf`.

Zhang, Sixin, Anna Choromanska, and Yann LeCun (2015). "Deep Learning with Elastic Averaging SGD". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'15. Montreal, Canada: MIT Press, pp. 685–693. URL: `http://dl.acm.org/citation.cfm?id=2969239.2969316`.

Chapter 3

CONTINUOUS TIME ANALYSIS OF MOMENTUM METHODS

3.1 Introduction

Background and Literature Review

At the core of many machine learning tasks is solution of the optimization problem

$$\arg\min_{u \in \mathbb{R}^d} \; \Phi(u) \tag{3.1}$$

where $\Phi : \mathbb{R}^d \to \mathbb{R}$ is an objective (or loss) function that is, in general, non-convex and differentiable. Finding global minima of such objective functions is an important and challenging task with a long history, one in which the use of stochasticity has played a prominent role for many decades, with papers in the early development of machine learning (S. Geman and D. Geman, 1987; Styblinski and Tang, 1990), together with concomitant theoretical analyses for both discrete (Bertsimas, Tsitsiklis, et al., 1993) and continuous problems (Harold J Kushner, 1987; Harold Joseph Kushner and Clark, 2012). Recent successes in the training of deep neural networks have built on this older work, leveraging the enormous computer power now available, together with empirical experience about good design choices for the architecture of the networks; reviews may be found in (Goodfellow, Bengio, and Courville, 2016; LeCun, Bengio, and Geoffrey E. Hinton, 2015). Gradient descent plays a prominent conceptual role in many algorithms, following from the observation that the equation

$$\frac{du}{dt} = -\nabla\Phi(u) \tag{3.2}$$

will decrease Φ along trajectories. The most widely adopted methods use stochastic gradient decent (SGD), a concept introduced in (Robbins and Monro, 1951); the basic idea is to use gradient decent steps based on a noisy approximation to the gradient of Φ. Building on deep work in the convex optimization literature, momentum-based modifications to stochastic gradient decent have also become widely used in optimization. Most notable amongst these momentum-based methods are the Heavy Ball Method (HB), due to (B. Polyak, 1964), and Nesterov's method of accelerated gradients (NAG) (Nesterov, 1983). To the best of our knowledge, the first application of HB to neural network training appears in (Rumelhart, G. E. Hinton, and Williams, 1986). More recent work, such as (Sutskever et al., 2013), has even argued for the indispensability of such momentum based methods for the field of deep learning.

From these two basic variants on gradient decent, there have come a plethora of adaptive methods, incorporating momentum-like ideas, such as Adam (Kingma and Ba, 2014), Adagrad (Duchi, Hazan, and Singer, 2011), and RMSProp (Tieleman and G. Hinton, 2012). There is no consensus on which method performs best and results vary based on application. The recent work of (Ashia C Wilson et al., 2017) argues that the rudimentary, non-adaptive schemes SGD, HB, and NAG result in solutions with the greatest generalization performance for supervised learning applications with deep neural network models.

There is a natural physical analogy for momentum methods, namely that they relate to a damped second order Hamiltonian dynamic with potential Φ:

$$m\frac{d^2u}{dt^2} + \gamma(t)\frac{du}{dt} + \nabla\Phi(u) = 0. \tag{3.3}$$

This perspective goes back to Polyak's original work (B. Polyak, 1964; B. T. Polyak, 1987) and was further expanded on in (Qian, 1999), although no proof was given. For NAG, the work of (Su, Boyd, and Candes, 2014) proves that the method approximates a damped Hamiltonian system of precisely this form, with a time-dependent damping coefficient. The analysis in (Su, Boyd, and Candes, 2014) holds when the momentum factor is chosen according to the rule

$$\lambda = \lambda_n = \frac{n}{n+3}, \tag{3.4}$$

where n is the iteration count; this choice was proposed in the original work of (Nesterov, 1983) and results in a choice of λ which is asymptotic to 1. In the setting where Φ is μ-strongly convex, it is proposed in (Nesterov, 2014) that the momentum factor is fixed and chosen close to 1; specifically it is proposed that

$$\lambda = \frac{1-\sqrt{\mu h}}{1+\sqrt{\mu h}} \tag{3.5}$$

where $h > 0$ is the time-step (learning rate). In (Ashia C. Wilson, Recht, and Michael I. Jordan, 2016), a limiting equation for both HB and NAG of the form

$$\ddot{u} + 2\sqrt{\mu}\dot{u} + \nabla\Phi(u) = 0$$

is derived under the assumption that λ is fixed with respect to iteration number n, and dependent on the time-step h as specified in (3.5); convergence is obtained to order $\mathcal{O}(h^{1/2})$. Using insight from this limiting equation it is possible to choose the optimal value of μ to maximize the convergence rate in the neighborhood of a locally strongly convex objective function. Further related work is developed in (Shi et al., 2018) where separate limiting equations for HB and NAG are derived both in the cases of λ given by (3.4) and (3.5), obtaining convergence to order $\mathcal{O}(h^{3/2})$. Much work has also gone into analyzing these methods in the discrete setting, without appeal to the continuous time limits, see (Hu and Lessard, 2017; Lessard, Recht, and Packard, 2016), as well as in the stochastic setting, establishing how the effect on the generalization error, for example, (Gadat, Panloup, Saadane, et al., 2018; Loizou and Richtárik, 2017; Yang, Lin, and Z. Li, 2016). In this paper, however, our focus is on the use of continuous time limits as a methodology to explain optimization algorithms.

In many machine learning applications, especially for deep learning, NAG and HB are often used with a constant momentum factor λ that is chosen independently of the iteration count n (contrary to (3.4)) and independently of the learning rate h (contrary to (3.5)). In fact, popular books on the subject such as (Goodfellow, Bengio, and Courville, 2016) introduce the methods in this way, and popular articles, such as (He et al., 2016) to name one of many, simply state the value of the constant momentum factor used in their experiments. Widely used deep learning libraries such as Tensorflow (Martín Abadi et al., 2015) and PyTorch (Paszke et al., 2017) implement the methods with a fixed choice of momentum factor. Momentum based methods used in this way, with fixed momentum, have not been carefully analyzed. We will undertake such an analysis, using ideas from numerical analysis, and in particular the concept of *modified equations* (Griffiths and Sanz-Serna, 1986; Chartier, Hairer, and Vilmart, 2007) and from the theory of *attractive invariant manifolds* (Hirsch, Pugh, and Shub, 2006; Wiggins, 2013); both ideas are explained in the text (A. Stuart and Humphries, 1998). It is noteworthy that the *high resolution ODE approximation* described in (Shi et al., 2018) may be viewed as a rediscovery of the method of modified equations. We emphasize the fact that our work is not at odds with any previous analyses of these methods; rather, we consider a setting which is widely adopted in deep learning applications and has not been subjected to continuous time analysis to date.

Remark 5. *Since publication of the article (Kovachki and Andrew M. Stuart, 2021), we became aware of related, and earlier, work by (Farazmand, 2018). Farazmand starts from the Bregman Lagrangian introduced in (Wibisono, Ashia C Wilson, and*

Michael I Jordan, 2016) and uses ideas from geometric singular perturbation theory to derive an invariant manifold. The work leads to a more general description of the invariant manifold than the one given by our equation (3.20)*. Farazmand's work was published in (Farazmand, 2020).*

Our Contribution

We study momentum-based optimization algorithms for the minimization task (3.1), with learning rate independent momentum, fixed at every iteration step, focusing on deterministic methods for clarity of exposition. Our approach is to derive continuous time approximations of the discrete algorithms; these continuous time approximations provide insights into the mechanisms at play within the discrete algorithms. We prove three such approximations. The first shows that the asymptotic limit of the momentum methods, as learning rate approaches zero, is simply a rescaled gradient flow (3.2). The second two approximations include small perturbations to the rescaled gradient flow, on the order of the learning rate, and give insight into the behavior of momentum methods when implemented with momentum and fixed learning rate. Through these approximation theorems, and accompanying numerical experiments, we make the following contributions to the understanding of momentum methods as often implemented within machine learning:

- We show that momentum-based methods with a fixed momentum factor, satisfy, in the continuous-time limit obtained by sending the learning rate to zero, a rescaled version of the gradient flow equation (3.2).
- We show that such methods also approximate a damped Hamiltonian system of the form (3.3), with small mass m (on the order of the learning rate) and constant damping $\gamma(t) = \gamma$; this approximation has the same order of accuracy as the approximation of the rescaled equation (3.2) but provides a better qualitative understanding of the fixed learning rate momentum algorithm in its transient phase.
- We also show that, for the approximate Hamiltonian system, the dynamics admit an exponentially attractive invariant manifold, locally representable as a graph mapping co-ordinates to their velocities. The map generating this graph describes a gradient flow in a potential which is a small (on the order of the learning rate) perturbation of Φ – see (3.21); the correction to the potential is

convexifying, does not change the global minimum, and provides insight into the fixed learning rate momentum algorithm beyond its initial transient phase.

- We provide numerical experiments which illustrate the foregoing considerations, for simple linear test problems, and for the MNIST digit classification problem; in the latter case we consider SGD and thereby demonstrate that the conclusions of our theory have relevance for understanding the stochastic setting as well.

Taken together our results are interesting because they demonstrate that the popular belief that (fixed) momentum methods resemble the dynamics induced by (3.3) is misleading. Whilst it is true, the mass in the approximating equation is small and as a consequence understanding the dynamics as gradient flows (3.2), with modified potential, is more instructive. In fact, in the first application of HB to neural networks described in (Rumelhart, G. E. Hinton, and Williams, 1986), the authors state that "[their] experience has been that [one] get[s] the same solutions by setting [the momentum factor to zero] and reducing the size of [the learning rate]." However our theorems should not be understood to imply that there is no practical difference between momentum methods (with fixed learning rate) and SGD. There is indeed a practical difference as has been demonstrated in numerous papers throughout the machine learning literature, and our experiments in Section 3.5 further confirm this. We show that while these methods have the same transient dynamics, they are approximated differently. Our results demonstrate that, although momentum methods behave like a gradient descent algorithm, asymptotically, this algorithm has a modified potential. Furthermore, although this modified potential (3.20) is on the order of the learning rate, the fact that the learning rate is often chosen as large as possible, constrained by numerical stability, means that the correction to the potential may be significant. Our results may be interpreted as indicating that the practical success of momentum methods stems from the fact that they provide a more stable discretization to (3.2) than the forward Euler method employed in SGD. The damped Hamiltonian dynamic (3.11), as well the modified potential, give insight into how this manifests. Our work gives further theoretical justification for the exploration of the use of different numerical integrators for the purposes of optimization such as those performed in (Scieur et al., 2017; Betancourt, Michael I. Jordan, and Ashia C. Wilson, 2018; Zhang et al., 2018).

While our analysis is confined to the non-stochastic case to simplify the exposition, the results will, with some care, extend to the stochastic setting using ideas from averaging and homogenization (Pavliotis and A. Stuart, 2008) as well as continuum analyses of SGD as in (Q. Li, Tai, and E, 2017; Feng, L. Li, and Liu, 2018); indeed, in the stochastic setting, sharp uniform in time error estimates are to be expected for empirical averages (Mattingly, Andrew M Stuart, and Tretyakov, 2010; Dieuleveut, Durmus, and Bach, 2017). To demonstrate that our analysis is indeed relevant in the stochastic setting, we train a deep autoencoder with mini-batching (stochastic) and verify that our convergence results still hold. The details of this experiment are given in section 3.5. Furthermore we also confine our analysis to fixed learning rate, and impose global bounds on the relevant derivatives of Φ; this further simplifies the exposition of the key ideas, but is not essential to them; with considerably more analysis the ideas exposed in this paper will transfer to adaptive time-stepping methods and much less restrictive classes of Φ.

The paper is organized as follows. Section 3.2 introduces the optimization procedures and states the convergence result to a rescaled gradient flow. In section 3.3 we derive the modified, second-order equation and state convergence of the schemes to this equation. Section 3.4 asserts the existence of an attractive invariant manifold, demonstrating that it results in a gradient flow with respect to a small perturbation of Φ. In section 3.5, we train a deep autoencoder, showing that our results hold in a stochastic setting with Assumption 6 violated. We conclude in section 3.6. All proofs of theorems are given in the appendices so that the ideas of the theorems can be presented clearly within the main body of the text.

Notation

We use $|\cdot|$ to denote the Euclidean norm on $\mathbb{R}^d$. We define $f : \mathbb{R}^d \to \mathbb{R}^d$ by $f(u) := -\nabla\Phi(u)$ for any $u \in \mathbb{R}^d$. Given parameter $\lambda \in [0, 1)$ we define $\bar{\lambda} := (1-\lambda)^{-1}$.

For two Banach spaces A, B, and A_0 a subset in A, we denote by $C^k(A_0; B)$ the set of k-times continuously differentiable functions with domain A_0 and range B. For a function $u \in C^k(A_0; B)$, we let $D^j u$ denote its j-th (total) Fréchet derivative for $j = 1, \dots, k$. For a function $u \in C^k([0,\infty), \mathbb{R}^d)$, we denote its derivatives by $\frac{du}{dt}, \frac{d^2u}{dt^2}$, etc. or equivalently by $\dot{u}, \ddot{u}$, etc.

To simplify our proofs, we make the following assumption about the objective function.

Assumption 6. *Suppose $\Phi \in C^3(\mathbb{R}^d; \mathbb{R})$ with uniformly bounded derivatives. Namely,* *there exist constants* $B_0, B_1, B_2 > 0$ *such that*

$$\|D^{j-1}f\| = \|D^j\Phi\| \leq B_{j-1}$$

for $j = 1, 2, 3$ *where* $\|\cdot\|$ *denotes any appropriate operator norm.*

We again stress that this assumption is not key to developing the ideas in this work, but is rather a simplification used to make our results global. Without Assumption 6, and no further assumption on Φ such as convexity, one could only hope to give local results, i.e. in the neighborhood of a critical point of Φ. Such analysis could indeed be carried out (see for example (Carr, 2012)), but we choose not to do so here for the sake of clarity of exposition. In section 3.5, we give a practical example where this assumption is violated and yet the behavior is as predicted by our theory.

Finally we observe that the nomenclature "learning rate" is now prevalent in machine learning, and so we use it in this paper; it refers to the object commonly referred to as "time-step" in the field of numerical analysis.

3.2 Momentum Methods and Convergence to Gradient Flow

In subsection 3.2 we state Theorem 7 concerning the convergence of a class of momentum methods to a rescaled gradient flow. Subsection 3.2 demonstrates that the HB and NAG methods are special cases of our general class of momentum methods, and gives intuition for proof of Theorem 7; the proof itself is given in Appendix A. Subsection 3.2 contains a numerical illustration of Theorem 7.

Main Result

The standard Euler discretization of (3.2) gives the discrete time optimization scheme

$$\mathsf{u}_{n+1} = \mathsf{u}_n + hf(\mathsf{u}_n), \quad n = 0, 1, 2, \ldots . \tag{3.6}$$

Implementation of this scheme requires an initial guess $\mathsf{u}_0 \in \mathbb{R}^d$. For simplicity we consider a fixed learning rate $h > 0$. Equation (3.2) has a unique solution $u \in C^3([0,\infty); \mathbb{R}^d)$ under Assumption 6 and for $u_n = u(nh)$

$$\sup_{0 \leq nh \leq T} |\mathsf{u}_n - u_n| \leq C(T)h;$$

see (A. Stuart and Humphries, 1998), for example.

In this section we consider a general class of momentum methods for the minimization task (3.1) which can be written in the form, for some $a \geq 0$ and $\lambda \in (0,1)$,

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + \lambda(\mathsf{u}_n - \mathsf{u}_{n-1}) + hf(\mathsf{u}_n + a(\mathsf{u}_n - \mathsf{u}_{n-1})), \quad n = 0,1,2,\dots\,, \\ \mathsf{u}_1 &= \mathsf{u}_0 + hf(\mathsf{u}_0)\,. \end{aligned} \tag{3.7}$$

Again, implementation of this scheme requires an an initial guess $\mathsf{u}_0 \in \mathbb{R}^d$. The parameter choice $a = 0$ gives HB and $a = \lambda$ gives NAG. In Appendix A we prove the following:

Theorem 7. *Suppose Assumption 6 holds and let $u \in C^3([0,\infty);\mathbb{R}^d)$ be the solution to*

$$\begin{aligned} \frac{du}{dt} &= -(1-\lambda)^{-1}\nabla\Phi(u) \\ u(0) &= \mathsf{u}_0 \end{aligned} \tag{3.8}$$

with $\lambda \in (0,1)$. For $n = 0,1,2,\dots$ let u_n be the sequence given by (3.7) *and define $u_n := u(nh)$. Then for any $T \geq 0$, there is a constant $C = C(T) > 0$ such that*

$$\sup_{0 \leq nh \leq T} |u_n - \mathsf{u}_n| \leq Ch.$$

Note that (3.8) is simply a sped-up version of (3.2): if v solves (3.2) and w solves (3.8) then $v(t) = w((1-\lambda)t)$ for any $t \in [0,\infty)$. This demonstrates that introduction of momentum in the form used within both HB and NAG results in numerical methods that do not differ substantially from gradient descent.

Link to HB and NAG

The HB method is usually written as a two-step scheme taking the form ((Sutskever et al., 2013))

$$\begin{aligned} \mathsf{v}_{n+1} &= \lambda\mathsf{v}_n + hf(\mathsf{u}_n) \\ \mathsf{u}_{n+1} &= \mathsf{u}_n + \mathsf{v}_{n+1} \end{aligned}$$

with $\mathsf{v}_0 = 0$, $\lambda \in (0,1)$ the momentum factor, and $h > 0$ the learning rate. We can re-write this update as

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + \lambda\mathsf{v}_n + hf(\mathsf{u}_n) \\ &= \mathsf{u}_n + \lambda(\mathsf{u}_n - \mathsf{u}_{n-1}) + hf(\mathsf{u}_n) \end{aligned}$$

hence the method reads

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + \lambda(\mathsf{u}_n - \mathsf{u}_{n-1}) + hf(\mathsf{u}_n) \\ \mathsf{u}_1 &= \mathsf{u}_0 + hf(\mathsf{u}_0). \end{aligned} \tag{3.9}$$

Similarly NAG is usually written as ((Sutskever et al., 2013))

$$\begin{aligned} \mathsf{v}_{n+1} &= \lambda \mathsf{v}_n + hf(\mathsf{u}_n + \lambda \mathsf{v}_n) \\ \mathsf{u}_{n+1} &= \mathsf{u}_n + \mathsf{v}_{n+1} \end{aligned}$$

with $\mathsf{v}_0 = 0$. Define $\mathsf{w}_n \coloneqq \mathsf{u}_n + \lambda \mathsf{v}_n$, then

$$\begin{aligned} \mathsf{w}_{n+1} &= \mathsf{u}_{n+1} + \lambda \mathsf{v}_{n+1} \\ &= \mathsf{u}_{n+1} + \lambda(\mathsf{u}_{n+1} - \mathsf{u}_n) \end{aligned}$$

and

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + \lambda \mathsf{v}_n + hf(\mathsf{u}_n + \lambda \mathsf{v}_n) \\ &= \mathsf{u}_n + (\mathsf{w}_n - \mathsf{u}_n) + hf(\mathsf{w}_n) \\ &= \mathsf{w}_n + hf(\mathsf{w}_n). \end{aligned}$$

Hence the method may be written as

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + \lambda(\mathsf{u}_n - \mathsf{u}_{n-1}) + hf(\mathsf{u}_n + \lambda(\mathsf{u}_n - \mathsf{u}_{n-1})) \\ \mathsf{u}_1 &= \mathsf{u}_0 + hf(\mathsf{u}_0). \end{aligned} \tag{3.10}$$

It is clear that (3.9) and (3.10) are special cases of (3.7) with $a = 0$ giving HB and $a = \lambda$ giving NAG. To intuitively understand Theorem 7, re-write (3.8) as

$$\frac{du}{dt} - \lambda \frac{du}{dt} = f(u).$$

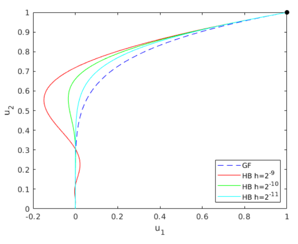
If we discretize the du/dt term using forward differences and the $-\lambda du/dt$ term using backward differences, we obtain

$$\frac{u(t+h) - u(t)}{h} - \lambda \frac{u(t) - u(t-h)}{h} \approx f(u(t)) \approx f\left(u(t) + ha\frac{u(t) - u(t-h)}{h}\right)$$

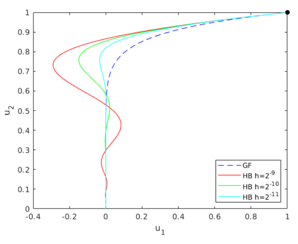
with the second approximate equality coming from the Taylor expansion of f. This can be rearranged as

$$u(t+h) \approx u(t) + \lambda(u(t) - u(t-h)) + hf(u(t) + a(u(t) - u(t-h)))$$

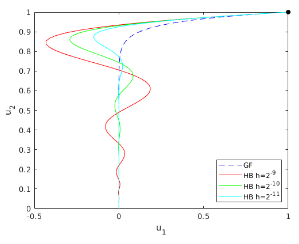
which has the form of (3.7) with the identification $\mathsf{u}_n \approx u(nh)$.



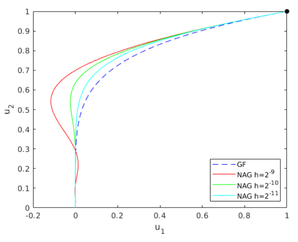
(a) HB: $\kappa = 5$



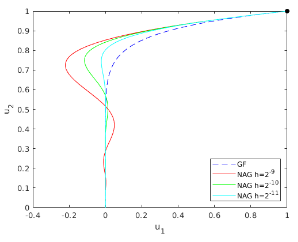
(b) HB: $\kappa = 10$



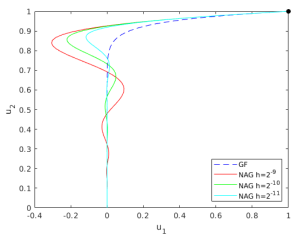
(c) HB: $\kappa = 20$



(d) NAG: $\kappa = 5$

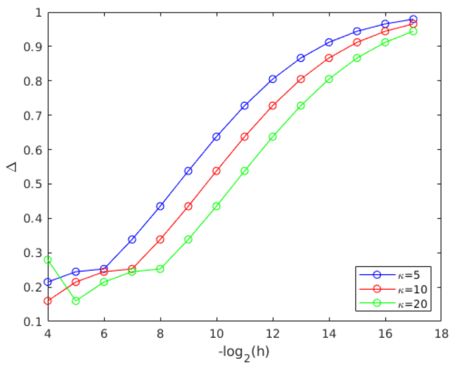


(e) NAG: $\kappa = 10$

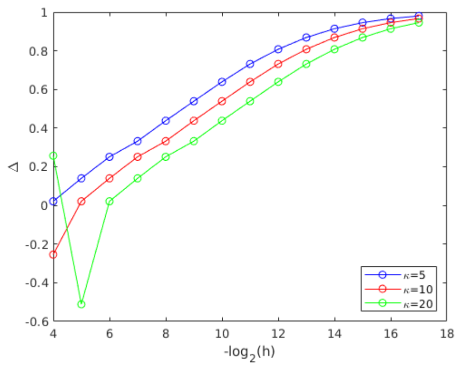


(f) NAG: $\kappa = 20$

Figure 31: Comparison of trajectories for HB and NAG with the gradient flow (3.8) on the two-dimensional problem $\Phi(u) = \frac{1}{2}\langle u, Qu\rangle$ with $\lambda = 0.9$ fixed. We vary the condition number of Q as well as the learning rate h.



(a) HB



(b) NAG

Figure 32: The numerical rate of convergence, as a function of the learning rate h, of HB and NAG to the gradient flow (3.8) for the problem described in Figure 31.

Numerical Illustration

Figure 31 compares trajectories of the momentum numerical method (3.7) with the rescaled gradient flow (3.8), for the two-dimensional problem $\Phi(u) = \frac{1}{2}\langle u, Qu\rangle$. We pick Q to be positive-definite so that the minimum is achieved at the point $(0, 0)^T$ and make it diagonal so that we can easily control its condition number. In particular, the condition number of Q is given as

$$\kappa = \frac{\max\{Q_{11}, Q_{22}\}}{\min\{Q_{11}, Q_{22}\}}.$$

We see that, as the condition number is increased, both HB and NAG exhibit more pronounced transient oscillations and are thus further away from the trajectory of (3.8), however, as the learning rate h is decreased, the oscillations dampen and the trajectories match more and more closely. This observation from Figure 31 is quantified in Figure 32 where we estimate the rate of convergence as a function of h, which is defined as

$$\Delta = \log_2 \frac{\|\mathsf{u}^{(h)} - u\|_\infty}{\|\mathsf{u}^{(h/2)} - u\|_\infty}$$

where $\mathsf{u}^{(\alpha)}$ is the numerical solution using time-step α. The figure shows that the rate of convergence is indeed close to 1, as predicted by our theory. In summary the behavior of the momentum methods is precisely that of a rescaled gradient flow, but with initial transient oscillations which capture momentum effects, but disappear as the learning rate is decreased. We model these oscillations in the next section via use of a modified equation.

3.3 Modified Equations

The previous section demonstrates how the momentum methods approximate a time rescaled version of the gradient flow (3.2). In this section we show how the same methods may also be viewed as approximations of the damped Hamiltonian system (3.3), with mass m on the order of the learning rate, using the method of modified equations. In subsection 3.3 we state and discuss the main result of the section, Theorem 8. Subsection 3.3 gives intuition for proof of Theorem 8; the proof itself is given in Appendix B. And the section also contains comments on generalizing the idea of modified equations. In subsection 3.3 we describe a numerical illustration of Theorem 8.

Main Result

The main result of this section quantifies the sense in which momentum methods do, in fact, approximate a damped Hamiltonian system; it is proved in Appendix B.

Theorem 8. *Fix $\lambda \in (0,1)$ and assume that $a \geq 0$ is chosen so that $\alpha := \frac{1}{2}(1 + \lambda - 2a(1-\lambda))$ is strictly positive. Suppose Assumption 6 holds and let $u \in C^4([0,\infty);\mathbb{R}^d)$ be the solution to*

$$\begin{aligned} & h\alpha \frac{d^2u}{dt^2} + (1-\lambda)\frac{du}{dt} = -\nabla\Phi(u) \\ & u(0) = \mathsf{u}_0, \quad \frac{du}{dt}(0) = \mathsf{u}_0'. \end{aligned} \tag{3.11}$$

Suppose further that $h \leq (1-\lambda)^2/2\alpha B_1$. For $n = 0, 1, 2, \dots$ let u_n be the sequence given by (3.7) *and define $u_n := u(nh)$. Then for any $T \geq 0$, there is a constant $C = C(T) > 0$ such that*

$$\sup_{0 \leq nh \leq T} |u_n - \mathsf{u}_n| \leq Ch.$$

Theorem 7 demonstrates the same order of convergence, namely $\mathcal{O}(h)$, to the rescaled gradient flow equation (3.8), obtained from (3.11) simply by setting $h = 0$. In the standard method of modified equations the limit system (here (3.8)) is perturbed by small terms (in terms of the assumed small learning rate) and an increased rate of convergence is obtained to the modified equation (here (3.11)). In our setting however, because the small modification is to a higher derivative (here second) than appears in the limit equation (here first order), an increased rate of convergence is not obtained. This is due to the nature of the modified equation, whose solution has derivatives that are inversely proportional to powers of h; this fact is quantified in Lemma 13 from Appendix B. It is precisely because the modified equation does not lead to a higher rate of convergence that the initial parameter u_0' is arbitrary; the same rate of convergence is obtained no matter what value it takes.

It is natural to ask, therefore, what is learned from the convergence result in Theorem 8. The answer is that, although the modified equation (3.11) is approximated at the same order as the limit equation (3.8), it actually contains considerably more qualitative information about the dynamics of the system, particularly in the early transient phase of the algorithm; this will be illustrated in subsection 3.3. Indeed we will make a specific choice of u_0' in our numerical experiments, namely

$$\frac{du}{dt}(0) = \frac{1-2\alpha}{2\alpha - \lambda + 1} f(\mathsf{u}_0), \tag{3.12}$$

to better match the transient dynamics.

Intuition and Wider Context

Idea Behind The Modified Equations

In this subsection, we show that the scheme (3.7) exhibits momentum, in the sense of approximating a momentum equation, but the size of the momentum term is on the order of the step size h. To see this intuitively, we add and subtract $\mathsf{u}_n - \mathsf{u}_{n-1}$ to the right hand size of (3.7) then we can rearrange it to obtain

$$h\frac{\mathsf{u}_{n+1} - 2\mathsf{u}_n + \mathsf{u}_{n-1}}{h^2} + (1-\lambda)\frac{\mathsf{u}_n - \mathsf{u}_{n-1}}{h} = f(\mathsf{u}_n + a(\mathsf{u}_n - \mathsf{u}_{n-1})).$$

This can be seen as a second order central difference and first order backward difference discretization of the momentum equation

$$h\frac{d^2u}{dt^2} + (1-\lambda)\frac{du}{dt} = f(u),$$

noting that the second derivative term has size of order h.

Higher Order Modified Equations For HB

We will now show that, for HB, we may derive higher order modified equations that are consistent with (3.9). Taking the limit of these equations yields an operator that agrees with with our intuition for discretizing (3.8). To this end, suppose $\Phi \in C_b^\infty(\mathbb{R}^d, \mathbb{R})$ and consider the ODE(s),

$$\sum_{k=1}^{p} \frac{h^{k-1}(1+(-1)^k\lambda)}{k!}\frac{d^k u}{dt^k} = f(u) \tag{3.13}$$

noting that $p=1$ gives (3.8) and $p=2$ gives (3.11). Let $u \in C^\infty([0,\infty), \mathbb{R}^d)$ be the solution to (3.13) and define $u_n := u(nh)$, $u_n^{(k)} := \frac{d^k u}{dt^k}(nh)$ for $n = 0, 1, 2, \dots$ and $k = 1, 2, \dots, p$. Taylor expanding yields

$$u_{n\pm1} = u_n + \sum_{k=1}^{p} \frac{(\pm1)^k h^k}{k!} u_n^{(k)} + h^{p+1} I_n^{\pm}$$

where

$$I_n^{\pm} = \frac{(\pm1)^{p+1}}{p!}\int_0^1 (1-s)^p \frac{d^{p+1}u}{dt^{p+1}}((n\pm s)h)ds.$$

Then

$$\begin{aligned} u_{n+1} - u_n - \lambda(u_n - u_{n-1}) &= \sum_{k=1}^{p}\frac{h^k}{k!}u_n^{(k)} + \lambda\sum_{k=1}^{p}\frac{(-1)^k h^k}{k!}u_n^{(k)} + h^{p+1}(I_n^+ - \lambda I_n^-) \\ &= h\sum_{k=1}^{p}\frac{h^{k-1}(1+(-1)^k\lambda)}{k!}u_n^{(k)} + h^{p+1}(I_n^+ - \lambda I_n^-) \\ &= hf(u_n) + h^{p+1}(I_n^+ - \lambda I_n^-), \end{aligned}$$

showing consistency to order $p+1$. As is the case with (3.11) however, the $I_n^{\pm}$ terms will be inversely proportional to powers of h hence global accuracy will not improve.

We now study the differential operator on the l.h.s. of (3.13) as $p \to \infty$. Define the sequence of differential operators $T_p : C^\infty([0,\infty), \mathbb{R}^d) \to C^\infty([0,\infty), \mathbb{R}^d)$ by

$$T_p u = \sum_{k=1}^{p}\frac{h^{k-1}(1+(-1)^k\lambda)}{k!}\frac{d^k u}{dt^k}, \quad \forall u \in C^\infty([0,\infty), \mathbb{R}^d)$$

and suppose $u, T_p u \in L^1([0,\infty); \mathbb{R}^d)$. Taking the Fourier transform yields

$$\mathcal{F}(T_p u)(\omega) = \sum_{k=1}^{p} \frac{h^{k-1}(1+(-1)^k\lambda)(i\omega)^k}{k!}\mathcal{F}(u)(\omega)$$

where $i = \sqrt{-1}$ denotes the imaginary unit. Suppose there is a limiting operator $T_p \to T$ as $p \to \infty$ then taking the limit yields

$$\mathcal{F}(Tu)(\omega) = \frac{1}{h}(e^{ih\omega} + \lambda e^{-ih\omega} - \lambda - 1)\mathcal{F}(u)(\omega).$$

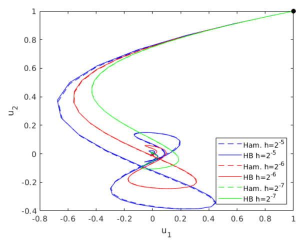
Taking the inverse transform and using the convolution theorem, we obtain

$$\begin{aligned}
(Tu)(t) &= \frac{1}{h}\mathcal{F}^{-1}(e^{ih\omega} + \lambda e^{-ih\omega} - \lambda - 1)(t) * u(t) \\
&= \frac{1}{h}\left(-(1+\lambda)\delta(t) + \lambda\delta(t+h) + \delta(t-h)\right) * u(t) \\
&= \frac{1}{h}\int_{-\infty}^{\infty} \left(-(1+\lambda)\delta(t-\tau) + \lambda\delta(t-\tau+h) + \delta(t-\tau-h)\right) u(\tau)\, d\tau \\
&= \frac{1}{h}\left(-(1+\lambda)u(t) + \lambda u(t-h) + u(t+h)\right) \\
&= \frac{u(t+h)-u(t)}{h} - \lambda\left(\frac{u(t)-u(t-h)}{h}\right)
\end{aligned}$$

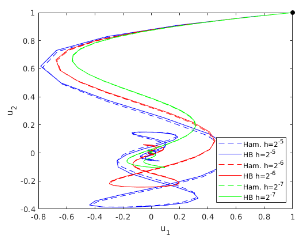
where $\delta(\cdot)$ denotes the Dirac-delta distribution and we abuse notation by writing its action as an integral. The above calculation does not prove convergence of T_p to T, but simply confirms our intuition that (3.9) is a forward and backward discretization of (3.8).

Numerical Illustration

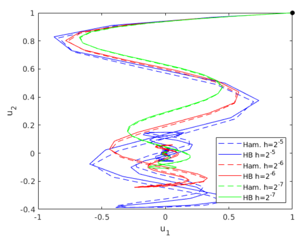
Figure 33 shows trajectories of (3.7) and (3.11) for different values of a and h on the two-dimensional problem $\Phi(u) = \frac{1}{2}\langle u, Qu\rangle$, varying the condition number of Q. We make the specific choice of u_0' implied by the initial condition (3.12). Figure 34 shows the numerical order of convergence as a function of h, as defined in Section 3.2, which is near 1, matching our theory. We note that the oscillations in HB are captured well by (3.11), except for a slight shift when h and κ are large. This is due to our choice of initial condition which cancels the maximum number of terms in the Taylor expansion initially, but the overall rate of convergence remains $\mathcal{O}(h)$ due to Lemma 13. Other choices of u_0' also result in $\mathcal{O}(h)$ convergence and can be picked on a case-by-case basis to obtain consistency with different qualitative phenomena of interest in the dynamics. Note also that $\alpha|_{a=\lambda} < \alpha|_{a=0}$. As a result the transient oscillations in (3.11) are more quickly damped in the NAG case than in the HB case;



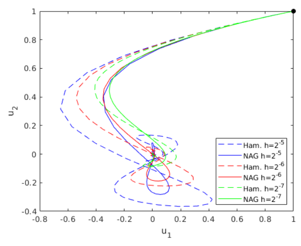
(a) HB: $\kappa = 5$



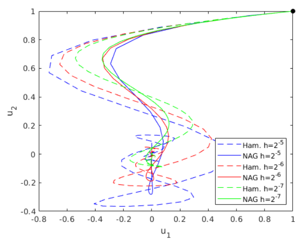
(b) HB: $\kappa = 10$



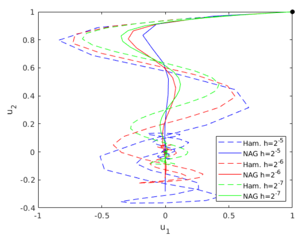
(c) HB: $\kappa = 20$



(d) NAG: $\kappa = 5$

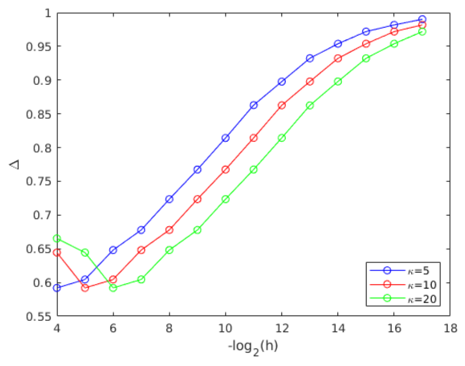


(e) NAG: $\kappa = 10$

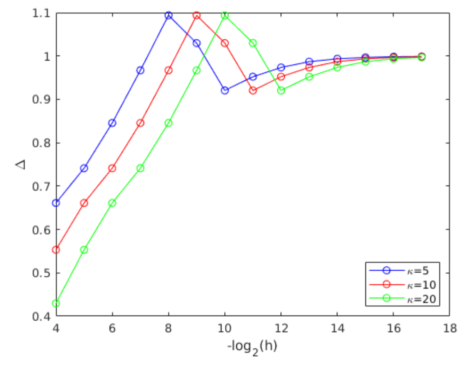


(f) NAG: $\kappa = 20$

Figure 33: Comparison of trajectories for HB and NAG with the Hamiltonian dynamic (3.11) on the two-dimensional problem $\Phi(u) = \frac{1}{2}\langle u, Qu\rangle$ with $\lambda = 0.9$ fixed. We vary the condition number of Q as well as the learning rate h.



(a) HB



(b) NAG

Figure 34: The numerical rate of convergence, as a function of the learning rate h, of HB and NAG to the momentum equation (3.11) for the problem described in Figure 33.

this is consistent with the numerical results. However panels (d)-(f) in Figure 31 show that (3.11) is not able to adequately capture the oscillations of NAG when h is relatively large. We leave for future work the task of finding equations that are able to appropriately capture the oscillations of NAG in the large h regime.

3.4 Invariant Manifold

The key lessons of the previous two sections are that the momentum methods approximate a rescaled gradient flow of the form (3.2) and a damped Hamiltonian system of the form (3.3), with small mass m which scales with the learning rate, and constant damping γ. Both approximations hold with the same order of accuracy, in terms of the learning rate, and numerics demonstrate that the Hamiltonian system is particularly useful in providing intuition for the transient regime of the algorithm. In this section we link the two theorems from the two preceding sections by showing that the Hamiltonian dynamics with small mass from section 3.3 has an exponentially attractive invariant manifold on which the dynamics is, to leading order, a gradient flow. That gradient flow is a small, in terms of the learning rate, perturbation of the time-rescaled gradient flow from section 3.2.

Main Result

Define

$$\mathsf{v}_n := (\mathsf{u}_n - \mathsf{u}_{n-1})/h \tag{3.14}$$

noting that then (3.7) becomes

$$\mathsf{u}_{n+1} = \mathsf{u}_n + h\lambda\mathsf{v}_n + hf(\mathsf{u}_n + ha\mathsf{v}_n)$$

and

$$\mathsf{v}_{n+1} = \frac{\mathsf{u}_{n+1} - \mathsf{u}_n}{h} = \lambda\mathsf{v}_n + f(\mathsf{u}_n + ha\mathsf{v}_n).$$

Hence we can re-write (3.7) as

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + h\lambda\mathsf{v}_n + hf(\mathsf{u}_n + ha\mathsf{v}_n) \\ \mathsf{v}_{n+1} &= \lambda\mathsf{v}_n + f(\mathsf{u}_n + ha\mathsf{v}_n). \end{aligned} \tag{3.15}$$

Note that if $h = 0$ then (3.15) shows that $\mathsf{u}_n = \mathsf{u}_0$ is constant in n, and that v_n converges to $(1-\lambda)^{-1}f(\mathsf{u}_0)$. This suggests that, for h small, there is an invariant manifold which is a small perturbation of the relation $\mathsf{v}_n = \bar{\lambda}f(\mathsf{u}_n)$ and is representable as a graph. Motivated by this, we look for a function $g : \mathbb{R}^d \to \mathbb{R}^d$ such that the manifold

$$\mathsf{v} = \bar{\lambda}f(\mathsf{u}) + hg(\mathsf{u}) \tag{3.16}$$

is invariant for the dynamics of the numerical method:

$$\mathsf{v}_n = \bar{\lambda}f(\mathsf{u}_n) + hg(\mathsf{u}_n) \Longleftrightarrow \mathsf{v}_{n+1} = \bar{\lambda}f(\mathsf{u}_{n+1}) + hg(\mathsf{u}_{n+1}). \tag{3.17}$$

We will prove the existence of such a function g by use of the contraction mapping theorem to find fixed point of mapping T defined in subsection 3.4 below. We seek this fixed point in set Γ which we now define:

Definition 9. *Let $\gamma, \delta > 0$ be as in Lemmas 14, 15. Define $\Gamma := \Gamma(\gamma, \delta)$ to be the closed subset of $C(\mathbb{R}^d; \mathbb{R}^d)$ consisting of γ-bounded functions:*

$$\|g\|_\Gamma := \sup_{\xi \in \mathbb{R}^d} |g(\xi)| \leq \gamma, \quad \forall g \in \Gamma$$

that are δ-Lipshitz:

$$|g(\xi) - g(\eta)| \leq \delta|\xi - \eta|, \quad \forall g \in \Gamma, \xi, \eta \in \mathbb{R}^d.$$

Theorem 10. *Fix $\lambda \in (0, 1)$. Suppose that h is chosen small enough so that Assumption 16 holds. For $n = 0, 1, 2, \ldots$, let u_n, v_n be the sequences given by* (3.15). *Then there is a $\tau > 0$ such that, for all $h \in (0, \tau)$, there is a unique $g \in \Gamma$ such that* (3.17) *holds. Furthermore,*

$$|\mathsf{v}_n - \bar{\lambda} f(\mathsf{u}_n) - hg(\mathsf{u}_n)| \leq (\lambda + h^2\lambda\delta)^n |\mathsf{v}_0 - \bar{\lambda} f(\mathsf{u}_0) - hg(\mathsf{u}_0)|$$

where $\lambda + h^2\lambda\delta < 1$.

The statement of Assumption 16, and the proof of the preceding theorem, are given in Appendix C. The assumption appears somewhat involved at first glance but inspection reveals that it simply places an upper bound on the learning rate h, as detailed in Lemmas 14, 15. The proof of the theorem rests on the Lemmas 18, 19, and 20 which establish that the operator T is well-defined, maps Γ to Γ, and is a contraction on Γ. The operator T is defined, and expressed in a helpful form for the purposes of analysis, in the next subsection.

In the next subsection we obtain the leading order approximation for g, given in equation (3.31). Theorem 10 implies that the large-time dynamics are governed by the dynamics on the invariant manifold. Substituting the leading order approximation for g into the invariant manifold (3.16) and using this expression in the definition (3.14) shows that

$$\mathsf{v}_n = -(1 - \lambda)^{-1} \nabla \left(\Phi(\mathsf{u}_n) + \frac{1}{2} h\bar{\lambda}(\bar{\lambda} - a)|\nabla\Phi(\mathsf{u}_n)|^2 \right), \tag{3.18a}$$

$$\mathsf{u}_n = \mathsf{u}_{n-1} - h(1 - \lambda)^{-1} \nabla \left(\Phi(\mathsf{u}_n) + \frac{1}{2} h\bar{\lambda}(\bar{\lambda} - a)|\nabla\Phi(\mathsf{u}_n)|^2 \right). \tag{3.18b}$$

Setting

$$c = \bar{\lambda}\left(\bar{\lambda} - a + \frac{1}{2}\right) \tag{3.19}$$

we see that for large time the dynamics of momentum methods, including HB and NAG, are approximately those of the modified gradient flow

$$\frac{du}{dt} = -(1-\lambda)^{-1}\nabla\Phi_h(u) \tag{3.20}$$

with

$$\Phi_h(u) = \Phi(u) + \frac{1}{2}hc|\nabla\Phi(u)|^2. \tag{3.21}$$

To see this we proceed as follows. Note that from (3.20)

$$\frac{d^2u}{dt^2} = -\frac{1}{2}(1-\lambda)^{-2}\nabla|\nabla\Phi(u)|^2 + \mathcal{O}(h)$$

then Taylor expansion shows that, for $u_n = u(nh)$,

$$\begin{aligned} u_n &= u_{n-1} + h\dot{u}_n - \frac{h^2}{2}\ddot{u}_n + \mathcal{O}(h^3) \\ &= u_{n-1} - h\bar{\lambda}\left(\nabla\Phi(u_n) + \frac{1}{2}hc\nabla|\nabla\Phi(u_n)|^2\right) + \frac{1}{4}h^2\bar{\lambda}^2\nabla|\nabla\Phi(u_n)|^2 + \mathcal{O}(h^3) \end{aligned}$$

where we have used that

$$Df(u)f(u) = \frac{1}{2}\nabla\left(|\nabla\Phi(u)|^2\right).$$

Choosing $c = \bar{\lambda}(\bar{\lambda} - a + 1/2)$ we see that

$$u_n = u_{n-1} - h(1-\lambda)^{-1}\nabla\left(\Phi(u_n) + \frac{1}{2}h\bar{\lambda}(\bar{\lambda} - a)|\nabla\Phi(u_n)|^2\right) + \mathcal{O}(h^3). \tag{3.22}$$

Notice that comparison of (3.18b) and (3.22) shows that, on the invariant manifold, the dynamics are to $\mathcal{O}(h^2)$ the same as the equation (3.20); this is because the truncation error between (3.18b) and (3.22) is $\mathcal{O}(h^3)$.

Thus we have proved:

Theorem 11. *Suppose that the conditions of Theorem 10 hold. Then for initial data started on the invariant manifold and any $T \geq 0$, there is a constant $C = C(T) > 0$ such that*

$$\sup_{0 \leq nh \leq T} |u_n - \mathsf{u}_n| \leq Ch^2,$$

where $u_n = u(nh)$ solves the modified equation (3.20) *with $c = \bar{\lambda}(\bar{\lambda} - a + 1/2)$.*

Intuition

We will define mapping $T : C(\mathbb{R}^d;\mathbb{R}^d) \to C(\mathbb{R}^d;\mathbb{R}^d)$ via the equations

$$\begin{aligned} p &= \xi + h\lambda\big(\bar\lambda f(\xi) + hg(\xi)\big) + hf\Big(\xi + ha\big(\bar\lambda f(\xi) + hg(\xi)\big)\Big) \\ \bar\lambda f(p) + h(Tg)(p) &= \lambda\big(\bar\lambda f(\xi) + hg(\xi)\big) + f\Big(\xi + ha\big(\bar\lambda f(\xi) + hg(\xi)\big)\Big). \end{aligned} \tag{3.23}$$

A fixed point of the mapping $g \mapsto Tg$ will give function g so that, under (3.23), identity (3.17) holds. Later we will show that, for g in Γ and all h sufficiently small, ξ can be found from (3.23a) for every p, and that thus (3.23b) defines a mapping from $g \in \Gamma$ into $Tg \in C(\mathbb{R}^d;\mathbb{R}^d)$. We will then show that, for h sufficiently small, $T : \Gamma \mapsto \Gamma$ is a contraction.

For any $g \in C(\mathbb{R}^d;\mathbb{R}^d)$ and $\xi \in \mathbb{R}^d$ define

$$w_g(\xi) \coloneqq \bar\lambda f(\xi) + hg(\xi) \tag{3.24}$$

$$z_g(\xi) \coloneqq \lambda w_g(\xi) + f\big(\xi + haw_g(\xi)\big). \tag{3.25}$$

With this notation the fixed point mapping (3.23) for g may be written

$$\begin{aligned} p &= \xi + hz_g(\xi), \\ \bar\lambda f(p) + h(Tg)(p) &= z_g(\xi). \end{aligned} \tag{3.26}$$

Then, by Taylor expansion,

$$\begin{aligned} f\Big(\xi + ha\big(\bar\lambda f(\xi) + hg(\xi)\big)\Big) &= f\big(\xi + haw_g(\xi)\big) \\ &= f(\xi) + ha\int_0^1 Df\big(\xi + shaw_g(\xi)\big)w_g(\xi)ds \\ &= f(\xi) + haI_g^{(1)}(\xi) \end{aligned} \tag{3.27}$$

where the last line defines $I_g^{(1)}$. Similarly

$$\begin{aligned} f(p) &= f(\xi + hz_g(\xi)) \\ &= f(\xi) + h\int_0^1 Df\big(\xi + shz_g(\xi)\big)z_g(\xi)ds \\ &= f(\xi) + hI_g^{(2)}(\xi), \end{aligned} \tag{3.28}$$

where the last line now defines $I_g^{(2)}$. Then (3.23b) becomes

$$\bar\lambda\big(f(\xi) + hI_g^{(2)}(\xi)\big) + h(Tg)(p) = \lambda\bar\lambda f(\xi) + h\lambda g(\xi) + f(\xi) + haI_g^{(1)}(\xi)$$

and we see that

$$(Tg)(p) = \lambda g(\xi) + aI_g^{(1)}(\xi) - \bar{\lambda} I_g^{(2)}(\xi).$$

In this light, we can rewrite the defining equations (3.23) for T as

$$p = \xi + hz_g(\xi), \tag{3.29}$$

$$(Tg)(p) = \lambda g(\xi) + aI_g^{(1)}(\xi) - \bar{\lambda} I_g^{(2)}(\xi). \tag{3.30}$$

for any $\xi \in \mathbb{R}^d$.

Perusal of the above definitions reveals that, to leading order in h,

$$w_g(\xi) = z_g(\xi) = \bar{\lambda} f(\xi), I_g^{(1)}(\xi) = I_g^{(2)}(\xi) = \bar{\lambda} Df(\xi)f(\xi).$$

Thus setting $h = 0$ in (3.29), (3.30) shows that, to leading order in h,

$$g(p) = \bar{\lambda}^2(a - \bar{\lambda})Df(p)f(p). \tag{3.31}$$

Note that since $f(p) = -\nabla\Phi(p)$, Df is the negative Hessian of Φ and is thus symmetric. Hence we can write g in gradient form, leading to

$$g(p) = \frac{1}{2}\bar{\lambda}^2(a - \bar{\lambda})\nabla\big(|\nabla\Phi(p)|^2\big). \tag{3.32}$$

Remark 12. *This modified potential* (3.21) *also arises in the construction of Lyapunov functions for the one-stage theta method – see Corollary 5.6.2 in (A. Stuart and Humphries, 1998).*

Numerical Illustration

In Figure 35 panels (a),(b),(d),(e), we plot the components u_n and v_n found by solving (3.15) with initial conditions $\mathsf{u}_0 = (1, 1)^T$ and $\mathsf{v}_n = (0, 0)^T$ in the case where $\Phi(u) = \frac{1}{2}\langle u, Qu\rangle$. These initial conditions correspond to initializing the map off the invariant manifold. To leading order in h the invariant manifold is given by (see equation (3.18))

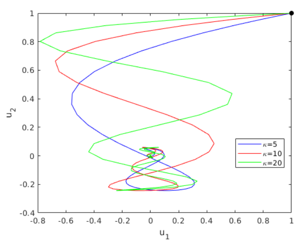
$$v = -(1 - \lambda)^{-1}\nabla\left(\Phi(u) + \frac{1}{2}h\bar{\lambda}(\bar{\lambda} - a)|\nabla\Phi(u)|^2\right). \tag{3.33}$$

To measure the distance of the trajectory shown in panels (a),(b),(d),(e) from the invariant manifold we define

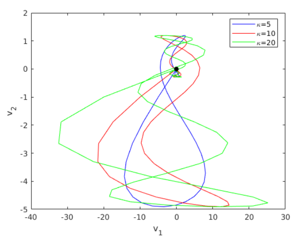
$$\mathsf{e}_n = \left|\mathsf{v}_n + (1 - \lambda)^{-1}\nabla\left(\Phi(\mathsf{u}_n) + \frac{1}{2}h\bar{\lambda}(\bar{\lambda} - a)|\nabla\Phi(\mathsf{u}_n)|^2\right)\right|. \tag{3.34}$$

Panels (c),(f) show the evolution of e_n as well as the (approximate) bound on it found from substituting the leading order approximation of g into the following upper bound from Theorem 10:

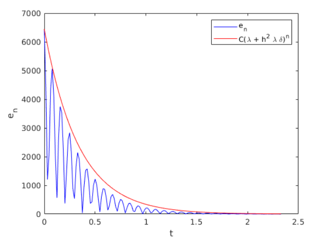
$$(\lambda + h^2\lambda\delta)^n|\mathsf{v}_0 - \bar{\lambda} f(\mathsf{u}_0) - hg(\mathsf{u}_0)|.$$



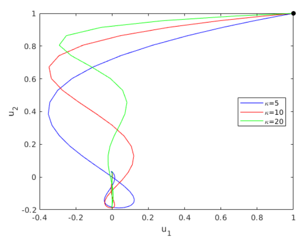
(a) HB: u_n given by (3.15)



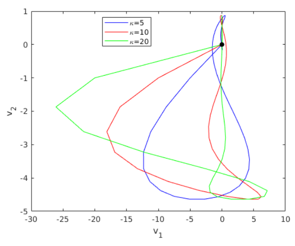
(b) HB: v_n given by (3.15)



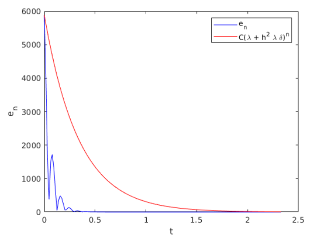
(c) HB: e_n given by (3.34)



(d) NAG: u_n given by (3.15)



(e) NAG: v_n given by (3.15)



(f) NAG: e_n given by (3.34)

Figure 35: Invariant manifold for HB and NAG with $h = 2^{-6}$ and $\lambda = 0.9$ on the two-dimensional problem $\Phi(u) = \frac{1}{2}\langle u, Qu\rangle$, varying the condition number of Q. Panels (c), (f) show the distance from the invariant manifold for the largest condition number $\kappa = 20$.

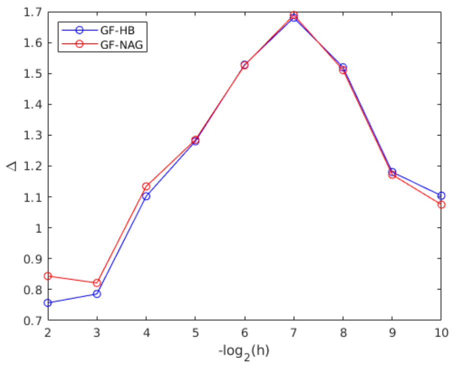
3.5 Deep Learning Example

Our theory is developed under quite restrictive assumptions, in order to keep the proofs relatively simple and to allow a clearer conceptual development. The purpose of the numerical experiments in this section is twofold: firstly to demonstrate that our theory sheds light on a stochastic version of gradient descent applied, furthermore, to a setting in which the objective function does not satisfy the global assumptions which facilitate our analysis; and second to show that methods implemented as we use them here (with learning-rate independent momentum, fixed at every step of the iteration) can out-perform other choices on specific problems.

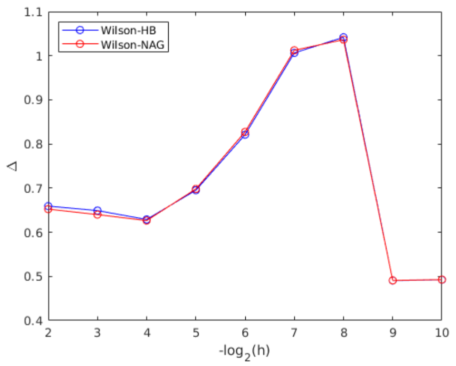
Our numerical experiments in this section are undertaken with in the context of the example given in (Sutskever et al., 2013). We train a deep autoencoder, using the architecture of (Geoffrey Hinton and Salakhutdinov, 2006) on the MNIST dataset (LeCun and Cortes, 2010). Since our work is concerned only with optimization and not generalization, we present our results only on the training set of 60,000 images and ignore the testing set. We fix an initialization of the autoencoder following (Glorot and Bengio, 2010) and use it to test every optimization method. Furthermore, we fix a batch size of 200 and train for 500 epochs, not shuffling the data set during training so that each method sees the same realization of the noise. We use the

	$h = 2^0$	$h = 2^{-1}$	$h = 2^{-2}$	$h = 2^{-3}$	$h = 2^{-4}$	$h = 2^{-5}$	$h = 2^{-6}$
GF	n/a	4.3948	4.5954	5.6769	**7.0049**	**8.6468**	**10.6548**
HB	3.6775	4.0157	4.5429	5.6447	7.0720	8.7070	10.6848
NAG	**3.2808**	**3.7166**	**4.4579**	**5.6087**	7.0557	8.6987	10.6814
Wilson	6.7395	7.5177	8.3491	9.2543	10.2761	11.3776	12.4123
HB-μ	5.7099	6.6146	7.6202	8.6629	9.7838	11.0039	12.1743
NAG-μ	**5.6867**	**6.6033**	**7.6131**	**8.6556**	**9.7783**	**11.0015**	**12.1738**

Figure 36: Final training errors for the autoencoder on MNIST for six training methods over different learning rates. GF refers to equation (3.35) while HB and NAG to (3.7) all with fixed $\lambda = 0.9$.



(a) HB, NAG to (3.35)



(b) HB-μ, NAG-μ to (3.36)

Figure 37: The numerical rate of convergence for the parameters of the autoencoder, as a function of the learning rate h, of HB and NAG to (3.35) (a), as well as of HB-μ and NAG-μ to (3.36) (b).

mean-squared error as our loss function.

We compare HB and NAG given by (3.7) to the re-scaled gradient flow (3.8) which we discretize in the standard way to yield the numerical method

$$\mathsf{u}_{n+1} = \mathsf{u}_n - \frac{h}{(1-\lambda)} \nabla \Phi(\mathsf{u}_n), \tag{3.35}$$

hence the momentum term λ only acts to re-scale the learning rate. We do not test against equation (3.11) because, to discretize it faithfully, we would need to use a time-step much lower than h (because (3.11) contains a term of order h), but doing so would mean that we need to train for many more epochs compared to HB and NAG so that the same final time is reached. This, in turn, implies that the methods would see different realization of the noise. Thus, to compare them well, we would need to perform a Monte Carlo simulation, however, since we do not state any of our results in a stochastic setting, we leave this for future work.

We also compare our results to those of (Ashia C. Wilson, Recht, and Michael I. Jordan, 2016) which analyze HB and NAG in the setting where Φ is μ-strongly convex and λ is given by (3.5) that is

$$\lambda = \frac{1-\sqrt{\mu h}}{1+\sqrt{\mu h}}.$$

They obtain the limiting equation

$$\ddot{u} + 2\sqrt{\mu}\dot{u} + \nabla\Phi(u) = 0$$

which we discretize via a split-step method to yield

$$\begin{aligned} \mathsf{u}_{n+1} &= \mathsf{u}_n + \frac{1}{2\sqrt{\mu}}\left(1 - e^{-2\sqrt{\mu h}}\right)\mathsf{v}_n \\ \mathsf{v}_{n+1} &= e^{-2\sqrt{\mu h}}\mathsf{v}_n - \sqrt{h}\nabla\Phi(\mathsf{u}_{n+1}) \end{aligned} \tag{3.36}$$

where we have mapped the the time-step h in HB and NAG to $\sqrt{h}$ as in done in (Ashia C. Wilson, Recht, and Michael I. Jordan, 2016). We choose this discretization because it allows us to directly solve for the linear parts of the ODE (in the enlarged state-space), yielding a more accurate approximation than the forward-Euler method used to obtain (3.35). A detailed derivation is given in Appendix D. We will refer to the method in equation (3.36) as Wilson. Further we refer to equation (3.7) with λ given by (3.5) and $a = 0$ as HB-μ and equation (3.7) with λ given by (3.5) and $a = \lambda$ as NAG-μ. Since deep neural networks are not strongly convex, there is no single optimal choice of μ; we simply set $\mu = 1$ in our experiments.

Figure 36 gives the final training errors for each method for several learning rates. We were unable to train the autoencoder using (3.35) with $h = 1$ since $\lambda = 0.9$ implies an effective learning rate of 10 for which the system blows up. In general, NAG is the best performing method for relatively large h which is an observation that is consistently made in the deep learning literature. Further, we note that as the learning rate decreases, the final errors become closer indicating convergence to the appropriate limiting equations. Figure 36 showcases the practical effectiveness of momentum methods as they provide a way of discretizing the gradient flow (3.2) with a large effective learning rate that forward Euler cannot accommodate. From this perspective, we can view momentum methods as providing a more stable discretization to gradient flows in a manner illustrated by (3.20). Such a viewpoint informs the works (Scieur et al., 2017; Betancourt, Michael I. Jordan, and Ashia C. Wilson, 2018; Zhang et al., 2018).

To further illustrate the point of convergence to the limiting equation, we compute the numerical rate of convergence, defined in Section 3.2, as a function of h for the neural network parameters between (3.35) and HB and NAG as well as between (3.36) and HB-μ and NAG-μ. Figure 37 gives the results. We note that this rate is around 1 as predicted by our theory while the rate for (3.36) is around 0.5 which is also consistent with the theory in (Ashia C. Wilson, Recht, and Michael I. Jordan, 2016).

3.6 Conclusion

Together, equations (3.8), (3.11) and (3.20) describe the dynamical systems which are approximated by momentum methods, when implemented with fixed momentum, in a manner made precise by the four theorems in this paper. The insight obtained from these theorems sheds light on how momentum methods perform optimization tasks.

3.7 Proof of Theorem 7

Proof. Taylor expanding yields

$$u_{n+1} = u_n + h\bar{\lambda} f(u_n) + \mathcal{O}(h^2)$$

and

$$u_n = u_{n-1} + h\bar{\lambda} f(u_n) + \mathcal{O}(h^2).$$

Hence

$$(1+\lambda)u_n - \lambda u_{n-1} = u_n + h\lambda\bar{\lambda} f(u_n) + \mathcal{O}(h^2).$$

Subtracting the third identity from the first, we find that

$$u_{n+1} - ((1+\lambda)u_n - \lambda u_{n-1}) = hf(u_n) + \mathcal{O}(h^2)$$

by noting $\bar{\lambda} - \bar{\lambda}\lambda = 1$. Similarly,

$$a(u_n - u_{n-1}) = ha\bar{\lambda} f(u_n) + \mathcal{O}(h^2)$$

hence Taylor expanding yields

$$\begin{aligned} f(u_n + a(u_n - u_{n-1})) &= f(u_n) + aDf(u_n)(u_n - u_{n-1}) \\ &\quad + a^2 \int_0^1 (1-s) D^2 f(u_n + sa(u_n - u_{n-1}))[u_n - u_{n-1}]^2 ds \\ &= f(u_n) + ha\bar{\lambda} Df(u_n) f(u_n) + \mathcal{O}(h^2). \end{aligned}$$

From this, we conclude that

$$hf(u_n + a(u_n - u_{n-1})) = hf(u_n) + \mathcal{O}(h^2)$$

hence

$$u_{n+1} = (1+\lambda)u_n - \lambda u_{n-1} + hf(u_n + a(u_n - u_{n-1})) + \mathcal{O}(h^2).$$

Define the error $e_n \coloneqq u_n - \mathsf{u}_n$ then

$$\begin{aligned} e_{n+1} &= (1+\lambda)e_n - \lambda e_{n-1} + h\left(f(u_n + a(u_n - u_{n-1})) - f(\mathsf{u}_n + a(\mathsf{u}_n - \mathsf{u}_{n-1}))\right) + \mathcal{O}(h^2) \\ &= (1+\lambda)e_n - \lambda e_{n-1} + h\mathsf{M}_n((1+a)e_n - ae_{n-1}) + \mathcal{O}(h^2) \end{aligned}$$

where, from the mean value theorem, we have

$$\mathsf{M}_n = \int_0^1 Df\Big(s\big(u_n + a(u_n - u_{n-1})\big) + \big(1-s\big)\big(\mathsf{u}_n + a(\mathsf{u}_n - \mathsf{u}_{n-1})\big)\Big)ds.$$

Now define the concatenation $E_{n+1} \coloneqq [e_{n+1}, e_n] \in \mathbb{R}^{2d}$ then

$$E_{n+1} = A^{(\lambda)}E_n + hA_n^{(a)}E_n + \mathcal{O}(h^2)$$

where $A^{(\lambda)}, A_n^{(a)} \in \mathbb{R}^{2d\times 2d}$ are the block matrices

$$A^{(\lambda)} \coloneqq \begin{bmatrix} (1+\lambda)I & -\lambda I \\ I & 0I \end{bmatrix}, \quad A_n^{(a)} \coloneqq \begin{bmatrix} (1+a)\mathsf{M}_n & -a\mathsf{M}_n \\ 0I & 0I \end{bmatrix}$$

with $I \in \mathbb{R}^{d\times d}$ the identity. We note that $A^{(\lambda)}$ has minimal polynomial

$$\mu_{A^{(\lambda)}}(z) = (z-1)(z-\lambda)$$

and is hence diagonalizable. Thus there is a norm on $\|\cdot\|$ on $\mathbb{R}^{2d}$ such that its induced matrix norm $\|\cdot\|_m$ satifies $\|A^{(\lambda)}\|_m = \rho(A^{(\lambda)})$ where $\rho : \mathbb{R}^{2d\times 2d} \to \mathbb{R}_+$ maps a matrix to its spectral radius. Hence, since $\lambda \in (0,1)$, we have $\|A^{(\lambda)}\|_m = 1$. Thus

$$\|E_{n+1}\| \leq (1 + h\|A_n^{(a)}\|_m)\|E_n\| + \mathcal{O}(h^2).$$

Then, by finite dimensional norm equivalence, there is a constant $\alpha > 0$, independent of h, such that

$$\begin{aligned} \|A_n^{(a)}\|_m &\leq \alpha \left\| \begin{bmatrix} 1+a & -a \\ 0 & 0 \end{bmatrix} \otimes \mathsf{M}_n \right\|_2 \\ &= \alpha\sqrt{2a^2 + 2a + 1}\|\mathsf{M}_n\|_2 \end{aligned}$$

where $\|\cdot\|_2$ denotes the spectral 2-norm. Using Assumption 6, we have

$$\|\mathsf{M}_n\|_2 \leq B_1$$

thus, letting $c := \alpha\sqrt{2a^2 + 2a + 1}B_1$, we find

$$\|E_{n+1}\| \leq (1 + hc)\|E_n\| + \mathcal{O}(h^2).$$

Then, by Grönwall lemma,

$$\begin{aligned}\|E_{n+1}\| &\leq (1 + hc)^n\|E_1\|_n + \frac{(1 + hc)^{n+1} - 1}{ch}\mathcal{O}(h^2)\\ &= (1 + hc)^n\|E_1\|_n + \mathcal{O}(h)\end{aligned}$$

noting that the constant in the $\mathcal{O}(h)$ term is bounded above in terms of T, but independently of h. Finally, we check the initial condition

$$E_1 = \begin{bmatrix} u_1 - \mathsf{u}_1 \\ u_0 - \mathsf{u}_0 \end{bmatrix} = \begin{bmatrix} h(\bar{\lambda} - 1)f(\mathsf{u}_0) + \mathcal{O}(h^2) \\ 0 \end{bmatrix} = \mathcal{O}(h)$$

as desired. □

3.8 Proof of Theorem 8

Proof. Taylor expanding yields

$$u_{n\pm1} = u_n \pm h\dot{u}_n + \frac{h^2}{2}\ddot{u}_n \pm \frac{h^3}{2}I_n^{\pm}$$

where

$$I_n^{\pm} = \int_0^1 (1 - s)^2\dddot{u}((n \pm s)h)ds.$$

Then using equation (3.11)

$$\begin{aligned}u_{n+1} - u_n - \lambda(u_n - u_{n-1}) &= h(1 - \lambda)\dot{u}_n + \frac{h^2}{2}(1 + \lambda)\ddot{u}_n + \frac{h^3}{2}(I_n^+ - \lambda I_n^-)\\ &= hf(u_n) + h^2a(1 - \lambda)\ddot{u}_n + \frac{h^3}{2}(I_n^+ - \lambda I_n^-).\end{aligned} \tag{3.37}$$

Similarly

$$a(u_n - u_{n-1}) = ha\dot{u}_n - \frac{h^2}{2}a\ddot{u}_n + \frac{h^3}{2}aI_n^-$$

hence

$$f(u_n + a(u_n - u_{n-1})) = f(u_n) + haDf(u_n)\dot{u}_n - Df(u_n)\left(\frac{h^2}{2}a\ddot{u}_n - \frac{h^3}{2}aI_n^-\right) + I_n^f$$

where

$$I_n^f = a^2 \int_0^1 (1-s) D^2 f(u_n + sa(u_n - u_{n-1}))[u_n - u_{n-1}]^2 ds.$$

Differentiating (3.11) yields

$$h\alpha \frac{d^3 u}{dt^3} + (1-\lambda)\frac{d^2 u}{dt^2} = Df(u)\frac{du}{dt}$$

hence

$$\begin{aligned} hf(u_n + a(u_n - u_{n-1})) &= hf(u_n) + h^2 a\left(h\alpha \dddot{u}_n + (1-\lambda)\ddot{u}_n\right) \\ &\quad - Df(u_n)\left(\frac{h^3}{2} a\ddot{u}_n - \frac{h^4}{2} a I_n^-\right) + hI_n^f \\ &= hf(u_n) + h^2 a(1-\lambda)\ddot{u}_n + h^3 a\alpha \dddot{u}_n \\ &\quad - Df(u_n)\left(\frac{h^3}{2} a\ddot{u}_n - \frac{h^4}{2} a I_n^-\right) + hI_n^f. \end{aligned}$$

Rearranging this we obtain an expression for $hf(u_n)$ which we plug into equation (3.37) to yield

$$u_{n+1} - u_n - \lambda(u_n - u_{n-1}) = hf(u_n + a(u_n - u_{n-1})) + \mathrm{LT}_n$$

where

$$\mathrm{LT}_n = \underbrace{\frac{h^3}{2}(I_n^+ - \lambda I_n^-)}_{\mathcal{O}\left(h\exp\left(-\frac{(1-\lambda)}{2\alpha}n\right)\right)} - \underbrace{h^3 a\alpha \dddot{u}_n}_{\mathcal{O}\left(h\exp\left(-\frac{(1-\lambda)}{2\alpha}n\right)\right)} + \underbrace{Df(u_n)\left(\frac{h^3}{2} a\ddot{u}_n - \frac{h^4}{2} a I_n^-\right)}_{\mathcal{O}(h^2)} - \underbrace{hI_n^f}_{\mathcal{O}(h^3)}.$$

The bounds (in braces) on the four terms above follow from employing Assumption 6 and Lemma 13. From them we deduce the existence of constants $K_1, K_2 > 0$ independent of h such that

$$|\mathrm{LT}_n| \le hK_1 \exp\left(-\frac{(1-\lambda)}{2\alpha}n\right) + h^2 K_2.$$

We proceed similarly to the proof of Theorem 7, but with a different truncation error structure, and find the error satsifies

$$\|E_{n+1}\| \le (1+hc)\|E_n\| + hK_1 \exp\left(-\frac{(1-\lambda)}{2\alpha}n\right) + h^2 K_2$$

where we abuse notation and continue to write K_1, K_2 when, in fact, the constants have changed by use of finite-dimensional norm equivalence. Define $K_3 := K_2/c$

then summing this error, we find

$$\begin{aligned}\|E_{n+1}\| &\le (1+hc)^n\|E_1\| + hK_3((1+hc)^{n+1}-1)\\ &\quad + hK_1\sum_{j=0}^{n}(1+hc)^j\text{exp}\left(-\frac{(1-\lambda)}{2\alpha}(n-j)\right)\\ &= (1+hc)^n\|E_1\| + hK_3((1+hc)^{n+1}-1) + hK_1S_n\end{aligned}$$

where

$$S_n = \text{exp}\left(-\frac{(1-\lambda)}{2\alpha}n\right)\left(\frac{(1+hc)^{n+1}\text{exp}\left(\frac{(1-\lambda)}{2\alpha}(n+1)\right)-1}{(1+hc)\text{exp}\left(\frac{1-\lambda}{2\alpha}\right)-1}\right).$$

Let $T = nh$ then

$$\begin{aligned}S_n &\le \frac{(1+hc)^{n+1}\text{exp}\left(\frac{1-\lambda}{2\alpha}\right)}{(1+hc)\text{exp}\left(\frac{1-\lambda}{2\alpha}\right)-1}\\ &\le \frac{2\text{exp}\left(cT+\frac{1-\lambda}{2\alpha}\right)}{\text{exp}\left(\frac{1-\lambda}{2\alpha}\right)-1}.\end{aligned}$$

From this we deduce that

$$\|E_{n+1}\| \le (1+hc)^n\|E_1\| + \mathcal{O}(h)$$

noting that the constant in the $\mathcal{O}(h)$ term is bounded above in terms of T, but independently of h. For the initial condition, we check

$$u_1 - \mathsf{u}_1 = h(\mathsf{u}_0' - f(\mathsf{u}_0)) + \frac{h^2}{2}\ddot{u}_0 + \frac{h^3}{2}I_0^+$$

which is $\mathcal{O}(h)$ by Lemma 13. Putting the bounds together we obtain

$$\sup_{0\le nh\le T}\|E_n\| \le C(T)h.$$

□

Lemma 13. *Suppose Assumption 6 holds and let $u \in C^3([0,\infty);\mathbb{R}^d)$ be the solution to*

$$\begin{aligned}&h\alpha\frac{d^2u}{dt^2} + (1-\lambda)\frac{du}{dt} = f(u)\\ &u(0) = \mathsf{u}_0, \quad \frac{du}{dt}(0) = \mathsf{v}_0\end{aligned}$$

for some $\mathsf{u}_0, \mathsf{v}_0 \in \mathbb{R}^d$ *and* $\alpha > 0$ *independent of* h. *Suppose* $h \leq (1-\lambda)^2/2\alpha B_1$ *then there are constants* $C^{(1)}, C_1^{(2)}, C_2^{(2)}, C_1^{(3)}, C_2^{(3)} > 0$ *independent of* h *such that for any* $t \in [0, \infty)$*,*

$$\begin{aligned}
|\dot{u}(t)| &\leq C^{(1)}, \\
|\ddot{u}(t)| &\leq \frac{C_1^{(2)}}{h} \mathit{exp}\left(-\frac{(1-\lambda)}{2h\alpha}t\right) + C_2^{(2)}, \\
|\dddot{u}(t)| &\leq \frac{C_1^{(3)}}{h^2} \mathit{exp}\left(-\frac{(1-\lambda)}{2h\alpha}t\right) + C_2^{(3)}.
\end{aligned}$$

Proof. Define $v := \dot{u}$ then

$$\dot{v} = -\frac{1}{h\alpha}\left((1-\lambda)v - f(u)\right).$$

Define $w := (1-\lambda)v - f(u)$ hence $\dot{v} = -(1/h\alpha)w$ and $\dot{u} = v = \bar{\lambda}(w + f(u))$. Thus

$$\begin{aligned}
\dot{w} &= (1-\lambda)\dot{v} - Df(u)\dot{u} \\
&= -\frac{(1-\lambda)}{h\alpha}w - Df(u)(\bar{\lambda}(w + f(u))).
\end{aligned}$$

Hence we find

$$\begin{aligned}
\frac{1}{2}\frac{d}{dt}|w|^2 &= -\frac{(1-\lambda)}{h\alpha}|w|^2 - \bar{\lambda}\langle w, Df(u)w\rangle - \bar{\lambda}\langle w, Df(u)f(u)\rangle \\
&\leq -\frac{(1-\lambda)}{h\alpha}|w|^2 + \bar{\lambda}|\langle w, Df(u)w\rangle| + \bar{\lambda}|\langle w, Df(u)f(u)\rangle| \\
&\leq -\frac{(1-\lambda)}{h\alpha}|w|^2 + \bar{\lambda}B_1|w|^2 + \bar{\lambda}B_0B_1|w| \\
&\leq -\frac{(1-\lambda)}{h\alpha}|w|^2 + \frac{(1-\lambda)}{2h\alpha}|w|^2 + \bar{\lambda}B_0B_1|w| \\
&= -\frac{(1-\lambda)}{2h\alpha}|w|^2 + \bar{\lambda}B_0B_1|w|
\end{aligned}$$

by noting that our assumption $h \leq (1-\lambda)^2/2\alpha B_1$ implies $\bar{\lambda}B_1 \leq (1-\lambda)/2h\alpha$. Hence

$$\frac{d}{dt}|w| \leq -\frac{(1-\lambda)}{2h\alpha}|w| + \bar{\lambda}B_0B_1$$

so, by Grönwall lemma,

$$\begin{aligned}
|w(t)| &\leq \exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right)|w(0)| + 2h\bar{\lambda}^2\alpha B_0B_1\left(1 - \exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right)\right) \\
&\leq \exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right)|w(0)| + h\beta_1
\end{aligned}$$

where we define $\beta_1 := 2\bar{\lambda}^2\alpha B_0 B_1$. Hence

$$\begin{aligned}
|\ddot{u}(t)| &= |\dot{v}(t)| \\
&= \frac{1}{h\alpha}|w(t)| \\
&\leq \frac{1}{h\alpha}\exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right)|w(0)| + \frac{\beta_1}{\alpha} \\
&= \frac{|(1-\lambda)\mathsf{v}_0 - f(\mathsf{u}_0)|}{h\alpha}\exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right) + \frac{\beta_1}{\alpha},
\end{aligned}$$

thus setting $C_1^{(2)} = |(1-\lambda)\mathsf{v}_0 - f(\mathsf{u}_0)|/\alpha$ and $C_1^{(2)} = \beta_1/\alpha$ gives the desired result. Further,

$$\begin{aligned}
|\dot{u}(t)| &= |v(t)| \\
&\leq \bar{\lambda}(|w(t)| + |f(u(t))|) \\
&\leq \bar{\lambda}(|w(0)| + h\beta_1 + B_0),
\end{aligned}$$

hence we deduce the existence of $C^{(1)}$. Now define $z := \dot{w}$ then

$$\dot{z} = -\frac{(1-\lambda)}{h\alpha}z - \bar{\lambda}Df(u)z + G(u,v,w)$$

where we define

$$G(u,v,w) := -\bar{\lambda}(Df(u)(Df(u)v) + D^2f(u)[v,w] + D^2f(u)[Df(u)v, f(u)]).$$

Using Assumption 6 and our bounds on w and v, we deduce that there is a constant $C > 0$ independent of h such that

$$|G(u,v,w)| \leq C,$$

hence

$$\begin{aligned}
\frac{1}{2}\frac{d}{dt}|z|^2 &= -\frac{(1-\lambda)}{h\alpha}|z|^2 - \bar{\lambda}\langle z, Df(u)z\rangle + \langle z, G(u,v,w)\rangle \\
&\leq -\frac{(1-\lambda)}{h\alpha}|z|^2 + \bar{\lambda}B_1|z|^2 + C|z| \\
&\leq -\frac{(1-\lambda)}{2h\alpha}|z|^2 + C|z|
\end{aligned}$$

as before. Thus we find

$$\frac{d}{dt}|z| \leq -\frac{(1-\lambda)}{2h\alpha}|z| + C$$

so, by Grönwall lemma,

$$|z(t)| \leq \exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right)|z(0)| + h\beta_2$$

where we define $\beta_2 := 2\bar{\lambda}\alpha C$. Recall that

$$\dddot{u} = \ddot{v} = -\frac{1}{h\alpha}\dot{w} = -\frac{1}{h\alpha}z$$

and note

$$|z(0)| \leq \frac{(1-\lambda)|(1-\lambda)\mathsf{v}_0 - f(\mathsf{u}_0)|}{h\alpha} + B_1|\mathsf{v}_0|,$$

hence we find

$$|\dddot{u}(t)| \leq \left(\frac{(1-\lambda)|(1-\lambda)\mathsf{v}_0 - f(\mathsf{u}_0)}{h^2\alpha^2} + \frac{B_1|\mathsf{v}_0|}{h\alpha}\right)\exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right) + \frac{\beta_2}{\alpha}.$$

Thus we deduce that there is a constant $C_1^{(3)} > 0$ independent of h such that

$$|\dddot{u}(t)| \leq \frac{C_1^{(3)}}{h^2}\exp\left(-\frac{(1-\lambda)}{2h\alpha}t\right) + C_2^{(3)}$$

as desired where $C_2^{(3)} = \beta_2/\alpha$. □

One readily verifies that the result of Lemma 13 is tight by considering the one-dimensional case with $f(u) = -u$. This implies that the result of Theorem 8 cannot be improved without further assumptions.

3.9 Proof of Theorem 10

For the results of Section 3.4 we make the following assumption on the size of h. Recall first that by Assumption 6 there are constants $B_0, B_1, B_2 > 0$ such that

$$\|D^{j-1}f\| = \|D^j\Phi\| \leq B_{j-1}$$

for $j = 1, 2, 3$.

Lemma 14. *Suppose $h > 0$ is small enough such that*

$$\lambda + hB_1(a + \lambda\bar{\lambda}) < 1$$

then there is a $\tau_1 > 0$ such that for any $\gamma \in [\tau_1, \infty)$

$$(\lambda + hB_1(a + \lambda\bar{\lambda}))\gamma + \bar{\lambda}B_0B_1(a + \bar{\lambda}) \leq \gamma. \tag{3.38}$$

Using Lemma 14 fix $\gamma \in [\tau_1, \infty)$ and define the constants

$$\begin{aligned}
K_1 &:= \bar{\lambda} B_0 + h\gamma \\
K_3 &:= B_0 + \lambda K_1 \\
\alpha_2 &:= h^2(\lambda + haB_1), \\
\alpha_1 &:= \lambda - 1 + h\left(B_1(\bar{\lambda} + a(1 + h\bar{\lambda}B_1)\right) + \lambda\bar{\lambda}(B_1 + hB_2K_3) \\
&\quad + ha\left(aB_2K_1 + B_1\bar{\lambda}(B_1 + hB_2K_3)\right), \\
\alpha_0 &:= aB_2K_1(1 + ha\bar{\lambda}B_1) + \bar{\lambda}(aB_1^2 + B_2K_3) + \bar{\lambda}^2 B_1(1 + haB_1)(B_1 + hB_2K_3).
\end{aligned} \tag{3.39}$$

Lemma 15. *Suppose $h > 0$ is small enough such that*

$$\alpha_1^2 > 4\alpha_2\alpha_0, \quad \alpha_1 < 0$$

then there are $\tau_2^{\pm} > 0$ such that for any $\delta \in (\tau_2^-, \tau_2^+]$

$$\alpha_2\delta^2 + \alpha_1\delta + \alpha_0 \leq 0. \tag{3.40}$$

Using Lemma 15 fix $\delta \in (\tau_2^-, \tau_2^+]$. We make the following assumption on the size of the learning rate h which is achievable since $\lambda \in (0, 1)$.

Assumption 16. *Let Assumption 6 hold and suppose $h > 0$ is small enough such that the assumptions of Lemmas 14, 15 hold. Define $K_2 := \bar{\lambda}B_1 + h\delta$ and suppose $h > 0$ is small enough such that*

$$c := h(\lambda K_2 + B_1(1 + haK_2)) < 1. \tag{3.41}$$

Define constants

$$\begin{aligned}
Q_1 &:= \lambda\delta + a(B_1K_2 + B_2K_1(1 + haK_2)) \\
&\quad + \bar{\lambda}((B_1 + hB_2K_3)(\lambda K_2 + B_1(1 + haK_2)) + B_2K_3), \\
Q_2 &:= h(a(B_1 + haB_2K_1) + \bar{\lambda}(\lambda + haB_1)(B_1 + hB_2K_3)), \\
Q_3 &:= h(\lambda K_2 + B_1(1 + haK_2)), \\
\mu &:= \lambda + Q_2 + \frac{h^2(\lambda + haB_1)Q_1}{1 - Q_3}.
\end{aligned} \tag{3.42}$$

Suppose $h > 0$ is small enough such that

$$Q_3 < 1, \quad \mu < 1. \tag{3.43}$$

Lastly assume $h > 0$ is small enough such that

$$\lambda + h^2\lambda\delta < 1. \tag{3.44}$$

We now prove Lemma 14.

Proof. Since $\lambda + hB_1(a+\lambda\bar{\lambda}) < 1$ and $\bar{\lambda}B_0B_1(a+\bar{\lambda}) > 0$ the line defined by

$$(\lambda + hB_1(a+\lambda\bar{\lambda}))\gamma + \bar{\lambda}B_0B_1(a+\bar{\lambda})$$

will intersect the identity line at a positive γ and lie below it thereafter. Hence setting

$$\tau_1 = \frac{\bar{\lambda}B_0B_1(a+\bar{\lambda})}{1-\lambda+hB_1(a+\lambda\bar{\lambda})}$$

completes the proof. □

We now prove Lemma 15.

Proof. Note that since $\alpha_2 > 0$, the parabola defined by

$$\alpha_2\delta^2 + \alpha_1\delta + \alpha_0$$

is upward-pointing and has roots

$$\zeta_\pm = \frac{-\alpha_1 \pm \sqrt{\alpha_1^2 - 4\alpha_2\alpha_0}}{2\alpha_2}.$$

Since $\alpha_1^2 > 4\alpha_2\alpha_0$, $\zeta_\pm \in \mathbb{R}$ with $\zeta_+ \neq \zeta_-$. Since $\alpha_1 < 0$, $\zeta_+ > 0$ hence setting $\tau_2^+ = \zeta_+$ and $\tau_2^- = \max\{0, \zeta_-\}$ completes the proof. □

We now prove Theorem 10. The proof refers to four lemmas whose statements and proofs follow it.

Proof. Define $\tau > 0$ as the maximum h such that Assumption 16 holds. The contraction mapping principle together with Lemmas 18, 19, and 20 show that the operator T defined by (3.29) and (3.30) has a unique fixed point in Γ. Hence, from its definition and equation (3.23b), we immediately obtain the existence result. We now show exponential attractivity. Recall the definition of the operator T namely equations (3.29), (3.30):

$$p = \xi + hz_g(\xi)$$
$$(Tg)(p) = \lambda g(\xi) + aI_g^{(1)}(\xi) - \bar{\lambda}I_g^{(2)}(\xi).$$

Let $g \in \Gamma$ be the fixed point of T and set

$$\begin{aligned} p &= \mathsf{u}_n + hz_g(\mathsf{u}_n) \\ g(p) &= \lambda g(\mathsf{u}_n) + aI_g^{(1)}(\mathsf{u}_n) - \bar{\lambda} I_g^{(2)}(\mathsf{u}_n). \end{aligned}$$

Then

$$\begin{aligned} |\mathsf{v}_{n+1} - \bar{\lambda} f(\mathsf{u}_{n+1}) - hg(\mathsf{u}_{n+1})| &\leq |\mathsf{v}_{n+1} - \bar{\lambda} f(\mathsf{u}_{n+1}) - hg(p)| + h|g(p) - g(\mathsf{u}_{n+1})| \\ &\leq |\mathsf{v}_{n+1} - \bar{\lambda} f(\mathsf{u}_{n+1}) - hg(p)| + h\delta|p - \mathsf{u}_{n+1}| \end{aligned}$$

since $g \in \Gamma$. Since, by definition,

$$\mathsf{v}_{n+1} = \lambda \mathsf{v}_n + f(\mathsf{u}_n + ha\mathsf{v}_n)$$

we have,

$$\begin{aligned} |\mathsf{v}_{n+1} - \bar{\lambda} f(\mathsf{u}_{n+1}) - hg(p)| &= |\lambda \mathsf{v}_n + f(\mathsf{u}_n + ha\mathsf{v}_n) - \bar{\lambda} f(\mathsf{u}_{n+1}) \\ &\quad - h(\lambda g(\mathsf{u}_n) + aI_g^{(1)}(\mathsf{u}_n) - \bar{\lambda} I_g^{(2)}(\mathsf{u}_n))| \\ &= \lambda |\mathsf{v}_n - \bar{\lambda} f(\mathsf{u}_n) - hg(\mathsf{u}_n)| \end{aligned}$$

by noting that

$$\begin{aligned} f(\mathsf{u}_n + ha\mathsf{v}_n) &= f(\mathsf{u}_n) + haI_g^{(1)}(\mathsf{u}_n) \\ f(\mathsf{u}_{n+1}) &= f(\mathsf{u}_n) + hI_g^{(2)}(\mathsf{u}_n). \end{aligned}$$

From definition,

$$\mathsf{u}_{n+1} = \mathsf{u}_n + h\lambda \mathsf{v}_n + hf(\mathsf{u}_n + ha\mathsf{v}_n),$$

thus

$$\begin{aligned} |p - \mathsf{u}_{n+1}| &= |\mathsf{u}_n + hz_g(\mathsf{u}_n) - \mathsf{u}_n - h\lambda \mathsf{v}_n - hf(\mathsf{u}_n + ha\mathsf{v}_n)| \\ &= h|\lambda(\bar{\lambda} f(\mathsf{u}_n) + hg(\mathsf{u}_n)) + f(\mathsf{u}_n + ha\mathsf{v}_n) - \lambda \mathsf{v}_n - f(\mathsf{u}_n + ha\mathsf{v}_n)| \\ &= h\lambda |\mathsf{v}_n - \bar{\lambda} f(\mathsf{u}_n) - hg(\mathsf{u}_n)|. \end{aligned}$$

Hence

$$|\mathsf{v}_{n+1} - \bar{\lambda} f(\mathsf{u}_{n+1}) - hg(\mathsf{u}_{n+1})| \leq (\lambda + h^2 \lambda \delta)|\mathsf{v}_n - \bar{\lambda} f(\mathsf{u}_n) - hg(\mathsf{u}_n)|$$

as desired. By Assumption 16, $\lambda + h^2\lambda\delta < 1$. □

The following lemma gives basic bounds which are used in the proofs of Lemmas 18, 19, 20.

Lemma 17. *Let $g, q \in \Gamma$ and $\xi, \eta \in \mathbb{R}^d$ then the quantities defined by* (3.24)*,* (3.25)*,* (3.27)*, and* (3.28) *satisfy the following:*

$$\begin{aligned}
|w_g(\xi)| &\leq K_1, \\
|w_g(\xi) - w_g(\eta)| &\leq K_2|\xi - \eta|, \\
|w_g(\xi) - w_q(\xi)| &\leq h|g(\xi) - q(\xi)|, \\
|z_g(\xi)| &\leq K_3, \\
|z_g(\xi) - z_g(\eta)| &\leq (\lambda K_2 + B_1(1 + haK_2))\,|\xi - \eta|, \\
|z_g(\xi) - z_q(\xi)| &\leq h\,(\lambda + haB_1)\,|g(\xi) - q(\xi)|, \\
|I_g^{(1)}(\xi)| &\leq B_1K_1, \\
|I_g^{(1)}(\xi) - I_g^{(1)}(\eta)| &\leq (B_1K_2 + B_2K_1(1 + haK_2))|\xi - \eta|, \\
|I_g^{(1)}(\xi) - I_q^{(1)}(\xi)| &\leq h(B_1 + haB_2K_1)|g(\xi) - q(\xi)|, \\
|I_g^{(2)}(\xi)| &\leq B_1K_3 \\
|I_g^{(2)}(\xi) - I_g^{(2)}(\eta)| &\leq ((B_1 + hB_2K_3)(\lambda K_2 + B_1(1 + haK_2)) + B_2K_3)|\xi - \eta|, \\
|I_g^{(2)}(\xi) - I_q^{(2)}(\xi)| &\leq h(\lambda + hB_1a)(B_1 + hB_2K_3)|g(\xi) - q(\xi)|.
\end{aligned}$$

Proof. These bounds relay on applications of the triangle inequality together with boundedness of f and its derivatives as well as the fact that functions in Γ are bounded and Lipschitz. To illustrate the idea, we will prove the bounds for $w_g, w_q, I_g^{(1)}$, and $I_q^{(1)}$. To that end,

$$\begin{aligned}
|w_g(\xi)| &= |\bar{\lambda} f(\xi) + hg(\xi)| \\
&\leq \bar{\lambda}|f(\xi)| + h|g(\xi)| \\
&\leq \bar{\lambda}B_0 + h\gamma \\
&= K_1,
\end{aligned}$$

establishing the first bound. For the second,

$$\begin{aligned}
|w_g(\xi) - w_g(\eta)| &\leq \bar{\lambda}|f(\xi) - f(\eta)| + h|g(\xi) - g(\eta)| \\
&\leq \bar{\lambda}B_1|\xi - \eta| + h\delta|\xi - \eta| \\
&= K_2|\xi - \eta|
\end{aligned}$$

as desired. Finally,

$$\begin{aligned}
|w_g(\xi) - w_q(\xi)| &= |\bar{\lambda} f(\xi) + hg(\xi) - \bar{\lambda} f(\xi) - hq(\xi)| \\
&= h|g(\xi) - q(\xi)|
\end{aligned}$$

as desired. We now turn to the bounds for $I_g^{(1)}, I_q^{(1)}$,

$$\begin{aligned}|I_g^{(1)}(\xi)| &\leq \int_0^1 |Df(\xi + shaw_g(\xi))||w_g(\xi)|ds \\ &\leq \int_0^1 B_1K_1 ds \\ &= B_1K_1,\end{aligned}$$

establishing the first bound. For the second bound,

$$\begin{aligned}|I_g^{(1)}(\xi) - I_g^{(1)}(\eta)| \leq &\int_0^1 |Df(\xi + shaw_g(\xi))w_g(\xi) - Df(\eta + shaw_g(\eta))w_g(\xi)|ds \\ &+ \int_0^1 |Df(\eta + shaw_g(\eta))w_g(\xi) - Df(\eta + shaw_g(\eta))w_g(\eta)|ds \\ \leq\ & K_1B_2 \int_0^1 (|\xi - \eta| + sha|w_g(\xi) - w_g(\eta)|)ds + B_1|w_g(\xi) - w_g(\eta)| \\ \leq\ & K_1B_2(|\xi - \eta| + haK_2|\xi - \eta|) + B_1K_2|\xi - \eta| \\ =\ & (B_1K_2 + B_2K_1(1 + haK_2))|\xi - \eta|\end{aligned}$$

as desired. Finally

$$\begin{aligned}|I_g^{(1)}(\xi) - I_q^{(1)}(\xi)| \leq &\int_0^1 |Df(\xi + shaw_g(\xi))w_g(\xi) - Df(\xi + shaw_g(\xi))w_q(\xi)|ds \\ &+ \int_0^1 |Df(\xi + shaw_g(\xi))w_q(\xi) - Df(\xi + shaw_q(\xi))w_q(\xi)|ds \\ \leq\ & B_1 \int_0^1 |w_g(\xi) - w_q(\xi)|ds \\ &+ K_1B_2 \int_0^1 |\xi + shaw_g(\xi) - \xi - shaw_q(\xi)|ds \\ \leq\ & hB_1|g(\xi) - q(\xi)| + h^2aB_2K_1|g(\xi) - q(\xi)| \\ =\ & h(B_1 + haB_2K_1)|g(\xi) - q(\xi)|\end{aligned}$$

as desired. The bounds for $z_g, z_q, I_g^{(2)}$, and $I_q^{(2)}$ follow similarly. □

We also need the following three lemmas.

Lemma 18. *Suppose Assumption 16 holds. For any $g \in \Gamma$ and $p \in \mathbb{R}^d$ there exists a unique $\xi \in \mathbb{R}^d$ satisfying* (3.29).

Proof. Consider the iteration of the form

$$\xi^{k+1} = p - hz_g(\xi^k).$$

For any two sequences $\{\xi^k\}$, $\{\eta^k\}$ generated by this iteration we have, by Lemma 17,

$$\begin{aligned}|\xi^{k+1} - \eta^{k+1}| &\le h|z_g(\eta^k) - z_g(\xi^k)| \\ &\le h(\lambda K_2 + B_1(1 + haK_2))|\xi^k - \eta^k| \\ &= c|\xi^k - \eta^k|\end{aligned}$$

which is a contraction by (3.41). $\square$

Lemma 19. *Suppose Assumption 16 holds. The operator T defined by* (3.30) *satisfies* $T : \Gamma \to \Gamma$.

Proof. Let $g \in \Gamma$ and $p \in \mathbb{R}^d$ then by Lemma 18 there is a unique $\xi \in \mathbb{R}^d$ such that (3.29) is satisfied. Then

$$\begin{aligned}|(Tg)(p)| &\le \lambda|g(\xi)| + a|I_g^{(1)}(\xi)| + \tilde{\lambda}|I_g^{(2)}(\xi)| \\ &\le \lambda\gamma + aB_1(\tilde{\lambda}B_0 + h\gamma) + \tilde{\lambda}B_1(\lambda(\tilde{\lambda}B_0 + h\gamma) + B_0) \\ &= (\lambda + hB_1(a + \lambda\tilde{\lambda}))\gamma + \tilde{\lambda}B_0B_1(a + \tilde{\lambda}) \\ &\le \gamma\end{aligned}$$

with the last inequality following from (3.38).

Let $p_1, p_2 \in \mathbb{R}^d$ then, by Lemma 18, there exist $\xi_1, \xi_2 \in \mathbb{R}^d$ such that (3.29) is satisfied with $p = \{p_1, p_2\}$. Hence, by Lemma 17,

$$\begin{aligned}|(Tg)(p_1) - (Tg)(p_2)| &\le \lambda|g(\xi_1) - g(\xi_2)| + a|I_g^{(1)}(\xi_1) - I_g^{(1)}(\xi_2)| + \tilde{\lambda}|I_g^{(2)}(\xi_1) - I_g^{(2)}(\xi_2)| \\ &\le K|\xi_1 - \xi_2|\end{aligned}$$

where we define

$$K := \lambda\delta + a(B_1K_2 + B_2K_1(1 + haK_2)) + \tilde{\lambda}((B_1 + hB_2K_3)(\lambda K_2 + B_1(1 + haK_2)) + B_2K_3).$$

Now, using (3.29) and the proof of Lemma 18,

$$\begin{aligned}|\xi_1 - \xi_2| &\le |p_1 - p_2| + h|z_g(\xi_1) - z_g(\xi_2)| \\ &\le |p_1 - p_2| + c|\xi_1 - \xi_2|.\end{aligned}$$

Since $c < 1$ by (3.41), we obtain

$$|\xi_1 - \xi_2| \le \frac{1}{1 - c}|p_1 - p_2|$$

thus

$$|(Tg)(p_1) - (Tg)(p_2)| \le \frac{K}{1-c}|p_1 - p_2| \le \delta|p_1 - p_2|.$$

To see the last inequality, we note that

$$\frac{K}{1-c} \le \delta \iff K - \delta(1-c) \le 0$$

and $K - \delta(1-c) = \alpha_2\delta^2 + \alpha_1\delta + \alpha_0$ by (3.39), and thus (3.40) gives the desired result. □

Lemma 20. *Suppose Assumption 16 holds. For any $g_1, g_2 \in \Gamma$, we have*

$$\|Tg_1 - Tg_2\|_\Gamma \le \mu\|g_1 - g_2\|_\Gamma$$

where $\mu < 1$.

Proof. By Lemma 18, for any $p \in \mathbb{R}^d$ and $g_1, g_2 \in \Gamma$, there are $\xi_1, \xi_2 \in \mathbb{R}^d$ such that

$$\begin{aligned} p &= \xi_j + hz_{g_j}(\xi_j) \\ (Tg_j)(p) &= \lambda g_j(\xi_j) + aI^{(1)}_{g_j}(\xi_j) - \tilde{\lambda} I^{(2)}_{g_j}(\xi_j) \end{aligned}$$

for $j = 1, 2$. Then

$$|(Tg_1)(p) - (Tg_2)(p)| \le \lambda|g_1(\xi_1) - g_2(\xi_2)| + a|I^{(1)}_{g_1}(\xi_1) - I^{(1)}_{g_2}(\xi_2)| + \tilde{\lambda}|I^{(2)}_{g_1}(\xi_1) - I^{(2)}_{g_2}(\xi_2)|.$$

Note that

$$\begin{aligned} |g_1(\xi_1) - g_2(\xi_2)| &= |g_1(\xi_1) - g_2(\xi_2) - g_2(\xi_1) + g_2(\xi_1)| \\ &\le |g_1(\xi_1) - g_2(\xi_1)| + \delta|\xi_1 - \xi_2|. \end{aligned}$$

Similarly, by Lemma 17,

$$\begin{aligned} |I^{(1)}_{g_1}(\xi_1) - I^{(1)}_{g_2}(\xi_2)| &= |I^{(1)}_{g_1}(\xi_1) - I^{(1)}_{g_2}(\xi_2) - I^{(1)}_{g_2}(\xi_1) + I^{(1)}_{g_2}(\xi_1)| \\ &\le |I^{(1)}_{g_1}(\xi_1) - I^{(1)}_{g_2}(\xi_1)| + |I^{(1)}_{g_2}(\xi_1) - I^{(1)}_{g_2}(\xi_2)| \\ &\le h(B_1 + haB_2K_1)|g_1(\xi_1) - g_2(\xi_1)| \\ &\quad + (B_1K_2 + B_2K_1(1 + haK_2))|\xi_1 - \xi_2|. \end{aligned}$$

Finally,

$$\begin{aligned} |I^{(2)}_{g_1}(\xi_1) - I^{(2)}_{g_2}(\xi_2)| &= |I^{(2)}_{g_1}(\xi_1) - I^{(2)}_{g_2}(\xi_2) - I^{(2)}_{g_2}(\xi_1) + I^{(2)}_{g_2}(\xi_1)| \\ &\le |I^{(2)}_{g_1}(\xi_1) - I^{(2)}_{g_2}(\xi_1)| + |I^{(2)}_{g_2}(\xi_1) - I^{(2)}_{g_2}(\xi_2)| \\ &\le h(\lambda + hB_1a)(B_1 + hB_2K_3)|g_1(\xi_1) - g_2(\xi_1)| + \\ &\quad + ((B_1 + hB_2K_3)(\lambda K_2 + B_1(1 + haK_2)) + B_2K_3)|\xi_1 - \xi_2|. \end{aligned}$$

Putting these together and using (3.42), we obtain

$$|(Tg_1)(p) - (Tg_2)(p)| \leq (\lambda + Q_2)|g_1(\xi_1) - g_2(\xi_1)| + Q_1|\xi_1 - \xi_2|.$$

Now, by Lemma 17,

$$\begin{aligned}
|\xi_1 - \xi_2| &\leq h|z_{g_1}(\xi_1) - z_{g_2}(\xi_2) - z_{g_2}(\xi_1) + z_{g_2}(\xi_1)| \\
&\leq h(|z_{g_1}(\xi_1) - z_{g_2}(\xi_1)| + |z_{g_2}(\xi_1) - z_{g_2}(\xi_2)|) \\
&\leq h^2(\lambda + haB_1)|g_1(\xi) - g_2(\xi_1)| + h(\lambda K_2 + B_1(1 + haK_2))|\xi_1 - \xi_2| \\
&= h^2(\lambda + haB_1)|g_1(\xi) - g_2(\xi_1)| + Q_3|\xi_1 - \xi_2|
\end{aligned}$$

using (3.42). Since, by (3.43), $Q_3 < 1$, we obtain

$$|\xi_1 - \xi_2| \leq \frac{h^2(\lambda + haB_1)}{1 - Q_3}|g_1(\xi_1) - g_2(\xi_1)|$$

and thus

$$\begin{aligned}
|(Tg_1)(p) - (Tg_2)(p)| &\leq \left(\lambda + Q_2 + \frac{h^2(\lambda + haB_1)Q_1}{1 - Q_3}\right)|g_1(\xi_1) - g_2(\xi_1)| \\
&= \mu|g_1(\xi_1) - g_2(\xi_1)|
\end{aligned}$$

by (3.42). Taking the supremum over ξ_1 then over p gives the desired result. Since $\mu < 1$ by (3.43), we obtain that T is a contraction on Γ. □

3.10 Derivation of Split-step Method

We consider the equation

$$\begin{aligned}
&\ddot{u} + 2\sqrt{\mu}\dot{u} + \nabla\Phi(u) = 0 \\
&u(0) = \mathsf{u}_0, \quad \dot{u}(0) = \mathsf{v}_0.
\end{aligned}$$

Set $v = \dot{u}$ then we have

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \\ -2\sqrt{\mu}v - \nabla\Phi(u) \end{bmatrix}.$$

Define the maps

$$f_1(u, v) := \begin{bmatrix} v \\ -2\sqrt{\mu}v \end{bmatrix}, \quad f_2(u, v) := \begin{bmatrix} 0 \\ -\nabla\Phi(u) \end{bmatrix}$$

and then

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = f_1(u, v) + f_2(u, v).$$

We first solve the system

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = f_1(u, v).$$

Clearly

$$v(t) = e^{-2\sqrt{\mu}t}\mathsf{v}_0,$$

hence

$$\begin{aligned} u(t) &= \mathsf{u}_0 + \int_0^t e^{-2\sqrt{\mu}s}\mathsf{v}_0 \, ds \\ &= \mathsf{u}_0 + \frac{1}{2\sqrt{\mu}}\left(1 - e^{-2\sqrt{\mu}t}\right)\mathsf{v}_0. \end{aligned}$$

This gives us the flow map

$$\psi_1(\mathsf{u}, \mathsf{v}; t) = \begin{bmatrix} \mathsf{u} + \frac{1}{2\sqrt{\mu}}\left(1 - e^{-2\sqrt{\mu}t}\right)\mathsf{v} \\ e^{-2\sqrt{\mu}t}\mathsf{v} \end{bmatrix}.$$

We now solve the system

$$\begin{bmatrix} \dot{u} \\ \dot{v} \end{bmatrix} = f_2(u, v).$$

Clearly

$$u(t) = \mathsf{u}_0,$$

hence

$$v(t) = \mathsf{v}_0 - t\nabla\Phi(\mathsf{u}_0).$$

This gives us the flow map

$$\psi_2(\mathsf{u}, \mathsf{v}; t) = \begin{bmatrix} \mathsf{u} \\ \mathsf{v} - t\nabla\Phi(\mathsf{u}) \end{bmatrix}.$$

The composition of the flow maps is then

$$(\psi_2 \circ \psi_1)(\mathsf{u}, \mathsf{v}; t) = \begin{bmatrix} \mathsf{u} + \frac{1}{2\sqrt{\mu}}\left(1 - e^{-2\sqrt{\mu}t}\right)\mathsf{v} \\ e^{-2\sqrt{\mu}t}\mathsf{v} - t\nabla\Phi\left(\mathsf{u} + \frac{1}{2\sqrt{\mu}}\left(1 - e^{-2\sqrt{\mu}t}\right)\mathsf{v}\right) \end{bmatrix}.$$

Mapping t to the time-step $\sqrt{h}$ gives the numerical method (3.36).

References

Bertsimas, Dimitris, John Tsitsiklis, et al. (1993). "Simulated annealing". In: *Statistical science* 8.1, pp. 10–15.

Betancourt, Michael, Michael I. Jordan, and Ashia C. Wilson (2018). *On Symplectic Optimization*. arXiv: `1802.03653 [stat.CO]`.

Carr, Jack (2012). *Applications of centre manifold theory*. Vol. 35. Springer Science & Business Media.

Chartier, Philippe, Ernst Hairer, and Gilles Vilmart (2007). "Numerical integrators based on modified differential equations". In: *Mathematics of computation* 76.260, pp. 1941–1953.

Dieuleveut, Aymeric, Alain Durmus, and Francis Bach (2017). "Bridging the gap between constant step size stochastic gradient descent and markov chains". In: *arXiv preprint arXiv:1707.06386*.

Duchi, John, Elad Hazan, and Yoram Singer (July 2011). "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *J. Mach. Learn. Res.* 12, pp. 2121–2159. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

Farazmand, Mohammad (2018). "Multiscale analysis of accelerated gradient methods". In: *arXiv:1807.11354*.

– (2020). "Multiscale analysis of accelerated gradient methods". In: *SIAM Journal on Optimization* 30.3, pp. 2337–2354.

Feng, Yuanyuan, Lei Li, and Jian-Guo Liu (2018). "Semigroups of stochastic gradient descent and online principal component analysis: properties and diffusion approximations". In: *Communications in Mathematical Sciences* 16.3, pp. 777–789.

Gadat, Sébastien, Fabien Panloup, Sofiane Saadane, et al. (2018). "Stochastic heavy ball". In: *Electronic Journal of Statistics* 12.1, pp. 461–529.

Geman, Stuart and Donald Geman (1987). "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images". In: *Readings in computer vision*. Elsevier, pp. 564–584.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Griffiths, DF and JM Sanz-Serna (1986). "On the scope of the method of modified equations". In: *SIAM Journal on Scientific and Statistical Computing* 7.3, pp. 994–1008.

He, Kaiming et al. (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hinton, Geoffrey and Ruslan Salakhutdinov (2006). “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786, pp. 504–507.

Hirsch, Morris W, Charles Chapman Pugh, and Michael Shub (2006). *Invariant manifolds*. Vol. 583. Springer.

Hu, Bin and Laurent Lessard (2017). “Dissipativity theory for Nesterov’s accelerated method”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 1549–1557.

Kingma, Diederik P. and Jimmy Ba (2014). “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980. arXiv: `1412.6980`. URL: `http://arxiv.org/abs/1412.6980`.

Kovachki, Nikola B. and Andrew M. Stuart (2021). “Continuous Time Analysis of Momentum Methods”. In: *Journal of Machine Learning Research* 22.17, pp. 1–40. URL: `http://jmlr.org/papers/v22/19-466.html`.

Kushner, Harold J (1987). “Asymptotic global behavior for stochastic approximation and diffusions with slowly decreasing noise effects: global minimization via Monte Carlo”. In: *SIAM Journal on Applied Mathematics* 47.1, pp. 169–185.

Kushner, Harold Joseph and Dean S Clark (2012). *Stochastic approximation methods for constrained and unconstrained systems*. Vol. 26. Springer Science & Business Media.

LeCun, Yann, Yoshua Bengio, and Geoffrey E. Hinton (2015). “Deep learning”. In: *Nature* 521.7553, pp. 436–444. DOI: `10.1038/nature14539`. URL: `https://doi.org/10.1038/nature14539`.

LeCun, Yann and Corinna Cortes (2010). “MNIST handwritten digit database”. In: URL: `http://yann.lecun.com/exdb/mnist/`.

Lessard, Laurent, Benjamin Recht, and Andrew Packard (2016). “Analysis and design of optimization algorithms via integral quadratic constraints”. In: *SIAM Journal on Optimization* 26.1, pp. 57–95.

Li, Qianxiao, Cheng Tai, and Weinan E (June 2017). “Stochastic Modified Equations and Adaptive Stochastic Gradient Algorithms”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. International Convention Centre, Sydney, Australia: PMLR, pp. 2101–2110.

Loizou, Nicolas and Peter Richtárik (2017). “Linearly convergent stochastic heavy ball method for minimizing generalization error”. In: *arXiv preprint arXiv:1710.10737*.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Mattingly, Jonathan C, Andrew M Stuart, and Michael V Tretyakov (2010). “Convergence of numerical time-averaging and stationary measures via Poisson equations”. In: *SIAM Journal on Numerical Analysis* 48.2, pp. 552–577.

Nesterov, Yurii (1983). “A method of solving a convex programming problem with convergence rate O(1/k2)”. In: *Soviet Mathematics Doklady* 27.2, pp. 372–376.

– (2014). *Introductory Lectures on Convex Optimization: A Basic Course*. 1st ed. Springer Publishing Company, Incorporated. ISBN: 1461346916, 9781461346913.

Paszke, Adam et al. (2017). “Automatic differentiation in PyTorch”. In: *NIPS-W*.

Pavliotis, Grigorios and Andrew Stuart (Jan. 2008). *Multiscale Methods: Averaging and Homogenization*. Vol. 53. DOI: 10.1007/978-0-387-73829-1.

Polyak, Boris (Dec. 1964). “Some methods of speeding up the convergence of iteration methods”. In: *Ussr Computational Mathematics and Mathematical Physics* 4, pp. 1–17. DOI: 10.1016/0041-5553(64)90137-5.

Polyak, Boris T. (1987). *Introduction to optimization*. New York: Optimization Software, Inc.

Qian, Ning (Jan. 1999). “On the Momentum Term in Gradient Descent Learning Algorithms”. In: *Neural Netw.* 12.1, pp. 145–151. ISSN: 0893-6080. DOI: 10.1016/S0893-6080(98)00116-6. URL: http://dx.doi.org/10.1016/S0893-6080(98)00116-6.

Robbins, Herbert and Sutton Monro (Sept. 1951). “A Stochastic Approximation Method”. In: *Ann. Math. Statist.* 22.3, pp. 400–407. DOI: 10.1214/aoms/1177729586. URL: https://doi.org/10.1214/aoms/1177729586.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). “Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1”. In: ed. by David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press. Chap. Learning Internal Representations by Error Propagation, pp. 318–362. ISBN: 0-262-68053-X. URL: http://dl.acm.org/citation.cfm?id=104279.104293.

Scieur, Damien et al. (2017). “Integration methods and optimization algorithms”. In: *Advances in Neural Information Processing Systems*, pp. 1109–1118.

Shi, Bin et al. (2018). “Understanding the acceleration phenomenon via high-resolution differential equations”. In: *arXiv preprint arXiv:1810.08907*.

Stuart, Andrew and Anthony R Humphries (1998). *Dynamical systems and numerical analysis*. Vol. 2. Cambridge University Press.

Styblinski, MA and T-S Tang (1990). “Experiments in nonconvex optimization: stochastic approximation with function smoothing and simulated annealing”. In: *Neural Networks* 3.4, pp. 467–483.

Su, Weijie, Stephen Boyd, and Emmanuel Candes (2014). “A Differential Equation for Modeling Nesterov’s Accelerated Gradient Method: Theory and Insights”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 2510–2518. URL: `http://papers.nips.cc/paper/5322-a-differential-equation-for-modeling-nesterovs-accelerated-gradient-method-theory-and-insights.pdf`.

Sutskever, Ilya et al. (2013). “On the Importance of Initialization and Momentum in Deep Learning”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA: JMLR.org, pp. III-1139–III-1147. URL: `http://dl.acm.org/citation.cfm?id=3042817.3043064`.

Tieleman, T. and G. Hinton (2012). *Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude*. COURSERA: Neural Networks for Machine Learning.

Wibisono, Andre, Ashia C Wilson, and Michael I Jordan (2016). “A variational perspective on accelerated methods in optimization”. In: *proceedings of the National Academy of Sciences* 113.47, E7351–E7358.

Wiggins, Stephen (2013). *Normally hyperbolic invariant manifolds in dynamical systems*. Vol. 105. Springer Science & Business Media.

Wilson, Ashia C., Benjamin Recht, and Michael I. Jordan (2016). “A Lyapunov Analysis of Momentum Methods in Optimization”. In: *CoRR* abs/1611.02635. arXiv: `1611.02635`. URL: `http://arxiv.org/abs/1611.02635`.

Wilson, Ashia C et al. (2017). “The Marginal Value of Adaptive Gradient Methods in Machine Learning”. In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 4148–4158. URL: `http://papers.nips.cc/paper/7003-the-marginal-value-of-adaptive-gradient-methods-in-machine-learning.pdf`.

Yang, Tianbao, Qihang Lin, and Zhe Li (2016). “Unified convergence analysis of stochastic momentum methods for convex and non-convex optimization”. In: *arXiv preprint arXiv:1604.03257*.

Zhang, Jingzhao et al. (2018). “Direct Runge-Kutta Discretization Achieves Acceleration”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS’18. Montréal, Canada: Curran Associates Inc., pp. 3904–3913.

Chapter 4

NEURAL NETWORKS AND MODEL REDUCTION FOR PARAMETRIC PDE(S)

4.1 Introduction

At the core of many computational tasks arising in science and engineering is the problem of repeatedly evaluating the output of an expensive forward model for many statistically similar inputs. Such settings include the numerical solution of parametric partial differential equations (PDEs), time-stepping for evolutionary PDEs and, more generally, the evaluation of input-output maps defined by black-box computer models. The key idea in this paper is the development of a new data-driven emulator which is defined to act between the infinite-dimensional input and output spaces of maps such as those defined by PDEs. By defining approximation architectures on infinite-dimensional spaces, we provide the basis for a methodology which is robust to the resolution of the finite-dimensionalizations used to create implementable algorithms.

This work is motivated by the recent empirical success of neural networks in machine learning applications such as image classification, aiming to explore whether this success has any implications for algorithm development in different applications arising in science and engineering. We further wish to compare the resulting new methods with traditional algorithms from the field of numerical analysis for the approximation of infinite-dimensional maps, such as the maps defined by parametric PDEs or the solution operator for time-dependent PDEs. We propose a method for approximation of such solution maps purely in a data-driven fashion by lifting the concept of neural networks to produce maps acting between infinite-dimensional spaces. Our method exploits approximate finite-dimensional structure in maps between Banach spaces of functions through three separate steps: (i) reducing the dimension of the input, (ii) reducing the dimension of the output, and (iii) finding a map between the two resulting finite-dimensional latent spaces. Our approach takes advantage of the approximation power of neural networks while allowing for the use of well-understood, classical dimension reduction (and reconstruction) techniques. Our goal is to reduce the complexity of the input-to-output map by replacing it with a data-driven emulator. In achieving this goal we design an emulator which enjoys mesh-independent approximation properties, a fact which we establish through a combination of theory and numerical experiments; to the best of our knowledge, these are the first such results in the area of neural networks for PDE problems.

To be concrete, and to guide the literature review which follows, consider the following prototypical parametric PDE

$$(\mathcal{P}_x y)(s) = 0, \qquad \forall s \in D,$$

where $D \subset \mathbb{R}^d$ is a bounded open set, $\mathcal{P}_x$ is a differential operator depending on a parameter $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ is the solution to the PDE (given appropriate boundary conditions). The Banach spaces $\mathcal{X}$ and $\mathcal{Y}$ are assumed to be spaces of real-valued functions on D. Here, and in the rest of this paper, we consistently use s to denote the independent variable in spatially dependent PDEs, and reserve x and y for the input and output of the PDE model of interest. We adopt this idiosyncratic notation (from the PDE perspective) to keep our exposition in line with standard machine learning notation for input and output variables.

Example 21. *Consider second order elliptic PDEs of the form*

$$\begin{aligned} -\nabla \cdot (a(s)\nabla u(s)) &= f(s), \quad & s \in D \\ u(s) &= 0, \quad & s \in \partial D \end{aligned} \tag{4.1}$$

which are prototypical of many scientific applications. As a concrete example of a mapping defined by this equation, we restrict ourselves to the setting where the forcing term f is fixed, and consider the diffusion coefficient a as the input parameter x and the PDE solution u as output y. In this setting, we have $\mathcal{X} = L^\infty(D; \mathbb{R}_+)$, $\mathcal{Y} = H_0^1(D; \mathbb{R})$, and $\mathcal{P}_x = -\nabla_s \cdot (a\nabla_s \cdot) - f$, equipped with homogeneous Dirichlet boundary conditions. This is the Darcy flow problem which we consider numerically in Section 4.4.

Literature Review

The recent success of neural networks on a variety of high-dimensional machine learning problems (LeCun, Bengio, and Hinton, 2015) has led to a rapidly growing body of research pertaining to applications in scientific problems (Adler and Oktem, 2017; Bhatnagar et al., 2019; Cheng et al., 2019; E and Yu, 2018; Gilmer et al., 2017; Holland, Baeder, and Duraisamy, 2019; Raissi, Perdikaris, and George E Karniadakis, 2019; Zhu and Zabaras, 2018; Smith, Azizzadenesheli, and Ross, 2020). In particular, there is a substantial number of articles which investigate the use of neural networks as surrogate models, and more specifically for obtaining the solution of (possibly parametric) PDEs.

We summarize the two most prevalent existing neural network based strategies in the approximation of PDEs in general, and parametric PDEs specifically. The first approach can be thought of as image-to-image regression. The goal is to approximate the parametric solution operator mapping elements of $\mathcal{X}$ to $\mathcal{Y}$. This is achieved by discretizing both spaces to obtain finite-dimensional input and output spaces of dimension K. We assume to have access to data in the form of observations of input x and output y discretized on K-points within the domain D. The methodology then proceeds by defining a neural network $F : \mathbb{R}^K \to \mathbb{R}^K$ and regresses the input-to-output map by minimizing a misfit functional defined using the point values of x and y on the discretization grid. The articles (Adler and Oktem, 2017; Bhatnagar et al., 2019; Holland, Baeder, and Duraisamy, 2019; Zhu and Zabaras, 2018; Geist et al., 2020) apply this methodology for various forward and inverse problems in physics and engineering, utilizing a variety of neural network architectures in the regression step; the related paper (Khoo, J. Lu, and Ying, 2017) applies a similar approach, but the output space is $\mathbb{R}$. This innovative set of papers demonstrate some success. However, from the perspective of the goals of our work, their approaches are not robust to mesh-refinement: the neural network is defined as a mapping between two Euclidean spaces of values on mesh points. The rates of approximation depend on the underlying discretization and an overhaul of the architecture would be required to produce results consistent across different discretizations. The papers (L. Lu, Jin, and George Em Karniadakis, 2019; L. Lu, Jin, Pang, et al., 2020) make a conceptual step in the direction of interest to us in this paper, as they introduce an architecture based on a neural network approximation theorem for operators from (T. Chen and H. Chen, 1995); but as implemented the method still results in parameters which depend on the mesh used. Applications of this methodology may be found in (Cai et al., 2020; Mao et al., 2020; C. Lin et al., 2020).

The second approach does not directly seek to find the parametric map from $\mathcal{X}$ to $\mathcal{Y}$ but rather is thought of, for fixed $x \in \mathcal{X}$, as being a parametrization of the solution $y \in \mathcal{Y}$ by means of a deep neural network (Dockhorn, 2019; E and Yu, 2018; Hsieh et al., 2019; Lagaris, Likas, and Fotiadis, 1998; Raissi, Perdikaris, and George E Karniadakis, 2019; Shin, Darbon, and George Em Karniadakis, 2020). This methodology parallels collocation methods for the numerical solution of PDEs by searching over approximation spaces defined by neural networks. The solution of the PDE is written as a neural network approximation in which the spatial (or, in the time-dependent case, spatio-temporal) variables in D are inputs and the solution is the output. This parametric function is then substituted into the PDE and the residual is made small by optimization. The resulting neural network may be thought of as a novel structure which composes the action of the operator $\mathcal{P}_x$, for fixed x, with a neural network taking inputs in D (Raissi, Perdikaris, and George E Karniadakis, 2019). While this method leads to an approximate solution map defined on the input domain D (and not on a $K-$point discretization of the domain), the parametric dependence of the approximate solution map is fixed. Indeed for a new input parameter x, one needs to re-train the neural network by solving the associated optimization problem in order to produce a new map $y : D \to \mathbb{R}$; this may be prohibitively expensive when parametric dependence of the solution is the target of analysis. Furthermore the approach cannot be made fully data-driven as it needs knowledge of the underlying PDE, and furthermore the operations required to apply the differential operator may interact poorly with the neural network approximator during the back-propagation (adjoint calculation) phase of the optimization.

The work (Ruthotto and Haber, 2019) examines the forward propagation of neural networks as the flow of a time-dependent PDE, combining the continuous time formulation of ResNet (Haber and Ruthotto, 2017; Weinan, 2017) with the idea of neural networks acting on spaces of functions: by considering the initial condition as a function, this flow map may be thought of as a neural network acting between infinite-dimensional spaces. The idea of learning PDEs from data using neural networks, again generating a flow map between infinite dimensional spaces, was studied in the 1990s in the papers (Krischer et al., 1993; Gonzalez-Garcia, Rico-Martinez, and Kevrekidis, 1998) with the former using a PCA methodology, and the latter using the method of lines. More recently the works (J. Hesthaven and Ubbiali, 2018; Wang, Jan S. Hesthaven, and Ray, 2019) also employ a PCA methodology for the output space but only consider very low dimensional input spaces. Furthermore the works (Lee and Carlberg, 2020; Fresca, Dede, and Manzoni, 2020; Gonzalez and Balajewicz, 2018; Fresca, Dede, and Manzoni, 2020) proposed a model reduction approach for dynamical systems by use of dimension reducing neural networks (autoencoders). However only a fixed discretization of space is considered, yielding a method which does not produce a map between two infinite-dimensional spaces.

The development of numerical methods for parametric problems is not, of course, restricted to the use of neural networks. Earlier works in the engineering literature started in the 1970s focused on computational methods which represent PDE solutions in terms of known basis functions that contain information about the solution structure (Almroth, Stern, and Brogan, 1978; Nagy, 1979). This work led to the development of the reduced basis method (RBM) which is widely adopted in engineering; see (Barrault et al., 2004; Jan S Hesthaven, Rozza, Stamm, et al., 2016; Quarteroni, Manzoni, and Negri, 2015) and the references therein. The methodology was also used for stochastic problems, in which the input space $\mathcal{X}$ is endowed with a probabilistic structure, in (Boyaval et al., 2010). The study of RBMs led to broader interest in the approximation theory community focusing on rates of convergence for the RBM approximation of maps between Banach spaces, and in particular maps defined through parametric dependence of PDEs; see (R. A. DeVore, 2014) for an overview of this work.

Ideas from model reduction have been combined with data-driven learning in the sequence of papers (Peherstorfer and Willcox, 2016; McQuarrie, Huang, and Willcox, 2020; Benner et al., 2020; Peherstorfer, 2019; Qian et al., 2020). The setting is the learning of data-driven approximations to time-dependent PDEs. Model reduction is used to find a low-dimensional approximation space and then a system of ordinary differential equations (ODEs) is learned in this low-dimensional latent space. These ODEs are assumed to have vector fields from a known class with unknown linear coefficients; learning is thus reduced to a least squares problem. The known vector fields mimic properties of the original PDE (for example are restricted to linear and quadratic terms for the equations of geophysical fluid dynamics); additionally transformations may be used to render the original PDE in a desirable form (such as having only quadratic nonlinearities.)

The development of theoretical analyses to understand the use of neural networks to approximate PDEs is currently in its infancy, but interesting results are starting to emerge (Herrmann, Ch Schwab, and Zech, 2020; Kutyniok et al., 2019; Christoph Schwab and Jakob Zech, 2019; Laakmann and Petersen, 2020). A recurrent theme in the analysis of neural networks, and in these papers in particular, is that the work typically asserts the *existence* of a choice of neural network parameters which achieve a certain approximation property; because of the non-convex optimization techniques used to determine the network parameters, the issue of *finding* these parameters in practice is rarely addressed. Recent works take a different perspective on data-driven approximation of PDEs, motivated by small-data scenarios; see the paper (Albert Cohen, Dahmen, and Ron DeVore, 2020) which relates, in part, to earlier work focused on the small-data setting (Binev et al., 2017; Maday et al., 2015). These approaches are more akin to data assimilation (Reich and Cotter, 2015; Law, Andrew Stuart, and Zygalakis, 2015) where the data is incorporated into a model.

Our Contribution

The primary contributions of this paper are as follows:

1. we propose a novel data-driven methodology capable of learning mappings between Hilbert spaces;
2. the proposed method combines model reduction with neural networks to obtain algorithms with controllable approximation errors as maps between Hilbert spaces;
3. as a result of this approximation property of maps between Hilbert spaces, the learned maps exhibit desirable mesh-independence properties;
4. we prove that our architecture is sufficiently rich to contain approximations of arbitrary accuracy, as a mapping between function spaces;
5. we present numerical experiments that demonstrate the efficacy of the proposed methodology, demonstrate desirable mesh-indepence properties, elucidate its properties beyond the confines of the theory, and compare with other methods for parametric PDEs.

Section 4.2 outlines the approximation methodology, which is based on use of *principal component analysis (PCA)* in a Hilbert space to finite-dimensionalize the input and output spaces, and a neural network between the resulting finite-dimensional spaces. Section 4.3 contains statement and proof of our main approximation result, which invokes a global Lipschitz assumption on the map to be approximated. In Section 4.4 we present our numerical experiments, some of which relax the global Lipschitz assumption, and others which involve comparisons with other approaches from the literature. Section 4.5 contains concluding remarks, including directions for further study. We also include auxiliary results in the appendix that complement and extend the main theoretical developments of the article. Appendix 4.6 extends the analysis of Section 4.3 from globally Lipschitz maps to locally Lipschitz maps with controlled growth rates. Appendix 4.7 contains supporting lemmas that are used throughout the paper while Appendix 4.8 proves an analyticity result pertaining to the solution map of the Poisson equation that is used in one of the numerical experiments in Section 4.4.

4.2 Proposed Method

Our method combines PCA-based dimension reduction on the input and output spaces $\mathcal{X}, \mathcal{Y}$ with a neural network that maps the dimension-reduced spaces. After a pre-amble in Subsection 4.2, giving an overview of our approach, we continue in Subsection 4.2 with a description of PCA in the Hilbert space setting, including intuition about its approximation quality. Subsection 4.2 gives the background on neural networks needed for this paper, and Subsection 4.2 compares our methodology to existing methods.

Overview

Let $\mathcal{X}$, $\mathcal{Y}$ be separable Hilbert spaces and $\Psi : \mathcal{X} \to \mathcal{Y}$ be some, possibly nonlinear, map. Our goal is to approximate Ψ from a finite collection of evaluations $\{x_j, y_j\}_{j=1}^N$ where $y_j = \Psi(x_j)$. We assume that the x_j are i.i.d. with respect to (w.r.t.) a probability measure μ supported on $\mathcal{X}$. Note that with this notation the output samples y_j are i.i.d. w.r.t. the push-forward measure $\Psi_\sharp\mu$. The approximation of Ψ from the data $\{x_j, y_j\}_{j=1}^N$ that we now develop should be understood as being designed to be accurate with respect to norms defined by integration with respect to the measures μ and $\Psi_\sharp\mu$ on the spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively.

Instead of attempting to directly approximate Ψ, we first try to exploit possible finite-dimensional structure within the measures μ and $\Psi_\sharp\mu$. We accomplish this by approximating the identity mappings $I_\mathcal{X} : \mathcal{X} \to \mathcal{X}$ and $I_\mathcal{Y} : \mathcal{Y} \to \mathcal{Y}$ by a composition of two maps, known as the *encoder* and the *decoder* in the machine learning literature (Hinton and Salakhutdinov, 2006; Goodfellow, Bengio, and Courville, 2016), which have finite-dimensional range and domain, respectively. We will then interpolate between the finite-dimensional outputs of the encoders, usually referred to as the *latent codes*. Our approach is summarized in Figure 41.

Here, $F_\mathcal{X}$ and $F_\mathcal{Y}$ are the encoders for the spaces $\mathcal{X}, \mathcal{Y}$ respectively, whilst $G_\mathcal{X}$ and $G_\mathcal{Y}$ are the decoders, and φ is the map interpolating the latent codes. The intuition behind Figure 41, and, to some extent, the main focus of our analysis, concerns the

$$\begin{tikzcd}
\mathcal{X} \arrow[r, "F_{\mathcal{X}}"] \arrow[d, "\Psi"'] & \mathbb{R}^{d_{\mathcal{X}}} \arrow[r, "G_{\mathcal{X}}"] \arrow[d, "\varphi"'] & \mathcal{X} \arrow[d, "\Psi"'] \\
\mathcal{Y} \arrow[r, "F_{\mathcal{Y}}"] & \mathbb{R}^{d_{\mathcal{Y}}} \arrow[r, "G_{\mathcal{Y}}"] & \mathcal{Y}
\end{tikzcd}$$

Figure 41: A diagram summarizing various maps of interest in our proposed approach for the approximation of input-output maps between infinite-dimensional spaces.

quality of the the approximations

$$G_{\mathcal{X}} \circ F_{\mathcal{X}} \approx I_{\mathcal{X}}, \tag{4.2a}$$

$$G_{\mathcal{Y}} \circ F_{\mathcal{Y}} \approx I_{\mathcal{Y}}, \tag{4.2b}$$

$$G_{\mathcal{Y}} \circ \varphi \circ F_{\mathcal{X}} \approx \Psi. \tag{4.2c}$$

In order to achieve (4.2c) it is natural to choose φ as

$$\varphi := F_{\mathcal{Y}} \circ \Psi \circ G_{\mathcal{X}}; \tag{4.3}$$

then the approximation (4.2c) is limited only by the approximations (4.2a), (4.2b) of the identity maps on $I_{\mathcal{X}}$ and $I_{\mathcal{Y}}$. We further label the approximation in (4.2c) by

$$\Psi_{PCA} := G_{\mathcal{Y}} \circ \varphi \circ F_{\mathcal{X}}, \tag{4.4}$$

since we later choose PCA as our dimension reduction method. We note that Ψ_{PCA} is not used in practical computations since φ is generally unknown. To make it practical we replace φ with a data-driven approximation $\chi \approx \varphi$ obtaining,

$$\Psi_{NN} := G_{\mathcal{Y}} \circ \chi \circ F_{\mathcal{X}}. \tag{4.5}$$

Later we choose χ to be a neural network, hence the choice of the subscript NN. The combination of PCA for the encoding/decoding along with the neural network approximation χ for φ, forms the basis of our computational methodology.

The compositions $G_{\mathcal{X}} \circ F_{\mathcal{X}}$ and $G_{\mathcal{Y}} \circ F_{\mathcal{Y}}$ are commonly referred to as *autoencoders*. There is a large literature on dimension-reduction methods (Pearson, 1901; Schölkopf, Smola, and Müller, 1998; Coifman et al., 2005; Belkin and Niyogi, 2003; Hinton and Salakhutdinov, 2006) both classical and rooted in neural networks. In this work, we will focus on PCA which is perhaps one of the simplest such methods known (Pearson, 1901). We make this choice due to its simplicity of implementation, excellent numerical performance on the problems we study in Section 4.4, and its amenability to analysis. The dimension reduction in the input and output spaces is essential, as it allows for function space algorithms that make use of powerful finite-dimensional approximation methods, such as the neural networks we use here.

Many classical dimension reduction methods may be seen as encoders. But not all are as easily inverted as PCA—often there is no unambiguous, or no efficient, way to obtain the decoder. Whilst neural network based methods such as deep autoencoders (Hinton and Salakhutdinov, 2006) have shown empirical success in finite dimensional applications they currently lack theory and practical implementation in the setting of function spaces, and are therefore not currently suitable in the context of the goals of this paper.

Nonetheless methods other than PCA are likely to be useful within the general goals of high or infinite-dimensional function approximation. Indeed, with PCA, we approximate the solution manifold (image space) of the operator Ψ by the *linear* space defined in equation (4.8). We emphasize however that, usually, Ψ is a nonlinear operator and our approximation succeeds by capturing the induced nonlinear input-output relationship within the latent codes by using a neural network. We will show in Section 4.3 that the approximation error of the linear space to the solution manifold goes to zero as the dimension increases, however, this decay may be very slow (Albert Cohen and Ronald DeVore, 2015; R. A. DeVore, 1998). Therefore, it may be beneficial to construct nonlinear dimension reducing maps such as deep autoencoders on function spaces. We leave this as an interesting direction for future work.

Regarding the approximation of φ by neural networks, we acknowledge that there is considerable scope for the construction of the neural network, within different families and types of networks, and potentially by using other approximators. For our theory and numerics however we will focus on relatively constrained families of such networks, described in the following Subsection 4.2.

PCA On Function Space

Since we will perform PCA on both $\mathcal{X}$ and $\mathcal{Y}$, and since PCA requires a Hilbert space setting, the development here is in a generic real, separable Hilbert space $\mathcal{H}$ with inner-product and norm denoted by $\langle \cdot, \cdot \rangle$ and $\| \cdot \|$ respectively. We let ν denote a probability measure supported on $\mathcal{H}$, and make the assumption of a finite fourth moment: $\mathbb{E}_{u\sim\nu}\|u\|^4 < \infty$. We denote by $\{u_j\}_{j=1}^N$ a finite collection of N i.i.d. draws from ν that will be used as the training data on which PCA is based. Later we apply the PCA methodology in two distinct settings where the space $\mathcal{H}$ is taken to be the input space $\mathcal{X}$ and the data $\{u_j\}$ are the input samples $\{x_j\}$ drawn from the input measure μ, or $\mathcal{H}$ is taken to be the output space $\mathcal{Y}$ and the data $\{u_j\}$ are the corresponding outputs $\{y_j = \Psi(x_j)\}$ drawn from the push-forward measure $\Psi_\sharp\mu$. The following exposition, and the subsequent analysis in Section 4.3, largely follows the works (Blanchard, Bousquet, and Zwald, 2007; J. Shawe-Taylor et al., 2005; John Shawe-Taylor et al., 2002). We will consider the standard version of non-centered PCA, although more sophisticated versions such as kernel PCA have been widely used and analyzed (Schölkopf, Smola, and Müller, 1998) and could be of potential interest within the overall goals of this work. We choose to work in the non-kernelized setting as there is an unequivocal way of producing the decoder.

For any subspace $V \subseteq \mathcal{H}$, denote by $\Pi_V : \mathcal{H} \to V$ the orthogonal projection operator and define the *empirical projection error*,

$$R_N(V) \coloneqq \frac{1}{N}\sum_{j=1}^{N} \|u_j - \Pi_V u_j\|^2. \tag{4.6}$$

PCA consists of projecting the data onto a finite-dimensional subspace of $\mathcal{H}$ for which this error is minimal. To that end, consider the *empirical, non-centered covariance* operator

$$C_N \coloneqq \frac{1}{N}\sum_{j=1}^{N} u_j \otimes u_j \tag{4.7}$$

where $\otimes$ denotes the outer product. It may be shown that C_N is a non-negative, self-adjoint, trace-class operator on $\mathcal{H}$, of rank at most N (Zeidler, 2012). Let $\phi_{1,N}, \dots \phi_{N,N}$ denote the eigenvectors of C_N and $\lambda_{1,N} \geq \lambda_{2,N} \geq \cdots \geq \lambda_{N,N} \geq 0$ its corresponding eigenvalues in decreasing order. Then for any $d \geq 1$ we define the *PCA subspaces*

$$V_{d,N} = \text{span}\{\phi_{1,N}, \phi_{2,N}, \dots, \phi_{d,N}\} \subset \mathcal{H}. \tag{4.8}$$

It is well known (Murphy, 2012, Thm. 12.2.1) that $V_{d,N}$ solves the minimization problem

$$\min_{V \in \mathcal{V}_d} R_N(V),$$

where $\mathcal{V}_d$ denotes the set of all d-dimensional subspaces of $\mathcal{H}$. Furthermore

$$R_N(V_{d,N}) = \sum_{j=d+1}^{N} \lambda_{j,N}, \tag{4.9}$$

hence the approximation is controlled by the rate of decay of the spectrum of C_N.

With this in mind, we define the *PCA encoder* $F_{\mathcal{H}} : \mathcal{H} \to \mathbb{R}^d$ as the mapping from $\mathcal{H}$ to the coefficients of the orthogonal projection onto $V_{d,N}$ namely,

$$F_{\mathcal{H}}(u) = (\langle u, \phi_{1,N}\rangle, \dots, \langle u, \phi_{d,N}\rangle)^T \in \mathbb{R}^d. \tag{4.10}$$

Correspondingly, the PCA decoder $G_{\mathcal{H}} : \mathbb{R}^d \to \mathcal{H}$ constructs an element of $\mathcal{H}$ by taking as its input the coefficients constructed by $F_{\mathcal{H}}$ and forming an expansion in the empirical basis by zero-padding the PCA basis coefficients, that is

$$G_{\mathcal{H}}(s) = \sum_{j=1}^{d} s_j \phi_{j,N} \qquad \forall s \in \mathbb{R}^d. \tag{4.11}$$

In particular,

$$(G_{\mathcal{H}} \circ F_{\mathcal{H}})(u) = \sum_{j=1}^{d} \langle u, \phi_{j,N}\rangle \phi_{j,N}, \quad \text{equivalently} \quad G_{\mathcal{H}} \circ F_{\mathcal{H}} = \sum_{j=1}^{d} \phi_{j,N} \otimes \phi_{j,N}.$$

Hence $G_{\mathcal{H}} \circ F_{\mathcal{H}} = \Pi_{V_{d,N}}$, a d-dimensional approximation to the identity $I_{\mathcal{H}}$.

We will now give a qualitative explanation of this approximation to be made quantitative in Subsection 4.3. It is natural to consider minimizing the infinite data analog of (4.6), namely the *projection error*

$$R(V) := \mathbb{E}_{u\sim\nu}\|u - \Pi_V u\|^2, \tag{4.12}$$

over $\mathcal{V}_d$ for $d \geq 1$. Assuming ν has a finite second moment, there exists a unique, self-adjoint, non-negative, trace-class operator $C : \mathcal{H} \to \mathcal{H}$ termed the *non-centered covariance* such that $\langle v, Cz\rangle = \mathbb{E}_{u\sim\nu}[\langle v, u\rangle\langle z, u\rangle]$, $\forall v, z \in \mathcal{H}$ (see (Baxendale, 1976)). From this, one readily finds the form of C by noting that

$$\langle v, \mathbb{E}_{u\sim\nu}[u \otimes u]z\rangle = \mathbb{E}_{u\sim\nu}[\langle v, (u \otimes u)z\rangle] = \mathbb{E}_{u\sim\nu}[\langle v, u\rangle\langle z, u\rangle], \tag{4.13}$$

implying that $C = \mathbb{E}_{u\sim\nu}[u \otimes u]$. Moreover, it follows that

$$\text{tr } C = \mathbb{E}_{u\sim\nu}[\text{tr } u \otimes u] = \mathbb{E}_{u\sim\nu}\|u\|^2 < \infty.$$

Let $\phi_1, \phi_2, \dots$ denote the eigenvectors of C and $\lambda_1 \geq \lambda_2 \geq \dots$ the corresponding eigenvalues. In the infinite data setting $(N = \infty)$ it is natural to think of C and its first d eigenpairs as known. We then define the *optimal projection space*

$$V_d = \text{span}\{\phi_1, \phi_2, \dots, \phi_d\}. \tag{4.14}$$

It may be verified that V_d solves the minimization problem $\min_{V \in \mathcal{V}_d} R(V)$ and that $R(V_d) = \sum_{j=d+1}^{\infty} \lambda_j$.

With this infinite data perspective in mind observe that PCA makes the approximation $V_{d,N} \approx V_d$ from a finite dataset. The approximation quality of $V_{d,N}$ w.r.t. V_d is related to the approximation quality of ϕ_j by $\phi_{j,N}$ for $j = 1, \dots, N$ and therefore to the approximation quality of C by C_N. Another perspective is via the Karhunen-Loeve Theorem (KL) (Lord, Powell, and Shardlow, 2014). For simplicity, assume that ν is mean zero, then $u \sim \nu$ admits an expansion of the form $u = \sum_{j=1}^{\infty} \sqrt{\lambda_j} \xi_j \phi_j$ where $\{\xi_j\}_{j=1}^{\infty}$ is a sequence of scalar-valued, mean zero, pairwise uncorrelated random variables. We can then truncate this expansion and make the approximations

$$u \approx \sum_{j=1}^{d} \sqrt{\lambda_j} \xi_j \phi_j \approx \sum_{j=1}^{d} \sqrt{\lambda_{j,N}} \xi_j \phi_{j,N},$$

where the first approximation corresponds to using the optimal projection subspace V_d while the second approximation replaces V_d with $V_{d,N}$. Since it holds that $\mathbb{E} C_N = C$, we expect $\lambda_j \approx \lambda_{j,N}$ and $\phi_j \approx \phi_{j,N}$. These discussions suggest that the quality of the PCA approximation is controlled, on average, by the rate of decay of the eigenvalues of C, and the approximation of the eigenstructure of C by that of C_N.

Neural Networks

A neural network is a nonlinear function $\chi : \mathbb{R}^n \to \mathbb{R}$ defined by a sequence of compositions of affine maps with point-wise nonlinearities. In particular,

$$\chi(s) = W_t \sigma(\dots \sigma(W_2 \sigma(W_1 s + b_1) + b_2)) + b_t, \qquad s \in \mathbb{R}^n, \tag{4.15}$$

where $W_1, \dots, W_t$ are *weight matrices* (that are not necessarily square) and $b_1, \dots, b_t$ are vectors, referred to as *biases*. We refer to $t \geq 1$ as the *depth of the neural network*. The function $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ is a monotone, nonlinear *activation function* that is defined from a monoton function $\sigma : \mathbb{R} \to \mathbb{R}$ applied entrywise to any vector in $\mathbb{R}^d$ with $d \geq 1$. Note that in (4.15) the input dimension of σ may vary between layers but regardless of the input dimension the function σ applies the same operations to all entries of the input vector. We primarily consider the Rectified Linear Unit (ReLU) activation functions, i.e.,

$$\sigma(s) := (\max\{0, s_1\}, \max\{0, s_2\}, \dots, \max\{0, s_d\})^T \in \mathbb{R}^d \qquad \forall s \in \mathbb{R}^d. \tag{4.16}$$

The weights and biases constitute the parameters of the network. In this paper we learn these parameters in the following standard way (LeCun, Bengio, and Hinton, 2015): given a set of data $\{x_j, y_j\}_{j=1}^N$ we choose the parameters of χ to solve an appropriate regression problem by minimizing a data-dependent cost functional, using stochastic gradient methods. Neural networks have been demonstrated to constitute an efficient class of regressors and interpolators for high-dimensional problems empirically, but a complete theory of their efficacy is elusive. For an overview of various neural network architectures and their applications, see (Goodfellow, Bengio, and Courville, 2016). For theories concerning their approximation capabilities see (Maiorov and Pinkus, 1999; Yarotsky, 2017; Christoph Schwab and Jakob Zech, 2019; Daubechies et al., 2019; Kutyniok et al., 2019).

For the approximation results given in Section 4.3, we will work with a specific class of neural networks, following (Yarotsky, 2017); we note that other approximation schemes could be used, however, and that we have chosen a proof setting that aligns with, but is not identical to, what we implement in the computations described in Section 4.4. We will fix $\sigma \in C(\mathbb{R};\mathbb{R})$ to be the ReLU function (4.16) and consider the set of neural networks mapping $\mathbb{R}^n$ to $\mathbb{R}$

$$\mathcal{M}(n;t,r) := \left\{ \begin{aligned} &\chi(s) = W_t\sigma(\dots\sigma(W_2\sigma(W_1 s + b_1) + b_2)) + b_t \in \mathbb{R}, \\ &\text{for all } s \in \mathbb{R}^n \text{ and such that } \sum_{k=1}^{t} |W_k|_0 + |b_k|_0 \le r. \end{aligned} \right\}$$

Here $|\cdot|_0$ gives the number of non-zero entries in a matrix so that $r \ge 0$ denotes the number of active weights and biases in the network while $t \ge 1$ is the total number of layers. Moreover, we define the class of stacked neural networks mapping $\mathbb{R}^n$ to $\mathbb{R}^m$:

$$\mathcal{M}(n,m;t,r) := \left\{ \begin{aligned} &\chi(s) = (\chi^{(1)}(s), \dots, \chi^{(m)}(s))^T \in \mathbb{R}^m, \\ &\text{where } \chi^{(j)} \in \mathcal{M}(n; t^{(j)}, r^{(j)}), \text{ with } t^{(j)} \le t, r^{(j)} \le r. \end{aligned} \right\}$$

From this, we build the set of *zero-extended* neural networks

$$\mathcal{M}(n,m;t,r,M) := \left\{ \chi = \left\{ \begin{aligned} \tilde{\chi}(s), &\quad s \in [-M,M]^n \\ 0, &\quad s \notin [-M,M]^n \end{aligned} \right\}, \text{ for some } \tilde{\chi} \in \mathcal{M}(n,m,t,r) \right\},$$

where the new parameter $M > 0$ is the side length of the hypercube in $\mathbb{R}^n$ within which χ can be non-zero. This construction is essential to our approximation as it allows us to handle non-compactness of the latent spaces after PCA dimension reduction.

Comparison to Existing Methods

In the general setting of arbitrary encoders, the formula (4.2c) for the approximation of Ψ yields a complicated map, the representation of which depends on the dimension reduction methods being employed. However, in the setting where PCA is used, a clear representation emerges which we now elucidate in order to highlight similarities and differences between our methodology and existing methods appearing in the literature.

Let $F_{\mathcal{X}} : \mathcal{X} \to \mathbb{R}^{d_{\mathcal{X}}}$ be the PCA encoder w.r.t. the data $\{x_j\}_{j=1}^N$ given by (4.10) and, in particular, let $\phi_{1,N}^{\mathcal{X}}, \dots, \phi_{d_{\mathcal{X}},N}^{\mathcal{X}}$ be the eigenvectors of the resulting empirical covariance. Similarly let $\phi_{1,N}^{\mathcal{Y}}, \dots, \phi_{d_{\mathcal{Y}},N}^{\mathcal{Y}}$ be the eigenvectors of the empirical covariance w.r.t. the data $\{y\}_{j=1}^N$. For the function φ defined in (4.3), or similarly for approximations χ thereof found through the use of neural networks, we denote the components by $\varphi(s) = (\varphi_1(s), \dots, \varphi_{d_{\mathcal{Y}}}(s))$ for any $s \in \mathbb{R}^{d_{\mathcal{X}}}$. Then (4.2c) becomes $\Psi(x) \approx \sum_{j=1}^{d_{\mathcal{Y}}} \alpha_j(x)\phi_{j,N}^{\mathcal{Y}}$ with coefficients

$$\alpha_j(x) = \varphi_j\big(F_{\mathcal{X}}(x)\big) = \varphi_j\big(\langle x, \phi_{1,N}^{\mathcal{X}}\rangle_{\mathcal{X}}, \dots, \langle x, \phi_{d_{\mathcal{X}},N}^{\mathcal{X}}\rangle_{\mathcal{X}}\big), \qquad \forall x \in \mathcal{X}.$$

The solution data $\{y\}_{j=1}^N$ fixes a basis for the output space, and the dependence of $\Psi(x)$ on x is captured solely via the scalar-valued coefficients α_j. This parallels the formulation of the classical reduced basis method (R. A. DeVore, 2014) where the approximation is written as

$$\Psi(x) \approx \sum_{j=1}^{m} \alpha_j(x)\phi_j. \tag{4.17}$$

Many versions of the method exist, but two particularly popular ones are: (i) when $m = N$ and $\phi_j = y_j$; and (ii) when, as is done here, $m = d_{\mathcal{Y}}$ and $\phi_j = \phi_{j,N}^{\mathcal{Y}}$. The latter choice is also referred to as the reduced basis with a proper orthogonal decomposition.

The crucial difference between our method and the RBM is in the formation of the coefficients α_j. In RBM these functions are obtained in an intrusive manner by approximating the PDE within the finite-dimensional reduced basis and as a consequence the method cannot be used in a setting where a PDE relating inputs and outputs is not known, or may not exist. In contrast, our proposed methodology approximates φ by regressing or interpolating the latent representations $\{F_{\mathcal{X}}(x_j), F_{\mathcal{Y}}(y_j)\}_{j=1}^N$. Thus our proposed method makes use of the entire available dataset and does not require explicit knowledge of the underlying PDE mapping, making it a non-intrusive method applicable to black-box models.

The form (4.17) of the approximate solution operator can also be related to the Taylor approximations developed in (Chkifa et al., 2013; Albert Cohen, Ronald DeVore, and Christoph Schwab, 2010) where a particular form of the input x is considered, namely $x = \bar{x} + \sum_{j\geq 1} a_j \tilde{x}_j$ where $\bar{x} \in \mathcal{X}$ is fixed, $\{a_j\}_{j\geq 1} \in \ell^\infty(\mathbb{N};\mathbb{R})$ are uniformly bounded, and $\{\tilde{x}_j\}_{j\geq 1} \in \mathcal{X}$ have some appropriate norm decay. Then, assuming that the solution operator $\Psi : \mathcal{X} \to \mathcal{Y}$ is analytic (Cohen, DeVore, and Schwab, 2011), it is possible to make use of the Taylor expansion

$$\Psi(x) = \sum_{h\in\mathcal{F}} \alpha_h(x)\psi_h,$$

where $\mathcal{F} = \{h \in \mathbb{N}^\infty : |h|_0 < \infty\}$ is the set of multi-indices and

$$\alpha_h(x) = \prod_{j\geq 1} a_j^{h_j} \in \mathbb{R}, \qquad \psi_h = \frac{1}{h!}\partial^h \Psi(0) \in \mathcal{Y};$$

here the differentiation ∂^h is with respect to the sequence of coefficients $\{a_j\}_{j\geq 1}$. Then Ψ is approximated by truncating the Taylor expansion to a finite subset of $\mathcal{F}$. For example this may be done recursively, by starting with $h = 0$ and building up the index set in a greedy manner. The method is not data-driven, and requires knowledge of the PDE to define equations to be solved for the ψ_h.

4.3 Approximation Theory

In this section, we prove our main approximation result: given any $\epsilon > 0$, we can find an $\epsilon-$approximation Ψ_{NN} of Ψ. We achieve this by making the appropriate choice of PCA truncation parameters, by choosing sufficient amounts of data, and by choosing a sufficiently rich neural network architecture to approximate φ by χ.

In what follows we define $F_\mathcal{X}$ to be a PCA encoder given by (4.10), using the input data $\{x_j\}_{j=1}^N$ drawn i.i.d. from μ, and $G_\mathcal{Y}$ to be a PCA decoder given by (4.11), using the data $\{y_j = \Psi(x_j)\}_{j=1}^N$. We also define

$$\begin{aligned} e_{NN}(x) &= \|(G_\mathcal{Y} \circ \chi \circ F_\mathcal{X})(x) - \Psi(x)\|_Y \\ &= \|\Psi_{NN}(x) - \Psi(x)\|_\mathcal{Y}. \end{aligned}$$

We prove the following theorem:

Theorem 22. *Let $\mathcal{X}$, $\mathcal{Y}$ be real, separable Hilbert spaces and let μ be a probability measure supported on $\mathcal{X}$ such that $\mathbb{E}_{x\sim\mu}\|x\|_\mathcal{X}^4 < \infty$. Suppose $\Psi : \mathcal{X} \to \mathcal{Y}$ is a μ-measurable, globally Lipschitz map. For any $\epsilon > 0$, there are dimensions $d_\mathcal{X} = d_\mathcal{X}(\epsilon) \in \mathbb{N}$, $d_\mathcal{Y} = d_\mathcal{Y}(\epsilon) \in \mathbb{N}$, a requisite amount of data $N = N(d_\mathcal{X}, d_\mathcal{Y}) \in \mathbb{N}$,* *parameters t, r, M depending on $d_{\mathcal{X}}, d_{\mathcal{Y}}$ and ϵ, and a zero-extended stacked neural network $\chi \in \mathcal{M}(d_{\mathcal{X}}, d_{\mathcal{Y}}; t, r, M)$ such that*

$$\mathbb{E}_{\{x_j\}\sim\mu}\mathbb{E}_{x\sim\mu}\big(e_{NN}(x)^2\big) < \epsilon.$$

Remark 23. *This theorem is a consequence of Theorem 26 which we state and prove below. For clarity and ease of exposition we state and prove Theorem 26 in a setting where Ψ is globally Lipschitz. With a more stringent moment condition on μ, the result can also be proven when Ψ is locally Lipschitz; we state and prove this result in Theorem 27.*

Remark 24. *The neural network $\chi \in \mathcal{M}(d_{\mathcal{X}}, d_{\mathcal{Y}}; t, r, M)$ has maximum number of layers $t \leq c[\log(M^2 d_{\mathcal{Y}}/\epsilon) + 1]$, with the number of active weights and biases in each component of the network $r \leq c(\epsilon/4M^2)^{-d_{\mathcal{X}}/2}[\log(M^2 d_{\mathcal{Y}}/\epsilon) + 1]$, with an appropriate constant $c = c(d_{\mathcal{X}}, d_{\mathcal{Y}}) \geq 0$ and support side-length $M = M(d_{\mathcal{X}}, d_{\mathcal{Y}}) > 0$. These bounds on t and r follow from Theorem 26 with $\tau = \epsilon^{\frac{1}{2}}$. Note, however, that in order to achieve error ϵ, the dimensions $d_{\mathcal{X}}, d_{\mathcal{Y}}$ must be chosen to grow as $\epsilon \to 0$; thus the preceding statements do not explicitly quantify the needed number of parameters, and depth, for error ϵ; to do so would require quantifying the dependence of c, M on $d_{\mathcal{X}}, d_{\mathcal{Y}}$ (a property of neural networks) and the dependence of $d_{\mathcal{X}}, d_{\mathcal{Y}}$ on ϵ (a property of the measure μ and spaces $\mathcal{X}, \mathcal{Y}$ – see Theorem 25). The theory in (Yarotsky, 2017), which we employ for the existence result for the neural network produces the constant c which depends on the dimensions $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ in an unspecified way.*

The double expectation reflects averaging over all possible new inputs x drawn from μ (inner expectation) and over all possible realizations of the i.i.d. dataset $\{x_j, y_j = \Psi(x_j)\}_{j=1}^N$ (outer expectation). The theorem as stated above is a consequence of Theorem 26 in which the error is broken into multiple components that are then bounded separately. Note that the theorem does not address the question of whether the optimization technique used to fit the neural network actually finds the choice which realizes the theorem; this gap between theory and practice is difficult to overcome, because of the non-convex nature of the training problem, and is a standard feature of theorems in this area (Kutyniok et al., 2019; Christoph Schwab and Jakob Zech, 2019).

The idea of the proof is to quantify the approximations $G_{\mathcal{X}} \circ F_{\mathcal{X}} \approx I_{\mathcal{X}}$ and $G_{\mathcal{Y}} \circ F_{\mathcal{Y}} \approx I_{\mathcal{Y}}$ and $\chi \approx \varphi$ so that Ψ_{NN} given by (4.5) is close to Ψ. The first two approximations, which show that Ψ_{PCA} given by (4.4) is close to Ψ, are studied in Subsection 4.3 (see Theorem 25). Then, in Subsection 4.3, we find a neural network χ able to approximate φ to the desired level of accuracy; this fact is part of the proof of Theorem 26. The zero-extension of the neural network arises from the fact that we employ a density theorem for a class of neural networks within continuous functions defined on compact sets. Since we cannot guarantee that $F_{\mathcal{X}}$ is bounded, we simply set the neural network output to zero on the set outside a hypercube with side-length $2M$. We then use the fact that this set has small μ-measure, for sufficiently large M.

PCA And Approximation

We work in the general notation and setting of Subsection 4.2 so as to obtain approximation results that are applicable to both using PCA on the inputs and on the outputs. In addition, denote by $(\text{HS}(\mathcal{H}), \langle \cdot, \cdot \rangle_{HS}, \| \cdot \|_{HS})$ the space of Hilbert-Schmidt operators over $\mathcal{H}$. We are now ready to state the main result of this subsection. Our goal is to control the projection error $R(V_{d,N})$ when using the finite-data PCA subspace in place of the optimal projection space since the PCA subspace is what is available in practice. Theorem 25 accomplishes this by bounding the error $R(V_{d,N})$ by the optimal error $R(V_d)$ plus a term related to the approximation $V_{d,N} \approx V_d$. While previous results such as (Blanchard, Bousquet, and Zwald, 2007; J. Shawe-Taylor et al., 2005; John Shawe-Taylor et al., 2002) focused on bounds for the excess error in probability w.r.t. the data, we present bounds in expectation, averaging over the data. Such bounds are weaker, but allow us to remove strict conditions on the data distribution to obtain more general results; for example, our theory allows for ν to be a Gaussian measure.

Theorem 25. *Let R be given by* (4.12) *and $V_{d,N}$, V_d by* (4.8), (4.14) *respectively. Then there exists a constant $Q \geq 0$, depending only on the data generating measure ν, such that*

$$\mathbb{E}_{\{u_j\}\sim\nu}[R(V_{d,N})] \leq \sqrt{\frac{Qd}{N}} + R(V_d),$$

where the expectation is over the dataset $\{u_j\}_{j=1}^N \overset{iid}{\sim} \nu$.

The proof generalizes that employed in (Blanchard, Bousquet, and Zwald, 2007, Thm. 3.1). We first find a bound on the average excess error $\mathbb{E}[R(V_{d,N}) - R_N(V_{d,N})]$ using Lemma 29. Then using Fan's Theorem (Fan, 1949) (Lemma 28), we bound the average sum of the tail eigenvalues of C_N by the sum of the tail eigenvalues of C, in particular, $\mathbb{E}[R_N(V_{d,N})] \leq R(V_d)$.

Proof. For brevity we simply write $\mathbb{E}$ instead of $\mathbb{E}_{\{u_j\}\sim\nu}$ throughout the proof. For any subspace $V \subseteq \mathcal{H}$, we have

$$\begin{aligned}R(V) &= \mathbb{E}_{u\sim\nu}[\|u\|^2 - 2\langle u, \Pi_V u\rangle + \langle \Pi_V u, \Pi_V u\rangle] = \mathbb{E}_{u\sim\nu}[\text{tr } (u\otimes u) - \langle \Pi_V u, \Pi_V u\rangle] \\ &= \mathbb{E}_{u\sim\nu}[\text{tr } (u\otimes u) - \langle \Pi_V, u\otimes u\rangle_{HS}] = \text{tr } C - \langle \Pi_V, C\rangle_{HS}\end{aligned}$$

where we used two properties of the fact that Π_V is an orthogonal projection operator, namely $\Pi_V^2 = \Pi_V = \Pi_V^*$ and

$$\langle \Pi_V, v\otimes z\rangle_{HS} = \langle v, \Pi_V z\rangle = \langle \Pi_V v, \Pi_V z\rangle \quad \forall v, z\in\mathcal{H}.$$

Repeating the above arguments for $R_N(V)$ in place of $R(V)$, with the expectation replaced by the empirical average, yields $R_N(V) = \text{tr } C_N - \langle \Pi_V, C_N\rangle_{HS}$. By noting that $\mathbb{E}[C_N] = C$ we then write

$$\begin{aligned}\mathbb{E}[R(V_{d,N}) - R_N(V_{d,N})] = \mathbb{E}\langle \Pi_{V_{d,N}}, C_N - C\rangle_{HS} &\le \sqrt{d}\,\mathbb{E}\|C_N - C\|_{HS} \\ &\le \sqrt{d}\sqrt{\mathbb{E}\|C_N - C\|_{HS}^2}\end{aligned}$$

where we used Cauchy-Schwarz twice along with the fact that $\|\Pi_{V_{d,N}}\|_{HS} = \sqrt{d}$ since $V_{d,N}$ is d-dimensional. Now by Lemma 29, which quantifies the Monte Carlo error between C and C_N in the Hilbert-Schmidt norm, we have that

$$\mathbb{E}[R(V_{d,N}) - R_N(V_{d,N})] \le \sqrt{\frac{Qd}{N}},$$

for a constant $Q \ge 0$. Hence by(4.9),

$$\mathbb{E}[R(V_{d,N})] \le \sqrt{\frac{Qd}{N}} + \mathbb{E}\sum_{j=d+1}^{N} \lambda_{j,N}.$$

It remains to estimate the second term above. Letting S_d denote the set of subspaces of d orthonormal elements in $\mathcal{H}$, Fan's Theorem (Proposition 28) gives

$$\begin{aligned}\sum_{j=1}^{d} \lambda_j &= \max_{\{v_1,\dots,v_d\}\in S_d} \sum_{j=1}^{d} \langle C v_j, v_j\rangle = \max_{\{v_1,\dots,v_d\}\in S_d} \mathbb{E}_{u\sim\nu} \sum_{j=1}^{d} |\langle u, v_j\rangle|^2 \\ &= \max_{\{v_1,\dots,v_d\}\in S_d} \mathbb{E}_{u\sim\nu} \sum_{j=1}^{d} \|\Pi_{\text{span}\{v_j\}} u\|^2 = \max_{V\in\mathcal{V}_d} \mathbb{E}_{u\sim\nu}\|\Pi_V u\|^2 \\ &= \mathbb{E}_{u\sim\nu}\|u\|^2 - \min_{V\in\mathcal{V}_d} E_{u\sim\nu}\|\Pi_{V^\perp} u\|^2.\end{aligned}$$

Observe that $\sum_{j=1}^{\infty} \lambda_j = \text{tr } C = \mathbb{E}_{u\sim\nu}\|u\|^2$ and so

$$\sum_{j=d+1}^{\infty} \lambda_j = \mathbb{E}_{u\sim\nu}\|u\|^2 - \sum_{j=1}^{d} \lambda_j = \min_{V\in\mathcal{V}_d} \mathbb{E}_{u\sim\nu}\|\Pi_{V^\perp} u\|^2.$$

We now repeat the above calculations for $\lambda_{j,N}$, the eigenvalues of C_N, by replacing the expectation with the empirical average to obtain

$$\sum_{j=d+1}^{N} \lambda_{j,N} = \min_{V \in \mathcal{V}_d} \frac{1}{N} \sum_{k=1}^{N} \|\Pi_{V^\perp} u_k\|^2,$$

and so

$$\mathbb{E} \sum_{j=d+1}^{N} \lambda_{j,N} \leq \min_{V \in \mathcal{V}_d} \mathbb{E} \frac{1}{N} \sum_{k=1}^{N} \|\Pi_{V^\perp} u_k\|^2 = \min_{V \in \mathcal{V}_d} \mathbb{E}_{u \sim \mu} \|\Pi_{V^\perp} u\|^2 = \sum_{j=d+1}^{\infty} \lambda_j.$$

Finally, we conclude that

$$\mathbb{E}[R(V_{d,N})] \leq \sqrt{\frac{Qd}{N}} + \sum_{j=d+1}^{\infty} \lambda_j = \sqrt{\frac{Qd}{N}} + R(V_d).$$

□

Neural Networks And Approximation

In this subsection we study the approximation of φ given in (4.3) by neural networks, combining the analysis with results from the preceding subsection to prove our main approximation result, Theorem 26. We will work in the notation of Section 4.2. We assume that $(\mathcal{X}, \langle \cdot, \cdot \rangle_{\mathcal{X}}, \| \cdot \|_{\mathcal{X}})$ and $(\mathcal{Y}, \langle \cdot, \cdot \rangle_{\mathcal{Y}}, \| \cdot \|_{\mathcal{Y}})$ are real, separable Hilbert spaces; μ is a probability measure supported on $\mathcal{X}$ with a finite fourth moment $\mathbb{E}_{x \sim \mu} \|x\|_{\mathcal{X}}^4 < \infty$, and $\Psi : \mathcal{X} \to \mathcal{Y}$ is measurable and globally L-Lipschitz: there exists a constant $L > 0$ such that

$$\forall x, z \in \mathcal{X} \quad \|\Psi(x) - \Psi(z)\|_{\mathcal{Y}} \leq L \|x - z\|_{\mathcal{X}}.$$

Note that this implies that Ψ is linearly bounded: for any $x \in \mathcal{X}$

$$\|\Psi(x)\|_{\mathcal{Y}} \leq \|\Psi(0)\|_{\mathcal{Y}} + \|\Psi(x) - \Psi(0)\|_{\mathcal{Y}} \leq \|\Psi(0)\|_{\mathcal{Y}} + L \|x\|_{\mathcal{X}}.$$

Hence we deduce existence of the fourth moment of the pushforward $\Psi_\sharp \mu$:

$$\mathbb{E}_{y \sim \Psi_\sharp \mu} \|y\|_{\mathcal{Y}}^4 = \int_{\mathcal{X}} \|\Psi(x)\|_{\mathcal{Y}}^4 d\mu(x) \leq \int_{\mathcal{X}} (\|\Psi(0)\|_{\mathcal{Y}} + L\|x\|_{\mathcal{X}})^4 d\mu(x) < \infty$$

since we assumed $\mathbb{E}_{x \sim \mu} \|x\|_{\mathcal{X}}^4 < \infty$.

Let us recall some of the notation from Subsections 4.2 and 4.2. Let $V_{d_{\mathcal{X}}}^{\mathcal{X}}$ be the $d_{\mathcal{X}}$-dimensional optimal projection space given by (4.14) for the measure μ and

$V^{\mathcal{X}}_{d_{\mathcal{X}},N}$ be the $d_{\mathcal{X}}$-dimensional PCA subspace given by (4.8) with respect to the input dataset $\{x_j\}_{j=1}^N$. Similarly let $V^{\mathcal{Y}}_{d_{\mathcal{Y}}}$ be the $d_{\mathcal{Y}}$-dimensional optimal projection space for the pushforward measure $\Psi_\sharp\mu$ and $V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}$ be the $d_{\mathcal{Y}}$-dimensional PCA subspace with respect to the output dataset $\{y_j = \Psi(x_j)\}_{j=1}^N$. We then define the input PCA encoder $F_{\mathcal{X}} : \mathcal{X} \to \mathbb{R}^{d_{\mathcal{X}}}$ by (4.10) and the input PCA decoder $G_{\mathcal{X}} : \mathbb{R}^{d_{\mathcal{X}}} \to \mathcal{X}$ by (4.11) both with respect to the orthonormal basis used to construct $V^{\mathcal{X}}_{d_{\mathcal{X}},N}$. Similarly we define the output PCA encoder $F_{\mathcal{Y}} : \mathcal{Y} \to \mathbb{R}^{d_{\mathcal{Y}}}$ and decoder $G_{\mathcal{Y}} : \mathbb{R}^{d_{\mathcal{Y}}} \to \mathcal{Y}$ with respect to the orthonormal basis used to construct $V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}$. Finally we recall $\varphi : \mathbb{R}^{d_{\mathcal{X}}} \to \mathbb{R}^{d_{\mathcal{Y}}}$ the map connecting the two latent spaces defined in (4.3). The approximation Ψ_{PCA} to Ψ based only on the PCA encoding and decoding is given by (4.4). In the following theorem, we prove the existence of a neural network giving an ϵ-close approximation to φ for fixed latent code dimensions $d_{\mathcal{X}}, d_{\mathcal{Y}}$ and quantify the error of the full approximation Ψ_{NN}, given in (4.5), to Ψ. We will be explicit about which measure the projection error is defined with respect to. In particular, we will write (4.12) as

$$R^\mu(V) = \mathbb{E}_{x\sim\mu}\|x - \Pi_V x\|^2_{\mathcal{X}}$$

for any subspace $V \subseteq \mathcal{X}$ and similarly

$$R^{\Psi_\sharp\mu}(V) = \mathbb{E}_{y\sim\Psi_\sharp\mu}\|y - \Pi_V y\|^2_{\mathcal{Y}}$$

for any subspace $V \subseteq \mathcal{Y}$.

Theorem 26. *Let $\mathcal{X}$, $\mathcal{Y}$ be real, separable Hilbert spaces and let μ be a probability measure supported on $\mathcal{X}$ such that $\mathbb{E}_{x\sim\mu}\|x\|^4_{\mathcal{X}} < \infty$. Suppose $\Psi : \mathcal{X} \to \mathcal{Y}$ is a μ-measurable, globally Lipschitz map. Fix $d_{\mathcal{X}}, d_{\mathcal{Y}}$, $N \geq \max\{d_{\mathcal{X}}, d_{\mathcal{Y}}\}$, $\delta \in (0,1)$ and $\tau > 0$. Define $M = \sqrt{\mathbb{E}_{x\sim\mu}\|x\|^2_{\mathcal{X}}/\delta}$. Then there exists a constant $c = c(d_{\mathcal{X}}, d_{\mathcal{Y}}) \geq 0$ and a zero-extended stacked neural network $\chi \in \mathcal{M}(d_{\mathcal{X}}, d_{\mathcal{Y}}; t, r, M)$ with $t \leq c(d_{\mathcal{X}}, d_{\mathcal{Y}})[\log(M\sqrt{d_{\mathcal{Y}}}/\tau) + 1]$ and $r \leq c(d_{\mathcal{X}}, d_{\mathcal{Y}})(\tau/2M)^{-d_{\mathcal{X}}}[\log(M\sqrt{d_{\mathcal{Y}}}/\tau) + 1]$, so that*

$$\mathbb{E}_{\{x_j\}\sim\mu}\mathbb{E}_{x\sim\mu}\big(e_{NN}(x)\big)^2 \leq C\left(\tau^2 + \sqrt{\delta} + \sqrt{\frac{d_{\mathcal{X}}}{N}} + R^\mu(V^{\mathcal{X}}_{d_{\mathcal{X}}}) + \sqrt{\frac{d_{\mathcal{Y}}}{N}} + R^{\Psi_\sharp\mu}(V^{\mathcal{Y}}_{d_{\mathcal{Y}}})\right), \tag{4.18}$$

where $C > 0$ is independent of $d_{\mathcal{X}}, d_{\mathcal{Y}}, N, \delta$, and τ.

The first two terms on the r.h.s. arise from the neural network approximation of φ while the last two pairs of terms are from the finite-dimensional approximation of $\mathcal{X}$ and $\mathcal{Y}$ respectively as prescribed by Theorem 25. The way to interpret the result is as follows: first choose $d_{\mathcal{X}}, d_{\mathcal{Y}}$ so that $R^{\mu}(V^{\mathcal{X}}_{d_{\mathcal{X}}})$ and $R^{\Psi_\sharp\mu}(V^{\mathcal{Y}}_{d_{\mathcal{Y}}})$ are small—these are intrinsic properties of the measures μ and $\Psi_\sharp\mu$; secondly, choose the amount of data N large enough to make $\max\{d_{\mathcal{X}}, d_{\mathcal{Y}}\}/N$ small, essentially controlling how well we approximate the intrinsic covariance structure of μ and $\Psi_\sharp\mu$ using samples; thirdly choose δ small enough to control the error arising from restricting the domain of φ; and finally choose τ sufficiently small to control the approximation of φ by a neural network restricted to a compact set. Note that the size and values of the parameters of the neural network χ will depend on the choice of δ as well as $d_{\mathcal{X}}, d_{\mathcal{Y}}$ and N in a manner which we do not specify. In particular, the dependence of c on $d_{\mathcal{X}}, d_{\mathcal{Y}}$ is not explicit in the theorem of (Yarotsky, 2017) which furnishes the existence of the requisite neural network χ. The parameter τ specifies the error tolerance between χ and φ on $[-M, M]^{d_{\mathcal{X}}}$. Intuitively, as $(\delta, \tau) \to 0$, we expect the number of parameters in the network to also grow (Maiorov and Pinkus, 1999). Quantifying this growth would be needed to fully understand the computational complexity of our method.

Proof. Recall the constant Q from Theorem 25. In what follows we take Q to be the maximum of the two such constants when arising from application of the theorem on the two different probability spaces $(\mathcal{X}, \mu)$ and $(\mathcal{Y}, \Psi_\sharp\mu)$. Through the proof we use $\mathbb{E}$ to denote $\mathbb{E}_{\{x_j\}\sim\mu}$ the expectation with respect to the dataset $\{x_j\}_{j=1}^N$.

We begin by approximating the error incurred by using Ψ_{PCA} given by (4.4):

$$\begin{aligned}
&\mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{PCA}(x) - \Psi(x)\|^2_{\mathcal{Y}} \\
&\quad = \mathbb{E}\mathbb{E}_{x\sim\mu}\|(G_{\mathcal{Y}} \circ F_{\mathcal{Y}} \circ \Psi \circ G_{\mathcal{X}} \circ F_{\mathcal{X}})(x) - \Psi(x)\|^2_{\mathcal{Y}} \\
&\quad = \mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x) - \Psi(x)\right\|^2_{\mathcal{Y}} \\
&\quad \le 2\mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x) - \Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(x)\right\|^2_{\mathcal{Y}} \\
&\qquad + 2\mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(x) - \Psi(x)\right\|^2_{\mathcal{Y}} \\
&\quad \le 2L^2\mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x - x\right\|^2_{\mathcal{X}} + 2\mathbb{E}\mathbb{E}_{y\sim\Psi_\sharp\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}y - y\right\|^2_{\mathcal{Y}} \\
&\quad = 2L^2\mathbb{E}[R^{\mu}(V^{\mathcal{X}}_{d_{\mathcal{X}},N})] + 2\mathbb{E}[R^{\Psi_\sharp\mu}(V^{\mathcal{Y}}_{d_{\mathcal{Y}},N})]
\end{aligned} \tag{4.19}$$

noting that the operator norm of an orthogonal projection is 1. Theorem 25 allows us to control this error, and leads to

$$\begin{aligned}\mathbb{E}\mathbb{E}_{x\sim\mu}\big(e_{NN}(x)^2\big) &= \mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{NN}(x)-\Psi(x)\|_{\mathcal{Y}}^2\\ &\le 2\mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{NN}(x)-\Psi_{PCA}(x)\|_{\mathcal{Y}}^2 + 2\mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{PCA}(x)-\Psi(x)\|_{\mathcal{Y}}^2\\ &\le 2\mathbb{E}_{x\sim\mu}\|\Psi_{NN}(x)-\Psi_{PCA}(x)\|_{\mathcal{Y}}^2\\ &\qquad + 4L^2\left(\sqrt{\frac{Qd_{\mathcal{X}}}{N}} + R^{\mu}(V_{d_{\mathcal{X}}}^{\mathcal{X}})\right) + 4\left(\sqrt{\frac{Qd_{\mathcal{Y}}}{N}} + R^{\Psi_\sharp\mu}(V_{d_{\mathcal{Y}}}^{\mathcal{Y}})\right).\end{aligned} \tag{4.20}$$

We now approximate φ by a neural network χ as a step towards estimating $\|\Psi_{NN}(x)-\Psi_{PCA}(x)\|_{\mathcal{Y}}$. To that end we first note from Lemma 30 that φ is Lipschitz, and hence continuous, as a mapping from $\mathbb{R}^{d_{\mathcal{X}}}$ into $\mathbb{R}^{d_{\mathcal{Y}}}$. Identify the components $\varphi(s) = (\varphi^{(1)}(s), \dots, \varphi^{(d_{\mathcal{Y}})}(s))$ where each function $\varphi^{(j)} \in C(\mathbb{R}^{d_{\mathcal{X}}};\mathbb{R})$. We consider the restriction of each component function to the set $[-M,M]^{d_{\mathcal{X}}}$. Let us now change variables by defining $\tilde{\varphi}^{(j)} : [0,1]^{d_{\mathcal{X}}} \to \mathbb{R}$ by $\tilde{\varphi}^{(j)}(s) = (1/2M)\varphi^{(j)}(2Ms - M)$ for any $s \in [0,1]^{d_{\mathcal{X}}}$. Note that equivalently we have $\varphi^{(j)}(s) = 2M\tilde{\varphi}^{(j)}((s+M)/2M)$ for any $s \in [-M,M]^{d_{\mathcal{X}}}$ and further $\varphi^{(j)}$ and $\tilde{\varphi}^{(j)}$ have the same Lipschitz constants on their respective domains. Applying (Yarotsky, 2017, Thm. 1) to the $\tilde{\varphi}^{(j)}$(s) then yields existence of neural networks $\tilde{\chi}^{(1)}, \dots, \tilde{\chi}^{(d_{\mathcal{Y}})} : [0,1]^{d_{\mathcal{X}}} \to \mathbb{R}$ such that

$$|\tilde{\chi}^{(j)}(s) - \tilde{\varphi}^{(j)}(s)| < \frac{\tau}{2M\sqrt{d_{\mathcal{Y}}}} \quad \forall s \in [0,1]^{d_{\mathcal{X}}},$$

for any $j \in \{1, \dots, d_{\mathcal{Y}}\}$. In fact, each neural network $\tilde{\chi}^{(j)} \in \mathcal{M}(d_{\mathcal{X}}; t^{(j)}, r^{(j)})$ with parameters $t^{(j)}$ and $r^{(j)}$ satisfying

$$t^{(j)} \le c^{(j)}\left[\log(M\sqrt{d_{\mathcal{Y}}}/\tau)+1\right], \qquad r^{(j)} \le c^{(j)}\left(\frac{\tau}{2M}\right)^{-d_{\mathcal{X}}}\left[\log(M\sqrt{d_{\mathcal{Y}}}/\tau)+1\right],$$

with constants $c^{(j)}(d_{\mathcal{X}}) > 0$. Hence defining $\chi^{(j)} : \mathbb{R}^{d_{\mathcal{X}}} \to \mathbb{R}$ by $\chi^{(j)}(s) := 2M\tilde{\chi}^{(j)}((s+M)/2M)$ for any $s \in [-M,M]^{d_{\mathcal{X}}}$, we have that

$$\big|\big(\chi^{(1)}(s), \dots, \chi^{(d_{\mathcal{Y}})}(s)\big) - \varphi(s)\big|_2 < \tau \quad \forall s \in [-M,M]^{d_{\mathcal{X}}}.$$

We can now simply define $\chi : \mathbb{R}^{d_{\mathcal{X}}} \to \mathbb{R}^{d_{\mathcal{Y}}}$ as the stacked network $(\chi^{(1)}, \dots, \chi^{d_{\mathcal{Y}}})$ extended by zero outside of $[-M,M]^{d_{\mathcal{X}}}$ to immediately obtain

$$\sup_{s\in[-M,M]^{d_{\mathcal{X}}}} \big|\chi(s) - \varphi(s)\big|_2 < \tau. \tag{4.21}$$

Thus, by construction $\chi \in \mathcal{M}(d_{\mathcal{X}}, d_{\mathcal{Y}}, t, r, M)$ with at most $t \le \max_j t^{(j)}$ many layers and $r \le r^{(j)}$ many active weights and biases in each of its components.

Let us now define the set $A = \{x \in \mathcal{X} : F_{\mathcal{X}}(x) \in [-M, M]^{d_{\mathcal{X}}}\}$. By Lemma 31, $\mu(A) \geq 1 - \delta$ and $\mu(A^c) \leq \delta$. Define the approximation error

$$e_{PCA}(x) = \|\Psi_{NN}(x) - \Psi_{PCA}(x)\|_{\mathcal{Y}}$$

and decompose its expectation as

$$\mathbb{E}_{x\sim\mu}\big(e_{PCA}(x)^2\big) = \underbrace{\int_A e_{PCA}(x)^2 d\mu(x)}_{:=I_A} + \underbrace{\int_{A^c} e_{PCA}(x)^2 d\mu(x)}_{:=I_{A^c}}.$$

For the first term,

$$I_A \leq \int_A \|(G_{\mathcal{Y}} \circ \chi \circ F_{\mathcal{X}})(x) - (G_{\mathcal{Y}} \circ \varphi \circ F_{\mathcal{X}})(x)\|_{\mathcal{Y}}^2 d\mu(x) \leq \tau^2, \tag{4.22}$$

by using the fact, established in Lemma 30, that $G_{\mathcal{Y}}$ is Lipschitz with Lipschitz constant 1, the τ-closeness of χ to φ from (4.21), and $\mu(A) \leq 1$. For the second term we have, using that $G_{\mathcal{Y}}$ has Lipschitz constant 1 and that χ vanishes on A^c,

$$\begin{aligned} I_{A^c} &\leq \int_{A^c} \|(G_{\mathcal{Y}} \circ \chi \circ F_{\mathcal{X}})(x) - (G_{\mathcal{Y}} \circ \varphi \circ F_{\mathcal{X}})(x)\|_{\mathcal{Y}}^2 d\mu(x) \\ &\leq \int_{A^c} |\chi(F_{\mathcal{X}}(x)) - \varphi(F_{\mathcal{X}}(x))|_2^2 d\mu(x) = \int_{A^c} |\varphi(F_{\mathcal{X}}(x))|_2^2 d\mu(x). \end{aligned} \tag{4.23}$$

Once more from Lemma 30, we have that

$$|F_{\mathcal{X}}(x)|_2 \leq \|x\|_{\mathcal{X}}; \quad |\varphi(x)|_2 \leq |\varphi(0)|_2 + L|x|_2,$$

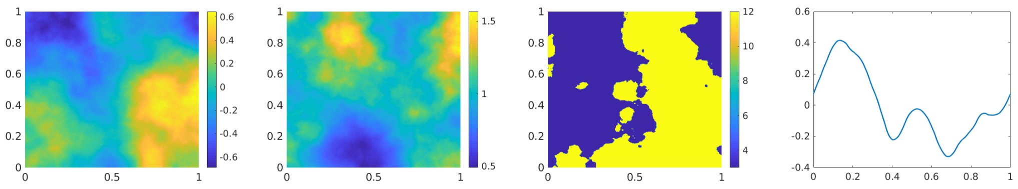
so that

$$\begin{aligned} I_{A^c} &\leq 2\big(\mu(A^c)|\varphi(0)|_2^2 + \mu(A_c)^{\frac{1}{2}} L^2 (\mathbb{E}_{x\sim\mu}\|x\|_{\mathcal{X}}^4)^{\frac{1}{2}}\big), \\ &\leq 2\big(\delta|\varphi(0)|_2^2 + \delta^{\frac{1}{2}} L^2 (\mathbb{E}_{x\sim\mu}\|x\|_{\mathcal{X}}^4)^{\frac{1}{2}}\big). \end{aligned} \tag{4.24}$$

Combining (4.20), (4.22), and (4.24), we obtain the desired result. □

4.4 Numerical Results

We now present a series of numerical experiments that demonstrate the effectiveness of our proposed methodology in the context of the approximation of parametric PDEs. We work in settings which both verify our theoretical results and show that the ideas work outside the confines of the theory. The key idea underlying our work is to construct the neural network architecture so that it is defined *as a map between Hilbert spaces* and only then to discretize and obtain a method that

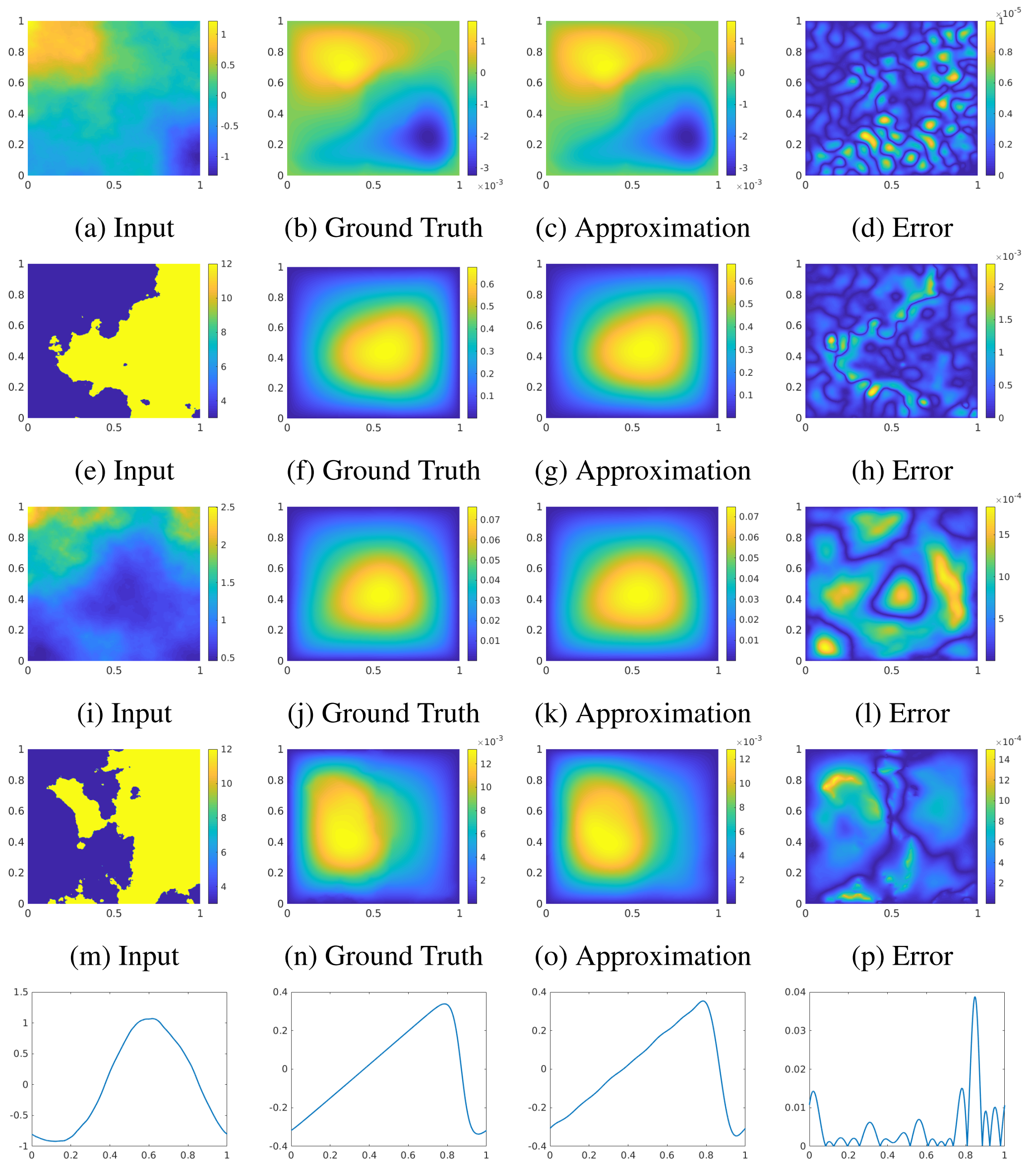


(a) μ_{G} (b) μ_{L} (c) μ_{P} (d) μ_{B}

Figure 42: Representative samples for each of the probability measures μ_{G}, μ_{L}, μ_{P}, and μ_{B} defined in Subsection 4.4. μ_{G} and μ_{P} are used in Subsection 4.4 to model the inputs, μ_{L} and μ_{P} are used in Subsection 4.4, and μ_{B} is used in Subsection 4.4.

is implementable in practice; prevailing methodologies first discretize and then apply a standard neural network. Our approach leads, when discretized, to methods that have properties which are uniform with respect to the mesh size used. We demonstrate this through our numerical experiments. In practice, we obtain an approximation Ψ_{num} to Ψ_{NN}, reflecting the numerical discretization used, and the fact that μ and its pushforward under Ψ are only known to us through samples and, in particular, samples of the pushforward of μ under the numerical approximation of the input-output map. However since, as we will show, our method is robust to the discretization used, we will not explicitly reflect the dependence of the numerical method in the notation that appears in the remainder of this section.

In Subsection 4.4 we introduce a class of parametric elliptic PDEs arising from the Darcy model of flow in porous media, as well as the time-dependent, parabolic, Burgers' equation, that define a variety of input-output maps for our numerical experiments; we also introduce the probability measures that we use on the input spaces. Subsection 4.4 presents numerical results for a Lipschitz map. Subsections 4.4, 4.4 present numerical results for the Darcy flow problem and the flow map for the Burgers' equation; this leads to non-Lipschitz input-output maps, beyond our theoretical developments. We emphasize that while our method is designed for approximating nonlinear operators Ψ, we include some numerical examples where Ψ is linear. Doing so is helpful for confirming some of our theory and comparing against other methods in the literature. Note that when Ψ is linear, each piece in the approximate decomposition (4.4) is also linear, in particular, φ is linear. Therefore it is sufficient to parameterize φ as a linear map (matrix of unknown coefficients) instead of a neural network. We include such experiments in Section 4.4 revealing that, while a neural network approximating φ arbitrarily well exists, the optimization methods used for training the neural network fail to find it. It may therefore be beneficial to directly build into the parametrization known properties of φ, such as



(a) Input (b) Ground Truth (c) Approximation (d) Error

(e) Input (f) Ground Truth (g) Approximation (h) Error

(i) Input (j) Ground Truth (k) Approximation (l) Error

(m) Input (n) Ground Truth (o) Approximation (p) Error

(q) Input (r) Ground Truth (s) Approximation (t) Error

Figure 43: Randomly chosen examples from the test set for each of the five considered problems. Each row is a different problem: linear elliptic, Poisson, Darcy flow with log-normal coefficients, Darcy flow with piecewise constant coefficients, and Burgers' equation respectively from top to bottom. The approximations are constructed with our best performing method (for $N = 1024$): Linear $d = 150$, Linear $d = 150$, NN $d = 70$, NN $d = 70$, and NN $d = 15$ respectively from top to bottom.

linearity, when they are known. We emphasize that, for general nonlinear maps, linear methods significantly underperform in comparison with our neural network approximation and we will demonstrate this for the Darcy flow problem, and for

Burgers' equation.

We use standard implementations of PCA, with dimensions specified for each computational example below. All computational examples use an identical neural network architecture: a 5-layer dense network with layer widths $500, 1000, 2000, 1000, 500$, ordered from first to last layer, and the SELU nonlinearity (Klambauer et al., 2017). We note that Theorem 26 requires greater depth for greater accuracy but that we have found our 5-layer network to suffice for all of the examples described here. Thus we have not attempted to optimize the architecture of the neural network. We use stochastic gradient descent with Nesterov momentum (0.99) to train the network parameters (Goodfellow, Bengio, and Courville, 2016), each time picking the largest learning rate that does not lead to blow-up in the error. While the network must be re-trained for each new choice of reduced dimensions $d_{\mathcal{X}}, d_{\mathcal{Y}}$, initializing the hidden layers with a pre-trained network can help speed up convergence.

PDE Setting

We will consider a variety of solution maps defined by second order elliptic PDEs of the form (4.1). which are prototypical of many scientific applications. We take $D = (0,1)^2$ to be the unit box, $a \in L^\infty(D;\mathbb{R}_+), f \in L^2(D;\mathbb{R})$, and let $u \in H_0^1(D;\mathbb{R})$ be the unique weak solution of (4.1). Note that, since D is bounded, $L^\infty(D;\mathbb{R}_+)$ is continuously embedded within the Hilbert space $L^2(D;\mathbb{R}_+)$. We will consider two variations of the input-output map generated by the solution operator for (4.1); in one, it is Lipschitz and lends itself to the theory of Subsection 4.3 and, in the other, it is not Lipschitz. We obtain numerical results which demonstrate our theory as well as demonstrating the effectiveness of our proposed methodology in the non-Lipschitz setting.

Furthermore, we consider the one-dimensional viscous Burgers' equation on the torus given as

$$\begin{aligned} \frac{\partial}{\partial t}u(s,t) + \frac{1}{2}\frac{\partial}{\partial s}(u(s,t))^2 &= \beta\frac{\partial^2}{\partial s^2}u(s,t), \qquad && (s,t) \in \mathbb{T}^1 \times (0,\infty) \\ u(s,0) &= u_0(s), && s \in \mathbb{T}^1 \end{aligned} \tag{4.25}$$

where $\beta > 0$ is the viscosity coefficient and $\mathbb{T}^1$ is the one dimensional unit torus obtained by equipping the interval $[0,1]$ with periodic boundary conditions. We take $u_0 \in L^2(\mathbb{T}^1;\mathbb{R})$ and have that, for any $t > 0$, $u(\cdot,t) \in H^r(\mathbb{T}^1;\mathbb{R})$ for any $r > 0$ is the unique weak solution to (4.25) (Temam, 2012). In Subsection 4.4, we consider the input-output map generated by the flow map of (4.25) evaluated at a fixed time which is a locally Lipschitz operator.

We make use of four probability measures which we now describe. The first, which will serve as a base measure in two dimensions, is the Gaussian $\mu_{\text{G}} = \mathcal{N}(0, (-\Delta + 9I)^{-2})$ with a zero Neumann boundary condition on the operator Δ. Then we define μ_{L} to be the log-normal measure defined as the push-forward of μ_{G} under the exponential map, i.e. $\mu_{\text{L}} = \exp_\sharp \mu_{\text{G}}$. Furthermore, we define $\mu_{\text{P}} = T_\sharp \mu_{\text{G}}$ to be the push-forward of μ_{G} under the piecewise constant map

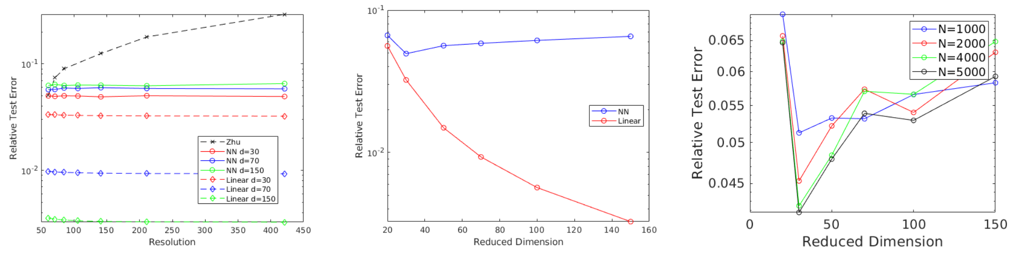
$$T(s) = \begin{cases} 12 & s \geq 0, \\ 3 & s < 0. \end{cases}$$

Lastly, we consider the Gaussian $\mu_{\text{B}} = \mathcal{N}(0, 7^4(-\frac{d^2}{ds^2} + 7^2 I)^{-2.5})$ defined on $\mathbb{T}^1$. Figure 42 shows an example draw from each of the above measures. We will use as μ one of these four measures in each experiment we conduct. Such probability measures are commonly used in the stochastic modeling of physical phenomenon (Lord, Powell, and Shardlow, 2014). For example, μ_{P} may be thought of as modeling the permeability of a porous medium containing two different constituent parts (Iglesias, K. Lin, and A.M. Stuart, 2014). Note that it is to be expected that a good choice of architecture will depend on the probability measure used to generate the inputs. Indeed good choices of the reduced dimensions $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ are determined by the input measure and its pushforward under Ψ, respectively.

For each subsequently described problem we use, unless stated otherwise, $N = 1024$ training examples from μ and its pushforward under Ψ, from which we construct Ψ_{NN}, and then 5000 unseen testing examples from μ in order to obtain a Monte Carlo estimate of the relative test error:

$$\mathbb{E}_{x \sim \mu} \frac{\|(G_2 \circ \chi \circ F_1)(x) - \Psi(x)\|_{\mathcal{Y}}}{\|\Psi(x)\|_{\mathcal{Y}}}.$$

For problems arising from (4.1), all data is collected on a uniform 421×421 mesh and the PDE is solved with a second order finite-difference scheme. For problems arising from (4.25), all data is collected on a uniform 4096 point mesh and the PDE is solved using a pseudo-spectral method. Data for all other mesh sizes is sub-sampled from the original. We refer to the size of the discretization in one direction e.g. 421, as the *resolution*. We fix $d_{\mathcal{X}} = d_{\mathcal{Y}}$ (the dimensions after PCA in the input and output spaces) and refer to this as *the reduced dimension*. We experiment with using a linear map as well as a dense neural network for approximating φ; in all figures we



(a) (b) (c)

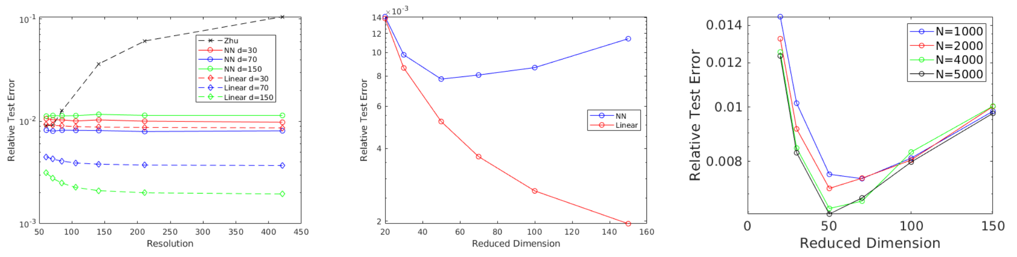
Figure 44: Relative test errors on the linear elliptic problem. Using $N = 1024$ training examples, panel (a) shows the errors as a function of the resolution while panel (b) fixes a 421×421 mesh and shows the error as a function of the reduced dimension. Panel (c) only shows results for our method using a neural network, fixing a 421×421 mesh and showing the error as a function of the reduced dimension for different amounts of training data.

distinguish between these by referring to Linear or NN approximations respectively. When parameterizing with a neural network, we use the aforementioned stochastic gradient based method for training, while, when parameterizing with a linear map, we simply solve the linear least squares problem by the standard normal equations.

We also compare all of our results to the work of (Zhu and Zabaras, 2018) which utilizes a 19-layer fully-connected convolutional neural network, referencing this approach as Zhu within the text. This is done to show that the image-to-image regression approach that many such works utilize yields approximations that are not consistent in the continuum, and hence across different discretizations; in contrast, our methodology is designed as a mapping between Hilbert spaces and as a consequence is robust across different discretizations. For some problems in Subsection 4.4, we compare to the method developed in (Chkifa et al., 2013), which we refer to as Chkifa. For the problems in Subsection 4.4, we also compare to the reduced basis method (R. A. DeVore, 2014; Quarteroni, Manzoni, and Negri, 2015) when instantiated with PCA. We note that both Chkifa and the reduced basis method are intrusive, i.e., they need knowledge of the governing PDE. Furthermore the method of Chkifa needs full knowledge of the generating process of the inputs. We re-emphasize that our proposed method is fully data-driven.

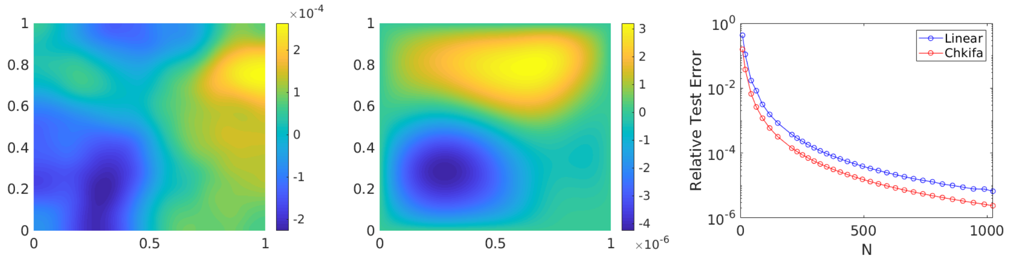
Globally Lipschitz Solution Map

We consider the input-output map $\Psi : L^2(D;\mathbb{R}) \to H_0^1(D;\mathbb{R})$ mapping $f \mapsto u$ in (4.1) with the coefficient a fixed. Since (4.1) is a linear PDE, Ψ is linear and therefore Lipschitz. We study two instantiations of this problem. In the first, we



(a) (b) (c)

Figure 45: Relative test errors on the Poisson problem. Using $N = 1024$ training examples, panel (a) shows the errors as a function of the resolution while panel (b) fixes a 421×421 mesh and shows the error as a function of the reduced dimension. Panel (c) only shows results for our method using a neural network, fixing a 421×421 mesh and showing the error as a function of the reduced dimension for different amounts of training data.



(a) (b) (c)

Figure 46: Panel (a) shows a sample drawn from the model (4.26) while panel (b) shows the solution of the Poisson equation with the sample from (a) as the r.h.s. Panel (c) shows the relative test error as a function of the amount of PDE solves/ training data for the method of Chkifa and our method respectively. We use the reduced dimension $d = N$.

draw a single $a \sim \mu_{\mathrm{P}}$ and fix it. We then solve (4.1) with data $f \sim \mu_{\mathrm{G}}$. We refer to this as the *linear elliptic* problem. See row 1 of Figure 43 for an example. In the second, we set $a(w) = 1 \; \forall w \in D$, in which case (4.1) becomes the Poisson equation which we solve with data $f \sim \mu = \mu_{\mathrm{G}}$. We refer to this as the *Poisson* problem. See row 2 of Figure 43 for an example.

Figure 44 (a) shows the relative test errors as a function of the resolution on the linear elliptic problem, while Figure 45 (a) shows them on the Poisson problem. The primary observation to make about panel (a) in these two figures is that it shows that the error in our proposed method does not change as the resolution changes. In contrast, it also shows that the image-to-image regression approach of Zhu (Zhu and Zabaras, 2018), whilst accurate at low mesh resolution, fails to be invariant to the size of the discretization and errors increase in an uncontrolled fashion as greater resolution is used. The fact that our dimension reduction approach achieves constant error as we refine the mesh, reflects its design as a method on Hilbert space which may be approximated consistently on different meshes. Since the operator Ψ here is linear, the true map of interest φ given by (4.3) is linear since $F_{\mathcal{Y}}$ and $G_{\mathcal{X}}$ are, by the definition of PCA, linear. It is therefore unsurprising that the linear approximation consistently outperforms the neural network, a fact also demonstrated in panel (a) of the two figures. While it is theoretically possible to find a neural network that can, at least, match the performance of the linear map, in practice, the non-convexity of the associated optimization problem can cause non-optimal behavior. Panels (b) of Figures 44 and 45 show the relative error as a function of the reduced dimension for a fixed mesh size. We see that while the linear maps consistently improve with the reduced dimension, the neural networks struggle as the complexity of the optimization problem is increased. This problem can usually be alleviated with the addition of more data as shown in panels (c), but there are still no guarantees that the optimal neural network is found. Since we use a highly-nonlinear 5-layer network to represent the linear φ, this issue is exacerbated for this problem and the addition of more data only slightly improves the accuracy as seen in panels (c). In Appendix 4.9, we show the relative test error during the training process and observe that some overfitting occurs, indicating that the optimization problem is stuck in a local minima away from the optimal linear solution. This is an issue that is inherent to most deep neural network based methods. Our results suggest that building in *a priori* information about the solution map, such as linearity, can be very beneficial for the approximation scheme as it can help reduce the complexity of the optimization.

To compare to the method of Chkifa (Chkifa et al., 2013), we will assume the following model for the inputs,

$$f = \sum_{j=1}^{\infty} \xi_j \phi_j \tag{4.26}$$

where $\xi_j \sim U(-1, 1)$ is an i.i.d. sequence, and $\phi_j = \sqrt{\lambda_j}\psi_j$ where λ_j, ψ_j are the eigenvalues and eigenfunctions of the operator $(-\Delta + 100I)^{-4.1}$ with a zero Neumann boundary. This construction ensures that there exists $p \in (0, 1)$ such that $(\|\phi_j\|_{L^\infty})_{j\geq 1} \in \ell^p(\mathbb{N}; \mathbb{R})$ which is required for the theory in (Cohen, DeVore, and Schwab, 2011). We assume this model for f, the r.h.s. of the Poisson equation, and consider the solution operator $\Psi : \ell^\infty(\mathbb{N}; \mathbb{R}) \to H_0^1(D; \mathbb{R})$ mapping $(\xi_j)_{j\geq 1} \mapsto u$.

Figure 46 panels (a)-(b) show an example input from (4.26) and its corresponding solution u. Since this operator is linear, its Taylor series representation simply amounts to

$$\Psi((\xi_j)_{j\geq 1}) = \sum_{j=1}^{\infty} \xi_j \eta_j \tag{4.27}$$

where $\eta_j \in H_0^1(D;\mathbb{R})$ satisfy

$$-\Delta \eta_j = \phi_j.$$

This is easily seen by plugging in our model (4.26) for f into the Poisson equation and formally inverting the Laplacian. We further observe that the $\ell^1(\mathbb{N};\mathbb{R})$ summability of the sequence $(\|\eta_j\|_{H_0^1})_{j\geq 1}$ (inherited from $(\|\phi_j\|_{L^\infty})_{j\geq 1} \in \ell^p(\mathbb{N};\mathbb{R})$) implies that our power series (4.27) is summable in $H_0^1(D;\mathbb{R})$. Combining the two observations yields analyticity of Ψ with the same rates as in (Cohen, DeVore, and Schwab, 2011) obtained via Stechkin's inequality. For a proof, see Theorem 32.

We employ the method of Chkifa simply by truncation of (4.27) to d elements, noting that in this simple linear setting there is no longer a need for greedy selection of the index set. We note that this truncation requires d PDE solves of the Poisson equation hence we compare to our method when using $N = d$ data points, since this also counts the number of PDE solves. Since the problem is linear, we use a linear map to interpolate the PCA latent spaces and furthermore set the reduced dimension of our PCA(s) to N. Panel (c) of Figure 46 shows the results. We see that the method of Chkifa outperforms our method for any fixed number of PDE solves, although the empirical rate of convergence appears very similar for both methods. Furthermore we highlight that while our method appears to have a larger error constant than that of Chkifa, it has the advantage that it requires no knowledge of the model 4.26 or of the Poisson equation; it is driven entirely by the training data.

Darcy Flow

We now consider the input-output map $\Psi : L^\infty(D;\mathbb{R}_+) \to H_0^1(D;\mathbb{R})$ mapping $a \mapsto u$ in (4.1) with $f(s) = 1\ \forall s \in D$ fixed. In this setting, the solution operator is nonlinear and is locally Lipschitz as a mapping from $L^\infty(D;\mathbb{R}_+)$ to $H_0^1(D;\mathbb{R})$ (Dashti, Harris, and A.M. Stuart, 2012). However our results require a Hilbert space structure, and we view the solution operator as a mapping from $L^2(D;\mathbb{R}_+) \supset L^\infty(D;\mathbb{R}_+)$ into $H_0^1(D;\mathbb{R}_+)$, noting that we will choose the probability measure μ on $L^2(D;\mathbb{R}_+)$ to satisfy $\mu(L^\infty(D;\mathbb{R}_+)) = 1$. In this setting, Ψ is not locally Lipschitz and hence Theorem 26 is not directly applicable. Nevertheless, our

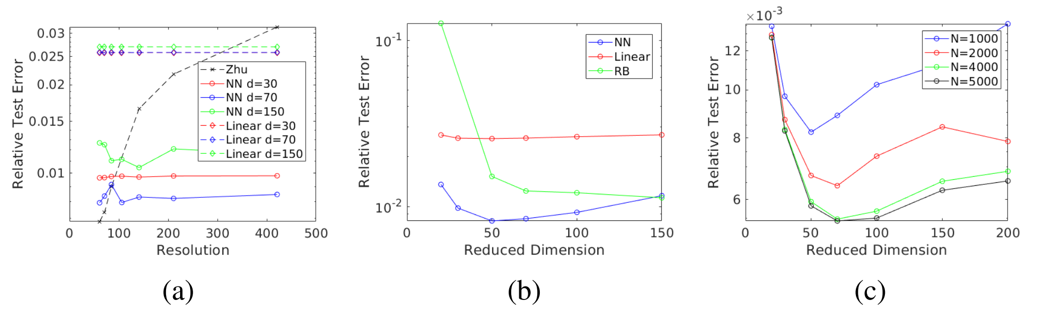


Figure 47: Relative test errors on the Darcy flow problem with log-normal coefficients. Using $N = 1024$ training examples, panel (a) shows the errors as a function of the resolution while panel (b) fixes a 421×421 mesh and shows the error as a function of the reduced dimension. Panel (c) only shows results for our method using a neural network, fixing a 421×421 mesh and showing the error as a function of the reduced dimension for different amounts of training data.

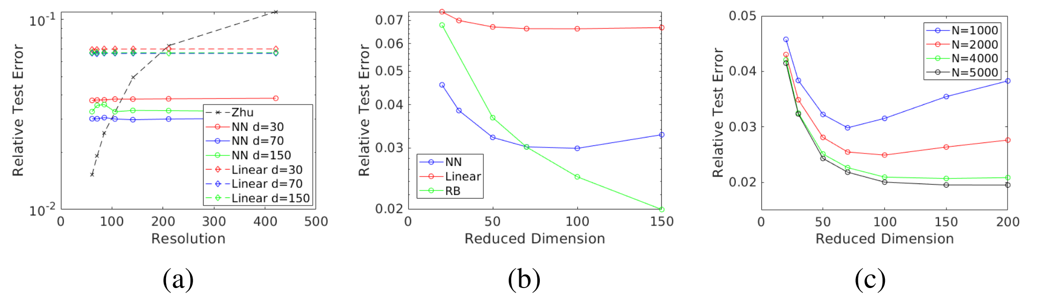
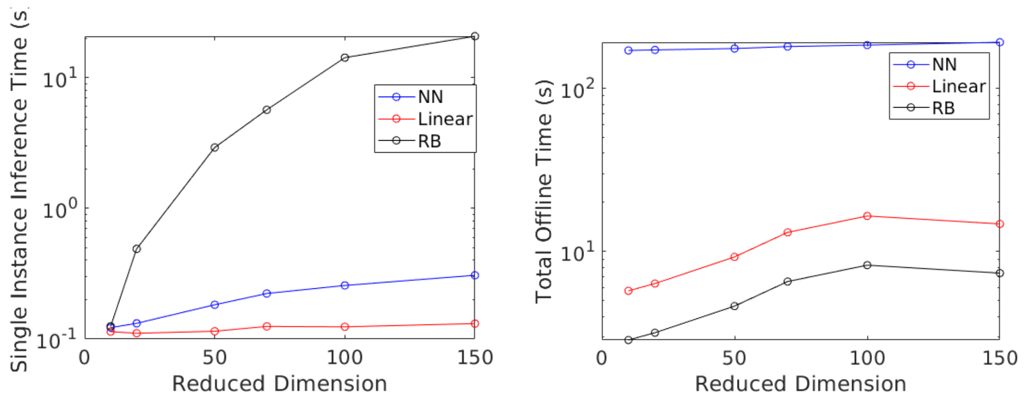


Figure 48: Relative test errors on the Darcy flow problem with piecewise constant coefficients. Using $N = 1024$ training examples, panel (a) shows the errors as a function of the resolution while panel (b) fixes a 421×421 mesh and shows the error as a function of the reduced dimension. Panel (c) only shows results for our method using a neural network, fixing a 421×421 mesh and showing the error as a function of the reduced dimension for different amounts of training data.

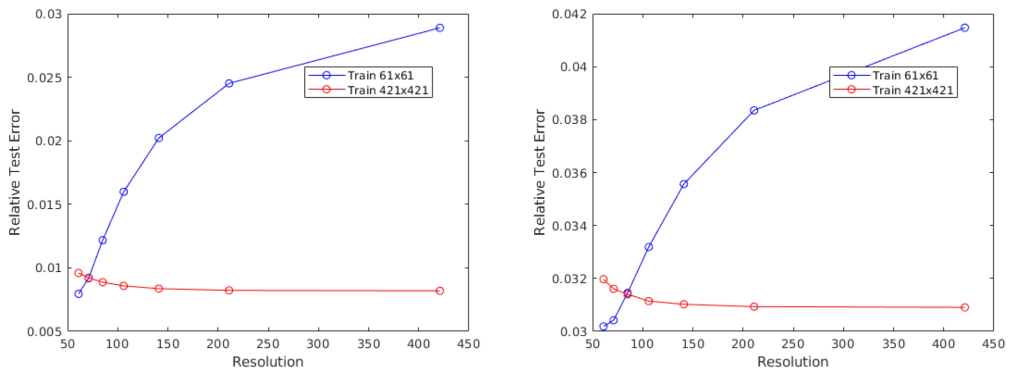
methodology exhibits competitive numerical performance. See rows 3 and 4 of Figure 43 for an example.

Figure 47 (a) shows the relative test errors as a function of the resolution when $a \sim \mu = \mu_{\text{L}}$ is log-normal while Figure 48 (a) shows them when $a \sim \mu = \mu_{\text{P}}$ is piecewise constant. In both settings, we see that the error in our method is invariant to mesh-refinement. Since the problem is nonlinear, the neural network outperforms the linear map. However we see the same issue as in Figure 44 where increasing the reduced dimension does not necessarily improve the error due to the increased complexity of the optimization problem. Panels (b) of Figures 47 and 48 confirm this observation. This issue can be alleviated with additional training data. Indeed,



(a) Online (b) Offline

Figure 49: The online and offline computation times for the Darcy flow problem with piecewise constant coefficients. The number of training examples $N = 1024$ and grid resolution 421×421 are fixed. The results are reported in seconds and all computations are done on a single GTX 1080 Ti GPU.



(a) μ_{L} (b) μ_{P}

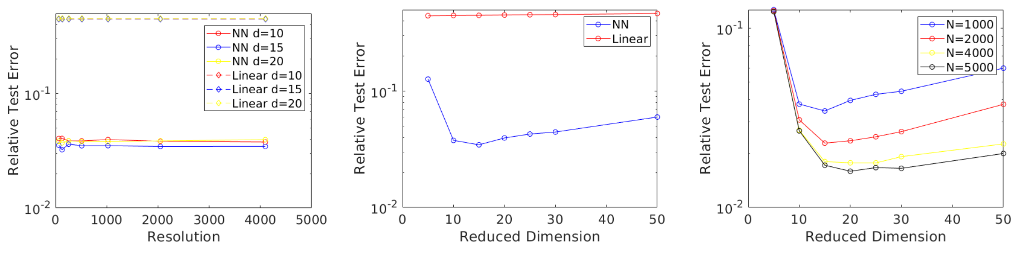
Figure 410: Relative test errors on both Darcy flow problems with reduced dimension $d = 70$, training on a single mesh and transferring the solution to other meshes. When the training mesh is smaller than the desired output mesh, the PCA basis are interpolated using cubic splines. When the training mesh is larger than the desired output mesh, the PCA basis are sub-sampled.

panels (c) of Figures 47 and 48 show that the error curve is flattened with more data. We highlight that these results are consistent with our interpretation of Theorem 22: the reduced dimensions $d_{\mathcal{X}}, d_{\mathcal{Y}}$ are determined first by the properties of the measure μ and its pushforward, and then the amount of data necessary is obtained to ensure that the finite data approximation error is of the same order of magnitude as the finite-dimensional approximation error. In summary, the size of the training dataset N should increase with the number of reduced dimensions.

For this problem, we also compare to the reduced basis method (RB) when instantiated with PCA. We implement this by a standard Galerkin projection, expanding the solution in the PCA basis and using the weak form of (4.1) to find the coefficients. We note that the errors of both methods are very close, but we find that the online runtime of our method is significantly better. Letting K denote the mesh-size and d the reduced dimension, the reduced basis method has a runtime of $\mathcal{O}(d^2K + d^3)$ while our method has the runtime $\mathcal{O}(dK)$ plus the runtime of the neural network which, in practice, we have found to be negligible. We show the online inference time as well as the offline training time of the methods in Figure 49. While the neural network has the highest offline cost, its small online cost makes it a more practical method. Indeed, without parallelization when $d = 150$, the total time (online and offline) to compute all 5000 test solutions is around 28 hours for the RBM. On the other hand, for the neural network, it is 28 minutes. The difference is pronounced when needing to compute many solutions in parallel. Since most modern architectures are able to internally parallelize matrix-matrix multiplication, the total time to train and compute the 5000 examples for the neural network is only 4 minutes. This issue can however be slightly alleviated for the reduced basis method with more stringent multi-core parallelization. We note that the linear map has the lowest online cost and only a slightly worse offline cost than the RBM. This makes it the most suitable method for linear operators such as those presented in Section 4.4 or for applications where larger levels of approximation error can be tolerated.

We again note that the image-to-image regression approach of (Zhu and Zabaras, 2018) does not scale with the mesh size. We do however acknowledge that for the small meshes for which the method was designed, it does outperform all other approaches. This begs the question of whether one can design neural networks that match the performance of image-to-image regression but remain invariant with respect to the size of the mesh. The contemporaneous work (Li et al., 2020) takes a step in this direction.

Lastly, we show that our method also has the ability to transfer a solution learned on one mesh to another. This is done by interpolating or sub-sampling both of the input and output PCA basis from the training mesh to the desired mesh. Justifying this requires a smoothness assumption on the PCA basis; we are, however, not aware of any such results and believe this is an interesting future direction. The neural network is fixed and does not need to be re-trained on a new mesh. We show this in Figure 410 for both Darcy flow problems. We note that when training on a small



(a) (b) (c)

Figure 411: Relative test errors on the Burgers' Equation problem. Using $N = 1024$ training examples, panel (a) shows the errors as a function of the resolution while panel (b) fixes a 4096 mesh and shows the error as a function of the reduced dimension. Panel (c) only shows results for our method using a neural network, fixing a 4096 mesh and showing the error as a function of the reduced dimension for different amounts of training data.

mesh, the error increases as we move to larger meshes, reflecting the interpolation error of the basis. Nevertheless, this increase is rather small: as shown in Figure 410, we obtain a 3% and a 1% relative error increasing when transferring solutions trained on a 61×61 grid to a 421×421 grid on each respective Darcy flow problem. On the other hand, when training on a large mesh, we see almost no error increase on the small meshes. This indicates that the neural network learns a property that is intrinsic to the solution operator and independent of the discretization.

Burgers' Equation

We now consider the input-output map $\Psi : L^2(\mathbb{T}^1;\mathbb{R}) \to H^r(\mathbb{T}^1;\mathbb{R})$ mapping $u_0 \mapsto u|_{t=1}$ in (4.25) with $\beta = 10^{-2}$ fixed. In this setting, Ψ is nonlinear and locally Lipschitz but since we do not know the precise Lipschitz constant as defined in Appendix 4.6, we cannot verify that the assumptions of Theorem 27 hold; nevertheless the numerical results demonstrate the effectiveness of our methodology. We take $u_0 \sim \mu = \mu_{\text{B}}$; see rows 5 of Figure 43 for an example. Figure 411 (a) shows the relative test errors as a function of the resolution again demonstrating that our method is invariant to mesh-refinement. We note that, for this problem, the linear map does significantly worse than the neural network in contrast to the Darcy flow problem where the results were comparable. This is likely attributable to the fact that the solution operator for Burgers' equation is more strongly nonlinear. As before, we observe from Figure 411 panel (b) that increasing the reduced dimension does not necessarily improve the error due to the increased complexity of the optimization problem. This can again be mitigated by increasing the volume of training data, as indicated in Figure 411(c); the curve of error versus reduced dimension is flattened as N increases.

4.5 Conclusion

In this paper, we proposed a general data-driven methodology that can be used to learn mappings between separable Hilbert spaces. We proved consistency of the approach when instantiated with PCA in the setting of globally Lipschitz forward maps. We demonstrated the desired mesh-independent properties of our approach on parametric PDE problems, showing good numerical performance even on problems outside the scope of the theory.

This work leaves many interesting directions open for future research. To understand the interplay between the reduced dimension and the amount of data needed requires a deeper understanding of neural networks and their interaction with the optimization algorithms used to produce the approximation architecture. Even if the optimal neural network is found by that optimization procedure, the question of the number of parameters needed to achieve a given level of accuracy, and how this interacts with the choice of reduced dimensions $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$ (choice of which is determined by the input space probability measure), warrants analysis in order to reveal the computational complexity of the proposed approach. Furthermore, the use of PCA limits the scope of problems that can be addressed to Hilbert, rather than general Banach spaces; even in Hilbert space, PCA may not be the optimal choice of dimension reduction. The development of autoenconders on function space is a promising direction that has the potential to address these issues; it also has many potential applications that are not limited to deployment within the methodology proposed here. Finally we also wish to study the use of our methodology in more challenging PDE problems, such as those arising in materials science, as well as for time-dependent problems such as multi-phase flow in porous media. Broadly speaking we view our contribution as a first step in the development of methods that generalize the ideas and applications of neural networks by operating on, and between, spaces of functions.

4.6 Neural Networks And Approximation (Locally Lipschitz Case)

This extends the approximation theory of Subsection 4.3 to the case of solution maps $\Psi : \mathcal{X} \to \mathcal{Y}$ that are μ-measurable and locally Lipschitz in the following sense

$$\forall x, z \in \mathcal{X} \quad \|\Psi(x) - \Psi(z)\|_{\mathcal{Y}} \leq L(x,z)\|x - z\|_{\mathcal{X}}. \tag{4.28}$$

where the function $L : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ is symmetric in its arguments, i.e., $L(x, z) = L(z, x)$, and for any fixed $w \in \mathcal{X}$ the function $L(\cdot, w) : \mathcal{X} \to \mathbb{R}_+$ is μ-measurable and non-decreasing in the sense that $L(s, w) \leq L(x, w)$ if $\|x\|_{\mathcal{X}} \geq \|w\|_{\mathcal{X}}$. Note that this implies that Ψ is locally bounded: for any $x \in \mathcal{X}$

$$\|\Psi(x)\|_{\mathcal{Y}} \leq \|\Psi(0)\|_{\mathcal{Y}} + \|\Psi(x) - \Psi(0)\|_{\mathcal{Y}} \leq \|\Psi(0)\|_{\mathcal{Y}} + L(x, 0)\|x\|_{\mathcal{X}}.$$

Hence we deduce that the pushforward $\Psi_\sharp\mu$ has bounded fourth moments provided that $\mathbb{E}_{x\sim\mu}(L(x, 0)\|x\|_{\mathcal{X}})^4 < +\infty$:

$$\begin{aligned}\mathbb{E}_{y\sim\Psi_\sharp\mu}\|y\|_{\mathcal{Y}}^4 &= \int_{\mathcal{X}} \|\Psi(x)\|_{\mathcal{Y}}^4 d\mu(x) \leq \int_{\mathcal{X}} (\|\Psi(0)\|_{\mathcal{Y}} + L(x, 0)\|x\|_{\mathcal{X}})^4 d\mu(x) \\ &\leq 2^3 \left(\|\Psi(0)\|_{\mathcal{Y}} + \int_{\mathcal{X}} L(x, 0)^4 \|x\|_{\mathcal{X}}^4 d\mu(x) \right) < \infty,\end{aligned}$$

where we used a generalized triangle inequality proven in (Takahasi et al., 2010, Cor. 3.1).

Theorem 27. *Let $\mathcal{X}$, $\mathcal{Y}$ be real, separable Hilbert spaces, Ψ a mapping from $\mathcal{X}$ into $\mathcal{Y}$ and let μ be a probability measure supported on $\mathcal{X}$ such that*

$$\mathbb{E}_{x\sim\mu}L(x, x)^2 < +\infty, \quad \mathbb{E}_{x\sim\mu}L(x, 0)^2\|x\|_{\mathcal{X}}^2 < \infty,$$

where $L(\cdot, \cdot)$ is given in (4.28). *Fix $d_{\mathcal{X}}, d_{\mathcal{Y}}$, $N \geq \max\{d_{\mathcal{X}}, d_{\mathcal{Y}}\}$, $\delta \in (0, 1)$, $\tau > 0$ and define $M = \sqrt{\mathbb{E}_{x\sim\mu}\|x\|_{\mathcal{X}}^2/\delta}$. Then there exists a constant $c(d_{\mathcal{X}}, d_{\mathcal{Y}}) \geq 0$ and neural network $\chi \in \mathcal{M}(d_{\mathcal{X}}, d_{\mathcal{Y}}; t, r, M)$ with $t \leq c(\log(\sqrt{d_{\mathcal{Y}}}/\tau) + 1)$ layers and $r \leq c(\epsilon^{-d_{\mathcal{X}}} \log(\sqrt{d_{\mathcal{Y}}}/\tau) + 1)$ active weights and biases in each component, so that*

$$\begin{aligned}\mathbb{E}_{\{x_j\}\sim\mu}\mathbb{E}_{x\sim\mu}\big(e_{NN}(x)\big) \leq C\Bigg(&\tau + \sqrt{\delta} \\ &+ \left(\sqrt{\frac{d_{\mathcal{X}}}{N}} + R^{\mu}(V_{d_{\mathcal{X}}}^{\mathcal{X}})\right)^{1/2} + \left(\sqrt{\frac{d_{\mathcal{Y}}}{N}} + R^{\Psi_\sharp\mu}(V_{d_{\mathcal{Y}}}^{\mathcal{Y}})\right)^{1/2}\Bigg),\end{aligned} \tag{4.29}$$

where $e_{NN}(x) := \|\Psi_{NN}(x) - \Psi(x)\|_{\mathcal{Y}}$ and $C > 0$ is independent of $d_{\mathcal{X}}, d_{\mathcal{Y}}, N, \delta$ and ϵ.

Proof. Our method of proof is similar to the proof of Theorem 26 and for this reason we shorten some of the arguments. Recall the constant Q from Theorem 25. In what follows we take Q to be the maximum of the two such constants when arising from application of the theorem on the two different probability spaces $(\mathcal{X}, \mu)$

and $(\mathcal{Y}, \Psi_\sharp\mu)$. As usual we employ the shorthand notation $\mathbb{E}$ to denote $\mathbb{E}_{\{x_j\}\sim\mu}$ the expectation with respect to the dataset $\{x_j\}_{j=1}^N$.

We begin by approximating the error incurred by using Ψ_{PCA} given by (4.4):

$$\begin{aligned}
\mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{PCA}(x)-\Psi(x)\|_{\mathcal{Y}} &= \mathbb{E}\mathbb{E}_{x\sim\mu}\|(G_{\mathcal{Y}}\circ F_{\mathcal{Y}}\circ\Psi\circ G_{\mathcal{X}}\circ F_{\mathcal{X}})(x)-\Psi(x)\|_{\mathcal{Y}} \\
&= \mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x)-\Psi(x)\right\|_{\mathcal{Y}} \\
&\le \mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x)-\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(x)\right\|_{\mathcal{Y}}+\mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}\Psi(x)-\Psi(x)\right\|_{\mathcal{Y}} \\
&\le \mathbb{E}\mathbb{E}_{x\sim\mu}L(\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x,x)\left\|\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x-x\right\|_{\mathcal{X}}+\mathbb{E}\mathbb{E}_{y\sim\Psi_\sharp\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}y-y\right\|_{\mathcal{Y}},
\end{aligned} \tag{4.30}$$

noting that the operator norm of an orthogonal projection is 1. Now since $L(\cdot,x)$ is non-decreasing we infer that $L(\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x,x)\le L(x,x)$ and then using Cauchy-Schwarz we obtain

$$\begin{aligned}
\mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{PCA}(x)-\Psi(x)\|_{\mathcal{Y}} &\le \left(\mathbb{E}_{x\sim\mu}|L(x,x)|^2\right)^{1/2}\left(\mathbb{E}\mathbb{E}_{x\sim\mu}\left\|\Pi_{V^{\mathcal{X}}_{d_{\mathcal{X}},N}}x-x\right\|_{\mathcal{X}}^2\right)^{1/2} \\
&\quad+\mathbb{E}\mathbb{E}_{y\sim\Psi_\sharp\mu}\left\|\Pi_{V^{\mathcal{Y}}_{d_{\mathcal{Y}},N}}y-y\right\|_{\mathcal{Y}}, \\
&= L'\left(\mathbb{E}R^{\mu}(V^{\mathcal{X}}_{d_{\mathcal{X}},N})\right)^{1/2}+\left(\mathbb{E}[R^{\Psi_\sharp\mu}(V^{\mathcal{Y}}_{d_{\mathcal{Y}},N})]\right)^{1/2}
\end{aligned} \tag{4.31}$$

where we used Hölder's inequality in the last line and defined the new constant $L':=\left(\mathbb{E}_{x\sim\mu}|L(x,x)|^2\right)^{1/2}$. Theorem 25 allows us to control this error, and leads to

$$\begin{aligned}
\mathbb{E}\mathbb{E}_{x\sim\mu}e_{NN} &\le \mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{NN}(x)-\Psi_{PCA}(x)\|_{\mathcal{Y}}+\mathbb{E}\mathbb{E}_{x\sim\mu}\|\Psi_{PCA}(x)-\Psi(x)\|_{\mathcal{Y}} \\
&\le \mathbb{E}_{x\sim\mu}\|\Psi_{NN}(x)-\Psi_{PCA}(x)\|_{\mathcal{Y}} \\
&\quad+L'\left(\sqrt{\frac{Qd_{\mathcal{X}}}{N}}+R^{\mu}(V^{\mathcal{X}}_{d_{\mathcal{X}}})\right)^{1/2}+\left(\sqrt{\frac{Qd_{\mathcal{Y}}}{N}}+R^{\Psi_\sharp\mu}(V^{\mathcal{Y}}_{d_{\mathcal{Y}}})\right)^{1/2}.
\end{aligned} \tag{4.32}$$

We now approximate φ by a neural network χ as before. To that end we first note from Lemma 30 that φ is locally Lipschitz, and hence continuous, as a mapping from $\mathbb{R}^{d_{\mathcal{X}}}$ into $\mathbb{R}^{d_{\mathcal{Y}}}$. Identify the components $\varphi(s)=(\varphi^{(1)}(s),\dots,\varphi^{(d_{\mathcal{Y}})}(s))$ where each function $\varphi^{(j)}\in C(\mathbb{R}^{d_{\mathcal{X}}};\mathbb{R})$. We consider the restriction of each component function to the set $[-M,M]^{d_{\mathcal{X}}}$.

By (Yarotsky, 2017, Thm. 1) and using the same arguments as in the proof of Theorem 26 there exist neural networks $\chi^{(1)},\dots,\chi^{(d_{\mathcal{Y}})}:\mathbb{R}^{d_{\mathcal{X}}}\to\mathbb{R}$, $\chi^{(j)}\in\mathcal{M}(d_{\mathcal{X}};t^{(j)},r^{(j)})$, with layer and active weight parameters $t^{(j)}$ and $r^{(j)}$ satisfying

$$t^{(j)} \leq c^{(j)}[\log(M\sqrt{d_{\mathcal{Y}}}/\tau) + 1],$$

and

$$r^{(j)} \leq c^{(j)}(\tau/2M)^{-d_{\mathcal{X}}}[\log(M\sqrt{d_{\mathcal{Y}}}/\tau) + 1]$$

with constants $c^{(j)}(d_{\mathcal{X}}) > 0$, so that

$$\left|\left(\chi^{(1)}(s), \dots, \chi^{(d_{\mathcal{Y}})}(s)\right) - \varphi(s)\right|_2 < \tau \quad \forall s \in [-M, M]^{d_{\mathcal{X}}}.$$

We now simply define $\chi : \mathbb{R}^{d_{\mathcal{X}}} \to \mathbb{R}^{d_{\mathcal{Y}}}$ as the stacked network $(\chi^{(1)}, \dots, \chi^{d_{\mathcal{Y}}})$ extended by zero outside of $[-M, M]^{d_{\mathcal{X}}}$ to immediately obtain

$$\sup_{s \in [-M,M]^{d_{\mathcal{X}}}} \left|\chi(s) - \varphi(s)\right|_2 < \tau. \tag{4.33}$$

Thus, by construction $\chi \in \mathcal{M}(d_{\mathcal{X}}, d_{\mathcal{Y}}, t, r, M)$ with at most $t \leq \max_j t^{(j)}$ many layers and $r \leq r^{(j)}$ many active weights and biases in each of its components.

Define the set $A = \{x \in \mathcal{X} : F_{\mathcal{X}}(x) \in [-M, M]^{d_{\mathcal{X}}}\}$. By Lemma 31 below, $\mu(A) \geq 1 - \delta$ and $\mu(A^c) \leq \delta$. Define the approximation error $e_{PCA}(x) := \|\Psi_{NN}(x) - \Psi_{PCA}(x)\|_{\mathcal{Y}}$ and decompose its expectation as

$$\mathbb{E}_{x\sim\mu}[e_{PCA}(x)] = \int_A e_{PCA}(x)d\mu(x) + \int_{A^c} e_{PCA}(x)d\mu(x) =: I_A + I_{A^c}.$$

For the first term,

$$I_A \leq \int_A \|(G_{\mathcal{Y}} \circ \chi \circ F_{\mathcal{X}})(x) - (G_{\mathcal{Y}} \circ \varphi \circ F_{\mathcal{X}})(x)\|_{\mathcal{Y}} d\mu(x) \leq \tau, \tag{4.34}$$

by using the fact, established in Lemma 30, that $G_{\mathcal{Y}}$ is Lipschitz with Lipschitz constant 1, the τ-closeness of χ to φ from (4.33), and $\mu(A) \leq 1$. For the second term we have, using that $G_{\mathcal{Y}}$ has Lipschitz constant 1 and that χ takes value zero on A^c,

$$\begin{aligned} I_{A^c} &\leq \int_{A^c} \|(G_{\mathcal{Y}} \circ \chi \circ F_{\mathcal{X}})(x) - (G_{\mathcal{Y}} \circ \varphi \circ F_{\mathcal{X}})(x)\|_{\mathcal{Y}} d\mu(x) \\ &\leq \int_{A^c} |\chi(F_{\mathcal{X}}(x)) - \varphi(F_{\mathcal{X}}(x))|_2 d\mu(x) = \int_{A^c} |\varphi(F_{\mathcal{X}}(x))|_2 d\mu(x). \end{aligned} \tag{4.35}$$

Once more from Lemma 30 we have that

$$|F_{\mathcal{X}}(x)|_2 \leq \|x\|_{\mathcal{X}}; \quad |\varphi(v)|_2 \leq |\varphi(0)|_2 + L(G_{\mathcal{X}}(v), 0)|v|_2$$

so that, using the hypothesis on L and global Lipschitz property of $F_{\mathcal{X}}$ and $G_{\mathcal{X}}$ we can write

$$\begin{aligned} I_{A^c} &\leq \mu(A^c)|\varphi(0)|_2 + \int_{A^c} L(x,0)|F_{\mathcal{X}}(x)|_2 d\mu(x) \\ &\leq \mu(A^c)|\varphi(0)|_2 + \int_{A^c} L(x,0)\|x\|_{\mathcal{X}} d\mu(x) \\ &\leq \mu(A^c)|\varphi(0)|_2 + \mu(A^c)^{\frac{1}{2}} \left(\mathbb{E}_{x\sim\mu} L(x,0)^2\|x\|_{\mathcal{X}}^2\right)^{1/2} \\ &\leq \delta|\varphi(0)|_2 + \delta^{\frac{1}{2}} L'' \\ &\leq \sqrt{\delta}(|\varphi(0)|_2 + L'') \end{aligned} \tag{4.36}$$

where $L'' := (\mathbb{E}_{x\sim\mu} L(x,0)^2\|x\|_{\mathcal{X}}^2)^{1/2}$. Combining (4.20), (4.22), and (4.24) we obtain the desired result. $\square$

4.7 Supporting Lemmas

In this Subsection we present and prove auxiliary lemmas that are used throughout the proofs in the article. The proof of Theorem 25 made use of the following proposition, known as Fan's Theorem, proved originally in (Fan, 1949). We state and prove it here in the infinite-dimensional setting as this generalization may be of independent interest. Our proof follows the steps of Fan's original proof in the finite-dimensional setting. The work (Overton and Womersley, 1992), through which we first became aware of Fan's result, gives an elegant generalization; however it is unclear whether that approach is easily applicable in infinite dimensions due to issues of compactness.

Lemma 28 (Fan (Fan, 1949))**.** *Let $(\mathcal{H}, \langle\cdot,\cdot\rangle, \|\cdot\|)$ be a separable Hilbert space and $C : \mathcal{H} \to \mathcal{H}$ a non-negative, self-adjoint, compact operator. Denote by $\lambda_1 \geq \lambda_2 \geq \dots$ the eigenvalues of C and, for any $d \in \mathbb{N}\setminus\{0\}$, let S_d denote the set of collections of d orthonormal elements of $\mathcal{H}$. Then*

$$\sum_{j=1}^{d} \lambda_j = \max_{\{u_1,\dots,u_d\}\in S_d} \sum_{j=1}^{d} \langle Cu_j, u_j\rangle.$$

Proof. Let $\phi_1, \phi_2, \dots$ denote the orthonormal eigenfunctions of C corresponding to the eigenvalues $\lambda_1, \lambda_2, \dots$ respectively. Note that for $\{\phi_1, \dots, \phi_d\} \in S_d$, we have

$$\sum_{j=1}^{d} \langle C\phi_j, \phi_j\rangle = \sum_{j=1}^{d} \lambda_j \|\phi_j\|^2 = \sum_{j=1}^{d} \lambda_j.$$

Now let $\{u_1, \dots, u_d\} \in S_d$ be arbitrary. Then for any $j \in \{1, \dots, d\}$, we have $u_j = \sum_{k=1}^{\infty} \langle u_j, \phi_k \rangle \phi_k$ and thus

$$\begin{aligned}\langle Cu_j, u_j \rangle &= \sum_{k=1}^{\infty} \lambda_k |\langle u_j, \phi_k \rangle|^2 \\ &= \lambda_d \sum_{k=1}^{\infty} |\langle u_j, \phi_k \rangle|^2 + \sum_{k=d+1}^{\infty} (\lambda_k - \lambda_d) |\langle u_j, \phi_k \rangle|^2 + \sum_{k=1}^{d} (\lambda_k - \lambda_d) |\langle u_j, \phi_k \rangle|^2.\end{aligned}$$

Since $\|u_j\|^2 = 1$, we have $\|u_j\|^2 = \sum_{k=1}^{\infty} |\langle u_j, \phi_k \rangle|^2 = 1$ therefore

$$\begin{aligned}\lambda_d \sum_{k=1}^{\infty} |\langle u_j, \phi_k \rangle|^2 + \sum_{k=d+1}^{\infty} (\lambda_k - \lambda_d) |\langle u_j, \phi_k \rangle|^2 &= \lambda_d \sum_{k=1}^{d} |\langle u_j, \phi_k \rangle|^2 + \sum_{k=d+1}^{\infty} \lambda_k |\langle u_j, \phi_k \rangle|^2 \\ &\leq \lambda_d \sum_{k=1}^{\infty} |\langle u_j, \phi_k \rangle|^2 = \lambda_d\end{aligned}$$

using the fact that $\lambda_k \leq \lambda_d, \forall k > d$. We have shown $\langle Cu_j, u_j \rangle \leq \lambda_d + \sum_{k=1}^{d} (\lambda_k - \lambda_d) |\langle u_j, \phi_k \rangle|^2$. Thus

$$\begin{aligned}\sum_{j=1}^{d} (\lambda_j - \langle Cu_j, u_j \rangle) &\geq \sum_{j=1}^{d} \left(\lambda_j - \lambda_d - \sum_{k=1}^{d} (\lambda_k - \lambda_d) |\langle u_j, \phi_k \rangle|^2 \right) \\ &= \sum_{j=1}^{d} (\lambda_j - \lambda_d) \left(1 - \sum_{k=1}^{d} |\langle u_k, \phi_j \rangle|^2 \right).\end{aligned}$$

We now extend the finite set of $\{u_k\}_{k=1}^{d}$ from a $d-$dimensional orthonormal set to an orthonormal basis $\{u_k\}_{k=1}^{\infty}$ for $\mathcal{H}$. Note that $\lambda_j \geq \lambda_d, \forall j \leq d$ and that

$$\sum_{k=1}^{d} |\langle u_k, \phi_j \rangle|^2 \leq \sum_{k=1}^{\infty} |\langle u_k, \phi_j \rangle|^2 = \|\phi_j\|^2 = 1$$

therefore $\sum_{j=1}^{d} (\lambda_j - \langle Cu_j, u_j \rangle) \geq 0$ concluding the proof.

□

Theorem 25 relies on a Monte Carlo estimate of the Hilbert-Schmidt distance between C and C_N that we state and prove below.

Lemma 29. *Let C be given by* (4.13) *and C_N by* (4.7) *then there exists a constant $Q \geq 0$, depending only on ν, such that*

$$\mathbb{E}_{\{u_j\} \sim \nu} \|C_N - C\|_{HS}^2 = \frac{Q}{N}.$$

Proof. Define $C^{(u_j)} := u_j \otimes u_j$ for any $j \in \{1, \dots, N\}$ and $C^{(u)} := u \otimes u$ for any $u \in \mathcal{H}$, noting that $\mathbb{E}_{u\sim\nu}[C^{(u)}] = C = \mathbb{E}_{u_j\sim\nu}[C^{(u_j)}]$. Further we note that

$$\mathbb{E}_{u\sim\nu}\|C^{(u)}\|_{HS}^2 = \mathbb{E}_{u\sim\nu}\|u\|^4 < \infty$$

and, by Jensen's inequality, $\|C\|_{HS}^2 \le \mathbb{E}_{u\sim\nu}\|C^{(u)}\|_{HS}^2 < \infty$. Once again using the shorthand notation $\mathbb{E}$ in place of $\mathbb{E}_{\{u_j\}\sim\nu}$ we compute,

$$\begin{aligned}\mathbb{E}\|C_N - C\|_{HS}^2 = \mathbb{E}\|\frac{1}{N}\sum_{j=1}^{N} C^{(u_j)} - C\|_{HS}^2 = \mathbb{E}\|\frac{1}{N}\sum_{j=1}^{N} C^{(u_j)}\|_{HS}^2 - \|C\|_{HS}^2 \\ = \frac{1}{N}\mathbb{E}_{u\sim\nu}\|C^{(u)}\|_{HS}^2 + \frac{1}{N^2}\sum_{j=1}^{N}\sum_{k\neq j}^{N}\langle \mathbb{E}[C^{(u_j)}], \mathbb{E}[C^{(u_k)}]\rangle_{HS} - \|C\|_{HS}^2 \\ = \frac{1}{N}\mathbb{E}_{u\sim\nu}\|C^{(u)}\|_{HS}^2 + \frac{N^2 - N}{N^2}\|C\|_{HS}^2 - \|C\|_{HS}^2 \\ = \frac{1}{N}\big(\mathbb{E}_{u\sim\nu}\|C^{(u)}\|_{HS}^2 - \|C\|_{HS}^2\big) = \frac{1}{N}\mathbb{E}_{u\sim\nu}\|C^{(u)} - C\|_{HS}^2.\end{aligned}$$

Setting $Q = \mathbb{E}_{u\sim\nu}\|C^{(u)} - C\|_{HS}^2$ completes the proof. $\square$

The following lemma, used in the proof of Theorem 26, estimates Lipschitz constants of various maps required in the proof.

Lemma 30. *The maps $F_\mathcal{X}$, $F_\mathcal{Y}$, $G_\mathcal{X}$ and $G_\mathcal{Y}$ are globally Lipschitz:*

$$\begin{aligned}|F_\mathcal{X}(v) - F_\mathcal{X}(z)|_2 \le \|v - z\|_\mathcal{X}, \quad \forall v, z \in \mathcal{X} \\ |F_\mathcal{Y}(v) - F_\mathcal{Y}(v)|_2 \le \|v - z\|_\mathcal{Y}, \quad \forall v, z \in \mathcal{Y} \\ \|G_\mathcal{X}(v) - G_\mathcal{X}(z)\|_\mathcal{X} \le |v - z|_2, \quad \forall v, z \in \mathbb{R}^{d_\mathcal{X}} \\ \|G_\mathcal{Y}(v) - G_\mathcal{Y}(z)\|_\mathcal{X} \le |v - z|_2, \quad \forall v, z \in \mathbb{R}^{d_\mathcal{Y}}.\end{aligned}$$

Furthermore, if Ψ is locally Lipschitz and satisfies

$$\forall x, w \in \mathcal{X} \quad \|\Psi(x) - \Psi(w)\|_\mathcal{Y} \le L(x, w)\|x - w\|_\mathcal{X},$$

with $L : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ that is symmetric with respect to its arguments, and increasing in the sense that $L(s, w) \le L(x, w)$, if $\|x\|_\mathcal{X} \ge \|s\|_\mathcal{X}$. Then φ is also locally Lipschitz and

$$|\varphi(v) - \varphi(z)|_2 \le L(G_\mathcal{X}(v), G_\mathcal{X}(z))|v - z|_2, \quad \forall v, z \in \mathbb{R}^{d_\mathcal{X}}.$$

Proof. We establish that $F_{\mathcal{Y}}$ and $G_{\mathcal{X}}$ are Lipschitz and estimate the Lipschitz constants; the proofs for $F_{\mathcal{X}}$ and $G_{\mathcal{Y}}$ are similar. Let $\phi_{1,N}^{\mathcal{Y}}, \dots, \phi_{d_{\mathcal{Y}},N}^{\mathcal{Y}}$ denote the eigenvectors of the empirical covariance with respect to the data $\{y_j\}_{j=1}^N$ which span $V_{d_{\mathcal{Y}},N}^{\mathcal{Y}}$ and let $\phi_{d_{\mathcal{Y}}+1,N}^{\mathcal{Y}}, \phi_{d_{\mathcal{Y}}+2,N}^{\mathcal{Y}}, \dots$ be an orthonormal extension to $\mathcal{Y}$. Then, by Parseval's identity,

$$|F_{\mathcal{Y}}(v) - F_{\mathcal{Y}}(z)|_2^2 = \sum_{j=1}^{d_{\mathcal{Y}}} \langle v - z, \phi_{j,N}^{\mathcal{Y}} \rangle_{\mathcal{Y}}^2 \leq \sum_{j=1}^{\infty} \langle v - z, \phi_{j,N}^{\mathcal{Y}} \rangle_{\mathcal{Y}}^2 = \|v - z\|_{\mathcal{Y}}^2.$$

A similar calculation for $G_{\mathcal{X}}$, using $\phi_{1,N}^{\mathcal{X}}, \dots, \phi_{d_{\mathcal{X}},N}^{\mathcal{X}}$ the eigenvectors of the empirical covariance of the data $\{x_j\}_{j=1}^N$ yields

$$\|G_{\mathcal{X}}(v) - G_{\mathcal{X}}(z)\|_{\mathcal{X}}^2 = \| \sum_{j=1}^{d_{\mathcal{X}}} (v_j - z_j) \phi_{j,N}^{\mathcal{X}} \|_{\mathcal{X}}^2 = \sum_{j=1}^{d_{\mathcal{X}}} |v_j - z_j|^2 = |v - z|_2^2$$

for any $v, z \in \mathbb{R}^{d_{\mathcal{X}}}$, using the fact that the empirical eigenvectors can be extended to an orthonormal basis for $\mathcal{X}$. Recalling that $\varphi = F_{\mathcal{Y}} \circ \Psi \circ G_{\mathcal{X}}$, the above estimates immediately yield

$$|\varphi(v) - \varphi(z)|_2 \leq L(G_{\mathcal{X}}(v), G_{\mathcal{X}}(z)) |v - z|_2, \quad \forall v, z \in \mathbb{R}^{d_{\mathcal{X}}}.$$

□

The following lemma establishes a bound on the size of the set A that was defined in the proof of Theorems 26 and 27.

Lemma 31. *Fix $0 < \delta < 1$, let $x \sim \mu$ be a random variable and set $M = \sqrt{\mathbb{E}_{x \sim \mu} \|x\|_{\mathcal{X}}^2 / \delta}$. Define $F_{\mathcal{X}}$ using the random dataset $\{x_j\}_{j=1}^N \sim \mu$ then,*

$$\mathbb{P}\left(F_{\mathcal{X}}(x) \notin [-M, M]^{d_{\mathcal{X}}} \right) \leq \delta,$$

where the probability is computed with respect to both x and the x_j's.

Proof. Denote by $\phi_{1,N}^{\mathcal{X}}, \dots, \phi_{d_{\mathcal{X}},N}^{\mathcal{X}}$ the orthonormal set used to define $V_{d_{\mathcal{X}},N}^{\mathcal{X}}$ (4.8) and let $\phi_{d_{\mathcal{X}}+1,N}^{\mathcal{X}}, \phi_{d_{\mathcal{X}}+2,N}^{\mathcal{X}}, \dots$ be an orthonormal extension of this basis to $\mathcal{X}$. For any $j \in \{1, \dots, d_{\mathcal{X}}\}$, by Chebyshev's inequality, we have

$$\mathbb{P}_{x \sim \mu}(|\langle x, \phi_{j,N}^{\mathcal{X}} \rangle_{\mathcal{X}}| \geq M) \leq \frac{\mathbb{E}_{x \sim \mu} |\langle x, \phi_{j,N}^{\mathcal{X}} \rangle_{\mathcal{X}}|^2}{M^2}.$$

Note that the expectation is taken only with respect to the randomness in x and not $\phi^{\mathcal{X}}_{1,N}, \dots, \phi^{\mathcal{X}}_{d_{\mathcal{X}},N}$, so the right hand side is itself a random variable. We further compute

$$\begin{aligned}&\mathbb{P}_{x\sim\mu}(|\langle x, \phi^{\mathcal{X}}_{1,N}\rangle_{\mathcal{X}}| \geq M, \dots, |\langle x, \phi^{\mathcal{X}}_{d_{\mathcal{X}},N}\rangle_{\mathcal{X}}| \geq M)\\ &\qquad \leq \frac{1}{M^2}\mathbb{E}_{x\sim\mu}\sum_{j=1}^{d_{\mathcal{X}}}|\langle x, \phi^{\mathcal{X}}_{j,N}\rangle_{\mathcal{X}}|^2 \leq \frac{1}{M^2}\mathbb{E}_{x\sim\mu}\sum_{j=1}^{\infty}|\langle x, \phi^{\mathcal{X}}_{j,N}\rangle_{\mathcal{X}}|^2 = \frac{1}{M^2}\mathbb{E}_{x\sim\mu}\|x\|^2_{\mathcal{X}}\end{aligned}$$

noting that $\|x\|^2_{\mathcal{X}} = \sum_{j=1}^\infty |\langle x, \xi_j\rangle_{\mathcal{X}}|^2$ for any orthonormal basis $\{\xi_j\}_{j=1}^\infty$ of $\mathcal{X}$ hence the randomness in $\phi^{\mathcal{X}}_{1,N}, \phi^{\mathcal{X}}_{2,N}, \dots$ is inconsequential. Thus we find that, with $\mathbb{P}$ denoting probability with respect to both $x \sim \mu$ and the random data used to define $F_{\mathcal{X}}$,

$$\mathbb{P}(|\langle x, \phi^{\mathcal{X}}_{1,N}\rangle_{\mathcal{X}}| \leq M, \dots, |\langle x, \phi^{\mathcal{X}}_{d_{\mathcal{X}},N}\rangle_{\mathcal{X}}| \leq M) \geq 1 - \frac{1}{M^2}\mathbb{E}_{x\sim\mu}\|x\|^2_{\mathcal{X}}, = 1-\delta$$

the desired result. $\square$

4.8 Analyticity of the Poisson Solution Operator

Define $\mathcal{X} = \{\xi \in \ell^\infty(\mathbb{N};\mathbb{R}) : \|\xi\|_{\ell^\infty} \leq 1\}$ and let $\{\phi_j\}_{j=1}^\infty$ be some sequence of functions with the property that $(\|\phi_j\|_{L^\infty})_{j\geq 1} \in \ell^p(\mathbb{N};\mathbb{R})$ for some $p \in (0,1)$. Define $\Psi : \mathcal{X} \to H^1_0(D;\mathbb{R})$ as mapping a set of coefficients $\xi = (\xi_1, \xi_2, \dots) \in \mathcal{X}$ to $u \in H^1_0(D;\mathbb{R})$ the unique weak solution of

$$-\Delta u = \sum_{j=1}^\infty \xi_j \phi_j \quad \text{in} \quad D, \qquad u|_{\partial D} = 0.$$

Note that since D is a bounded domain and $\xi \in \mathcal{X}$, we have that $\sum_{j=1}^\infty \xi_j\phi_j \in L^2(\Omega;\mathbb{R})$ since our assumption implies $(\|\phi_j\|_{L^\infty})_{j\geq 1} \in \ell^1(\mathbb{N};\mathbb{R})$. Therefore u is indeed the unique weak-solution of the Poisson equation (Evans, 2010, Chap. 6) and Ψ is well-defined.

Theorem 32. *Suppose that there exists $p \in (0,1)$ such that $(\|\phi_j\|_{L^\infty})_{j\geq 1} \in \ell^p(\mathbb{N};\mathbb{R})$. Then*

$$\lim_{K\to\infty}\sup_{\xi\in\mathcal{X}}\|\Psi(\xi) - \sum_{j=1}^K \xi_j\eta_j\|_{H^1_0} = 0$$

where, for each $j \in \mathbb{N}$, $\eta_j \in H^1_0(D;\mathbb{R})$ satisfies

$$-\Delta \eta_j = \phi_j \quad \textit{in} \quad D, \qquad u|_{\partial D} = 0.$$

Proof. By linearity and Poincaré inequality, we obtain the Lipschitz estimate

$$\|\Psi(\xi^{(1)}) - \Psi(\xi^{(2)})\|_{H_0^1} \leq C\| \sum_{j=1}^{\infty}(\xi_j^{(1)} - \xi_j^{(2)})\phi_j\|_{L^2}, \quad \forall \xi^{(1)}, \xi^{(2)} \in \mathcal{X}$$

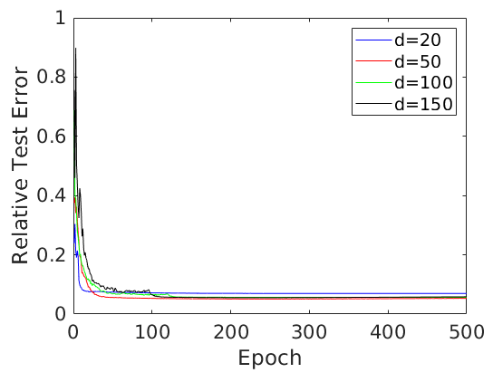
for some $C > 0$. Now let $\xi = (\xi_1, \xi_2, \dots) \in \mathcal{X}$ be arbitrary and define the sequence $\xi^{(1)} = (\xi_1, 0, 0, \dots), \xi^{(2)} = (\xi_1, \xi_2, 0, \dots), \dots$. Note that, by linearity, for any $K \in \mathbb{N}$, $\Psi(\xi^{(K)}) = \sum_{j=1}^{K} \xi_j \eta_j$. Then, using our Lipschitz estimate,

$$\|\Psi(\xi) - \Psi(\xi^K)\|_{H_0^1} \lesssim \| \sum_{j=K+1}^{\infty} \xi_j \phi_j\|_{L^2} \lesssim \sum_{j=K+1}^{\infty} \|\phi_j\|_{L^\infty} \leq K^{1-\frac{1}{p}} \|(\|\phi_j\|_{L^\infty})_{j\geq 1}\|_{\ell^p}$$

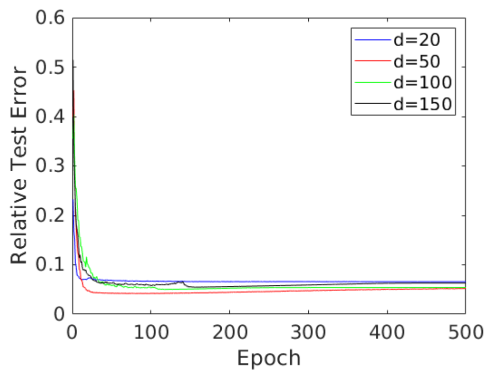
where the last line follows by Stechkin's inequality (Cohen, DeVore, and Schwab, 2011, Sec. 3.3). Taking the supremum over $\xi \in \mathcal{X}$ and the limit $K \to \infty$ completes the proof. □

4.9 Error During Training

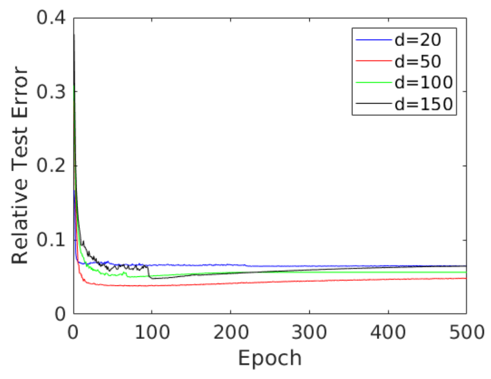
Figures 412 and 413 show the relative test error computed during the training process for the problems presented in Section 4.4. For both problems, we observe a slight amount of overfitting when more training samples are used and the reduced dimension is sufficiently large. This is because the true map of interest φ is linear while the neural network parameterization is highly non-linear hence more prone to overfitting larger amounts of data. While this suggests that simpler neural networks might perform better on this problem, we do not carry out such experiments as our goal is simply to show that building in *a priori* information about the problem (here linearity) can be beneficial as show in Figures 44 and 45.



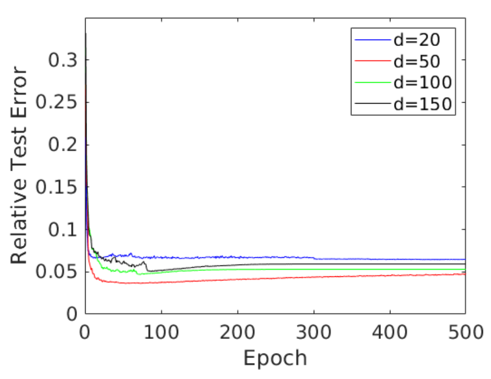
(a) $N = 1000$



(b) $N = 2000$

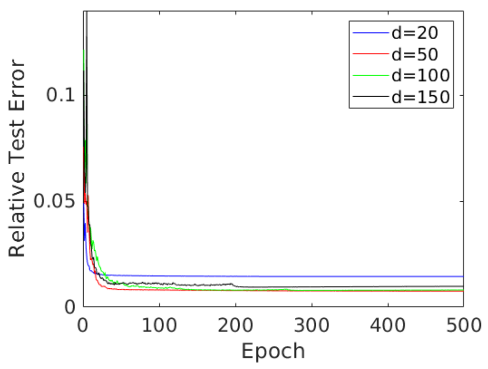


(c) $N = 4000$

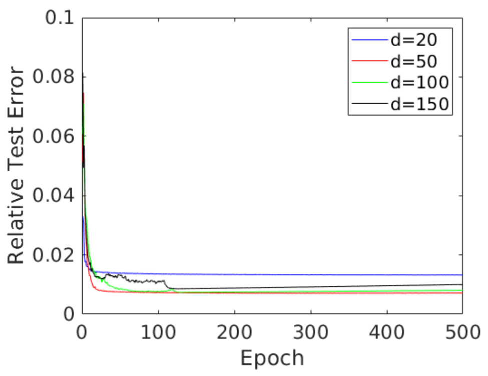


(d) $N = 5000$

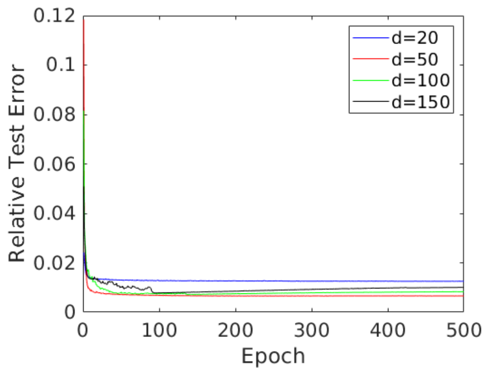
Figure 412: Relative test errors as a function of the training epoch on the linear elliptic problem. The amount of training examples used is varied in each panel.



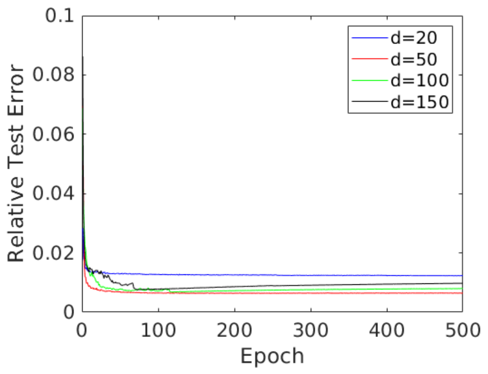
(a) $N = 1000$



(b) $N = 2000$



(c) $N = 4000$



(d) $N = 5000$

Figure 413: Relative test errors as a function of the training epoch on the Poisson problem. The amount of training examples used is varied in each panel.

Chapter 5

NEURAL OPERATORS: APPROXIMATING MAPS BETWEEN FUNCTION SPACES

5.1 Introduction

Learning mappings between infinite-dimensional function spaces is a challenging problem with widespread potential application across various disciplines. Examples include numerous differential equation models in science and engineering, in robotics and in computer vision. Of particular interest are cases where the inputs and/or outputs are themselves functions over a domain in Euclidean space. The possibility of learning such mappings opens up a new class of problems in the design of neural networks with widespread applicability. New ideas are required to build upon traditional neural networks which are mappings between finite-dimensional Euclidean spaces and/or sets of finite cardinality.

A naive approach to this problem is simply to discretize the (input or output) function spaces and apply standard ideas from neural networks. Instead we formulate a new class of deep neural network based models, called neural operators, which directly map between spaces of functions on bounded domains $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$; once designed on function space these maps can be discretized by a variety of different methods, and at different levels of resolution, to produce families of finite dimensional networks. Such models, once trained, have the property of being discretization invariant: it is possible to share the same network parameters between different discretizations of the underlying functional data. In contrast, the naive approach leads to neural network architectures which depend heavily on this discretization: new architectures with new parameters are needed to achieve the same error for differently discretized data. We demonstrate, numerically, that the same neural operator can achieve a constant error for any discretization of the data while standard feed-forward and convolutional neural networks cannot. In the context of partial differential equations (PDEs) we demonstrate numerically that, at fixed resolution, the resulting methods are highly competitive when compared with other neural networks and are orders of magnitude faster than the PDE solvers used to generate data. Finally we establish a universal approximation theorem for the neural operators we introduce, proving their ability to approximate linear and non-linear operators arbitrary well.

In this paper we study various solution operators or flow maps arising from PDE models; in particular, we investigate mappings between function spaces where the input data can be, for example, the initial condition, boundary condition, or coefficient function, and the output data is the respective solution. We perform numerical experiments with operators arising from the one-dimensional Poisson equation (Evans, 2010), the time-dependent one-space-dimensional Burgers' Equation (Evans, 2010), two-dimensional steady Darcy Flow (Bear and Corapcioglu, 2012) and the time-dependent two-space dimensional incompressible Navier-Stokes Equation (Constantin and Foias, 1988; Lemarié-Rieusset, 2018; Temam, 2001).

Subsection 5.1 contains background and context for our work. Subsection 5.1 details our contributions and outlines the contents of the paper. We conclude this introduction in Subsection 5.1 which provides a literature review.

Background and Context

PDEs. "Differential equations [...] represent the most powerful tool humanity has ever created for making sense of the material world." Strogatz (2009). Over the past few decades, significant progress has been made on formulating (Gurtin, 1982) and solving (Johnson, 2012) the governing PDEs in many scientific fields from micro-scale problems (e.g., quantum and molecular dynamics) to macro-scale applications (e.g., civil and marine engineering). Despite the success in the application of PDEs to solve real-world problems, two significant challenges remain:

- identifying the governing model for complex systems;
- efficiently solving large-scale nonlinear systems of equations.

Identifying/formulating the underlying PDEs appropriate for modeling a specific problem usually requires extensive prior knowledge in the corresponding field which is then combined with universal conservation laws to design a predictive model. For example, modeling the deformation and failure of solid structures requires detailed knowledge of the relationship between stress and strain in the constituent material. For complicated systems such as living cells, acquiring such knowledge is often elusive and formulating the governing PDE for these systems remains prohibitive, or the models proposed are too simplistic to be informative. The possibility of acquiring such knowledge from data can revolutionize these fields. Second, solving complicated nonlinear PDE systems (such as those arising in turbulence and plasticity) is computationally demanding and can often make realistic simulations intractable. Again the possibility of using instances of data from such computations to design fast approximate solvers holds great potential for accelerating numerous problems in computational science and engineering and for revolutionizing scientific discovery.

Learning PDE Solution Operators. Supervised learning has the potential to address these challenges when designed in a way that allows for the emulation of mappings between function spaces (Khoo, J. Lu, and L. Ying, 2017; L. Lu, Jin, and George Em Karniadakis, 2019; Bhattacharya et al., 2020; Nelsen and Andrew M Stuart, 2021; Z. Li, Kovachki, et al., 2020a; Z. Li, Kovachki, et al., 2020b; Z. Li, Kovachki, et al., 2020c; Patel et al., 2021; Opschoor, Christoph Schwab, and Jakob Zech, 2020; Christoph Schwab and Jakob Zech, 2019; O'Leary-Roseberry et al., 2020; Wu and Xiu, 2020). In PDE applications, the governing equations are by definition local, whilst the solution operator exhibits non-local properties. Such non-local effects can be described by integral operators explicitly in the spatial domain, or by means of spectral domain multiplication; convolution is an archetypal example. For integral equations, the graph approximations of Nyström type (Belongie et al., 2002) provide a consistent way of connecting different grid or data structures arising in computational methods and understanding their continuum limits (Von Luxburg, Belkin, and Bousquet, 2008; Trillos and Slepvcev, 2018; Trillos, Gerlach, et al., 2020). For spectral domain calculations, there are well-developed tools that exist for approximating the continuum (Boyd, 2001; Trefethen, 2000). For these reasons, neural networks that incorporate non-locality via integral operators or spectral domain calculations are natural. This is the governing principle underlying our work aimed at designing mesh invariant neural network approximations for the solution operators of PDEs.

Our Contributions and Outline of The Paper

Neural Operators. We introduce the concept of neural operators by generalizing standard feed-forward neural networks to learn mappings between infinite-dimensional spaces of functions defined on bounded domains of $\mathbb{R}^d$. The non-local component of the architecture is instantiated through either a parameterized integral operator or through multiplication in the spectral domain. We make the following contributions.

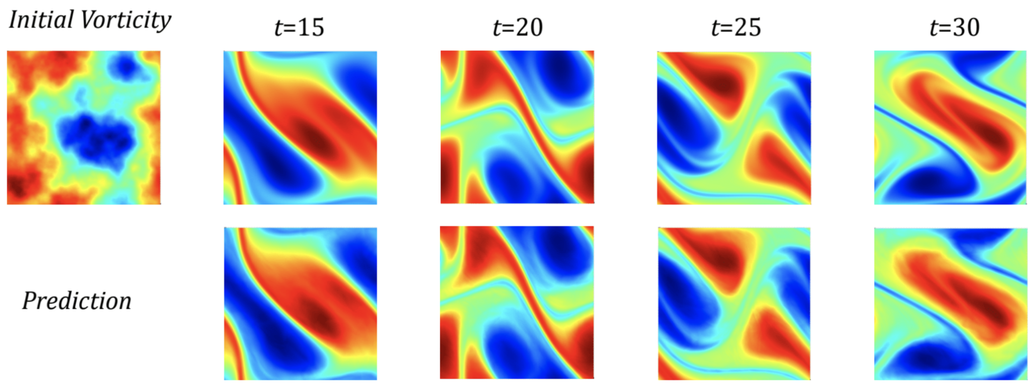


Figure 51: Zero-shot super-resolution. Vorticity field of the solution to the two-dimensional Navier-Stokes equation with viscosity 10^4 (Re=$O(200)$); Ground truth on top and prediction on bottom. The model is trained on data that is discretized on a uniform 64×64 spatial grid and on a 20-point uniform temporal grid. The model is evaluated with a different initial condition that is discretized on a uniform 256×256 spatial grid and a 80-point uniform temporal grid (see Section 5.7).

1. We propose neural operators, generalizing neural networks that map between finite-dimensional Euclidean spaces to neural networks that map between infinite-dimensional function spaces.

2. By construction, our architectures share the same parameters irrespective of the discretization used on the input and output spaces for the purposes of computation. Consequently, neural operators are capable of zero-shot super-resolution as demonstrated in Figure 51.

3. We propose four practical methods for implementing the neural operator framework: graph-based operators, low-rank operators, multipole graph-based operators, and Fourier operators. Specifically, we develop a Nyström extension to connect the integral operator formulation of the neural operator to families of graph neural networks (GNNs) on arbitrary grids. Furthermore, we study the spectral domain formulation of the neural operator which leads to efficient algorithms in settings where fast transform methods are applicable. We include an exhaustive numerical study of the four formulations.

4. Numerically, we show that the proposed methodology consistently outperforms all existing deep learning methods even on the resolutions for which the standard neural networks were designed. For the two-dimensional Navier-Stokes equation, when learning the entire flow map, the method achieves

$< 1\%$ error for a Reynolds number of 20 and 8% error for a Reynolds number of 200.

5. The Fourier neural operator (FNO) has an inference time that is three orders of magnitude faster than the pseudo-spectral method used to generate the data for the Navier-Stokes problem (Chandler and Kerswell, 2013) – 0.005s compared to the $2.2s$ on a 256×256 uniform spatial grid. Despite its tremendous speed advantage, the method does not suffer from accuracy degradation when used in downstream applications such as solving Bayesian inverse problems. Furthermore, we demonstrate that FNO is robust to noise on the testing problems we consider here.

In Section 5.2, we define the general operator learning problem, which is not limited to PDEs. In Section 5.3, we define the general framework in terms of kernel integral operators and relate our proposed approach to existing methods in the literature.

In Section 5.5, we define four different ways of efficiently computing with neural operators: graph-based operators (GNO), low-rank operators (LNO), multipole graph-based operators (MGNO), and Fourier operators (FNO). In Section 5.6 we define four partial differential equations which serve as a test-bed of various problems which we study numerically. In Section 4.4, we show the numerical results for our four approximation methods on the four test problems, and on two linear operators defined by linear PDEs, and we discuss and compare the properties, including robustness, of each method. In Section 4.5 we conclude the work, discuss potential limitations and outline directions for future work.

Literature Review

We outline the major neural network-based approaches for the solution of PDEs. To make the discussion concrete, we will consider the family of PDEs in the form

$$\begin{aligned} (\mathsf{L}_a u)(x) &= f(x), \qquad && x \in D, \\ u(x) &= 0, \qquad && x \in \partial D, \end{aligned} \tag{5.1}$$

for some $a \in \mathcal{A}$, $f \in \mathcal{U}^*$ and $D \subset \mathbb{R}^d$ a bounded domain. We assume that the solution $u : D \to \mathbb{R}$ lives in the Banach space $\mathcal{U}$ and $\mathsf{L}_a : \mathcal{A} \to \mathcal{L}(\mathcal{U};\mathcal{U}^*)$ is a mapping from the parameter Banach space $\mathcal{A}$ to the space of (possibly unbounded) linear operators mapping $\mathcal{U}$ to its dual $\mathcal{U}^*$. A natural operator which arises from this PDE is $\mathcal{G}^\dagger := \mathsf{L}_a^{-1} f : \mathcal{A} \to \mathcal{U}$ defined to map the parameter to the solution $a \mapsto u$. A simple example that we study further in Section 5.6 is when L_a is the weak form of the second-order elliptic operator $-\nabla \cdot (a\nabla)$ subject to homogeneous Dirichlet boundary conditions. In this setting, $\mathcal{A} = L^\infty(D; \mathbb{R}_+), \mathcal{U} = H_0^1(D; \mathbb{R})$, and $\mathcal{U}^* = H^{-1}(D; \mathbb{R})$. When needed, we will assume that the domain D is discretized into $K \in \mathbb{N}$ points and that we observe $N \in \mathbb{N}$ pairs of coefficient functions and (approximate) solution functions $\{a_j, u_j\}_{j=1}^N$ that are used to train the model (see Section 5.2). We assume that a_j are i.i.d. samples from a probability measure μ supported on $\mathcal{A}$ and y_j are the pushforwards under $\mathcal{G}^\dagger$.

Finite-dimensional Operators. An immediate approach to approximate $\mathcal{G}^\dagger$ is to parameterize it as a deep convolutional neural network (CNN) between the finite-dimensional Euclidean spaces on which the data is discretized i.e. $\mathcal{G} : \mathbb{R}^K \times \Theta \to \mathbb{R}^K$ (Guo, W. Li, and Iorio, 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019). Khoo, J. Lu, and L. Ying (2017) concerns a similar setting, but with output space $\mathbb{R}$. Such approaches are, by definition, not mesh independent and need modifications to the architecture for different resolution and discretization of D in order to achieve consistent error (if at all possible). We demonstrate this issue numerically in Section 4.4. Furthermore, these approaches are limited to the discretization size and geometry of the training data and hence it is not possible to query solutions at new points in the domain. In contrast for our method, we show in Section 4.4 both invariance of the error to grid resolution, and the ability to transfer the solution between meshes. The work Ummenhofer et al. (2020) proposed a continuous convolution network for fluid problems, where off-grid points are sampled and linearly interpolated. However the continuous convolution method is still constrained by the underlying grid which prevents generalization to higher resolutions. Similarly, to get finer resolution solution, Jiang et al. (2020) proposed learning super-resolution with a U-Net structure for fluid mechanics problems. However fine-resolution data is needed for training, while neural operators are capable of zero-shot super-resolution with no new data.

DeepONet A novel operator regression architecture, named DeepONet, was recently proposed by L. Lu, Jin, and George Em Karniadakis, 2019; L. Lu, Jin, Pang, et al., 2021; it builds an iterated or deep structure on top of the shallow architecture proposed in T. Chen and H. Chen, 1995. The architecture consists of two neural networks: a branch net applied on the input functions and a trunk net applied on the querying locations in the output space. The original work of T. Chen and H.

Chen, 1995 provides a universal approximation theorem, and more recently Lanthaler, Mishra, and George Em Karniadakis, 2021 developed an error estimate for DeepONet itself. The standard DeepONet structure is a linear approximation of the target operator, where the trunk net and branch net learn the coefficients and basis. On the other hand, the neural operator is a non-linear approximation, which makes it constructively more expressive. We include an detailed discussion of DeepONet in Section 5.3 and as well as a numerical comparison to DeepONet in Section 5.7.

Physics Informed Neural Networks (PINNs), Deep Ritz Method (DRM), and Deep Galerkin Method (DGM). A different approach is to directly parameterize the solution u as a neural network $u : \bar{D} \times \Theta \to \mathbb{R}$ (E and Yu, 2018; Raissi, Perdikaris, and George E Karniadakis, 2019; Sirignano and Spiliopoulos, 2018; Bar and Sochen, 2019; Smith, Azizzadenesheli, and Ross, 2020; Pan and Duraisamy, 2020). This approach is designed to model one specific instance of the PDE, not the solution operator. It is mesh-independent, but for any given new parameter coefficient function $a \in \mathcal{A}$, one would need to train a new neural network u_a which is computationally costly and time consuming. Such an approach closely resembles classical methods such as finite elements, replacing the linear span of a finite set of local basis functions with the space of neural networks.

ML-based Hybrid Solvers Similarly, another line of work proposes to enhance existing numerical solvers with neural networks by building hybrid models (Pathak et al., 2020; Um, Holl, et al., 2020; Greenfeld et al., 2019). These approaches suffer from the same computational issue as classical methods: one needs to solve an optimization problem for every new parameter similarly to the PINNs setting. Furthermore, the approaches are limited to a setting in which the underlying PDE is known. Purely data-driven learning of a map between spaces of functions is not possible.

Reduced Basis Methods. Our methodology most closely resembles the classical reduced basis method (RBM) (R. A. DeVore, 2014) or the method of Cohen and R. DeVore (2015). The method introduced here, along with the contemporaneous work introduced in the papers (Bhattacharya et al., 2020; Nelsen and Andrew M Stuart, 2021; Opschoor, Christoph Schwab, and Jakob Zech, 2020; Christoph Schwab and Jakob Zech, 2019; O'Leary-Roseberry et al., 2020; L. Lu, Jin, and George Em Karniadakis, 2019), is, to the best of our knowledge, amongst the first practical supervised learning methods designed to learn maps between infinite-dimensional spaces. It addresses the mesh-dependent nature of the approach in the papers (Guo, W. Li, and Iorio, 2016; Zhu and Zabaras, 2018; Adler and Oktem, 2017; Bhatnagar et al., 2019) by producing a single set of network parameters that can be used with different discretizations. Furthermore, it has the ability to transfer solutions between meshes and indeed between different discretization methods. Moreover, it need only be trained once on the equation set $\{a_j, u_j\}_{j=1}^N$. Then, obtaining a solution for a new $a \sim \mu$ only requires a forward pass of the network, alleviating the major computational issues incurred in (E and Yu, 2018; Raissi, Perdikaris, and George E Karniadakis, 2019; Herrmann, Ch Schwab, and Zech, 2020; Bar and Sochen, 2019) where a different network would need to be trained for each input parameter. Lastly, our method requires no knowledge of the underlying PDE: it is purely data-driven and therefore non-intrusive. Indeed the true map can be treated as a black-box, perhaps to be learned from experimental data or from the output of a costly computer simulation, not necessarily from a PDE.

Continuous Neural Networks. Using continuity as a tool to design and interpret neural networks is gaining currency in the machine learning community, and the formulation of ResNet as a continuous time process over the depth parameter is a powerful example of this (Haber and Ruthotto, 2017; Weinan, 2017). The concept of defining neural networks in infinite-dimensional spaces is a central problem that long been studied (Williams, 1996; Neal, 1996; Roux and Bengio, 2007; Globerson and Livni, 2016; Guss, 2016). The general idea is to take the infinite-width limit which yields a non-parametric method and has connections to Gaussian Process Regression (Neal, 1996; Matthews et al., 2018; Garriga-Alonso, Rasmussen, and Aitchison, 2018), leading to the introduction of deep Gaussian processes (Damianou and Lawrence, 2013; Dunlop et al., 2018). Thus far, such methods have not yielded efficient numerical algorithms that can parallel the success of convolutional or recurrent neural networks for the problem of approximating mappings between finite dimensional spaces. Despite the superficial similarity with our proposed work, this body of work differs substantially from what we are proposing: in our work we are motivated by the continuous dependence of the data, in the input or output spaces, in spatial or spatio-temporal variables; in contrast the work outlined in this paragraph uses continuity in an artificial algorithmic depth or width parameter to study the network architecture when the depth or width approaches infinity, but the input and output spaces remain of fixed finite dimension.

Nyström Approximation, GNNs, and Graph Neural Operators (GNOs). The graph neural operators (Section 5.5) has an underlying Nyström approximation formulation (Nyström, 1930) which links different grids to a single set of network parameters. This perspective relates our continuum approach to Graph Neural Networks (GNNs). GNNs are a recently developed class of neural networks that apply to graph-structured data; they have been used in a variety of applications. Graph networks incorporate an array of techniques from neural network design such as graph convolution, edge convolution, attention, and graph pooling (Kipf and Welling, 2016; Hamilton, Z. Ying, and Leskovec, 2017; Gilmer et al., 2017; Veličković et al., 2017; Murphy et al., 2018). GNNs have also been applied to the modeling of physical phenomena such as molecules (C. Chen et al., 2019) and rigid body systems (Battaglia et al., 2018) since these problems exhibit a natural graph interpretation: the particles are the nodes and the interactions are the edges. The work (Alet et al., 2019) performs an initial study that employs graph networks on the problem of learning solutions to Poisson's equation, among other physical applications. They propose an encoder-decoder setting, constructing graphs in the latent space, and utilizing message passing between the encoder and decoder. However, their model uses a nearest neighbor structure that is unable to capture non-local dependencies as the mesh size is increased. In contrast, we directly construct a graph in which the nodes are located on the spatial domain of the output function. Through message passing, we are then able to directly learn the kernel of the network which approximates the PDE solution. When querying a new location, we simply add a new node to our spatial graph and connect it to the existing nodes, avoiding interpolation error by leveraging the power of the Nyström extension for integral operators.

Low-rank Kernel Decomposition and Low-rank Neural Operators (LNOs). Low-rank decomposition is a popular method used in kernel methods and Gaussian process (Kulis, Sustik, and Dhillon, 2006; Bach, 2013; Lan et al., 2017; Gardner et al., 2018). We present the low-rank neural operator in Section 5.5 where we structure the kernel network as a product of two factor networks inspired by Fredholm theory. The low-rank method, while simple, is very efficient and easy to train especially when the target operator is close to linear. Khoo and L. Ying (2019) proposed a related neural network with low-rank structure to approximate the inverse of differential operators. The framework of two factor networks is also similar to the trunk and branch network used in DeepONet (L. Lu, Jin, and George Em Karniadakis, 2019). But in our work, the factor networks are defined on the physical domain and non-local information is accumulated through integration with respect to the Lebesgue measure. In contrast, DeepONet(s) integrate against delta measures at a set of pre-defined nodal points that are usually taken to be the grid on which the data is given. See section 5.3 for further discussion.

Multipole, Multi-resolution Methods, and Multipole Graph Neural Operators (MGNOs). To efficiently capture long-range interaction, multi-scale methods such as the classical fast multipole methods (FMM) have been developed (Greengard and Rokhlin, 1997). Based on the assumption that long-range interactions decay quickly, FMM decomposes the kernel matrix into different ranges and hierarchically imposes low-rank structures on the long-range components (hierarchical matrices) (Börm, Grasedyck, and Hackbusch, 2003). This decomposition can be viewed as a specific form of the multi-resolution matrix factorization of the kernel (Kondor, Teneva, and Garg, 2014; Börm, Grasedyck, and Hackbusch, 2003). For example, the works of Fan, Lin, et al. (2019), Fan, Feliu-Faba, et al. (2019), and J. He and Xu (2019) propose a similar multipole expansion for solving parametric PDEs on structured grids. However, the classical FMM requires nested grids as well as the explicit form of the PDEs. In Section 5.5, we propose the multipole graph neural operator (MGNO) by generalizing this idea to arbitrary graphs in the data-driven setting, so that the corresponding graph neural networks can learn discretization-invariant solution operators which are fast and can work on complex geometries.

Fourier Transform, Spectral Methods, and Fourier Neural Operators (FNOs). The Fourier transform is frequently used in spectral methods for solving differential equations since differentiation is equivalent to multiplication in the Fourier domain. Fourier transforms have also played an important role in the development of deep learning. They are used in theoretical work, such as the proof of the neural network universal approximation theorem (Hornik, Stinchcombe, White, et al., 1989) and related results for random feature methods (Rahimi and Recht, 2008); empirically, they have been used to speed up convolutional neural networks (Mathieu, Henaff, and LeCun, 2013). Neural network architectures involving the Fourier transform or the use of sinusoidal activation functions have also been proposed and studied (Bengio, LeCun, et al., 2007; Mingo et al., 2004; Sitzmann et al., 2020). Recently, some spectral methods for PDEs have been extended to neural networks (Fan, Bohorquez, and L. Ying, 2019; Fan, Lin, et al., 2019; Kashinath, Marcus, et al., 2020). In

Section 5.5, we build on these works by proposing the Fourier neural operator architecture defined directly in Fourier space with quasi-linear time complexity and state-of-the-art approximation capabilities.

Sources of Error In this paper we will study the error resulting from approximating an operator (mapping between Banach spaces) from within a class of finitely-parameterized operators. We show that the resulting error, expressed in terms of universal approximation of operators over a compact set or in terms of a resulting risk, can be driven to zero by increasing the number of parameters, and refining the approximations inherent in the neural operator architecture. In practice there will be two other sources of approximation error: firstly from the discretization of the data; and secondly from the use of empirical risk minimization over a finite data set to determine the parameters. Balancing all three sources of error is key to making algorithms efficient. However we do not study these other two sources of error in this work. Furthermore we do not study how the number of parameters in our approximation grows as the error tolerance is refined. Generally, this growth may be super-exponential as shown in (Kovachki, Lanthaler, and Mishra, 2021). However, for certain classes of operators and related approximation methods, it is possible to beat the curse of dimensionality; we refer the reader to the works (Lanthaler, Mishra, and George Em Karniadakis, 2021; Kovachki, Lanthaler, and Mishra, 2021) for detailed analyses demonstrating this. Finally we also emphasize that there is a potential source of error from the optimization procedure which attempts to minimize the empirical risk: it may not achieve the global minumum. Analysis of this error in the context of operator approximation has not been undertaken.

5.2 Learning Operators

In subsection 5.2, we outline the general problem of operator learning as well as our approach to solving it. In subsection 5.2, we discuss the functional data that is available and how we work with it numerically.

Problem Setting

Our goal is to learn a mapping between two infinite dimensional spaces by using a finite collection of observations of input-output pairs from this mapping. We make this problem concrete in the following setting. Let $\mathcal{A}$ and $\mathcal{U}$ be Banach spaces of functions defined on bounded domains $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ respectively and $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ be a (typically) non-linear map. Suppose we have observations

$\{a_j, u_j\}_{j=1}^N$ where $a_j \sim \mu$ are i.i.d. samples drawn from some probability measure μ supported on $\mathcal{A}$ and $u_j = \mathcal{G}^\dagger(a_j)$ is possibly corrupted with noise. We aim to build an approximation of $\mathcal{G}^\dagger$ by constructing a parametric map

$$\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}, \quad \theta \in \mathbb{R}^p \tag{5.2}$$

with parameters from the finite-dimensional space $\mathbb{R}^p$ and then choosing $\theta^\dagger \in \mathbb{R}^p$ so that $\mathcal{G}_{\theta^\dagger} \approx \mathcal{G}^\dagger$.

We will be interested in controlling the error of the approximation on average with respect to μ. In particular, assuming $\mathcal{G}^\dagger$ is μ-measurable, we will aim to control the $L^2_\mu(\mathcal{A};\mathcal{U})$ Bochner norm of the approximation

$$\|\mathcal{G}^\dagger - \mathcal{G}_\theta\|^2_{L^2_\mu(\mathcal{A};\mathcal{U})} = \mathbb{E}_{a\sim\mu}\|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|^2_\mathcal{U} = \int_\mathcal{A} \|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|^2_\mathcal{U} \, d\mu(a). \tag{5.3}$$

This is a natural framework for learning in infinite-dimensions as one could seek to solve the associated empirical-risk minimization problem

$$\min_{\theta\in\mathbb{R}^p} \mathbb{E}_{a\sim\mu}\|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|^2_\mathcal{U} \approx \min_{\theta\in\mathbb{R}^p} \frac{1}{N}\sum_{j=1}^N \|u_j - \mathcal{G}_\theta(a_j)\|^2_\mathcal{U} \tag{5.4}$$

which directly parallels the classical finite-dimensional setting (Vapnik, 1998).

Discretization

Since our data a_j and u_j are, in general, functions, to work with them numerically, we assume access only to their point-wise evaluations. To illustrate this, we will continue with the example of the preceding paragraph. For simplicity, assume $D = D'$ and suppose that the input and output functions are both real-valued. Let $D_j = \{x_j^{(1)}, \dots, x_j^{(n_j)}\} \subset D$ be a n_j-point discretization of the domain D and assume we have observations $a_j|_{D_j}, u_j|_{D_j} \in \mathbb{R}^{n_j}$, for a finite collection of input-output pairs indexed by j. In the next section, we propose a kernel inspired graph neural network architecture which, while trained on the discretized data, can produce the solution $u(x)$ for any $x \in D$ given an input $a \sim \mu$. That is to say that our approach is independent of the discretization D_j. We refer to this as being a function space architecture, a mesh-invariant architecture or a discretization-invariant architecture; this claim is verified numerically by showing invariance of the error as $n_j \to \infty$. Such a property is highly desirable as it allows a transfer of solutions between different grid geometries and discretization sizes with a single architecture which has a fixed number of parameters.

We note that, while the application of our methodology is based on having point-wise evaluations of the function, it is not limited by it. One may, for example, represent a function numerically as a finite set of truncated basis coefficients. Invariance of the representation would then be with respect to the size of this set. Our methodology can, in principle, be modified to accommodate this scenario through a suitably chosen architecture. We do not pursue this direction in the current work.

5.3 Proposed Architecture

Subsection 5.3 defines neural operators while subsections 5.3 and 5.3 compare them to DeepONets and Transformers respectively.

Neural Operators

In this section, we outline the neural operator framework. We assume that the input functions $a \in \mathcal{A}$ are $\mathbb{R}^{d_a}$-valued and defined on the bounded domain $D \subset \mathbb{R}^d$ while the output functions $u \in \mathcal{U}$ are $\mathbb{R}^{d_u}$-valued and defined on the bounded domain $D' \subset \mathbb{R}^{d'}$. The proposed architecture $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}$ has the following overall structure:

1. **Lifting**: Using a pointwise function $\mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, map the input $\{a : D \to \mathbb{R}^{d_a}\} \mapsto \{v_0 : D \to \mathbb{R}^{d_{v_0}}\}$ to its first hidden representation. Usually, we choose $d_{v_0} > d_a$ and hence this is a lifting operation performed by a fully local operator.

2. **Iterative Kernel Integration**: For $t = 0, \dots, T-1$, map each hidden representation to the next $\{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \mapsto \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ via the action of the sum of a local linear operator, a non-local integral kernel operator, and a bias function, composing the sum with a fixed, pointwise nonlinearity. Here we set $D_0 = D$ and $D_T = D'$ and impose that $D_t \subset \mathbb{R}^{d_t}$ is a bounded domain.

3. **Projection**: Using a pointwise function $\mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$, map the last hidden representation $\{v_T : D' \to \mathbb{R}^{d_{v_T}}\} \mapsto \{u : D' \to \mathbb{R}^{d_u}\}$ to the output function. Analogously to the first step, we usually pick $d_{v_T} > d_u$ and hence this is a projection step performed by a fully local operator.

The outlined structure mimics that of a finite dimensional neural network where hidden representations are successively mapped to produce the final output. In particular, we have

$$\mathcal{G}_\theta \coloneqq \mathcal{Q} \circ \sigma_T(W_{T-1} + \mathcal{K}_{T-1} + b_{T-1}) \circ \cdots \circ \sigma_1(W_0 + \mathcal{K}_0 + b_0) \circ \mathcal{P} \qquad (5.5)$$

where $\mathcal{P} : \mathbb{R}^{d_a} \to \mathbb{R}^{d_{v_0}}$, $\mathcal{Q} : \mathbb{R}^{d_{v_T}} \to \mathbb{R}^{d_u}$ are the local lifting and projection mappings respectively, $W_t \in \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}}$ are local linear operators (matrices), $\mathcal{K}_t : \{v_t : D_t \to \mathbb{R}^{d_{v_t}}\} \to \{v_{t+1} : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}\}$ are integral kernel operators, $b_t : D_{t+1} \to \mathbb{R}^{d_{v_{t+1}}}$ are bias functions, and σ_t are fixed activation functions acting locally as maps $\mathbb{R}^{v_{t+1}} \to \mathbb{R}^{v_{t+1}}$ in each layer. The output dimensions $d_{v_0}, \dots, d_{v_T}$ as well as the input dimensions $d_1, \dots, d_{T-1}$ and domains of definition $D_1, \dots, D_{T-1}$ are hyperparameters of the architecture. By local maps, we mean that the action is pointwise, in particular, for the lifting and projection maps, we have $(\mathcal{P}(a))(x) = \mathcal{P}(a(x))$ for any $x \in D$ and $(\mathcal{Q}(v_T))(x) = \mathcal{Q}(v_T(x))$ for any $x \in D'$ and similarly, for the activation, $(\sigma(v_{t+1}))(x) = \sigma(v_{t+1}(x))$ for any $x \in D_{t+1}$. The maps, $\mathcal{P}$, $\mathcal{Q}$, and σ_t can thus be thought of as defining Nemitskiy operators (R. Dudley and Norvaisa, 2011, Chapters 6, 7) when each of their components are assumed to be Borel measurable. This interpretation allows us to define the general neural operator architecture when pointwise evaluation is not well-defined in the spaces $\mathcal{A}$ or $\mathcal{U}$, e.g. when they are Lebesgue, Sobolev, or Besov spaces.

The crucial difference between the proposed architecture (5.5) and a standard feed-forward neural network is that all operations are directly defined in function space (noting that $\mathcal{P}$ and $\mathcal{Q}$ are interpreted through their extension to Nemitskiy operators) and therefore do not depend on any discretization of the data. Intuitively, the lifting step locally maps the data to a space where the non-local part of $\mathcal{G}^\dagger$ is easier to capture. This is then learned by successively approximating using integral kernel operators composed with a local nonlinearity. Each integral kernel operator is the function space analog of the weight matrix in a standard feed-forward network since they are infinite-dimensional linear operators mapping one function space to another. We turn the biases, which are normally vectors, to functions and, using intuition from the ResNet architecture (K. He et al., 2016), we further add a local linear operator acting on the output of the previous layer before applying the nonlinearity. The final projection step simply gets us back to the space of our output function. We concatenate in $\theta \in \mathbb{R}^p$ the parameters of $\mathcal{P}$, $\mathcal{Q}$, $\{b_t\}$ which are usually themselves shallow neural networks, the parameters of the kernels representing $\{\mathcal{K}_t\}$ which are again usually shallow neural networks, and the matrices $\{W_t\}$. We note, however, that our framework is general and other parameterizations such as polynomials may also be employed.

Integral Kernel Operators We define three version of the integral kernel operator $\mathcal{K}_t$ used in (5.5). For the first, let $\kappa^{(t)} \in C(D_{t+1} \times D_t; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$ and let ν_t be a Borel measure on D_t. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1}. \tag{5.6}$$

Normally, we take ν_t to simply be the Lebesgue measure on $\mathbb{R}^{d_t}$ but, as discussed in Section 5.5, other choices can be used to speed up computation or aid the learning process by building in *a priori* information. The choice of integral kernel operator in (5.6) defines the basic form of the neural operator and is the one we analyze in Section 5.4 and study most in the numerical experiments of Section 4.4.

For the second, let $\kappa^{(t)} \in C(D_{t+1} \times D_t \times \mathbb{R}^{d_a} \times \mathbb{R}^{d_a}; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y, a(\Pi_{t+1}^D(x)), a(\Pi_t^D(y))) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1} \tag{5.7}$$

where $\Pi_t^D : D_t \to D$ are fixed mappings. We have found numerically that, for certain PDE problems, the form (5.7) outperforms (5.6) due to the strong dependence of the solution u on the parameters a. Indeed, if we think of (5.5) as a discrete time dynamical system, then the input $a \in \mathcal{A}$ only enters through the initial condition hence its influence diminishes with more layers. By directly building in a-dependence into the kernel, we ensure that it influences the entire architecture.

Lastly, let $\kappa^{(t)} \in C(D_{t+1} \times D_t \times \mathbb{R}^{d_{v_t}} \times \mathbb{R}^{d_{v_t}}; \mathbb{R}^{d_{v_{t+1}} \times d_{v_t}})$. Then we define $\mathcal{K}_t$ by

$$(\mathcal{K}_t(v_t))(x) = \int_{D_t} \kappa^{(t)}(x, y, v_t(\Pi_t(x)), v_t(y)) v_t(y) \, \mathrm{d}\nu_t(y) \qquad \forall x \in D_{t+1} \tag{5.8}$$

where $\Pi_t : D_{t+1} \to D_t$ are fixed mappings. Note that, in contrast to (5.6) and (5.7), the integral operator (5.8) is nonlinear since the kernel can depend on the input function v_t. With this definition and a particular choice of kernel κ_t and measure ν_t, we show in Section 5.3 that neural operators are a continuous input/output space generalization of the popular transformer architecture (Vaswani et al., 2017).

Single Hidden Layer Construction Having defined possible choices for the integral kernel operator, we are now in a position to explicitly write down a full layer of the architecture defined by (5.5). For simplicity, we choose the integral kernel operator given by (5.6), but note that the other definitions (5.7), (5.8) work analogously.

We then have that a single hidden layer update is given by

$$v_{t+1}(x) = \sigma_{t+1}\left(W_t v_t(\Pi_t(x)) + \int_{D_t} \kappa^{(t)}(x,y) v_t(y)\, \mathrm{d}\nu_t(y) + b_t(x)\right) \quad \forall x \in D_{t+1} \tag{5.9}$$

where $\Pi_t : D_{t+1} \to D_t$ are fixed mappings. We remark that, since we often consider functions on the same domain, we usually take Π_t to be the identity.

We will now give an example of a full single hidden layer architecture, i.e. when $T = 2$. We choose $D_1 = D$, take σ_2 as the identity, and denote σ_1 by σ, assuming it is any activation function. Furthermore, for simplicity, we set $W_1 = 0$, $b_1 = 0$, and assume that $\nu_0 = \nu_1$ is the Lebesgue measure on $\mathbb{R}^d$. Then (5.5) becomes

$$(\mathcal{G}_\theta(a))(x) = \\ \mathcal{Q}\left(\int_D \kappa^{(1)}(x,y)\sigma\left(W_0\mathcal{P}(a(y)) + \int_D \kappa^{(0)}(y,z)\mathcal{P}(a(z))\,\mathrm{d}z + b_0(y)\right)\mathrm{d}y\right) \tag{5.10}$$

for any $x \in D'$. In this example, $\mathcal{P} \in C(\mathbb{R}^{d_a}; \mathbb{R}^{d_{v_0}})$, $\kappa^{(0)} \in C(D \times D; \mathbb{R}^{d_{v_1} \times d_{v_0}})$, $b_0 \in C(D; \mathbb{R}^{d_{v_1}})$, $W_0 \in \mathbb{R}^{d_{v_1} \times d_{v_0}}$, $\kappa^{(0)} \in C(D' \times D; \mathbb{R}^{d_{v_2} \times d_{v_1}})$, and $\mathcal{Q} \in C(\mathbb{R}^{d_{v_2}}; \mathbb{R}^{d_u})$. One can then parametrize the continuous functions $\mathcal{P}$, $\mathcal{Q}$, $\kappa^{(0)}$, $\kappa^{(1)}$, b_0 by standard feed-forward neural networks (or by any other means) and the matrix W_0 simply by its entries. The parameter vector $\theta \in \mathbb{R}^p$ then becomes the concatenation of the parameters of $\mathcal{P}$, $\mathcal{Q}$, $\kappa^{(0)}$, $\kappa^{(1)}$, b_0 along with the entries of W_0. One can then optimize these parameters by minimizing with respect to θ using standard gradient based minimization techniques. To implement this minimization, the functions entering the loss need to be discretized; but the learned parameters may then be used with other discretizations. In Section 5.5, we discuss various choices for parametrizing the kernels, picking the integration measure, and how those choices affect the computational complexity of the architecture.

Preprocessing It is often beneficial to manually include features into the input functions a to help facilitate the learning process. For example, instead of considering the $\mathbb{R}^{d_a}$-valued vector field a as input, we use the $\mathbb{R}^{d+d_a}$-valued vector field $(x, a(x))$. By including the identity element, information about the geometry of the spatial domain D is directly incorporated into the architecture. This allows the neural networks direct access to information that is already known in the problem and therefore eases learning. We use this idea in all of our numerical experiments in Section 4.4. Similarly, when learning a smoothing operator, it may be beneficial to include a smoothed version of the inputs a_ϵ using, for example, Gaussian convolution. Derivative information may also be of interest and thus, as input, one may consider, for example, the $\mathbb{R}^{d+2d_a+dd_a}$-valued vector field $(x, a(x), a_\epsilon(x), \nabla_x a_\epsilon(x))$. Many other possibilities may be considered on a problem-specific basis.

DeepONets are Neural Operators

We will now draw a parallel between the recently proposed DeepONet architecture in L. Lu, Jin, and George Em Karniadakis, 2019 and our neural operator framework. In fact, we will show that a particular variant of functions from the DeepONets class is a special case of a single hidden layer neural operator construction once discretized appropriately. To that end, we work with (5.10) where we choose $W_0 = 0$ and denote b_0 by b. For simplicity, we will consider only real-valued functions, i.e. $d_a = d_u = 1$ and set $d_{v_0} = d_{v_1} = n$ and $d_{v_2} = p$ for some $n, p \in \mathbb{N}$. Define $\mathcal{P} : \mathbb{R} \to \mathbb{R}^n$ by $\mathcal{P}(x) = (x, \dots, x)$ and $\mathcal{Q} : \mathbb{R}^p \to \mathbb{R}$ by $\mathcal{Q}(x) = x_1 + \cdots + x_p$. Furthermore let $\kappa^{(1)} : D' \times D \to \mathbb{R}^{p \times n}$ be defined by some $\kappa^{(1)}_{jk} : D' \times D \to \mathbb{R}$ for $j = 1, \dots, p$ and $k = 1, \dots, n$. Similarly let $\kappa^{(0)} : D \times D \to \mathbb{R}^{n \times n}$ be given as $\kappa^{(0)}(x, y) = \text{diag}(\kappa^{(0)}_1(x, y), \dots, \kappa^{(0)}_n(x, y))$ for some $\kappa^{(0)}_1, \dots \kappa^{(0)}_n : D \times D \to \mathbb{R}$. Then (5.10) becomes

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^{p} \sum_{j=1}^{n} \int_D \kappa^{(1)}_{jk}(x, y) \sigma \left(\int_D \kappa^{(0)}_j(y, z) a(z) \, \mathrm{d}z + b_j(y) \right) \mathrm{d}y$$

where $b(y) = (b_1(y), \dots, b_n(y))$ for some $b_1, \dots, b_n : D \to \mathbb{R}$. Let $x_1, \dots, x_q \in D$ be the points at which the input function a is evaluated and denote by $\tilde{a} = \big(a(x_1), \dots, a(x_q)\big) \in \mathbb{R}^q$ the vector of evaluations. Choose $\kappa^{(0)}_{jj}(y, z) = \mathbb{1}(y) w_j(z)$ for some $w_1, \dots, w_n : D \to \mathbb{R}$ where $\mathbb{1}$ denotes the constant function taking the value one. Let

$$w_j(x_l) = \frac{q}{|D|} \tilde{w}_{jl}$$

for $j = 1, \dots, n$ and $l = 1, \dots, q$ where $\tilde{w}_{jl} \in \mathbb{R}$ are some constants. Furthermore let $b_j(y) = \tilde{b}_j \mathbb{1}(y)$ for some constants $\tilde{b}_j \in \mathbb{R}$. Then the Monte Carlo approximation of the inner-integral yields

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^{p} \sum_{j=1}^{n} \int_D \kappa^{(1)}_{jk}(x, y) \sigma \left(\langle \tilde{w}_j, \tilde{a} \rangle_{\mathbb{R}^q} + \tilde{b}_j \right) \mathbb{1}(y) \, \mathrm{d}y$$

where $\tilde{w}_j = \big(\tilde{w}_{j1}, \dots, \tilde{w}_{jq}\big)$. Choose $\kappa^{(1)}_{jk}(x, y) = (\tilde{c}_{jk}/|D|) \varphi_k(x) \mathbb{1}(y)$ for some constants $\tilde{c}_{jk} \in \mathbb{R}$ and functions $\varphi_1, \dots, \varphi_p : D' \to \mathbb{R}$. Then we obtain

$$(\mathcal{G}_\theta(a))(x) = \sum_{k=1}^{p} \left(\sum_{j=1}^{n} \tilde{c}_{jk} \sigma \left(\langle \tilde{w}_j, \tilde{a} \rangle_{\mathbb{R}^q} + \tilde{b}_j \right) \right) \varphi_k(x) = \sum_{k=1}^{p} G_k(\tilde{v}) \varphi_k(x) \quad (5.11)$$

where $G_k : \mathbb{R}^q \to \mathbb{R}$ can be viewed as the components of a single hidden layer neural network $G : \mathbb{R}^q \to \mathbb{R}^p$ with parameters $\tilde{w}_{jl}, \tilde{b}_j, \tilde{c}_{jk}$. The set of maps $\varphi_1, \dots, \varphi_p$ form the *trunk net* while G is the *branch net* of a DeepONet. Our construction above can clearly be generalized to yield arbitrary depth branch nets by adding more kernel integral layers, and, similarly, the trunk net can be chosen arbitrarily deep by parameterizing each φ_k as a deep neural network.

Note however that parameterizing as suggested by (5.11) yields an approximation that is inconsistent in function space since the number of parameters used to define G is *not independent* of the discretization used for a. Therefore, the number of parameters in a DeepONet grows as we refine the discretization of a, blowing up in the continuum. This issue could be resolved for DeepONet(s) by fixing the set of points on which the input function is evaluated independently of its discretization, by taking local spatial averages as in (Lanthaler, Mishra, and George Em Karniadakis, 2021) or more generally by taking a set of linear functionals on $\mathcal{A}$ as input to a finite-dimensional branch neural network. We demonstrate numerically in Section 4.4 that, when applied in the standard way, the error incurred by DeepONet(s) grows with the discretization of a while it remains constant for neural operators.

Linear Approximation and Nonlinear Approximation. We point out that parametrizations of the form (5.11) fall within the class of *linear* approximation methods since the nonlinear space $\mathcal{G}^\dagger(\mathcal{A})$ is approximated by the linear space span$\{\varphi_1, \dots, \varphi_p\}$ (R. A. DeVore, 1998). The quality of the best possible linear approximation to a nonlinear space is given by the Kolmogorov n-width where n is the dimension of the linear space used in the approximation (A. Pinkus, 1985). The rate of decay of the n-width as a function of n quantifies how well the linear space approximates the nonlinear one. It is well know that for some problems such as the flow maps of advection-dominated PDEs, the n-widths decay very slowly; hence a very large n is needed for a good approximation for such problems (Cohen and R. DeVore, 2015). This can be limiting in practice as more parameters are needed in order to describe more basis functions φ_j and therefore more data is needed to fit these parameters.

On the other hand, we point out that parametrizations of the form (5.5), and the particular case (5.10), constitute (in general) a form of *nonlinear* approximation. The benefits of nonlinear approximation are well understood in the setting of function approximation, see e.g. (R. A. DeVore, 1998); however the theory for the operator setting is still in its infancy (Bonito et al., 2020; Cohen, Devore, et al., 2020). We observe numerically in Section 4.4 that nonlinear parametrizations such as (5.10) outperform linear ones such as DeepONets or the low-rank method introduced in Section 5.5 when implemented with similar numbers of parameters.

Function Representation. An important difference between neural operators, introduced here, PCA-based operator approximation, introduced in Bhattacharya et al., 2020 and DeepONets, introduced in L. Lu, Jin, and George Em Karniadakis, 2019, is the manner in which the output function space is finite-dimensionalized. Neural operators as implemented in this paper typically use the same finite-dimensionalization in both the input and output function spaces; however different variants of the neural operator idea use different finite-dimensionalizations. As discussed in Section 5.5, the GNO and MGNO are finite-dimensionalized using pointwise values as the nodes of graphs; the FNO is finite-dimensionalized in Fourier space, requiring finite-dimensionalization on a uniform grid in real space; the Low-rank neural operator is finite-dimensionalized on a product space formed from the Barron space of neural networks. The PCA approach finite-dimensionalizes in the span of PCA modes. DeepONet, on the other hand, uses different input and output space finite-dimensionalizations; in its basic form it uses pointwise (grid) values on the input (branch net) whilst its output (trunk net) is represented as a function in Barron space. There also exist POD-DeepONet variants that finite-dimensionalize the output in the span of PCA modes L. Lu, Meng, et al., 2021, bringing them closer to the method introduced in Bhattacharya et al., 2020, but with a different finite-dimensionalization of the input space.

As is widely quoted, “all models are wrong, but some are useful” Box, 1976. For operator approximation, each finite-dimensionalization has its own induced biases and limitations, and therefore works best on a subset of problems. Finite-dimensionalization introduces a trade-off between flexibility and representation power of the resulting approximate architecture. The Barron space representation (Low-rank operator and DeepONet) is usually the most generic and flexible as it is widely applicable. However this can lead to induced biases and reduced representation power on specific problems; in practice, DeepONet sometimes needs problem-specific feature engineering and architecture choices as studied in L. Lu, Meng, et al., 2021. We conjecture that these problem-specific features compensate for the induced bias and reduced representation power that the basic form of the method (L. Lu, Jin, and George Em Karniadakis, 2019) sometimes exhibits. The

PCA (PCA operator, POD-DeepONet) and graph-based (GNO, MGNO) discretizations are also generic, but more specific compared to the DeepONet representation; for this reason POD-DeepONet can outperform DeepONet on some problems (L. Lu, Meng, et al., 2021). On the other hand, the uniform grid-based representation FNO is the most specific of all those operator approximators considered in this paper: in its basic form it applies by discretizing the input functions, assumed to be specified on a periodic domain, on a uniform grid. As shown in Section 4.4 FNO usually works out of the box on such problems. But, as a trade-off, it requires substantial additional treatments to work well on non-uniform geometries, such as extension, interpolation (explored in L. Lu, Meng, et al., 2021), and Fourier continuation (Bruno, Han, and Pohlman, 2007).

Transformers are Neural Operators

We will now show that our neural operator framework can be viewed as a continuum generalization to the popular transformer architecture (Vaswani et al., 2017) which has been extremely successful in natural language processing tasks (Devlin et al., 2018; Brown et al., 2020) and, more recently, is becoming a popular choice in computer vision tasks (Dosovitskiy et al., 2020). The parallel stems from the fact that we can view sequences of arbitrary length as arbitrary discretizations of functions. Indeed, in the context of natural language processing, we may think of a sentence as a "word"-valued function on, for example, the domain $[0, 1]$. Assuming our function is linked to a sentence with a fixed semantic meaning, adding or removing words from the sentence simply corresponds to refining or coarsening the discretization of $[0, 1]$. We will now make this intuition precise.

We will show that by making a particular choice of the nonlinear integral kernel operator (5.8) and discretizing the integral by a Monte-Carlo approximation, a neural operator layer reduces to a pre-normalized, single-headed attention, transformer block as originally proposed in (Vaswani et al., 2017). For simplicity, we assume $d_{v_t} = n \in \mathbb{N}$ and that $D_t = D$ for any $t = 0, \dots, T$, the bias term is zero, and $W = I$ is the identity. Further, to simplify notation, we will drop the layer index t from (5.9) and, employing (5.8), obtain

$$u(x) = \sigma\left(v(x) + \int_D \kappa_v(x, y, v(x), v(y)) v(y)\, \mathrm{d}y\right) \qquad \forall x \in D \tag{5.12}$$

a single layer of the neural operator where $v : D \to \mathbb{R}^n$ is the input function to the layer and we denote by $u : D \to \mathbb{R}^n$ the output function. We use the notation κ_v to indicate that the kernel depends on the entirety of the function v as well as on its pointwise values $v(x)$ and $v(y)$. While this is not explicitly done in (5.8), it is a straightforward generalization. We now pick a specific form for kernel, in particular, we assume $\kappa_v : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{n\times n}$ does not explicitly depend on the spatial variables (x, y) but only on the input pair $(v(x), v(y))$. Furthermore, we let

$$\kappa_v(v(x), v(y)) = g_v(v(x), v(y))R$$

where $R \in \mathbb{R}^{n\times n}$ is a matrix of free parameters, i.e. its entries are concatenated in θ so they are learned, and $g_v : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is defined as

$$g_v(v(x), v(y)) = \left(\int_D \exp\left(\frac{\langle Av(s), Bv(y)\rangle}{\sqrt{m}}\right) \mathrm{d}s \right)^{-1} \exp\left(\frac{\langle Av(x), Bv(y)\rangle}{\sqrt{m}}\right).$$

Here $A, B \in \mathbb{R}^{m\times n}$ are again matrices of free parameters, $m \in \mathbb{N}$ is a hyperparameter, and $\langle\cdot,\cdot\rangle$ is the Euclidean inner-product on $\mathbb{R}^m$. Putting this together, we find that (5.12) becomes

$$u(x) = \sigma\left(v(x) + \int_D \frac{\exp\left(\frac{\langle Av(x), Bv(y)\rangle}{\sqrt{m}}\right)}{\int_D \exp\left(\frac{\langle Av(s), Bv(y)\rangle}{\sqrt{m}}\right) \mathrm{d}s} Rv(y)\, \mathrm{d}y \right) \qquad \forall x \in D. \quad (5.13)$$

Equation (5.13) can be thought of as the continuum limit of a transformer block. To see this, we will discretize to obtain the usual transformer block.

To that end, let $\{x_1, \dots, x_k\} \subset D$ be a uniformly-sampled, k-point discretization of D and denote $v_j = v(x_j) \in \mathbb{R}^n$ and $u_j = u(x_j) \in \mathbb{R}^n$ for $j = 1, \dots, k$. Approximating the inner-integral in (5.13) by Monte-Carlo, we have

$$\int_D \exp\left(\frac{\langle Av(s), Bv(y)\rangle}{\sqrt{m}}\right) \mathrm{d}s \approx \frac{|D|}{k} \sum_{l=1}^{k} \exp\left(\frac{\langle Av_l, Bv(y)\rangle}{\sqrt{m}}\right).$$

Plugging this into (5.13) and using the same approximation for the outer integral yields

$$u_j = \sigma\left(v_j + \sum_{q=1}^{k} \frac{\exp\left(\frac{\langle Av_j, Bv_q\rangle}{\sqrt{m}}\right)}{\sum_{l=1}^{k} \exp\left(\frac{\langle Av_l, Bv_q\rangle}{\sqrt{m}}\right)} Rv_q \right), \qquad j = 1, \dots, k. \qquad (5.14)$$

Equation (5.14) can be viewed as a Nyström approximation of (5.13). Define the vectors $z_q \in \mathbb{R}^k$ by

$$z_q = \frac{1}{\sqrt{m}}(\langle Av_1, Bv_q\rangle, \dots, \langle Av_k, Bv_q\rangle), \qquad q = 1, \dots, k.$$

Define $S : \mathbb{R}^k \to \Delta_k$, where Δ_k denotes the k-dimensional probability simplex, as the softmax function

$$S(w) = \left(\frac{\exp(w_1)}{\sum_{j=1}^k \exp(w_j)}, \dots, \frac{\exp(w_k)}{\sum_{j=1}^k \exp(w_j)} \right), \qquad \forall w \in \mathbb{R}^k.$$

Then we may re-write (5.14) as

$$u_j = \sigma \left(v_j + \sum_{q=1}^k S_j(z_q) R v_q \right), \qquad j = 1, \dots, k.$$

Furthermore, if we re-parametrize $R = R^{\text{out}} R^{\text{val}}$ where $R^{\text{out}} \in \mathbb{R}^{n \times m}$ and $R^{\text{val}} \in \mathbb{R}^{m \times n}$ are matrices of free parameters, we obtain

$$u_j = \sigma \left(v_j + R^{\text{out}} \sum_{q=1}^k S_j(z_q) R^{\text{val}} v_q \right), \qquad j = 1, \dots, k$$

which is precisely the single-headed attention, transformer block with no layer normalization applied inside the activation function. In the language of transformers, the matrices A, B, and R^{val} correspond to the *queries*, *keys*, and *values* functions respectively. We note that tricks such as layer normalization (Ba, Kiros, and Hinton, 2016) can be adapted in a straightforward manner to the continuum setting and incorporated into (5.13). Furthermore multi-headed self-attention can be realized by simply allowing κ_v to be a sum of over multiple functions with form $g_v R$ all of which have separate trainable parameters. Including such generalizations yields the continuum limit of the transformer as implemented in practice. We do not pursue this here as our goal is simply to draw a parallel between the two methods.

While we have not rigorously experimented with using transformer architectures for the problems outlined in Section 5.6, we have found, in initial tests, that they perform worse, are slower, and are more memory expensive than neural operators using (5.6)-(5.8). Their high computational complexity is evident from (5.13) as we must evaluate a *nested* integral of v for each $x \in D$. Recently more efficient transformer architectures have been proposed, e.g. (Choromanski et al., 2020) and some have been applied to computer vision tasks. We leave as interesting future work experimenting and comparing these architectures to the neural operator both on problems in scientific computing and more traditional machine learning tasks.

5.4 Approximation Theory

The paper by T. Chen and H. Chen, 1995 provides the first universal approximation theorem for operator approximation via neural networks, and the paper by Bhattacharya et al., 2020 provides an alternative architecture and approximation result.

The analysis of T. Chen and H. Chen, 1995 was recently extended in significant ways in the paper by Lanthaler, Mishra, and George Em Karniadakis, 2021 where, for the first time, the curse of dimensionality is addressed, and resolved, for certain specific operator learning problems, using the DeepOnet generalization L. Lu, Jin, and George Em Karniadakis, 2019; L. Lu, Jin, Pang, et al., 2021 of T. Chen and H. Chen, 1995. The paper Lanthaler, Mishra, and George Em Karniadakis, 2021 was generalized to study operator approximation, and the curse of dimensionality, for the FNO, in Kovachki, Lanthaler, and Mishra, 2021.

Unlike the finite-dimensional setting, the choice of input and output spaces $\mathcal{A}$ and $\mathcal{U}$ for the mapping $\mathcal{G}^\dagger$ play a crucial role in the approximation theory due to the distinctiveness of the induced norm topologies. In this section, we prove universal approximation theorems for neural operators both with respect to the topology of uniform convergence over compact sets and with respect to the topology induced by the Bochner norm (5.3). We focus our attention on the Lebesgue, Sobolev, continuous, and continuously differentiable function classes as they have numerous applications in scientific computing and machine learning problems. Unlike the results of Bhattacharya et al., 2020; Kovachki, Lanthaler, and Mishra, 2021 which rely on the Hilbertian structure of the input and output spaces or the results of T. Chen and H. Chen, 1995; Lanthaler, Mishra, and George Em Karniadakis, 2021 which rely on the continuous functions, our results extend to more general Banach spaces as specified by Assumptions 34 and 35 and are, to the best of our knowledge, the first of their kind to apply at this level of generality.

Our method of proof proceeds by making use of the following two observations. First we establish the Banach space approximation property Grothendieck, 1955 for the input and output spaces of interest, which allows for a finite dimensionalization of the problem. In particular, we prove that the Banach space approximation property holds for various function spaces defined on Lipschitz domains; the precise result we need, while unsurprising, seems to be missing from the functional analysis literature and so we provide statement and proof. Details are given in Appendix A. Second, we establish that integral kernel operators with smooth kernels can be used to approximate linear functionals of various input spaces. In doing so, we establish a Reisz-type representation theorem for the continuously differentiable functions. Such a result is not surprising and mimics the well-known result for Sobolev spaces; however in the form we need it we could not find the result in the functional analysis literature and so we provide statement and proof. Details are given in Appendix

B. With these two facts, we construct a neural operator which linearly maps any input function to a finite vector then non-linearly maps this vector to a new finite vector which is then used to form the coefficients of a basis expansion for the output function. We reemphasize that our approximation theory uses the fact that neural operators can be reduced to a linear method of approximation (as pointed out in Section 5.3) and does not capture any benefits of nonlinear approximation. However these benefits *are* present in the architecture and are exploited by the trained networks we find in practice. Exploiting their nonlinear nature to potentially obtain improved rates of approximation remains an interesting direction for future research.

The rest of this section is organized as follows. In Subsection 5.4, we define allowable activation functions and the set of neural operators used in our theory, noting that they constitute a subclass of the neural operators defined in Section 5.3. In Subsection 5.4, we state and prove our main universal approximation theorems.

Neural Operators

For any $n \in \mathbb{N}$ and $\sigma : \mathbb{R} \to \mathbb{R}$, we define the set of real-valued n-layer neural networks on $\mathbb{R}^d$ by

$$\begin{aligned}\mathsf{N}_n(\sigma;\mathbb{R}^d) \coloneqq \{f : \mathbb{R}^d \to \mathbb{R} : f(x) = W_n\sigma(\dots W_1\sigma(W_0x + b_0) + b_1 \dots) + b_n,\\ W_0 \in \mathbb{R}^{d_0\times d}, W_1 \in \mathbb{R}^{d_0\times d_1}, \dots, W_n \in \mathbb{R}^{1\times d_n},\\ b_0 \in \mathbb{R}^{d_0}, b_1 \in \mathbb{R}^{d_1}, \dots, b_n \in \mathbb{R},\ d_0, d_1, \dots, d_n \in \mathbb{N}\}.\end{aligned}$$

We define the set of $\mathbb{R}^{d'}$-valued neural networks simply by stacking real-valued networks

$$\begin{aligned}\mathsf{N}_n(\sigma;\mathbb{R}^d,\mathbb{R}^{d'}) \coloneqq \{f : \mathbb{R}^d \to \mathbb{R}^{d'} :\\ f(x) = \big(f_1(x), \dots, f_{d'}(x)\big),\ f_1, \dots, f_{d'} \in \mathsf{N}_n(\sigma;\mathbb{R}^d)\}.\end{aligned}$$

We remark that we could have defined $\mathsf{N}_n(\sigma;\mathbb{R}^d,\mathbb{R}^{d'})$ by letting $W_n \in \mathbb{R}^{d'\times d_n}$ and $b_n \in \mathbb{R}^{d'}$ in the definition of $\mathsf{N}_n(\sigma;\mathbb{R}^d)$ because we allow arbitrary width, making the two definitions equivalent; however the definition as presented is more convenient for our analysis. We also employ the preceding definition with $\mathbb{R}^d$ and $\mathbb{R}^{d'}$ replaced by spaces of matrices. For any $m \in \mathbb{N}_0$, we define the set of allowable activation functions as the continuous $\mathbb{R} \to \mathbb{R}$ maps which make neural networks dense in $C^m(\mathbb{R}^d)$ on compacta at any fixed depth,

$$\mathsf{A}_m \coloneqq \{\sigma \in C(\mathbb{R}) : \exists n \in \mathbb{N} \text{ s.t. } \mathsf{N}_n(\sigma;\mathbb{R}^d) \text{ is dense in } C^m(K) \forall K \subset \mathbb{R}^d \text{ compact}\}.$$

It is shown in Allan Pinkus, 1999, Theorem 4.1 that $\{\sigma \in C^m(\mathbb{R}) : \sigma \text{ is not a polynomial}\}$ is contained in A_m with $n = 1$. Clearly $\mathsf{A}_{m+1} \subseteq \mathsf{A}_m$.

We define the set of linearly bounded activations as

$$\mathsf{A}_m^{\mathsf{L}} := \left\{ \sigma \in \mathsf{A}_m : \sigma \text{ is Borel measurable} \;,\; \sup_{x \in \mathbb{R}} \frac{|\sigma(x)|}{1 + |x|} < \infty \right\},$$

noting that any globally Lipschitz, non-polynomial, C^m-function is contained in $\mathsf{A}_m^{\mathsf{L}}$. Most activation functions used in practice fall within this class, for example, ReLU $\in \mathsf{A}_0^{\mathsf{L}}$, ELU $\in \mathsf{A}_1^{\mathsf{L}}$ while $\tanh$, sigmoid $\in \mathsf{A}_m^{\mathsf{L}}$ for any $m \in \mathbb{N}_0$.

For approximation in a Bochner norm, we will be interested in constructing globally bounded neural networks which can approximate the identity over compact sets as done in (Lanthaler, Mishra, and George Em Karniadakis, 2021; Bhattacharya et al., 2020). This allows us to control the potential unboundedness of the support of the input measure by exploiting the fact that the probability of an input must decay to zero in unbounded regions. Following (Lanthaler, Mishra, and George Em Karniadakis, 2021), we introduce the forthcoming definition which uses the notation of the diameter of a set. In particular, the diameter of any set $S \subseteq \mathbb{R}^d$ is defined as, for $|\cdot|_2$ the Euclidean norm on $\mathbb{R}^d$,

$$\text{diam}_2(S) := \sup_{x,y \in S} |x - y|_2.$$

Definition 33. *We denote by* BA *the set of maps* $\sigma \in \mathsf{A}_0$ *such that, for any compact set* $K \subset \mathbb{R}^d$*,* $\epsilon > 0$*, and* $C \geq \text{diam}_2(K)$ *, there exists a number* $n \in \mathbb{N}$ *and a neural network* $f \in \mathsf{N}_n(\sigma; \mathbb{R}^d, \mathbb{R}^{d'})$ *such that*

$$\begin{aligned} |f(x) - x|_2 &\leq \epsilon, \qquad \forall x \in K, \\ |f(x)|_2 &\leq C, \qquad \forall x \in \mathbb{R}^d. \end{aligned}$$

It is shown in Lanthaler, Mishra, and George Em Karniadakis, 2021, Lemma C.1 that ReLU $\in \mathsf{A}_0^{\mathsf{L}} \cap \mathsf{BA}$ with $n = 3$.

We will now define the specific class of neural operators for which we prove a universal approximation theorem. It is important to note that the class with which we work is a simplification of the one given in (5.5). In particular, the lifting and projection operators $\mathcal{Q}, \mathcal{P}$, together with the final activation function σ_n, are set to the identity, and the local linear operators $W_0, \dots, W_{n-1}$ are set to zero. In our numerical studies we have in any case typically set σ_n to the identity. However we have found that learning the local operators $\mathcal{Q}, \mathcal{P}$ and $W_0, \ldots, W_{n-1}$ is beneficial in practice; extending the universal approximation theorems given here to explain this benefit would be an important but non-trivial development of the analysis we present here.

Let $D \subset \mathbb{R}^d$ be a domain. For any $\sigma \in \mathsf{A}_0$, we define the set of affine kernel integral operators by

$$\mathsf{IO}(\sigma; D, \mathbb{R}^{d_1}, \mathbb{R}^{d_2}) = \{f \mapsto \int_D \kappa(\cdot, y) f(y) \, \mathsf{d}y + b : \; \kappa \in \mathsf{N}_{n_1}(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{d_2 \times d_1}), \\ b \in \mathsf{N}_{n_2}(\sigma; \mathbb{R}^d, \mathbb{R}^{d_2}), \; n_1, n_2 \in \mathbb{N}\},$$

for any $d_1, d_2 \in \mathbb{N}$. Clearly, since $\sigma \in \mathsf{A}_0$, any $S \in \mathsf{IO}(\sigma; D, \mathbb{R}^{d_1}, \mathbb{R}^{d_2})$ acts as $S : L^p(D; \mathbb{R}^{d_1}) \to L^p(D; \mathbb{R}^{d_2})$ for any $1 \leq p \leq \infty$ since $\kappa \in C(\bar{D} \times \bar{D}; \mathbb{R}^{d_2 \times d_1})$ and $b \in C(\bar{D}; \mathbb{R}^{d_2})$. For any $n \in \mathbb{N}_{\geq 2}$, $d_a, d_u \in \mathbb{N}$, $D \subset \mathbb{R}^d$, $D' \subset \mathbb{R}^{d'}$ domains, and $\sigma_1 \in \mathsf{A}_0^{\mathsf{L}}$, $\sigma_2, \sigma_3 \in \mathsf{A}_0$, we define the set of n-layer neural operators by

$$\begin{aligned} \mathsf{NO}_n(\sigma_1, &\sigma_2, \sigma_3; D, D', \mathbb{R}^{d_a}, \mathbb{R}^{d_u}) = \\ &\{f \mapsto \int_D \kappa_n(\cdot, y) \big(S_{n-1}\sigma_1(\ldots S_2\sigma_1(S_1(S_0 f)) \ldots)\big)(y) \, \mathsf{d}y : \\ &S_0 \in \mathsf{IO}(\sigma_2, D; \mathbb{R}^{d_a}, \mathbb{R}^{d_1}), \ldots S_{n-1} \in \mathsf{IO}(\sigma_2, D; \mathbb{R}^{d_{n-1}}, \mathbb{R}^{d_n}), \\ &\kappa_n \in \mathsf{N}_l(\sigma_3; \mathbb{R}^{d'} \times \mathbb{R}^d, \mathbb{R}^{d_u \times d_n}), \; d_1, \ldots, d_n, l \in \mathbb{N}\}. \end{aligned}$$

When $d_a = d_u = 1$, we will simply write $\mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3; D, D')$. Since σ_1 is linearly bounded, we can use a result about compositions of maps in L^p spaces such as R.M. Dudley and Norvaiša, 2010, Theorem 7.13 to conclude that any $G \in \mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3, D, D'; \mathbb{R}^{d_a}, \mathbb{R}^{d_u})$ acts as $G : L^p(D; \mathbb{R}^{d_a}) \to L^p(D'; \mathbb{R}^{d_u})$. Note that it is only in the last layer that we transition from functions defined over domain D to functions defined over domain D'.

When the input space of an operator of interest is $C^m(\bar{D})$, for $m \in \mathbb{N}$, we will need to take in derivatives explicitly as they cannot be learned using kernel integration as employed in the current construction given in Lemma 55; note that this is *not* the case for $W^{m,p}(D)$ as shown in Lemma 53. We will therefore define the set of m-th order neural operators by

$$\mathsf{NO}_n^m(\sigma_1, \sigma_2, \sigma_3; D, D', \mathbb{R}^{d_a}, \mathbb{R}^{d_u}) = \{(\partial^{\alpha_1} f, \ldots, \partial^{\alpha_{J_m}} f) \mapsto G(\partial^{\alpha_1} f, \ldots, \partial^{\alpha_{J_m}} f) : \\ G \in \mathsf{NO}_n(\sigma_1, \sigma_2, \sigma_3; D, D', \mathbb{R}^{J_m d_a}, \mathbb{R}^{d_u})\}$$

where $\alpha_1, \ldots, \alpha_{J_m} \in \mathbb{N}^d$ is an enumeration of the set $\{\alpha \in \mathbb{N}^d : 0 \leq |\alpha|_1 \leq m\}$. Since we only use the m-th order operators when dealing with spaces of continuous

$$\begin{tikzcd}
\mathcal{A} \arrow[r, "F"] \arrow[d, "\mathcal{G}^\dagger"'] & \mathbb{R}^J \arrow[r] \arrow[d, "\psi"'] & \mathcal{A} \arrow[d, "\mathcal{G}^\dagger"'] \\
\mathcal{U} \arrow[r] & \mathbb{R}^{J'} \arrow[r, "G"] & \mathcal{U}
\end{tikzcd}$$

Figure 52: A schematic overview of the maps used to approximate $\mathcal{G}^\dagger$.

functions, each element of NO_n^m can be thought of as a mapping from a product space of spaces of the form $C^{m-|\alpha_j|}(\bar{D};\mathbb{R}^{d_a})$ for all $j \in \{1,\dots,J_m\}$ to an appropriate Banach space of interest.

Main Theorems

Let $\mathcal{A}$ and $\mathcal{U}$ be Banach spaces of functions on the domains $D \subset \mathbb{R}^d$ and $D' \subset \mathbb{R}^{d'}$ respectively. We will work in the setting where functions in $\mathcal{A}$ or $\mathcal{U}$ are real-valued, but note that all results generalize in a straightforward fashion to the vector-valued setting. We are interested in the approximation of nonlinear operators $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ by neural operators. We will make the following assumptions on the spaces $\mathcal{A}$ and $\mathcal{U}$.

Assumption 34. *Let $D \subset \mathbb{R}^d$ be a Lipschitz domain for some $d \in \mathbb{N}$. One of the following holds*

1. *$\mathcal{A} = L^{p_1}(D)$ for some $1 \leq p_1 < \infty$.*
2. *$\mathcal{A} = W^{m_1,p_1}(D)$ for some $1 \leq p_1 < \infty$ and $m_1 \in \mathbb{N}$,*
3. *$\mathcal{A} = C(\bar{D})$.*

Assumption 35. *Let $D' \subset \mathbb{R}^{d'}$ be a Lipschitz domain for some $d' \in \mathbb{N}$. One of the following holds*

1. *$\mathcal{U} = L^{p_2}(D')$ for some $1 \leq p_2 < \infty$, and $m_2 = 0$,*
2. *$\mathcal{U} = W^{m_2,p_2}(D')$ for some $1 \leq p_2 < \infty$ and $m_2 \in \mathbb{N}$,*
3. *$\mathcal{U} = C^{m_2}(\bar{D}')$ and $m_2 \in \mathbb{N}_0$.*

We first show that neural operators are dense in the continuous operators $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ in the topology of uniform convergence on compacta. The proof proceeds by making three main approximations which are schematically shown in Figure 52. First, inputs are mapped to a finite-dimensional representation through a set of appropriate linear functionals on $\mathcal{A}$ denoted by $F : \mathcal{A} \to \mathbb{R}^J$. We show in Lemmas 21 and 23 that, when $\mathcal{A}$ satisfies Assumption 34, elements of $\mathcal{A}^*$ can be approximated by integration against smooth functions. This generalizes the idea from (T. Chen and H. Chen, 1995) where functionals on $C(\bar{D})$ are approximated by a weighted sum of Dirac measures. We then show in Lemma 25 that, by lifting the dimension, this representation can be approximated by a single element of IO. Second, the representation is non-linearly mapped to a new representation by a continuous function $\psi : \mathbb{R}^J \to \mathbb{R}^{J'}$ which finite-dimensionalizes the action of $\mathcal{G}^\dagger$. We show, in Lemma 28, that this map can be approximated by a neural operator by reducing the architecture to that of a standard neural network. Third, the new representation is used as the coefficients of an expansion onto representers of $\mathcal{U}$, the map denoted $G : \mathbb{R}^{J'} \to \mathcal{U}$, which we show can be approximated by a single IO layer in Lemma 27 using density results for continuous functions. The structure of the overall approximation is similar to (Bhattacharya et al., 2020) but generalizes the ideas from working on Hilbert spaces to the spaces in Assumptions 34 and 35. Statements and proofs of the lemmas used in the theorems are given in the appendices.

Theorem 36. *Let Assumptions 34 and 35 hold and suppose $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ is continuous. Let $\sigma_1 \in \mathsf{A}_0^{\mathsf{L}}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any compact set $K \subset \mathcal{A}$ and $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}(a)\|_{\mathcal{U}} \leq \epsilon.$$

Furthermore, if $\mathcal{U}$ is a Hilbert space and $\sigma_1 \in \mathsf{BA}$ and, for some $M > 0$, we have that $\|\mathcal{G}^\dagger(a)\|_{\mathcal{U}} \leq M$ for all $a \in \mathcal{A}$ then $\mathcal{G}$ can be chosen so that

$$\|\mathcal{G}(a)\|_{\mathcal{U}} \leq 4M, \qquad \forall a \in \mathcal{A}.$$

Proof. Lemma 51 allows us to apply Lemma 47 to find a mapping $\mathcal{G}_1 : \mathcal{A} \to \mathcal{U}$ such that

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}_1(a)\|_{\mathcal{U}} \leq \frac{\epsilon}{2}$$

where $\mathcal{G}_1 = G \circ \psi \circ F$ with $F : \mathcal{A} \to \mathbb{R}^J$, $G : \mathbb{R}^{J'} \to \mathcal{U}$ continuous linear maps and $\psi \in C(\mathbb{R}^J; \mathbb{R}^{J'})$ for some $J, J' \in \mathbb{N}$. By Lemma 57, we can find a sequence of maps $F_t \in \mathsf{IO}(\sigma_2; D, \mathbb{R}, \mathbb{R}^J)$ for $t = 1, 2, \dots$ such that

$$\sup_{a \in K} \sup_{x \in \bar{D}} |\big(F_t(a)\big)(x) - F(a)|_1 \leq \frac{1}{t}.$$

In particular, $F_t(a)(x) = w_t(a)\mathbb{1}(x)$ for some $w_t : \mathcal{A} \to \mathbb{R}^J$ which is constant in space. We can therefore identify the range of $F_t(a)$ with $\mathbb{R}^J$. Define the set

$$Z \coloneqq \bigcup_{t=1}^{\infty} F_t(K) \cup F(K) \subset \mathbb{R}^J$$

which is compact by Lemma 46. Since ψ is continuous, it is uniformly continuous on Z hence there exists a modulus of continuity $\omega : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is continuous, non-negative, and non-decreasing on $\mathbb{R}_{\geq 0}$, satisfies $\omega(s) \to \omega(0) = 0$ as $s \to 0$ and

$$|\psi(z_1) - \psi(z_2)|_1 \leq \omega(|z_1 - z_2|_1) \qquad \forall z_1, z_2 \in Z.$$

We can thus find $T \in \mathbb{N}$ large enough such that

$$\sup_{a \in K} \omega(|F(a) - F_T(a)|_1) \leq \frac{\epsilon}{6\|G\|}.$$

Since F_T is continuous, $F_T(K)$ is compact. Then, by Lemma 60, we can find $S_1 \in \mathsf{IO}(\sigma_1; D, \mathbb{R}^J, \mathbb{R}^{d_1}), \dots, S_{N-1} \in \mathsf{IO}(\sigma_1; D, \mathbb{R}^{d_{N-1}}, \mathbb{R}^{J'})$ for some $N \in \mathbb{N}_{\geq 2}$ and $d_1, \dots, d_{N-1} \in \mathbb{N}$ such that

$$\tilde{\psi}(f) \coloneqq \big(S_{N-1} \circ \sigma_1 \circ \cdots \circ S_2 \circ \sigma_1 \circ S_1\big)(f), \qquad \forall f \in L^1(D; \mathbb{R}^J)$$

satisfies

$$\sup_{q \in F_T(K)} \sup_{x \in \bar{D}} |\psi(q) - \tilde{\psi}(q\mathbb{1})(x)|_1 \leq \frac{\epsilon}{6\|G\|}.$$

By construction, $\tilde{\psi}$ maps constant functions into constant functions and is continuous in the appropriate subspace topology of constant functions hence we can identity it as an element of $C(\mathbb{R}^J; \mathbb{R}^{J'})$ for any input constant function taking values in $\mathbb{R}^J$. Then $(\tilde{\psi} \circ F_T)(K) \subset \mathbb{R}^{J'}$ is compact. Therefore, by Lemma 59, we can find a neural network $\kappa \in \mathcal{N}_L(\sigma_3; \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \mathbb{R}^{1 \times J'})$ for some $L \in \mathbb{N}$ such that

$$\tilde{G}(f) \coloneqq \int_{D'} \kappa(\cdot, y) f(y) \, \mathrm{d}y, \qquad \forall f \in L^1(D; \mathbb{R}^{J'})$$

satisfies

$$\sup_{y \in (\tilde{\psi} \circ F_T)(K)} \|G(y) - \tilde{G}(y\mathbb{1})\|_{\mathcal{U}} \leq \frac{\epsilon}{6}.$$

Define

$$\mathcal{G}(a) \coloneqq \big(\tilde{G} \circ \tilde{\psi} \circ F_T\big)(a) = \int_{D'} \kappa(\cdot, y)\big((S_{N-1} \circ \sigma_1 \circ \dots \sigma_1 \circ S_1 \circ F_T)(a)\big)(y) \mathsf{d}y, \quad \forall a \in \mathcal{A},$$

noting that $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$. For any $a \in K$, define $a_1 \coloneqq (\psi \circ F)(a)$ and $\tilde{a}_1 \coloneqq (\tilde{\psi} \circ F_T)(a)$ so that $\mathcal{G}_1(a) = G(a_1)$ and $\mathcal{G}(a) = \tilde{G}(\tilde{a}_1)$ then

$$\begin{aligned}
\|\mathcal{G}_1(a) - \mathcal{G}(a)\|_{\mathcal{U}} &\leq \|G(a_1) - G(\tilde{a}_1)\|_{\mathcal{U}} + \|G(\tilde{a}_1) - \tilde{G}(\tilde{a}_1)\|_{\mathcal{U}} \\
&\leq \|G\||a_1 - \tilde{a}_1|_1 + \sup_{y \in (\tilde{\psi} \circ F_T)(K)} \|G(y) - \tilde{G}(y\mathbb{1})\|_{\mathcal{U}} \\
&\leq \frac{\epsilon}{6} + \|G\||(\psi \circ F)(a) - (\psi \circ F_T)(a)|_1 \\
&\quad + \|G\||(\psi \circ F_T)(a) - (\tilde{\psi} \circ F_T)(a)|_1 \\
&\leq \frac{\epsilon}{6} + \|G\|\omega\big(|F(a) - F_T(a)|_1\big) + \|G\| \sup_{q \in F_T(K)} |\psi(q) - \tilde{\psi}(q)|_1 \\
&\leq \frac{\epsilon}{2}.
\end{aligned}$$

Finally we have

$$\|\mathcal{G}^\dagger(a) - \mathcal{G}(a)\|_{\mathcal{U}} \leq \|\mathcal{G}^\dagger(a) - \mathcal{G}_1(a)\|_{\mathcal{U}} + \|\mathcal{G}_1(a) - \mathcal{G}(a)\|_{\mathcal{U}} \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$$

as desired.

To show boundedness, we will exhibit a neural operator $\tilde{\mathcal{G}}$ that is ϵ-close to $\mathcal{G}$ in K and is uniformly bounded by $4M$. Note first that

$$\|\mathcal{G}(a)\|_{\mathcal{U}} \leq \|\mathcal{G}(a) - \mathcal{G}^\dagger(a)\|_{\mathcal{U}} + \|\mathcal{G}^\dagger(a)\|_{\mathcal{U}} \leq \epsilon + M \leq 2M, \qquad \forall a \in K$$

where, without loss of generality, we assume that $M \geq 1$. By construction, we have that

$$\mathcal{G}(a) = \sum_{j=1}^{J'} \tilde{\psi}_j(F_T(a))\varphi_j, \qquad \forall a \in \mathcal{A}$$

for some neural network $\varphi : \mathbb{R}^{d'} \to \mathbb{R}^{J'}$. Since $\mathcal{U}$ is a Hilbert space and by linearity, we may assume that the components φ_j are orthonormal since orthonormalizing them only requires multiplying the last layers of $\tilde{\psi}$ by an invertible linear map. Therefore

$$|\tilde{\psi}(F_T(a))|_2 = \|\mathcal{G}(a)\|_{\mathcal{U}} \leq 2M, \qquad \forall a \in K.$$

Define the set $W \coloneqq (\tilde{\psi} \circ F_T)(K) \subset \mathbb{R}^{J'}$ which is compact as before. We have

$$\text{diam}_2(W) = \sup_{x,y \in W} |x - y|_2 \leq \sup_{x,y \in W} |x|_2 + |y|_2 \leq 4M.$$

Since $\sigma_1 \in \mathsf{BA}$, there exists a number $R \in \mathbb{N}$ and a neural network $\beta \in \mathsf{N}_R(\sigma_1; \mathbb{R}^{J'}, \mathbb{R}^{J'})$ such that

$$\begin{aligned}
|\beta(x) - x|_2 &\leq \epsilon, \qquad \forall x \in W \\
|\beta(x)|_2 &\leq 4M, \quad \forall x \in \mathbb{R}^{J'}.
\end{aligned}$$

Define

$$\tilde{\mathcal{G}}(a) \coloneqq \sum_{j=1}^{J'} \beta_j(\tilde{\psi}(F_T(a)))\varphi_j, \qquad \forall a \in \mathcal{A}.$$

Lemmas 59 and 60 then shows that $\tilde{\mathcal{G}} \in \mathsf{NO}_{N+R}(\sigma_1, \sigma_2, \sigma_3; D, D')$. Notice that

$$\sup_{a \in K} \|\mathcal{G}(a) - \tilde{\mathcal{G}}(a)\|_{\mathcal{U}} \leq \sup_{w \in W} |w - \beta(w)|_2 \leq \epsilon.$$

Furthermore,

$$\|\tilde{\mathcal{G}}(a)\|_{\mathcal{U}} \leq \|\tilde{\mathcal{G}}(a) - \mathcal{G}(a)\|_{\mathcal{U}} + \|\mathcal{G}(a)\|_{\mathcal{U}} \leq \epsilon + 2M \leq 3M, \qquad \forall a \in K.$$

Let $a \in \mathcal{A} \setminus K$ then there exists $q \in \mathbb{R}^{J'} \setminus W$ such that $\tilde{\psi}(F_T(a)) = q$ and

$$\|\tilde{\mathcal{G}}(a)\|_{\mathcal{U}} = |\beta(q)|_2 \leq 4M$$

as desired. □

We extend this result to the case $\mathcal{A} = C^{m_1}(\bar{D})$, showing density of the m_1-th order neural operators.

Theorem 37. *Let $D \subset \mathbb{R}^d$ be a Lipschitz domain, $m_1 \in \mathbb{N}$, define $\mathcal{A} := C^{m_1}(\bar{D})$, suppose Assumption 35 holds and assume that $\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ is continuous. Let $\sigma_1 \in \mathsf{A}_0^{\mathsf{L}}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any compact set $K \subset \mathcal{A}$ and $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N^{m_1}(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\sup_{a \in K} \|\mathcal{G}^\dagger(a) - \mathcal{G}(a)\|_{\mathcal{U}} \leq \epsilon.$$

Furthermore, if $\mathcal{U}$ is a Hilbert space and $\sigma_1 \in \mathsf{BA}$ and, for some $M > 0$, we have that $\|\mathcal{G}^\dagger(a)\|_{\mathcal{U}} \leq M$ for all $a \in \mathcal{A}$ then $\mathcal{G}$ can be chosen so that

$$\|\mathcal{G}(a)\|_{\mathcal{U}} \leq 4M, \qquad \forall a \in \mathcal{A}.$$

Proof. The proof follows as in Theorem 36, replacing the use of Lemma 57 with Lemma 58. □

With these results in hand, we show density of neural operators in the space $L^2_\mu(\mathcal{A};\mathcal{U})$ where μ is a probability measure and $\mathcal{U}$ is a separable Hilbert space. The Hilbertian structure of $\mathcal{U}$ allows us to uniformly control the norm of the approximation due to the isomorphism with ℓ_2 as shown in Theorem 36. It remains an interesting future direction to obtain similar results for Banach spaces. The proof follows the ideas in (Lanthaler, Mishra, and George Em Karniadakis, 2021) where similar results are obtained for DeepONet(s) on $L^2(D)$ by using Lusin's theorem to restrict the approximation to a large enough compact set and exploit the decay of μ outside it. Bhattacharya et al., 2020 also employ a similar approach but explicitly constructs the necessary compact set after finite-dimensionalizing.

Theorem 38. *Let $D' \subset \mathbb{R}^{d'}$ be a Lipschitz domain, $m_2 \in \mathbb{N}_0$, and suppose Assumption 34 holds. Let μ be a probability measure on $\mathcal{A}$ and suppose $\mathcal{G}^\dagger : \mathcal{A} \to H^{m_2}(D)$ is μ-measurable and $\mathcal{G}^\dagger \in L^2_\mu(\mathcal{A}; H^{m_2}(D))$. Let $\sigma_1 \in \mathsf{A}_0^{\mathsf{L}} \cap \mathsf{BA}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\|\mathcal{G}^\dagger - \mathcal{G}\|_{L^2_\mu(\mathcal{A}; H^{m_2}(D))} \leq \epsilon.$$

Proof. Let $\mathcal{U} = H^{m_2}(D)$. For any $R > 0$, define

$$\mathcal{G}^\dagger_R(a) := \begin{cases} \mathcal{G}^\dagger(a), & \|\mathcal{G}^\dagger(a)\|_\mathcal{U} \leq R \\ \frac{R}{\|\mathcal{G}^\dagger(a)\|_\mathcal{U}} \mathcal{G}^\dagger(a), & \text{otherwise} \end{cases}$$

for any $a \in \mathcal{A}$. Since $\mathcal{G}^\dagger_R \to \mathcal{G}^\dagger$ as $R \to \infty$ μ-almost everywhere, $\mathcal{G}^\dagger \in L^2_\mu(\mathcal{A}; \mathcal{U})$, and clearly $\|\mathcal{G}^\dagger_R(a)\|_\mathcal{U} \leq \|\mathcal{G}^\dagger(a)\|_\mathcal{U}$ for any $a \in \mathcal{A}$, we can apply the dominated convergence theorem for Bochner integrals to find $R > 0$ large enough such that

$$\|\mathcal{G}^\dagger_R - \mathcal{G}^\dagger\|_{L^2_\mu(\mathcal{A};\mathcal{U})} \leq \frac{\epsilon}{3}.$$

Since $\mathcal{A}$ and $\mathcal{U}$ are Polish spaces, by Lusin's theorem Aaronson, 1997, Theorem 1.0.0 we can find a compact set $K \subset \mathcal{A}$ such that

$$\mu(\mathcal{A} \setminus K) \leq \frac{\epsilon^2}{153R^2}$$

and $\mathcal{G}^\dagger_R|_K$ is continuous. Since K is closed, by a generalization of the Tietze extension theorem Dugundji, 1951, Theorem 4.1, there exist a continuous mapping $\tilde{\mathcal{G}}^\dagger_R : \mathcal{A} \to \mathcal{U}$ such that $\tilde{\mathcal{G}}^\dagger_R(a) = \mathcal{G}^\dagger_R(a)$ for all $a \in K$ and

$$\sup_{a \in \mathcal{A}} \|\tilde{\mathcal{G}}^\dagger_R(a)\| \leq \sup_{a \in \mathcal{A}} \|\mathcal{G}^\dagger_R(a)\| \leq R.$$

Applying Theorem 36 to $\tilde{\mathcal{G}}^\dagger_R$, we find that there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that

$$\sup_{a \in K} \|\mathcal{G}(a) - \mathcal{G}^\dagger_R(a)\|_\mathcal{U} \leq \frac{\sqrt{2}\epsilon}{3}$$

and

$$\sup_{a \in \mathcal{A}} \|\mathcal{G}(a)\|_{\mathcal{U}} \leq 4R.$$

We then have

$$\begin{aligned}
\|\mathcal{G}^\dagger - \mathcal{G}\|_{L^2_\mu(\mathcal{A};\mathcal{U})} &\leq \|\mathcal{G}^\dagger - \mathcal{G}^\dagger_R\|_{L^2_\mu(\mathcal{A};\mathcal{U})} + \|\mathcal{G}^\dagger_R - \mathcal{G}\|_{L^2_\mu(\mathcal{A};\mathcal{U})} \\
&\leq \frac{\epsilon}{3} + \bigg(\int_K \|\mathcal{G}^\dagger_R(a) - \mathcal{G}(a)\|^2_{\mathcal{U}} \, \mathsf{d}\mu(a) \\
&\quad + \int_{\mathcal{A} \setminus K} \|\mathcal{G}^\dagger_R(a) - \mathcal{G}(a)\|^2_{\mathcal{U}} \, \mathsf{d}\mu(a) \bigg)^{\frac{1}{2}} \\
&\leq \frac{\epsilon}{3} + \left(\frac{2\epsilon^2}{9} + 2 \left(\sup_{a \in \mathcal{A}} \|\mathcal{G}^\dagger_R(a)\|^2_{\mathcal{U}} + \|\mathcal{G}(a)\|^2_{\mathcal{U}} \right) \mu(\mathcal{A} \setminus K) \right)^{\frac{1}{2}} \\
&\leq \frac{\epsilon}{3} + \left(\frac{2\epsilon^2}{9} + 34R^2 \mu(\mathcal{A} \setminus K) \right)^{\frac{1}{2}} \\
&\leq \frac{\epsilon}{3} + \left(\frac{4\epsilon^2}{9} \right)^{\frac{1}{2}} \\
&= \epsilon
\end{aligned}$$

as desired. □

We again extend this to the case $\mathcal{A} = C^{m_1}(D)$ using the m_1-th order neural operators.

Theorem 39. *Let $D \subset \mathbb{R}^d$ be a Lipschitz domain, $m_1 \in \mathbb{N}$, define $\mathcal{A} := C^{m_1}(D)$ and suppose Assumption 35 holds. Let μ be a probability measure on $C^{m_1}(D)$ and let $\mathcal{G}^\dagger : C^{m_1}(D) \to \mathcal{U}$ be μ-measurable and suppose $\mathcal{G}^\dagger \in L^2_\mu(C^{m_1}(D);\mathcal{U})$. Let $\sigma_1 \in \mathsf{A}_0^{\mathsf{L}} \cap \mathsf{BA}$, $\sigma_2 \in \mathsf{A}_0$, and $\sigma_3 \in \mathsf{A}_{m_2}$. Then for any $0 < \epsilon \leq 1$, there exists a number $N \in \mathbb{N}$ and a neural operator $\mathcal{G} \in \mathsf{NO}_N^{m_1}(\sigma_1, \sigma_2, \sigma_3; D, D')$ such that*

$$\|\mathcal{G}^\dagger - \mathcal{G}\|_{L^2_\mu(C^{m_1}(D);\mathcal{U})} \leq \epsilon.$$

Proof. The proof follows as in Theorem 38 by replacing the use of Theorem 36 with Theorem 37. □

5.5 Parameterization and Computation

In this section, we discuss various ways of parameterizing the infinite dimensional architecture (5.5). The goal is to find an intrinsic infinite dimensional paramterization that achieves small error (say ϵ), and then rely on numerical approximation to ensure that this parameterization delivers an error of the same magnitude (say 2ϵ), for all data discretizations fine enough. In this way the number of parameters used to achieve $\mathcal{O}(\epsilon)$ error is independent of the data discretization. In many applications we have in mind the data discretization is something we can control, for example when generating input/output pairs from solution of partial differential equations via numerical simulation. The proposed approach allows us to train a neural operator approximation using data from different discretizations, and to predict with discretizations different from those used in the data, all by relating everything to the underlying infinite dimensional problem.

We also discuss the computational complexity of the proposed parameterizations and suggest algorithms which yield efficient numerical methods for approximation. Subsections 5.5-5.5 delineate each of the proposed methods.

To simplify notation, we will only consider a single layer of (5.5), i.e. (5.9) and choose the input and output domains to be the same. Furthermore, we will drop the layer index t and write the single layer update as

$$u(x) = \sigma\left(Wv(x) + \int_D \kappa(x,y)v(y)\,\mathrm{d}\nu(y) + b(x)\right) \qquad \forall x \in D \tag{5.15}$$

where $D \subset \mathbb{R}^d$ is a bounded domain, $v : D \to \mathbb{R}^n$ is the input function and $u : D \to \mathbb{R}^m$ is the output function.

When the domain domains D of v and u are different, we will usually extend them to be on a larger domain. We will consider σ to be fixed, and, for the time being, take $\mathrm{d}\nu(y) = \mathrm{d}y$ to be the Lebesgue measure on $\mathbb{R}^d$. Equation (5.15) then leaves three objects which can be parameterized: W, κ, and b. Since W is linear and acts only locally on v, we will always parametrize it by the values of its associated $m \times n$ matrix; hence $W \in \mathbb{R}^{m\times n}$ yielding mn parameters. We have found empirically that letting $b : D \to \mathbb{R}^m$ be a constant function over any domain D works at least as well as allowing it to be an arbitrary neural network. Perusal of the proof of Theorem 36 shows that we do not lose any approximation power by doing this, and we reduce the total number of parameters in the architecutre. Therefore we will always parametrize b by the entries of a fixed m-dimensional vector; in particular, $b \in \mathbb{R}^m$ yielding m parameters. Notice that both parameterizations are independent of any discretization of v.

The rest of this section will be dedicated to choosing the kernel function $\kappa : D \times D \to \mathbb{R}^{m\times n}$ and the computation of the associated integral kernel operator. For clarity of exposition, we consider only the simplest proposed version of this operator (5.6) but note that similar ideas may also be applied to (5.7) and (5.8). Furthermore, in order to focus on learning the kernel κ, here we drop σ, W, and b from (5.15) and simply consider the linear update

$$u(x) = \int_D \kappa(x,y)v(y)\,\mathrm{d}\nu(y) \qquad \forall x \in D. \tag{5.16}$$

To demonstrate the computational challenges associated with (5.16), let $\{x_1,\dots,x_J\} \subset D$ be a uniformly-sampled J-point discretization of D. Recall that we assumed $\mathrm{d}\nu(y) = \mathrm{d}y$ and, for simplicity, suppose that $\nu(D) = 1$, then the Monte Carlo approximation of (5.16) is

$$u(x_j) = \frac{1}{J}\sum_{l=1}^{J} \kappa(x_j,x_l)v(x_l), \qquad j = 1,\dots,J.$$

Therefore to compute u on the entire grid requires $\mathcal{O}(J^2)$ matrix-vector multiplications. Each of these matrix-vector multiplications requires $\mathcal{O}(mn)$ operations; for the rest of the discussion, we treat $mn = \mathcal{O}(1)$ as constant and consider only the cost with respect to J the discretization parameter since m and n are fixed by the architecture choice whereas J varies depending on required discretization accuracy and hence may be arbitrarily large. This cost is not specific to the Monte Carlo approximation but is generic for quadrature rules which use the entirety of the data. Therefore, when J is large, computing (5.16) becomes intractable and new ideas are needed in order to alleviate this. Subsections 5.5-5.5 propose different approaches to the solution to this problem, inspired by classical methods in numerical analysis. We finally remark that, in contrast, computations with W, b, and σ only require $\mathcal{O}(J)$ operations which justifies our focus on computation with the kernel integral operator.

Kernel Matrix. It will often times be useful to consider the kernel matrix associated to κ for the discrete points $\{x_1,\dots,x_J\} \subset D$. We define the kernel matrix $K \in \mathbb{R}^{mJ\times nJ}$ to be the $J \times J$ block matrix with each block given by the value of the kernel, i.e.

$$K_{jl} = \kappa(x_j,x_l) \in \mathbb{R}^{m\times n}, \qquad j,l = 1,\dots,J$$

where we use (j,l) to index an individual block rather than a matrix element. Various numerical algorithms for the efficient computation of (5.16) can be derived based on assumptions made about the structure of this matrix, for example, bounds on its rank or sparsity.

Graph Neural Operator (GNO)

We first outline the Graph Neural Operator (GNO) which approximates (5.16) by combining a Nyström approximation with domain truncation and is implemented with the standard framework of graph neural networks. This construction was originally proposed in Z. Li, Kovachki, et al., 2020c.

Nyström Approximation. A simple yet effective method to alleviate the cost of computing (5.16) is employing a Nyström approximation. This amounts to sampling uniformly at random the points over which we compute the output function u. In particular, let $x_{k_1}, \dots, x_{k_{J'}} \subset \{x_1, \dots, x_J\}$ be $J' \ll J$ randomly selected points and, assuming $\nu(D) = 1$, approximate (5.16) by

$$u(x_{k_j}) \approx \frac{1}{J'} \sum_{l=1}^{J'} \kappa(x_{k_j}, x_{k_l}) v(x_{k_l}), \qquad j = 1, \dots, J'.$$

We can view this as a low-rank approximation to the kernel matrix K, in particular,

$$K \approx K_{JJ'} K_{J'J'} K_{J'J} \tag{5.17}$$

where $K_{J'J'}$ is a $J' \times J'$ block matrix and $K_{JJ'}$, $K_{J'J}$ are interpolation matrices, for example, linearly extending the function to the whole domain from the random nodal points. The complexity of this computation is $\mathcal{O}(J'^2)$ hence it remains quadratic but only in the number of subsampled points J' which we assume is much less than the number of points J in the original discretization.

Truncation. Another simple method to alleviate the cost of computing (5.16) is to truncate the integral to a sub-domain of D which depends on the point of evaluation $x \in D$. Let $s : D \to \mathcal{B}(D)$ be a mapping of the points of D to the Lebesgue measurable subsets of D denoted $\mathcal{B}(D)$. Define $\mathrm{d}\nu(x, y) = \mathbb{1}_{s(x)} \mathrm{d}y$ then (5.16) becomes

$$u(x) = \int_{s(x)} \kappa(x, y) v(y) \, \mathrm{d}y \qquad \forall x \in D. \tag{5.18}$$

If the size of each set $s(x)$ is smaller than D then the cost of computing (5.18) is $\mathcal{O}(c_s J^2)$ where $c_s < 1$ is a constant depending on s. While the cost remains quadratic in J, the constant c_s can have a significant effect in practical computations, as we demonstrate in Section 5.7. For simplicity and ease of implementation, we only consider $s(x) = B(x, r) \cap D$ where $B(x, r) = \{y \in \mathbb{R}^d : \|y - x\|_{\mathbb{R}^d} < r\}$ for some fixed $r > 0$. With this choice of s and assuming that $D = [0, 1]^d$, we can explicitly calculate that $c_s \approx r^d$.

Furthermore notice that we do not lose any expressive power when we make this approximation so long as we combine it with composition. To see this, consider the example of the previous paragraph where, if we let $r = \sqrt{2}$, then (5.18) reverts to (5.16). Pick $r < 1$ and let $L \in \mathbb{N}$ with $L \geq 2$ be the smallest integer such that $2^{L-1}r \geq 1$. Suppose that $u(x)$ is computed by composing the right hand side of (5.18) L times with a different kernel every time. The domain of influence of $u(x)$ is then $B(x, 2^{L-1}r) \cap D = D$, hence it is easy to see that there exist L kernels such that computing this composition is equivalent to computing (5.16) for any given kernel with appropriate regularity. Furthermore the cost of this computation is $\mathcal{O}(Lr^d J^2)$ and therefore the truncation is beneficial if $r^d(\log_2 1/r + 1) < 1$ which holds for any $r < 1/2$ when $d = 1$ and any $r < 1$ when $d \geq 2$. Therefore we have shown that we can always reduce the cost of computing (5.16) by truncation and composition. From the perspective of the kernel matrix, truncation enforces a sparse, block diagonally-dominant structure at each layer. We further explore the hierarchical nature of this computation using the multipole method in subsection 5.5.

Besides being a useful computational tool, truncation can also be interpreted as explicitly building local structure into the kernel κ. For problems where such structure exists, explicitly enforcing it makes learning more efficient, usually requiring less data to achieve the same generalization error. Many physical systems such as interacting particles in an electric potential exhibit strong local behavior that quickly decays, making truncation a natural approximation technique.

Graph Neural Networks. We utilize the standard architecture of message passing graph networks employing edge features as introduced in Gilmer et al., 2017 to efficiently implement (5.16) on arbitrary discretizations of the domain D. To do so, we treat a discretization $\{x_1, \dots, x_J\} \subset D$ as the nodes of a weighted, directed graph and assign edges to each node using the function $s : D \to \mathcal{B}(D)$ which, recall from the section on truncation, assigns to each point a domain of integration. In particular, for $j = 1, \dots, J$, we assign the node x_j the value $v(x_j)$ and emanate from it edges to the nodes $s(x_j) \cap \{x_1, \dots, x_J\} = \mathcal{N}(x_j)$ which we call the neighborhood of x_j. If $s(x) = D$ then the graph is fully-connected. Generally, the sparsity structure of the graph determines the sparsity of the kernel matrix K, indeed, the adjacency matrix of the graph and the block kernel matrix have the same zero entries. The weights of each edge are assigned as the arguments of the kernel. In particular, for the case of (5.16), the weight of the edge between nodes x_j and x_k is simply the concatenation $(x_j, x_k) \in \mathbb{R}^{2d}$. More complicated weighting functions are considered for the implementation of the integral kernel operators (5.7) or (5.8).

With the above definition the message passing algorithm of Gilmer et al., 2017, with averaging aggregation, updates the value $v(x_j)$ of the node x_j to the value $u(x_j)$ as

$$u(x_j) = \frac{1}{|\mathcal{N}(x_j)|} \sum_{y \in \mathcal{N}(x_j)} \kappa(x_j, y) v(y), \qquad j = 1, \dots, J$$

which corresponds to the Monte-Carlo approximation of the integral (5.18). More sophisticated quadrature rules and adaptive meshes can also be implemented using the general framework of message passing on graphs, see, for example, Pfaff et al., 2020. We further utilize this framework in subsection 5.5.

Convolutional Neural Networks. Lastly, we compare and contrast the GNO framework to standard convolutional neural networks (CNNs). In computer vision, the success of CNNs has largely been attributed to their ability to capture local features such as edges that can be used to distinguish different objects in a natural image. This property is obtained by enforcing the convolution kernel to have local support, an idea similar to our truncation approximation. Furthermore by directly using a translation invariant kernel, a CNN architecture becomes translation equivariant; this is a desirable feature for many vision models, e.g. ones that perform segmentation. We will show that similar ideas can be applied to the neural operator framework to obtain an architecture with built-in local properties and translational symmetries that, unlike CNNs, remain consistent in function space.

To that end, let $\kappa(x, y) = \kappa(x - y)$ and suppose that $\kappa : \mathbb{R}^d \to \mathbb{R}^{m \times n}$ is supported on $B(0, r)$. Let $r^* > 0$ be the smallest radius such that $D \subseteq B(x^*, r^*)$ where $x^* \in \mathbb{R}^d$ denotes the center of mass of D and suppose $r \ll r^*$. Then (5.16) becomes the convolution

$$u(x) = (\kappa * v)(x) = \int_{B(x,r) \cap D} \kappa(x - y) v(y) \, \mathrm{d}y \qquad \forall x \in D. \tag{5.19}$$

Notice that (5.19) is precisely (5.18) when $s(x) = B(x, r) \cap D$ and $\kappa(x, y) = \kappa(x - y)$. When the kernel is parameterized by e.g. a standard neural network and the radius r is chosen independently of the data discretization, (5.19) becomes a layer of a convolution neural network that is consistent in function space. Indeed the parameters of (5.19) do not depend on any discretization of v. The choice $\kappa(x, y) = \kappa(x - y)$ enforces translational equivariance in the output while picking r small enforces locality in the kernel; hence we obtain the distinguishing features of a CNN model.

We will now show that, by picking a parameterization that is *inconsistent* in function space and applying a Monte Carlo approximation to the integral, (5.19) becomes a standard CNN. This is most easily demonstrated when $D = [0,1]$ and the discretization $\{x_1, \dots, x_J\}$ is equispaced i.e. $|x_{j+1} - x_j| = h$ for any $j = 1, \dots, J-1$. Let $k \in \mathbb{N}$ be an odd filter size and let $z_1, \dots, z_k \in \mathbb{R}$ be the points $z_j = (j - 1 - (k-1)/2)h$ for $j = 1, \dots, k$. It is easy to see that $\{z_1, \dots, z_k\} \subset \bar{B}(0, (k-1)h/2)$ which we choose as the support of κ. Furthermore, we parameterize κ directly by its pointwise values which are $m \times n$ matrices at the locations $z_1, \dots, z_k$, thus yielding kmn parameters. Then (5.19) becomes

$$u(x_j)_p \approx \frac{1}{k} \sum_{l=1}^{k} \sum_{q=1}^{n} \kappa(z_l)_{pq} v(x_j - z_l)_q, \qquad j = 1, \dots, J, \;\; p = 1, \dots, m$$

where we define $v(x) = 0$ if $x \notin \{x_1, \dots, x_J\}$. Up to the constant factor $1/k$ which can be re-absobred into the parameterization of κ, this is precisely the update of a stride 1 CNN with n input channels, m output channels, and zero-padding so that the input and output signals have the same length. This example can easily be generalized to higher dimensions and different CNN structures, we made the current choices for simplicity of exposition. Notice that if we double the amount of discretization points for v, i.e. $J \mapsto 2J$ and $h \mapsto h/2$, the support of κ becomes $\bar{B}(0, (k-1)h/4)$ hence the model changes due to the discretization of the data. Indeed, if we take the limit to the continuum $J \to \infty$, we find $\bar{B}(0, (k-1)h/2) \to \{0\}$ hence the model becomes completely local. To fix this, we may try to increase the filter size k (or equivalently add more layers) simultaneously with J, but then the number of parameters in the model goes to infinity as $J \to \infty$ since, as we previously noted, there are kmn parameters in this layer. Therefore standard CNNs are not consistent models in function space. We demonstrate their inability to generalize to different resolutions in Section 5.7.

Low-rank Neural Operator (LNO)

By directly imposing that the kernel κ is of a tensor product form, we obtain a layer with $\mathcal{O}(J)$ computational complexity. We term this construction the Low-rank Neural Operator (LNO) due to its equivalence to directly parameterizing a finite-rank operator. We start by assuming $\kappa : D \times D \to \mathbb{R}$ is scalar valued and later generalize to the vector valued setting. We express the kernel as

$$\kappa(x,y) = \sum_{j=1}^{r} \varphi^{(j)}(x) \psi^{(j)}(y) \qquad \forall x, y \in D$$

for some functions $\varphi^{(1)}, \psi^{(1)}, \dots, \varphi^{(r)}, \psi^{(r)} : D \to \mathbb{R}$ that are normally given as the components of two neural networks $\varphi, \psi : D \to \mathbb{R}^r$ or a single neural network $\Xi : D \to \mathbb{R}^{2r}$ which couples all functions through its parameters. With this definition, and supposing that $n = m = 1$, we have that (5.16) becomes

$$\begin{aligned} u(x) &= \int_D \sum_{j=1}^{r} \varphi^{(j)}(x)\psi^{(j)}(y)v(y)\, \mathrm{d}y \\ &= \sum_{j=1}^{r} \int_D \psi^{(j)}(y)v(y)\, \mathrm{d}y\, \varphi^{(j)}(x) \\ &= \sum_{j=1}^{r} \langle \psi^{(j)}, v \rangle \varphi^{(j)}(x) \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ denotes the $L^2(D;\mathbb{R})$ inner product. Notice that the inner products can be evaluated independently of the evaluation point $x \in D$ hence the computational complexity of this method is $\mathcal{O}(rJ)$ which is linear in the discretization.

We may also interpret this choice of kernel as directly parameterizing a rank $r \in \mathbb{N}$ operator on $L^2(D;\mathbb{R})$. Indeed, we have

$$u = \sum_{j=1}^{r} (\varphi^{(j)} \otimes \psi^{(j)})v \tag{5.20}$$

which corresponds preceisely to applying the SVD of a rank r operator to the function v. Equation (5.20) makes natural the vector valued generalization. Assume $m, n \geq 1$ and $\varphi^{(j)} : D \to \mathbb{R}^m$ and $\psi^{(j)} : D \to \mathbb{R}^n$ for $j = 1, \dots, r$ then, (5.20) defines an operator mapping $L^2(D;\mathbb{R}^n) \to L^2(D;\mathbb{R}^m)$ that can be evaluated as

$$u(x) = \sum_{j=1}^{r} \langle \psi^{(j)}, v \rangle_{L^2(D;\mathbb{R}^n)} \varphi^{(j)}(x) \qquad \forall x \in D.$$

We again note the linear computational complexity of this parameterization. Finally, we observe that this method can be interpreted as directly imposing a rank r structure on the kernel matrix. Indeed,

$$K = K_{Jr}K_{rJ}$$

where K_{Jr}, K_{rJ} are $J \times r$ and $r \times J$ block matricies respectively. This construction is similar to the DeepONet construction of L. Lu, Jin, and George Em Karniadakis, 2019 discussed in Section 5.3, but parameterized to be consistent in function space. While this method enjoys a linear computational complexity, it also constitutes a *linear* approximation method which may not be able to effectively capture the solution manifold; see Section 5.3 for further discussion.

Multipole Graph Neural Operator (MGNO)

A natural extension to directly working with kernels in a tensor product form as in Section 5.5 is to instead consider kernels that can be well approximated by such a form. This assumption gives rise to the fast multipole method (FMM) which employs a multi-scale decomposition of the kernel in order to achieve linear complexity in computing (5.16); for a detailed discussion see e.g. (E, 2011, Section 3.2). FMM can be viewed as a systematic approach to combine the sparse and low-rank approximations to the kernel matrix. Indeed, the kernel matrix is decomposed into different ranges and a hierarchy of low-rank structures is imposed on the long-range components. We employ this idea to construct hierarchical, multi-scale graphs, without being constrained to particular forms of the kernel. We will elucidate the workings of the FMM through matrix factorization. This approach was first outlined in Z. Li, Kovachki, et al., 2020b and is referred to as the Multipole Graph Neural Operator (MGNO).

The key to the fast multipole method's linear complexity lies in the subdivision of the kernel matrix according to the range of interaction, as shown in Figure 53:

$$K = K_1 + K_2 + \ldots + K_L, \tag{5.21}$$

where K_ℓ with $\ell = 1$ corresponds to the shortest-range interaction, and $\ell = L$ corresponds to the longest-range interaction; more generally index ℓ is ordered by the range of interaction. While the uniform grids depicted in Figure 53 produce an orthogonal decomposition of the kernel, the decomposition may be generalized to arbitrary discretizations by allowing slight overlap of the ranges.

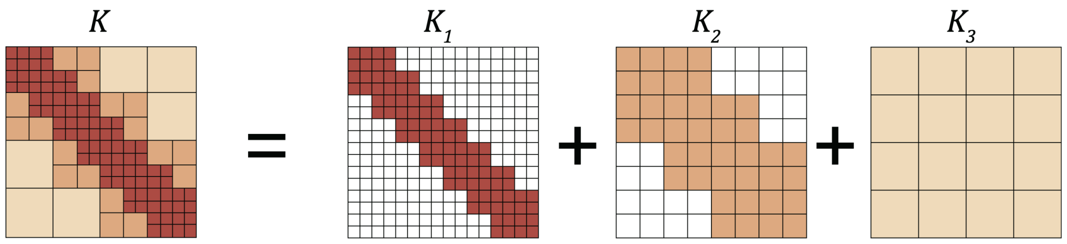


Figure 53: Hierarchical matrix decomposition. The kernel matrix K is decomposed with respect to its interaction ranges. K_1 corresponds to short-range interaction; it is sparse but full-rank. K_3 corresponds to long-range interaction; it is dense but low-rank.

Multi-scale Discretization. We produce a hierarchy of L discretizations with a decreasing number of nodes $J_1 \geq \ldots \geq J_L$ and increasing kernel integration radius

$r_1 \leq \ldots \leq r_L$. Therefore, the shortest-range interaction K_1 has a fine resolution but is truncated locally, while the longest-range interaction K_L has a coarse resolution, but covers the entire domain. This is shown pictorially in Figure 53. The number of nodes $J_1 \geq \ldots \geq J_L$, and the integration radii $r_1 \leq \ldots \leq r_L$ are hyperparameter choices and can be picked so that the total computational complexity is linear in J.

A special case of this construction is when the grid is uniform. Then our formulation reduces to the standard fast multipole algorithm and the kernels K_l form an orthogonal decomposition of the full kernel matrix K. Assuming the underlying discretization $\{x_1, \ldots, x_J\} \subset D$ is a uniform grid with resolution s such that $s^d = J$, the L multi-level discretizations will be grids with resolution $s_l = s/2^{l-1}$, and consequentially $J_l = s_l^d = (s/2^{l-1})^d$. In this case r_l can be chosen as $1/s$ for $l = 1, \ldots, L$. To ensure orthogonality of the discretizations, the fast multipole algorithm sets the integration domains to be $B(x, r_l) \setminus B(x, r_{l-1})$ for each level $l = 2, \ldots, L$, so that the discretization on level l does not overlap with the one on level $l - 1$. Details of this algorithm can be found in (Greengard and Rokhlin, 1997).

Recursive Low-rank Decomposition. The coarse discretization representation can be understood as recursively applying an inducing points approximation (Quiñonero-Candela and Rasmussen, 2005): starting from a discretization with $J_1 = J$ nodes, we impose inducing points of size $J_2, J_3, \ldots, J_L$ which all admit a low-rank kernel matrix decomposition of the form (5.17). The original $J \times J$ kernel matrix K_l is represented by a much smaller $J_l \times J_l$ kernel matrix, denoted by $K_{l,l}$. As shown in Figure 53, K_1 is full-rank but very sparse while K_L is dense but low-rank. Such structure can be achieved by applying equation (5.17) recursively to equation (5.21), leading to the multi-resolution matrix factorization (Kondor, Teneva, and Garg, 2014):

$$K \approx K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \tag{5.22}$$

where $K_{1,1} = K_1$ represents the shortest range, $K_{1,2}K_{2,2}K_{2,1} \approx K_2$, represents the second shortest range, etc. The center matrix $K_{l,l}$ is a $J_l \times J_l$ kernel matrix corresponding to the l-level of the discretization described above. The matrices $K_{l+1,l}, K_{l,l+1}$ are $J_{l+1} \times J_l$ and $J_l \times J_{l+1}$ wide and long respectively block transition matrices. Denote $v_l \in R^{J_l \times n}$ for the representation of the input v at each level of the discretization for $l = 1, \ldots, L$, and $u_l \in R^{J_l \times n}$ for the output (assuming the inputs and outputs has the same dimension). We define the matrices $K_{l+1,l}, K_{l,l+1}$ as moving the representation v_l between different levels of the discretization via an integral kernel that we learn. Combining with the truncation idea introduced in

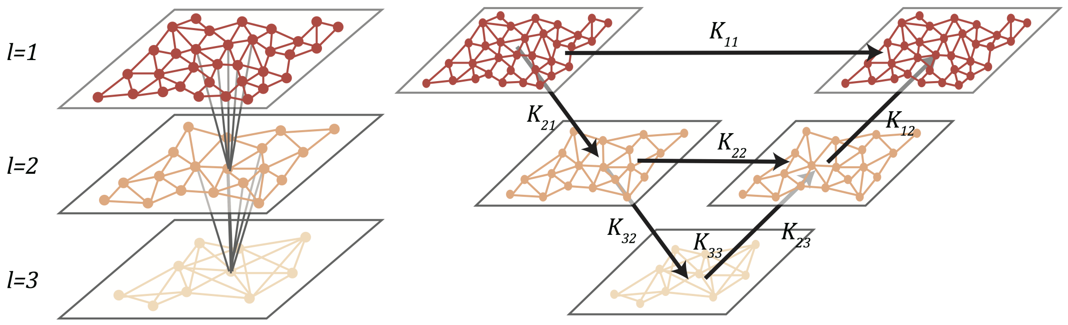


Figure 54: Left: the multi-level discretization. Right: one V-cycle iteration for the multipole neural operator.

subsection 5.5, we define the transition matrices as discretizations of the following integral kernel operators:

$$K_{l,l} : v_l \mapsto u_l = \int_{B(x,r_{l,l})} \kappa_{l,l}(x,y) v_l(y)\, \mathrm{d}y \tag{5.23}$$

$$K_{l+1,l} : v_l \mapsto u_{l+1} = \int_{B(x,r_{l+1,l})} \kappa_{l+1,l}(x,y) v_l(y)\, \mathrm{d}y \tag{5.24}$$

$$K_{l,l+1} : v_{l+1} \mapsto u_l = \int_{B(x,r_{l,l+1})} \kappa_{l,l+1}(x,y) v_{l+1}(y)\, \mathrm{d}y \tag{5.25}$$

where each kernel $\kappa_{l,l'} : D \times D \to \mathbb{R}^{n\times n}$ is parameterized as a neural network and learned.

V-cycle Algorithm We present a V-cycle algorithm, see Figure 54, for efficiently computing (5.22). It consists of two steps: the *downward pass* and the *upward pass*. Denote the representation in downward pass and upward pass by $\check{v}$ and $\hat{v}$ respectively. In the downward step, the algorithm starts from the fine discretization representation $\check{v}_1$ and updates it by applying a downward transition $\check{v}_{l+1} = K_{l+1,l}\check{v}_l$. In the upward step, the algorithm starts from the coarse presentation $\hat{v}_L$ and updates it by applying an upward transition and the center kernel matrix $\hat{v}_l = K_{l,l-1}\hat{v}_{l-1} + K_{l,l}\check{v}_l$. Notice that applying one level downward and upward exactly computes $K_{1,1} + K_{1,2}K_{2,2}K_{2,1}$, and a full L-level V-cycle leads to the multi-resolution decomposition (5.22).

Employing (5.23)-(5.25), we use L neural networks $\kappa_{1,1}, \ldots, \kappa_{L,L}$ to approximate the kernel operators associated to $K_{l,l}$, and $2(L-1)$ neural networks $\kappa_{1,2}, \kappa_{2,1}, \ldots$ to approximate the transitions $K_{l+1,l}, K_{l,l+1}$. Following the iterative architecture (5.5), we introduce the linear operator $W \in \mathbb{R}^{n\times n}$ (denoting it by W_l for each corresponding resolution) to help regularize the iteration, as well as the nonlinear activation function σ to increase the expensiveness. Since W acts pointwise (requiring J remains the same for input and output), we employ it only along with the kernel $K_{l,l}$ and not the transitions. At each layer $t = 0, \dots, T-1$, we perform a full V-cycle as:

- Downward Pass

  For $l = 1, \dots, L$:
  $$\check{v}_{l+1}^{(t+1)} = \sigma(\hat{v}_{l+1}^{(t)} + K_{l+1,l}\check{v}_{l}^{(t+1)}) \tag{5.26}$$

- Upward Pass

  For $l = L, \dots, 1$:
  $$\hat{v}_{l}^{(t+1)} = \sigma((W_l + K_{l,l})\check{v}_{l}^{(t+1)} + K_{l,l-1}\hat{v}_{l-1}^{(t+1)}). \tag{5.27}$$

Notice that one full pass of the V-cycle algorithm defines a mapping $v \mapsto u$.

Multi-level Graphs. We emphasize that we view the discretization $\{x_1, \dots, x_J\} \subset D$ as a graph in order to facilitate an efficient implementation through the message passing graph neural network architecture. Since the V-cycle algorithm works at different levels of the discretization, we build multi-level graphs to represent the coarser and finer discretizations. We present and utilize two constructions of multi-level graphs, the orthogonal multipole graph and the generalized random graph. The orthogonal multipole graph is the standard grid construction used in the fast multiple method which is adapted to a uniform grid, see e.g. Greengard and Rokhlin (1997). In this construction, the decomposition in (5.21) is orthogonal in that the finest graph only captures the closest range interaction, the second finest graph captures the second closest interaction minus the part already captured in the previous graph and so on, recursively. In particular, the ranges of interaction for each kernel do not overlap. While this construction is usually efficient, it is limited to uniform grids which may be a bottleneck for certain applications. Our second construction is the generalized random graph as shown in Figure 53 where the ranges of the kernels are allowed to overlap. The generalized random graph is very flexible as it can be applied on any domain geometry and discretization. Further it can also be combined with random sampling methods to work on problems where J is very large or combined with an active learning method to adaptively choose the regions where a finer discretization is needed.

Linear Complexity. Each term in the decomposition (5.21) is represented by the kernel matrix $K_{l,l}$ for $l = 1, \dots, L$, and $K_{l+1,l}, K_{l,l+1}$ for $l = 1, \dots, L-1$ corresponding to the appropriate sub-discretization. Therefore the complexity of the multipole method is $\sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d) + \sum_{l=1}^{L-1} \mathcal{O}(J_l J_{l+1} r_l^d) = \sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d)$. By designing the sub-discretization so that $\mathcal{O}(J_l^2 r_l^d) \leq \mathcal{O}(J)$, we can obtain complexity linear in J. For example, when $d = 2$, pick $r_l = 1/\sqrt{J_l}$ and $J_l = \mathcal{O}(2^{-l} J)$ such that r_L is large enough so that there exists a ball of radius r_L containing D. Then clearly $\sum_{l=1}^{L} \mathcal{O}(J_l^2 r_l^d) = \mathcal{O}(J)$. By combining with a Nyström approximation, we can obtain $\mathcal{O}(J')$ complexity for some $J' \ll J$.

Fourier Neural Operator (FNO)

Instead of working with a kernel directly on the domain D, we may consider its representation in Fourier space and directly parameterize it there. This allows us to utilize Fast Fourier Transform (FFT) methods in order to compute the action of the kernel integral operator (5.16) with almost linear complexity. A similar idea was used in (Nelsen and Andrew M Stuart, 2021) to construct random features in function space. The method we outline was first described in Z. Li, Kovachki, et al., 2020a and is termed the Fourier Neural Operator (FNO). We note that the theory of Section 4 is designed for general kernels and does not apply to the FNO formulation; however, similar universal approximation results were developed for it in (Kovachki, Lanthaler, and Mishra, 2021) when the input and output spaces are Hilbert space. For simplicity, we will assume that $D = \mathbb{T}^d$ is the unit torus and all functions are complex-valued. Let $\mathcal{F} : L^2(D; \mathbb{C}^n) \to \ell^2(\mathbb{Z}^d; \mathbb{C}^n)$ denote the Fourier transform of a function $v : D \to \mathbb{C}^n$ and $\mathcal{F}^{-1}$ its inverse. For $v \in L^2(D; \mathbb{C}^n)$ and $w \in \ell^2(\mathbb{Z}^d; \mathbb{C}^n)$, we have

$$
\begin{aligned}
(\mathcal{F}v)_j(k) &= \langle v_j, \psi_k \rangle_{L^2(D;\mathbb{C})}, && j \in \{1, \dots, n\}, \quad k \in \mathbb{Z}^d, \\
(\mathcal{F}^{-1}w)_j(x) &= \sum_{k \in \mathbb{Z}^d} w_j(k) \psi_k(x), && j \in \{1, \dots, n\}, \quad x \in D
\end{aligned}
$$

where, for each $k \in \mathbb{Z}^d$, we define

$$
\psi_k(x) = e^{2\pi i k_1 x_1} \cdots e^{2\pi i k_d x_d}, \qquad x \in D
$$

with $i = \sqrt{-1}$ the imaginary unit. By letting $\kappa(x, y) = \kappa(x - y)$ for some $\kappa : D \to \mathbb{C}^{m \times n}$ in (5.16) and applying the convolution theorem, we find that

$$
u(x) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa) \cdot \mathcal{F}(v)\big)(x) \qquad \forall x \in D.
$$

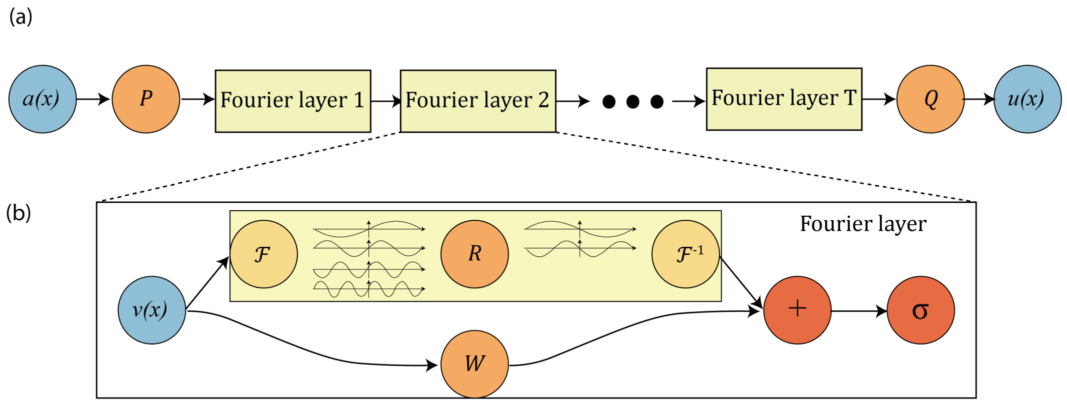


Figure 55: (a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network $\mathcal{P}$. 2. Apply T (typically $T = 4$) layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. (b) Fourier layers: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes which also filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

We therefore propose to directly parameterize κ by its Fourier coefficients. We write

$$u(x) = \mathcal{F}^{-1}\big(R_\phi \cdot \mathcal{F}(v)\big)(x) \qquad \forall x \in D \tag{5.28}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : D \to \mathbb{C}^{m\times n}$ parameterized by some $\phi \in \mathbb{R}^p$.

For frequency mode $k \in \mathbb{Z}^d$, we have $(\mathcal{F}v)(k) \in \mathbb{C}^n$ and $R_\phi(k) \in \mathbb{C}^{m\times n}$. We pick a finite-dimensional parameterization by truncating the Fourier series at a maximal number of modes $k_{\text{max}} = |Z_{k_{\text{max}}}| = |\{k \in \mathbb{Z}^d : |k_j| \le k_{\text{max},j}, \text{ for } j = 1,\dots,d\}|$. We thus parameterize R_ϕ directly as complex-valued $(k_{\text{max}} \times m \times n)$-tensor comprising a collection of truncated Fourier modes and therefore drop ϕ from our notation. In the case where we have real-valued v and we want u to also be real-valued, we impose that κ is real-valued by enforcing conjugate symmetry in the parameterization, i.e.

$$R(-k)_{j,l} = R^*(k)_{j,l} \qquad \forall k \in Z_{k_{\text{max}}}, \quad j = 1,\dots,m, \;\; l = 1,\dots,n.$$

We note that the set $Z_{k_{\text{max}}}$ is not the canonical choice for the low frequency modes of v_t. Indeed, the low frequency modes are usually defined by placing an upper-bound on the ℓ_1-norm of $k \in \mathbb{Z}^d$. We choose $Z_{k_{\text{max}}}$ as above since it allows for an efficient implementation. Figure 55 gives a pictorial representation of an entire Neural Operator architecture employing Fourier layers.

The Discrete Case and the FFT. Assuming the domain D is discretized with $J \in \mathbb{N}$ points, we can treat $v \in \mathbb{C}^{J\times n}$ and $\mathcal{F}(v) \in \mathbb{C}^{J\times n}$. Since we convolve v with a function which only has k_{max} Fourier modes, we may simply truncate the higher modes to obtain $\mathcal{F}(v) \in \mathbb{C}^{k_{\text{max}}\times n}$. Multiplication by the weight tensor $R \in \mathbb{C}^{k_{\text{max}}\times m\times n}$ is then

$$\big(R\cdot(\mathcal{F}v_t)\big)_{k,l} = \sum_{j=1}^{n} R_{k,l,j}(\mathcal{F}v)_{k,j}, \qquad k=1,\dots,k_{\text{max}}, \quad l=1,\dots,m. \tag{5.29}$$

When the discretization is uniform with resolution $s_1\times\cdots\times s_d = J$, $\mathcal{F}$ can be replaced by the Fast Fourier Transform. For $v\in\mathbb{C}^{J\times n}$, $k=(k_1,\dots,k_d)\in\mathbb{Z}_{s_1}\times\cdots\times\mathbb{Z}_{s_d}$, and $x=(x_1,\dots,x_d)\in D$, the FFT $\hat{\mathcal{F}}$ and its inverse $\hat{\mathcal{F}}^{-1}$ are defined as

$$\begin{aligned}(\hat{\mathcal{F}}v)_l(k) &= \sum_{x_1=0}^{s_1-1}\cdots\sum_{x_d=0}^{s_d-1} v_l(x_1,\dots,x_d)e^{-2i\pi\sum_{j=1}^{d}\frac{x_jk_j}{s_j}},\\ (\hat{\mathcal{F}}^{-1}v)_l(x) &= \sum_{k_1=0}^{s_1-1}\cdots\sum_{k_d=0}^{s_d-1} v_l(k_1,\dots,k_d)e^{2i\pi\sum_{j=1}^{d}\frac{x_jk_j}{s_j}}\end{aligned}$$

for $l=1,\dots,n$. In this case, the set of truncated modes becomes

$$Z_{k_{\text{max}}} = \{(k_1,\dots,k_d)\in\mathbb{Z}_{s_1}\times\cdots\times\mathbb{Z}_{s_d} \mid k_j\le k_{\text{max},j} \text{ or } \ s_j-k_j\le k_{\text{max},j}, \\ \text{for } j=1,\dots,d\}.$$

When implemented, R is treated as a $(s_1\times\cdots\times s_d\times m\times n)$-tensor and the above definition of $Z_{k_{\text{max}}}$ corresponds to the “corners” of R, which allows for a straightforward parallel implementation of (5.29) via matrix-vector multiplication. In practice, we have found the choice $k_{\text{max},j}=12$, which yields $k_{\text{max}}=12^d$ parameters per channel, to be sufficient for all the tasks that we consider.

Choices for R. In general, R can be defined to depend on $(\mathcal{F}a)$, the Fourier transform of the input $a\in\mathcal{A}$ to parallel our construction (5.7). Indeed, we can define $R_\phi:\mathbb{Z}^d\times\mathbb{C}^{d_a}\to\mathbb{C}^{m\times n}$ as a parametric function that maps $\big(k,(\mathcal{F}a)(k)\big)$ to the values of the appropriate Fourier modes. We have experimented with the following parameterizations of R_ϕ:

- *Direct.* Define the parameters $\phi_k\in\mathbb{C}^{m\times n}$ for each wave number k:

$$R_\phi\big(k,(\mathcal{F}a)(k)\big) := \phi_k.$$

- *Linear.* Define the parameters $\phi_{k_1} \in \mathbb{C}^{m \times n \times d_a}$, $\phi_{k_2} \in \mathbb{C}^{m \times n}$ for each wave number k:
$$R_\phi\big(k, (\mathcal{F}a)(k)\big) := \phi_{k_1}(\mathcal{F}a)(k) + \phi_{k_2}.$$
- *Feed-forward neural network.* Let $\Phi_\phi : \mathbb{Z}^d \times \mathbb{C}^{d_a} \to \mathbb{C}^{m \times n}$ be a neural network with parameters ϕ:
$$R_\phi\big(k, (\mathcal{F}a)(k)\big) := \Phi_\phi(k, (\mathcal{F}a)(k)).$$

We find that the *linear* parameterization has a similar performance to the *direct* parameterization above, however, it is not as efficient both in terms of computational complexity and the number of parameters required. On the other hand, we find that the *feed-forward neural network* parameterization has a worse performance. This is likely due to the discrete structure of the space $\mathbb{Z}^d$; numerical evidence suggests neural networks are not adept at handling this structure. Our experiments in this work focus on the direct parameterization presented above.

Invariance to Discretization. The Fourier layers are discretization-invariant because they can learn from and evaluate functions which are discretized in an arbitrary way. Since parameters are learned directly in Fourier space, resolving the functions in physical space simply amounts to projecting on the basis elements $e^{2\pi i \langle x, k \rangle}$; these are well-defined everywhere on $\mathbb{C}^d$.

Quasi-linear Complexity. The weight tensor R contains $k_{\text{max}} < J$ modes, so the inner multiplication has complexity $\mathcal{O}(k_{\text{max}})$. Therefore, the majority of the computational cost lies in computing the Fourier transform $\mathcal{F}(v)$ and its inverse. General Fourier transforms have complexity $\mathcal{O}(J^2)$, however, since we truncate the series the complexity is in fact $\mathcal{O}(Jk_{\text{max}})$, while the FFT has complexity $\mathcal{O}(J \log J)$. Generally, we have found using FFTs to be very efficient, however, a uniform discretization is required.

Non-uniform and Non-periodic Geometry. Our implementation of the Fourier neural operator relies on the fast Fourier transform which is only defined on uniform mesh discretizations of $D = \mathbb{T}^d$, or for functions on the square satisfying homogeneous Dirichlet (fast Fourier sine transform) or homogeneous Neumann (fast Fourier cosine transform) boundary conditions. However, the Fourier neural operator can be applied in more general geometries via Fourier continuations. Given any compact manifold $D = \mathcal{M}$, we can always embed it into a periodic cube (torus),

$$i : \mathcal{M} \to \mathbb{T}^d$$

where the regular FFT can be applied. Conventionally, in numerical analysis applications, the embedding i is defined through a continuous extension by fitting polynomials (Bruno, Han, and Pohlman, 2007). However, in the Fourier neural operator, the idea can be applied simply by padding the input with zeros. The loss is computed only on the original space during training. The Fourier neural operator will automatically generate a smooth extension to the padded domain in the output space.

Summary

We summarize the main computational approaches presented in this section and their complexity:

- **GNO**: Subsample J' points from the J-point discretization and compute the truncated integral
$$u(x) = \int_{B(x,r)} \kappa(x, y)v(y) \, \mathrm{d}y \tag{5.30}$$
at a $\mathcal{O}(JJ')$ complexity.

- **LNO**: Decompose the kernel function tensor product form and compute
$$u(x) = \sum_{j=1}^{r} \langle \psi^{(j)}, v \rangle \varphi^{(j)}(x) \tag{5.31}$$
at a $\mathcal{O}(J)$ complexity.

- **MGNO**: Compute a multi-scale decomposition of the kernel
$$\begin{aligned} K &= K_{1,1} + K_{1,2}K_{2,2}K_{2,1} + K_{1,2}K_{2,3}K_{3,3}K_{3,2}K_{2,1} + \cdots \\ u(x) &= (Kv)(x) \end{aligned} \tag{5.32}$$
at a $\mathcal{O}(J)$ complexity.

- **FNO**: Parameterize the kernel in the Fourier domain and compute the using the FFT
$$u(x) = \mathcal{F}^{-1}\big(R_\phi \cdot \mathcal{F}(v)\big)(x) \tag{5.33}$$
at a $\mathcal{O}(J \log J)$ complexity.

5.6 Test Problems

A central application of neural operators is learning solution operators defined by parametric partial differential equations. In this section, we define four test problems for which we numerically study the approximation properties of neural operators. To that end, let $(\mathcal{A}, \mathcal{U}, \mathcal{F})$ be a triplet of Banach spaces. The first two problem classes considered are derived from the following general class of PDEs:

$$\mathsf{L}_a u = f \tag{5.34}$$

where, for every $a \in \mathcal{A}$, $\mathsf{L}_a : \mathcal{U} \to \mathcal{F}$ is a, possibly nonlinear, partial differential operator, and $u \in \mathcal{U}$ corresponds to the solution of the PDE (5.34) when $f \in \mathcal{F}$ and appropriate boundary conditions are imposed. The second class will be evolution equations with initial condition $a \in \mathcal{A}$ and solution $u(t) \in \mathcal{U}$ at every time $t > 0$. We seek to learn the map from a to $u := u(\tau)$ for some fixed time $\tau > 0$; we will also study maps on paths (time-dependent solutions).

Our goal will be to learn the mappings

$$\mathcal{G}^\dagger : a \mapsto u \qquad \text{or} \qquad \mathcal{G}^\dagger : f \mapsto u;$$

we will study both cases, depending on the test problem considered. We will define a probability measure μ on $\mathcal{A}$ or $\mathcal{F}$ which will serve to define a model for likely input data. Furthermore, measure μ will define a topology on the space of mappings in which $\mathcal{G}^\dagger$ lives, using the Bochner norm (5.3). We will assume that each of the spaces $(\mathcal{A}, \mathcal{U}, \mathcal{F})$ are Banach spaces of functions defined on a bounded domain $D \subset \mathbb{R}^d$. All reported errors will be Monte-Carlo estimates of the relative error

$$\mathbb{E}_{a\sim\mu} \frac{\|\mathcal{G}^\dagger(a) - \mathcal{G}_\theta(a)\|_{L^2(D)}}{\|\mathcal{G}^\dagger(a)\|_{L^2(D)}}$$

or equivalently replacing a with f in the above display and with the assumption that $\mathcal{U} \subseteq L^2(D)$. The domain D will be discretized, usually uniformly, with $J \in \mathbb{N}$ points.

Poisson Equation

First we consider the one-dimensional Poisson equation with a zero boundary condition. In particular, (5.34) takes the form

$$\begin{aligned} -\frac{d^2}{dx^2} u(x) &= f(x), \qquad x \in (0,1) \\ u(0) &= u(1) = 0 \end{aligned} \tag{5.35}$$

for some source function $f : (0,1) \to \mathbb{R}$). In particular, for $D(\mathsf{L}) := H_0^1((0,1);\mathbb{R}) \cap H^2((0,1);\mathbb{R})$, we have $\mathsf{L} : D(\mathsf{L}) \to L^2((0,1);\mathbb{R})$ defined as $-d^2/dx^2$, noting that that L has no dependence on any parameter $a \in \mathcal{A}$ in this case. We will consider the weak form of (5.35) with source function $f \in H^{-1}((0,1);\mathbb{R})$ and therefore the solution operator $\mathcal{G}^\dagger : H^{-1}((0,1);\mathbb{R}) \to H_0^1((0,1);\mathbb{R})$ defined as

$$\mathcal{G}^\dagger : f \mapsto u.$$

We define the probability measure $\mu = N(0, C)$ where

$$C = \Big(\mathsf{L} + I\Big)^{-2},$$

defined through the spectral theory of self-adjoint operators. Since μ charges a subset of $L^2((0,1);\mathbb{R})$, we will learn $\mathcal{G}^\dagger : L^2((0,1);\mathbb{R}) \to H_0^1((0,1);\mathbb{R})$ in the topology induced by (5.3).

In this setting, $\mathcal{G}^\dagger$ has a closed-form solution given as

$$\mathcal{G}^\dagger(f) = \int_0^1 G(\cdot, y) f(y) \, \mathrm{d}y$$

where

$$G(x,y) = \frac{1}{2}\left(x + y - |y - x|\right) - xy, \qquad \forall (x,y) \in [0,1]^2$$

is the Green's function. Note that while $\mathcal{G}^\dagger$ is a linear operator, the Green's function G is non-linear as a function of its arguments. We will consider only a single layer of (5.5) with $\sigma_1 = \text{Id}$, $\mathcal{P} = \text{Id}$, $\mathcal{Q} = \text{Id}$, $W_0 = 0$, $b_0 = 0$, and

$$\mathcal{K}_0(f) = \int_0^1 \kappa_\theta(\cdot, y) f(y) \, \mathrm{d}y$$

where $\kappa_\theta : \mathbb{R}^2 \to \mathbb{R}$ will be parameterized as a standard neural network with parameters θ.

The purpose of the current example is two-fold. First we will test the efficacy of the neural operator framework in a simple setting where an exact solution is analytically available. Second we will show that by building in the right inductive bias, in particular, paralleling the form of the Green's function solution, we obtain a model that generalizes outside the distribution μ. That is, once trained, the model will generalize to any $f \in L^2((0,1);\mathbb{R})$ that may be outside the support of μ. For example, as defined, the random variable $f \sim \mu$ is a continuous function, however, if κ_θ approximates the Green's function well then the model $\mathcal{G}^\dagger$ will approximate the solution to (5.35) accurately even for discontinuous inputs.

To create the dataset used for training, solutions to (5.35) are obtained by numerical integration using the Green's function on a uniform grid with 85 collocation points. We use $N = 1000$ training examples.

Darcy Flow

We consider the steady state of Darcy Flow in two dimensions which is the second order elliptic equation

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \qquad && x \in D \\ u(x) &= 0, && x \in \partial D \end{aligned} \tag{5.36}$$

where $D = (0,1)^2$ is the unit square. In this setting $\mathcal{A} = L^\infty(D;\mathbb{R}_+)$, $\mathcal{U} = H_0^1(D;\mathbb{R})$, and $\mathcal{F} = H^{-1}(D;\mathbb{R})$. We fix $f \equiv 1$ and consider the weak form of (5.36) and therefore the solution operator $\mathcal{G}^\dagger : L^\infty(D;\mathbb{R}^+) \to H_0^1(D;\mathbb{R})$ defined as

$$\mathcal{G}^\dagger : a \mapsto u. \tag{5.37}$$

Note that while (5.36) is a linear PDE, the solution operator $\mathcal{G}^\dagger$ is nonlinear. We define the probability measure $\mu = T_\sharp N(0, C)$ where

$$C = (-\Delta + 9I)^{-2}$$

with $D(-\Delta)$ defined to impose zero Neumann boundary on the Laplacian. We extend T to be a Nemytskii operator acting on functions, defined through the map $T : \mathbb{R} \to \mathbb{R}_+$ defined as

$$T(x) = \begin{cases} 12, & x \geq 0 \\ 3, & x < 0 \end{cases}.$$

The random variable $a \sim \mu$ is a piecewise-constant function with random interfaces given by the underlying Gaussian random field. Such constructions are prototypical models for many physical systems such as permeability in sub-surface flows and (in a vector generalization) material microstructures in elasticity.

To create the dataset used for training, solutions to (5.36) are obtained using a second-order finite difference scheme on a uniform grid of size 421×421. All other resolutions are downsampled from this data set. We use $N = 1000$ training examples.

Burgers' Equation

We consider the one-dimensional viscous Burgers' equation

$$\begin{aligned} \frac{\partial}{\partial t} u(x,t) + \frac{1}{2}\frac{\partial}{\partial x}\big(u(x,t)\big)^2 &= \nu \frac{\partial^2}{\partial x^2} u(x,t), \qquad && x \in (0,2\pi), t \in (0,\infty) \\ u(x,0) &= u_0(x), && x \in (0,2\pi) \end{aligned} \tag{5.38}$$

with periodic boundary conditions and a fixed viscosity $\nu = 0.1$. Let

$$\Psi : L^2_{\text{per}}((0,2\pi);\mathbb{R}) \times \mathbb{R}_+ \to H^s_{\text{per}}((0,2\pi);\mathbb{R}),$$

for any $s > 0$, be the flow map associated to (5.38), in particular,

$$\Psi(u_0, t) = u(\cdot, t), \qquad t > 0.$$

We consider the solution operator defined by evaluating Ψ at a fixed time. Fix any $s \geq 0$. Then we may define $\mathcal{G}^\dagger : L^2_{\text{per}}((0,2\pi);\mathbb{R}) \to H^s_{\text{per}}((0,2\pi);\mathbb{R})$ by

$$\mathcal{G}^\dagger : u_0 \mapsto \Psi(u_0, 1). \tag{5.39}$$

We define the probability measure $\mu = N(0, C)$ where

$$C = 625\Big(-\frac{d^2}{dx^2} + 25I\Big)^{-2}$$

with domain of the Laplacian defined to impose periodic boundary conditions. We chose the initial condition for (5.38) by drawing $u_0 \sim \mu$, noting that μ charges a subset of $L^2_{\text{per}}((0,2\pi);\mathbb{R})$.

To create the dataset used for training, solutions to (5.38) are obtained using a pseudo-spectral split step method where the heat equation part is solved exactly in Fourier space and then the non-linear part is advanced using a forward Euler method with a very small time step. We use a uniform spatial grid with $2^{13} = 8192$ collocation points and subsample all other resolutions from this data set. We use $N = 1000$ training examples.

Navier-Stokes Equation

We consider the two-dimensional Navier-Stokes equation for a viscous, incompressible fluid

$$\begin{aligned} \partial_t u(x,t) + u(x,t)\cdot\nabla u(x,t) + \nabla p(x,t) &= \nu \Delta u(x,t) + f(x), \quad && x \in \mathbb{T}^2, t \in (0,\infty) \\ \nabla \cdot u(x,t) &= 0, && x \in \mathbb{T}^2, t \in [0,\infty) \\ u(x,0) &= u_0(x), && x \in \mathbb{T}^2 \end{aligned} \tag{5.40}$$

where $\mathbb{T}^2$ is the unit torus i.e. $[0,1]^2$ equipped with periodic boundary conditions, and $\nu \in \mathbb{R}_+$ is a fixed viscosity. Here $u : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{R}^2$ is the velocity field, $p : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{R}^2$ is the pressure field, and $f : \mathbb{T}^2 \to \mathbb{R}$ is a fixed forcing function.

Equivalently, we study the vorticity-streamfunction formulation of the equation

$$\begin{aligned} \partial_t w(x,t) + \nabla^\perp \psi \cdot \nabla w(x,t) &= \nu \Delta w(x,t) + g(x), && x \in \mathbb{T}^2, t \in (0,\infty), \\ -\Delta \psi &= \omega, && x \in \mathbb{T}^2, t \in (0,\infty), \\ w(x,0) &= w_0(x), && x \in \mathbb{T}^2, \end{aligned} \tag{5.41}$$

where w is the out-of-plane component of the vorticity field $\nabla \times u : \mathbb{T}^2 \times \mathbb{R}_+ \to \mathbb{R}^3$. Since, when viewed in three dimensions, $u = \big(u_1(x_1,x_2), u_2(x_1,x_2), 0\big)$, it follows that $\nabla \times u = (0,0,\omega)$. The stream function ψ is related to the velocity by $u = \nabla^\perp \psi$, enforcing the divergence-free condition. Similar considerations as for the curl of u apply to the curl of f, showing that $\nabla \times f = (0,0,g)$. We define the forcing term as

$$g(x_1,x_2) = 0.1(\sin(2\pi(x_1+x_2)) + \cos(2\pi(x_1+x_2))), \qquad \forall (x_1,x_2) \in \mathbb{T}^2.$$

The corresponding Reynolds number is estimated as $Re = \frac{\sqrt{0.1}}{\nu(2\pi)^{3/2}}$ (Chandler and Kerswell, 2013). Let $\Psi : L^2(\mathbb{T}^2;\mathbb{R}) \times \mathbb{R}_+ \to H^s(\mathbb{T}^2;\mathbb{R})$, for any $s > 0$, be the flow map associated to (5.41), in particular,

$$\Psi(w_0, t) = w(\cdot, t), \qquad t > 0.$$

Notice that this is well-defined for any $w_0 \in L^2(\mathbb{T};\mathbb{R})$.

We will define two notions of the solution operator. In the first, we will proceed as in the previous examples, in particular, $\mathcal{G}^\dagger : L^2(\mathbb{T}^2;\mathbb{R}) \to H^s(\mathbb{T}^2;\mathbb{R})$ is defined as

$$\mathcal{G}^\dagger : w_0 \mapsto \Psi(w_0, T) \tag{5.42}$$

for some fixed $T > 0$. In the second, we will map an initial part of the trajectory to a later part of the trajectory. In particular, we define $\mathcal{G}^\dagger : L^2(\mathbb{T}^2;\mathbb{R}) \times C\big((0,10]; H^s(\mathbb{T}^2;\mathbb{R})\big) \to C\big((10,T]; H^s(\mathbb{T}^2;\mathbb{R})\big)$ by

$$\mathcal{G}^\dagger : \big(w_0, \Psi(w_0,t)|_{t\in(0,10]}\big) \mapsto \Psi(w_0,t)|_{t\in(10,T]} \tag{5.43}$$

for some fixed $T > 10$. We define the probability measure $\mu = N(0, C)$ where

$$C = 7^{3/2}(-\Delta + 49I)^{-2.5}$$

with periodic boundary conditions on the Laplacian. We model the initial vorticity $w_0 \sim \mu$ to (5.41) as μ charges a subset of $L^2(\mathbb{T}^2;\mathbb{R})$. Its pushforward onto $\Psi(w_0,t)|_{t\in(0,10]}$ is required to define the measure on input space in the second case defined by (5.43).

To create the dataset used for training, solutions to (5.41) are obtained using a pseudo-spectral split step method where the viscous terms are advanced using a Crank–Nicolson update and the nonlinear and forcing terms are advanced using Heun's method. Dealiasing is used with the $2/3$ rule. For further details on this approach see (Chandler and Kerswell, 2013). Data is obtained on a uniform 256×256 grid and all other resolutions are subsampled from this data set. We experiment with different viscosities ν, final times T, and amounts of training data N.

Bayesian Inverse Problem

As an application of operator learning, we consider the inverse problem of recovering the initial vorticity in the Navier-Stokes equation (5.41) from partial, noisy observations of the vorticity at a later time. Consider the first solution operator defined in subsection 5.6, in particular, $\mathcal{G}^\dagger : L^2(\mathbb{T}^2;\mathbb{R}) \to H^s(\mathbb{T}^2;\mathbb{R})$ defined as

$$\mathcal{G}^\dagger : w_0 \mapsto \Psi(w_0, 50)$$

where Ψ is the flow map associated to (5.41). We then consider the inverse problem

$$y = O\big(\mathcal{G}^\dagger(w_0)\big) + \eta \tag{5.44}$$

of recovering $w_0 \in L^2(\mathbb{T}^2;\mathbb{R})$ where $O : H^s(\mathbb{T}^2;\mathbb{R}) \to \mathbb{R}^{49}$ is the evaluation operator on a uniform 7×7 interior grid, and $\eta \sim N(0,\Gamma)$ is observational noise with covariance $\Gamma = (1/\gamma^2)I$ and $\gamma = 0.1$. We view (5.44) as the Bayesian inverse problem mapping prior measure μ on w_0 to posterior measure π^y on w_0/y. In particular, π^y has density with respect to μ, given by the Randon-Nikodym derivative

$$\frac{\mathrm{d}\pi^y}{\mathrm{d}\mu}(w_0) \propto \exp\Big(-\frac{1}{2}\|y - O\big(\mathcal{G}^\dagger(w_0)\big)\|_\Gamma^2\Big)$$

where $\|\cdot\|_\Gamma = \|\Gamma^{-1/2}\cdot\|$ and $\|\cdot\|$ is the Euclidean norm in $\mathbb{R}^{49}$. For further details on Bayesian inversion for functions see (Simon L Cotter et al., 2009; A. M. Stuart, 2010), and see (S. L. Cotter et al., 2013) for MCMC methods adapted to the function-space setting.

We solve (5.44) by computing the posterior mean $\mathbb{E}_{w_0\sim\pi^y}[w_0]$ using the pre-conditioned Crank–Nicolson (pCN) MCMC method described in S. L. Cotter et al., 2013 for this task. We employ pCN in two cases: (i) using $\mathcal{G}^\dagger$ evaluated with the pseudo-spectral method described in section 5.6, and (ii) using $\mathcal{G}_\theta$, the neural operator approximating $\mathcal{G}^\dagger$. After a 5,000 sample burn-in period, we generate 25,000 samples from the posterior using both approaches and use them to compute the posterior mean.

Spectra

Because of the constant-in-time forcing term the energy reaches a non-zero equilibrium in time which is statistically reproducible for different initial conditions. To compare the complexity of the solution to the Navier-Stokes problem outlined in subsection 5.6 we show, in Figure 56, the Fourier spectrum of the solution data at time $t = 50$ for three different choices of the viscosity ν. The figure demonstrates that, for a wide range of wavenumbers k, which grows as ν decreases, the rate of decay of the spectrum is $-5/3$, matching what is expected in the turbulent regime (Kraichnan, 1967). This is a statistically stationary property of the equation, sustained for all positive times.

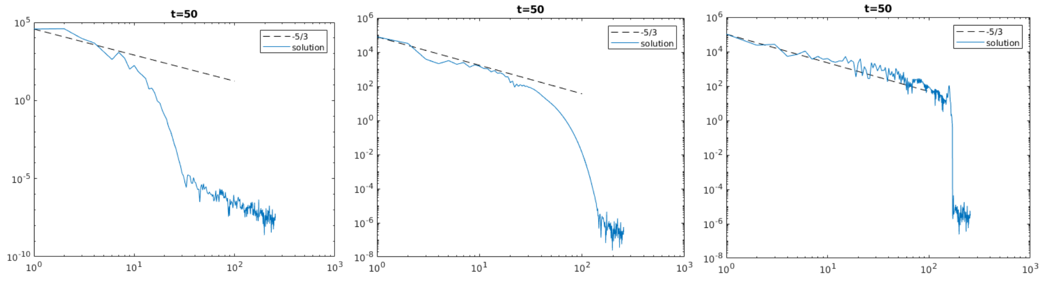


Figure 56: The spectral decay of the Navier-stokes equation data. The y-axis is represents the value of each mode; the x-axis is the wavenumber $|k| = k_1 + k_2$. From left to right, the solutions have viscosity $\nu = 10^{-3},\ 10^{-4},\ 10^{-5}$ respectively.

Choice of Loss Criteria

In general, the model has the best performance when trained and tested using the same loss criteria. If one trains the model using one norm and tests with another norm, the model may overfit in the training norm. Furthermore, the choice of loss function plays a key role. In this work, we use the relative L_2 error to measure the performance in all our problems. Both the L_2 error and its square, the mean squared error (MSE), are common choices of the testing criteria in the numerical analysis and machine learning literature. We observed that using the relative error to train the model has a good normalization and regularization effect that prevents overfitting.

In practice, training with the relative L_2 loss results in around half the testing error rate compared to training with the MSE loss.

5.7 Numerical Results

In this section, we compare the proposed neural operator with other supervised learning approaches, using the four test problems outlined in Section 5.6. In Subsection 5.7 we study the Poisson equation, and learning a Greens function; Subsection 5.7 considers the coefficient to solution map for steady Darcy flow, and the initial condition to solution at positive time map for Burgers equation. In Subsection 5.6 we study the Navier-Stokes equation.

We compare with a variety of architectures found by discretizing the data and applying finite-dimensional approaches, as well as with other operator-based approximation methods. We do not compare against traditional solvers (FEM/FDM/Spectral), although our methods, once trained, enable evaluation of the input to output map orders of magnitude more quickly than by use of such traditional solvers on complex problems. We demonstrate the benefits of this speed-up in a prototypical application, Bayesian inversion, in Subsubection 5.7.

All the computations are carried on a single Nvidia V100 GPU with 16GB memory. The code is available at `https://github.com/zongyi-li/graph-pde` and `https://github.com/zongyi-li/fourier_neural_operator`.

Setup of the Four Methods: We construct the neural operator by stacking four integral operator layers as specified in (5.5) with the ReLU activation. No batch normalization is needed. Unless otherwise specified, we use $N = 1000$ training instances and 200 testing instances. We use the Adam optimizer to train for 500 epochs with an initial learning rate of 0.001 that is halved every 100 epochs. We set the channel dimensions $d_{v_0} = \cdots = d_{v_3} = 64$ for all one-dimensional problems and $d_{v_0} = \cdots = d_{v_3} = 32$ for all two-dimensional problems. The kernel networks $\kappa^{(0)}, \ldots, \kappa^{(3)}$ are standard feed-forward neural networks with three layers and widths of 256 units. We use the following abbreviations to denote the methods introduced in Section 5.5.

- **GNO:** The method introduced in subsection 5.5, truncating the integral to a ball with radius $r = 0.25$ and using the Nyström approximation with $J' = 300$ sub-sampled nodes.

- **LNO:** The low-rank method introduced in subsection 5.5 with rank $r = 4$.
- **MGNO:** The multipole method introduced in subsection 5.5. On the Darcy flow problem, we use the random construction with three graph levels, each sampling $J_1 = 400, J_2 = 100, J_3 = 25$ nodes nodes respectively. On the Burgers' equation problem, we use the orthogonal construction without sampling.
- **FNO:** The Fourier method introduced in subsection 5.5. We set $k_{\text{max},j} = 16$ for all one-dimensional problems and $k_{\text{max},j} = 12$ for all two-dimensional problems.

Remark on the Resolution. Traditional PDE solvers such as FEM and FDM approximate a single function and therefore their error to the continuum decreases as the resolution is increased. The figures we show here exhibit something different: the error is independent of resolution, once enough resolution is used, but is not zero. This reflects the fact that there is a residual approximation error, in the infinite dimensional limit, from the use of a finite-parametrized neural operator, trained on a finite amount of data. Invariance of the error with respect to (sufficiently fine) resolution is a desirable property that demonstrates that an intrinsic approximation of the operator has been learned, independent of any specific discretization; see Figure 58. Furthermore, resolution-invariant operators can do zero-shot super-resolution, as shown in Subsubection 5.7.

Poisson Equation

Recall the Poisson equation (5.35) introduced in subsection 5.6. We use a zero hidden layer neural operator construction without lifting the input dimension. In particular, we simply learn a kernel $\kappa_\theta : \mathbb{R}^2 \to \mathbb{R}$ parameterized as a standard feed-forward neural network with parameters θ. Using only $N = 1000$ training examples, we obtain a relative test error of 10^{-7}. The neural operator gives an almost perfect approximation to the true solution operator in the topology of (5.3).

To examine the quality of the approximation in the much stronger uniform topology, we check whether the kernel κ_θ approximates the Green's function for this problem. To see why this is enough, let $K \subset L^2([0,1];\mathbb{R})$ be a bounded set, i.e.

$$\|f\|_{L^2([0,1];\mathbb{R})} \leq M, \qquad \forall f \in K$$

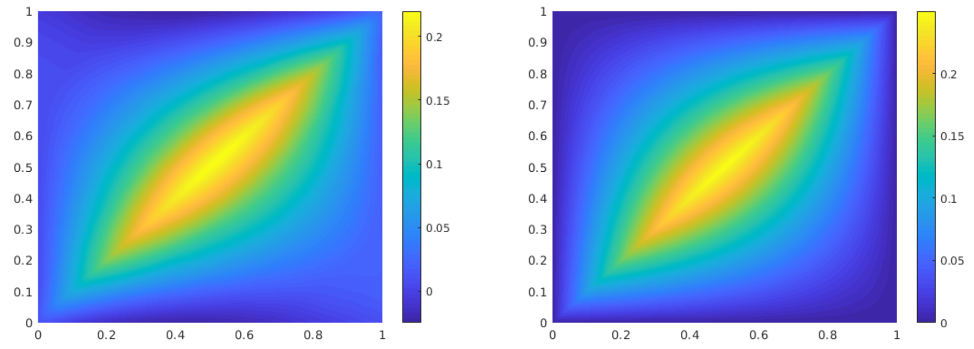


Figure 57: Kernel for one-dimensional Green's function, with the Nystrom approximation method. Left: learned kernel function; Right: the analytic Green's funciton. This is a proof of concept of the graph kernel network on 1 dimensional Poisson equation and the comparison of learned and truth kernel.

and suppose that

$$\sup_{(x,y)\in[0,1]^2} |\kappa_\theta(x,y) - G(x,y)| < \frac{\epsilon}{M}$$

for some $\epsilon > 0$. Then it is easy to see that

$$\sup_{f\in K} \|\mathcal{G}^\dagger(f) - \mathcal{G}_\theta(f)\|_{L^2([0,1];\mathbb{R})} < \epsilon,$$

in particular, we obtain an approximation in the topology of uniform convergence over bounded sets, while having trained only in the topology of the Bochner norm (5.3). Figure 57 shows the results from which we can see that κ_θ does indeed approximate the Green's function well. This result implies that by constructing a suitable architecture, we can generalize to the entire space and data that is well outside the support of the training set.

Darcy and Burgers Equations

In the following section, we compare four methods presented in this paper, with different operator approximation benchmarks; we study the Darcy flow problem introduced in Subsection 5.6 and the Burgers' equation problem introduced in Subsection 5.6. The solution operators of interest are defined by (5.37) and (5.39). We use the following abbreviations for the methods against which we benchmark.

- **NN** is a standard point-wise feedforward neural network. It is mesh-free, but performs badly due to lack of neighbor information. We use standard fully connected neural networks with 8 layers and width 1000.

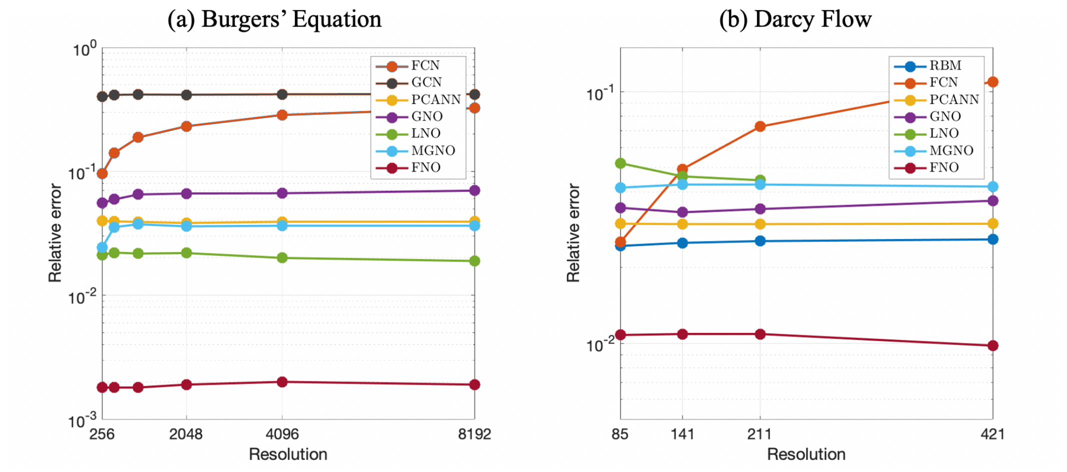


Figure 58: (a) Benchmarks on Burgers equation; (b) benchmarks on Darcy Flow for different resolutions. Train and test on the same resolution.

- **FCN** is the state of the art neural network method based on Fully Convolution Network (Zhu and Zabaras, 2018). It has a dominating performance for small grids $s = 61$. But fully convolution networks are mesh-dependent and therefore their error grows when moving to a larger grid.

- **PCA+NN** is an instantiation of the methodology proposed in Bhattacharya et al., 2020: using PCA as an autoencoder on both the input and output spaces and interpolating the latent spaces with a standard fully connected neural network with width 200. The method provably obtains mesh-independent error and can learn purely from data, however the solution can only be evaluated on the same mesh as the training data.

- **RBM** is the classical Reduced Basis Method (using a PCA basis), which is widely used in applications and provably obtains mesh-independent error (R. A. DeVore, 2014). This method has good performance, but the solutions can only be evaluated on the same mesh as the training data and one needs knowledge of the PDE to employ it.

- **DeepONet** is the Deep Operator network (L. Lu, Jin, and George Em Karniadakis, 2019) that comes equipped with an approximation theory (Lanthaler, Mishra, and George Em Karniadakis, 2021). We use the unstacked version with width 200 which is precisely defined in the original work (L. Lu, Jin, and George Em Karniadakis, 2019). We use standard fully connected neural networks with 8 layers and width 200.

Darcy Flow

The results of the experiments on Darcy flow are shown in Figure 58 and Table 51. All the methods, except for FCN, achieve invariance of the error with respect to the resolution s. In the experiment, we tune each model across of range of different widths and depth to obtain the choices used here; for DeepONet for example this leads to 8 layers and width 200 as reported above.

Within our hyperparameter search, the Fourier neural operator (FNO) obtains the lowest relative error. The Fourier based method likely sees this advantage because the output functions are smooth in these test problems. We also note that is also possible to obtain better results on each model using modified architectures and problem specific feature engineering. For example for DeepONet, using CNN on the branch net and PCA on the trunk net (the latter being similar to the method used in Bhattacharya et al., 2020) can achieve 0.0232 relative L_2 error, as shown in L. Lu, Meng, et al., 2021, about half the size of the error we obtain here, but for a very coarse grid with $s = 29$. In the experiments the different approximation architectures are such their training cost are similar across all the methods considered, for given s. Noting this, and for example comparing the graph-based neural operator methods such as GNO and MGNO that use Nyström sampling in physical space with FNO, we see that FNO is more accurate.

Networks	$s = 85$	$s = 141$	$s = 211$	$s = 421$
NN	0.1716	0.1716	0.1716	0.1716
FCN	0.0253	0.0493	0.0727	0.1097
PCANN	0.0299	0.0298	0.0298	0.0299
RBM	0.0244	0.0251	0.0255	0.0259
DeepONet	0.0476	0.0479	0.0462	0.0487
GNO	0.0346	0.0332	0.0342	0.0369
LNO	0.0520	0.0461	0.0445	$-$
MGNO	0.0416	0.0428	0.0428	0.0420
FNO	**0.0108**	**0.0109**	**0.0109**	**0.0098**

Table 51: Relative error on 2-d Darcy Flow for different resolutions s.

Burgers' Equation

The results of the experiments on Burgers' equation are shown in Figure 58 and Table 52. As for the Burgers' problem, our instantiation of the Fourier neural operator obtains nearly one order of magnitude lower relative error compared to any benchmarks. The Fourier neural operator has standard deviation 0.0010 and mean training error 0.0012. If one replaces the ReLU activation by GeLU, the test error of the FNO is further reduced from 0.0018 to 0.0007. We again observe the invariance of the error with respect to the resolution. It is possible to improve the performance on each model using modified architectures and problem specific feature engineering. Similarly, the PCA-enhanced DeepONet with a proper scaling can achieve 0.0194 relative L_2 error, as shown in L. Lu, Meng, et al., 2021, on a grid of resolution $s = 128$.

Networks	$s = 256$	$s = 512$	$s = 1024$	$s = 2048$	$s = 4096$	$s = 8192$
NN	0.4714	0.4561	0.4803	0.4645	0.4779	0.4452
GCN	0.3999	0.4138	0.4176	0.4157	0.4191	0.4198
FCN	0.0958	0.1407	0.1877	0.2313	0.2855	0.3238
PCANN	0.0398	0.0395	0.0391	0.0383	0.0392	0.0393
DeepONet	0.0569	0.0617	0.0685	0.0702	0.0833	0.0857
GNO	0.0555	0.0594	0.0651	0.0663	0.0666	0.0699
LNO	0.0212	0.0221	0.0217	0.0219	0.0200	0.0189
MGNO	0.0243	0.0355	0.0374	0.0360	0.0364	0.0364
FNO	**0.0018**	**0.0018**	**0.0018**	**0.0019**	**0.0020**	**0.0019**

Table 52: Relative errors on 1-d Burgers' equation for different resolutions s.

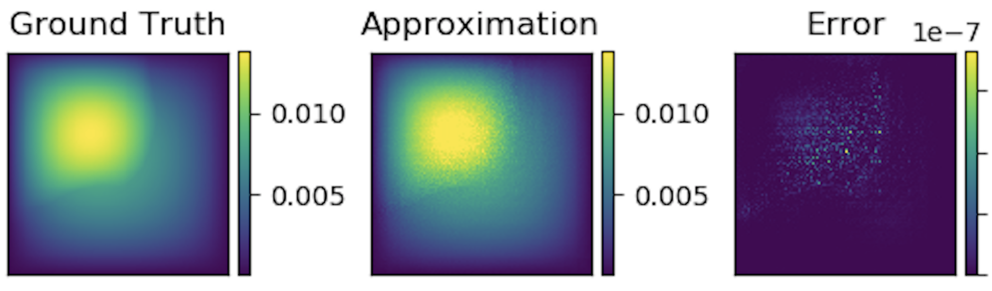


Figure 59: Darcy, trained on 16×16, tested on 241×241. Graph kernel network for the solution of (5.6). It can be trained on a small resolution and will generalize to a large one. The Error is point-wise absolute squared error.

Zero-shot super-resolution.

The neural operator is mesh-invariant, so it can be trained on a lower resolution and evaluated at a higher resolution, without seeing any higher resolution data (zero-shot super-resolution). Figure 59 shows an example of the Darcy Equation where we train the GNO model on 16×16 resolution data in the setting above and transfer to 256×256 resolution, demonstrating super-resolution in space.

Navier-Stokes Equation

In the following section, we compare our four methods with different benchmarks on the Navier-Stokes equation introduced in subsection 5.6. The operator of interest is given by (5.43). We use the following abbreviations for the methods against which we benchmark.

- **ResNet:** 18 layers of 2-d convolution with residual connections K. He et al., 2016.
- **U-Net:** A popular choice for image-to-image regression tasks consisting of four blocks with 2-d convolutions and deconvolutions Ronneberger, Fischer, and Brox, 2015.
- **TF-Net:** A network designed for learning turbulent flows based on a combination of spatial and temporal convolutions R. Wang et al., 2020.
- **FNO-2d:** 2-d Fourier neural operator with an auto-regressive structure in time. We use the Fourier neural operator to model the local evolution from the previous 10 time steps to the next one time step, and iteratively apply the model to get the long-term trajectory. We set and $k_{\max,j} = 12, d_v = 32$.
- **FNO-3d:** 3-d Fourier neural operator that directly convolves in space-time. We use the Fourier neural operator to model the global evolution from the initial 10 time steps directly to the long-term trajectory. We set $k_{\max,j} = 12, d_v = 20$.

As shown in Table 53, the FNO-3D has the best performance when there is sufficient data ($\nu = 10^{-3}, N = 1000$ and $\nu = 10^{-4}, N = 10000$). For the configurations where the amount of data is insufficient ($\nu = 10^{-4}, N = 1000$ and $\nu = 10^{-5}, N = 1000$), all methods have $> 15\%$ error with FNO-2D achieving the lowest among our hyperparameter search. Note that we only present results for spatial resolution 64×64 since all benchmarks we compare against are designed for this resolution. Increasing the spatial resolution degrades their performance while FNO achieves the same errors.

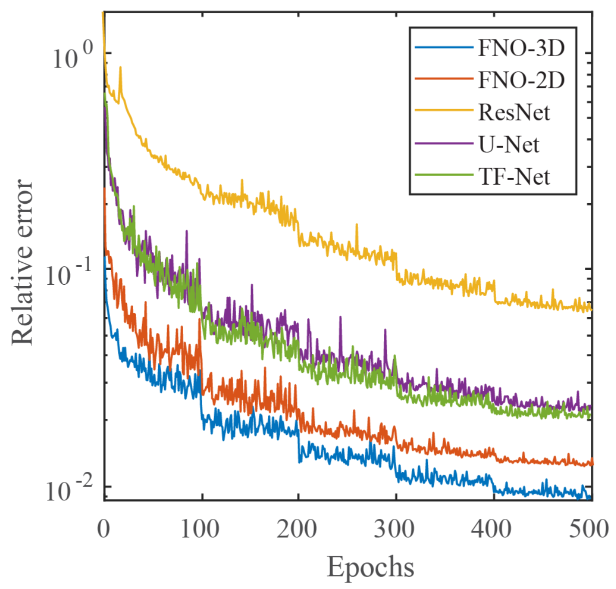


Figure 510: The learning curves on Navier-Stokes $\nu = 1\mathrm{e}{-3}$ with different benchmarks. Train and test on the same resolution.

Config	Parameters	Time per epoch	$\nu = 10^{-3}$ $T = 50$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 1000$	$\nu = 10^{-4}$ $T = 30$ $N = 10000$	$\nu = 10^{-5}$ $T = 20$ $N = 1000$
FNO-3D	$6,558,537$	$38.99s$	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	$414,517$	$127.80s$	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	$24,950,491$	$48.67s$	0.0245	0.2051	0.1190	0.1982
TF-Net	$7,451,724$	$47.21s$	0.0225	0.2253	0.1168	0.2268
ResNet	$266,641$	$78.47s$	0.0701	0.2871	0.2311	0.2753

Table 53: Benchmarks on Navier Stokes (fixing resolution 64×64 for both training and testing).

Auto-regressive (2D) and Temporal Convolution (3D). We investigate two standard formulation to model the time evolution: the auto-regressive model (2D) and the temporal convolution model (3D). **Auto-regressive models**: FNO-2D, U-Net, TF-Net, and ResNet all do 2D-convolution in the spatial domain and recurrently propagate in the time domain (2D+RNN). The operator maps the solution at previous time steps to the next time step (2D functions to 2D functions). **Temporal convolution models**: on the other hand, FNO-3D performs convolution in space-time – it approximation the integral in time by a convolution. FNO-3D maps the initial time interval directly to the full trajectory (3D functions to 3D functions). The 2D+RNN structure can propagate the solution to any arbitrary time T in increments of a fixed

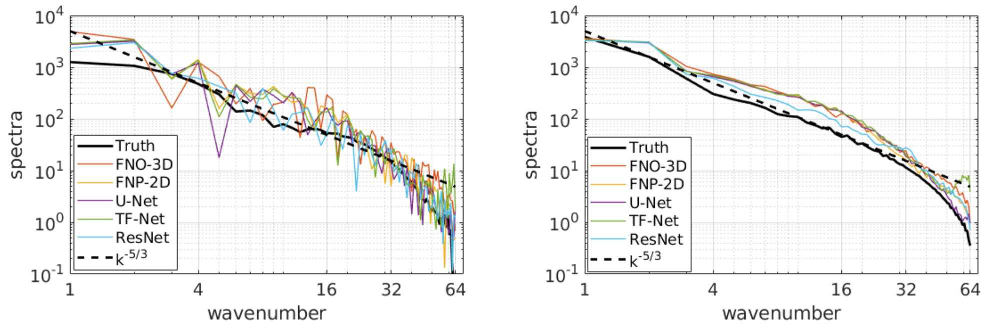


Figure 511: The spectral decay of the predictions of different models on the Navier-Stokes equation. The y-axis is the spectrum; the x-axis is the wavenumber. Left is the spectrum of one trajectory; right is the average of 40 trajectories.

interval length Δt, while the Conv3D structure is fixed to the interval $[0, T]$ but can transfer the solution to an arbitrary time-discretization. We find the 2D method work better for short time sequences while the 3D method more expressive and easier to train on longer sequences.

Networks	$s = 64$	$s = 128$	$s = 256$
FNO-3D	0.0098	0.0101	0.0106
FNO-2D	0.0129	0.0128	0.0126
U-Net	0.0253	0.0289	0.0344
TF-Net	0.0277	0.0278	0.0301

Table 54: Resolution study on Navier-stokes equation ($\nu = 10^{-3}$, $N = 200$, $T = 20$.)

Zero-shot super-resolution.

The neural operator is mesh-invariant, so it can be trained on a lower resolution and evaluated at a higher resolution, without seeing any higher resolution data (zero-shot super-resolution). Figure 51 shows an example where we train the FNO-3D model on $64 \times 64 \times 20$ resolution data in the setting above with ($\nu = 10^{-4}$, $N = 10000$) and transfer to $256 \times 256 \times 80$ resolution, demonstrating super-resolution in space-time. The Fourier neural operator is the only model among the benchmarks (FNO-2D, U-Net, TF-Net, and ResNet) that can do zero-shot super-resolution; the method works well not only on the spatial but also on the temporal domain.

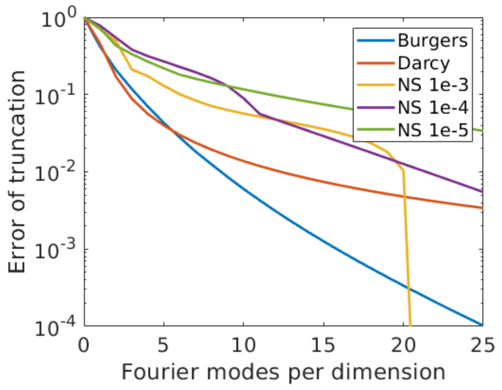


Figure 512: The error of truncation in one single Fourier layer without applying the linear transform R. The y-axis is the normalized truncation error; the x-axis is the truncation mode k_{max}.

Spectral analysis

Figure 511 shows that all the methods are able to capture the spectral decay of the Navier-Stokes equation. Notice that, while the Fourier method truncates the higher frequency modes during the convolution, FNO can still recover the higher frequency components in the final prediction. Due to the way we parameterize R_ϕ, the function output by (5.28) has at most $k_{\max,j}$ Fourier modes per channel. This, however, does not mean that the Fourier neural operator can only approximate functions up to $k_{\max,j}$ modes. Indeed, the activation functions which occurs between integral operators and the final decoder network Q recover the high frequency modes. As an example, consider a solution to the Navier-Stokes equation with viscosity $\nu = 10^{-3}$. Truncating this function at 20 Fourier modes yields an error around 2% as shown in Figure 512, while the Fourier neural operator learns the parametric dependence and produces approximations to an error of $\leq 1\%$ with only $k_{\max,j} = 12$ parameterized modes.

Non-periodic boundary condition.

Traditional Fourier methods work only with periodic boundary conditions. However, the Fourier neural operator does not have this limitation. This is due to the linear transform W (the bias term) which keeps the track of non-periodic boundary. As an example, the Darcy Flow and the time domain of Navier-Stokes have non-periodic boundary conditions, and the Fourier neural operator still learns the solution operator with excellent accuracy.

Bayesian Inverse Problem

As discussed in Section 5.6, we use the pCN method of S. L. Cotter et al., 2013 to draw samples from the posterior distribution of initial vorticities in the Navier-Stokes equation given sparse, noisy observations at time $T = 50$. We compare the Fourier neural operator acting as a surrogate model with the traditional solvers used to generate our train-test data (both run on GPU). We generate 25,000 samples from the posterior (with a 5,000 sample burn-in period), requiring 30,000 evaluations of the forward operator.

As shown in Figure 513, FNO and the traditional solver recover almost the same posterior mean which, when pushed forward, recovers well the later-time solution of the Navier-Stokes equation. In sharp contrast, FNO takes $0.005s$ to evaluate a single instance while the traditional solver, after being optimized to use the largest possible internal time-step which does not lead to blow-up, takes $2.2s$. This amounts to 2.5 minutes for the MCMC using FNO and over 18 hours for the traditional solver. Even if we account for data generation and training time (offline steps) which take 12 hours, using FNO is still faster. Once trained, FNO can be used to quickly perform multiple MCMC runs for different initial conditions and observations, while the traditional solver will take 18 hours for every instance. Furthermore, since FNO is differentiable, it can easily be applied to PDE-constrained optimization problems in which adjoint calculations are used as part of the solution procedure.

Discussion and Comparison of the Four methods

In this section we will compare the four methods in term of expressiveness, complexity, refinabilibity, and ingenuity.

Ingenuity

First we will discuss ingenuity, in other words, the design of the frameworks. The first method, GNO, relies on the Nyström approximation of the kernel, or the Monte Carlo approximation of the integration. It is the most simple and straightforward method. The second method, LNO, relies on the low-rank decomposition of the kernel operator. It is efficient when the kernel has a near low-rank structure. The third method, MGNO, is the combination of the first two. It has a hierarchical, multi-resolution decomposition of the kernel. The last one, FNO, is different from the first three; it restricts the integral kernel to induce a convolution.

GNO and MGNO are implemented using graph neural networks, which helps to

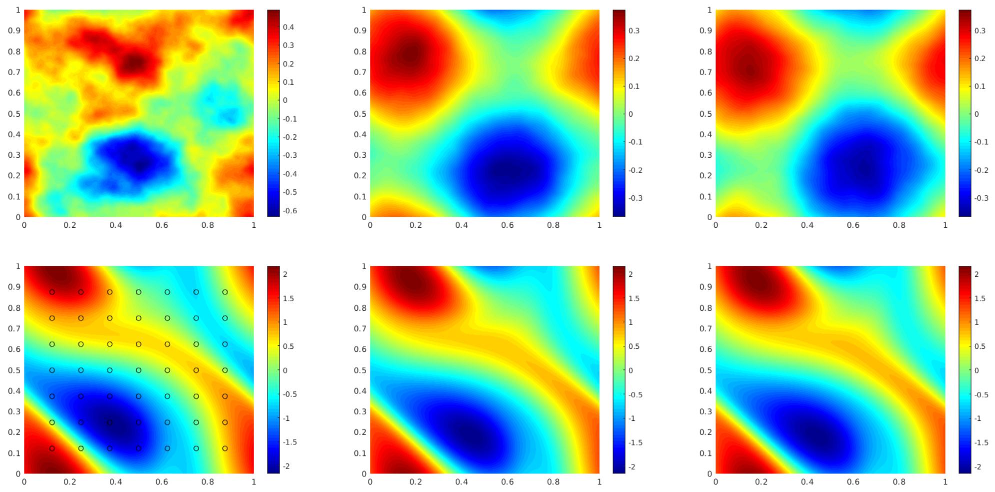


Figure 513: Results of the Bayesian inverse problem for the Navier-Stokes equation. The top left panel shows the true initial vorticity while the bottom left panel shows the true observed vorticity at $T = 50$ with black dots indicating the locations of the observation points placed on a 7×7 grid. The top middle panel shows the posterior mean of the initial vorticity given the noisy observations estimated with MCMC using the traditional solver, while the top right panel shows the same thing but using FNO as a surrogate model. The bottom middle and right panels show the vorticity at $T = 50$ when the respective approximate posterior means are used as initial conditions.

define sampling and integration. The graph network library also allows sparse and distributed message passing. The LNO and FNO don't have sampling. They are faster without using the graph library.

	scheme	graph-based	kernel network
GNO	Nyström approximation	Yes	Yes
LNO	Low-rank approximation	No	Yes
MGNO	Multi-level graphs on GNO	Yes	Yes
FNO	Convolution theorem; Fourier features	No	No

Table 55: Ingenuity.

Expressiveness

We measure the expressiveness by the training and testing error of the method. The full $O(J^2)$ integration always has the best results, but it is usually too expensive. As shown in the experiments 5.7 and 5.7, GNO usually has good accuracy, but its performance suffers from sampling. LNO works the best on the 1d problem (Burgers equation). It has difficulty on the 2d problem because it doesn't employ sampling to speed-up evaluation. MGNO has the multi-level structure, which gives it the benefit of the first two. Finally, FNO has overall the best performance. It is also the only method that can capture the challenging Navier-Stokes equation.

Complexity

The complexity of the four methods are listed in Table 56. GNO and MGNO have sampling. Their complexity depends on the number of nodes sampled J'. When using the full nodes. They are still quadratic. LNO has the lowest complexity $O(J)$. FNO, when using the fast Fourier transform, has complexity $O(J \log J)$.

In practice. FNO is faster then the other three methods because it doesn't have the kernel network κ. MGNO is relatively slower because of its multi-level graph structure.

	Complexity	Time per epochs in training
GNO	$O(J'^2 r^2)$	$4s$
LNO	$O(J)$	$20s$
MGNO	$\sum_l O(J_l^2 r_l^2) \sim O(J)$	$8s$
FNO	$(J \log J)$	$4s$

Table 56: Complexity (roundup to second on a single Nvidia V100 GPU).

Refinability

Refineability measures the number of parameters used in the framework. Table 57 lists the accuracy of the relative error on Darcy Flow with respect to different number of parameters. Because GNO, LNO, and MGNO have the kernel networks, the slope of their error rates are flat: they can work with a very small number of parameters. On the other hand, FNO does not have the sub-network. It needs at a larger magnitude of parameters to obtain an acceptable error rate.

Robustness

We conclude with experiments investigating the robustness of Fourier neural operator to noise. We study: a) training on clean (noiseless) data and testing with clean and noisy data, and b) training on clean (noiseless) data and testing with clean and noisy data. When creating noisy data we map a to noisy a' as follows: at every grid-point

Number of parameters	10^3	10^4	10^5	10^6
GNO	0.075	0.065	0.060	0.035
LNO	0.080	0.070	0.060	0.040
MGNO	0.070	0.050	0.040	0.030
FNO	0.200	0.035	0.020	0.015

Table 57: The relative error on Darcy Flow with respect to different number of parameters. The errors above are approximated value roundup to 0.05. They are the lowest test error achieved by the model, given the model's number of parameters $|\theta|$ is bounded by $10^3, 10^4, 10^5, 10^6$ respectively.

x we set

$$a(x)' = a(x) + 0.1 \cdot \|a\|_\infty \xi,$$

where $\xi \sim \mathcal{N}(0, 1)$ is drawn i.i.d. at every grid point; this is similar to the setting adopted in L. Lu, Meng, et al., 2021. We also study the 1d advection equation as an additional test case, following the setting in L. Lu, Meng, et al., 2021 in which the input data is a random square wave, defined by an $\mathbb{R}^3$-valued random variable.

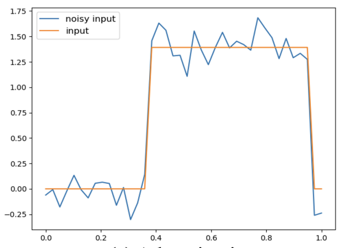
Problems	Training error	Test (clean)	Test (noisy)
Burgers	0.002	0.002	0.018
Advection	0.002	0.002	0.094
Darcy	0.006	0.011	0.012
Navier-Stokes	0.024	0.024	0.039
Burgers (train with noise)	0.011	0.004	0.011
Advection (train with noise)	0.020	0.010	0.019
Darcy (train with noise)	0.007	0.012	0.012
Navier-Stokes (train with noise)	0.026	0.026	0.025

Table 58: Robustness to noise of the FNO method.

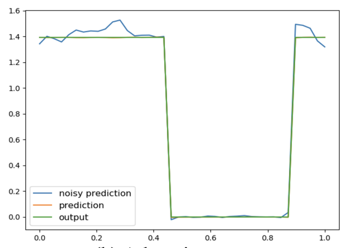
As shown in the top half of Table 58 and Figure 514, we observe the Fourier neural operator is robust with respect to the (test) noise level on all four problems. In particular, on the advection problem, it has about 10% error with 10% noise. The Darcy and Navier-Stokes operators are smoothing, and the Fourier neural operator obtains lower than 10% error in all scenarios. However the FNO is less robust on the advection equation, which is not smoothing, and on Burgers equation which, whilst smoothing also forms steep fronts.

A straightforward approach to enhance the robustness is to train the model with noise. As shown in the bottom half of Table 58, the Fourier neural operator has no gap between the clean data and noisy data when training with noise. However, noise in training may degrade the performance on the clean data, as a trade-off. In general, augmenting the training data with noise leads to robustness. For example, in the auto-regressive modeling of dynamical systems, training the model with noise will reduce error accumulation in time, and thereby help the model to predict over longer time-horizons (Pfaff et al., 2020). We also observed that other regularization techniques such as early-stopping and weight decay improve robustness. Using a higher spatial resolution also helps.

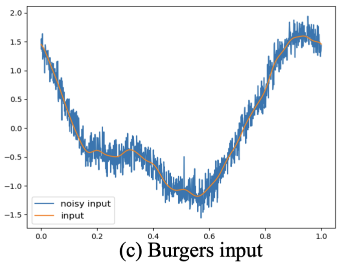
The advection problem is a hard problem for the FNO since it has discontinuities; similar issues arise when using spectral methods for conservation laws. One can modify the architecture to address such discontinuities accordingly. For example, Wen et al., 2021 enhance the FNO by composing a CNN or UNet branch with the Fourier layer; the resulting composite model outperforms the basic FNO on multiphase flow with high contrast and sharp shocks. However the CNN and UNet take the method out of the realm of discretization-invariant methods; further work is required to design discretization-invariant image-processing tools, such as the identification of discontinuities.



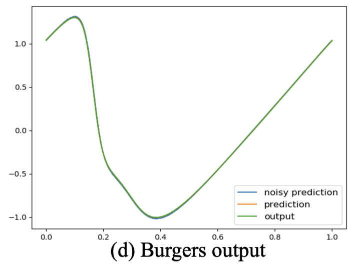
(a) Advection input



(b) Advection output



(c) Burgers input



(d) Burgers output

Figure 514: Robustness on Advection and Burgers equations. (a) The input of Advection equation ($s = 40$). The orange curve is the clean input; the blue curve is the noisy input. (b) The output of Advection equation. The green curve is the ground truth output; the orange curve is the prediction of FNO with clean input (overlapping with the ground truth); the blue curve is the prediction on the noisy input. Figure (c) and (d) are for Burgers' equation ($s = 1000$) correspondingly.

5.8 Conclusions

We have introduced the concept of Neural Operator, the goal being to construct a neural network architecture adapted to the problem of mapping elements of one function space into elements of another function space. The network is comprised of four steps which, in turn, (i) extract features from the input functions, (ii) iterate a recurrent neural network on feature space, defined through composition of a sigmoid function and a nonlocal operator, and (iii) a final mapping from feature space into the output function.

We have studied four nonlocal operators in step (iii), one based on graph kernel networks, one based on the low-rank decomposition, one based on the multi-level graph structure, and the last one based on convolution in Fourier space. The designed network architectures are constructed to be mesh-free and our numerical experiments demonstrate that they have the desired property of being able to train and generalize on different meshes. This is because the networks learn the mapping between infinite-dimensional function spaces, which can then be shared with approximations at different levels of discretization. A further advantage of the integral operator approach is that data may be incorporated on unstructured grids, using the Nyström approximation; these methods, however, are quadratic in the number of discretization points; we describe variants on this methodology, using low rank and multiscale ideas, to reduce this complexity. On the other hand the Fourier approach leads directly to fast methods, linear-log linear in the number of discretization points, provided structured grids are used. We demonstrate that our methods can achieve competitive performance with other mesh-free approaches developed in the numerical analysis community. Specifically, the Fourier neural operator achieves the best numerical performance among our experiments, potentially due to the smoothness of the solution function and the underlying uniform grids. The methods developed in the numerical analysis community are less flexible than the approach we introduce here, relying heavily on the structure of an underlying PDE mapping input to output; our method is entirely data-driven.

Future Directions

We foresee three main directions in which this work will develop: firstly as a method to speed-up scientific computing tasks which involve repeated evaluation of a mapping between spaces of functions, following the example of the Bayesian inverse problem 5.7, or when the underlying model is unknown as in computer vision or robotics; and secondly the development of more advanced methodologies beyond the four approximation schemes presented in Section 5.5 that are more efficient or better in specific situations; thirdly, the development of an underpinning theory which captures the expressive power, and approximation error properties, of the proposed neural network, following Section 5.4, and quantifies the computational complexity required to achieve given error.

New Applications

The proposed neural operator is a blackbox surrogate model for function-to-function mappings. It naturally fits into solving PDEs for physics and engineering problems. In the paper we mainly studied three partial differential equations: Darcy Flow, Burgers' equation, and Navier-Stokes equation, which cover a board range of scenarios. Due to its blackbox structure, the neural operator is easily applied on other problems. We foresee applications on more challenging turbulent flows, such as those arising in subgrid models with in climate GCMs, high contrast media in geological models generalizing the Darcy model, and general physics simulation for games and visual effects. The operator setting leads to an efficient and accurate representation, and the resolution-invariant properties make it possible to train on small resolution datasets and evaluate on arbitrary resolutions.

The operator learning setting is not restricted to scientific computing. For example, in computer vision, images can naturally be viewed as real-valued functions on 2D domains and videos simply add a temporal structure. Our approach is therefore a natural choice for problems in computer vision where invariance to discretization is crucial. We leave this as an interesting future direction.

New Methodologies

Despite their excellent performance, there is still room for improvement upon the current methodologies. For example, the full $O(J^2)$ integration method still outperforms the FNO by about 40%, albeit at greater cost. It is of potential interest to develop more advanced integration techniques or approximation schemes that follows the neural operator framework. For example, one can use adaptive graph or probability estimation in the Nyström approximation. It is also possible to use a basis other than the Fourier basis such as the PCA basis and the Chebyshev basis.

Another direction for new methodologies is to combine the neural operator in other settings. The current problem is set as a supervised learning problem. Instead, one can combine the neural operator with solvers (Pathak et al., 2020; Um, Raymond, et al., 2020), augmenting and correcting the solvers to get faster and more accurate approximation. Similarly, one can combine operator learning with physics constraints (S. Wang, H. Wang, and Perdikaris, 2021; Z. Li, Zheng, et al., 2021).

Theory

In this work, we develop a universal approximation theory (Section 5.4) for neural operators. As in the work of L. Lu, Jin, and George Em Karniadakis, 2019 studying universal approximation for DeepONet, we use linear approximation techniques. The power of non-linear approximation (R. A. DeVore, 1998), which is likely intrinsic to the success of neural operators in some settings, is still less studied, as discussed in Section 5.3; we note that DeepOnet is intrinsically limited by linear approximation properties. For functions between Euclidean spaces, we clearly know, by combining two layers of linear functions with one layer of non-linear activation function, the neural network can approximate arbitrary continuous functions, and that deep neural networks can be exponentially more expressive compared to shallow networks (Poole et al., 2016). However issues are less clear when it comes to the choice of architecture and the scaling of the number of parameters within neural operators between Banach spaces. The approximation theory of operators is much more complex and challenging compared to that of functions over Euclidean spaces. It is important to study the class of neural operators with respect to their architecture, i.e. what spaces the true solution operators lie in, and which classes of PDEs the neural operator approximate efficiently. We leave these as exciting, but open, research directions.

5.9 Approximation Theory Results

We write $\mathbb{N} = \{1, 2, 3, \dots\}$ and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. Furthermore, we denote by $|\cdot|_p$ the p-norm on any Euclidean space. We say $\mathcal{X}$ is a Banach space if it is a Banach space over the real field $\mathbb{R}$. We denote by $\|\cdot\|_{\mathcal{X}}$ its norm and by $\mathcal{X}^*$ its topological (continuous) dual. In particular, $\mathcal{X}^*$ is the Banach space consisting of all continuous linear functionals $f : \mathcal{X} \to \mathbb{R}$ with the operator norm

$$\|f\|_{\mathcal{X}^*} = \sup_{\substack{x \in \mathcal{X} \\ \|x\|_{\mathcal{X}}=1}} |f(x)| < \infty.$$

For any Banach space $\mathcal{Y}$, we denote by $\mathcal{L}(\mathcal{X};\mathcal{Y})$ the Banach space of continuous linear maps $T : \mathcal{X} \to \mathcal{Y}$ with the operator norm

$$\|T\|_{\mathcal{X}\to\mathcal{Y}} = \sup_{\substack{x\in\mathcal{X}\\ \|x\|_{\mathcal{X}}=1}} \|Tx\|_{\mathcal{Y}} < \infty.$$

We will abuse notation and write $\|\cdot\|$ for any operator norm when there is no ambiguity about the spaces in question.

Let $d \in \mathbb{N}$. We say that $D \subset \mathbb{R}^d$ is a *domain* if it is a bounded and connected open set that is topologically regular i.e. $\text{int}(\bar{D}) = D$. Note that, in the case $d = 1$, a domain is any bounded, open interval. For $d \geq 2$, we say D is a *Lipschitz domain* if ∂D can be locally represented as the graph of a Lipschitz continuous function defined on an open ball of $\mathbb{R}^{d-1}$. If $d = 1$, we will call any domain a Lipschitz domain. For any multi-index $\alpha \in \mathbb{N}_0^d$, we write $\partial^\alpha f$ for the α-th weak partial derivative of f when it exists.

Let $D \subset \mathbb{R}^d$ be a domain. For any $m \in \mathbb{N}_0$, we define the following spaces

$$\begin{aligned}
C(D) &= \{f : D \to \mathbb{R} : f \text{ is continuous}\},\\
C^m(D) &= \{f : D \to \mathbb{R} : \partial^\alpha f \in C^{m-|\alpha|_1}(D) \ \forall\, 0 \leq |\alpha|_1 \leq m\},\\
C^m_{\mathsf{b}}(D) &= \left\{f \in C^m(D) : \max_{0\leq|\alpha|_1\leq m} \sup_{x\in D} |\partial^\alpha f(x)| < \infty\right\},\\
C^m(\bar{D}) &= \{f \in C^m_{\mathsf{b}}(D) : \partial^\alpha f \text{ is uniformly continuous} \ \forall\, 0 \leq |\alpha|_1 \leq m\}
\end{aligned}$$

and make the equivalent definitions when D is replaced with $\mathbb{R}^d$. Note that any function in $C^m(\bar{D})$ has a unique, bounded, continuous extension from D to $\bar{D}$ and is hence uniquely defined on ∂D. We will work with this extension without further notice. We remark that when D is a Lipschitz domain, the following definition for $C^m(\bar{D})$ is equivalent

$$C^m(\bar{D}) = \{f : \bar{D} \to \mathbb{R} : \exists F \in C^m(\mathbb{R}^d) \text{ such that } f \equiv F|_{\bar{D}}\},$$

see Whitney, 1934; A. Brudnyi and Y. Brudnyi, 2012. We define $C^\infty(D) = \bigcap_{m=0}^\infty C^m(D)$ and, similarly, $C^\infty_{\mathsf{b}}(D)$ and $C^\infty(\bar{D})$. We further define

$$C^\infty_c(D) = \{f \in C^\infty(D) : \text{supp}(f) \subset D \text{ is compact}\}$$

and, again, note that all definitions hold analogously for $\mathbb{R}^d$. We denote by $\|\cdot\|_{C^m} : C^m_{\mathsf{b}}(D) \to \mathbb{R}_{\geq 0}$ the norm

$$\|f\|_{C^m} = \max_{0\leq|\alpha|_1\leq m} \sup_{x\in D} |\partial^\alpha f(x)|$$

which makes $C^m_{\mathrm{b}}(D)$ (also with $D = \mathbb{R}^d$) and $C^m(\bar{D})$ Banach spaces. For any $n \in \mathbb{N}$, we write $C(D; \mathbb{R}^n)$ for the n-fold Cartesian product of $C(D)$ and similarly for all other spaces we have defined or will define subsequently. We will continue to write $\| \cdot \|_{C^m}$ for the norm on $C^m_{\mathrm{b}}(D; \mathbb{R}^n)$ and $C^m(\bar{D}; \mathbb{R}^n)$ defined as

$$\|f\|_{C^m} = \max_{j \in \{1,\dots,n\}} \|f_j\|_{C^m}.$$

For any $m \in \mathbb{N}$ and $1 \leq p \leq \infty$, we use the notation $W^{m,p}(D)$ for the standard L^p-type Sobolev space with m derivatives; we refer the reader to Adams and Fournier, 2003 for a formal definition. Furthermore, we, at times, use the notation $W^{0,p}(D) = L^p(D)$ and $W^{m,2}(D) = H^m(D)$. Since we use the standard definitions of Sobolev spaces that can be found in any reference on the subject, we do not give the specifics here.

The Approximation Property

In this section we gather various results on the approximation property of Banach spaces. The main results are Lemma 47 which states that if two Banach spaces have the approximation property then continuous maps between them can be approximated in a finite-dimensional manner, and Lemma 51 which states the spaces in Assumptions 34 and 35 have the approximation property.

Definition 40. *A Banach space $\mathcal{X}$ has a Schauder basis if there exist some $\{\varphi_j\}_{j=1}^\infty \subset \mathcal{X}$ and $\{c_j\}_{j=1}^\infty \subset \mathcal{X}^*$ such that*

1. $c_j(\varphi_k) = \delta_{jk}$ *for any* $j, k \in \mathbb{N}$,
2. $\lim\limits_{n \to \infty} \|x - \sum_{j=1}^n c_j(x) \varphi_j\|_{\mathcal{X}} = 0$ *for all* $x \in \mathcal{X}$.

We remark that definition 40 is equivalent to the following. The elements $\{\varphi_j\}_{j=1}^\infty \subset \mathcal{X}$ are called a *Schauder basis* for $\mathcal{X}$ if, for each $x \in \mathcal{X}$, there exists a unique sequence $\{\alpha_j\}_{j=1}^\infty \subset \mathbb{R}$ such that

$$\lim_{n \to \infty} \|x - \sum_{j=1}^n \alpha_j \varphi_j\|_{\mathcal{X}} = 0.$$

For the equivalence, see, for example Albiac and Kalton, 2006, Theorem 1.1.3. Throughout this paper we will simply write the term *basis* to mean Schauder basis. Furthermore, we note that if $\{\varphi\}_{j=1}^\infty$ is a basis then so is $\{\varphi_j / \|\varphi\|_{\mathcal{X}}\}_{j=1}^\infty$, so we will assume that any basis we use is normalized.

Definition 41. *Let $\mathcal{X}$ be a Banach space and $U \in \mathcal{L}(\mathcal{X};\mathcal{X})$. U is called a finite rank operator if $U(\mathcal{X}) \subseteq \mathcal{X}$ is finite dimensional.*

By noting that any finite dimensional subspace has a basis, we may equivalently define a finite rank operator $U \in \mathcal{L}(\mathcal{X};\mathcal{X})$ to be one such that there exists a number $n \in \mathbb{N}$ and some $\{\varphi_j\}_{j=1}^n \subset \mathcal{X}$ and $\{c_j\}_{j=1}^n \subset \mathcal{X}^*$ such that

$$Ux = \sum_{j=1}^{n} c_j(x)\varphi_j, \qquad \forall x \in \mathcal{X}.$$

Definition 42. *A Banach space $\mathcal{X}$ is said to have the approximation property (*AP*) if, for any compact set $K \subset \mathcal{X}$ and $\epsilon > 0$, there exists a finite rank operator $U : \mathcal{X} \to \mathcal{X}$ such that*

$$\|x - Ux\|_{\mathcal{X}} \leq \epsilon, \qquad \forall x \in K.$$

We now state and prove some well-known results about the relationship between basis and the AP. We were unable to find the statements of the following lemmas in the form given here in the literature and therefore we provide full proofs.

Lemma 43. *Let $\mathcal{X}$ be a Banach space with a basis then $\mathcal{X}$ has the* AP.

Proof. Let $\{c_j\}_{j=1}^\infty \subset \mathcal{X}^*$ and $\{\varphi_j\}_{j=1}^\infty \subset \mathcal{X}$ be a basis for $\mathcal{X}$. Note that there exists a constant $C > 0$ such that, for any $x \in \mathcal{X}$ and $n \in \mathbb{N}$,

$$\|\sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} \leq \sup_{J \in \mathbb{N}} \|\sum_{j=1}^{J} c_j(x)\varphi_j\|_{\mathcal{X}} \leq C\|x\|_{\mathcal{X}},$$

see, for example Albiac and Kalton, 2006, Remark 1.1.6. Assume, without loss of generality, that $C \geq 1$. Let $K \subset \mathcal{X}$ be compact and $\epsilon > 0$. Since K is compact, we can find a number $n = n(\epsilon, C) \in \mathbb{N}$ and elements $y_1, \dots, y_n \in K$ such that for any $x \in K$ there exists a number $l \in \{1, \dots, n\}$ with the property that

$$\|x - y_l\|_{\mathcal{X}} \leq \frac{\epsilon}{3C}.$$

We can then find a number $J = J(\epsilon, n) \in \mathbb{N}$ such that

$$\max_{j \in \{1,\dots,n\}} \|y_j - \sum_{k=1}^{J} c_k(y_j)\varphi_k\|_{\mathcal{X}} \leq \frac{\epsilon}{3}.$$

Define the finite rank operator $U : \mathcal{X} \to \mathcal{X}$ by

$$Ux = \sum_{j=1}^{J} c_j(x)\varphi_j, \qquad \forall x \in \mathcal{X}.$$

Triangle inequality implies that, for any $x \in K$,

$$\begin{aligned} \|x - U(x)\|_{\mathcal{X}} &\leq \|x - y_l\|_{\mathcal{X}} + \|y_l - U(y_l)\|_{\mathcal{X}} + \|U(y_l) - U(x)\|_{\mathcal{X}} \\ &\leq \frac{2\epsilon}{3} + \|\sum_{j=1}^{J} \big(c_j(y_l) - c_j(x)\big)\varphi_j\|_{\mathcal{X}} \\ &\leq \frac{2\epsilon}{3} + C\|y_l - x\|_{\mathcal{X}} \\ &\leq \epsilon \end{aligned}$$

as desired. □

Lemma 44. *Let $\mathcal{X}$ be a Banach space with a basis and $\mathcal{Y}$ be any Banach space. Suppose there exists a continuous linear bijection $T : \mathcal{X} \to \mathcal{Y}$. Then $\mathcal{Y}$ has a basis.*

Proof. Let $y \in \mathcal{Y}$ and $\epsilon > 0$. Since T is a bijection, there exists an element $x \in \mathcal{X}$ so that $Tx = y$ and $T^{-1}y = x$. Since $\mathcal{X}$ has a basis, we can find $\{\varphi_j\}_{j=1}^{\infty} \subset \mathcal{X}$ and $\{c_j\}_{j=1}^{\infty} \subset \mathcal{X}^*$ and a number $n = n(\epsilon, \|T\|) \in \mathbb{N}$ such that

$$\|x - \sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} \leq \frac{\epsilon}{\|T\|}.$$

Note that

$$\|y - \sum_{j=1}^{n} c_j(T^{-1}y)T\varphi_j\|_{\mathcal{Y}} = \|Tx - T\sum_{j=1}^{n} c_j(x)\varphi_j\| \leq \|T\|\|x - \sum_{j=1}^{n} c_j(x)\varphi_j\|_{\mathcal{X}} \leq \epsilon$$

hence $\{T\varphi_j\}_{j=1}^{\infty} \subset \mathcal{Y}$ and $\{c_j(T^{-1}\cdot)\}_{j=1}^{\infty} \subset \mathcal{Y}^*$ form a basis for $\mathcal{Y}$ by linearity and continuity of T and T^{-1}. □

Lemma 45. *Let $\mathcal{X}$ be a Banach space with the* AP *and $\mathcal{Y}$ be any Banach space. Suppose there exists a continuous linear bijection $T : \mathcal{X} \to \mathcal{Y}$. Then $\mathcal{Y}$ has the* AP.

Proof. Let $K \subset \mathcal{Y}$ be a compact set and $\epsilon > 0$. The set $R = T^{-1}(K) \subset \mathcal{X}$ is compact since T^{-1} is continuous. Since $\mathcal{X}$ has the AP, there exists a finite rank operator $U : \mathcal{X} \to \mathcal{X}$ such that

$$\|x - Ux\|_{\mathcal{X}} \leq \frac{\epsilon}{\|T\|}, \qquad \forall x \in R.$$

Define the operator $W : \mathcal{Y} \to \mathcal{Y}$ by $W = TUT^{-1}$. Clearly W is a finite rank operator since U is a finite rank operator. Let $y \in K$ then, since $K = T(R)$, there exists $x \in R$ such that $Tx = y$ and $x = T^{-1}y$. Then

$$\|y - Wy\|_{\mathcal{Y}} = \|Tx - TUx\|_{\mathcal{Y}} \le \|T\|\|x - Ux\|_{\mathcal{X}} \le \epsilon,$$

hence $\mathcal{Y}$ has the AP. □

The following lemma shows than the infinite union of compact sets is compact if each set is the image of a fixed compact set under a convergent sequence of continuous maps. The result is instrumental in proving Lemma 47.

Lemma 46. *Let $\mathcal{X}, \mathcal{Y}$ be Banach spaces and $F : \mathcal{X} \to \mathcal{Y}$ be a continuous map. Let $K \subset \mathcal{X}$ be a compact set in $\mathcal{X}$ and $\{F_n : \mathcal{X} \to \mathcal{Y}\}_{n=1}^{\infty}$ be a sequence of continuous maps such that*

$$\lim_{n\to\infty} \sup_{x \in K} \|F(x) - F_n(x)\|_{\mathcal{Y}} = 0.$$

Then the set

$$W := \bigcup_{n=1}^{\infty} F_n(K) \cup F(K)$$

is compact in $\mathcal{Y}$.

Proof. Let $\epsilon > 0$ then there exists a number $N = N(\epsilon) \in \mathbb{N}$ such that

$$\sup_{x \in K} \|F(x) - F_n(x)\|_{\mathcal{Y}} \le \frac{\epsilon}{2}, \qquad \forall n \ge N.$$

Define the set

$$W_N = \bigcup_{n=1}^{N} F_n(K) \cup F(K)$$

which is compact since F and each F_n are continuous. We can therefore find a number $J = J(\epsilon, N) \in \mathbb{N}$ and elements $y_1, \dots, y_J \in W_N$ such that, for any $z \in W_N$, there exists a number $l = l(z) \in \{1, \dots, J\}$ such that

$$\|z - y_l\|_{\mathcal{Y}} \le \frac{\epsilon}{2}.$$

Let $y \in W \setminus W_N$ then there exists a number $m > N$ and an element $x \in K$ such that $y = F_m(x)$. Since $F(x) \in W_N$, we can find a number $l \in \{1, \dots, J\}$ such that

$$\|F(x) - y_l\|_{\mathcal{Y}} \le \frac{\epsilon}{2}.$$

Therefore,

$$\|y - y_l\|_{\mathcal{Y}} \leq \|F_m(x) - F(x)\|_{\mathcal{Y}} + \|F(x) - y_l\|_{\mathcal{Y}} \leq \epsilon,$$

hence $\{y_j\}_{j=1}^J$ forms a finite ϵ-net for W, showing that W is totally bounded.

We will now show that W is closed. To that end, let $\{p_n\}_{n=1}^\infty$ be a convergent sequence in W, in particular, $p_n \in W$ for every $n \in \mathbb{N}$ and $p_n \to p \in \mathcal{Y}$ as $n \to \infty$. We can thus find convergent sequences $\{x_n\}_{n=1}^\infty$ and $\{\alpha_n\}_{n=1}^\infty$ such that $x_n \in K$, $\alpha_n \in \mathbb{N}_0$, and $p_n = F_{\alpha_n}(x_n)$ where we define $F_0 := F$. Since K is closed, $\lim_{n\to\infty} x_n = x \in K$ thus, for each fixed $n \in \mathbb{N}$,

$$\lim_{j\to\infty} F_{\alpha_n}(x_j) = F_{\alpha_n}(x) \in W$$

by continuity of F_{α_n}. Since uniform convergence implies point-wise convergence

$$p = \lim_{n\to\infty} F_{\alpha_n}(x) = F_\alpha(x) \in W$$

for some $\alpha \in \mathbb{N}_0$ thus $p \in W$, showing that W is closed. □

The following lemma shows that any continuous operator acting between two Banach spaces with the AP can be approximated in a finite-dimensional manner. The approximation proceeds in three steps which are shown schematically in Figure 41. First an input is mapped to a finite-dimensional representation via the action of a set of functionals on $\mathcal{X}$. This representation is then mapped by a continuous function to a new finite-dimensional representation which serves as the set of coefficients onto representers of $\mathcal{Y}$. The resulting expansion is an element of $\mathcal{Y}$ that is ϵ-close to the action of $\mathcal{G}$ on the input element. A similar finite-dimensionalization was used in (Bhattacharya et al., 2020) by using PCA on $\mathcal{X}$ to define the functionals acting on the input and PCA on $\mathcal{Y}$ to define the output representers. However the result in that work is restricted to separable Hilbert spaces; here, we generalize it to Banach spaces with the AP.

Lemma 47. *Let $\mathcal{X}, \mathcal{Y}$ be two Banach spaces with the AP and let $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ be a continuous map. For every compact set $K \subset \mathcal{X}$ and $\epsilon > 0$, there exist numbers $J, J' \in \mathbb{N}$ and continuous linear maps $F_J : \mathcal{X} \to \mathbb{R}^J$, $G_{J'} : \mathbb{R}^{J'} \to \mathcal{Y}$ as well as $\varphi \in C(\mathbb{R}^J; \mathbb{R}^{J'})$ such that*

$$\sup_{x\in K} \|\mathcal{G}(x) - (G_{J'} \circ \varphi \circ F_J)(x)\|_{\mathcal{Y}} \leq \epsilon.$$

Furthermore there exist $w_1, \dots, w_J \in \mathcal{X}^*$ such that F_J has the form

$$F_J(x) = \big(w_1(x), \dots, w_J(x)\big), \qquad \forall x \in \mathcal{X}$$

and there exist $\beta_1, \dots, \beta_{J'} \in \mathcal{Y}$ such that $G_{J'}$ has the form

$$G_{J'}(v) = \sum_{j=1}^{J'} v_j \beta_j, \qquad \forall v \in \mathbb{R}^{J'}.$$

If $\mathcal{Y}$ admits a basis then $\{\beta_j\}_{j=1}^{J'}$ can be picked so that there is an extension $\{\beta_j\}_{j=1}^{\infty} \subset \mathcal{Y}$ which is a basis for $\mathcal{Y}$.

Proof. Since $\mathcal{X}$ has the AP, there exists a sequence of finite rank operators $\{U_n^{\mathcal{X}} : \mathcal{X} \to \mathcal{X}\}_{n=1}^{\infty}$ such that

$$\lim_{n\to\infty} \sup_{x\in K} \|x - U_n^{\mathcal{X}} x\|_{\mathcal{X}} = 0.$$

Define the set

$$Z = \bigcup_{n=1}^{\infty} U_n^{\mathcal{X}}(K) \cup K$$

which is compact by Lemma 46. Therefore, $\mathcal{G}$ is uniformly continuous on Z hence there exists a modulus of continuity $\omega : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ which is non-decreasing and satisfies $\omega(t) \to \omega(0) = 0$ as $t \to 0$ as well as

$$\|\mathcal{G}(z_1) - \mathcal{G}(z_2)\|_{\mathcal{Y}} \leq \omega\big(\|z_1 - z_2\|_{\mathcal{X}}\big) \qquad \forall z_1, z_2 \in Z.$$

We can thus find, a number $N = N(\epsilon) \in \mathbb{N}$ such that

$$\sup_{x\in K} \omega\big(\|x - U_N^{\mathcal{X}} x\|_{\mathcal{X}}\big) \leq \frac{\epsilon}{2}.$$

Let $J = \dim U_N^{\mathcal{X}}(\mathcal{X}) < \infty$. There exist elements $\{\alpha_j\}_{j=1}^{J} \subset \mathcal{X}$ and $\{w_j\}_{j=1}^{J} \subset \mathcal{X}^*$ such that

$$U_N^{\mathcal{X}} x = \sum_{j=1}^{J} w_j(x) \alpha_j, \qquad \forall x \in X.$$

Define the maps $F_J^{\mathcal{X}} : \mathcal{X} \to \mathbb{R}^J$ and $G_J^{\mathcal{X}} : \mathbb{R}^J \to \mathcal{X}$ by

$$F_J^{\mathcal{X}}(x) = (w_1(x), \dots, w_J(x)), \qquad \forall x \in \mathcal{X},$$

$$G_J^{\mathcal{X}}(v) = \sum_{j=1}^{J} v_j \alpha_j, \qquad \forall v \in \mathbb{R}^J,$$

noting that $U_N^{\mathcal{X}} = G_J^{\mathcal{X}} \circ F_J^{\mathcal{X}}$. Define the set $W = (\mathcal{G} \circ U_N^{\mathcal{X}})(\mathcal{X}) \subseteq \mathcal{Y}$ which is clearly compact. Since $\mathcal{Y}$ has the AP, we can similarly find a finite rank operator $U_{J'}^{\mathcal{Y}} : \mathcal{Y} \to \mathcal{Y}$ with $J' = \dim U_{J'}^{\mathcal{Y}}(\mathcal{Y}) < \infty$ such that

$$\sup_{y \in W} \|y - U_{J'}^{\mathcal{Y}} y\|_{\mathcal{Y}} \leq \frac{\epsilon}{2}.$$

Analogously, define the maps $F_{J'}^{\mathcal{Y}} : \mathcal{Y} \to \mathbb{R}^{J'}$ and $G_{J'}^{\mathcal{Y}} : \mathbb{R}^{J'} \to \mathcal{Y}$ by

$$\begin{aligned} F_{J'}^{\mathcal{Y}}(y) &= (q_1(y), \dots, q_{J'}(y)), && \forall y \in \mathcal{Y}, \\ G_{J'}^{\mathcal{Y}}(v) &= \sum_{j=1}^{J'} v_j \beta_j, && \forall v \in \mathbb{R}^{J'} \end{aligned}$$

for some $\{\beta_j\}_{j=1}^{J'} \subset \mathcal{Y}$ and $\{q_j\}_{j=1}^{J'} \subset \mathcal{Y}^*$ such that $U_{J'}^{\mathcal{Y}} = G_{J'}^{\mathcal{Y}} \circ F_{J'}^{\mathcal{Y}}$. Clearly if $\mathcal{Y}$ admits a basis then we could have defined $F_{J'}^{\mathcal{Y}}$ and $G_{J'}^{\mathcal{Y}}$ through it instead of through $U_{J'}^{\mathcal{Y}}$. Define $\varphi : \mathbb{R}^J \to \mathbb{R}^{J'}$ by

$$\varphi(v) = (F_{J'}^{\mathcal{Y}} \circ \mathcal{G} \circ G_J^{\mathcal{X}})(v), \qquad \forall v \in \mathbb{R}^J$$

which is clearly continuous and note that $G_{J'}^{\mathcal{Y}} \circ \varphi \circ F_J^{\mathcal{X}} = U_{J'}^{\mathcal{Y}} \circ \mathcal{G} \circ U_N^{\mathcal{X}}$. Set $F_J = F_J^{\mathcal{X}}$ and $G_{J'} = G_{J'}^{\mathcal{Y}}$ then, for any $x \in K$,

$$\begin{aligned} \|\mathcal{G}(x) - (G_{J'} \circ \varphi \circ F_J)(x)\|_{\mathcal{Y}} &\leq \|\mathcal{G}(x) - \mathcal{G}(U_N^{\mathcal{X}} x)\|_{\mathcal{Y}} \\ &\quad + \|\mathcal{G}(U_N^{\mathcal{X}} x) - (U_{J'}^{\mathcal{Y}} \circ \mathcal{G} \circ U_N^{\mathcal{X}})(x)\|_{\mathcal{Y}} \\ &\leq \omega\big(\|x - U_N^{\mathcal{X}} x\|_{\mathcal{X}}\big) + \sup_{y \in W} \|y - U_{J'}^{\mathcal{Y}} y\|_{\mathcal{Y}} \\ &\leq \epsilon \end{aligned}$$

as desired. □

We now state and prove some results about isomorphisms of function spaces defined on different domains. These results are instrumental in proving Lemma 51.

Lemma 48. *Let $D, D' \subset \mathbb{R}^d$ be domains. Suppose that, for some $m \in \mathbb{N}_0$, there exists a C^m-diffeomorphism $\tau : \bar{D}' \to \bar{D}$. Then the mapping $T : C^m(\bar{D}) \to C^m(\bar{D}')$ defined as*

$$T(f)(x) = f(\tau(x)), \qquad \forall f \in C^m(\bar{D}),\ x \in \bar{D}'$$

is a continuous linear bijection.

Proof. Clearly T is linear since the evaluation functional is linear. To see that it is continuous, note that by the chain rule we can find a constant $Q = Q(m) > 0$ such that

$$\|T(f)\|_{C^m} \leq Q\|\tau\|_{C^m}\|f\|_{C^m}, \qquad \forall f \in C^m(\bar{D}).$$

We will now show that it is bijective. Let $f, g \in C^m(\bar{D})$ so that $f \neq g$. Then there exists a point $x \in \bar{D}$ such that $f(x) \neq g(x)$. Then $T(f)(\tau^{-1}(x)) = f(x)$ and $T(g)(\tau^{-1}(x)) = g(x)$ hence $T(f) \neq T(g)$ thus T is injective. Now let $g \in C^m(\bar{D}')$ and define $f : \bar{D} \to \mathbb{R}$ by $f = g \circ \tau^{-1}$. Since $\tau^{-1} \in C^m(\bar{D}; \bar{D}')$, we have that $f \in C^m(\bar{D})$. Clearly, $T(f) = g$ hence T is surjective. □

Corollary 49. *Let $M > 0$ and $m \in \mathbb{N}_0$. There exists a continuous linear bijection $T : C^m([0,1]^d) \to C^m([-M, M]^d)$.*

Proof. Let $\mathbb{1} \in \mathbb{R}^d$ denote the vector in which all entries are 1. Define the map $\tau : \mathbb{R}^d \to \mathbb{R}^d$ by

$$\tau(x) = \frac{1}{2M}x + \frac{1}{2}\mathbb{1}, \qquad \forall x \in \mathbb{R}^d. \tag{5.45}$$

Clearly τ is a C^∞-diffeomorphism between $[-M, M]^d$ and $[0,1]^d$ hence Lemma 48 implies the result. □

Lemma 50. *Let $M > 0$ and $m \in \mathbb{N}$. There exists a continuous linear bijection $T : W^{m,1}((0,1)^d) \to W^{m,1}((-M, M)^d)$.*

Proof. Define the map $\tau : \mathbb{R}^d \to \mathbb{R}^d$ by (5.45). We have that $\tau((-M, M)^d) = (0,1)^d$. Define the operator T by

$$Tf = f \circ \tau, \qquad \forall f \in W^{m,1}((0,1)^d).$$

which is clearly linear since composition is linear. We compute that, for any $0 \leq |\alpha|_1 \leq m$,

$$\partial^\alpha(f \circ \tau) = (2M)^{-|\alpha|_1}(\partial^\alpha f) \circ \tau,$$

hence, by the change of variables formula,

$$\|Tf\|_{W^{m,1}((-M,M)^d)} = \sum_{0 \leq |\alpha|_1 \leq m} (2M)^{d-|\alpha|_1}\|\partial^\alpha f\|_{L^1((0,1)^d)}.$$

We can therefore find numbers $C_1, C_2 > 0$, depending on M and m, such that

$$C_1\|f\|_{W^{m,1}((0,1)^d)} \leq \|Tf\|_{W^{m,1}((-M,M)^d)} \leq C_2\|f\|_{W^{m,1}((0,1)^d)}.$$

This shows that $T : W^{m,1}((0,1)^d) \to W^{m,1}((-M,M)^d)$ is continuous and injective. Now let $g \in W^{m,1}((-M,M)^d)$ and define $f = g \circ \tau^{-1}$. A similar argument shows that $f \in W^{m,1}((0,1)^d)$ and, clearly, $Tf = g$ hence T is surjective. $\square$

We now show that the spaces in Assumptions 34 and 35 have the AP. While the result is well-known when the domain is $(0,1)^d$ or $\mathbb{R}^d$, we were unable to find any results in the literature for Lipschitz domains and we therefore give a full proof here. The essence of the proof is to either exhibit an isomorphism to a space that is already known to have AP or to directly show AP by embedding the Lipschitz domain into an hypercube for which there are known basis constructions. Our proof shows the stronger result that $W^{m,p}(D)$ for $m \in \mathbb{N}_0$ and $1 \leq p < \infty$ has a basis, but, for $C^m(\bar{D})$, we only establish the AP and not necessarily a basis. The discrepancy comes from the fact that there is an isomorphism between $W^{m,p}(D)$ and $W^{m,p}(\mathbb{R}^d)$ while there is not one between $C^m(\bar{D})$ and $C^m(\mathbb{R}^d)$.

Lemma 51. *Let Assumptions 34 and 35 hold. Then $\mathcal{A}$ and $\mathcal{U}$ have the* AP.

Proof. It is enough to show that the spaces $W^{m,p}(D)$, and $C^m(\bar{D})$ for any $1 \leq p < \infty$ and $m \in \mathbb{N}_0$ with $D \subset \mathbb{R}^d$ a Lipschitz domain have the AP. Consider first the spaces $W^{0,p}(D) = L^p(D)$. Since the Lebesgue measure on D is σ-finite and has no atoms, $L^p(D)$ is isometrically isomorphic to $L^p((0,1))$ (see, for example, Albiac and Kalton, 2006, Chapter 6). Hence by Lemma 45, it is enough to show that $L^p((0,1))$ has the AP. Similarly, consider the spaces $W^{m,p}(D)$ for $m > 0$ and $p > 1$. Since D is Lipschitz, there exists a continuous linear operator $W^{m,p}(D) \to W^{m,p}(\mathbb{R}^d)$ Stein, 1970, Chapter 6, Theorem 5 (this also holds for $p = 1$). We can therefore apply Pełczyński and Wojciechowski, 2001, Corollary 4 (when $p > 1$) to conclude that $W^{m,p}(D)$ is isomorphic to $L^p((0,1))$. By Albiac and Kalton, 2006, Proposition 6.1.3, $L^p((0,1))$ has a basis hence Lemma 43 implies the result.

Now, consider the spaces $C^m(\bar{D})$. Since D is bounded, there exists a number $M > 0$ such that $\bar{D} \subseteq [-M,M]^d$. Hence, by Corollary 49, $C^m([0,1]^d)$ is isomorphic to $C^m([-M,M]^d)$. Since $C^m([0,1]^d)$ has a basis Ciesielski and Domsta, 1972, Theorem 5, Lemma 44 then implies that $C^m([-M,M]^d)$ has a basis. By Fefferman, 2007, Theorem 1, there exists a continuous linear operator $E : C^m(\bar{D}) \to C^m_{\text{b}}(\mathbb{R}^d)$ such that $E(f)|_{\bar{D}} = f$ for all $f \in C(\bar{D})$. Define the restriction operators $R_M : C^m_{\text{b}}(\mathbb{R}^d) \to C^m([-M,M]^d)$ and $R_D : C^m([-M,M]^d) \to C^m(\bar{D})$ which are both clearly linear and continuous and $\|R_M\| = \|R_D\| = 1$. Let $\{c_j\}_{j=1}^{\infty} \subset \big(C^m([-M,M]^d)\big)^*$ and $\{\varphi_j\}_{j=1}^{\infty} \subset C^m([-M,M]^d)$ be a basis for

$C^m([-M,M]^d)$. As in the proof of Lemma 43, there exists a constant $C_1 > 0$ such that, for any $n \in \mathbb{N}$ and $f \in C^m([-M,M]^d)$,

$$\|\sum_{j=1}^{n} c_j(f)\varphi_j\|_{C^m([-M,M]^d)} \leq C_1 \|f\|_{C^m([-M,M]^d)}.$$

Suppose, without loss of generality, that $C_1\|E\| \geq 1$. Let $K \subset C^m(\bar{D})$ be a compact set and $\epsilon > 0$. Since K is compact, we can find a number $n = n(\epsilon) \in \mathbb{N}$ and elements $y_1, \dots, y_n \in K$ such that, for any $f \in K$ there exists a number $l \in \{1, \dots, n\}$ such that

$$\|f - y_l\|_{C^m(\bar{D})} \leq \frac{\epsilon}{3C_1\|E\|}.$$

For every $l \in \{1, \dots, n\}$, define $g_l = R_M(E(y_l))$ and note that $g_l \in C^m([-M,M]^d)$ hence there exists a number $J = J(\epsilon, n) \in \mathbb{N}$ such that

$$\max_{l \in \{1,\dots,n\}} \|g_l - \sum_{j=1}^{J} c_j(g_l)\varphi_j\|_{C^m([-M,M]^d)} \leq \frac{\epsilon}{3}.$$

Notice that, since $y_l = R_D(g_l)$, we have

$$\begin{aligned}\max_{l \in \{1,\dots,n\}} \|y_l - \sum_{j=1}^{J} c_j\big(R_M(E(y_l))\big)R_D(\varphi_j)\|_{C^m(\bar{D})} \leq \\ \|R_D\| \max_{l \in \{1,\dots,n\}} \|g_l - \sum_{j=1}^{J} c_j(g_l)\varphi_j\|_{C^m([-M,M]^d)} \\ \leq \frac{\epsilon}{3}.\end{aligned}$$

Define the finite rank operator $U : C^m(\bar{D}) \to C^m(\bar{D})$ by

$$Uf = \sum_{j=1}^{J} c_j\big(R_M(E(f))\big)R_D(\varphi_j), \qquad \forall f \in C^m(\bar{D}).$$

We then have that, for any $f \in K$,

$$\begin{aligned}\|f - Uf\|_{C^m(\bar{D})} &\leq \|f - y_l\|_{C^m(\bar{D})} + \|y_l - Uy_l\|_{C^m(\bar{D})} + \|Uy_l - Uf\|_{C^m(\bar{D})} \\ &\leq \frac{2\epsilon}{3} + \|\sum_{j=1}^{J} c_j\big(R_M(E(y_l - f))\big)\varphi_j\|_{C^m([-M,M]^d)} \\ &\leq \frac{2\epsilon}{3} + C_1\|R_M(E(y_l - f))\|_{C^m([-M,M]^d)} \\ &\leq \frac{2\epsilon}{3} + C_1\|E\|\|y_l - f\|_{C^m(\bar{D})} \\ &\leq \epsilon,\end{aligned}$$

hence $C^m(\bar{D})$ has the AP.

We are left with the case $W^{m,1}(D)$. A similar argument as for the $C^m(\bar{D})$ case holds. In particular the basis from Ciesielski and Domsta, 1972, Theorem 5 is also a basis for $W^{m,1}((0,1)^d)$. Lemma 50 gives an isomorphism between $W^{m,1}((0,1)^d)$ and $W^{m,1}((-M,M)^d)$ hence we may use the extension operator $W^{m,1}(D) \to W^{m,1}(\mathbb{R}^d)$ from Stein, 1970, Chapter 6, Theorem 5 to complete the argument. In fact, the same construction yields a basis for $W^{m,1}(D)$ due to the isomorphism with $W^{m,1}(\mathbb{R}^d)$, see, for example Pełczyński and Wojciechowski, 2001, Theorem 1. □

Representation Theorems

In this section, we prove various results about the approximation of linear functionals by kernel integral operators. Lemma 52 establishes a Reisz-representation theorem for C^m. The proof proceeds exactly as in the well-known result for $W^{m,p}$ but, since we did not find it in the literature, we give full details here. Lemma 53 shows that linear functionals on $W^{m,p}$ can be approximated uniformly over compact set by integral kernel operators with a C^∞ kernel. Lemmas 55 and 56 establish similar results for C and C^m respectively by employing Lemma 52. These lemmas are crucial in showing that NO(s) are universal since they imply that the functionals from Lemma 47 can be approximated by elements of IO.

Lemma 52. *Let $D \subset \mathbb{R}^d$ be a domain and $m \in \mathbb{N}_0$. For every $L \in \big(C^m(\bar{D})\big)^*$ there exist finite, signed, Radon measures $\{\lambda_\alpha\}_{0 \le |\alpha|_1 \le m}$ such that*

$$L(f) = \sum_{0 \le |\alpha|_1 \le m} \int_{\bar{D}} \partial^\alpha f \, \mathsf{d}\lambda_\alpha, \qquad \forall f \in C^m(\bar{D}).$$

Proof. The case $m = 0$ follow directly from Leoni, 2009, Theorem B.111, so we assume that $m > 0$. Let $\alpha_1, \dots, \alpha_J$ be an enumeration of the set $\{\alpha \in \mathbb{N}^d : |\alpha|_1 \le m\}$. Define the mapping $T : C^m(\bar{D}) \to C(\bar{D}; \mathbb{R}^J)$ by

$$Tf = \big(\partial^{\alpha_0} f, \dots, \partial^{\alpha_J} f\big), \qquad \forall f \in C^m(\bar{D}).$$

Clearly $\|Tf\|_{C(\bar{D};\mathbb{R}^J)} = \|f\|_{C^m(\bar{D})}$ hence T is an injective, continuous linear operator. Define $W \coloneqq T(C^m(\bar{D})) \subset C(\bar{D}; \mathbb{R}^J)$ then $T^{-1} : W \to C^m(\bar{D})$ is a continuous linear operator since T preserves norm. Thus $W = \big(T^{-1}\big)^{-1}(C^m(\bar{D}))$ is closed as the pre-image of a closed set under a continuous map. In particular, W is a Banach space since $C(\bar{D}; \mathbb{R}^J)$ is a Banach space and T is an isometric isomorphism between $C^m(\bar{D})$ and W. Therefore, there exists a continuous linear functional $\tilde{L} \in W^*$ such that

$$L(f) = \tilde{L}(Tf), \qquad \forall f \in C^m(\bar{D}).$$

By the Hahn-Banach theorem, $\tilde{L}$ can be extended to a continuous linear functional $\bar{L} \in \big(C(\bar{D};\mathbb{R}^J)\big)^*$ such that $\|L\|_{(C^m(\bar{D}))^*} = \|\tilde{L}\|_{W^*} = \|\bar{L}\|_{(C(\bar{D};\mathbb{R}^J))^*}$. We have that

$$L(f) = \tilde{L}(Tf) = \bar{L}(Tf), \qquad \forall f \in C^m(\bar{D}).$$

Since

$$\big(C(\bar{D};\mathbb{R}^J)\big)^* \cong \bigtimes_{j=1}^{J} \big(C(\bar{D})\big)^* \cong \bigoplus_{j=1}^{J} \big(C(\bar{D})\big)^*,$$

we have, by applying Leoni, 2009, Theorem B.111 J times, that there exist finite, signed, Radon measures $\{\lambda_\alpha\}_{0\leq|\alpha|_1\leq m}$ such that

$$\bar{L}(Tf) = \sum_{0\leq|\alpha|_1\leq m} \int_{\bar{D}} \partial^\alpha f \, \mathsf{d}\lambda_\alpha, \qquad \forall f \in C^m(\bar{D})$$

as desired. □

Lemma 53. *Let $D \subset \mathbb{R}^d$ be a bounded, open set and $L \in (W^{m,p}(D))^*$ for some $m \geq 0$ and $1 \leq p < \infty$. For any closed and bounded set $K \subset W^{m,p}(D)$ (compact if $p = 1$) and $\epsilon > 0$, there exists a function $\kappa \in C_c^\infty(D)$ such that*

$$\sup_{u\in K} |L(u) - \int_D \kappa u \, \mathsf{d}x| < \epsilon.$$

Proof. First consider the case $m = 0$ and $1 \leq p < \infty$. By the Reisz Representation Theorem Conway, 2007, Appendix B, there exists a function $v \in L^q(D)$ such that

$$L(u) = \int_D vu \, \mathsf{d}x.$$

Since K is bounded, there is a constant $M > 0$ such that

$$\sup_{u\in K} \|u\|_{L^p} \leq M.$$

Suppose $p > 1$, so that $1 < q < \infty$. Density of $C_c^\infty(D)$ in $L^q(D)$ Adams and Fournier, 2003, Corollary 2.30 implies there exists a function $\kappa \in C_c^\infty(D)$ such that

$$\|v - \kappa\|_{L^q} < \frac{\epsilon}{M}.$$

By the Hölder inequality,

$$|L(u) - \int_D \kappa u \, \mathsf{d}x| \leq \|u\|_{L^p}\|v - \kappa\|_{L^q} < \epsilon.$$

Suppose that $p = 1$ then $q = \infty$. Since K is totally bounded, there exists a number $n \in \mathbb{N}$ and functions $g_1, \dots, g_n \in K$ such that, for any $u \in K$,

$$\|u - g_l\|_{L^1} < \frac{\epsilon}{3\|v\|_{L^\infty}}$$

for some $l \in \{1, \dots, n\}$. Let $\psi_\eta \in C_c^\infty(D)$ denote a standard mollifier for any $\eta > 0$. We can find $\eta > 0$ small enough such that

$$\max_{l \in \{1,\dots,n\}} \|\psi_\eta * g_l - g_l\|_{L^1} < \frac{\epsilon}{9\|v\|_{L^\infty}}$$

Define $f = \psi_\eta * v \in C(D)$ and note that $\|f\|_{L^\infty} \leq \|v\|_{L^\infty}$. By Fubini's theorem, we find

$$|\int_D (f - v) g_l \, \mathsf{d}x| = \int_D v(\psi_\eta * g_l - g_l) \, \mathsf{d}x \leq \|v\|_{L^\infty} \|\psi_\eta * g_l - g_l\|_{L^1} < \frac{\epsilon}{9}.$$

Since $g_l \in L^1(D)$, by Lusin's theorem, we can find a compact set $A \subset D$ such that

$$\max_{l \in \{1,\dots,n\}} \int_{D \setminus A} |g_l| \, \mathsf{d}x < \frac{\epsilon}{18\|v\|_{L^\infty}}.$$

Since $C_c^\infty(D)$ is dense in $C(D)$ over compact sets Leoni, 2009, Theorem C.16, we can find a function $\kappa \in C_c^\infty(D)$ such that

$$\sup_{x \in A} |\kappa(x) - f(x)| \leq \frac{\epsilon}{9M}$$

and $\|\kappa\|_{L^\infty} \leq \|f\|_{L^\infty} \leq \|v\|_{L^\infty}$. We have

$$\begin{aligned}
|\int_D (\kappa - v) g_l \, \mathsf{d}x| &\leq \int_A |(\kappa - v) g_l| \, \mathsf{d}x + \int_{D \setminus A} |(\kappa - v) g_l| \, \mathsf{d}x \\
&\leq \int_A |(\kappa - f) g_l| \, \mathsf{d}x + \int_D |(f - v) g_l| \, \mathsf{d}x + 2\|v\|_{L^\infty} \int_{D \setminus A} |g_l| \, \mathsf{d}x \\
&\leq \sup_{x \in A} |\kappa(x) - f(x)| \|g_l\|_{L^1} + \frac{2\epsilon}{9} \\
&< \frac{\epsilon}{3}.
\end{aligned}$$

Finally,

$$\begin{aligned}
|L(u) - \int_D \kappa u \, \mathsf{d}x| &\leq |\int_D vu \, \mathsf{d}x - \int_D vg_l \, \mathsf{d}x| + |\int_D vg_l \, \mathsf{d}x - \int_D \kappa u \, \mathsf{d}x| \\
&\leq \|v\|_{L^\infty} \|u - g_l\|_{L^1} + |\int_D \kappa u \, \mathsf{d}x - \int_D \kappa g_l \, \mathsf{d}x| \\
&\quad + |\int_D \kappa g_l \, \mathsf{d}x - \int_D vg_l \, \mathsf{d}x| \\
&\leq \frac{\epsilon}{3} + \|\kappa\|_{L^\infty} \|u - g_l\|_{L^1} + |\int_D (\kappa - v) g_l \, \mathsf{d}x| \\
&\leq \frac{2\epsilon}{3} + \|v\|_{L^\infty} \|u - g_l\|_{L^1} \\
&< \epsilon.
\end{aligned}$$

Suppose $m \geq 1$. By the Reisz Representation Theorem Adams and Fournier, 2003, Theorem 3.9, there exist elements $(v_\alpha)_{0 \leq |\alpha|_1 \leq m}$ of $L^q(D)$ where $\alpha \in \mathbb{N}^d$ is a multi-index such that

$$L(u) = \sum_{0 \leq |\alpha|_1 \leq m} \int_D v_\alpha \partial_\alpha u \, \mathsf{d}x.$$

Since K is bounded, there is a constant $M > 0$ such that

$$\sup_{u \in K} \|u\|_{W^{m,p}} \leq M.$$

Suppose $p > 1$, so that $1 < q < \infty$. Density of $C_0^\infty(D)$ in $L^q(D)$ implies there exist functions $(f_\alpha)_{0 \leq |\alpha|_1 \leq m}$ in $C_c^\infty(D)$ such that

$$\|f_\alpha - v_\alpha\|_{L^q} < \frac{\epsilon}{MJ}$$

where $J = |\{\alpha \in \mathbb{N}^d : |\alpha|_1 \leq m\}|$. Let

$$\kappa = \sum_{0 \leq |\alpha|_1 \leq m} (-1)^{|\alpha|_1} \partial_\alpha f_\alpha$$

then, by definition of a weak derivative,

$$\int_D \kappa u \, \mathsf{d}x = \sum_{0 \leq |\alpha|_1 \leq m} (-1)^{|\alpha|_1} \int_D \partial_\alpha f_\alpha u \, \mathsf{d}x = \sum_{0 \leq |\alpha|_1 \leq m} \int_D f_\alpha \partial_\alpha u \, \mathsf{d}x.$$

By the Hölder inequality,

$$|L(u) - \int_D \kappa u \, \mathsf{d}x| \leq \sum_{0 \leq |\alpha|_1 \leq m} \|\partial_\alpha u\|_{L^p} \|f_\alpha - v_\alpha\|_{L^q} < M \sum_{0 \leq |\alpha|_1 \leq m} \frac{\epsilon}{MJ} = \epsilon.$$

Suppose that $p = 1$ then $q = \infty$. Define the constant $C_v > 0$ by

$$C_v = \sum_{0 \leq |\alpha|_1 \leq m} \|v_\alpha\|_{L^\infty}.$$

Since K is totally bounded, there exists a number $n \in \mathbb{N}$ and functions $g_1, \dots, g_n \in K$ such that, for any $u \in K$,

$$\|u - g_l\|_{W^{m,1}} < \frac{\epsilon}{3C_v}$$

for some $l \in \{1, \dots, n\}$. Let $\psi_\eta \in C_c^\infty(D)$ denote a standard mollifier for any $\eta > 0$. We can find $\eta > 0$ small enough such that

$$\max_\alpha \max_{l \in \{1,\dots,n\}} \|\psi_\eta * \partial_\alpha g_l - \partial_\alpha g_l\|_{L^1} < \frac{\epsilon}{9C_v}.$$

Define $f_\alpha = \psi_\eta * v_\alpha \in C(D)$ and note that $\|f_\alpha\|_{L^\infty} \leq \|v_\alpha\|_{L^\infty}$. By Fubini's theorem, we find

$$\begin{aligned}\sum_{0 \leq |\alpha|_1 \leq m} \left| \int_D (f_\alpha - v_\alpha) \partial_\alpha g_l \, \mathsf{d}x \right| &= \sum_{0 \leq |\alpha|_1 \leq m} \left| \int_D v_\alpha (\psi_\eta * \partial_\alpha g_l - \partial_\alpha g_l) \, \mathsf{d}x \right| \\ &\leq \sum_{0 \leq |\alpha|_1 \leq m} \|v_\alpha\|_{L^\infty} \|\psi_\eta * \partial_\alpha g_l - \partial_\alpha g_l\|_{L^1} \\ &< \frac{\epsilon}{9}.\end{aligned}$$

Since $\partial_\alpha g_l \in L^1(D)$, by Lusin's theorem, we can find a compact set $A \subset D$ such that

$$\max_\alpha \max_{l \in \{1,\dots,n\}} \int_{D \setminus A} |\partial_\alpha g_l| \, \mathsf{d}x < \frac{\epsilon}{18C_v}.$$

Since $C_c^\infty(D)$ is dense in $C(D)$ over compact sets, we can find functions $w_\alpha \in C_c^\infty(D)$ such that

$$\sup_{x \in A} |w_\alpha(x) - f_\alpha(x)| \leq \frac{\epsilon}{9MJ}$$

where $J = |\{\alpha \in \mathbb{N}^d : |\alpha|_1 \leq m\}|$ and $\|w_\alpha\|_{L^\infty} \leq \|f_\alpha\|_{L^\infty} \leq \|v_\alpha\|_{L^\infty}$. We have,

$$\begin{aligned}
\sum_{0\leq|\alpha|_1\leq m} \int_D |(w_\alpha - v_\alpha)\partial_\alpha g_l| &= \sum_{0\leq|\alpha|_1\leq m} \Big(\int_A |(w_\alpha - v_\alpha)\partial_\alpha g_l| dx \\
&\quad + \int_{D\setminus A} |(w_\alpha - v_\alpha)\partial_\alpha g_l| dx \Big) \\
&\leq \sum_{0\leq|\alpha|_1\leq m} \Big(\int_A |(w_\alpha - f_\alpha)\partial_\alpha g_l| \,\mathsf{d}x \\
&\quad + \int_D |(f_\alpha - v_\alpha)\partial_\alpha g_l| \,\mathsf{d}x + 2\|v_\alpha\|_{L^\infty} \int_{D\setminus A} |\partial_\alpha g_l| \,\mathsf{d}x \Big) \\
&\leq \sum_{0\leq|\alpha|_1\leq m} \sup_{x\in A} |w_\alpha(x) - f_\alpha(x)| \|\partial_\alpha g_l\|_{L^1} + \frac{2\epsilon}{9} \\
&< \frac{\epsilon}{3}.
\end{aligned}$$

Let

$$\kappa = \sum_{0\leq|\alpha|_1\leq m} (-1)^{|\alpha|_1} \partial_\alpha w_\alpha.$$

then, by definition of a weak derivative,

$$\int_D \kappa u \,\mathsf{d}x = \sum_{0\leq|\alpha|_1\leq m} (-1)^{|\alpha|_1} \int_D \partial_\alpha w_\alpha u \,\mathsf{d}x = \sum_{0\leq|\alpha|_1\leq m} \int_D w_\alpha \partial_\alpha u \,\mathsf{d}x.$$

Finally,

$$\begin{aligned}
|L(u) - \int_D \kappa u \,\mathsf{d}x| &\leq \sum_{0\leq|\alpha|_1\leq m} \int_D |v_\alpha \partial_\alpha u - w_\alpha \partial_\alpha u| \,\mathsf{d}x \\
&\leq \sum_{0\leq|\alpha|_1\leq m} \Big(\int_D |v_\alpha(\partial_\alpha u - \partial_\alpha g_l)| \,\mathsf{d}x \\
&\quad + \int_D |v_\alpha \partial_\alpha g_l - w_\alpha \partial_\alpha u| \,\mathsf{d}x \Big) \\
&\leq \sum_{0\leq|\alpha|_1\leq m} \Big(\|v_\alpha\|_{L^\infty} \|u - g_l\|_{W^{m,1}} + \int_D |(v_\alpha - w_\alpha)\partial_\alpha g_l| \,\mathsf{d}x \\
&\quad + \int_D |(\partial_\alpha g_l - \partial_\alpha u) w_\alpha| \,\mathsf{d}x \Big) \\
&< \frac{2\epsilon}{3} + \sum_{0\leq|\alpha|_1\leq m} \|w_\alpha\|_{L^\infty} \|u - g_l\|_{W^{m,1}} \\
&< \epsilon.
\end{aligned}$$

□

Lemma 54. *Let $D \subset \mathbb{R}^d$ be a domain and $L \in \left(C^m(\bar{D})\right)^*$ for some $m \in \mathbb{N}_0$. For any compact set $K \subset C^m(\bar{D})$ and $\epsilon > 0$, there exists distinct points $y_{11}, \dots, y_{1n_1}, \dots, y_{Jn_J} \in D$ and numbers $c_{11}, \dots, c_{1n_1}, \dots, c_{Jn_J} \in \mathbb{R}$ such that*

$$\sup_{u \in K} |L(u) - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \epsilon$$

where $\alpha_1, \dots, \alpha_J$ is an enumeration of the set $\{\alpha \in \mathbb{N}_0^d : 0 \leq |\alpha|_1 \leq m\}$.

Proof. By Lemma 52, there exist finite, signed, Radon measures $\{\lambda_\alpha\}_{0 \leq |\alpha|_1 \leq m}$ such that

$$L(u) = \sum_{0 \leq |\alpha|_1 \leq m} \int_{\bar{D}} \partial^\alpha u \, \mathsf{d}\lambda_\alpha, \qquad \forall u \in C^m(\bar{D}).$$

Let $\alpha_1, \dots, \alpha_J$ be an enumeration of the set $\{\alpha \in \mathbb{N}_0^d : 0 \leq |\alpha|_1 \leq m\}$. By weak density of the Dirac measures Bogachev, 2007, Example 8.1.6, we can find points $y_{11}, \dots, y_{1n_1}, \dots, y_{J1}, \dots, y_{Jn_J} \in \bar{D}$ as well as numbers $c_{11}, \dots, c_{Jn_J} \in \mathbb{R}$ such that

$$|\int_{\bar{D}} \partial^{\alpha_j} u \, \mathsf{d}\lambda_{\alpha_j} - \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{4J}, \qquad \forall u \in C^m(\bar{D})$$

for any $j \in \{1, \dots, J\}$. Therefore,

$$|\sum_{j=1}^{J} \int_{\bar{D}} \partial^{\alpha_j} u \, \mathsf{d}\lambda_{\alpha_j} - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{4}, \qquad \forall u \in C^m(\bar{D}).$$

Define the constant

$$Q \coloneqq \sum_{j=1}^{J} \sum_{k=1}^{n_j} |c_{jk}|.$$

Since K is compact, we can find functions $g_1, \dots, g_N \in K$ such that, for any $u \in K$, there exists $l \in \{1, \dots, N\}$ such that

$$\|u - g_l\|_{C^k} \leq \frac{\epsilon}{4Q}.$$

Suppose that some $y_{jk} \in \partial D$. By uniform continuity, we can find a point $\tilde{y}_{jk} \in D$ such that

$$\max_{l \in \{1, \dots, N\}} |\partial^{\alpha_j} g_l(y_{jk}) - \partial^{\alpha_j} g_l(\tilde{y}_{jk})| \leq \frac{\epsilon}{4Q}.$$

Denote

$$S(u) = \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})$$

and by $\tilde{S}(u)$ the sum $S(u)$ with y_{jk} replaced by $\tilde{y}_{jk}$. Then, for any $u \in K$, we have

$$\begin{aligned}|L(u) - \tilde{S}(u)| &\leq |L(u) - S(u)| + |S(u) - \tilde{S}(u)| \\ &\leq \frac{\epsilon}{4} + |c_{jk}\partial^{\alpha_j} u(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j} u(y_{jk})| \\ &\leq \frac{\epsilon}{4} + |c_{jk}\partial^{\alpha_j} u(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j} g_l(\tilde{y}_{jk})| \\ &\quad + |c_{jk}\partial^{\alpha_j} g_l(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j} u(y_{jk})| \\ &\leq \frac{\epsilon}{4} + |c_{jk}|\|u - g_l\|_{C^m} + |c_{jk}\partial^{\alpha_j} g_l(\tilde{y}_{jk}) - c_{jk}\partial^{\alpha_j} g_l(y_{jk})| \\ &\quad + |c_{jk}\partial^{\alpha_j} g_l(y_{jk}) - c_{jk}\partial^{\alpha_j} u(y_{jk})| \\ &\leq \frac{\epsilon}{4} + 2|c_{jk}|\|u - g_l\|_{C^m} + |c_{jk}||\partial^{\alpha_j} g_l(\tilde{y}_{jk}) - \partial^{\alpha_j} g_l(y_{jk})| \\ &\leq \epsilon.\end{aligned}$$

Since there are a finite number of points, this implies that all points y_{jk} can be chosen in D. Suppose now that $y_{jk} = y_{qp}$ for some $(j,k) \neq (q,p)$. As before, we can always find a point $\tilde{y}_{jk}$ distinct from all others such that

$$\max_{l \in \{1,\dots,N\}} |\partial^{\alpha_j} g_l(y_{jk}) - \partial^{\alpha_j} g_l(\tilde{y}_{jk})| \leq \frac{\epsilon}{4Q}.$$

Repeating the previous argument then shows that all points y_{jk} can be chosen distinctly as desired. □

Lemma 55. *Let $D \subset \mathbb{R}^d$ be a domain and $L \in \big(C(\bar{D})\big)^*$. For any compact set $K \subset C(\bar{D})$ and $\epsilon > 0$, there exists a function $\kappa \in C_c^\infty(D)$ such that*

$$\sup_{u \in K} |L(u) - \int_D \kappa u \, \mathsf{d}x| < \epsilon.$$

Proof. By Lemma 54, we can find points distinct points $y_1, \dots, y_n \in D$ as well as numbers $c_1, \dots, c_n \in \mathbb{R}$ such that

$$\sup_{u \in K} |L(u) - \sum_{j=1}^n c_j u(y_j)| \leq \frac{\epsilon}{3}.$$

Define the constants

$$Q \coloneqq \sum_{j=1}^n |c_j|.$$

Since K is compact, there exist functions $g_1, \dots, g_J \in K$ such that, for any $u \in K$, there exists some $l \in \{1, \dots, J\}$ such that

$$\|u - g_l\|_C \leq \frac{\epsilon}{6nQ}.$$

Let $r > 0$ be such that the open balls $B_r(y_j) \subset D$ and are pairwise disjoint. Let $\psi_\eta \in C_c^\infty(\mathbb{R}^d)$ denote the standard mollifier with parameter $\eta > 0$, noting that supp $\psi_r = B_r(0)$. We can find a number $0 < \gamma \leq r$ such that

$$\max_{\substack{l \in \{1,\dots,J\} \\ j \in \{1,\dots,n\}}} \Big| \int_D \psi_\gamma(x - y_j) g_l(x) \,\mathsf{d}x - g_l(y_j) \Big| \leq \frac{\epsilon}{3nQ}.$$

Define $\kappa : \mathbb{R}^d \to \mathbb{R}$ by

$$\kappa(x) = \sum_{j=1}^n c_j \psi_\gamma(x - y_j), \qquad \forall x \in \mathbb{R}^d.$$

Since supp $\psi_\gamma(\cdot - y_j) \subseteq B_r(y_j)$, we have that $\kappa \in C_c^\infty(D)$. Then, for any $u \in K$,

$$\begin{aligned}
|L(u) - \int_D \kappa u \,\mathsf{d}x| &\leq |L(u) - \sum_{j=1}^n c_j u(y_j)| + |\sum_{j=1}^n c_j u(y_j) - \int_D \kappa u \,\mathsf{d}x| \\
&\leq \frac{\epsilon}{3} + \sum_{j=1}^n |c_j||u(y_j) - \int_D \psi_\eta(x - y_j) u(x) \,\mathsf{d}x| \\
&\leq \frac{\epsilon}{3} + Q \sum_{j=1}^n |u(y_j) - g_l(y_j)| + |g_l(y_j) - \int_D \psi_\eta(x - y_j) u(x) \,\mathsf{d}x| \\
&\leq \frac{\epsilon}{3} + nQ\|u - g_l\|_C + Q \sum_{j=1}^n |g_l(y_j) - \int_D \psi_\eta(x - y_j) g_l(x) \,\mathsf{d}x| \\
&\qquad\qquad + |\int_D \psi_\eta(x - y_j)\big(g_l(x) - u(x)\big) \,\mathsf{d}x| \\
&\leq \frac{\epsilon}{3} + nQ\|u - g_l\|_C + nQ\frac{\epsilon}{3nQ} \\
&\quad + Q\|g_l - u\|_C \sum_{j=1}^n \int_D \psi_\gamma(x - y_j) \,\mathsf{d}x \\
&= \frac{2\epsilon}{3} + 2nQ\|u - g_l\|_C \\
&= \epsilon
\end{aligned}$$

where we use the fact that mollifiers are non-negative and integrate to one. □

Lemma 56. *Let $D \subset \mathbb{R}^d$ be a domain and $L \in \big(C^m(\bar{D})\big)^*$. For any compact set $K \subset C^m(\bar{D})$ and $\epsilon > 0$, there exist functions $\kappa_1, \dots, \kappa_J \in C_c^\infty(D)$ such that*

$$\sup_{u \in K} |L(u) - \sum_{j=1}^J \int_D \kappa_j \partial^{\alpha_j} u \,\mathsf{d}x| < \epsilon$$

where $\alpha_1, \dots, \alpha_J$ is an enumeration of the set $\{\alpha \in \mathbb{N}_0^d : 0 \leq |\alpha|_1 \leq m\}$.

Proof. By Lemma 54, we find distinct points $y_{11}, \dots, y_{1n_1}, \dots, y_{Jn_J} \in D$ and numbers $c_{11}, \dots, c_{Jn_J} \in \mathbb{R}$ such that

$$\sup_{u \in K} |L(u) - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{2}.$$

Applying the proof of Lemma 56 J times to each of the inner sums, we find functions $\kappa_1, \dots, \kappa_J \in C_c^\infty(D)$ such that

$$\max_{j \in \{1,\dots,J\}} | \int_D \kappa_j \partial^{\alpha_j} u \, \mathsf{d}x - \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \leq \frac{\epsilon}{2J}.$$

Then, for any $u \in K$,

$$\begin{aligned} |L(u) - \sum_{j=1}^{J} \int_D \kappa_j \partial^{\alpha_j} u \, \mathsf{d}x| \\ &\leq |L(u) - \sum_{j=1}^{J} \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \\ &\quad + \sum_{j=1}^{J} | \int_D \kappa_j \partial^{\alpha_j} u \, \mathsf{d}x - \sum_{k=1}^{n_j} c_{jk} \partial^{\alpha_j} u(y_{jk})| \\ &\leq \epsilon \end{aligned}$$

as desired. □

NO Approximation

The following lemmas show that the three pieces used in constructing the approximation from Lemma 47, which are schematically depicted in Figure 41, can all be approximated by NO(s). Lemma 57 shows that $F_J : \mathcal{A} \to \mathbb{R}^J$ can be approximated by an element of IO by mapping to a vector-valued constant function. Similarly, Lemma 59 shows that $G_{J'} : \mathbb{R}^{J'} \to \mathcal{U}$ can be approximated by an element of IO by mapping a vector-valued constant function to the coefficients of a basis expansion. Finally, Lemma 60 shows that NO(s) can exactly represent any standard neural network by viewing the inputs and outputs as vector-valued constant functions.

Lemma 57. *Let Assumption 34 hold. Let $\{c_j\}_{j=1}^n \subset \mathcal{A}^*$ for some $n \in \mathbb{N}$. Define the map $F : \mathcal{A} \to \mathbb{R}^n$ by*

$$F(a) = \big(c_1(a), \dots, c_n(a)\big), \qquad \forall a \in \mathcal{A}.$$

Then, for any compact set $K \subset \mathcal{A}$, $\sigma \in \mathsf{A}_0$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and neural network $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{n \times 1})$ such that

$$\sup_{a \in K} \sup_{y \in \bar{D}} |F(a) - \int_D \kappa(y,x) a(x) \, \mathsf{d}x|_1 \leq \epsilon.$$

Proof. Since K is bounded, there exists a number $M > 0$ such that

$$\sup_{a \in K} \|a\|_{\mathcal{A}} \leq M.$$

Define the constant

$$Q := \begin{cases} M, & \mathcal{A} = W^{m,p}(D) \\ M|D|, & A = C(\bar{D}) \end{cases}$$

and let $p = 1$ if $\mathcal{A} = C(\bar{D})$. By Lemma 53 and Lemma 55, there exist functions $f_1, \dots, f_n \in C_c^\infty(D)$ such that

$$\max_{j \in \{1,\dots,n\}} \sup_{a \in K} |c_j(a) - \int_D f_j a \, \mathsf{d}x| \leq \frac{\epsilon}{2n^{\frac{1}{p}}}.$$

Since $\sigma \in \mathsf{A}_0$, there exits some $L \in \mathbb{N}$ and neural networks $\psi_1, \dots, \psi_n \in \mathsf{N}_L(\sigma; \mathbb{R}^d)$ such that

$$\max_{j \in \{1,\dots,n\}} \|\psi_j - f_j\|_C \leq \frac{\epsilon}{2Qn^{\frac{1}{p}}}.$$

By setting all weights associated to the first argument to zero, we can modify each neural network ψ_j to a neural network $\psi_j \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d)$ so that

$$\psi_j(y,x) = \psi_j(x) \mathbb{1}(y), \qquad \forall y, x \in \mathbb{R}^d.$$

Define $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{n \times 1})$ by

$$\kappa(y,x) = [\psi_1(y,x), \dots, \psi_n(y,x)]^T.$$

Then for any $a \in K$ and $y \in \bar{D}$, we have

$$\begin{aligned} |F(a) - \int_D \kappa(y,x) a \, \mathsf{d}x|_p^p &= \sum_{j=1}^n |c_j(a) - \int_D \mathbb{1}(y) \psi_j(x) a(x) \, \mathsf{d}x|^p \\ &\leq 2^{p-1} \sum_{j=1}^n |c_j(a) - \int_D f_j a \, \mathsf{d}x|^p + |\int_D (f_j - \psi_j) a \, \mathsf{d}x|^p \\ &\leq \frac{\epsilon^p}{2} + 2^{p-1} n Q^p \|f_j - \psi_j\|_C^p \\ &\leq \epsilon^p \end{aligned}$$

and the result follows by finite dimensional norm equivalence. □

Lemma 58. *Suppose $D \subset \mathbb{R}^d$ is a domain and let $\{c_j\}_{j=1}^n \subset \big(C^m(\bar{D})\big)^*$ for some $m, n \in \mathbb{N}$. Define the map $F : \mathcal{A} \to \mathbb{R}^n$ by*

$$F(a) = \big(c_1(a), \dots, c_n(a)\big), \qquad \forall a \in C^m(\bar{D}).$$

Then, for any compact set $K \subset C^m(\bar{D})$, $\sigma \in \mathsf{A}_0$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and neural network $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^d \times \mathbb{R}^d, \mathbb{R}^{n \times J})$ such that

$$\sup_{a \in K} \sup_{y \in \bar{D}} |F(a) - \int_D \kappa(y,x)\big(\partial^{\alpha_1} a(x), \dots, \partial^{\alpha_J} a(x)\big) \, \mathsf{d}x|_1 \leq \epsilon$$

where $\alpha_1, \dots, \alpha_J$ is an enumeration of the set $\{\alpha \in \mathbb{N}^d : 0 \leq |\alpha|_1 \leq m\}$.

Proof. The proof follows as in Lemma 57 by replacing the use of Lemmas 53 and 55 by Lemma 56. □

Lemma 59. *Let Assumption 35 hold. Let $\{\varphi_j\}_{j=1}^n \subset \mathcal{U}$ for some $n \in \mathbb{N}$. Define the map $G : \mathbb{R}^n \to \mathcal{U}$ by*

$$G(w) = \sum_{j=1}^n w_j \varphi_j, \qquad \forall w \in \mathbb{R}^n.$$

Then, for any compact set $K \subset \mathbb{R}^n$, $\sigma \in \mathsf{A}_{m_2}$, and $\epsilon > 0$, there exists a number $L \in \mathbb{N}$ and a neural network $\kappa \in \mathsf{N}_L(\sigma; \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \mathbb{R}^{1 \times n})$ such that

$$\sup_{w \in K} \|G(w) - \int_{D'} \kappa(\cdot, x) w \mathbb{1}(x) \, \mathsf{d}x\|_{\mathcal{U}} \leq \epsilon.$$

Proof. Since $K \subset \mathbb{R}^n$ is compact, there is a number $M > 1$ such that

$$\sup_{w \in K} |w|_1 \leq M.$$

If $\mathcal{U} = L^{p_2}(D')$, then density of $C_c^\infty(D')$ implies there are functions $\tilde{\psi}_1, \dots, \tilde{\psi}_n \in C^\infty(\bar{D}')$ such that

$$\max_{j \in \{1,\dots,n\}} \|\varphi_j - \tilde{\psi}_j\|_{\mathcal{U}} \leq \frac{\epsilon}{2nM}.$$

Similarly if $U = W^{m_2,p_2}(D')$, then density of the restriction of functions in $C_c^\infty(\mathbb{R}^{d'})$ to D' Leoni, 2009, Theorem 11.35 implies the same result. If $\mathcal{U} = C^{m_2}(\bar{D}')$ then we set $\tilde{\psi}_j = \varphi_j$ for any $j \in \{1, \dots, n\}$. Define $\tilde{\kappa} : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \to \mathbb{R}^{1 \times n}$ by

$$\tilde{\kappa}(y,x) = \frac{1}{|D'|}[\tilde{\psi}_1(y), \dots, \tilde{\psi}_n(y)].$$

Then, for any $w \in K$,

$$\begin{aligned}\|G(w) - \int_{D'} \tilde{\kappa}(\cdot, x) w \mathbb{1}(x) \, \mathsf{d}x\|_{\mathcal{U}} &= \|\sum_{j=1}^{n} w_j \varphi_j - \sum_{j=1}^{n} w_j \tilde{\psi}_j\|_{\mathcal{U}} \\ &\leq \sum_{j=1}^{n} |w_j| \|\varphi_j - \tilde{\psi}_j\|_{\mathcal{U}} \\ &\leq \frac{\epsilon}{2}.\end{aligned}$$

Since $\sigma \in \mathsf{A}_{m_2}$, there exists neural networks $\psi_1, \dots, \psi_n \in \mathsf{N}_1(\sigma; \mathbb{R}^{d'})$ such that

$$\max_{j \in \{1,\dots,n\}} \|\tilde{\psi}_j - \psi_j\|_{C^{m_2}} \leq \frac{\epsilon}{2nM(J|D'|)^{\frac{1}{p_2}}}$$

where, if $\mathcal{U} = C^{m_2}(\bar{D}')$, we set $J = 1/|D'|$ and $p_2 = 1$, and otherwise $J = |\{\alpha \in \mathbb{N}^d : |\alpha|_1 \leq m_2\}|$. By setting all weights associated to the second argument to zero, we can modify each neural network ψ_j to a neural network $\psi_j \in \mathsf{N}_1(\sigma; \mathbb{R}^{d'} \times \mathbb{R}^{d'})$ so that

$$\psi_j(y, x) = \psi_j(y) \mathbb{1}(x), \qquad \forall y, x \in \mathbb{R}^{d'}.$$

Define $\kappa \in \mathsf{N}_1(\sigma; \mathbb{R}^{d'} \times \mathbb{R}^{d'}, \mathbb{R}^{1 \times n})$ as

$$\kappa(y, x) = \frac{1}{|D'|} [\psi_1(y, x), \dots, \psi_n(y, x)].$$

Then, for any $w \in \mathbb{R}^n$,

$$\int_{D'} \kappa(y, x) w \mathbb{1}(x) \, \mathsf{d}x = \sum_{j=1}^{n} w_j \psi_j(y).$$

We compute that, for any $j \in \{1, \dots, n\}$,

$$\|\psi_j - \tilde{\psi}_j\|_{\mathcal{U}} \leq \begin{cases} |D'|^{\frac{1}{p_2}} \|\psi_j - \tilde{\psi}_j\|_{C^{m_2}}, & \mathcal{U} = L^{p_2}(D') \\ (J|D'|)^{\frac{1}{p_2}} \|\psi_j - \tilde{\psi}_j\|_{C^{m_2}}, & \mathcal{U} = W^{m_2,p_2}(D') \\ \|\psi_j - \tilde{\psi}_j\|_{C^{m_2}}, & \mathcal{U} = C^{m_2}(\bar{D}') \end{cases}$$

hence, for any $w \in K$,

$$\|\int_{D'} \kappa(y, x) w \mathbb{1}(x) \, \mathsf{d}x - \sum_{j=1}^{n} w_j \tilde{\psi}_j\|_{\mathcal{U}} \leq \sum_{j=1}^{n} |w_j| \|\psi_j - \tilde{\psi}_j\|_{\mathcal{U}} \leq \frac{\epsilon}{2}.$$

By triangle inequality, for any $w \in K$, we have

$$\begin{aligned}\|G(w) - \int_D \kappa(\cdot,x)w\mathbb{1}(x)\,\mathsf{d}x\|_{\mathcal{U}} &\leq \|G(w) - \int_D \tilde{\kappa}(\cdot,x)w\mathbb{1}(x)\,\mathsf{d}x\|_{\mathcal{U}} \\ &\quad + \|\int_D \tilde{\kappa}(\cdot,x)w\mathbb{1}(x)\,\mathsf{d}x - \int_D \kappa(\cdot,x)w\mathbb{1}(x)\,\mathsf{d}x\|_{\mathcal{U}} \\ &\leq \frac{\epsilon}{2} + \|\int_D \kappa(\cdot,x)w\mathbb{1}(x) - \sum_{j=1}^n w_j\tilde{\psi}_n\|_{\mathcal{U}} \\ &\leq \epsilon\end{aligned}$$

as desired. □

Lemma 60. *Let $N, d, d', p, q \in \mathbb{N}$, $m, n \in \mathbb{N}_0$, $D \subset \mathbb{R}^p$ and $D' \subset \mathbb{R}^q$ be domains and $\sigma_1 \in \mathsf{A}^{\mathsf{L}}_m$. For any $\varphi \in \mathsf{N}_N(\sigma_1;\mathbb{R}^d,\mathbb{R}^{d'})$ and $\sigma_2,\sigma_3 \in \mathsf{A}_n$, there exists a $G \in \mathsf{NO}_N(\sigma_1,\sigma_2,\sigma_3;D,D',\mathbb{R}^d,\mathbb{R}^{d'})$ such that*

$$\varphi(w) = G(w\mathbb{1})(x), \qquad \forall w \in \mathbb{R}^d,\ \forall x \in D'.$$

.

Proof. We have that

$$\varphi(x) = W_N\sigma_1(\dots W_1\sigma_1(W_0x + b_0) + b_1 \dots) + b_N, \qquad \forall x \in \mathbb{R}^d$$

where $W_0 \in \mathbb{R}^{d_0\times d}, W_1 \in \mathbb{R}^{d_1\times d_0}, \dots, W_N \in \mathbb{R}^{d'\times d_{N-1}}$ and $b_0 \in \mathbb{R}^{d_0}, b_1 \in \mathbb{R}^{d_1}, \dots, b_N \in \mathbb{R}^{d'}$ for some $d_0,\dots,d_{N-1} \in \mathbb{N}$. By setting all parameters to zero except for the last bias term, we can find $\kappa^{(0)} \in \mathsf{N}_1(\sigma_2;\mathbb{R}^p\times\mathbb{R}^p,\mathbb{R}^{d_0\times d})$ such that

$$\kappa_0(x,y) = \frac{1}{|D|}W_0, \qquad \forall x,y \in \mathbb{R}^p.$$

Similarly, we can find $\tilde{b}_0 \in \mathsf{N}_1(\sigma_2;\mathbb{R}^p,\mathbb{R}^{d_0})$ such that

$$\tilde{b}_0(x) = b_0, \qquad \forall x \in \mathbb{R}^p.$$

Then

$$\int_D \kappa_0(y,x)w\mathbb{1}(x)\,\mathsf{d}x + \tilde{b}(y) = (W_0w + b_0)\mathbb{1}(y), \qquad \forall w \in \mathbb{R}^d,\ \forall y \in D.$$

Continuing a similar construction for all layers clearly yields the result. □

References

Aaronson, J. (1997). *An Introduction to Infinite Ergodic Theory*. Mathematical surveys and monographs. American Mathematical Society. ISBN: 9780821804940.

Adams, R. A. and J. J. Fournier (2003). *Sobolev Spaces*. Elsevier Science.

Adler, Jonas and Ozan Oktem (Nov. 2017). “Solving ill-posed inverse problems using iterative deep neural networks”. In: *Inverse Problems*. DOI: 10.1088/1361-6420/aa9581.

Albiac, Fernando and Nigel J. Kalton (2006). *Topics in Banach space theory*. 1st ed. Graduate Texts in Mathematics. Springer.

Alet, Ferran et al. (2019). “Graph Element Networks: adaptive, structured computation and memory”. In: *36th International Conference on Machine Learning*. PMLR. URL: http://proceedings.mlr.press/v97/alet19a.html.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). “Layer normalization”. In: *arXiv preprint arXiv:1607.06450*.

Bach, Francis (2013). “Sharp analysis of low-rank kernel matrix approximations”. In: *Conference on Learning Theory*, pp. 185–209.

Bar, Leah and Nir Sochen (2019). “Unsupervised deep learning algorithm for PDE-based forward and inverse problems”. In: *arXiv preprint arXiv:1904.05417*.

Battaglia, Peter W et al. (2018). “Relational inductive biases, deep learning, and graph networks”. In: *arXiv preprint arXiv:1806.01261*.

Bear, Jacob and M Yavuz Corapcioglu (2012). *Fundamentals of transport phenomena in porous media*. Springer Science & Business Media.

Belongie, Serge et al. (2002). “Spectral partitioning with indefinite kernels using the Nyström extension”. In: *European conference on computer vision*. Springer.

Bengio, Yoshua, Yann LeCun, et al. (2007). “Scaling learning algorithms towards AI”. In: *Large-scale kernel machines* 34.5, pp. 1–41.

Bhatnagar, Saakaar et al. (2019). “Prediction of aerodynamic flow fields using convolutional neural networks”. In: *Computational Mechanics*, pp. 1–21. DOI: 10.1007/s00466-019-01740-0.

Bhattacharya, Kaushik et al. (2020). “Model reduction and neural networks for parametric PDEs”. In: *arXiv preprint arXiv:2005.03180*.

Bogachev, V. I. (2007). *Measure Theory*. Vol. 2. Springer-Verlag Berlin Heidelberg.

Bonito, Andrea et al. (2020). “Nonlinear methods for model reduction”. In: *arXiv preprint arXiv:2005.02565*.

Börm, Steffen, Lars Grasedyck, and Wolfgang Hackbusch (2003). “Hierarchical matrices”. In: *Lecture notes* 21, p. 2003.

Box, George EP (1976). “Science and statistics”. In: *Journal of the American Statistical Association* 71.356, pp. 791–799.

Boyd, John P (2001). *Chebyshev and Fourier spectral methods*. Courier Corporation.

Brown, Tom B et al. (2020). “Language models are few-shot learners”. In: *arXiv preprint arXiv:2005.14165*.

Brudnyi, Alexander and Yuri Brudnyi (2012). *Methods of Geometric Analysis in Extension and Trace Problems*. Vol. 1. Birkhäuser Basel.

Bruno, Oscar P, Youngae Han, and Matthew M Pohlman (2007). “Accurate, high-order representation of complex three-dimensional surfaces via Fourier continuation analysis”. In: *Journal of computational Physics* 227.2, pp. 1094–1125.

Chandler, Gary J. and Rich R. Kerswell (2013). “Invariant recurrent solutions embedded in a turbulent two-dimensional Kolmogorov flow”. In: *Journal of Fluid Mechanics* 722, pp. 554–595.

Chen, Chi et al. (2019). “Graph networks as a universal machine learning framework for molecules and crystals”. In: *Chemistry of Materials* 31.9, pp. 3564–3572.

Chen, Tianping and Hong Chen (1995). “Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems”. In: *IEEE Transactions on Neural Networks* 6.4, pp. 911–917.

Choromanski, Krzysztof et al. (2020). “Rethinking attention with performers”. In: *arXiv preprint arXiv:2009.14794*.

Ciesielski, Z. and J. Domsta (1972). “Construction of an Orthonormal Basis in Cm(Id) and Wmp(Id)”. In: *Studia Mathematica* 41, pp. 211–224.

Cohen, Albert and Ronald DeVore (2015). “Approximation of high-dimensional parametric PDEs”. In: *Acta Numerica*. DOI: `10.1017/S0962492915000033`.

Cohen, Albert, Ronald Devore, et al. (2020). “Optimal Stable Nonlinear Approximation”. In: *arXiv preprint arXiv:2009.09907*.

Constantin, Peter and Ciprian Foias (1988). *Navier-stokes equations*. University of Chicago Press.

Conway, J. B. (2007). *A Course in Functional Analysis*. Springer-Verlag New York.

Cotter, S. L. et al. (Aug. 2013). “MCMC Methods for Functions: Modifying Old Algorithms to Make Them Faster”. In: *Statistical Science* 28.3, pp. 424–446. ISSN: 0883-4237. DOI: `10.1214/13-sts421`. URL: `http://dx.doi.org/10.1214/13-STS421`.

Cotter, Simon L et al. (2009). “Bayesian inverse problems for functions and applications to fluid mechanics”. In: *Inverse problems* 25.11, p. 115008.

Damianou, Andreas and Neil Lawrence (2013). “Deep gaussian processes”. In: *Artificial Intelligence and Statistics*, pp. 207–215.

Devlin, Jacob et al. (2018). “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805*.

DeVore, Ronald A. (1998). “Nonlinear approximation”. In: *Acta Numerica* 7, pp. 51–150.

– (2014). “The Theoretical Foundation of Reduced Basis Methods”. In: *Model Reduction and Approximation*. SIAM, Philadelphia. DOI: 10.1137/1.9781611974829.ch3.

Dosovitskiy, Alexey et al. (2020). “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929*.

Dudley, R. and Rimas Norvaisa (Jan. 2011). *Concrete Functional Calculus*. Vol. 149. ISBN: 978-1-4419-6949-1.

Dudley, R.M. and R. Norvaiša (2010). *Concrete Functional Calculus*. Springer Monographs in Mathematics. Springer New York.

Dugundji, J. (1951). “An extension of Tietze’s theorem”. In: *Pacific Journal of Mathematics* 1.3, pp. 353–367.

Dunlop, Matthew M et al. (2018). “How deep are deep Gaussian processes?” In: *The Journal of Machine Learning Research* 19.1, pp. 2100–2145.

E, Weinan (2011). *Principles of Multiscale Modeling*. Cambridge: Cambridge University Press.

E, Weinan and Bing Yu (Mar. 2018). “The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems”. English (US). In: *Communications in Mathematics and Statistics*. ISSN: 2194-6701. DOI: 10.1007/s40304-018-0127-z.

Evans, Lawrence C (2010). *Partial Differential Equations*. Vol. 19. American Mathematical Soc.

Fan, Yuwei, Cindy Orozco Bohorquez, and Lexing Ying (2019). “BCR-Net: A neural network based on the nonstandard wavelet form”. In: *Journal of Computational Physics* 384, pp. 1–15.

Fan, Yuwei, Jordi Feliu-Faba, et al. (2019). “A multiscale neural network based on hierarchical nested bases”. In: *Research in the Mathematical Sciences* 6.2, p. 21.

Fan, Yuwei, Lin Lin, et al. (2019). “A multiscale neural network based on hierarchical matrices”. In: *Multiscale Modeling & Simulation* 17.4, pp. 1189–1213.

Fefferman, Charles (2007). “Cm extension by linear operators”. In: *Annals of Mathematics* 166, pp. 779–835.

Gardner, Jacob R et al. (2018). “Product kernel interpolation for scalable Gaussian processes”. In: *arXiv preprint arXiv:1802.08903*.

Garriga-Alonso, Adrià, Carl Edward Rasmussen, and Laurence Aitchison (Aug. 2018). “Deep Convolutional Networks as shallow Gaussian Processes”. In: *arXiv e-prints*, arXiv:1808.05587, arXiv:1808.05587. arXiv: `1808.05587 [stat.ML]`.

Gilmer, Justin et al. (2017). “Neural message passing for quantum chemistry”. In: *Proceedings of the 34th International Conference on Machine Learning*. URL: `http://proceedings.mlr.press/v70/gilmer17a.html`.

Globerson, Amir and Roi Livni (2016). “Learning Infinite-Layer Networks: Beyond the Kernel Trick”. In: *CoRR* abs/1606.05316. arXiv: `1606.05316`. URL: `http://arxiv.org/abs/1606.05316`.

Greenfeld, Daniel et al. (2019). “Learning to optimize multigrid PDE solvers”. In: *International Conference on Machine Learning*. PMLR, pp. 2415–2423.

Greengard, Leslie and Vladimir Rokhlin (1997). “A new version of the fast multipole method for the Laplace equation in three dimensions”. In: *Acta numerica* 6, pp. 229–269.

Grothendieck, A (1955). *Produits tensoriels topologiques et espaces nucléaires*. Vol. 16. American Mathematical Society Providence.

Guo, Xiaoxiao, Wei Li, and Francesco Iorio (2016). “Convolutional neural networks for steady flow approximation”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Gurtin, Morton E (1982). *An introduction to continuum mechanics*. Academic press.

Guss, William H. (Dec. 2016). “Deep Function Machines: Generalized Neural Networks for Topological Layer Expression”. In: *arXiv e-prints*, arXiv:1612.04799, arXiv:1612.04799. arXiv: `1612.04799 [stat.ML]`.

Haber, Eldad and Lars Ruthotto (2017). “Stable architectures for deep neural networks”. In: *Inverse Problems* 34.1, p. 014004.

Hamilton, Will, Zhitao Ying, and Jure Leskovec (2017). “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems*, pp. 1024–1034.

He, Juncai and Jinchao Xu (2019). “MgNet: A unified framework of multigrid and convolutional neural network”. In: *Science china mathematics* 62.7, pp. 1331–1354.

He, Kaiming et al. (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Herrmann, L, Ch Schwab, and J Zech (2020). “Deep ReLU Neural Network Expression Rates for Data-to-QoI Maps in Bayesian PDE Inversion”. URL: `https://www.sam.math.ethz.ch/sam_reports/reports_final/reports2020/2020-02.pdf`.

Hornik, Kurt, Maxwell Stinchcombe, Halbert White, et al. (1989). “Multilayer feedforward networks are universal approximators.” In: *Neural networks* 2.5, pp. 359–366.

Jiang, Chiyu Max et al. (2020). “MeshfreeFlowNet: A Physics-Constrained Deep Continuous Space-Time Super-Resolution Framework”. In: *arXiv preprint arXiv:2005.01463*.

Johnson, Claes (2012). *Numerical solution of partial differential equations by the finite element method*. Courier Corporation.

Kashinath, Karthik, Philip Marcus, et al. (2020). “Enforcing Physical Constraints in CNNs through Differentiable PDE Layer”. In: *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*.

Khoo, Yuehaw, Jianfeng Lu, and Lexing Ying (2017). “Solving parametric PDE problems with artificial neural networks”. In: *arXiv preprint arXiv:1707.03351*.

Khoo, Yuehaw and Lexing Ying (2019). “SwitchNet: a neural network model for forward and inverse scattering problems”. In: *SIAM Journal on Scientific Computing* 41.5, A3182–A3201.

Kipf, Thomas N and Max Welling (2016). “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907*.

Kondor, Risi, Nedelina Teneva, and Vikas Garg (2014). “Multiresolution matrix factorization”. In: *International Conference on Machine Learning*, pp. 1620–1628.

Kovachki, Nikola, Samuel Lanthaler, and Siddhartha Mishra (2021). “On universal approximation and error bounds for Fourier Neural Operators”. In: *arXiv preprint arXiv:2107.07562*.

Kraichnan, Robert H. (1967). “Inertial Ranges in Two-Dimensional Turbulence”. In: *The Physics of Fluids* 10.7, pp. 1417–1423.

Kulis, Brian, Mátyás Sustik, and Inderjit Dhillon (2006). “Learning low-rank kernel matrices”. In: *Proceedings of the 23rd international conference on Machine learning*, pp. 505–512.

Lan, Liang et al. (2017). “Low-rank decomposition meets kernel learning: A generalized Nyström method”. In: *Artificial Intelligence* 250, pp. 1–15.

Lanthaler, Samuel, Siddhartha Mishra, and George Em Karniadakis (2021). “Error estimates for DeepOnets: A deep learning framework in infinite dimensions”. In: *arXiv preprint arXiv:2102.09618*.

Lemarié-Rieusset, Pierre Gilles (2018). *The Navier-Stokes problem in the 21st century*. CRC Press.

Leoni, G. (2009). *A First Course in Sobolev Spaces*. Graduate studies in mathematics. American Mathematical Soc.

Li, Zongyi, Nikola Kovachki, et al. (2020a). *Fourier Neural Operator for Parametric Partial Differential Equations*. arXiv: `2010.08895 [cs.LG]`.

Li, Zongyi, Nikola Kovachki, et al. (2020b). *Multipole Graph Neural Operator for Parametric Partial Differential Equations*. arXiv: `2006.09535 [cs.LG]`.

– (2020c). “Neural operator: Graph kernel network for partial differential equations”. In: *arXiv preprint arXiv:2003.03485*.

Li, Zongyi, Hongkai Zheng, et al. (2021). “Physics-Informed Neural Operator for Learning Partial Differential Equations”. In: *arXiv preprint arXiv:2111.03794*.

Lu, Lu, Pengzhan Jin, and George Em Karniadakis (2019). “DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators”. In: *arXiv preprint arXiv:1910.03193*.

Lu, Lu, Pengzhan Jin, Guofei Pang, et al. (2021). “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators”. In: *Nature Machine Intelligence* 3.3, pp. 218–229.

Lu, Lu, Xuhui Meng, et al. (2021). “A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data”. In: *arXiv preprint arXiv:2111.05512*.

Mathieu, Michael, Mikael Henaff, and Yann LeCun (2013). *Fast Training of Convolutional Networks through FFTs*. arXiv: `1312.5851 [cs.CV]`.

Matthews, Alexander G. de G. et al. (Apr. 2018). “Gaussian Process Behaviour in Wide Deep Neural Networks”. In:

Mingo, Luis et al. (2004). “Fourier neural networks: An approach with sinusoidal activation functions”. In:

Murphy, Ryan L et al. (2018). “Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs”. In: *arXiv preprint arXiv:1811.01900*.

Neal, Radford M. (1996). *Bayesian Learning for Neural Networks*. Springer-Verlag. ISBN: 0387947248.

Nelsen, Nicholas H and Andrew M Stuart (2021). “The random feature model for input-output maps between banach spaces”. In: *SIAM Journal on Scientific Computing* 43.5, A3212–A3243.

Nyström, Evert J (1930). “Über die praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben”. In: *Acta Mathematica*.

O’Leary-Roseberry, Thomas et al. (2020). “Derivative-Informed Projected Neural Networks for High-Dimensional Parametric Maps Governed by PDEs”. In: *arXiv preprint arXiv:2011.15110*.

Opschoor, Joost A.A., Christoph Schwab, and Jakob Zech (2020). “Deep learning in high dimension: ReLU network Expression Rates for Bayesian PDE inversion”. In: *SAM Research Report* 2020-47.

Pan, Shaowu and Karthik Duraisamy (2020). “Physics-informed probabilistic learning of linear embeddings of nonlinear dynamics with guaranteed stability”. In: *SIAM Journal on Applied Dynamical Systems* 19.1, pp. 480–509.

Patel, Ravi G et al. (2021). “A physics-informed operator regression framework for extracting data-driven continuum models”. In: *Computer Methods in Applied Mechanics and Engineering* 373, p. 113500.

Pathak, Jaideep et al. (2020). *Using Machine Learning to Augment Coarse-Grid Computational Fluid Dynamics Simulations*. arXiv: 2010.00072 [physics.comp-ph].

Pełczyński, Aleksander and Michał Wojciechowski (2001). “Contribution to the isomorphic classification of Sobolev spaces Lpk(Omega)”. In: *Recent Progress in Functional Analysis* 189, pp. 133–142.

Pfaff, Tobias et al. (2020). *Learning Mesh-Based Simulation with Graph Networks*. arXiv: 2010.03409 [cs.LG].

Pinkus, A. (1985). *N-Widths in Approximation Theory*. Springer-Verlag Berlin Heidelberg.

Pinkus, Allan (1999). “Approximation theory of the MLP model in neural networks”. In: *Acta Numerica* 8, pp. 143–195.

Poole, Ben et al. (2016). “Exponential expressivity in deep neural networks through transient chaos”. In: *Advances in neural information processing systems* 29, pp. 3360–3368.

Quiñonero-Candela, Joaquin and Carl Edward Rasmussen (2005). “A Unifying View of Sparse Approximate Gaussian Process Regression”. In: *J. Mach. Learn. Res.* 6, pp. 1939–1959.

Rahimi, Ali and Benjamin Recht (2008). “Uniform approximation of functions with random bases”. In: *2008 46th Annual Allerton Conference on Communication, Control, and Computing*. IEEE, pp. 555–561.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378, pp. 686–707. DOI: 10.1016/j.jcp.2018.10.045.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer, pp. 234–241.

Roux, Nicolas Le and Yoshua Bengio (2007). “Continuous Neural Networks”. In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. Ed. by Marina Meila and Xiaotong Shen.

Schwab, Christoph and Jakob Zech (2019). “Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ”. In: *Analysis and Applications* 17.01, pp. 19–55. DOI: 10.1142/S0219530518500203.

Sirignano, Justin and Konstantinos Spiliopoulos (2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of computational physics* 375, pp. 1339–1364.

Sitzmann, Vincent et al. (2020). “Implicit neural representations with periodic activation functions”. In: *arXiv preprint arXiv:2006.09661*.

Smith, Jonathan D, Kamyar Azizzadenesheli, and Zachary E Ross (2020). “EikoNet: Solving the Eikonal equation with Deep Neural Networks”. In: *arXiv preprint arXiv:2004.00361*.

Stein, Elias M. (1970). *Singular Integrals and Differentiability Properties of Functions*. Princeton University Press.

Strogatz, Steven (2009). “Loves Me, Loves Me Not (Do the Math)”. In:

Stuart, A. M. (2010). “Inverse problems: A Bayesian perspective”. In: *Acta Numerica* 19, pp. 451–559.

Temam, Roger (2001). *Navier-Stokes equations: theory and numerical analysis*. Vol. 343. American Mathematical Soc.

Trefethen, Lloyd N (2000). *Spectral methods in MATLAB*. Vol. 10. Siam.

Trillos, Nicolas Garcia, Moritz Gerlach, et al. (2020). “Error estimates for spectral convergence of the graph Laplacian on random geometric graphs toward the Laplace–Beltrami operator”. In: *Foundations of Computational Mathematics* 20.4, pp. 827–887.

Trillos, Nicolas Garcia and Dejan Slepvcev (2018). “A variational approach to the consistency of spectral clustering”. In: *Applied and Computational Harmonic Analysis* 45.2, pp. 239–281.

Um, Kiwon, Philipp Holl, et al. (2020). “Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers”. In: *arXiv preprint arXiv:2007.00016*.

Um, Kiwon, Raymond, et al. (2020). *Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers*. arXiv: `2007.00016 [physics.comp-ph]`.

Ummenhofer, Benjamin et al. (2020). “Lagrangian fluid simulation with continuous convolutions”. In: *International Conference on Learning Representations*.

Vapnik, Vladimir N. (1998). *Statistical Learning Theory*. Wiley-Interscience.

Vaswani, Ashish et al. (2017). “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc.

Veličković, Petar et al. (2017). “Graph attention networks”. In:

Von Luxburg, Ulrike, Mikhail Belkin, and Olivier Bousquet (2008). “Consistency of spectral clustering”. In: *The Annals of Statistics*, pp. 555–586.

Wang, Rui et al. (2020). “Towards physics-informed deep learning for turbulent flow prediction”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1457–1466.

Wang, Sifan, Hanwen Wang, and Paris Perdikaris (2021). “Learning the solution operator of parametric partial differential equations with physics-informed Deep-Onets”. In: *arXiv preprint arXiv:2103.10974*.

Weinan, E (2017). “A proposal on machine learning via dynamical systems”. In: *Communications in Mathematics and Statistics* 5.1, pp. 1–11.

Wen, Gege et al. (2021). “U-FNO–an enhanced Fourier neural operator based-deep learning model for multiphase flow”. In: *arXiv preprint arXiv:2109.03697*.

Whitney, Hassler (1934). “Functions Differentiable on the Boundaries of Regions”. In: *Annals of Mathematics* 35.3, pp. 482–485.

Williams, Christopher K. I. (1996). “Computing with Infinite Networks”. In: *Proceedings of the 9th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press.

Wu, Kailiang and Dongbin Xiu (2020). “Data-driven deep learning of partial differential equations in modal space”. In: *Journal of Computational Physics* 408, p. 109307.

Zhu, Yinhao and Nicholas Zabaras (2018). “Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification”. In: *Journal of Computational Physics*. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2018.04.018`. URL: `http://www.sciencedirect.com/science/article/pii/S0021999118302341`.

Chapter 6

CONDITIONAL SAMPLING WITH MONOTONE GAN(S)

6.1 Introduction

Consider inputs $\boldsymbol{x} \in \mathbb{R}^n$ and outputs $\boldsymbol{y} \in \mathbb{R}^m$ distributed according to a joint probability measure $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$. We introduce a method called *monotone generative adversarial networks* (MGANs) to sample the conditional measure $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ given any input $\boldsymbol{x}^* \in \mathbb{R}^n$, by constructing a map $\mathsf{F}(\boldsymbol{x}^*, \cdot)$ that pushes forward a reference measure $\eta(\mathrm{d}\boldsymbol{y})$ to $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$.

Conditional sampling is a fundamental task for machine learning and statistics as it provides a way of quantifying uncertainty in predicted outputs. Standard supervised learning (SL) algorithms approximate the conditional expectation of $\boldsymbol{y}$ given $\boldsymbol{x}$, for instance via regression Hastie, Tibshirani, and Friedman, 2009, but do not fully characterize the uncertainty in the output. Probabilistic methods, in particular Bayesian techniques, improve on this by characterizing the distribution of $\boldsymbol{y}|\boldsymbol{x}$. To do so, most Bayesian approaches require prior information about the process generating $\boldsymbol{x}$ and $\boldsymbol{y}$, such as a (parametric) model for the distribution of $\boldsymbol{y}|\boldsymbol{x}$. These prior assumptions are often impossible to verify in practice, and hence model selection techniques are often employed to choose the "best" model within a specified class.

In contrast, recent unsupervised learning methods such as generative adversarial networks (GANs) Goodfellow et al., 2014; Nowozin, Cseke, and Tomioka, 2016; Arjovsky, Chintala, and Bottou, 2017; Gui et al., 2020, normalizing flows (NFs) Kobyzev, Prince, and Brubaker, 2020; Papamakarios, Nalisnick, et al., 2019 and variational autoencoders (VAEs) Doersch, 2016 have been remarkably successful at sampling complex and high-dimensional probability distributions under minimal assumptions on the underlying models for the data. Put simply, these generative methods train a map T that pushes forward a reference measure η to the target distribution ν. (In some cases, such as VAEs and GANs, the measure η is usually supported on a lower-dimensional space than ν.) The map T is often parameterized with a neural network and is trained using a data set consisting of samples from ν, without any explicit assumptions on the relationship between $\boldsymbol{y}$ and $\boldsymbol{x}$. Generative models are typically not designed with conditional sampling in mind, however, and to our knowledge—with a few recent exceptions (e.g., Lindgren, Whang, and Dimakis (2020), as discussed in Section 6.3)—it is not possible to utilize the map T constructed by GANs or VAEs to *provably* sample the conditionals $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ from the joint measure.

Here we propose a method that bridges the gap between SL and generative modeling. We train a map T without any model assumptions between $\boldsymbol{y}$ and $\boldsymbol{x}$, but we enforce sufficient constraints that allow us to extract another map F from T so that $\mathsf{F}(\boldsymbol{x}^*, \cdot)$ can be used to sample exactly from $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ for any $\boldsymbol{x}^* \in \mathbb{R}^n$. This idea has been explored in the uncertainty quantification (UQ) and inverse problems communities Y. Marzouk et al., 2016; El Moselhy and Y. M. Marzouk, 2012; Spantini, Baptista, and Y. Marzouk, 2019; Siahkoohi et al., 2021 for Bayesian inference and within machine learning Papamakarios, Pavlakou, and Murray, 2017; Jaini, Selby, and Yu, 2019; Brennan et al., 2020, by constructing monotone triangular maps T that approximate the well-known triangular Knothe–Rosenblatt (KR) rearrangement Santambrogio, 2015. By construction, the components of the KR map push forward the reference conditionals to the target conditionals, which is precisely what is desired for conditional sampling. Working with the KR map can be restrictive, however, since we must first choose a variable ordering and then employ a specific parameterization that ensures the map is triangular and monotone. Here, we relax the triangularity assumption to a *block* triangularity assumption and prove that this relaxed constraint is sufficient for correct conditional sampling. We then impose block triangularity and strict monotonicity of T as constraints in the GAN framework, to learn a map T from which the desired map F can be extracted easily, hence the name MGAN.

We summarize our main contributions as follows:

- We generalize triangular maps for conditional sampling by proving that block-triangular structure is sufficient to guarantee the exactness of conditionals.
- We propose an adversarial training procedure (MGAN) for learning correct conditional generative models.
- We compare the performance of MGANs to that of several approximate conditional sampling procedures, and show that MGANs yield more accurate conditional distributions in practice.
- We demonstrate the efficacy of MGANs on large-scale Bayesian inference and image in-painting examples.

6.2 Conditional sampling with block triangular maps

We now outline our approach for constructing block triangular maps for conditional sampling, beginning with the theoretical foundations of our framework followed by the details of our training and sampling strategies.

Guarantees for conditional sampling and density estimation

Following the notation of Section 6.1, we consider input and output pairs $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} \in \mathbb{R}^m$ and define their concatenation $\boldsymbol{z} \equiv (\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}^d$ with $d = n + m$. We then consider the measure transport problem (Villani, 2009) of pushing the *reference measure* $\eta(\mathrm{d}\boldsymbol{z})$ to the joint *target measure* $\nu(\mathrm{d}\boldsymbol{z}) \equiv \nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$ by finding a map $\mathsf{T} : \mathbb{R}^d \to \mathbb{R}^d$ so that $\mathsf{T}_\sharp \eta = \nu$. For brevity, we henceforth assume that $\nu(\mathrm{d}\boldsymbol{z})$ is absolutely continuous with respect to the Lebesgue measure on $\mathbb{R}^d$ with full support, i.e., its Lebesgue density is strictly positive. Also, we take the reference measure η to be of the form $\eta(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y}) = \nu(\mathrm{d}\boldsymbol{x}) \otimes N(0, I_m)$ where I_m denotes the $m \times m$ identity matrix and $\nu(\mathrm{d}\boldsymbol{x})$ denotes the marginal of ν on the $\boldsymbol{x}$ variable.

We require the map T to be of a particular *block triangular* form, that is,

$$\mathsf{T}(\boldsymbol{x}, \boldsymbol{y}) = \begin{bmatrix} \mathsf{Id}(\boldsymbol{x}) \\ \mathcal{G}(\boldsymbol{x}, \boldsymbol{y}) \end{bmatrix}, \quad \mathsf{Id} : \mathbb{R}^n \to \mathbb{R}^n,\ \mathcal{G} : \mathbb{R}^d \to \mathbb{R}^m, \tag{6.1}$$

where Id denotes the identity map. The existence of such a map can be guaranteed under very general conditions, for example by assuming that η and ν have no atoms Santambrogio, 2015. Furthermore, we say T is *strictly monotone* (Zeidler, 2013) if $\langle \mathsf{T}(\boldsymbol{z}) - \mathsf{T}(\boldsymbol{z}'), \boldsymbol{z} - \boldsymbol{z}' \rangle > 0$, for all $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{R}^d, \boldsymbol{z} \neq \boldsymbol{z}'$, with $\langle \cdot, \cdot \rangle$ denoting the Euclidean inner product, and it is *coercive* if $\lim_{\|\boldsymbol{z}\| \to \infty} \frac{\langle \mathsf{T}(\boldsymbol{z}), \boldsymbol{z} \rangle}{\|\boldsymbol{z}\|} = +\infty$. We use $\mathcal{T}(\mathbb{R}^d)$ to denote the space of block triangular maps from $\mathbb{R}^d$ into $\mathbb{R}^d$ of the form (6.1) that are bounded, continuous, strictly monotone, and coercive. It follows that such maps are surjective and that their inverses are well defined. We then formulate the optimization problem:

$$\min_{\mathsf{T}} D(\mathsf{T}_\sharp \eta \| \nu) \quad \text{s.t.} \quad \mathsf{T} \in \mathcal{T}(\mathbb{R}^d), \tag{6.2}$$

where D denotes an appropriate statistical divergence (J. Lin, 1991; Goodfellow et al., 2014), i.e., a functional that measures the difference between probability measures satisfying $D(\mu_1 \| \mu_2) \geq 0$ and $D(\mu_1 \| \mu_2) = 0$ if and only if $\mu_1 = \mu_2$.

Under the above assumptions, we obtain the following theorem, which is the foundation of our algorithm for conditional sampling. This result states that the map $\mathcal{G}$ extracted from any global minimizer of (6.2) can be used to map the marginal reference $\eta(\mathrm{d}\boldsymbol{y}) = N(0, I_m)$ to any target conditional $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ for a new input $\boldsymbol{x}^*$ in the support of $\nu(\mathrm{d}\boldsymbol{x})$. The following theorem generalizes Lemma 1 of Y. Marzouk et al. (2016).

Theorem 61. *Let* T *be a global minimizer of* (6.2) *achieving* $D(\mathsf{T}_\sharp\eta, \nu) = 0$. *Then for any new input* $\boldsymbol{x}^* \in$ *supp* $\nu(d\boldsymbol{x})$ *it holds that* $\mathcal{G}(\boldsymbol{x}^*, \cdot)_\sharp\eta(d\boldsymbol{y}) = \nu(d\boldsymbol{y}|\boldsymbol{x}^*)$.

Proof. First, let us recall some notation. For $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{x} \in \mathbb{R}^n$ we let $\pi_\eta(\boldsymbol{y})$ and $\pi_\nu(\boldsymbol{x})$ denote the Lebesgue densities of the marginals $\eta(\mathrm{d}\boldsymbol{y})$ and $\nu(\mathrm{d}\boldsymbol{x})$, respectively. Finally, let $\pi_\nu(\boldsymbol{y}|\boldsymbol{x})$ denote the Lebesgue density of the conditional $\nu(\mathrm{d}\boldsymbol{y}|\mathrm{d}\boldsymbol{x})$.

Fix $\boldsymbol{x}^* \in \mathbb{R}^n$ as the new input. Since T is monotone and hence invertible, by the change of variables formula we have

$$\begin{aligned}\mathcal{G}(\boldsymbol{x}^*, \cdot)_\sharp\pi_\eta(\boldsymbol{y}) &= \pi_\eta(\mathcal{G}^{-1}(\boldsymbol{x}^*, \boldsymbol{y}))|\det \nabla_{\boldsymbol{y}}\mathcal{G}^{-1}(\boldsymbol{x}^*, \boldsymbol{y})| \\ &= \frac{\pi_\nu(\boldsymbol{x}^*)\pi_\eta(\mathcal{G}^{-1}(\boldsymbol{x}^*, \boldsymbol{y}))}{\pi_\nu(\boldsymbol{x}^*)}|\det \nabla_{\boldsymbol{y}}\mathcal{G}^{-1}(\boldsymbol{x}^*, \boldsymbol{y})|.\end{aligned}$$

For the reference measure $\eta(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y}) = \nu(\mathrm{d}\boldsymbol{x}) \otimes \eta(\mathrm{d}\boldsymbol{y})$, its Lebesgue density has the form $\pi_\eta(\boldsymbol{x}, \boldsymbol{y}) = \pi_\nu(\boldsymbol{x})\pi_\eta(\boldsymbol{y})$. Therefore, the pushforward density can be written as

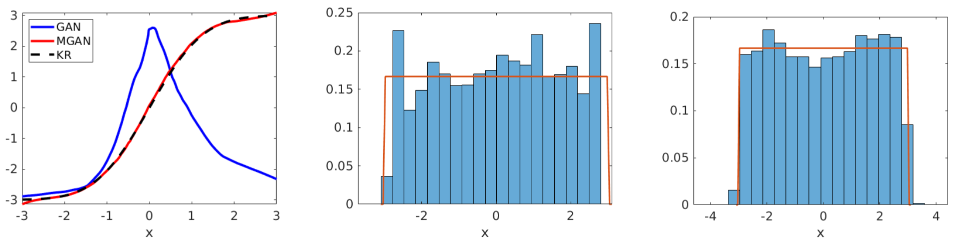
$$\begin{aligned}\mathcal{G}(\boldsymbol{x}^*, \cdot)_\sharp\pi_\eta(\boldsymbol{y}) &= \frac{\pi_\eta(\boldsymbol{x}^*, \mathcal{G}^{-1}(\boldsymbol{x}^*, \boldsymbol{y}))}{\pi_\nu(\boldsymbol{x}^*)}|\det \nabla_{\boldsymbol{y}}\mathcal{G}^{-1}(\boldsymbol{x}^*, \boldsymbol{y})| \\ &= \frac{\mathsf{T}_\sharp\pi_\eta(\boldsymbol{x}^*, \boldsymbol{y})}{\pi_\nu(\boldsymbol{x}^*)},\end{aligned}$$

where we have used the change of variables formula once more together with the fact that the first component of T is an identity map with respect to $\boldsymbol{x}$. For a global minimizer T that satisfies $\mathsf{T}_\sharp\pi_\eta(\boldsymbol{x}, \boldsymbol{y}) = \pi_\nu(\boldsymbol{x}, \boldsymbol{y})$, the numerator is equal to the Lebesgue density of $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$. Thus, by the definition of the conditional density

$$\mathcal{G}(\boldsymbol{x}^*, \cdot)_\sharp\pi_\eta(\boldsymbol{y}) = \frac{\pi_\nu(\boldsymbol{x}^*, \boldsymbol{y})}{\pi_\nu(\boldsymbol{x}^*)} = \pi_\nu(\boldsymbol{y}|\boldsymbol{x}^*).$$

□

Note that by the above theorem, any member of $\mathcal{T}(\mathbb{R}^d)$ that achieves $D(\mathsf{T}_\sharp\eta, \nu) = 0$ can be used for conditioning and so the uniqueness of the minimizer is not pertinent to the conditional sampling task. The existence of such minimizers can be guaranteed under mild conditions following the construction of the KR map pushing $\eta(\mathrm{d}\boldsymbol{z})$ to $\nu(\mathrm{d}\boldsymbol{z})$ (Santambrogio, 2015) although the KR map is by definition fully triangular and so satisfies stricter constraints than required by the class $\mathcal{T}(\mathbb{R}^d)$.



(a) $\mathsf{T}(x)$ (b) GAN (c) MGAN

Figure 61: Comparison between GANs and MGANs for pushing $N(0,1)$ to $U(-3,3)$. (a) The map T obtained from GANs, MGANs, and the analytically computed KR map demonstrating that the KR and MGAN maps are invertible while the GAN map is not. (b, c) Histogram of GAN and MGAN pushforward samples demonstrating that both methods approximate the uniform distribution $U(-3,3)$.

The monotonicity constraint on $\mathcal{T}(\mathbb{R}^d)$ is the most important constraint in problem (6.2), as it ensures the invertibility of T (see the proof of Theorem 61 in the supplementary material) which in turn enables conditional sampling with the map $\mathcal{G}(\boldsymbol{x}^*, \cdot)$. Eliminating the monotonicity constraint can easily result in a non-invertible T, as depicted in Figure 61 where we consider the problem of pushing forward a standard normal distribution $N(0,1)$ to the uniform measure $U(-3,3)$. As demonstrated in Figure 61(a), the map found by a standard GAN without monotonicity is not invertible, while the MGAN map is monotone and hence invertible; in fact, the MGAN map precisely coincides with the KR map in this case. We also note that Theorem 61 holds in the setting where the monotonicity constraint on $\mathcal{T}(\mathbb{R}^d)$ is replaced with the requirement that T is surjective. We do not pursue this direction here, however, since the monotonicity constraints allow for more flexibility in the parameterization of T.

Note that, from the GAN perspective, T would be the *generator* map, but our construction differs from standard GAN generators that typically map reference measures on a low-dimensional latent space to high-dimensional target measures. The existence of a transport map satisfying $\mathsf{T}_\sharp \eta = \nu$ cannot be guaranteed in that case unless ν is supported on a low-dimensional manifold embedded in $\mathbb{R}^d$ whose intrinsic dimension is at most the dimension of the latent space. We do not make this manifold assumption here, as we are interested in the general problem of ν with full support, and in extracting conditionals of arbitrary size from ν. Moreover, it is unclear how to define the necessary notions of monotonicity and invertibility without more specific knowledge of the manifold geometry.

We also note that global minimizers of (6.2) can also be used for conditional density estimation: Let $\pi_\nu(\boldsymbol{y}|\boldsymbol{x}^*)$ and $\pi_\eta(\boldsymbol{y})$ denote the probability density functions of $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ and $\eta(\mathrm{d}\boldsymbol{y})$ respectively. The change of variables formula yields $\pi_\nu(\boldsymbol{y}|\boldsymbol{x}^*) = \pi_\eta \circ \mathsf{F}^{-1}(\boldsymbol{x}^*, \boldsymbol{y})|\det \nabla_{\boldsymbol{y}} \mathsf{F}^{-1}(\boldsymbol{x}^*, \boldsymbol{y})|$, where $\mathsf{F}^{-1}(\boldsymbol{x}^*, \cdot)$ is the inverse of the map $\boldsymbol{\xi} \mapsto \mathsf{F}(\boldsymbol{x}^*, \boldsymbol{\xi})$. The inverse map $\mathsf{F}^{-1}(\boldsymbol{x}^*, \cdot)$ can be evaluated with standard root-finding methods (Y. Marzouk et al., 2016) or parameterized and trained via regression. In fact, if density estimation is the primary goal, then estimating directly the inverse map $\mathsf{F}^{-1}(\boldsymbol{x}^*, \boldsymbol{y}) := \mathsf{S}(\boldsymbol{x}^*, \boldsymbol{y})$ yields $\pi_\nu(\boldsymbol{y}|\boldsymbol{x}^*) = \pi_\eta \circ \mathsf{S}(\boldsymbol{x}^*, \boldsymbol{y})|\det \nabla_{\boldsymbol{y}} \mathsf{S}(\boldsymbol{x}^*, \boldsymbol{y})|$, which eliminates the need for inverting $\mathcal{G}$ altogether. Since we are focused here on efficient conditional sampling procedures, we leave this conditional density estimation direction for future research.

The training and sampling procedures

We now outline a practical procedure for solving (6.2) and for sampling the conditional $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$. Further details regarding the practical algorithm are summarized in the supplementary material. We consider (6.2) with a GAN loss functional for D. In particular we use either the least-squares GAN (LSGAN) loss of Mao et al. (2017) or the Wasserstein GAN with gradient penalty (WGAN-GP) loss of Gulrajani et al. (2017) for our numerical experiments in Section 6.4.

We approximate the $\mathcal{G}$ component of T with a neural network and replace the strict monotonicity constraint on T with an average monotonicity penalty. We also discard the coercivity constraint as it pertains to the behavior of the map in the tails, which is not relevant in practice where only a finite number of samples from the reference and target are available. Then, assuming T is in the form (6.1), we consider the optimization problem:

$$\begin{aligned} \min_{\mathsf{T}} \max_{f} \, & D_{\text{GAN}}(\mathsf{T}, f) \\ & - \lambda \mathbb{E}_{\boldsymbol{w}\sim\eta} \mathbb{E}_{\boldsymbol{w}'\sim\eta} \langle \mathsf{T}(\boldsymbol{w}) - \mathsf{T}(\boldsymbol{w}'), \boldsymbol{w} - \boldsymbol{w}' \rangle, \end{aligned} \tag{6.3}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ denotes the discriminator, D_{GAN} is an appropriate GAN loss measuring the quality of the map T and the discriminator f, and $\lambda > 0$ is a multiplier controlling the weight of the average monotonicity penalty. While this condition does not ensure that T is monotone everywhere on the support of η, numerically we find that it is sufficient to ensure that T is monotone with high probability, i.e., on most of the support of η. Indeed, the probability that T is monotone can easily be tracked during training to certify the invertibility of the minimizer. Furthermore, note that in contrast to training procedures for flow models based on minimizing

Kullback–Leibler (KL) divergence (or maximizing a log-likelihood), solving (6.3) does *not* require computing any Jacobian determinants of T; this permits complete flexibility in the choice of neural network parameterization.

Once the map T is trained, we proceed to generate new approximate samples from the conditional measure $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ as follows: we generate a set of i.i.d. samples $\boldsymbol{u}_i \sim \eta(\mathrm{d}\boldsymbol{y})$ drawn from the reference marginal and simply set $\boldsymbol{y}_i = \mathsf{F}(\boldsymbol{x}^*, \boldsymbol{u}_i)$. Assuming that the components of T are good approximations to a global minimizer of (6.2), we expect the $\boldsymbol{y}_i$ to be approximately distributed according to $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$. These new samples can then be used to compute statistics and uncertainty estimates.

6.3 Related work

Conditional generative models. Various architectures and learning algorithms have been proposed to sample approximately from conditional distributions. Conditional GANs and VAEs (Mirza and Osindero, 2014; Ivanov, Figurnov, and Vetrov, 2019) append inputs $\boldsymbol{x}$ to their models and train using samples from the joint distribution. These models have been shown to be effective at approximating multi-modal distributions on the output variables $\boldsymbol{y}$, for instance when modes correspond to different values of a discrete class label $\boldsymbol{x}$. Similarly, in the imaging setting, recent work (e.g., Zhu et al. (2017) and Isola et al. (2017)) employs modifications of GANs to solve image-to-image translation problems. In general, these models do not offer any guarantees for obtaining the correct conditionals. In comparison, MGAN uses a similar architecture but captures the true conditional distribution under the assumptions of Theorem 61 (e.g., monotonicity of T). Recent techniques have also addressed the harder task of learning all the conditionals of a joint distribution, i.e., for arbitrary subsets of conditioning variables. To this end, Ivanov, Figurnov, and Vetrov (2019) uses a conditional VAE model, while Belghazi et al. (2019) proposes an adversarial training approach for better performance. Both constructions employ a weighted loss over all possible choices of conditionals, but, similar to conditional GANs, this loss does not guarantee that any particular conditional is obtained correctly. In contrast, Lindgren, Whang, and Dimakis (2020) focuses on the problem of correctly extracting a single conditional from a fixed pre-trained flow model. The method uses a variational inference objective (Rezende and Mohamed, 2015), but must be re-trained for *each* new value of the observations or conditioning variables. On the other hand, MGAN enables sampling from the conditional $\nu(\boldsymbol{y}|\boldsymbol{x}^*)$ for any new value of $\boldsymbol{x}^*$ without retraining.

Conditional posterior sampling. Generative models have also been proposed specifically for sampling posterior distributions in Bayesian inference, often in the context of *inverse problems*. Perhaps the closest approach to MGAN is the deep posterior sampling method (Adler and Öktem, 2018), which uses the Wasserstein-1 cost function instead of the LSGAN or WGAN-GP losses and does not impose monotonicity on the map $\mathcal{G}$. Adler and Öktem (2018) also propose a special discriminator to help avoid mode collapse, but our empirical results below suggest that the method does not correctly characterize the posterior. For inverse problems, Ardizzone, Kruse, et al. (2018) propose to sample from the conditional distribution of the unknown parameters $\boldsymbol{y}$ by learning a map from the observed data $\boldsymbol{x}$, augmented with some latent variables, to $\boldsymbol{y}$. The authors prove that their approach is consistent for conditional sampling in the specific setting when $\boldsymbol{x}$ is a deterministic function of $\boldsymbol{y}$. This is in contrast to the more general setting considered here, where $(\boldsymbol{x}, \boldsymbol{y})$ have an absolutely continuous joint measure ν with minimal modelling assumptions.

Invertible normalizing flows. Given samples from a target measure, normalizing flows compose bijective transformations to define a map S that pushes forward the target to a reference (e.g., standard Gaussian) measure of the same dimension. These flows are learned by maximizing the likelihood of the samples under the approximate density provided by the model, which in turn requires computing the Jacobian determinant of S. As a result, specific structural choices are imposed to define transformations that allow tractable density evaluation without affecting the expressiveness of the approximations (Papamakarios, Nalisnick, et al., 2019). These flows can be repurposed for conditional modeling and simulation-based inference by appending inputs to the maps, representing the conditioning variables (Cranmer, Brehmer, and Louppe, 2020; Ardizzone, Lüth, et al., 2019; Y. Marzouk et al., 2016). In this context, MGAN can be seen as a flow-based model that is instead trained using a GAN loss function. In comparison to normalizing flows, MGAN does not require evaluations of the pushforward density or of any Jacobian determinants during training. Hence, MGAN is not constrained to any specific parameterizations or structure, only *block* triangularity, and it is not necessary to compose many maps to define an expressive flow. Furthermore, sampling from the target density using MGAN does not require inverting the map at each sample—a costly step for many other flow parameterizations.

Conditional density estimation. Another relevant body of work estimates and selects models for the conditional density directly. Examples include mixture models that are parameterized with neural networks (Bishop, 1994; Rothfuss et al., 2019), or more flexible nonparametric models Arbel and Gretton, 2018; Shiga, Tangkaratt, and Sugiyama, 2015; Ambrogioni et al., 2017. However, parametric methods impose structural constraints on the target conditional densities and do not focus on conditional *sampling* as we do here, while nonparametric methods typically have growing sampling costs with the size of the data set and require careful regularization to avoid overfitting.

6.4 Experiments

We now present a series of numerical experiments that demonstrate the effectiveness of MGANs in various conditional sampling applications, ranging from inverse problems to image in-painting. We compare MGANs to other methods in the literature and perform ablation studies to demonstrate the importance of various constraints in our formulation. Complete details on each experiment, such as the choice of architectures and training hyperparameters, can be found in the supplementary material.

Synthetic examples

We start with an example in two dimensions where the map T can be computed explicitly. Consider the following input-to-output maps,

$$y = \tanh(x) + \gamma, \qquad \gamma \sim \Gamma(1, 0.3), \tag{6.4}$$

$$y = \tanh(x + \gamma), \qquad \gamma \sim N(0, 0.05), \tag{6.5}$$

$$y = \gamma \tanh(x), \qquad \gamma \sim \Gamma(1, 0.3), \tag{6.6}$$

where $x \sim U[-3, 3]$ in all cases. We choose $\tanh$ as our regression function since it is nonlinear and can be used as a continuum model for classification problems.

We consider the problem of conditioning y on x and compare MGAN to the method of Adler and Öktem (2018), henceforth referred to as “Adler,” as well as the solution of (6.3) with the parameter $\lambda = 0$ which we refer to as CGANs. The latter comparison is made to highlight the importance of the monotonicity constraint, and the label CGAN is chosen as this approach resembles Conditional GANs Mirza and Osindero, 2014.

Figure 62 compares all methods on each of the above problems with $N = 50000$ training points, the LSGAN loss, and $\lambda = 0.01$. The upper rows of Figure 62 demonstrate the ability of MGANs to capture the true joint measures $\nu(\mathrm{d}x, \mathrm{d}y)$, while the Adler and CGAN methods represent these joint measures poorly: in particular, both methods overestimate the correlation between input and output, which manifests as regions of higher density in the 2D plane. In the last row of Figure 62 we also compare several conditional histograms of MGAN to the true conditional PDFs, explicitly showing MGAN's ability to capture the correct conditionals. In Table 61 we compare the relative L^2 distance and the KL divergence between the density of the true joint measure $\nu(\mathrm{d}x, \mathrm{d}y)$ and the estimated densities obtained from samples of the three methods. The errors are computed on the joint distribution in order to measure the errors across all of the conditionals $\nu(\mathrm{d}y|x)$. Similarly to the figure above, these results highlight MGAN's superior performance at correctly capturing the conditional distributions.

		MGAN	Adler	CGAN
Rel. L^2	(6.4)	**.910** (.212)	2.89 (1.51)	4.73 (1.22)
	(6.5)	**.766** (.276)	1.83 (.542)	1.37 (1.08)
	(6.6)	**.997** (.264)	4.53 (1.43)	6.89 (2.63)
KL	(6.4)	**.513** (.225)	5.10 (7.34)	17.2 (13.5)
	(6.5)	**.481** (.502)	3.03 (2.33)	1.75 (1.63)
	(6.6)	**.656** (.278)	18.0 (16.0)	53.8 (39.4)

Table 61: Distributional errors between the joint measure $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$ and the MGAN, Adler, and CGAN approximations for problems (6.4), (6.5) and (6.6). We present the mean errors computed over the last 10 epochs of training with standard deviations reported within brackets. Smallest mean in each column is highlighted in bold. Reported relative L^2 errors are scaled by 10 while KL errors are scaled by 10^3 for readability.

Block triangular versus triangular maps

We now perform an ablation study to highlight the benefits of using block triangular rather than triangular maps T. In this example, we consider a two-dimensional target distribution for $\boldsymbol{y} = (y_1, y_2)$ with no conditioning variables where $y_1 \sim \mathcal{N}(0, 1)$ and $y_2|y_1 \sim \mathcal{N}(y_1^2 + 1, 0.5^2)$. This joint density of $\boldsymbol{y}$ can be represented exactly as the push-forward of $\eta(\mathrm{d}\boldsymbol{y})$ through the map $\mathcal{G}(\boldsymbol{y}) = [y_1; y_1^2 + 1 + 0.5y_2]$. Hence, $\mathcal{G}$ can be easily approximated by a triangular map of this form. However, when the ordering of y_1, y_2 is reversed—i.e., when the first component of $\mathcal{G}$ depends on y_2 instead of y_1—the map is more challenging to approximate. To resolve this issue, one common approach is to compose many maps to define an expressive normalizing flow (see Papamakarios, Pavlakou, and Murray (2017) for a similar application). We now demonstrate that by using a block triangular parameterization we can avoid

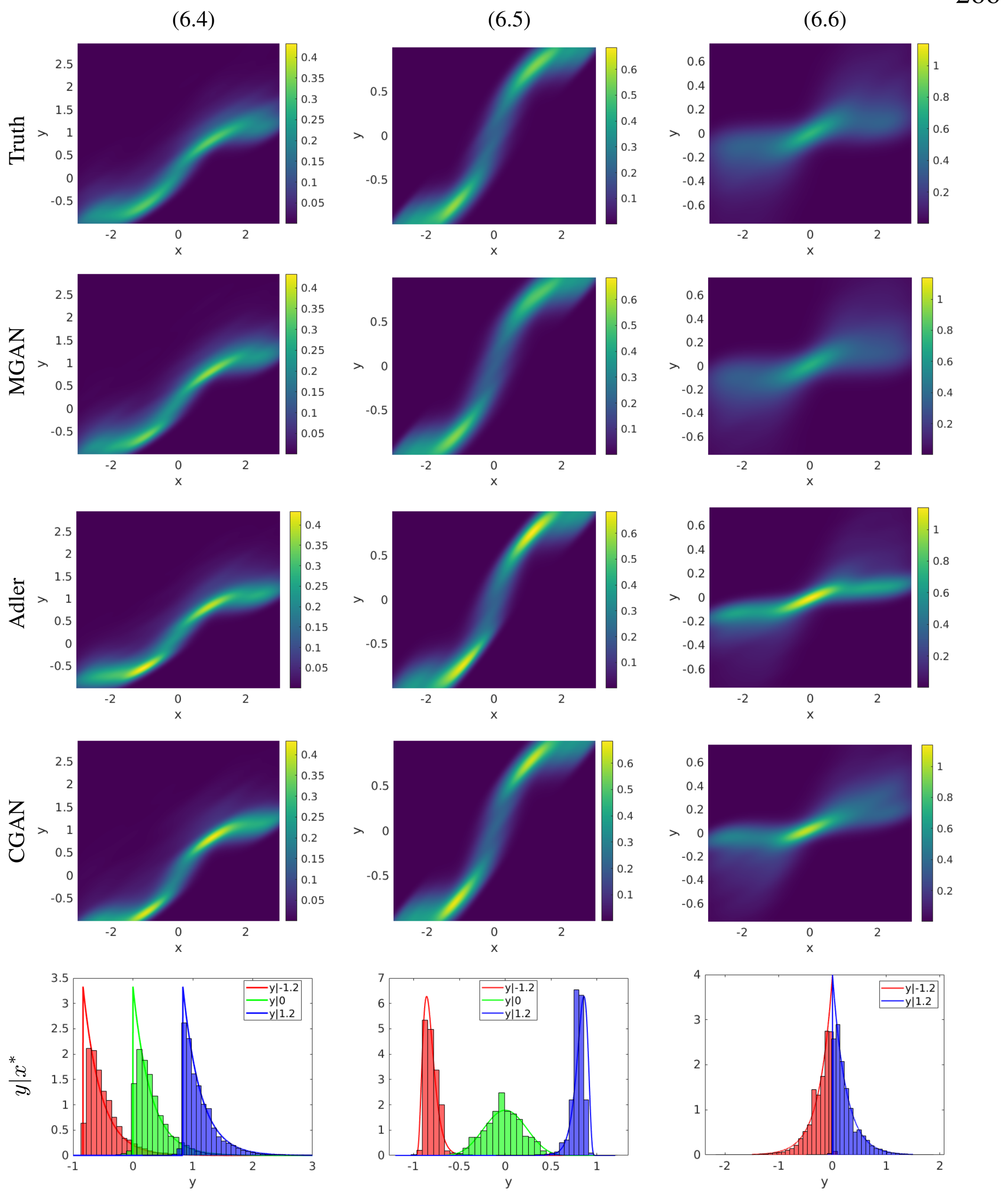
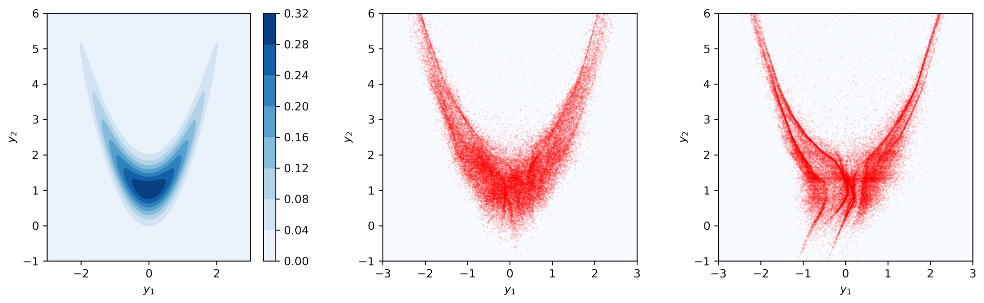


Figure 62: The columns correspond to problems (6.4), (6.5), and (6.6) respectively. The top four rows compare the true joint densities $\pi_\nu(x, y)$ to kernel density estimates (KDEs) from MGAN, Adler, and CGAN samples. The bottom row compares histograms of conditional samples from MGAN to the true conditional densities (solid lines) for all three problems.

issues pertaining to the ordering of the variables and achieve a more robust map in practice.

We use $N = 10^4$ training samples, $\lambda = 0.01$, and the LSGAN loss function to train an MGAN with either fully triangular or block triangular structure. Both maps have the same total number of parameters. Figure 63 compares the samples generated by the triangular and block triangular maps to the true density. We observe that the block triangular map is able to capture the target density independent of the ordering of the variables, unlike the triangular map. In Table 62 we report the KL divergence between the true and approximated distributions for both variable orderings. While the block triangular map has similar performance under both the favorable and reverse orderings, the performance of the triangular map degrades significantly depending on the ordering. This suggests that block triangular maps are less sensitive to variable order, a major advantage of MGANs in comparison to autoregressive models where it is necessary to specify a variable ordering in advance. See Section 6.3 for additional discussion relating MGANs to other invertible flow-based models.



(a) True density (b) Block triangular (c) Triangular

Figure 63: (a) The true density of (y_1, y_2) considered in Section 6.4. (b) Samples generated by MGAN using a block triangular map with the reverse ordering of the variables. (c) Samples generated by a fully triangular MGAN also with reverse ordering.

	Block triangular	Triangular
Favorable order	0.056 (0.003)	0.039 (0.002)
Reverse order	0.058 (0.002)	0.102 (0.004)

Table 62: KL divergence errors for block triangular and triangular MGANs computed using $N = 10^4$ training samples. The approximate densities are estimated using KDE with an optimal bandwidth parameter that is chosen using 5-fold cross-validation. Each KL divergence is based on 5×10^4 test samples and is reported together with its 95% standard error within brackets.

Biochemical oxygen demand model

We now turn to an inference problem for modeling biochemical oxygen demand (BOD), often used as a test case for sampling schemes (Y. Marzouk et al., 2016; Parno and Youssef M Marzouk, 2018; Bardsley et al., 2014). Consider the time-dependent model $\mathcal{B}(t) = A(1 - e^{-Bt}) + \gamma$, where A and B are unknown parameters and $\gamma \sim N(0, 10^{-3})$ is the observational noise. We model the parameters as $A = 0.4 + 0.4\left(1 + \text{erf}\left(\frac{\rho_1}{\sqrt{2}}\right)\right)$ and $B = 0.01 + 0.15\left(1 + \text{erf}\left(\frac{\rho_1}{\sqrt{2}}\right)\right)$ where $(\rho_1, \rho_2) \sim N(0, I_2)$. Our goal is to recover $\boldsymbol{y} = (\rho_1, \rho_2)$ from observations of the forward map $\mathcal{B}(t)$ at times $t = 1, 2, 3, 4, 5$. Hence, our training data is in the form of i.i.d. realizations of $\boldsymbol{y} \sim N(0, I_2)$ along with a simulated vector $\boldsymbol{x} = (\mathcal{B}(1), \ldots, \mathcal{B}(5))$ for each $\boldsymbol{y}$.

Following Y. Marzouk et al. (2016), we sample the conditional at

$$\boldsymbol{x}^* = (0.18, 0.32, 0.42, 0.49, 0.54).$$

Table 63 shows the relative L^2, KL, and MMD distances between the estimated conditionals and the true conditional $\boldsymbol{\rho}|\boldsymbol{x}^*$, where the latter is characterized using a long MCMC chain. We train an MGAN using the LSGAN loss with $\lambda = 0.01$, and compare it to the method of Adler. We show comparisons where both maps are trained with either $N = 5000$ or $N = 50000$ samples. In both settings, MGAN yields better approximations of the conditional according to all three performance metrics. Figure 64 compares the density estimates of the MGAN and Adler conditionals to MCMC. Once again, we observe that the Adler method overestimates the dependence between ρ_1, ρ_2, while MGAN does not. Further numerical results, including a comparison between MGAN and the method presented in Y. Marzouk et al. (2016), are available in the supplementary material.

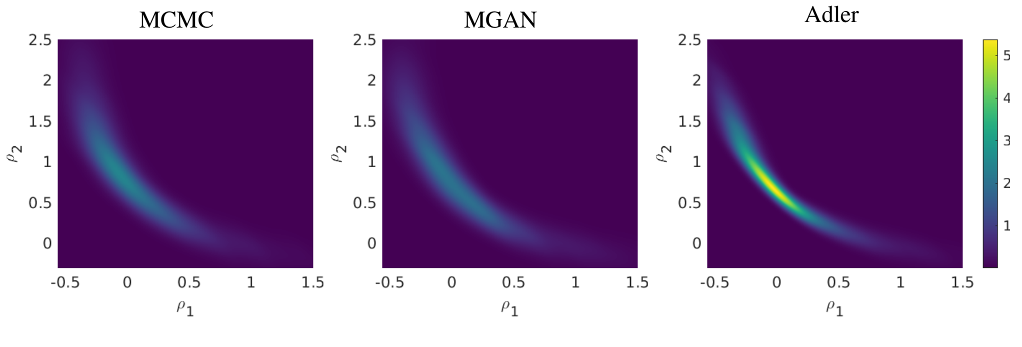


Figure 64: KDE of $\boldsymbol{y}|\boldsymbol{x}^*$ samples for the BOD problem obtained from 30,000 MCMC samples as a benchmark, compared with 30,000 samples from the MGAN and Adler maps, trained with 50000 data points.

	N	Rel. L^2	KL	MMD
Adler	5000	5.52 (1.49)	38.0 (22.5)	26.7 (23.6)
	50000	6.93 (1.61)	65.6 (16.1)	24.0 (18.8)
MGAN	5000	**3.72** (1.30)	**16.8** (10.0)	**19.5** (23.5)
	50000	**1.77** (.576)	**3.28** (1.85)	**4.84** (5.25)

Table 63: Conditional distributional errors for the BOD problem, comparing a reference MCMC approximation of $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ with MGAN or Adler. Mean errors computed over the last 10 epochs of training are reported with standard deviations within the brackets. Smallest mean for each amount of training data is highlighted in bold. Relative L^2 errors are scaled by 10 while KL and MMD errors are scaled by 10^3 to improve readability.

Darcy flow

Next, we consider a benchmark inverse problem from subsurface flow modeling Iglesias, K. Lin, and Stuart, 2014 and electrical impedance tomography Kaipio and Somersalo, 2005. Consider the partial differential equation (PDE)

$$\begin{aligned} -\nabla \cdot (a(s)\nabla p(s)) &= f(s), \qquad & s \in (0,1)^2, \\ p(s) &= 0, \qquad & s \in \partial(0,1)^2. \end{aligned} \tag{6.7}$$

We interpret $p(s)$ as the pressure field of subsurface flow in a reservoir with permeability coefficients $a(s)$ under the forcing $f(s)$. We further consider a two-scale model for the permeability field $a(s) = A\mathbb{1}_{\Omega_A}(s) + B\mathbb{1}_{\Omega_B}(s)$ where $\Omega_A, \Omega_B \subset [0,1]^2$ are disjoint and such that $\Omega_A \cup \Omega_B = [0,1]^2$, and we let $A \sim U(3,5)$ and $B \sim U(12,16)$. Figure 65(a) shows the sets Ω_A (yellow) and Ω_B (blue) that we use in our experiments. We consider the inverse problem of recovering the permeability parameters $\boldsymbol{y} = (A, B)$ from noisy measurements of the point values of the pressure field p, i.e., $\boldsymbol{x} = (p(s_1), \ldots, p(s_{16})) + \boldsymbol{\gamma}$, where $\boldsymbol{\gamma} \sim N(0, 10^{-7} I_{16})$. Figure 65(b) shows an example of a pressure field along with the measurement locations s_j (red dots).

We train an MGAN with LSGAN loss and $N = 10^5$ training points by sampling A, B and solving the PDE (6.7) using finite differences. We then test the MGAN conditional samples at new inputs $\boldsymbol{x}_j^*$ for $j = 1, 2, 3$, generated by solving the PDE (6.7) with $(A, B) = (3.5, 13)$, $(4, 14)$, and $(4.5, 15)$, respectively; hence we expect $\boldsymbol{y}|\boldsymbol{x}^*$ to concentrate around these prescribed values. Figures 65(c, d) show conditional KDEs computed with 30,000 MCMC samples and 30,000 MGAN samples. We observe that while MGAN captures the general shape of the conditionals away from the constraints imposed by the prior, it struggles to capture the sharp boundaries of the conditionals at the edges of the prior support. Future work will investigate how to embed such constraints into the generative model, in order to mitigate these boundary effects.

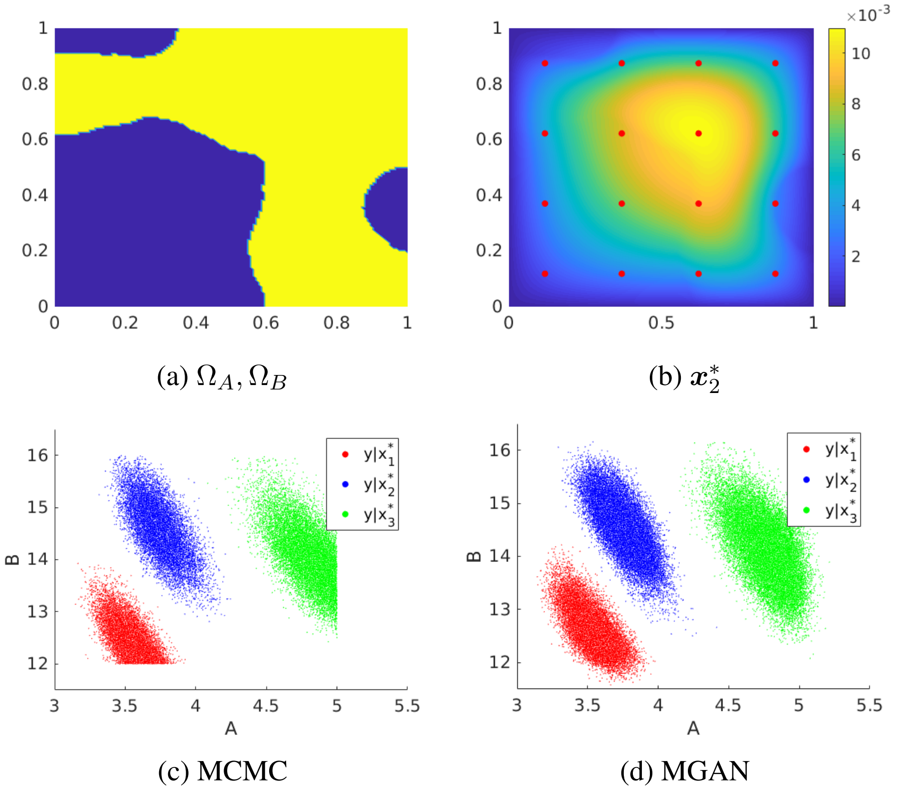


Figure 65: Setup and results for the Darcy flow inverse problem. (a) shows the sets Ω_A, Ω_B across which the permeability field varies. Blue regions have small permeability while yellow regions have large permeability. (b) depicts an example of the pressure field p as well the measurement locations. (c) and (d) show scatter plots of 30,000 MCMC and MGAN samples of $\boldsymbol{y}|\boldsymbol{x}_j^*$, for $j = 1, 2, 3$.

Image in-painting

For our final set of experiments, we consider the image in-painting problem, in which an image is reconstructed after having some sections removed. We view this problem as an image-to-image regression problem where the input $\boldsymbol{x}$ is the incomplete image and the output $\boldsymbol{y}$ is the in-painting.

We consider the MNIST and CelebA data sets. MNIST consists of 28×28 images of handwritten digits, and as input $\boldsymbol{x}$ we remove the middle 14×14 pixels by setting their values to zero, so that $\boldsymbol{x} \in \mathbb{R}^{588}$ and the output $\boldsymbol{y} \in \mathbb{R}^{14\times 14}$ corresponds to the removed pixels. CelebA consists of $64 \times 64 \times 3$ RGB images of celebrity faces (converted to a standard size using bi-cubic interpolation). The input $\boldsymbol{x} \in \mathbb{R}^{32\times 64\times 3}$

consists of the top half of each image, and the output $\boldsymbol{y} \in \mathbb{R}^{32\times 64\times 3}$ consists of the bottom half.

We train an MGAN on the MNIST training set using the LSGAN loss, with $\lambda = 0.01$. Then, for each of 800 randomly selected images $\boldsymbol{x}^*$ from the MNIST test set, we generate 1000 possible in-paintings. Figure 66 shows our in-painting results for the digits 9, 7, 2, 0, 1, 5, where we present conditional samples as well as mean and variances of the corresponding in-paintings. We note that the MGAN can produce different digits given the same corrupted input image, hence capturing multi-modal conditionals. We further label the conditional samples using the pre-trained LeNet classifier of Lecun et al. (1998) to obtain classification probabilities for the in-paintings, also reported in Figure 66. Interestingly, we observe that in most cases the labels with highest probabilities correspond to visual inspection of the means. We further tested the quality of the MGAN map T by computing the Fréchet inception distance (FID) Heusel et al., 2017 over the test set and achieved a score of approximately 3.5.

In the case of CelebA, we trained an MGAN on the training set of 162,770 images using the WGAN-GP loss, with $\lambda = 10^{-4}$. We added Gaussian white noise with standard deviation 0.05 to further corrupt the training set, as we found this to be crucial in countering conditional mode collapse. We also noticed that the trained MGAN map T was monotone even with a relatively small monotonicity penalty λ in this example. In general, however, we emphasize that monotonicity of the map should always be monitored during training.

Our results are summarized in Figure 67, where we show conditional samples of in-paintings for images in the CelebA test set together with pixelwise means and variances of the image intensities. We note the variability of the MGAN samples, producing different smiles, hair styles, jawlines, outfits, and backgrounds as one would expect from a distributional in-painting method. We also computed an FID score of approximately 35 for the full MGAN map T in this example. While FID is a good metric for photorealism, it fails to capture variability in the conditionals. For example, we noticed that models which collapse conditionally, i.e., which produce the same in-painting $\mathcal{G}(\boldsymbol{x}^*, \boldsymbol{y})$ for any realization of $\boldsymbol{y} \sim \eta(d\boldsymbol{y})$, can still obtain similarly good FID scores while clearly not being able to capture the true conditional. To our knowledge, distributional in-painting on the CelebA data set has not been explored in the literature, and thus we cannot compare the FID of our result to others. The closest to the state of the art is an FID of 30 reported in Lindgren, Whang, and

Dimakis (2020) for the CelebA-HQ data set.

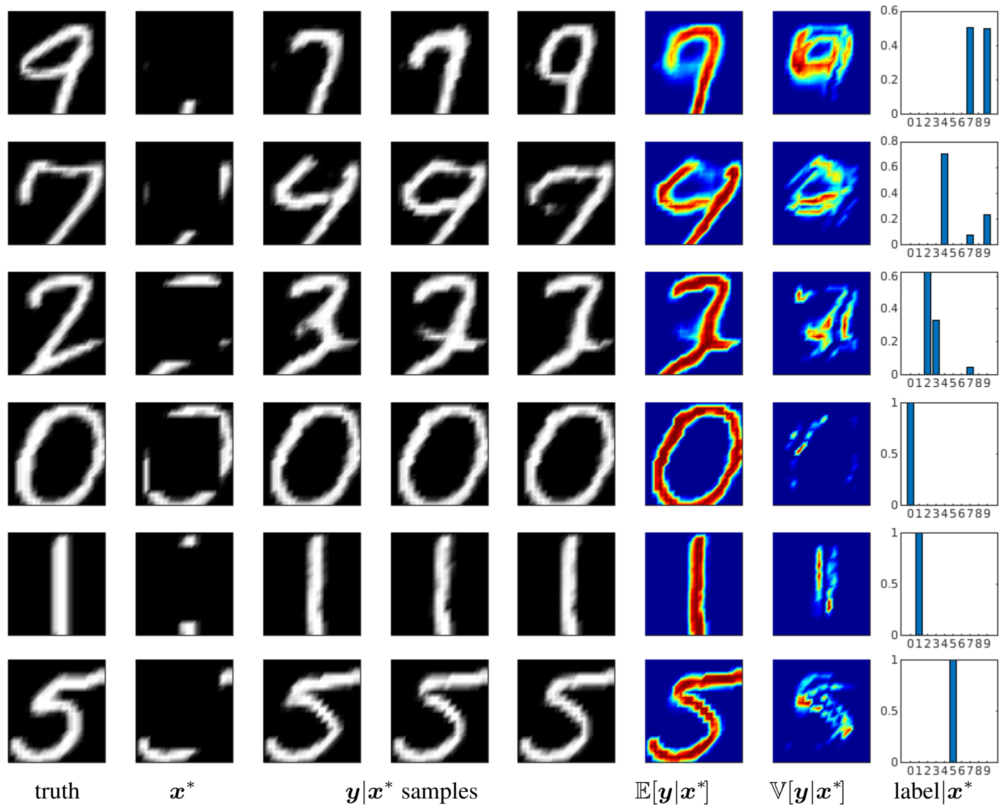


Figure 66: Example in-paintings using MGAN from the MNIST test set. The first column shows the ground truth images. The second column shows $\boldsymbol{x}^*$ along with the block to be in-painted. The third, fourth, and fifth columns show random in-paintings generated from the conditional $\boldsymbol{y}|\boldsymbol{x}^*$. The sixth and seventh columns show the pixel-wise conditional mean and variance computed from 1000 samples. The last column shows the label probabilities of samples from $\boldsymbol{y}|\boldsymbol{x}^*$ classified using LeNet.

6.5 Conclusion

We presented MGANs, a model-agnostic method for conditional sampling of outputs given new inputs—i.e., conditional generative modeling—with a theoretical guarantee for exact recovery of these conditionals. Our method adds monotonicity constraints and block triangular structure to a GAN with a full-dimensional latent space, and can also be seen as an easily parameterized invertible flow model trained with a GAN loss. Numerical experiments elucidate the effectiveness and versatility of MGANs, with applications in supervised learning and inverse problems. In future research, the interplay between the quality of the full MGAN map obtained from (6.3) and the accuracy of the derived conditionals warrants theoretical investigation. Relatedly, approximation results characterizing the expressiveness of block triangu-

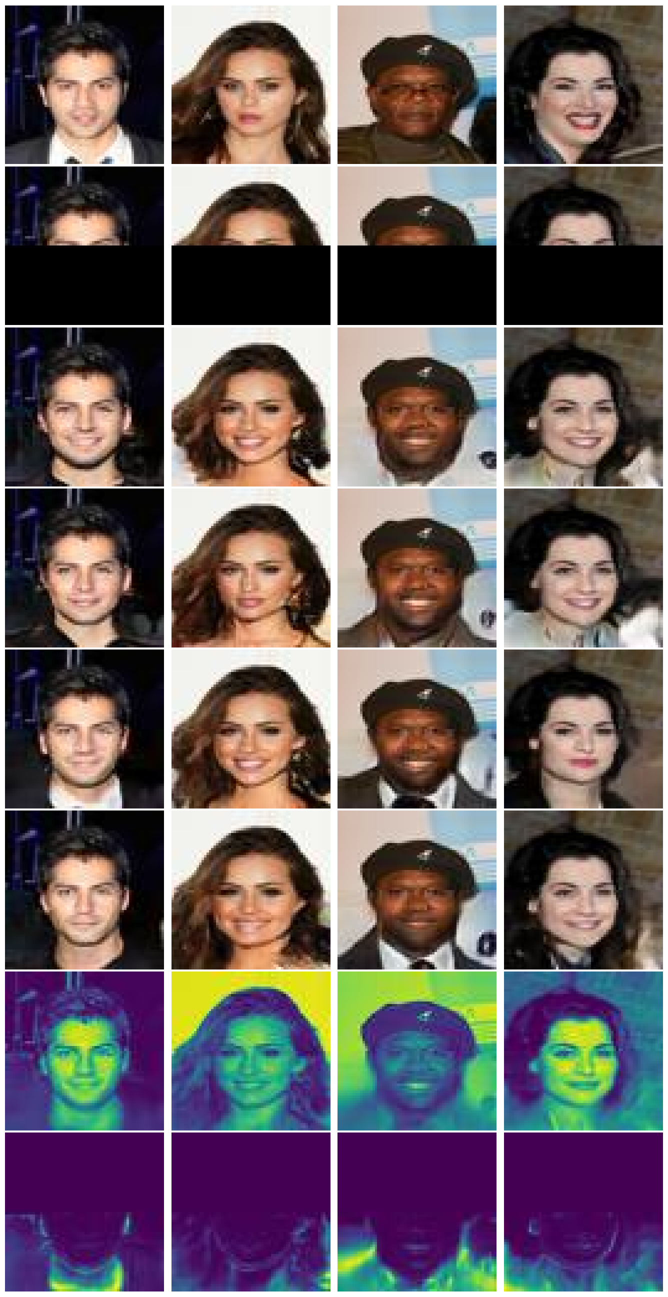


Figure 67: Example in-paintings using MGAN from the CelebA test set. First row depicts the ground truth image, second row shows the observed image $\boldsymbol{x}^*$ while the next four columns are random samples from the conditional $\boldsymbol{y}|\boldsymbol{x}^*$. The bottom two rows show the pointwise means and variances of the intensities of the conditional samples generated by the MGAN.

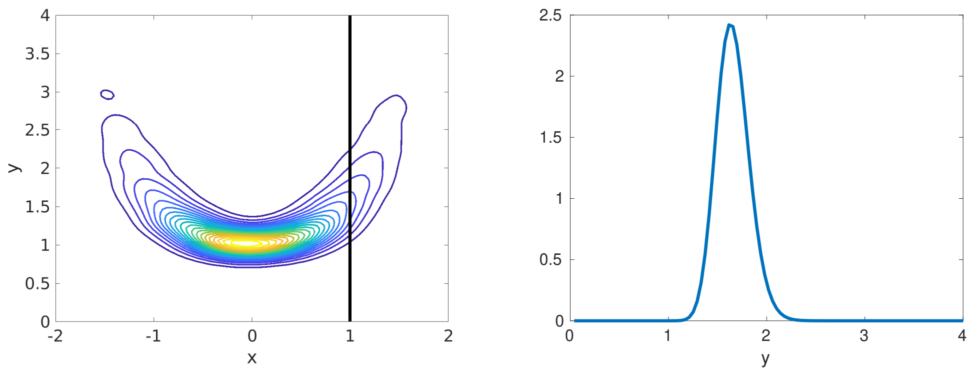
lar maps would be of interest. The extension of MGANs to conditional sampling on infinite-dimensional function spaces is another exciting research direction, pertinent to inverse problems.

6.6 Discussion

Here, we collect further discussions surrounding conditional sampling, MGANs, and other topics in statistics, machine learning, and applied mathematics. Section 6.6 discusses the connections between conditional sampling and supervised learning (SL) as well as the importance of UQ for SL. In Section 6.6, we present MGANs as a method for likelihood-free inference. In Section 6.6 we outline applications of MGANs in the solution of inverse problems with black-box forward maps.

A model agnostic approach to SL and UQ

Conditional sampling can be viewed as the problem of generating samples from certain "slices" of a probability measure $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$. As demonstrated in Figure 68, conditioning the output $\boldsymbol{y}$ on a new point $\boldsymbol{x}^*$ amounts to restricting $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$ along the hyperplane $\boldsymbol{x} = \boldsymbol{x}^*$, renormalizing, and then generating samples from the resulting distribution.



(a) $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$ (b) $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^* = 1)$

Figure 68: Schematic of conditional sampling. (a) The probability density function of the joint input and output measure $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$. (b) The probability density function of the conditional measure $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^* = 1)$ as a normalized slice of the joint measure.

As we briefly mentioned in Section 6.1, conditional sampling problems are ubiquitous in statistics, applied mathematics, and engineering. For example, most inference problems reduce to conditional sampling where one wishes to characterize an output parameter $\boldsymbol{y}$ at a new input parameter $\boldsymbol{x}^*$. In regression, we often wish to estimate $\boldsymbol{y}(\boldsymbol{x}^*)$ for $\boldsymbol{x}^* \in X \subset \mathbb{R}^n$, i.e., a certain subset of the input parameter space. As we also mentioned in Section 6.1, most SL algorithms such as ridge, LASSO, or neural network regression assume a functional relationship $\boldsymbol{y} = \mathcal{G}(\boldsymbol{x})$ and estimate $\mathcal{G}$ within certain parametric function spaces.

A core assumption at the heart of classic point estimation methods is the existence of a ground truth function $\mathcal{G}$ that could predict $\boldsymbol{y}$ at any new input $\boldsymbol{x}^*$. If a good approximation $\widehat{\mathcal{G}}$ to $\mathcal{G}$ can be computed, then one can predict the output $\boldsymbol{y}$ at the new input $\boldsymbol{x}^*$ simply by setting $\boldsymbol{y} = \widehat{\mathcal{G}}(\boldsymbol{x}^*)$. However, the estimate $\widehat{\mathcal{G}}$ is dependent on the training data and the approximation power of the chosen parameterization in relation to the ground truth $\mathcal{G}$.

With this issue in mind, it is natural to resort to a statistical model that characterizes $\widehat{\mathcal{G}}$ as a random variable, leading to the statistical or probabilistic perspective on SL Hastie, Tibshirani, and Friedman, 2009; Gressmann et al., 2019. Such probabilistic formulations, be they frequentist or Bayesian, have a major advantage over deterministic methods: the ability to characterize "uncertainties" in the predicted output $\boldsymbol{y}|\boldsymbol{x}^*$. Loosely speaking, the word "uncertainty" here refers to meaningful statistics of $\boldsymbol{y}|\boldsymbol{x}^*$ that characterize the variability of the predicted output, e.g., its mean and covariance. In the Bayesian setting, such statistics can be approximated using sampling methods such as MCMC C. Robert and Casella, 2013, variational techniques Fox and Roberts, 2012, or in simpler cases, via closed-form conjugacy relationships Gelman et al., 2013. In this regard, the MGANs approach introduced in the article belongs to the category of sampling techniques such as MCMC, whose goal is to generate independent samples from the law of $\boldsymbol{y}|\boldsymbol{x}^*$, as opposed to assuming some structural form of the probability measure directly.

However, a major advantage of MGANs (and also the flow-based models discussed in Section 6.3) is that the method is *model-agnostic*: the MGAN approach does not require prior knowledge of the form of $\mathcal{G}$ or any specific parameterization of $\widehat{\mathcal{G}}$. This is "in principle," of course, and under the assumption that the neural networks used to describe the transport maps in MGANs are expressive enough to capture the ground truth transport maps pushing the reference η to the target ν. The synthetic example in Section 6.4 demonstrates this feature of MGANs clearly: the joint target ν as well as the conditionals $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ are computed accurately for three different choices of $\mathcal{G}$, but using the same architecture and training procedure within the MGAN approach. Differences among the three models in that example are due *only* to the training sets.

It is important to note that while model-agnostic approaches such as MGANs and normalizing flows appear to be extremely flexible and general tools for inference, their theoretical properties—in particular the sample complexities needed to approximate the transport maps to a given accuracy—are largely open questions. A large body of theoretical research is dedicated to studying the consistency of classic statistical SL procedures from the Bayesian and frequentist perspectives, focusing on conditions and asymptotic regimes under which the estimated probability measure on $\boldsymbol{y}|\boldsymbol{x}^*$ or $\widehat{\mathcal{G}}$ contracts around a certain "ground truth" at a certain rate depending on the size and quality of training data Ghosal, Ghosh, Van Der Vaart, et al., 2000; Giné and Nickl, 2016. To the best of our knowledge, such detailed analysis has not been performed for measure transport techniques such as MGANs.

MGANs and likelihood-free Bayesian inference

Due to model agnosticism and the fact that the training process relies only on samples from the joint distribution $\nu(\mathrm{d}\boldsymbol{y}, \mathrm{d}\boldsymbol{x})$, MGANs can be readily applied for likelihood-free inference Grelaud et al., 2009; Gutmann and Corander, 2016; Papamakarios and Murray, 2016; Papamakarios, Sterratt, and Murray, 2019; Cranmer, Brehmer, and Louppe, 2020 and approximate Bayesian computation tasks, i.e., problems where the likelihood function is intractable or expensive to evaluate. Our numerical experiments in Sections 6.4 and 6.4 illustrate these applications, where evaluation of the likelihood may require the solution of a complicated ODE or PDE model. However, the training of MGANs does not require any likelihood evaluations or evaluations of the differential equations *during* training; once the training data set is given, the MGAN can be trained independently. This makes the MGAN approach particularly attractive in applications where the input-to-output map involves an stochastic process, differential equation, PDE, or a black-box expensive computer model. Furthermore, similarly to other conditional generative models, the training process amortizes the cost of inference by building a single model that can be used to sample from any conditional $\nu(\boldsymbol{y}|\boldsymbol{x}^*)$ given a realization of the inputs $\boldsymbol{x}^*$.

MGANs and Bayesian inverse problems

One specific application area for MGANs in the context of likelihood-free inference is inverse problems Kabanikhin, 2011; Stuart, 2010. Broadly speaking, inverse problems can be viewed as SL algorithms where the input and output parameters belong to high or possibly infinite-dimensional Banach spaces. More precisely, a prototypical inverse problem takes the form

$$\mathcal{L}(\boldsymbol{y})\boldsymbol{u} = 0, \qquad \boldsymbol{x} = g(\boldsymbol{u}), \tag{6.8}$$

where $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{u}) \in X \times Y \times U$ with Banach spaces X, Y, U. For any fixed $\boldsymbol{y}$, $\mathcal{L}(\boldsymbol{y})$ is a known map which is assumed to be invertible, although the inverse map need not have a closed form expression; in the Darcy flow example $\mathcal{L}$ is a differential operator. The function $g \colon U \to X$ is the observation map that extracts the "measurements" $\boldsymbol{x}$

from $\boldsymbol{u}$; in the Darcy flow example this map is given by the pointwise evaluation function on the pressure field. Then, solving the inverse problem refers to estimation of $\boldsymbol{y}$ given a fixed observation $\boldsymbol{x}^*$ possibly polluted by some noise. The main challenge in solving inverse problems is *ill-posedness*, i.e., the mapping from $\boldsymbol{y} \mapsto \boldsymbol{x}$ is not stably invertible. Hence, regularization is needed. Note that our notation is at odds with the standard notation in statistics and inverse problems where $\boldsymbol{y}$ often denotes the data and $\boldsymbol{x}$ often denotes the input parameters. We choose this idiosyncratic notation to remain consistent with machine learning notation where $\boldsymbol{x}$ denotes the input data and $\boldsymbol{y}$ denotes the output to be estimated.

A popular method for regularization of inverse problems, and in turn their solution, is to employ Bayes' rule Stuart, 2010 by imposing a prior on $\boldsymbol{y}$ along with a likelihood function extracted from (6.8) leading to a posterior measure on $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$. However, since the spaces X, Y are possibly high or infinite-dimensional and the forward map $\boldsymbol{y} \mapsto \boldsymbol{x}$ is often nonlinear, it is usually not possible to characterize $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ analytically. Thus, sampling techniques such as MCMC are employed to estimate moments and other attributes of the posterior measure, and in turn to characterize the true value of $\boldsymbol{y}$ given $\boldsymbol{x}^*$ and its uncertainty.

In high dimensions, methods such as Metropolis-Hastings require us to solve (6.8) many times to evaluate likelihood functions as part of the accept/reject step, which can be computationally demanding. This is the juncture where MGANs, and similarly the triangular maps of Y. Marzouk et al., 2016, offer an interesting alternative for sampling Bayesian posteriors. One can proceed to generate a fixed number of samples from the prior on $\boldsymbol{y}$, say $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N$ and solve (6.8) to obtain measurements $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$. The pairs $\{\boldsymbol{x}_j, \boldsymbol{y}_j\}_{j=1}^N$ can now be used as training data within the MGANs procedure to obtain a transport map that can push the reference $\eta(\mathrm{d}\boldsymbol{y})$ to the Bayesian posterior $\nu(\mathrm{d}\boldsymbol{y}|\boldsymbol{x}^*)$ at the new observed data $\boldsymbol{x}^*$. Note that we readily employed this approach in the BOD and Darcy flow examples in Section 6.4. While the works Y. Marzouk et al., 2016; Brennan et al., 2020; El Moselhy and Y. M. Marzouk, 2012; Spantini, Baptista, and Y. Marzouk, 2019 explore the use of triangular maps for solving inverse problems, it is interesting to study possible improvements and contributions of the MGANs procedure in this direction.

6.7 The MGAN training procedure

Here we present further details on the MGAN training procedure used in our numerical experiments.

Recall the measure transport problem (6.2) with the divergence D replaced with an appropriate GAN functional Nowozin, Cseke, and Tomioka, 2016

$$\begin{aligned}
&\min_{\mathsf{T}} \sup_{f\in\mathcal{F}} D_{\text{GAN}}(\mathsf{T}, f),\\
&\text{s.t. } \langle \mathsf{T}(\boldsymbol{w}) - \mathsf{T}(\boldsymbol{w}'), \boldsymbol{w} - \boldsymbol{w}'\rangle > 0,\ \forall \boldsymbol{w}, \boldsymbol{w}' \in \operatorname{supp} \eta,\\
&\quad \mathsf{T}(\boldsymbol{x}, \boldsymbol{y}) = \begin{bmatrix} \mathsf{Id}(\boldsymbol{x}) \\ \mathsf{F}(\boldsymbol{x}, \boldsymbol{y}) \end{bmatrix}
\end{aligned}$$

where we have explicitly outlined the parameterization and constraints imposed on T. In practice, we work with either the LSGAN functional, which can be written as

$$D_{\text{GAN}}(\mathsf{T}, f) = \frac{1}{2}\big(D_f(f) + D_{\mathsf{T}}(\mathsf{T})\big)$$

where

$$\begin{aligned}
D_f(f) &= \mathbb{E}_{\boldsymbol{z}\sim\nu}[(f(\boldsymbol{z}) - 1)^2] + \mathbb{E}_{\boldsymbol{w}\sim\eta}[(f(\mathsf{T}(\boldsymbol{w})))^2],\\
D_{\mathsf{T}}(\mathsf{T}) &= \mathbb{E}_{\boldsymbol{w}\sim\eta}[(f(\mathsf{T}(\boldsymbol{w})) - 1)^2],
\end{aligned}$$

or the WGAN-GP functional, which can be written as

$$\begin{aligned}
D_{\text{GAN}}(\mathsf{T}, f) = \mathbb{E}_{\boldsymbol{w}\sim\eta}[f(\mathsf{T}(\boldsymbol{w}))] - \mathbb{E}_{\boldsymbol{z}\sim\nu}[f(\boldsymbol{z})]&\\
+ \gamma\mathbb{E}_{\hat{\boldsymbol{z}}\sim\hat{\nu}}[(\|\nabla_{\hat{\boldsymbol{z}}} f(\hat{\boldsymbol{z}})\|_2 - 1)^2]&.
\end{aligned}$$

In the WGAN-GP functional, the gradient penalty on f is a relaxation of the 1-Lipschitz constraint in the Wasserstein metric. The measure $\hat{\nu} := \text{Law}[\hat{\boldsymbol{z}}]$ where $\hat{\boldsymbol{z}} = \alpha\boldsymbol{z} + (1-\alpha)\mathsf{T}(\boldsymbol{w})$ with $\alpha \sim U[0,1]$, $\boldsymbol{z} \sim \nu$, $\boldsymbol{w} \sim \eta$ and $\alpha, \boldsymbol{z}, \boldsymbol{w}$ are mutually independent. We note that during the alternating optimization procedure for minimizing the WGAN-GP divergence, the map T is taken to be fixed when the gradient penalty on f is estimated. The same is also true for the LSGAN divergence pertaining to the second term in $D_f(f)$. Throughout our numerical experiments we take $\gamma = 10$, as done in the original paper Gulrajani et al., 2017.

The function class $\mathcal{F}$ is currently arbitrary and denotes the space of discriminators; for example one can choose $\mathcal{F} = C(\mathbb{R}^d)$, the space of continuous functions on $\mathbb{R}^d$. We now approximate this optimization problem in three stages.

First, let $\widehat{\mathsf{F}}(\cdot;\boldsymbol{\theta})\colon \mathbb{R}^d \to \mathbb{R}^m$ be a neural network that approximates the component F of the map T, with $\boldsymbol{\theta}$ denoting the network parameters. We define the neural network $\widehat{\mathsf{T}}(\cdot;\boldsymbol{\theta}) : \mathbb{R}^d \to \mathbb{R}^d$ that approximates T by

$$\widehat{\mathsf{T}}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) = \begin{bmatrix} \mathsf{Id}(\boldsymbol{x}) \\ \widehat{\mathsf{F}}(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}) \end{bmatrix}. \tag{6.9}$$

Moreover, we parameterize f as a neural network with weights $\boldsymbol{\theta}'$, thereby taking $\mathcal{F}$ to be the space of functions spanned by the discriminator network architecture for admissible choices of the weights $\boldsymbol{\theta}'$.

Next, we relax the strict monotonicity constraint on T by replacing it with an average monotonicity condition on $\widehat{\mathsf{T}}$. More precisely, we require

$$\mathbb{E}_{\boldsymbol{w}\sim\eta}\mathbb{E}_{\boldsymbol{w}'\sim\eta}\langle \widehat{\mathsf{T}}(\boldsymbol{w};\boldsymbol{\theta}) - \widehat{\mathsf{T}}(\boldsymbol{w}';\boldsymbol{\theta}), \boldsymbol{w} - \boldsymbol{w}'\rangle > 0. \tag{6.10}$$

While this condition does not ensure that $\widehat{\mathsf{T}}$ is monotone everywhere (or at least on the support of η), numerically we find that it is sufficient to ensure that $\widehat{\mathsf{T}}$ is monotone with high probability. In particular, we compute a numerical approximation of

$$\mathbb{P}_{\boldsymbol{w},\boldsymbol{w}'\sim\eta}\big[\langle \widehat{\mathsf{T}}(\boldsymbol{w};\boldsymbol{\theta}) - \widehat{\mathsf{T}}(\boldsymbol{w}';\boldsymbol{\theta}), \boldsymbol{w} - \boldsymbol{w}'\rangle > 0\big], \tag{6.11}$$

that we refer to as the monotonicity probability of the map during the training procedure in our numerical experiments below (see Table 66). We find that tracking this probability is helpful during training as it reveals how $\widehat{\mathsf{T}}$ satisfies the sufficient constraints for the existence of the push-forward conditional measure and is a good indicator of the quality of the trained map.

In the third and final stage of approximation, we replace the expectations in D_{GAN} with empirical averages over training data and mini-batch samples from the reference η as in the standard GAN training procedure Goodfellow et al., 2014. We point out that since the reference η contains the $\boldsymbol{x}$-marginal of $\nu(\boldsymbol{x}, \boldsymbol{y})$, we sample the data each time we sample from η independently from sampling ν. In particular, each time the parameters $(\boldsymbol{\theta}, \boldsymbol{\theta}')$ are updated, three mini-batches are sampled from the training set: one for $\boldsymbol{z} \sim \nu$, one for $\boldsymbol{w} \sim \eta$, and one for $\boldsymbol{w}' \sim \eta$. When using LSGAN, the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$ are updated an equal number of times, while, when using WGAN-GP, $\boldsymbol{\theta}$ is updated once for every five updates of $\boldsymbol{\theta}'$, as is standard practice. Hyperparameters are given in Section 6.9 for each numerical experiment.

6.8 Non-existence of certain transport maps

In this section we consider a simple example where the reference η is supported on a lower dimensional space than the target ν, and show that in this setting there does not exist a transport map pushing η to ν. Let $\eta = N(0,1)$ and $\nu = N(0, I_2)$ and suppose that there exists a continuous map $\mathsf{T} : \mathbb{R} \to \mathbb{R}^2$ such that $\mathsf{T}_\sharp\eta = \nu$. Without loss of generality, we may express T as

$$\mathsf{T}(x) = \begin{bmatrix} \mathsf{T}^{(1)}(x) \\ \mathsf{T}^{(2)}(x) \end{bmatrix}$$

for some $\mathsf{T}^{(1)}, \mathsf{T}^{(2)} \in C(\mathbb{R};\mathbb{R})$. Let $\nu_1 = N(0,1)$, $\nu_2 = N(0,1)$ denote the two marginals of ν. Since, by assumption, $\mathsf{T}_\sharp\eta = \nu$, we must have that $\mathsf{T}^{(1)}_\sharp\eta = \nu_1$. Therefore, by continuity, $\mathsf{T}^{(1)}(x) = \pm x$. By definition of a marginal, $x_2 \sim \nu_2$ is independent of $T^1(x) \sim \nu_1$. However $\mathsf{T}^{(2)}$ depends solely on $x = \pm\mathsf{T}^{(1)}(x)$. Therefore ν_2 must have constant density, which is a contradiction with the fact that, by construction, $\nu_2 = N(0,1)$.

Note that the above construction can be generalized to include discontinuous maps. We show it in the continuous case only for simplicity, as there can exist an infinite number of discontinuous maps T with the property that $\mathsf{T}_\sharp N(0,1) = N(0, I_2)$, which complicates the proof. The choice of the Gaussian is also innocuous. We could have chosen any atomless measure η and set $\nu = \eta \otimes \eta$. In general, even when the target distribution is not a product of one-dimensional marginals, i.e., it has correlations, we still cannot guarantee existence of a transport map as we will always need "more noise" than the reference distribution can provide.

The goal of this simple example is to demonstrate that, in general, we cannot expect the existence of maps pushing low dimensional measures to high dimensional ones. Therefore, GANs and VAEs that employ low-dimensional latent spaces are not well suited for generic SL tasks unless we assume the data lie on a low dimensional manifold. This assumption is hard to check in practice and it becomes unclear how to create consistent algorithms without knowledge of the manifold geometry. We do note, however, that the massive empirical success of GANs and VAEs for image generation indicates that the manifold hypothesis, or some approximation of it, likely holds in imaging tasks.

6.9 Details on the numerical experiments

In this section we give further details regarding the numerical experiments presented in the main article, including the architectures and hyperparameter choices during training. The extra details are presented in the same order as in Section 6.4, with the addition of Section 6.9 where we discuss the average monotonicity constraints and their level of violation within the experiments.

Unless otherwise stated, in all experiments we use the Adam optimizer with the learning rate 2×10^{-4}, and parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$. Apart from Section 6.4 and the image in-painting problems, the remaining examples use three-layer, fully-connected neural networks with hidden layer sizes $256 - 512 - 128$ and the Leaky ReLU non-linearity with parameter $\alpha = 0.2$. We use a batch size of 100 and train for 300 epochs. Apart from image in-painting with the CelebA data set, we use the monotonicity penalty $\lambda = 0.01$ and the LSGAN loss functional. For the CelebA data set, we use the WGAN-GP loss functional with $\lambda = 10^{-4}$.

Synthetic examples

Since this example is two-dimensional, our parameterization (6.9) is automatically triangular and so we expect $\widehat{\mathsf{T}}$ to approximate the KR map as it is a global minimizer of (6.2); further constraints can ensure that the KR map is indeed the unique minimizer Y. Marzouk et al., 2016. Figure 69 shows the true KR map as well as our transport map $\widehat{\mathsf{T}}$ for each of the three problems. Interestingly, we observe that MGAN approximates the true KR map in all three cases.

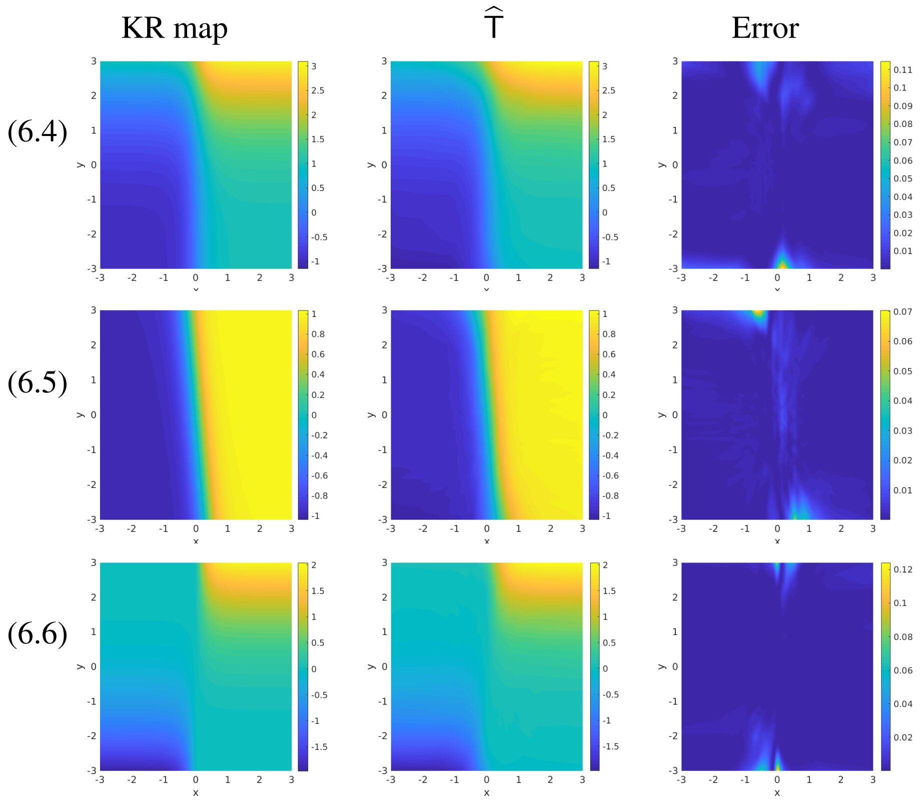


Figure 69: Each row corresponds to the problems (6.4), (6.5), and (6.6), respectively. The first column shows the true KR map on $[-3, 3]^2$, the second column shows our transport map $\widehat{\mathsf{T}}$, and the last column shows the absolute error between them.

Block-triangular versus triangular maps

To demonstrate the effect of constraining the map structure, we consider limited capacity neural networks in this example. We use three-layer, fully-connected neural networks with hidden layer sizes $32 - 64 - 32$ for the block triangular maps and neural networks with hidden layer sizes $22 - 46 - 22$ for each component of the triangular map. In total, the block triangular and triangular maps have about the same number of parameters. Lastly, we train the models with a learning rate of 0.5×10^{-4} and normalize the samples before optimizing the map.

Biochemical oxygen demand model

Following our discussion in Sections 6.6 and 6.6 we note that the BOD problem as presented in Section 6.4 is an instance of a likelihood-free Bayesian inference problem. Indeed the state variable $\mathcal{B}(t)$ is the solution to an ODE with model parameters $\boldsymbol{y} = (A, B)$. The input $\boldsymbol{x}$ is then chosen as noisy observations of the state $\mathcal{B}$ at certain time steps. Thus, in the language of inverse problems, the MGAN sampling procedure amounts to sampling the posterior distribution of the unknown model parameters (A, B) conditioned on limited observations of the state $\mathcal{B}(t)$.

A standard Bayesian procedure might require formulating a likelihood function of the form

$$\Phi(\boldsymbol{x}; \boldsymbol{y}) = (5 \times 10^2) \sum_{j=1}^{5} |x_j - \mathcal{B}(j; \boldsymbol{y})|^2,$$

where we used the notation $\mathcal{B}(j; \boldsymbol{y})$ to highlight the dependence of $\mathcal{B}$ on the parameters (A, B). Then a prior $\pi_0(\boldsymbol{y})$ should be chosen for the unknown parameters, yielding a posterior measure of the form

$$\pi_\nu(\boldsymbol{y}|\boldsymbol{x}^*) \propto \exp(-\Phi(\boldsymbol{x}^*; \boldsymbol{y}))\pi_0(\boldsymbol{y}).$$

In Section 6.4 we chose the prior $\pi_0 = N(0, I_2)$. The main difference between MGANs and standard sampling techniques such as MCMC is that MGANs does not require explicit evaluations of the likelihood function Φ. In the MGAN procedure we simply sample $\boldsymbol{y}_j \sim \pi_0$, and compute $\boldsymbol{x}_j$ by solving the ODE with $\boldsymbol{y}_j$ as the input and adding simulated observational noise to the outputs $\boldsymbol{x}_j$ to construct the training set, the size of which will be limited by our computational budget. Once the training set is available, the training of MGANs and the subsequent sampling of the posterior is done independently of the likelihood function or the ODE model.

Table 64 compares various empirically computed moments of the posterior to those found by the method in Y. Marzouk et al., 2016 which employs triangular transport maps that are parameterized using polynomials.

Table 64: Various moments for the BOD problem. Reported as the mean over the last 10 epochs of training. Closest result to MCMC for each amount of training data is in bold.

Statistic		MCMC	MGAN		Y. Marzouk et al., 2016	
			5,000	50,000	5,000	50,000
Mean	ρ_1	.058	**.050**	**.048**	.090	.034
	ρ_2	.915	**.911**	**.918**	.908	.902
Variance	ρ_1	.170	.153	**.177**	**.180**	.206
	ρ_2	.405	**.368**	**.419**	.490	.457
Skewness	ρ_1	1.81	**1.54**	**1.83**	2.97	1.63
	ρ_2	.681	.489	**.630**	**.707**	.872
Kurtosis	ρ_1	7.37	**5.62**	7.64	29.6	**7.57**
	ρ_2	3.60	**3.21**	**3.19**	16.3	3.88

Darcy flow

The Darcy flow example of Section 6.4 is another example of the likelihood-free solution of a Bayesian inverse problem. The main difference between this example and the BOD example above is the that the input-output map involves the solution of an elliptic PDE. From the likelihood-free inference viewpoint, however, this modification has little effect on the MGAN training procedure beyond generation of the training data. In fact, MGANs can readily be applied to the solution of Bayesian inverse problems with black-box forward maps.

We now present complimentary numerical results for the Darcy flow problem. Table 65 shows further conditional statistics for the three test problems for various sizes of training points in comparison to statistics computed with MCMC as a benchmark. We observe that even with the smallest training set (size $N = 5,000$), the mean and variance statistics are within acceptable range of the MCMC. While larger sample sizes help in most cases we observe a few outliers, especially in the skewness, where even with $N = 100,000$ the MGAN estimates are quite different from MCMC.

Figure 610 shows further kernel density estimates of the conditional measures approximated from MGANs samples versus MCMC samples for different training set sizes. As expected, the quality of the MGAN sample KDE approximation improves as N increases. Another interesting feature is the fact that in the $N = 100,000$ case with ground truth $\boldsymbol{x}_1^*$, the MGAN starts to detect the hard constraint that is imposed by the uniform prior. Note that this constraint appears sharply in the MCMC samples since it is *explicitly* enforced within the procedure; however, this the constraint is not explicitly implemented in the MGAN and is hence difficult to capture, especially in the case of the $\boldsymbol{x}_3^*$ input.

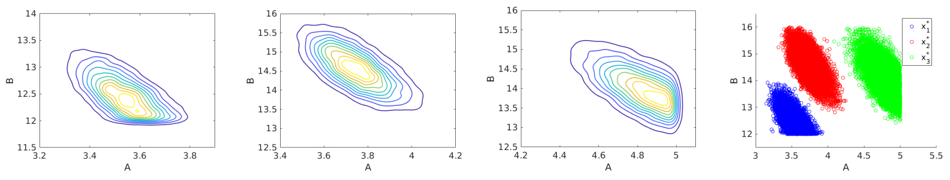
Map type	$\boldsymbol{y}\vert\boldsymbol{x}^*$	Mean		Variance	
		A	B	A	B
MCMC	$\boldsymbol{y}\vert\boldsymbol{x}_1^*$	3.531	12.507	.0103	.1124
	$\boldsymbol{y}\vert\boldsymbol{x}_2^*$	3.748	14.555	.0159	.2565
	$\boldsymbol{y}\vert\boldsymbol{x}_3^*$	4.792	14.059	.0191	.2715
MGAN	$\boldsymbol{y}\vert\boldsymbol{x}_1^*$	3.559	12.580	.0109	.1384
$N = 100000$	$\boldsymbol{y}\vert\boldsymbol{x}_2^*$	3.752	14.538	.0170	.2557
	$\boldsymbol{y}\vert\boldsymbol{x}_3^*$	4.772	14.086	.0216	.3436
MGAN	$\boldsymbol{y}\vert\boldsymbol{x}_1^*$	3.533	12.531	.0102	.1001
$N = 50000$	$\boldsymbol{y}\vert\boldsymbol{x}_2^*$	3.757	14.562	.0135	.2952
	$\boldsymbol{y}\vert\boldsymbol{x}_3^*$	4.784	14.048	.0153	.2991
MGAN	$\boldsymbol{y}\vert\boldsymbol{x}_1^*$	3.593	12.534	.0058	.1394
$N = 5000$	$\boldsymbol{y}\vert\boldsymbol{x}_2^*$	3.738	14.617	.0126	.1863
	$\boldsymbol{y}\vert\boldsymbol{x}_3^*$	4.794	14.053	.0193	.2857

Table 65: Various statistical moments for the conditional distributions in the Darcy flow problem that are estimated using MCMC and MGAN.

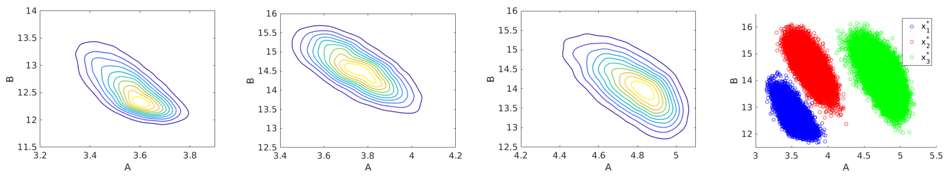
Image in-painting

In this section we present the details of the in-painting experiments in Section 6.4. For MNIST, we follow the simple convolutional architectures of DCGAN Radford, Metz, and Chintala, 2015. We view the digit with the middle part removed $\boldsymbol{x} \in \mathbb{R}^{588}$ as a 28×28 image with its middle 14×14 section taking the value zero as shown in Figure 611. This allows us to directly employ the convolutional architectures since we can view the noise vector $\boldsymbol{y} \in \mathbb{R}^{196}$ as a 14×14 image and embed it into the middle section of $\boldsymbol{x}$, hence viewing the pair $(\boldsymbol{x}, \boldsymbol{y})$ as a 28×28 image. Figure 611 presents examples of in-painting samples from MNIST along with uncertainty estimates for all digits. We bias the presented results towards images with higher variance, i.e., where more than one posterior label is likely to appear.

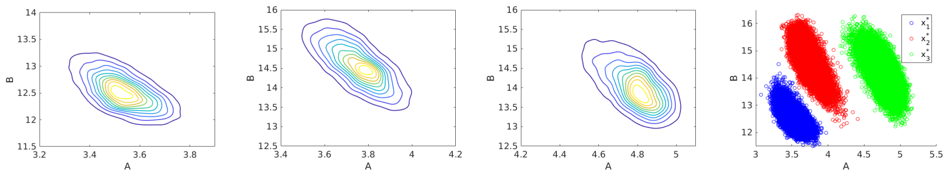
For the CelebA data set, we use the convolutional architectures in Pathak et al., 2016 suitably modified for the right input and output sizes. Since real images are thought to lie on a low dimensional manifold, we define $\eta(\mathrm{d}\boldsymbol{y}) = A_\sharp N(0, I_{100})$ where $A \in \mathbb{R}^{6144\times100}$ is a matrix whose entries we learn along with the neural network parameters $\boldsymbol{\theta}$. We view the noise vector $\boldsymbol{y} \in \mathbb{R}^{6144}$ as a $3 \times 32 \times 64$ image and concatenate it along the channel dimension of the input top half image $\boldsymbol{x} \in \mathbb{R}^{3\times32\times64}$ and hence view the pair $(\boldsymbol{x}, \boldsymbol{y})$ as a $6 \times 32 \times 64$ image. Furthermore, during training, we view the $\boldsymbol{x}$-marginal of $\nu(\mathrm{d}\boldsymbol{x}, \mathrm{d}\boldsymbol{y})$ as $\nu(\mathrm{d}\boldsymbol{x}) + N(0, (0.05)^2 I_{6114})$, i.e., we add a



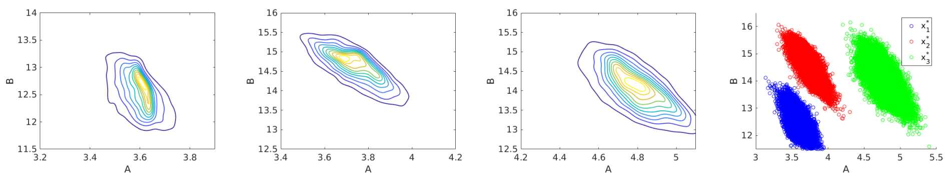
(a) MCMC $\boldsymbol{y}|\boldsymbol{x}_1^*$ (b) MCMC $\boldsymbol{y}|\boldsymbol{x}_2^*$ (c) MCMC $\boldsymbol{y}|\boldsymbol{x}_3^*$ (d) MCMC samples



(e) $\mathsf{F}_\sharp(\boldsymbol{x}_1^*,\cdot)\eta$ (f) $\mathsf{F}_\sharp(\boldsymbol{x}_2^*,\cdot)\eta$ (g) $\mathsf{F}_\sharp(\boldsymbol{x}_3^*,\cdot)\eta$ (h) F samples



(i) $\mathsf{F}_\sharp(\boldsymbol{x}_1^*,\cdot)\eta$ (j) $\mathsf{F}_\sharp(\boldsymbol{x}_2^*,\cdot)\eta$ (k) $\mathsf{F}_\sharp(\boldsymbol{x}_3^*,\cdot)\eta$ (l) F samples



(m) $\mathsf{F}_\sharp(\boldsymbol{x}_1^*,\cdot)\eta$ (n) $\mathsf{F}_\sharp(\boldsymbol{x}_2^*,\cdot)\eta$ (o) $\mathsf{F}_\sharp(\boldsymbol{x}_3^*,\cdot)\eta$ (p) F samples

Figure 610: KDEs of the conditionals $\boldsymbol{y}|\boldsymbol{x}_j^*$ for the Darcy flow problem with $j = 1, 2, 3$. Each density is obtained by using 30,000 representative samples shown in the last column. The first row represents the ground truth obtained by MCMC. Each subsequent row shows the MGAN results when training with $N = 100,000$, $N = 50,000$, and $N = 5,000$ data samples respectively.

small amount of Gaussian noise to each input image. This is a similar mechanism to the one employed in Karras, Laine, and Aila, 2019. We have found that it helps significantly in battling conditional mode collapse.

Average monotonicity violations

Our final set of results pertain to all of the numerical experiments above. Table 66 summarizes an approximation to the monotonicity probabilities of the trained MGANs in each experiment computed over samples from the reference. These probabilities were tracked during training using the mini-batch samples and averaged at each epoch. Here we report only the estimates computed at the final epoch as they constitute the approximate monotonicity probability of the trained model.We observe that even though the monotonicity constraint is only imposed in expectation,

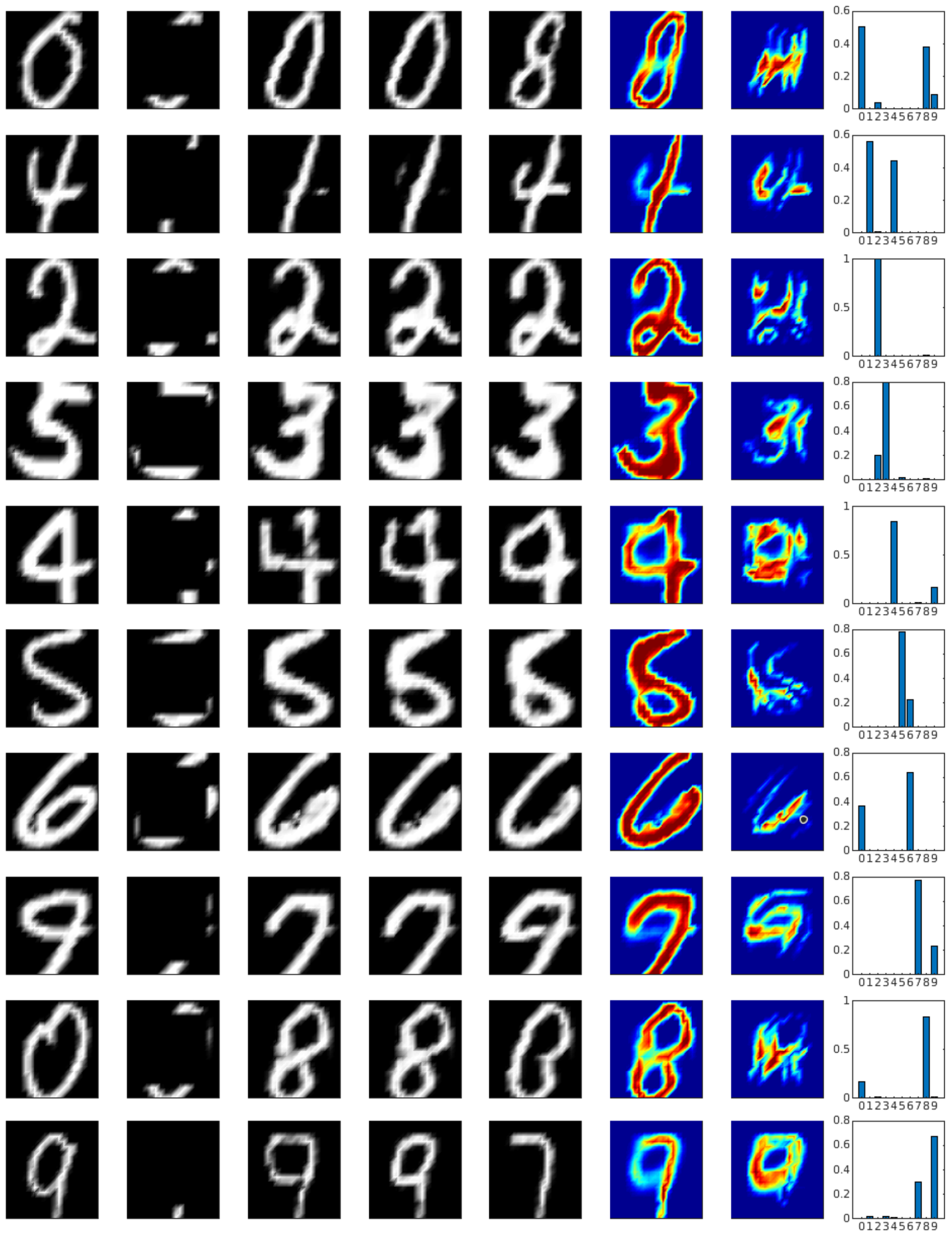


Figure 611: Example in-painted images using MGANs. The first column shows the ground truth images. The second column shows $\boldsymbol{x}^*$, the image to be in-painted. The third, fourth, and fifth columns show random in-paintings generated from the conditional $\boldsymbol{y}|\boldsymbol{x}^*$. The sixth and seventh columns show the pixel-wise conditional mean and variance computed from 1000 samples. The last column shows the label probabilities of samples from $\boldsymbol{y}|\boldsymbol{x}^*$ classified using LeNet Lecun et al., 1998.

it still holds with high probability for the trained MGAN. The BOD experiment has the lowest monotonicity probability at 89.36% while the Darcy flow example has

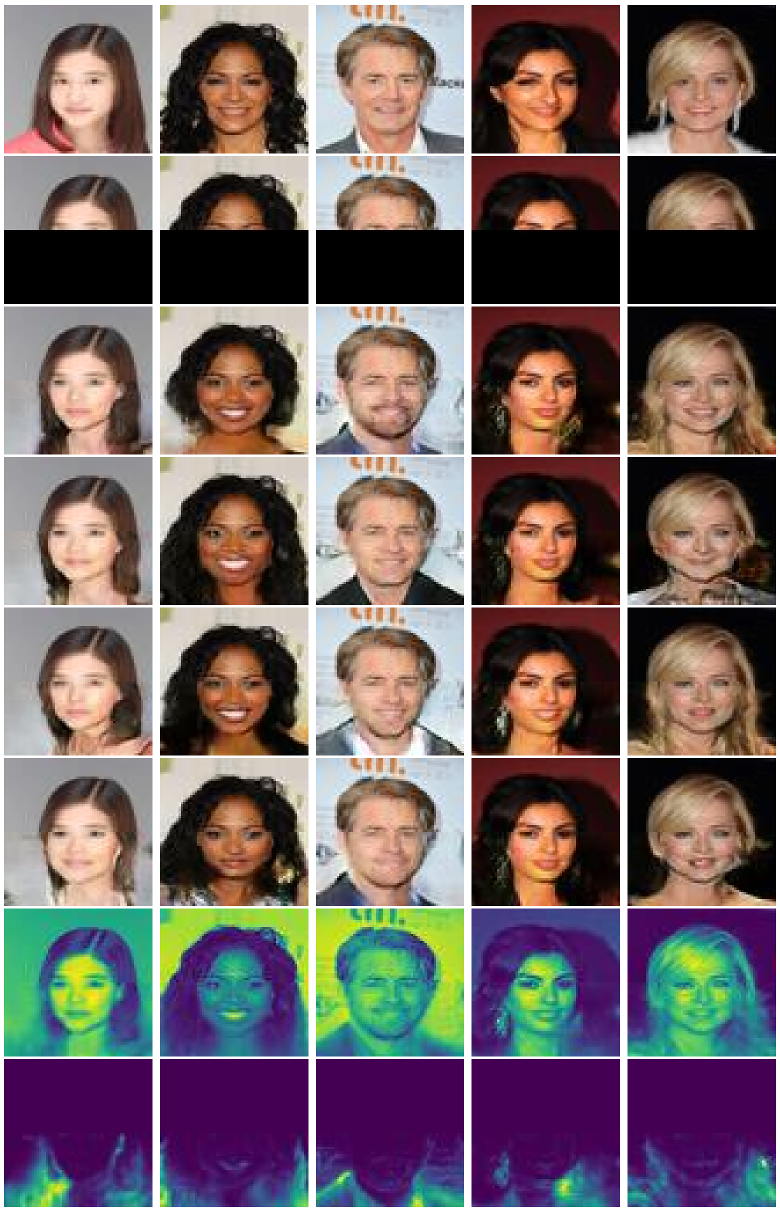


Figure 612: Example in-paintings using MGAN from the CelebA test set. First row depicts the ground truth image, second row shows the observed image $\boldsymbol{x}^*$ while the next four columns are random samples from the conditional $\boldsymbol{y}|\boldsymbol{x}^*$. The bottom two rows show the pointwise means and variances of the intensities of the conditional samples generated by the MGAN.

the highest probability at 100%.

Further theoretical and numerical analysis of this expected monotonicity constraint is needed to better understand the reason for this good performance of the trained maps and also to understand the regularizing effects of this constraint in providing stability to the trained neural networks.

Problem	Monotonicity Probability
(6.4)	95.83%
(6.5)	92.71%
(6.6)	98.60%
BT favorable	99.96%
BT reverse	99.95%
T favorable	95.23%
T reverse	99.77%
BOD	89.36%
Darcy	100.0%
MNIST	98.84%
CelebA	99.99%

Table 66: Final approximate probability that $\widehat{\mathsf{T}}$ is monotone for each problem. We denote block triangular and triangular maps by BT and T for the example in Section 6.4, respectively.

References

Adler, Jonas and Ozan Öktem (2018). “Deep Bayesian inversion”. In: *arXiv preprint:1811.05910*.

Ambrogioni, Luca et al. (2017). “The kernel mixture network: A nonparametric method for conditional density estimation of continuous random variables”. In: *arXiv preprint:1705.07111*.

Arbel, Michael and Arthur Gretton (2018). “Kernel Conditional Exponential Family”. In: *International Conference on Artificial Intelligence and Statistics*.

Ardizzone, Lynton, Jakob Kruse, et al. (2018). “Analyzing Inverse Problems with Invertible Neural Networks”. In: *International Conference on Learning Representations*.

Ardizzone, Lynton, Carsten Lüth, et al. (2019). “Guided image generation with conditional invertible neural networks”. In: *arXiv preprint:1907.02392*.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). “Wasserstein generative adversarial networks”. In: *International Conference on Machine Learning*, pp. 214–223.

Bardsley, Johnathan M et al. (2014). “Randomize-then-optimize: A method for sampling from posterior distributions in nonlinear inverse problems”. In: *SIAM Journal on Scientific Computing* 36.4, A1895–A1910.

Belghazi, Mohamed Ishmael et al. (2019). “Learning about an exponential amount of conditional distributions”. In: *Advances in Neural Information Processing Systems*.

Bishop, Christopher M (1994). *Mixture density networks*. Tech. rep. NCRG/94/004. Aston University, Department of Computer Science and Applied Mathematics.

Brennan, Michael et al. (2020). “Greedy inference with structure-exploiting lazy maps”. In: *Advances in Neural Information Processing Systems*.

Cranmer, Kyle, Johann Brehmer, and Gilles Louppe (2020). “The frontier of simulation-based inference”. In: *Proceedings of the National Academy of Sciences* 117.48, pp. 30055–30062.

Doersch, Carl (2016). “Tutorial on variational autoencoders”. In: *arXiv preprint:1606.05908*.

El Moselhy, T. A. and Y. M. Marzouk (2012). “Bayesian inference with optimal maps”. In: *Journal of Computational Physics* 231.23, pp. 7815–7850.

Fox, Charles W and Stephen J Roberts (2012). “A tutorial on variational Bayesian inference”. In: *Artificial intelligence review* 38.2, pp. 85–95.

Gelman, Andrew et al. (2013). *Bayesian data analysis*. CRC press.

Ghosal, Subhashis, Jayanta K Ghosh, Aad W Van Der Vaart, et al. (2000). “Convergence rates of posterior distributions”. In: *Annals of Statistics* 28.2, pp. 500–531.

Giné, Evarist and Richard Nickl (2016). *Mathematical foundations of infinite-dimensional statistical models*. Vol. 40. Cambridge University Press.

Goodfellow, Ian et al. (2014). “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*.

Grelaud, Aude et al. (2009). “ABC likelihood-free methods for model choice in Gibbs random fields”. In: *Bayesian Analysis* 4.2, pp. 317–335.

Gressmann, Frithjof et al. (2019). “Probabilistic supervised learning”. In: *arXiv preprint:1801.00753*.

Gui, Jie et al. (2020). “A review on generative adversarial networks: Algorithms, theory, and applications”. In: *arXiv preprint: 2001.06937*.

Gulrajani, Ishaan et al. (2017). “Improved training of Wasserstein GANs”. In: *International Conference on Neural Information Processing Systems*.

Gutmann, Michael U and Jukka Corander (2016). “Bayesian optimization for likelihood-free inference of simulator-based statistical models”. In: *The Journal of Machine Learning Research* 17.1, pp. 4256–4302.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Heusel, Martin et al. (2017). “GANs trained by a two time-scale update rule converge to a local Nash equilibrium”. In: *International Conference on Neural Information Processing Systems*.

Iglesias, M. A., K. Lin, and A. M. Stuart (2014). “Well-posed Bayesian geometric inverse problems arising in subsurface flow”. In: *Inverse Problems* 30.11, p. 114001.

Isola, Phillip et al. (2017). “Image-to-image translation with conditional adversarial networks”. In: *IEEE conference on computer vision and pattern recognition*, pp. 1125–1134.

Ivanov, O, M Figurnov, and D Vetrov (2019). “Variational autoencoder with arbitrary conditioning”. In: *International Conference on Learning Representations*.

Jaini, Priyank, Kira A Selby, and Yaoliang Yu (2019). “Sum-of-squares polynomial flow”. In: *International Conference on Machine Learning*. PMLR, pp. 3009–3018.

Kabanikhin, S. I. (2011). *Inverse and Ill-posed Problems: Theory and Applications*. De Gruyter.

Kaipio, J. and E. Somersalo (2005). *Statistical and Computational Inverse Problems*. Springer Science & Business Media.

Karras, Tero, Samuli Laine, and Timo Aila (2019). “A style-based generator architecture for generative adversarial networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Kobyzev, Ivan, Simon Prince, and Marcus Brubaker (2020). “Normalizing flows: An introduction and review of current methods”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Lecun, Yann et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*, pp. 2278–2324.

Lin, Jianhua (1991). “Divergence measures based on the Shannon entropy”. In: *IEEE Transactions on Information theory* 37.1, pp. 145–151.

Lindgren, Erik M, Jay Whang, and Alexandros G Dimakis (2020). “Conditional sampling from invertible generative models with applications to inverse problems”. In: *arXiv preprint:2002.11743*.

Mao, Xudong et al. (2017). “Least squares generative adversarial networks”. In: *IEEE international conference on computer vision*.

Marzouk, Youssef et al. (2016). “Sampling via measure transport: An introduction”. In: *Handbook of Uncertainty Quantification*, pp. 1–41.

Mirza, Mehdi and Simon Osindero (2014). “Conditional generative adversarial nets”. In: *arXiv preprint:1411.1784*.

Nowozin, Sebastian, Botond Cseke, and Ryota Tomioka (2016). “f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization”. In: *Advances in Neural Information Processing Systems 29*.

Papamakarios, George and Iain Murray (2016). “Fast ε-free inference of simulation models with Bayesian conditional density estimation”. In: *Advances in Neural Information Processing Systems*.

Papamakarios, George, Eric Nalisnick, et al. (2019). “Normalizing Flows for Probabilistic Modeling and Inference”. In: *arXiv preprint:1912.02762*.

Papamakarios, George, Theo Pavlakou, and Iain Murray (2017). “Masked Autoregressive Flow for Density Estimation”. In: *Advances in Neural Information Processing Systems*.

Papamakarios, George, David Sterratt, and Iain Murray (2019). “Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows”. In: *International Conference on Artificial Intelligence and Statistics*.

Parno, Matthew D and Youssef M Marzouk (2018). “Transport map accelerated Markov chain Monte Carlo”. In: *SIAM/ASA Journal on Uncertainty Quantification* 6.2, pp. 645–682.

Pathak, Deepak et al. (2016). “Context encoders: Feature learning by in-painting”. In: *IEEE conference on computer vision and pattern recognition*.

Radford, Alec, Luke Metz, and Soumith Chintala (2015). “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint:1511.06434*.

Rezende, Danilo and Shakir Mohamed (2015). “Variational inference with normalizing flows”. In: *International Conference on Machine Learning*.

Robert, Christian and George Casella (2013). *Monte Carlo statistical methods*. Springer Science & Business Media.

Rothfuss, Jonas et al. (2019). “Conditional Density Estimation with Neural Networks: Best Practices and Benchmarks”. In: *arXiv preprint:1903.00954*.

Santambrogio, Filippo (2015). *Optimal transport for applied mathematicians*. Springer.

Shiga, Motoki, Voot Tangkaratt, and Masashi Sugiyama (2015). “Direct conditional probability density estimation with sparse feature selection”. In: *Machine Learning* 100.2-3, pp. 161–182.

Siahkoohi, Ali et al. (2021). “Preconditioned training of normalizing flows for variational inference in inverse problems”. In: *arXiv preprint arXiv:2101.03709*.

Spantini, Alessio, Ricardo Baptista, and Youssef Marzouk (2019). “Coupling techniques for nonlinear ensemble filtering”. In: *arXiv preprint:1907.00389*.

Stuart, A. M. (2010). “Inverse problems: a Bayesian perspective”. In: *Acta Numerica* 19, pp. 451–559.

Villani, Cédric (2009). *Optimal transport: Old and new*. Springer.

Zeidler, Eberhard (2013). *Nonlinear Functional Analysis and Its Applications: II/B: Nonlinear Monotone Operators*. Springer Science & Business Media.

Zhu, Jun-Yan et al. (2017). “Toward Multimodal Image-to-Image Translation”. In: *Advances in Neural Information Processing Systems*.

Chapter 7

REGRESSION-CLUSTERING FOR MOLECULAR PREDICTIONS

7.1 Introduction

Machine-learning (ML) continues to emerge as a versatile strategy in the chemical sciences, with applications to drug discovery (Lavecchia, 2015; Gawehn, Hiss, and Schneider, 2016; Popova, Olexandr Isayev, and Tropsha, 2018; Kearnes et al., 2016; Mater and Coote, 2019), materials design (Kim et al., 2017; Ren et al., 2018; Butler et al., 2018; Sanchez-Lengeling and Aspuru-Guzik, 2018; Mater and Coote, 2019), and reaction prediction (Wei, Duvenaud, and Aspuru-Guzik, 2016; Raccuglia et al., 2016; Ulissi et al., 2017; Segler and Waller, 2017; Segler, Preuss, and Waller, 2018; Mater and Coote, 2019). An increasing number of ML methods have focused on the prediction of molecular properties, including quantum mechanical electronic energies (J. S. Smith, O. Isayev, and Roitberg, 2017; Justin S Smith et al., 2019; Lubbers, Justin S. Smith, and Barros, 2018; Bartók et al., 2010; Rupp et al., 2012; Montavon, Rupp, Gobre, Vazquez-Mayagoitia, Hansen, Tkatchenko, Müller, and von Lilienfeld, 2013; Hansen et al., 2013; Ramakrishnan, Dral, et al., 2015; Jörg Behler, 2016; Paesani, 2016; Schütt et al., 2017; Wu et al., 2018; Nguyen et al., 2018; Fujikake et al., 2018; Li et al., 2018; Zhang et al., 2018; Nandy et al., 2018), densities, and spectra (Ramakrishnan, Hartmann, et al., 2015; Gastegger, Jörg Behler, and Marquetand, 2017; Yao et al., 2018; Anders S. Christensen, Felix A. Faber, and O. Anatole von Lilienfeld, 2019; Ghosh et al., 2019). Most of this work has focused on ML in the representation of atom- or geometry-specific features, although more abstract representations are gaining increased attention(Brockherde et al., 2017; McGibbon et al., 2017; Nudejima et al., 2019; Townsend and Vogiatzis, 2019; Welborn, Cheng, and Miller, 2018; Cheng et al., 2019).

We recently introduced a rigorous factorization of the post-Hartree-Fock correlation energy into contributions from pairs of occupied molecular orbitals and showed that these pair contributions could be compactly represented in the space of molecular-orbital-based (MOB) features to allow for straightforward ML regression (Welborn, Cheng, and Miller, 2018; Cheng et al., 2019). This MOB-ML method was demonstrated to accurately predict second-order Møller-Plessett perturbation theory (MP2)(Møller and Plesset, 1934; Saebo and Pulay, 1993) and coupled cluster with singles, doubles and perturbative triples (CCSD(T))(Bartlett et al., 1990; Schütz, 2000) energies of different benchmark systems, including the QM7b-T and GDB-13-T datasets of thermalized drug-like organic molecules. While providing good accuracy with a modest amount of training data, the accuracy of MOB-ML in these initial studies was limited by the high computational cost ($\mathcal{O}(N^3)$) of applying Gaussian Process Regression (GPR) to the full set of training data (Cheng et al., 2019).

In this work, we combine MOB-ML with regression clustering (RC) to overcome this bottleneck in computational cost and accuracy. The training data are clustered via RC to discover locally linear structures. By independently regressing these subsets of the data, we obtain MOB-ML models with greatly reduced training costs while preserving prediction accuracy and transferability.

7.2 Theory

Molecular-orbital based machine learning (MOB-ML)

The MOB-ML method is based on the observation that the correlation energy for any post-Hartree-Fock wavefunction theory can be exactly decomposed as a sum over occupied molecular orbitals (MOs) via Nesbet's theorem (Nesbet, 1958; Szabo and Ostlund, 1996),

$$E_{\mathrm{c}} = \sum_{ij}^{\mathrm{occ}} \epsilon_{ij}, \tag{7.1}$$

where E_c is the correlation energy and ϵ_{ij} is the pair correlation energy corresponding to occupied MOs i and j. The pair correlation energies can be expressed as a functional of the set of (occupied and unoccupied) MOs, appropriately indexed by i and j, such that

$$\epsilon_{ij} = \epsilon\left[\{\phi_p\}^{ij}\right]. \tag{7.2}$$

The functional ϵ maps the Hartree-Fock MOs to the pair correlation energy, regardless of the molecular composition or geometry, such that it is a universal functional for all chemical systems. To bypass the expensive post-Hartree-Fock evaluation procedure, MOB-ML approximates ϵ_{ij} by machine learning two functionals, $\epsilon_{\mathrm{d}}^{\mathrm{ML}}$ and $\epsilon_{\mathrm{o}}^{\mathrm{ML}}$, which correspond to diagonal and off-diagonal terms of the sum in Eq. 7.1.

$$\epsilon_{ij} \approx \begin{cases} \epsilon_{\mathrm{d}}^{\mathrm{ML}}\left[\mathbf{f}_i\right] & \text{if } i = j \\ \epsilon_{\mathrm{o}}^{\mathrm{ML}}\left[\mathbf{f}_{ij}\right] & \text{if } i \neq j \end{cases} \tag{7.3}$$

The MOB-ML feature vectors $\mathbf{f}_i$ and $\mathbf{f}_{ij}$ are comprised of unique elements of the Fock, Coulomb and exchange matrices between ϕ_i, ϕ_j, and the set of virtual orbitals. Without loss of generality, we perform MOB-ML using localized MOs (LMOs) to improve transferability across chemical systems.(Welborn, Cheng, and Miller, 2018) Detailed descriptions of feature design are provided in our previous work (Cheng et al., 2019; Welborn, Cheng, and Miller, 2018), and the features employed here are unchanged from those detailed in Ref. (Cheng et al., 2019).

Local linearity of MOB feature space

It has been previously emphasized that MOB-ML facilitates transferability across chemical systems, even allowing for predictions involving molecules with elements that do not appear in the training set (Welborn, Cheng, and Miller, 2018), due to the fact that MOB features provide a compact and highly abstracted representation of the electronic structure. However, it is worth additionally emphasizing that this transferability benefits from the smooth variation and local linearity of the pair correlation energies as a function of MOB feature values associated with different molecular geometries and even different molecules.

Figure 71 illustrates these latter properties for a σ-bonding orbital in a series of simple molecules. On the y-axis, we plot the diagonal contribution to the correlation energy associated with this orbital (ϵ_{ii}), computed at the MP2/cc-pvTZ level of theory. On the x-axis, we plot the value of a particular MOB feature, the Fock matrix element for the that localized orbital, F_{ii}. For each molecule, a range of geometries is sampled from the Boltzmann distribution at 350 K, with each plotted point corresponding to a different sampled geometry.

It is immediately clear from the figure that the pair correlation energy varies smoothly and linearly as a function of the MOB feature value. Moreover, the slope of the linear curve is remarkably consistent across molecules. This illustration suggests that MOB features may lead to accurate regression of correlation energies using simple machine learning models (even linear models), and it also indicates the basis for the robust transferability of MOB-ML across diverse chemical systems, including those with elements that do not appear in the training set.

Regression clustering with a greedy algorithm

To take advantage of the local linearity of pair correlation energies as a function of MOB features, we propose a strategy to discover optimally linear clusters using regression clustering (RC).Späth (1979b) consider the set of M datapoints $\{\mathbf{f}_t, \epsilon_t\}$

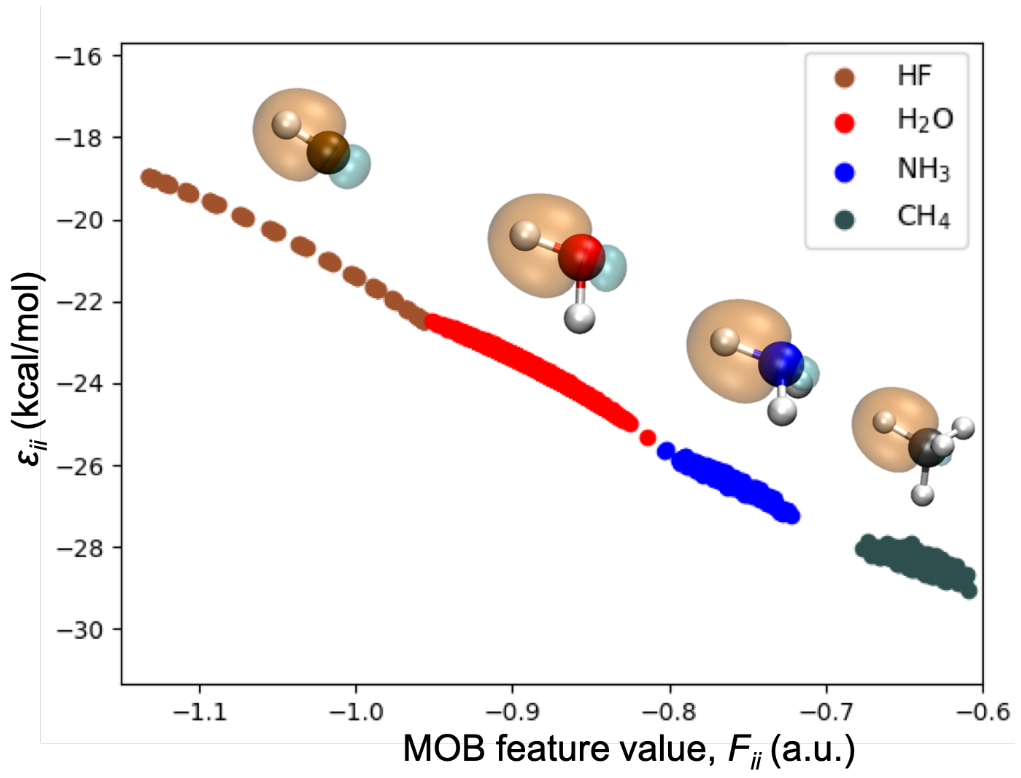


Figure 71: The diagonal pair correlation energy (ϵ_{ii}) for a localized σ-bond in four different molecules at thermally sampled geometries (at 350 K), computed at the MP2/cc-pvTZ level of theory. The diagonal pair correlation energies for HF, NH_3, and CH_4 are shifted vertically downward relative to those of HF by 3.407, 6.289, and 7.772 kcal/mol for H_2O, NH_3 and CH_4. Illustrative σ-bond LMOs are shown for each molecule.

$\subset \mathbb{R}^d \times \mathbb{R}$, where d is the length of the MOB feature vector and where each datapoint is indexed by t and corresponds to a MOB feature vector and the associated reference value (i.e., label) for the pair correlation energy. To separate these datapoints into locally linear clusters, $S_1, \ldots, S_N$, we seek a solution to the optimization problem

$$\min_{S_1,\ldots,S_N} \sum_{k=1}^{N} \sum_{t \in S_k} |A(S_k) \cdot \mathbf{f}_t + b(S_k) - \epsilon_t|^2 \tag{7.4}$$

where $A(S_k) \in \mathbb{R}^d$ and $b(S_k) \in \mathbb{R}$ are obtained via ordinary least squares (OLS) solution,

$$\begin{bmatrix} \mathbf{f}_{t_1}^T & 1 \\ \vdots & \vdots \\ \mathbf{f}_{t_{|S_k|}}^T & 1 \end{bmatrix} \begin{bmatrix} A(S_k) \\ b(S_k) \end{bmatrix} = \begin{bmatrix} \epsilon_{t_1} \\ \vdots \\ \epsilon_{t_{|S_k|}} \end{bmatrix}. \tag{7.5}$$

Each resulting S_k is the set of indices t assigned to cluster k comprised of $|S_k|$ datapoints. To perform the optimization in Eq. 7.4, we employ a modified version of the greedy algorithm proposed in Ref. (Späth, 1979a) (Algorithm 1). In general, solutions to Eq. 7.4 may overlap, such that $S_k \cap S_l \neq \emptyset$ for $k \neq l$; however, the proposed algorithm enforces that clusters remain pairwise-disjoint.

```
**Input:** Initial clusters: $S_1, \ldots, S_N$
**Output:** Data clusters $S_1, \ldots, S_N$
for $k \leftarrow 1$ to $N$ do
    $A(S_k), b(S_k) \leftarrow$ OLS solution of Eq. 7.5;
end
while not converged do
    for $k \leftarrow 1$ to $N$ do
        $S_k \leftarrow \{t \in \{1, \ldots, M\} : \arg\min_{n \in \{1,\ldots,N\}} |A(S_n) \cdot \mathbf{f}_t + b(S_n) - \epsilon_t|^2 = k\}$
    end
    for $k \leftarrow 1$ to $N$ do
        $A(S_k), b(S_k) \leftarrow$ OLS solution of Eq. 7.5
    end
end
```

Algorithm 1: Greedy algorithm for the solution of Eq. 7.4.

Algorithm 1 has a per-iteration runtime of $\mathcal{O}(Md^2)$, since we compute N OLS solutions each with runtime $\mathcal{O}(|S_k|d^2)$ and since $\sum_{k=1}^{N} |S_k| = M$. However, the algorithm can be trivially parallelized to reach a runtime of $\mathcal{O}(\max(|S_k|)d^2)$. A key operational step in this algorithm is line 1, which can be explained in simple terms as follows: we assign each datapoint, indexed by t, to the cluster to which it is closest, as measured by the squared linear regression distance metric,

$$|D_{n,t}|^2 = |A(S_n) \cdot \mathbf{f}_t + b(S_n) - \epsilon_t|^2 \tag{7.6}$$

where $D_{n,t}$ is the distance of this point to cluster n. In principle, a datapoint could be equidistant to two or more different clusters by this metric; in such cases, we randomly assign the datapoint to only one of those equidistant clusters to enforce the pairwise-disjointness of the resulting clusters. Convergence of the greedy algorithm is measured by the decrease in the objective function of Eq. 7.4.

Figure 72 illustrates RC in a simple one-dimensional example for which unsupervised clustering approaches will fail to reveal the underlying linear structure. To create two clusters of nearly linear data that overlap in feature space, the interval of feature values on $[0, 1]$ is uniformly discretized, such that $\mathbf{f}_t = (t - 1)/(M - 1)$

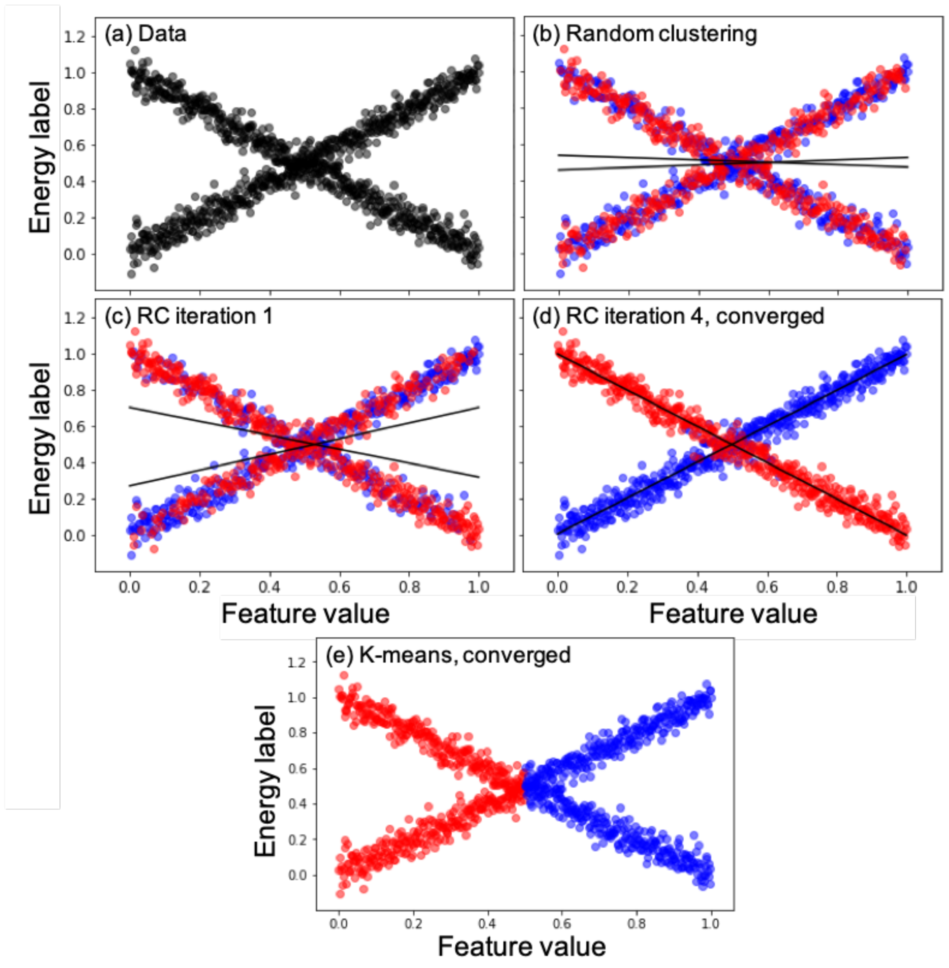


Figure 72: Comparison of clustering algorithms for (a) a dataset comprised of two cluster of nearly linear data that overlap in feature space, using (b-d) RC and (e) standard K-means clustering. (b) Random initialization of the clusters for the greedy algorithm, with datapoint color indicating cluster assignment. (c) Cluster assignments after one iteration of the greedy algorithm. (d) Converged cluster assignments after four iterations of the greedy algorithm. For panels (b-d), two linear regression lines at each iteration are shown in black. (e) Converged cluster assignments obtained using K-means clustering, which fails to reveal the underlying linear structure of the clusters.

for $t = 1, \ldots, M$. Then, $M/2$ of the feature values are randomly chosen without replacement for cluster S_1 while the remainder are placed in S_2; the energy labels associated with each feature value are then generated using

$$\epsilon_t = \mathbf{f}_t + \xi_{t,1}, \quad t \in S_1$$

and

$$\epsilon_t = -\mathbf{f}_t + 1 + \xi_{t,2}, \quad t \in S_2$$

where $\xi_{t,k} \sim \mathcal{N}(0, 0.1^2)$ is an i.d.d. sequence. The resulting dataset is shown in Fig. 72a.

Application of the RC method to this example is illustrated in Figs. 72(b-d). The greedy algorithm is initialized by randomly assigning each datapoint to either S_1 or S_2 (Fig. 72b). Then, with only a small number of iterations (Figs. 72c and d), the algorithm converges to clusters that reflect the underlying linear character. For comparison, Fig. 72e shows the clustering that is obtained upon convergence of the standard K-means algorithm,(Lloyd, 1982) initialized with random cluster assignments. Unlike RC, the K-means algorithm prioritizes the compactness of clusters, resulting in a final clustering that is far less amenable to simple regression. While we recognize that the correct clustering could potentially be obtained using K-means when the dimensions of $\mathbf{f}_t$ and ϵ_t are comparable, this is not the case for MOB-ML applications since $\mathbf{f}_t$ is typically at least 10-dimensional and ϵ_t is a scalar; the RC approach does not suffer from this issue. Finally, we have confirmed that initialization of RC from the clustering in Fig. 72e rapidly returns to the results in Fig. 72d, requiring only a couple of iterations of the greedy algorithm.

7.3 Calculation Details

Results are presented for QM7b-T,(Cheng et al., 2019) a thermalized version of the QM7b set(Montavon, Rupp, Gobre, Vazquez-Mayagoitia, Hansen, Tkatchenko, Müller, and O Anatole von Lilienfeld, 2013) of 7211 molecules with up to seven C, O, N, S, and Cl heavy atoms, as well as for GDB-13-T,(Cheng et al., 2019) a thermalized version of the GDB-13 set(Blum and Reymond, 2009) of molecules with thirteen C, O, N, S, and Cl heavy atoms. The MOB-ML features employed in the current study are identical to those previously provided.(Cheng et al., 2019) Reference pair correlation energies are computed using MP2(Møller and Plesset, 1934) and using CCSD(T).(Bartlett et al., 1990; Schütz, 2000) The MP2 reference data were obtained with the cc-pVTZ basis set,(Dunning, 1989) whereas the CCSD(T) data

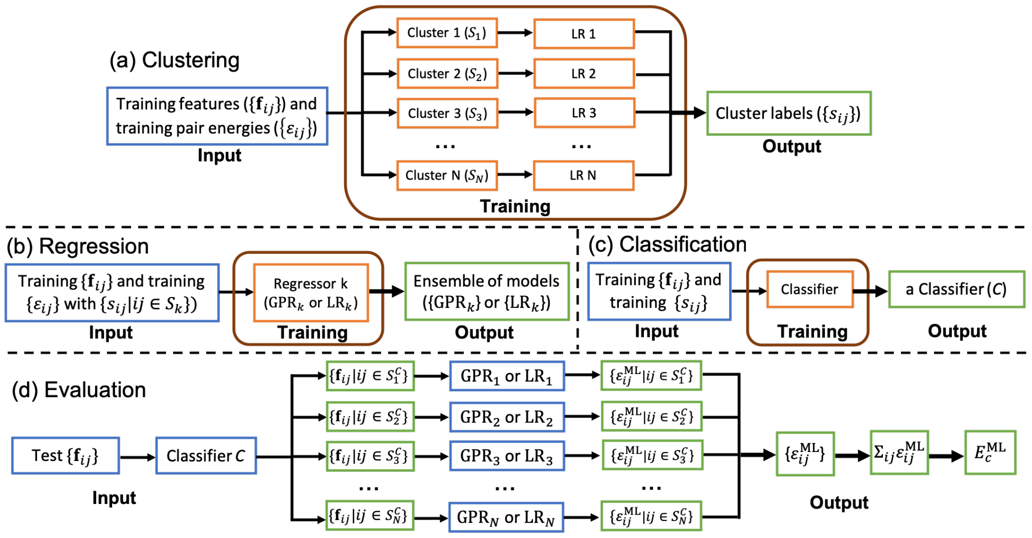

(a) Clustering
Training features ($\{\mathbf{f}_{ij}\}$) and training pair energies ($\{\epsilon_{ij}\}$)
Input
Cluster 1 (S_1)
Cluster 2 (S_2)
Cluster 3 (S_3)
Cluster N (S_N)
LR 1
LR 2
LR 3
LR N
Training
Cluster labels ($\{s_{ij}\}$)
Output
(b) Regression
Training $\{\mathbf{f}_{ij}\}$ and training $\{\epsilon_{ij}\}$ with $\{s_{ij}|ij \in S_k\}$
Input
Regressor k (GPR$_k$ or LR$_k$)
Training
Ensemble of models ($\{$GPR$_k\}$ or $\{$LR$_k\}$)
Output
(c) Classification
Training $\{\mathbf{f}_{ij}\}$ and training $\{s_{ij}\}$
Input
Classifier
Training
a Classifier (C)
Output
(d) Evaluation
Test $\{\mathbf{f}_{ij}\}$
Input
Classifier C
GPR$_1$ or LR$_1$
GPR$_2$ or LR$_2$
GPR$_3$ or LR$_3$
GPR$_N$ or LR$_N$
$\{\epsilon_{ij}^{\mathrm{ML}}\}$
$\Sigma_{ij}\epsilon_{ij}^{\mathrm{ML}}$
E_c^{ML}
Output

Figure 73: The MOB-ML clustering/regression/classification workflow. (a) Clustering of the training dataset of MOB-ML feature vectors and energy labels using RC to obtain optimized linear clusters and to provide the cluster labels for the feature vectors. (b) Regression of each cluster of training data (using LR or GPR), to obtain the ensemble of cluster-specific regression models. (c) Training a classifier (RFC) from the MOB-ML feature vectors and cluster labels for the training data. (d) Evaluating the predicted MOB-ML pair correlation energy from a test feature vector is performed by first classifying the feature vector into one of the clusters, and then evaluating the cluster-specific regression model. In each panel, blue boxes indicate input quantities, orange boxes indicate training intermediates, and green boxes indicate the resulting labels, models, and pair correlation energy predictions.

were obtained using the cc-pVDZ basis set.(Dunning, 1989) All employed training and test datasets are provided in Ref. (Cheng et al., 2019).

Regression Clustering (RC)

RC is performed using the ordinary least square linear regression implementation in the SCIKIT-LEARN package (Pedregosa et al., 2011). Unless otherwise specified, we initialize the greedy algorithm from the results of K-means clustering, also implemented in SCIKIT-LEARN; K-means initialization was found to improve the subsequent training of the random forest classifier (RFC) in comparison to random initialization. It is found that neither L1 nor L2 regularization had significant effect on the rate of convergence of the greedy algorithm, so neither is employed in the results presented here. It is found that a convergence threshold of 10^{-8} kcal^2/mol^2 for the loss function of the greedy algorithm (Eq. 7.4) leads to no degradation in the final MOB-ML regression accuracy (Fig. S2); this value is employed throughout.

Regression

Two different regression models are employed in the current work. The first is ordinary least-squares linear regression (LR), as implemented in SCIKIT-LEARN. The second is Gaussian Process Regression, as implemented in the GPY 1.9.6 software package (*GPy: A Gaussian process framework in python* since 2012). Regression is independently performed for the training data associated with each cluster, yielding a local regression model for each cluster. Also, as in our previous work,(Welborn, Cheng, and Miller, 2018; Cheng et al., 2019) regression is independently performed for the diagonal and off-diagonal pair correlation energies ($\epsilon_{\mathrm{d}}^{\mathrm{ML}}$ and $\epsilon_{\mathrm{o}}^{\mathrm{ML}}$) yielding independent regression models for each (Eq. 7.3).

GPR is performed using a negative log marginal likelihood objective. As in our previous work,(Cheng et al., 2019) the Matérn 5/2 kernel is used for regression of the diagonal pair correlation energies and the Matérn 3/2 kernel is used for the off-diagonal pair correlation energies; in both cases, white noise regularization(Rasmussen and Williams, 2006) is employed, and the GPR is initialized with unit lengthscale and variance.

Classification

An RFC is trained on MOB-ML features and cluster labels for a training set and then used to predict the cluster assignment of test datapoints in MOB-ML feature space. We employ the RFC implementation in SCIKIT-LEARN, using with 200 trees, the entropy split criteria,(Criminisi, Shotton, and Konukoglu, 2012) and balanced class weights.(Criminisi, Shotton, and Konukoglu, 2012) Alternative classifiers were also tested in this work, including K-means, Linear SVM,(Fan et al., 2008) and AdaBoost;(Hastie et al., 2009) however, these schemes were generally found to yield less accurate MOB-ML energy predictions than RFC.

For comparison, a "perfect" classifier is obtained by simply including the test data within the RC training set. While useful for the analysis of prediction errors due to classification, this scheme is not generally practical because it assumes prior knowledge of the reference energy labels for the test molecules. Since the perfect classifier avoids mis-classification of the test data by construction, it should be regarded as a best case scenario for the performance of the clustering/regression/classification approach.

The clustering/regression/classification workflow

Fig. 73 summarizes the combined work flow for training and evaluating a MOB-ML model with clustering. The training involves three steps: First, the training dataset of MOB-ML feature vectors and energy labels are assigned to clusters using the RC method (panel a). Second, for each cluster of training data, the regression model (LR or GPR) is trained to enable the prediction of pair correlation energies from the MOB-ML vector. Third, a classifier is trained from the MOB-ML feature vectors and cluster labels for the training data, to enable the prediction of the cluster assignment from a MOB-ML feature vector.

The resulting MOB-ML model is specified in terms of the method of clustering (RC, for all results presented here), the method of regression (either LR or GPR), and the method of classification (either RFC or the perfect classifier). In referring to a given MOB-ML model, we employ a notation that specifies these options (e.g., RC/LR/RFC or RC/GPR/perfect).

Evaluation of the trained MOB-ML model is explained in Fig. 73d. A given molecule is first decomposed into a set of test feature vectors associated with the pairs of occupied MOs. The classifier is then used to assign each feature vector to an associated cluster. The cluster-specific regression model is then used to predict the pair correlation energy from the MOB feature vector. And finally, the pair correlation energies are summed to yield the total correlation energy for the molecule.

To improve the accuracy and reduce the uncertainty in the MOB-ML predictions, ensembles of 10 independent models using the clustering/regression/classification workflow are trained, and the predictive mean and the corresponding standard error of the mean (SEM) are computed by averaging over the 10 models; a comparison between the learning curves(Cortes et al., 1994) from a single run and from averaging over the 10 independent models is included in Supporting Information Fig. S1. As described here, the predicted correlation energies may exhibit discontinuities as a function of nuclear position, due to changes in the assignment of feature vectors among the clusters; moving forward, this may be avoided with the use of soft (or fuzzy) clustering algorithms.(Baraldi and Blonda, 1999)

7.4 Results

Clustering and classification in MOB feature space

We begin by showing that the situation explored in Fig. 72, in which locally linear clusters overlap, also arises in realistic chemical applications of MOB-ML. We consider the QM7b-T set of drug-like molecules with thermalized geometries, using the diagonal pair correlation energies $\epsilon_{\mathrm{d}}^{\mathrm{ML}}$ computed at the MP2/cc-pVTZ level. Randomly selecting 1000 molecules for training, we perform RC on the dataset comprised of these energy labels and feature vectors, using $N = 20$ optimized clusters; the sensitivity of RC to the choice of N is examined later.

In many cases, the resulting clusters are well separated, such that the datapoints for one cluster have small distances (as measured by the linear regression distance metric, Eq. 7.6) to the cluster which it belongs to and large distances to all other clusters. However, the clusters can also overlap. Fig. 74a illustrates this overlap for two particular clusters (labeled 1 and 2) obtained from the QM7b-T diagonal-pair training data.

Each datapoint assigned to cluster 1 (blue) is plotted according to its distance to both cluster 1 and cluster 2; likewise for the datapoints in cluster 2 (red). The datapoints for which the distances to both clusters approach zero correspond to regions of overlap between the clusters in the high-dimensional space of MOB-ML features, akin to the case shown in Fig. 72.

Finally, in Fig. 74b, we illustrate the classification of the feature vectors into clusters. An RFC is trained on the feature vectors and cluster labels for the diagonal pairs of 1000 QM7b-T molecules in the training set, and the classifier is then used to predict the cluster assignment for the feature vectors associated with the remaining diagonal pairs of 6211 molecules in QM7b-T. For clusters 1 and 2, we then analyze the accuracy of the RFC by plotting the linear regression distance for each datapoint to the two clusters, as well as indicating the RFC classification of the feature vector. Each red datapoint in Fig. 74b that lies above the diagonal line of reflection is mis-classified into cluster 2, and similarly, each blue datapoint that lies below the line of reflection is mis-classified into cluster 1. The figure illustrates that while RFC is not a perfect means of classification, it is at least qualitatively correct. Later, in the results section, we will analyze the sources of MOB-ML prediction errors due to mis-classification by comparing energy predictions obtained with perfect classification versus RFC.

Chemically intuitive clusters

To address this, we employ a training set of 500 randomly selected molecules from QM7b-T, and we perform regression clustering for the diagonal pair correlation energies $\epsilon_{\mathrm{d}}^{\mathrm{ML}}$ with a range of total cluster numbers, up to $N = 20$. For each clustering,

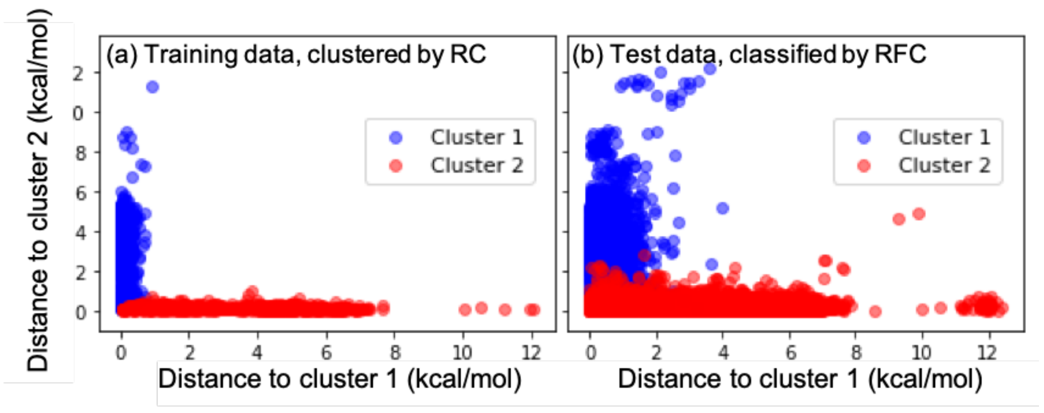


Figure 74: (a) Illustration of the overlap of clusters obtained via RC for the training set molecules from QM7b-T. (b) Classification of the datapoints for the remaining test molecules from QM7b-T, using RFC. Distances correspond to the linear regression metric defined in Eq. 7.6.

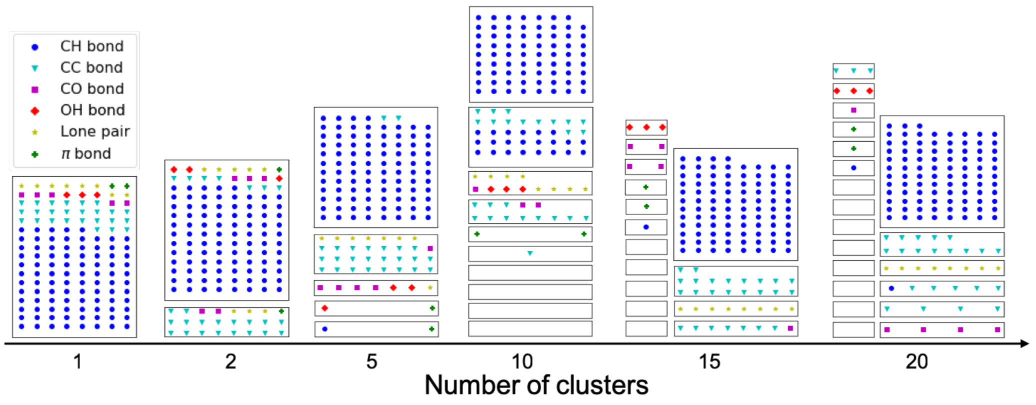


Figure 75: Analyzing the results of clustering/classification in terms of chemical intuition. Using a a training set of 500 randomly selected molecules from QM7b-T, RC is performed for the diagonal pair correlation energies, $\epsilon_{\mathrm{d}}^{\mathrm{ML}}$, with a range of cluster numbers, N, and for each clustering, an RFC is trained. Then, the trained classifier is applied to a set of test molecules (CH_4, C_2H_6, C_2H_4, C_3H_8, CH_3CH_2OH, CH_3OCH_3, $CH_3CH_2CH_2CH_3$, $CH_3CH(CH_3)CH_3$, $CH_3CH_2CH_2CH_2CH_2CH_2CH_3$, $(CH_3)_3CCH_2OH$, and $CH_3CH_2CH_2CH_2CH_2CH_2OH$) which have chemically intuitive LMO types, as indicated in the legend. The LMOs are successfully resolved according to type by the classifier as N increases. Empty boxes correspond to clusters into which none of the LMOs from the test set is classified; these are expected since the training set is more diverse than the test set.

we then train an RFC. Finally, each trained RFC is independently applied to a set of test molecules with easily characterized valence molecular orbitals (listed in the caption of Fig. 75), to see how the feature vectors associated with valence occupied LMOs are classified among the optimized clusters.

Figure 75 presents the results of this exercise, clearly indicating the agreement between chemical intuition and the predictions of the RFC. As the number of clusters increases, the feature vectors associated with different valence LMO types are resolved into different clusters; and with a sufficiently large number of clusters (15 or 20), each cluster is dominated by a single type of LMO while each LMO type is assigned to a small number of different clusters. The empty boxes in Fig. 75 reflect that the training set contains a larger diversity of LMO types than the 11 test molecules, which is expected. The observed consistency of the clustering/classification method presented here with chemical intuition is of course promising for the accurate local regression of pair correlation energies, which is the focus of the current work; however, the results of Fig. 75 also suggest that the clustering/classification of chemical systems in MOB-ML feature space provides a powerful and highly general way of mapping the structure of chemical space for other applications, including explorative or active ML applications.(Browning et al., 2017)

Sensitivity to the number of clusters

We now explore the sensitivity of the MOB-ML clustering/regression/classification implementation to the number of employed clusters. In particular, we investigate the mean absolute error (MAE) of the MOB-ML predictions for the diagonal ($\sum_i \epsilon_{ii}$) and off-diagonal ($\sum_{i\neq j} \epsilon_{ij}$) contributions to the total correlation energy, as a function of the number of clusters, N, used in the RC. The MOB-ML models employ linear regression and RFC classification (i.e., the RC/LR/RFC protocol); the training set is comprised of 1000 randomly chosen molecules from QM7b-T, and the test set contains the remaining molecules in QM7b-T.

Figure 76 presents the result of this calibration study, plotting the prediction MAE as a function of the number of clusters. Not surprisingly, the prediction accuracy for both the diagonal and off-diagonal contributions improves with N, although it eventually plateaus in both cases. For the diagonal contributions, the accuracy improves most rapidly up to approximately 20 clusters, in accord with the observations in Fig. 75; and for the off-diagonal contributions, a larger number of clusters is useful for reducing the MAE error, which is sensible given the greater variety of feature vectors that can be created from pairs of LMOs rather than only individual LMOs. Appealingly, there does not seem to be a strong indication of MAE increases due to "over-clustering". While recognizing that the optimal number of clusters will, in general, depend somewhat on the application and the regression method (i.e., LR versus GPR), the results in Fig. 76 nonetheless provide useful guidance with regard to the appropriate values of N. Throughout the remainder of the study, we employ a value of $N = 20$ for the MOB-ML prediction of diagonal contributions to the correlation energy and a value of $N = 70$ for the off-diagonal contributions; however, we recognize that these choices could be further optimized.

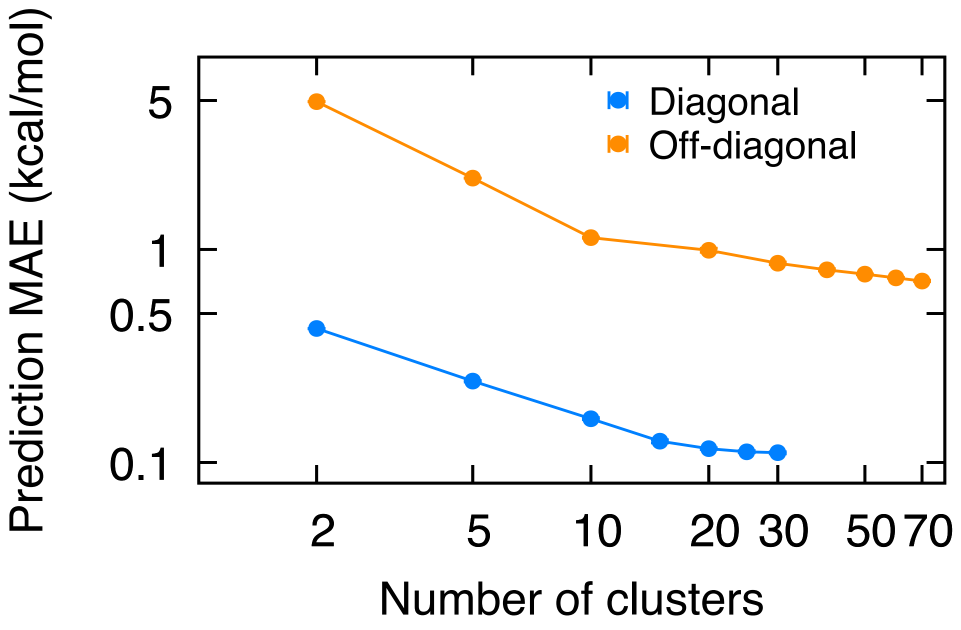


Figure 76: Illustration of the sensitivity of MOB-ML predictions for the diagonal and off-diagonal contributions to the correlation energy for the QM7b-T set of molecules, using a subset of 1000 molecules for training and the RC/LR/RFC protocol. The standard error of the mean (SEM) for the predictions is smaller than the size of the plotted points.

Performance and training cost of MOB-ML with RC

We now investigate the effect of clustering on the accuracy and training costs of MOB-ML for applications to sets of drug-like molecules. Figure 77a presents learning curves (on a linear-linear scale) for various implementations of MOB-ML applied to MP2/cc-pVTZ correlation energies, with the training and test sets corresponding to non-overlapping subsets of QM7b-T. In addition to the new results obtained using RC, we include the MOB-ML results from our previous work (GPR without clustering).(Cheng et al., 2019)

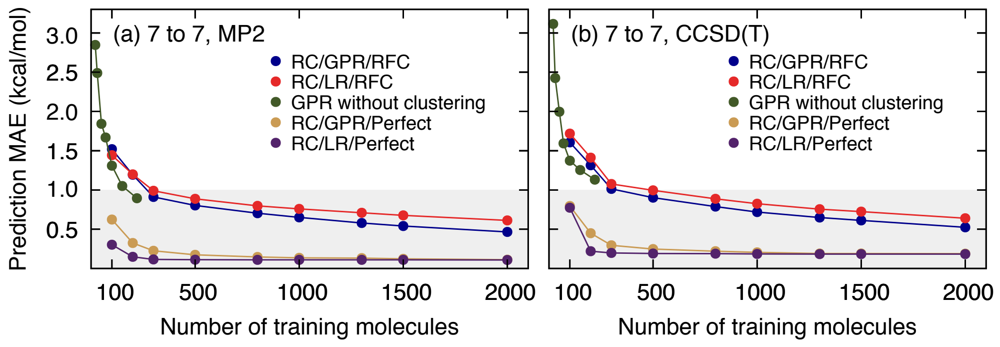


Figure 77: Learning curves for various implementations of MOB-ML applied to (a) MP2/cc-pVTZ and (b) CCSD(T)/cc-pVDZ correlation energies, with the training and test sets corresponding to non-overlapping subsets of the QM7b-T set of drug-like molecules with up to heavy seven atoms. Results obtained using GPR without clustering (green) are reproduced from Ref. (Cheng et al., 2019). The gray shaded area corresponds to a MAE of 1 kcal/mol per seven heavy atoms. The prediction SEM is smaller than the plotted points. The log-log version of this plot is provided in Fig. S3.

Figure 77a yields three clear observations. The first is that the use of RC with RFC (i.e., RC/GRP/RFC and RC/LR/RFC) leads to slightly less efficient learning curves than our previous implementation without clustering, at least when efficiency is measured in terms of the number of training molecules. Both the RC/GPR/RFC and RC/LR/RFC protocols require approximately 300 training molecules to reach the 1 kcal/mol per seven heavy atoms threshold for chemical accuracy employed here, whereas MOB-ML without clustering requires approximately half as many training molecules. The second observation is that the classifier is the dominant source of prediction error in these results. Comparison of results using RFC versus the perfect classifier (which utilizes prior knowledge of the energy labels and this thus not generally practical), reveals a dramatic reduction in the prediction error, regardless of the regression method. This result indicates that there is potentially much to be gained from the development of improved classifiers for MOB-ML applications. A third observation is that with a perfect classifier, the LR slightly outperforms GPR, given that the clusters are optimized to be locally linear; however, GPR slightly outperforms LR in combination with the RFC, indicating that GPR is less sensitive to classification error that LR.

Figure 77b presents the corresponding results at the CCSD(T)/cc-pVDZ level of theory. The same trends emerge as the ones at the MP2/cc-pVTZ level of theory. As seen in previous work, the training efficiency of MOB-ML with respect to the size

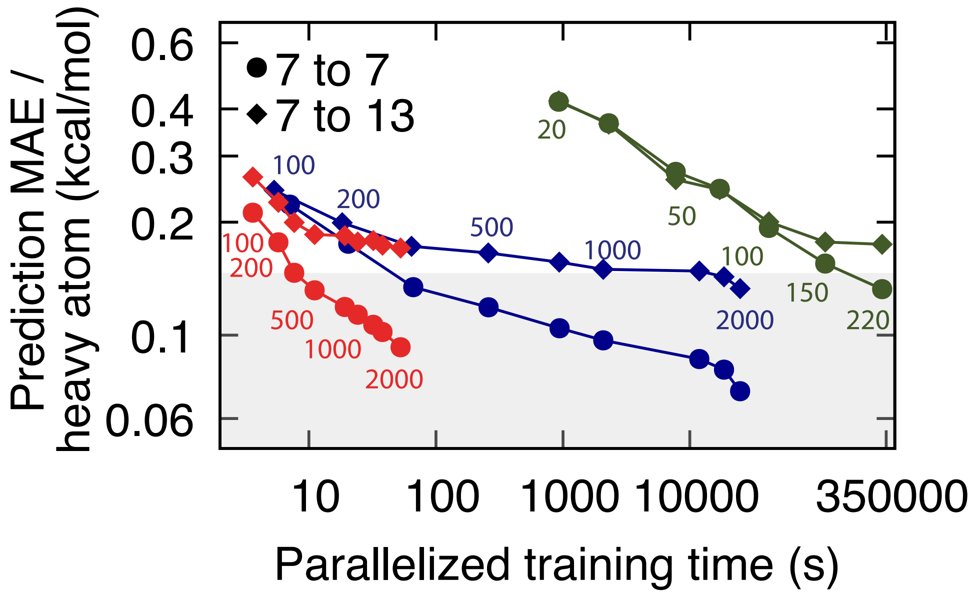


Figure 78: Training costs and transferability of MOB-ML with clustering (RC/LR/RFC, red; RC/GPR/RFC, blue) and without clustering (green, Ref. (Cheng et al., 2019)), applied to correlation energies at the MP2/cc-pVTZ level. Prediction errors are plotted as a function of wall-clock training time. Training sets are comprised of subsets of the QM7b-T dataset, with the number of training molecules indicated via datapoint labels. Correlation energy predictions are made for test sets comprised of the remaining seven-heavy-atom molecules from QM7b-T (circles) and the thirteen-heavy-atom molecules from GDB-13-T (diamonds). Both MAE prediction errors and parallelized wall-clock training times are plotted on a log scale. The gray shaded area corresponds to a MAE of 1 kcal/mol per seven heavy atoms. The prediction SEM is smaller than the plotted points. Details of the parallelization and employed computer hardware are described in the text.

reference dataset is found to be largely insensitive to the level of electronic structure theory.(Welborn, Cheng, and Miller, 2018; Cheng et al., 2019)

Figure 78 explores the training costs and transferability of MOB-ML models that employ RC. In all cases, the models are trained on random subsets of molecules from QM7b-T with up to seven heavy atoms, and predictions are made either on the remaining molecules of QM7b-T (circles) or on the GDB-13-T set (diamonds); it has previously been shown than that MOB-ML substantially outperforms the FCHL atom-based-feature method in terms of transferability from small to large molecules.(Cheng et al., 2019) The parallelization of the training steps are implemented as follows. Within the RC step, the LR for each cluster is performed independently on a different core of a 16-core Intel Skylake (2.1 GHz) CPU processor. Within the regression step, the LR or GPR for each cluster is likewise performed independently on a different core. For RFC training, we apply parallel 200 cores using the parallel implementation of SCIKIT-LEARN, since there are 200 trees. The regression and RFC training are independent of each other and are thus also trivially parallelizable.

Focusing first on the predictions for seven-heavy-atom molecules (circles), it is clear from Fig. 78 that RC leads to large improvements in the efficiency of the MOB-ML wall-clock training costs. Although it requires somewhat more training molecules than MOB-ML without clustering, MOB-ML with clustering enables chemical accuracy to be reached with the training cost reduced by a factor of approximately 4500 for RC/GPR/RFC and of 35000 for RC/LR/RFC. Remarkably, for predictions within the QM7b-T set, chemical accuracy can be achieved using RC/LR/RFC with a wall-clock training time of only 7.7 s.

Figure 78 also demonstrates the transferability of the MOB-ML models for predictions on the GDB-13-T set of thirteen-heavy-atom molecules (diamonds). In general, it is seen that the degradation in the MAE per atom is greater for the RC/LR/RFC than for RC/GPR/RFC, due to the previously mentioned sensitivity of LR to classification error. However, we note that the RC/GPR/RFC enables predictions on GDB-13-T (blue, diamonds) that meet the per-atom threshold of chemical accuracy used here, whereas that threshold was not achievable without clustering (green, diamonds) due to the prohibitive training costs involved.

The improved efficiency of MOB-ML training with the use of clustering arises from the cubic scaling of standard GPR in terms of training time ($\mathcal{O}(M^3)$, where M is number of training pairs).(Rasmussen and Williams, 2006) Trivial parallelization over the independent regression of the clusters reduces this training time cost to the cube of largest cluster. We note that other kernel-based ML methods with high complexity in training time, like Kernel Ridge Regression,(Murphy, 2012) would similarly benefit from clustering. For the RC/LR/RFC and RC/GPR/RFC results presented in Fig. 78, a breakdown of the training time contributions for each step of the clustering/regression/classification workflow as a function of the size of the training dataset is shown in Fig. S4; this supporting information figure confirms that the GPR regression dominates the total training (and prediction) costs for the RC/GPR/RFC implementation, whereas training the RFC dominates the training costs for RC/LR/RFC. In addition to improved efficiency in terms of training time, clustering also brings benefits in terms of the memory costs for MOB-ML training, due to the quadratic scaling of GPR memory costs in terms of the size of the dataset.

Finally, returning to the learning curves, we compare the results for MOB-ML both with and without clustering to recent work(Anders S Christensen et al., 2018, arXiv:1909.01946) using Faber-Christensen-Huang-Lilienfeld (FCHL) features. Fig. 79 shows these various learning curves for the MP2/cc-pVTZ correlation energies. For Fig. 79a, the training and test sets correspond to non-overlapping subsets of QM7b-T, and Fig. 79b shows the transferability of the same models trained using QM7b-T to predict the energies for GDB-13-T. Fig. 79a again shows that MOB-ML RC/GPR/RFC requires slightly more training geometries than MOB-ML without clustering, yet both MOB-ML protocols are more efficient in terms of training data than either the FCHL18(Felix A Faber et al., 2018) or FCHL19 implementations(Anders S Christensen et al., 2018, arXiv:1909.01946). Like MOB-ML with clustering, the FCHL19 implementation was developed to reduce training times.

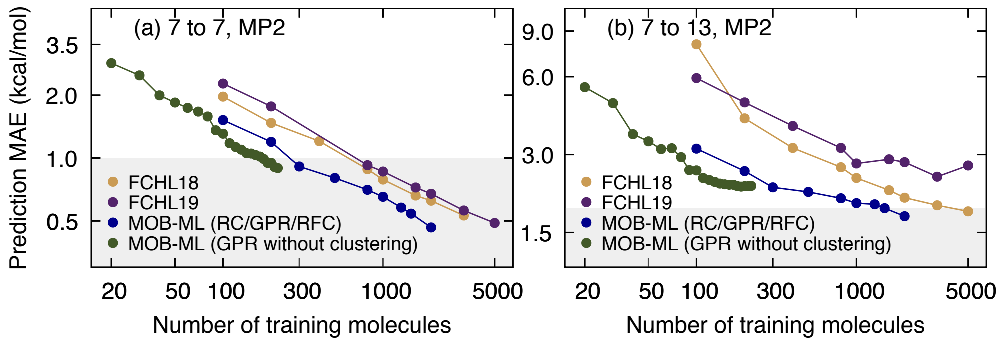


Figure 79: Comparison of learning curves for MP2/cc-pVTZ correlation energies obtained using MOB-ML (with and without clustering) versus FCHL18 and FCHL19. Part (a) presents results for which both the training and test sets include molecules from QM7b-T, and part (b) presents results for which the training set includes molecules from QM7b-T and the test set includes molecules from GDB-13-T. The MAE are plotted on a log-log scale as a function of number of training molecules. The gray shaded area corresponds to a MAE of 1 kcal/mol per seven heavy atoms. Results for FCHL18 and FCHL19 were digitally captured from Ref. (Anders S Christensen et al., 2018, arXiv:1909.01946).

Capping the cluster size

Since the parallelized training time for RC/GPR/RFC is dominated by the GPR regression of the largest cluster (Fig. S4), a natural question is whether additional computational savings and adequate prediction accuracy could achieved by simply capping the number of datapoints in the largest cluster. In doing so, we define $S_{\max}^{N_{\text{cap}}}$ to be the number of datapoints in the largest cluster obtained when the RC with the greedy algorithm is applied to a training dataset of N_{cap} molecules from QM7b-T. Upon specifying N_{cap} (and thus $S_{\text{max}}^{N_{\text{cap}}}$), the RC/GPR/RFC implementation is modified as follows. For a given number of training molecules (which will typically exceed N_{cap}), the RC step is performed as normal. However, at the end of the RC step, datapoints for clusters whose size exceeds $S_{\text{max}}^{N_{\text{cap}}}$ are discarded at random until all clusters contain $S_{\text{max}}^{N_{\text{cap}}}$ or fewer datapoints. The GPR and RFC training steps are performed as before, except using this set of clusters that are capped in size. The precise value of $S_{\text{max}}^{N_{\text{cap}}}$ will vary slightly depending on which training molecules are randomly selected for training and the convergence of the greedy algorithm, but typical values for $S_{\text{max}}^{N_{\text{cap}}}$ are $672, 1218, 1863, 3005,$ and 4896 for $N_{\text{cap}} = 100, 200, 300, 500,$ and 800, respectively, and those values will be used for the numerical tests presented here.

Figure 710a demonstrates that capping the maximum cluster size allows for substantial improvements in accuracy when the number of training molecules exceeds N_{cap}. Specifically, the figure shows the effect of capping on RC/GPR/RFC learning curves for MP2/cc-pVTZ correlation energies, with the training and test sets corresponding to non-overlapping subsets of QM7b-T. As a baseline, note that with 100 training molecules, the RC/GPR/RFC implementation yields a prediction MAE of approximately 1.5 kcal/mol. However, if the maximum cluster size is capped at $N_{\text{cap}} = 100$ and 300 training molecules are employed, then the prediction MAE drops to approximately 1.0 kcal/mol while the parallelized training cost for RC/GPR/RFC will be unchanged so long as it remains dominated by the size of the largest cluster. As expected, Fig. 710a shows that the learning curves saturate at higher prediction MAE values when smaller values of N_{cap} are employed. Nonetheless, the figure demonstrates that if additional training data is available, then the prediction accuracy for MOB-ML with RC can be substantially improved while capping the size of the largest cluster.

Figure 710b demonstrates the actual effect of capping on the parallelized training time, plotting the prediction MAE versus parallelized training time as a function of the number of training molecules. For reference, the results obtained using RC/LR/RFC and RC/GPR/RFC without capping are reproduced from Fig. 78. As is necessary, the RC/GPR/RFC results obtained with capping exactly overlap those obtained without capping when the number of training molecules is not greater than N_{cap}. However, for each value of N_{cap}, a sharp drop in the prediction MAE is seen when the number of training molecules begins to exceed N_{cap}, demonstrating that prediction accuracy can be greatly improved with minimal increase in parallelized training time. For example, it is seen that for RC/GPR/RFC with $N_{\text{cap}} = 100$, chemical accuracy can be reached with only 7.4 s of parallelized training, slightly less than even RC/LR/RFC. For small values of N_{cap}, this prediction MAE eventually levels-off versus the training time, since the RFC training step becomes the dominant contribution to the training time.

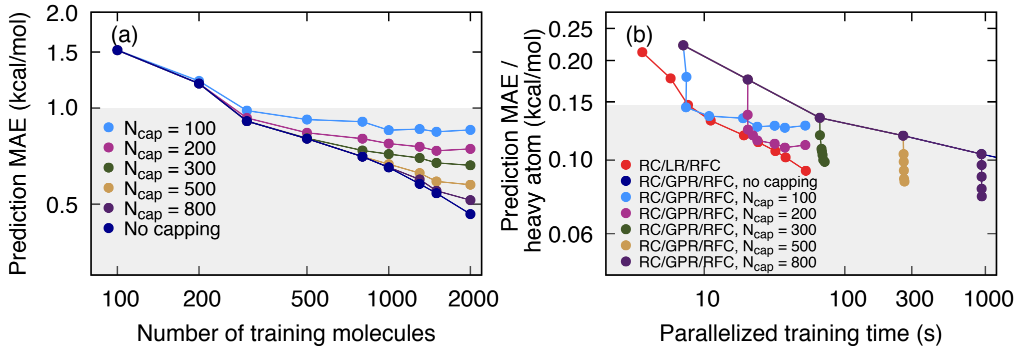


Figure 710: The effect of cluster-size capping on the prediction accuracy and training costs for MOB-ML with RC. Results reported for correlation energies at the MP2/cc-pVTZ level, with the training and test sets corresponding to non-overlapping subsets of the QM7b-T set of drug-like molecules with up to heavy seven atoms. (a) Prediction MAE versus the number of training molecules, with the clusters capped at various maximum sizes. The RC/GPR/RFC curve without capping is reproduced from Fig. 77a. (b) Prediction MAE per heavy atom versus parallelized training time as a function of the number of training molecules, as in Fig. 78. The results for MOB-ML with clustering and without capping cluster size (RC/LR/RFC, red; RC/GPR/RFC, blue) are reproduced from Fig. 78. Also, the results for RC/GPR/RFC with various capping sizes N_{cap} are shown. For part (a), the gray shaded area corresponds to a MAE of 1 kcal/mol, and for part (b), it corresponds to 1 kcal/mol per seven heavy atoms, to provide consistency with preceding figures. The prediction SEM is smaller than the plotted points.

7.5 Conclusions

Molecular-orbital-based (MOB) features offer a complete representation for mapping chemical space and a compact representation for evaluating correlation energies. In the current work, we take advantage of the intrinsic structure of MOB feature space, which cluster according to types of localized molecular orbitals, as well as the fact that orbital-pair contributions to the correlation energy contributions vary linearly with the MOB features, to overcome a fundamental bottleneck in the efficiency of machine learning (ML) correlation energies. Specifically, we introduce a regression clustering (RC) approach in which MOB features and pair correlation energies are clustered according to their local linearity; we then individually regress these clusters and train a classifier for the prediction of cluster assignments on the basis of MOB features. This combined clustering/regression/classification approach is found to reduce MOB-ML training times by 3-4 orders of magnitude, while enabling prediction accuracies that are substantially improved over that which is possible using MOB-ML without clustering. The use of a random forest classifier for the cluster assignments, while better than alternatives that were explored, is found to be the limiting factor in terms of MOB-ML accuracy within this new approach, motivating future work on improved classifiers. This work provides a useful step towards the development of accurate, transferable, and scalable quantum ML methods to describe ever-broader swathes of chemical space.

References

Baraldi, Andrea and Palma Blonda (1999). "A survey of fuzzy clustering algorithms for pattern recognition. I". In: *IEEE Trans. Syst. Man Cybern. B Cybern.* 29.6, pp. 778–785.

Bartlett, Rodney J. et al. (1990). "Non-iterative fifth-order triple and quadruple excitation energy corrections in correlated methods". In: *Chem. Phys. Lett.* 165.6, pp. 513–522. ISSN: 00092614. DOI: 10.1016/0009-2614(90)87031-L.

Bartók, Albert P. et al. (Apr. 2010). "Gaussian approximation potentials: the accuracy of quantum mechanics, without the electrons". In: *Phys. Rev. Lett.* 104.13, p. 136403. ISSN: 0031-9007. DOI: 10.1103/PhysRevLett.104.136403. URL: https://link.aps.org/doi/10.1103/PhysRevLett.104.136403.

Behler, Jörg (Nov. 2016). "Perspective: Machine learning potentials for atomistic simulations". In: *J. Chem. Phys.* 145.17, p. 170901. ISSN: 0021-9606. DOI: 10.1063/1.4966192. URL: http://aip.scitation.org/doi/10.1063/1.4966192.

Blum, L. C. and J.-L. Reymond (2009). "970 Million druglike small molecules for virtual screening in the chemical universe database GDB-13". In: *J. Am. Chem. Soc.* 131, p. 8732.

Brockherde, Felix et al. (2017). "Bypassing the Kohn-Sham equations with machine learning". In: *Nat. Commun.* 8.1, p. 872.

Browning, Nicholas J et al. (Apr. 2017). "Genetic Optimization of Training Sets for Improved Machine Learning Models of Molecular Properties". In: *J. Phys. Chem. Lett.* 8.7, pp. 1351–1359. ISSN: 1948-7185. DOI: 10.1021/acs.jpclett.7b00038. URL: https://doi.org/10.1021/acs.jpclett.7b00038.

Butler, Keith T. et al. (July 2018). “Machine learning for molecular and materials science”. In: *Nature* 559.7715, pp. 547–555. ISSN: 0028-0836. DOI: 10.1038/s41586-018-0337-2. URL: http://www.nature.com/articles/s41586-018-0337-2.

Cheng, Lixue et al. (Dec. 2019). “Regression Clustering for Improved Accuracy and Training Costs with Molecular-Orbital-Based Machine Learning”. In: *Journal of Chemical Theory and Computation* 15.12, pp. 6668–6677. ISSN: 1549-9618. DOI: 10.1021/acs.jctc.9b00884.

Christensen, Anders S., Felix A. Faber, and O. Anatole von Lilienfeld (2019). “Operators in quantum machine learning: Response properties in chemical space”. In: *J. Chem. Phys.* 150.6, p. 064105. DOI: 10.1063/1.5053562.

Christensen, Anders S et al. (2018, arXiv:1909.01946). “FCHL revisited: faster and more accurate quantum machine learning”. In:

Cortes, Corinna et al. (1994). “Learning Curves: Asymptotic Values and Rate of Convergence”. In: *Advances in Neural Information Processing Systems 6*. Ed. by J. D. Cowan, G. Tesauro, and J. Alspector. Morgan-Kaufmann, pp. 327–334.

Criminisi, Antonio, Jamie Shotton, and Ender Konukoglu (2012). “Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning”. In: *Found. Trends Comput. Graph. Vis.* 7.2–3, pp. 81–227.

Dunning, Thom H (1989). “Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen”. In: *J. Chem. Phys.* 90.2, p. 1007. DOI: 10.1063/1.456153.

Faber, Felix A et al. (2018). “Alchemical and structural distribution based representation for universal quantum machine learning”. In: *J. Chem. Phys.* 148.24, p. 241717. DOI: 10.1063/1.5020710.

Fan, Rong-En et al. (2008). “LIBLINEAR: A library for large linear classification”. In: *J. Mach. Learn. Res.* 9.Aug, pp. 1871–1874.

Fujikake, So et al. (June 2018). “Gaussian approximation potential modeling of lithium intercalation in carbon nanostructures”. In: *J. Chem. Phys.* 148.24, p. 241714. ISSN: 0021-9606. DOI: 10.1063/1.5016317. URL: http://aip.scitation.org/doi/10.1063/1.5016317.

Gastegger, Michael, Jörg Behler, and Philipp Marquetand (2017). “Machine learning molecular dynamics for the simulation of infrared spectra”. In: *Chem. Sci.* 8 (10), pp. 6924–6935. DOI: 10.1039/C7SC02267K. URL: http://dx.doi.org/10.1039/C7SC02267K.

Gawehn, Erik, Jan A. Hiss, and Gisbert Schneider (Jan. 2016). “Deep learning in drug discovery”. In: *Mol. Inform.* 35.1, pp. 3–14. ISSN: 18681743. DOI: 10.1002/minf.201501008. URL: http://doi.wiley.com/10.1002/minf.201501008.

Ghosh, Kunal et al. (2019). “Deep learning spectroscopy: Neural networks for molecular excitation spectra”. In: *Adv. Sci.* 6.9, p. 1801367. DOI: 10.1002/advs.201801367. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/advs.201801367. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/advs.201801367.

GPy: A Gaussian process framework in python (since 2012). http://github.com/SheffieldML/GPy.

Hansen, Katja et al. (2013). “Assessment and validation of machine learning methods for predicting molecular atomization energies”. In: *J. Chem. Theory Comput.* 9.8, p. 3404.

Hastie, Trevor et al. (2009). “Multi-class adaboost”. In: *Stat. Its Interface* 2.3, pp. 349–360.

Kearnes, Steven et al. (2016). “Molecular graph convolutions: Moving beyond fingerprints”. In: *J. Comput. Aided Mol. Des.* 30.8, p. 595.

Kim, Edward et al. (Dec. 2017). “Virtual screening of inorganic materials synthesis parameters with deep learning”. In: *npj Comput. Mater.* 3.1, p. 53. ISSN: 2057-3960. DOI: 10.1038/s41524-017-0055-6. URL: http://www.nature.com/articles/s41524-017-0055-6.

Lavecchia, Antonio (Mar. 2015). “Machine-learning approaches in drug discovery: Methods and applications”. In: *Drug Discov. Today* 20.3, pp. 318–331. ISSN: 1359-6446. DOI: 10.1016/J.DRUDIS.2014.10.012. URL: https://www.sciencedirect.com/science/article/pii/S1359644614004176.

Li, Haichen et al. (2018). “A density functional tight binding layer for deep learning of chemical Hamiltonians”. In: *J. Chem. Theory Comput.* 14.11, pp. 5764–5776. DOI: 10.1021/acs.jctc.8b00873.

Lloyd, Stuart (1982). “Least squares quantization in PCM”. In: *IEEE Trans. Inf. Theory* 28.2, pp. 129–137.

Lubbers, Nicholas, Justin S. Smith, and Kipton Barros (2018). “Hierarchical modeling of molecular energies using a deep neural network”. In: *J. Chem. Phys.* 148.24, p. 241715. DOI: 10.1063/1.5011181.

Mater, Adam C. and Michelle L. Coote (2019). “Deep learning in chemistry”. In: *J. Chem. Inf. Model* 59.6, pp. 2545–2559. DOI: 10.1021/acs.jcim.9b00266. eprint: https://doi.org/10.1021/acs.jcim.9b00266. URL: https://doi.org/10.1021/acs.jcim.9b00266.

McGibbon, Robert T et al. (2017). “Improving the accuracy of Møller-Plesset perturbation theory with neural networks”. In: *J. Chem. Phys.* 147.16, p. 161725.

Møller, Chr. and M S Plesset (1934). “Note on an approximation treatment for many-electron systems”. In: *Phys. Rev.* 46.7, p. 618. DOI: 10.1103/PhysRev.46.618.

Montavon, Grégoire, Matthias Rupp, Vivekanand Gobre, Alvaro Vazquez-Mayagoitia, Katja Hansen, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole von Lilienfeld (2013). "Machine learning of molecular electronic properties in chemical compound space". In: *New J. Phys.* 15.9, p. 095003. URL: `http://stacks.iop.org/1367-2630/15/i=9/a=095003`.

Montavon, Grégoire, Matthias Rupp, Vivekanand Gobre, Alvaro Vazquez-Mayagoitia, Katja Hansen, Alexandre Tkatchenko, Klaus-Robert Müller, and O Anatole von Lilienfeld (2013). "Machine learning of molecular electronic properties in chemical compound space". In: *New J. Phys.* 15.9, p. 95003.

Murphy, Kevin P (2012). *Machine learning: a probabilistic perspective*. Cambridge, MA: MIT Press, pp. 492–493.

Nandy, Aditya et al. (Oct. 2018). "Strategies and software for machine learning accelerated discovery in transition metal chemistry". In: *Ind. Eng. Chem. Res.* 57.42, pp. 13973–13986. ISSN: 0888-5885. DOI: `10.1021/acs.iecr.8b04015`. URL: `https://doi.org/10.1021/acs.iecr.8b04015`.

Nesbet, R K (1958). "Brueckner's theory and the method of superposition of configurations". In: *Phys. Rev.* 109.5, p. 1632.

Nguyen, Thuong T. et al. (2018). "Comparison of permutationally invariant polynomials, neural networks, and Gaussian approximation potentials in representing water interactions through many-body expansions". In: *J. Chem. Phys.* 148.24, p. 241725.

Nudejima, Takuro et al. (2019). "Machine-learned electron correlation model based on correlation energy density at complete basis set limit". In: *J. Chem. Phys.* 151.2, p. 024104. DOI: `10.1063/1.5100165`.

Paesani, Francesco (2016). "Getting the right answers for the right reasons: toward predictive molecular simulations of water with many-body potential energy functions". In: *Acc. Chem. Res.* 49.9, p. 1844. DOI: `10.1021/acs.accounts.6b00285`.

Pedregosa, F et al. (2011). "Scikit-learn: machine learning in python (v0.21.2)". In: *J. Mach. Learn. Res.* 12, p. 2825.

Popova, Mariya, Olexandr Isayev, and Alexander Tropsha (July 2018). "Deep reinforcement learning for de novo drug design". In: *Sci. Adv.* 4.7, eaap7885. ISSN: 2375-2548. DOI: `10.1126/sciadv.aap7885`. URL: `http://advances.sciencemag.org/lookup/doi/10.1126/sciadv.aap7885`.

Raccuglia, Paul et al. (May 2016). "Machine-learning-assisted materials discovery using failed experiments". In: *Nature* 533.7601, pp. 73–76. ISSN: 0028-0836. DOI: `10.1038/nature17439`. URL: `http://www.nature.com/articles/nature17439`.

Ramakrishnan, Raghunathan, Pavlo O Dral, et al. (2015). “Big data meets quantum chemistry approximations: the Δ-machine learning approach”. In: *J. Chem. Theory Comput.* 11.5, p. 2087.

Ramakrishnan, Raghunathan, Mia Hartmann, et al. (2015). “Electronic spectra from TDDFT and machine learning in chemical space”. In: *J. Chem. Phys.* 143.8, p. 084111. DOI: 10.1063/1.4928757. eprint: https://doi.org/10.1063/1.4928757. URL: https://doi.org/10.1063/1.4928757.

Rasmussen, Carl E and Christopher K I Williams (2006). *Gaussian processes for machine learning*. Cambridge, MA: MIT Press. URL: http://www.gaussianprocess.org/gpml/chapters/RW.pdf.

Ren, Fang et al. (Apr. 2018). “Accelerated discovery of metallic glasses through iteration of machine learning and high-throughput experiments”. In: *Sci. Adv.* 4.4, eaaq1566. ISSN: 2375-2548. DOI: 10.1126/sciadv.aaq1566. URL: http://advances.sciencemag.org/lookup/doi/10.1126/sciadv.aaq1566.

Rupp, Matthias et al. (2012). “Fast and accurate modeling of molecular atomization energies with machine learning”. In: *Phys. Rev. Lett.* 108.5, p. 58301.

Saebo, S and Peter Pulay (Oct. 1993). “Local treatment of electron correlation”. In: *Annu. Rev. Phys. Chem.* 44.1, pp. 213–236. ISSN: 0066-426X. DOI: 10.1146/annurev.pc.44.100193.001241. URL: http://www.annualreviews.org/doi/abs/10.1146/annurev.pc.44.100193.001241%20http://www.annualreviews.org/doi/10.1146/annurev.pc.44.100193.001241.

Sanchez-Lengeling, Benjamin and Alán Aspuru-Guzik (July 2018). “Inverse molecular design using machine learning: Generative models for matter engineering.” In: *Science* 361.6400, pp. 360–365. ISSN: 1095-9203. DOI: 10.1126/science.aat2663. URL: http://www.ncbi.nlm.nih.gov/pubmed/30049875.

Schütt, Kristof T et al. (2017). “Quantum-chemical insights from deep tensor neural networks”. In: *Nat. Commun.* 8, p. 13890.

Schütz, Martin (2000). “Low-order scaling local electron correlation methods. III. Linear scaling local perturbative triples correction (T)”. In: *J. Chem. Phys.* 113.22, pp. 9986–10001. ISSN: 00219606. DOI: 10.1063/1.1323265.

Segler, Marwin H. S., Mike Preuss, and Mark P. Waller (Mar. 2018). “Planning chemical syntheses with deep neural networks and symbolic AI”. In: *Nature* 555.7698, pp. 604–610. ISSN: 0028-0836. DOI: 10.1038/nature25978. URL: http://www.nature.com/doifinder/10.1038/nature25978.

Segler, Marwin H. S. and Mark P. Waller (May 2017). “Neural-symbolic machine learning for retrosynthesis and reaction prediction”. In: *Chem. - A Eur. J.* 23.25, pp. 5966–5971. ISSN: 09476539. DOI: 10.1002/chem.201605499. URL: http://doi.wiley.com/10.1002/chem.201605499.

Smith, J. S., O. Isayev, and A. E. Roitberg (2017). “ANI-1: an extensible neural network potential with DFT accuracy at force field computational cost”. In: *Chem. Sci.* 8.4, pp. 3192–3203. ISSN: 20416539. DOI: `10.1039/C6SC05720A`.

Smith, Justin S et al. (2019). “Approaching coupled cluster accuracy with a general-purpose neural network potential through transfer learning”. In: *Nat. Commun.* 10.1, p. 2903. ISSN: 2041-1723. DOI: `10.1038/s41467-019-10827-4`. URL: `https://doi.org/10.1038/s41467-019-10827-4`.

Späth, H. (1979a). “Algorithm 39: clusterwise linear regression”. In: *Computing* 22, pp. 367–373. DOI: `10.1007/BF02265317`.

– (1979b). “Correction to algorithm 39: clusterwise linear regression”. In: *Computing* 26, p. 275.

Szabo, Attila and Neil S. Ostlund (1996). *Modern Quantum Chemistry*. Mineola: Dover, pp. 231–239. ISBN: 0486691861.

Townsend, Jacob and Konstantinos D. Vogiatzis (2019). “Data-Driven acceleration of the coupled-cluster singles and doubles iterative solver”. In: *J. Phys. Chem. Lett.* 10.0, pp. 4129–4135. DOI: `10.1021/acs.jpclett.9b01442`.

Ulissi, Zachary W. et al. (Mar. 2017). “To address surface reaction network complexity using scaling relations machine learning and DFT calculations”. In: *Nat. Commun.* 8, p. 14621. ISSN: 2041-1723. DOI: `10.1038/ncomms14621`. URL: `http://www.nature.com/doifinder/10.1038/ncomms14621`.

Wei, Jennifer N., David Duvenaud, and Alán Aspuru-Guzik (Oct. 2016). “Neural networks for the prediction of organic chemistry reactions”. In: *ACS Cent. Sci.* 2.10, pp. 725–732. ISSN: 2374-7943. DOI: `10.1021/acscentsci.6b00219`. URL: `http://pubs.acs.org/doi/10.1021/acscentsci.6b00219`.

Welborn, Matthew, Lixue Cheng, and Thomas F. Miller III (Sept. 2018). “Transferability in machine learning for electronic structure via the molecular orbital basis”. In: *J. Chem. Theory Comput.* 14.9, pp. 4772–4779. DOI: `10.1021/acs.jctc.8b00636`. URL: `http://pubs.acs.org/doi/10.1021/acs.jctc.8b00636`.

Wu, Zhenqin et al. (2018). “MoleculeNet: a benchmark for molecular machine learning”. In: *Chem. Sci.* 9.2, p. 513.

Yao, Kun et al. (2018). “The TensorMol-0.1 model chemistry: a neural network augmented with long-range physics”. In: *Chem. Sci.* 9.8, pp. 2261–2269. ISSN: 20416539. DOI: `10.1039/c7sc04934j`.

Zhang, Linfeng et al. (Apr. 2018). “Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics”. In: *Phys. Rev. Lett.* 120 (14), p. 143001. DOI: `10.1103/PhysRevLett.120.143001`. URL: `https://link.aps.org/doi/10.1103/PhysRevLett.120.143001`.