Applications of machine learning to finite volume methods

Thesis by
Ben Stevens

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

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2022
Defended June 23rd, 2021

ACKNOWLEDGEMENTS

I have many people to acknowledge for supporting me throughout graduate school. First and foremost, I would like to thank my advisor Professor Tim Colonius. I truly cannot imagine a better advisor. You gave me the perfect balance of guidance and creative freedom to grow into an independent researcher.

I would also like to thank the rest of my thesis committee: Professors Yisong Yue, Guillaume Blanquart, and Anima Anandkumar. Your guidance also helped me take my research in meaningful directions.

I am also very grateful to my undergraduate mentors Professors Calvin Lui and Thom Adams. You both gave me early exposure to research, which helped me learn about this highly rewarding career path and gave me a jump start on developing research skills.

I'm very thankful to have had such excellent labmates as well. I really enjoyed getting to know everyone in the Computational Flow Physics group over the years. Andres Goza, Kazuki Maeda, Andre da Silva, Phillipe Tosi, Rahul Arun, Ke Yu, Jean Spratt, Ethan Pickering, Marcus Lee, Omar Kamal, Wei Hou, Jose Rodolfo Chreim, Lennart Schneiders, Mauro Rodriguez, Shunxiang Cao, Spencer Bryngelson, Qifan Wang, George Rigas, Kevin Schmidmayer, Erick Salcedo, Gianmarco Mengaldo, and Oliver Schmidt, you were such a fun group of people, and it made my PhD that much more enjoyable.

I would also like to acknowledge all of my amazing friends and colleagues outside of the group, who made the past four years such a wonderful time for me. Andrew Singletary, Rob Macedo, Nick Nelsen, Joe Sabuda, Hesham Zaini, and Zeke DeSantis: eating at tasty restaurants, going to crazy concerts, decompressing in the gym, discussing cross-disciplinary aspects of our research, and exploring LA really rounded out my grad school experience.

I'm also very grateful to my family for supporting me during my PhD. Mom and Dad, you were always there for me when I needed guidance on any aspect of my life, and it was so nice that I could always visit you guys whenever I needed to relax and get away from it all. Sarah, you are the best sister a brother could ask for, and an amazing friend. I love you all so much.

Lastly but certainly not least, I would like to thank the NSF for funding my PhD through the Graduate Research Followships Program. Having this funding source made it very easy for me to have flexibility in what I chose to research.

ABSTRACT

The finite volume method (FVM) has been one of the primary tools of computational fluid dynamics (CFD) for many decades. This method allows for the approximate solution of a partial differential equation (PDE) to be determined by breaking up a problem with no analytical solution into smaller pieces that can be solved together to get a physically realistic simulation. These algorithms can even be used for PDEs with discontinuous solutions, though they must be carefully designed for those situations because they cannot assume any level of smoothness in the solution. An FVM that has been designed for PDEs with discontinuous solutions is referred to as a shock-capturing method. For most of their history, FVM algorithms have been developed using rigorous mathematical arguments to formally maximize the order of convergence of the solution as the grid is refined. However, these arguments depend on the solution to the PDE being smooth, and therefore do not apply to shock-capturing methods. Instead, shock-capturing methods have traditionally been designed using human intuition to create algorithms that then perform well empirically. In this thesis, we instead follow a data-driven approach to train neural networks to use for enhanced FVM methods.

By including a neural network in our FVM, we can use empirical data to optimize the algorithm. We can also utilize ideas from traditional FVM algorithms to create hybrid methods that have tunable parameters and maintain convergence guarantees present in FVMs that have been designed by hand. We explore these hybrid methods in a variety of settings. First, we create a general-purpose shock-capturing method WENO-NN by hybridizing the popular shock-capturing method WENO-JS with a neural network. Additionally, we develop a network architecture, called FiniteNet, that can be used to learn a coarse-graining model associated with a specific PDE and embed it into an FVM scheme. Finally, we also explore the idea of using transfer learning to further improve the WENO-NN for specific problems and name the resulting algorithm WENO-TL. We demonstrate experimentally that this hybrid approach results in methods that can offer similar error levels as traditional FVMs at less computational cost. Although the neural network increases the computational cost of one evaluation of our hybrid FVM, these methods also allow the simulation to be carried out on a coarser grid, leading to a net reduction in both simulation time and memory usage.

PUBLISHED CONTENT AND CONTRIBUTIONS

Stevens, B. and T. Colonius (2020). “Enhancement of shock-capturing methods via machine learning”. In: *Theoretical and Computational Fluid Dynamics* 34, pp. 483–496. DOI: https://doi.org/10.1007/s00162-020-00531-1.

Stevens and T. Colonius (2020). “Finitenet: A fully convolutional lstm network architecture for time-dependent partial differential equations”. In: *arXiv preprint arXiv:2002.03014*. DOI: https://arxiv.org/abs/2002.03014.

In both articles listed above, the author participated in the conception of the project, developing the algorithms, writing the software, designing and running the experiments, analyzing the resulting data, and was the principal writer of each manuscript. The first article was adapted for chapter 3 of this thesis, and the second article was adapted for chapter 4.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Motivation

Partial differential equations (PDE) are fundamental to many areas of physics, such as fluid mechanics, electromagnetism, and quantum mechanics (Sommerfeld, 1949). However, it is very rare that analytical solutions exist for practical problems involving PDE. Hence, they are often solved numerically to get an approximate solution. Some popular algorithms for numerically solving PDE are the finite difference method (FDM), the finite volume method (FVM), and the finite element method (FEM). Each of these algorithms work by discretizing the continuous domain that the PDE is to be solved on into a finite amount of smaller pieces. This allows the spatial derivative terms of the PDE to be replaced with differences. For a time-invariant PDE, this converts the PDE into a system of algebraic equations that can then be solved with standard techniques such as matrix inversion or Newton's method. For time-dependent PDE, which this thesis focuses on, spatial discretization converts the PDE into a system of ODE's. This process is referred to as the method of lines. The system of ODEs can then be solved with numerical integration techniques such as Runge Kutta schemes. For some problems, the domain must be discretized into a very large number of nodes, cells, or elements, which can lead to a supercomputer being required to obtain an accurate approximation.

All classical physics models are approximations obtained by coarse-graining the true quantum mechanical behavior of matter. For example, the Navier-Stokes equations are obtained by treating a fluid as a continuum; despite this, the equations are still too computationally expensive to solve for most practical problems (Ishihara, Gotoh, and Kaneda, 2009), and so they are further coarse-grained by methods such as Large-Eddy Simulation (LES) (Halpern, 1993) or Reynolds Averaged Navier Stokes (RANS) (Chen, Patel, and Ju, 1990) to infer sub-grid behavior. Each of these coarse-grained models save multiple orders of magnitude of computational expense (Drikakis and Geurts, 2006; Wilcox, 1998). However, these approaches do not lead to models that are generally applicable (Spalart, 2010), and sometimes produce results that are untrustworthy when solving hard problems, such as transition from laminar to turbulent flow (Zhiyin, 2015). One instance of coarse graining that we will focus on in detail in this thesis is the modeling decision to ignore viscosity entirely in low viscosity flows that contain shockwaves. Physically, shock waves do have a finite-width associated with them (Puckett and Stewart, 1950), but because this occurs over such a small length scale fully resolving the shock requires a very fine grid. By ignoring viscosity, the exact solutions to the PDE instead become discontinuous, and the approximate solution instead smears the shockwave over several grid points via numerical error rather than to accurately capture the physical width of the shock. This approach saves computational expense without necessarily invalidating the rest of the solution (LeVeque et al., 2002).

Shock capturing is one specific instance of coarse-graining, but many other examples exist. PDEs give rise to vastly different solution structures in different problems (Sommerfeld, 1949). Hence, a solution approach that works well for one equation will not be generally applicable, as different difficulties can cause different methods to fail. As alluded to before, one major difficulty we examine in this thesis is PDEs with discontinuous solutions. We also briefly investigate PDEs with chaotic dynamics. PDEs with chaotic dynamics are challenging because small errors will grow quickly in time, causing the numerical solution to diverge from the true solution if the solver is not accurate enough (Strogatz, 2001). Typically, high-order numerical methods are used to solve these problems as they offer the best asymptotic error bounds (Deville et al., 2002). PDEs with discontinuous solutions are difficult to solve because high-order methods lead to Gibbs phenomena near discontinuities (Gottlieb and Shu, 1997), which can lead to numerical instabilities. High-order methods are derived with the assumption that the solution is smooth (LeVeque, 2007), but no method can achieve better than first-order accuracy in the presence of a discontinuity (LeVeque et al., 2002). Hence, we can see methods used to solve turbulence and discontinuities are at odds with each other, which makes it especially challenging to simulate problems that involve both of these issues. Other difficulties can arise in these problems such as multiphysics behavior like magnetohydrodynamics (Davidson, 2002; DeVore, 1991; Gammie, McKinney, and Tóth, 2003), combustion (Peters, 2001; Linan and Williams, 1993; Veynante and Vervisch, 2002), and fluid-structure interaction (Bungartz and Schäfer, 2006; Bazilevs et al., 2008; Tallec and Mouro, 2001; Dowell and Hall, 2001). Other problems involve complex geometries (Kim, Kim, and Choi, 2001; Mahesh, Constantinescu, and Moin, 2004; Mohd-Yusof, 1997), and many other challenges (Borggaard, Burns, and Zietsman, 2004; Gustafsson and Holmgren, 2010; Funaki, 1995; Ricardez-Sandoval, 2011; Givoli, 1991).

Numerical methods such as FDM and FVM are often developed without knowledge of the specific equation that they will be used to solve, and instead opt to maximize the rate of convergence (LeVeque, 2007). However, the dynamics of PDEs greatly influence the structure of their solutions. This information should be used to maximize the performance of the numerical methods used to simulate these equations. Machine learning is the natural tool to hybridize this information with traditional numerical methods, as detailed experimental/simulation data that contains this behavior is available and can be used to improve these numerical methods. Furthermore, machine learning techniques for capturing temporal behavior such as long short-term memory (LSTM) networks (Hochreiter and Schmidhuber, 1997a; Hochreiter and Schmidhuber, 1997b; Gers, Schmidhuber, and Cummins, 1999), and spatially local behavior such as convolutional neural networks (CNN) (Fukushima, 1980; LeCun, Boser, et al., 1989; LeCun, Bottou, et al., 1998), have undergone significant development. This makes machine learning highly effective for modeling problems with these structures, both of which occur in time-dependent PDEs.

By solving PDEs more efficiently, we can gain an improved understanding of the physics that underlies many important phenomena. This thesis will focus on applications to fluid mechanics. Understanding fluid mechanics has practical consequences in a wide variety of settings. Some examples that have been investigated by the Computational Flow Physics group include shockwave lithotripsy used in the breakup of kidney stones (Maeda et al., 2018; Johnsen and Colonius, 2008; Pishchalnikov et al., 2003), external aerodynamics of aircraft (Liska and Colonius, 2017; Mengaldo et al., 2017; Yu, Dorschner, and Colonius, 2020), turblent jets for reducing aircraft engine noise (Reba et al., 2003; Jang et al., 2012; Jordan and Colonius, 2013; Pickering et al., 2020), and even feeding patterns of whales (Bryngelson and Colonius, 2020; Bryngelson and Colonius, 2019).

1.2 Historical Context and Literature Review

1.2.1 Shock Capturing Methods

Shock-capturing methods are designed with the goal of sharply resolving a shock without inducing spurious oscillations, while also giving accurate solutions in smooth regions of the flow. One major breakthrough in this effort was the development of high-resolution methods (Harten, 1983), as these methods were capable of achieving second-order accuracy without introducing spurious oscillations around shocks. These methods gave rise to a class of high-resolution methods called essentially non-oscillatory (ENO) schemes (Harten et al., 1987) that measure the smoothness of the solution on several stencils, and then compute the flux based on the smoothest stencil to avoid interpolating through the discontinuity. These schemes are nonlinear (even when the PDE are linear) since the interpolation coefficients depend on the solution. These ideas were then modified to create WENO-JS (weighted ENO-Jiang Shu) methods (Jiang and Shu, 1996), which again compute the smoothness on several stencils. However, instead of taking only the smoothest stencil, these methods take a weighted average of the fluxes predicted on each stencil to emphasize the smoother ones. When each stencil is equally smooth, the weights are designed to cause the method to converge to the constant coefficient scheme that maximizes the order of accuracy over the union of the sub-stencils, which gives these methods a high order of accuracy for smooth solutions.

Many efforts have built on the original WENO-JS schemes by modifying the smoothness indicators (Ha et al., 2013; Kim, Ha, and Yoon, 2016; Rathan and Raju, 2018a), modifying the nonlinear weights (Borges et al., 2008; Castro, Costa, and Don, 2011; Rathan and Raju, 2018b), and using WENO-JS as part of a hybrid scheme (Li and Qiu, 2010; Peer, Dauhoo, and Bhuruth, 2009).

One commonality that has persisted since the original ENO scheme is a reliance on human intuition in shock-capturing method design, particularly in the nonlinear aspects of the schemes, i.e. smoothness indicators and weighting functions. While they have been well studied, there is no reason to believe that they are optimal. Efforts have been made to develop optimal spatial discretization methods by minimizing wave propagation errors (Kim and Lee, 1996; Lele, 1992; Liu, 2013; Tam and Webb, 1993) and minimizing error over certain frequency ranges (Zhang and Yao, 2013), and some of these techniques have even been combined with shock-capturing schemes (Fang, Li, and Lu, 2013; Wang and Chen, 2001). However, designing the optimization problem still requires human intuition with regards to balancing competing goals, rather than attempting to learn an optimal scheme from data in an unbiased way.

1.2.2 Overview of ML for PDEs and Fluid Mechanics

Over the past decades, machine learning has become ubiquitous in data analysis and is increasingly seen as having potential to improve (or reformulate) numerical methods for PDEs. Research intersecting PDEs with machine learning (ML) can be divided into two main goals: discovering PDEs from data, and using ML to better solve PDEs (Raissi, Perdikaris, and Karniadakis, 2019). Although this thesis focuses on solving PDEs, we will briefly discuss efforts to discover them from data. Brunton et al. (Brunton, Proctor, and Kutz, 2016) learned dynamical systems by applying lasso regression to a library of functions. Many papers used LSTMs or other sequence models to learn dynamical systems (Hagge et al., 2017; Yu, Zheng, et al., 2017; Vlachas, Byeon, et al., 2018; Wan et al., 2018; Li, Yu, et al., 2017; Vlachas, Pathak, et al., 2020). Other papers learned coarse-graining models by following a system identification approach (Ling, Kurzawski, and Templeton, 2016). An important quality of PDEs is the presence of spatiotemporal dynamics. Many ML approaches have been developed to make inferences about systems with this characteristic. Xingjian et al. (Xingjian et al., 2015) used a convolutional LSTM to predict short term rainfall based on radar maps. Mohan et al. (Mohan et al., 2019) developed a deep learning framework called compressed convolutional LSTM to reduce the dimensionality of turbulence.

Another interesting approach to turbulence modeling involves the use of multi-agent reinforcement learning (Novati, Laroussilhe, and Koumoutsakos, 2021) with the goal of achieving better generalization than is possible with supervised learning, which is by far the most popular machine learning approach to turbulence modeling. They demonstrate their approach on LES, and use statistical properties of corresponding direct numerical simulations (DNS) as a reward for their reinforcement learning agents. Another interesting paper focuses on the more practical side of CFD by focusing on computer-aided engineering (CAE) commonly used in industry (Molinaro et al., 2021). They use machine learning to increase the size of CFD databases used in CAE by following a hybrid physics and data-driven model, allowing their data to cover a wider range of parameters that may be encountered in practice. Their work resulted in what they call a Simulation Digital Twin (SDT). For a more general overview of machine learning in fluid mechanics, there is a useful review article (Brunton, Noack, and Koumoutsakos, 2020). This article gives an overview of the potential for useful interdisciplinary research merging these two fields, presents an overview of some popular techniques and models used in machine learning, reviews techniques that have been developed for flow modeling with machine learning, techniques for flow optimization and control using machine learning, and a discussion of where this research area is most likely headed and what progress can be expected. Another interesting review focuses on physics informed machine learning in general (Karniadakis et al., 2021). This review discusses several prominent approaches to developing data-driven models for physics problems that utilize domain knowledge to improve model performance, such as physics informed neural networks, kernel approaches, data-driven numerical methods, and other hybrid approaches.

Solving PDEs with machine learning can be further broken up into two main areas: using data to develop better solvers, and parameterizing solutions to PDEs as a neural network and learning weights to minimize the pointwise error of the PDE (Lagaris, Likas, and Fotiadis, 1998), although we will also discuss papers that do not fall into either of these two categories. This second idea has been further developed (Rudd, 2013) to solve problems such as high-dimensional PDEs (Sirignano and Spiliopoulos, 2018). Dissanayake et al. (Dissanayake and Phan-Thien, 1994) exploit the fact that neural networks are universal approximators to transform the problem of solving a PDE into an unconstrained minimization problem. Yu et al. (Yu, Hesthaven, and Yan, 2018) trained a neural network to classify the local smoothness and apply artificial viscosity based on this classification. Hsieh et al. (Hsieh et al., 2019) attempted to learn domain specific fast PDE solvers by learning how to iteratively improve the solution using a deep neural network, resulting in a 2-3 times speedup compared to state-of-the-art solvers. Pfau et al. (Pfau et al., 2018) parameterized the eigenfunctions of eigenvalue problems as a neural network and cast the training as a bilevel optimization problem to reduce bias, resulting in significantly decreased memory requirements.

One impactful framework for using neural networks to solve PDEs is Physics Informed Neural Networks (PINN) (Raissi, 2018). These neural networks can be used both for the discovery and solving of PDEs. Their paper proposes several frameworks for solving PDEs. The first is a continuous time approach, which does involve approximating the solution of a PDE as a neural network and penalizing violations to the PDE, and the initial and boundary conditions. They also propose a discrete time framework involving Runge-Kutta methods, where they place a multi-output network prior on the discrete time solution, and again penalize violations to the PDE and initial and boundary conditions. They later extended this idea to fractional PINNs, or fPINNs (Pang, Lu, and Karniadakis, 2019), which are capable of solving PDEs with fractional order derivatives. Another development to PINNs is the hp-VPINN (Kharazmi, Zhang, and Karniadakis, 2021), or hp-variational PINN that synthesizes ideas from hp-FEM. This approach uses hp-refinement via domain decomposition to learn the solution locally, where again the solution is paramaterized as a neural network. There have also been many papers studying the performance of PINNs (Yang and Perdikaris, 2019; Shin, Darbon, and Karniadakis, 2020; Wang, Teng, and Perdikaris, 2020; Zhang, Lu, et al., 2019), as well as many others applying them in a variety of settings (Mao, Jagtap, and Karniadakis, 2020; Sahli et al., 2020; Yang, Zafar, et al., 2019).

One successful application of machine learning to PDEs is the neural operator, introduced through the Graph Kernel Network (Li, Kovachki, et al., 2020c), which is used to learn an operator that can be used to solve PDEs. This paper models the Green's function corresponding to a PDE as a graph neural network (GNN), which leads to a resolution-independent solution to the PDE that can be obtained via integration. This technique was then extended to the multipole graph neural operator (Li, Kovachki, et al., 2020b), which allows the GNN to pass information over long distances and capture long range interactions with only linear complexity. Another related model is the Fourier Neural Operator (Li, Kovachki, et al., 2020a), in which the integral kernel is parameterized directly in Fourier space, which provides a family of functions that is very efficient for many physical problems. This approach leads to up to a three order of magnitude speedup compared to traditional PDE solvers. Another paper that uses kernel learning for solving PDEs uses the random feature model (Nelsen and Stuart, 2020) to parameterize an operator mapping the initial data associated with a PDE to its solution.

1.2.3 ML for finite difference and similar methods

In our work, we focus on time-dependent problems that are solved via time-stepping. There have been numerous efforts to develop discretization methods tailored to specific types of dynamics. There have also been machine learning strategies that develop equation-specific spatial discretization schemes. The idea of embedding machine learning models into spatial discretization methods for the purpose of coarse-graining was introduced by Bar-Sinai et al. (Bar-Sinai et al., 2019). In their paper, they trained a neural network to interpolate over the solution of a PDE to more accurately predict the solution at the next timestep. They then extended their method to include physical constraints, examined much larger cases (Kochkov et al., 2021), and found that they could achieve a comparable error by using a 10x coarser grid, which resulted in a roughly 80x speedup. In this paper, they also investigated learning corrections, which is similar to LES modeling, where they model a residual correction to the governing equations as a neural network. They also demonstrated strong potential for their learned schemes to generalize to unseen problems, as they train their model on forced turbulance and then show that it can accurately solve other problems such as decaying turbulence and turbulence with a previously unseen Reynold's number. PDE-Net followed a similar approach (Long, Lu, Ma, et al., 2017; Long, Lu, and Dong, 2019) in learning derivatives while also learning the PDE from data. Ranade et al. introduced DiscretizationNet (Ranade, Hill, and Pathak, 2021), which similar to other methods uses CNN-based encoder-decoder model that takes in PDE variables as its inputs and outputs new PDE variable values. Their model includes their discretization scheme into their computational graph to more quickly compute PDE residuals in order to solve the steady, incompressible Navier-Stokes equations. Recently, Bezgin et al. introduced a new technique for simulating PDEs with nonclassical undercompressive shocks (Bezgin, Schmidt, and Adams, 2021) in which they use a CNN for adaptive nonlinear flux reconstruction based on the local flow field. They utilize the idea of matching the truncation error of a discretization scheme using nonlinear terms. More specifically, they predict the time evolution of exact solutions of Riemann problems, and apply their method to hyperbolic conservation laws with non-convex fluxes. Another paper proposes MeshGraphNets (Pfaff et al., 2020), which trains a graph neural network to pass messages on a mesh and to adapt the discretization while the simulation is being stepped forward in time. They demonstrate their algorithm on several different physical systems such as aerodynamics, structural mechanics, and cloth, and show that their models can generalize to more complex systems than they were trained on and speed up the simulation by 1-2 orders of magnitude.

1.3 Summary of contributions and outline

This thesis presents several data-driven algorithms for solving partial differential equations numerically. We show that these algorithms offer several advantages over current state of the art methods in terms of simulation accuracy.

1.3.1 WENO-NN

In chapter 3 of this thesis, we present WENO-NN, a data-driven shock capturing method. This hybrid method enhances the popular WENO5-JS algorithm with a small neural network. We introduce the strategy of learning a perturbation to an already sophisticated finite-volume method to improve performance on certain classes of problems. Of note is that WENO-NN is problem-agnostic and can be applied to any PDE with discontinuous solutions, while most machine learned methods in this field are problem specific. We experimentally observe that WENO-NN is able to give faster runtime, lower error, and lower memory usage than WENO5-JS on problems where WENO5-JS is overly diffusive. Interestingly, WENO-NN is capable of maintaining a steady-state error profile at interfaces rather than continuing to smear the solution as time goes on.

1.3.2 FiniteNet

Chapter 4 of this thesis focuses on another algorithm that we have named FiniteNet, a data-driven coarse graining method. FiniteNet develops the idea of learning the temporal structure of PDEs simultaneously with the spatial structure, which is accomplished by utilizing a fully convolutional LSTM network. It also builds upon ideas developed in WENO-NN by casting the neural network outputs as perturbations to a well-known finite volume method, and trains neural networks over long time horizons in the effort of minimizing error accumulation. We demonstrate the benefits of including a temporal structure in our network by comparing how different temporal modeling techniques affect network performance, and show that including temporal modeling improves accuracy on a variety of PDEs. We also demonstrate that training over long time horizons improves the numerical stability of the learned scheme.

1.3.3 WENO-TL

Chapter 5 describes WENO-TL, in which we apply transfer learning to WENO-NN to tune it for specific problems. We see that by combining ideas from the previous two chapters, we can more reliably train an equation specific shock capturing method with powerful generalization potential. Most notably, we see that we can train the algorithm on data from 1D simulations, and then use the resulting numerical scheme to get a 10$\times$ speedup on 2D simulations of the analogous system of PDEs, as was demonstrated on our density bump test case for the Euler equations.

Chapter 2

BACKGROUND

This chapter will provide a brief description of information that is helpful for understanding the work presented in this thesis. It will first provide an overview of PDEs by discussing some basic PDE theory and the physical context of each PDE that will be considered in this thesis. This section is followed by a description of the numerical methods that are used to solve hyperbolic PDEs. It will then describe the basics of machine learning, neural networks, and the architectures that this thesis will utilize.

2.1 PDE background

In this thesis, we will solve a variety of PDEs. This section provides some general background about classifications of PDEs as relevant to this thesis, as well as context and information about each PDE that we will examine.

2.1.1 Hyperbolic vs. Elliptic PDEs

Two major classifications of PDEs are elliptic and hyperbolic PDEs. Canonical examples of each include Laplace's equation

$$\frac{\partial^2 u}{\partial t^2} + \frac{\partial^2 u}{\partial x^2} = 0, \tag{2.1}$$

which is elliptic, and the wave equation

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0, \tag{2.2}$$

which is hyperbolic. While these PDEs do appear to be similar, and are indeed identical up to a sign difference on the spatial derivative term, the solutions and strategies for solving them are very different. For a general second order, 2D linear PDE

$$L(u) = a\frac{\partial^2 u}{\partial t^2} + b\frac{\partial^2 u}{\partial x \partial t} + c\frac{\partial^2 u}{\partial x^2} + d\frac{\partial u}{\partial t} + e\frac{\partial u}{\partial x} = 0, \tag{2.3}$$

the characteristics are real if $b^2 - 4ac > 0$, which would make the PDE hyperbolic. If $b^2 - 4ac < 0$, the characteristics are complex and the PDE is elliptic, and if $b^2 - 4ac = 0$ the PDE is parabolic. For a PDE in n-dimensions (including temporal), characteristics are surfaces in $n - 1$ dimensions along which information propagates. As such, the classification of a PDE will affect its solution structure. This fact can be seen by applying a Fourier transformation to the 2nd order linear PDE operator to get

$$\hat{L} = ak_t^2 + bk_tk_x + ck_x^2 + dk_t + ek_x, \tag{2.4}$$

which is a quadratic in k_t and k_x. The solutions of this equation are defined by $\hat{L} = 0$, which illustrates the geometric differences underlying the structures of the solutions of the PDEs, as these solutions will correspond to ellipses, hyperbolas, or parabolas depending on the classification of the PDE. This relates back to the characteristics in an intuitive way, as information will propagate along the characteristics of a hyperbolic PDE out towards infinity in the same way that a hyperbola will approach infinity, while this behavior is not present in elliptic differential operators which tend to smooth out solutions.

2.1.2 Advection Equation

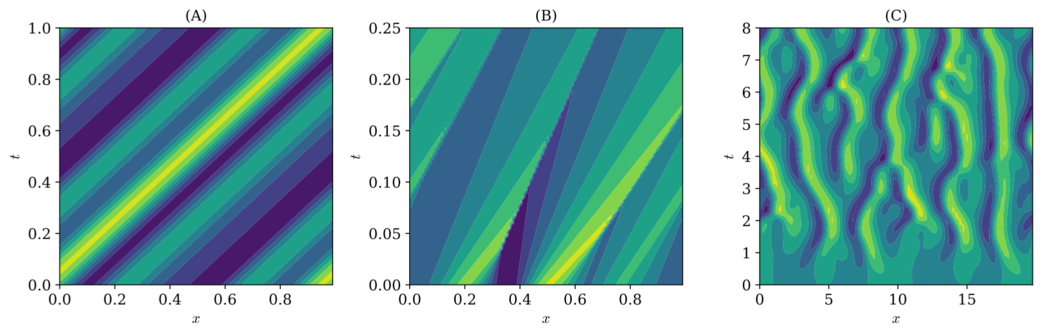


Figure 2.1: Example solution of (A) linear advection, (B) inviscid Burgers', and (C) KS equations for random initial conditions

The linear advection equation is written as

$$\frac{\partial u}{\partial t} + a\frac{\partial u}{\partial x} = 0. \tag{2.5}$$

It possesses solutions that are translations of the initial conditions at wavespeed a, i.e. for an initial condition of $u(x, 0) = f(x)$, the exact solution is $u(x, t) = f(x - at)$.

In terms of coarse-graining, the sub-grid behavior has no influence on the solution at subsequent times. Once discretization occurs, no inference can be made about what happens between grid points, as this is fully determined by the initial condition rather than the dynamics of the equation. When this equation is solved with a discontinuous initial condition, it serves as a toy problem for advecting different materials in a multicomponent/multiphase flow. Numerical error causes the discontinuity to become smeared out during the simulation.

2.1.3 Inviscid Burgers' Equation

The inviscid Burgers' equation is written in non-conservative form as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0, \tag{2.6}$$

and is used to model wave-breaking. The nonlinear term causes shockwaves to form in finite time from smooth initial conditions. A shockwave is a special case of a discontinuity that is forced by the dynamics: unlike the linear advection equation, the discontinuity can propogate without progressive diffusion.

2.1.4 Kuramoto-Sivashinsky Equation

The Kuramoto-Sivashinsky (KS) equation is written as

$$\frac{\partial u}{\partial t} + \nu\frac{\partial^4 u}{\partial x^4} + \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\frac{\partial u^2}{\partial x} = 0, \tag{2.7}$$

and is used as a toy problem for turbulent flame fronts. Its dynamics lead to chaotic spatiotemporal behavior (Hyman and Nicolaenko, 1986). Chaos for PDEs is analagous to dynamical systems, defined by a small perturbation in initial conditions drastically affecting the time-evolution of the system (Strogatz, 2001).

2.1.5 Euler Equations

The Euler equations describe inviscid fluid flow. Each equation accounts for a different conserved fluid property. In one dimension, they are written as

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} = 0, \tag{2.8}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial(P + \rho u^2)}{\partial x} = 0, \tag{2.9}$$

$$\frac{\partial E}{\partial t} + \frac{\partial((E + P)u)}{\partial x} = 0, \tag{2.10}$$

$$P = (\gamma - 1)(E - \frac{1}{2}\rho u^2), \tag{2.11}$$

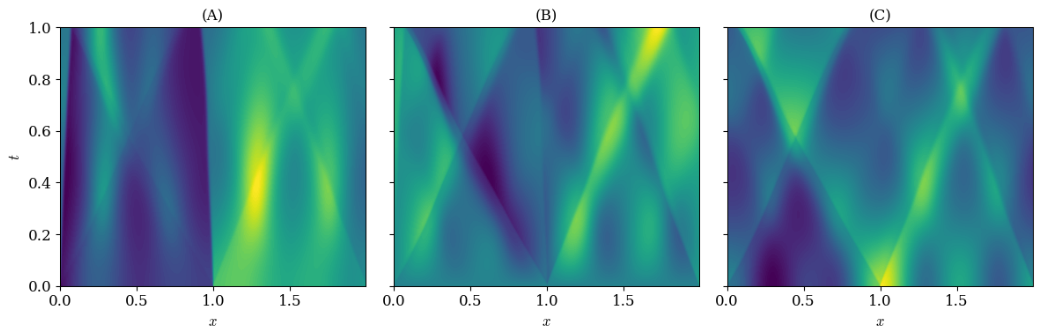


Figure 2.2: Example solution of the 1D Euler equations for random initial conditions, showing (A) density, (B) velocity, and (C) pressure

and describe conservation of mass, momentum, and total energy respectively. The last equation is the equation of state for an ideal gas. In two dimensions they are written as

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0, \tag{2.12}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial(P + \rho u^2)}{\partial x} + \frac{\partial(\rho u v)}{\partial y} = 0, \tag{2.13}$$

$$\frac{\partial \rho v}{\partial t} + \frac{\partial(P + \rho v^2)}{\partial y} + \frac{\partial(\rho u v)}{\partial x} = 0, \tag{2.14}$$

$$\frac{\partial E}{\partial t} + \frac{\partial((E + P)u)}{\partial x} + \frac{\partial((E + P)v)}{\partial y} = 0, \tag{2.15}$$

$$P = (\gamma - 1)(E - \frac{1}{2}\rho(u^2 + v^2)), \tag{2.16}$$

and once again describe conservation of mass, x and y momentum, and total energy, as well as the equation of state for an ideal gas. They possess a more complex solution structure than the other PDEs discussed, as discontinuous initial conditions can split into shockwaves, rarefactions, and contact discontinuities.

2.2 Numerical methods background

2.2.1 The finite-difference method

One spatial discretization method we combine with machine learning is FDM. In this method, derivatives of the solution are approximated as a weighted combination of local values of the solution. For example, one could approximate the first derivative of a function at a point as

$$\frac{\partial u}{\partial x} \approx \frac{1}{\Delta x}\sum_{i=-1}^{1} c_i u_i = \frac{u_1 - u_{-1}}{2\Delta x} \tag{2.17}$$

by using coefficients $c_{-1} = -\frac{1}{2}$, $c_0 = 0$, and $c_1 = \frac{1}{2}$. We use the facts that the bounds on i can be expanded and the values of c_i are not fully constrained to allow a neural network to chose these coefficient values based on the local solution as $c_{a:b} = f(u_{a:b})$.

2.2.2 The finite-volume method

We also consider equations that are more natually solved using FVM, a spatial discretization method similar to FDM. FVM offers the advantage of improved stability (Bar-Sinai et al., 2019), and is easier to extend to unstructured meshes (Chen, Liu, and Beardsley, 2003). Rather than splitting the domain up into a grid of nodes and solving the original PDE as is done in FDM, FVM splits the domain up into cells, and the integral form of the PDE is solved for each cell. The average value of the solution over the cell is modeled, and interpolated to determine fluxes between cells. This concept is illustrated by the example below.

2.2.2.1 Example

Consider the scalar conservation law

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{2.18}$$

One can split the x-domain into cells of width Δx and average over them as

$$\frac{1}{\Delta x}\int_{x_i}^{x_i+\Delta x} \frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} dx = 0 \tag{2.19}$$

to get

$$\frac{\partial \bar{u}_i}{\partial t} + \frac{f(u(x_i+\Delta x)) - f(u(x_i))}{\Delta x} = 0. \tag{2.20}$$

Note that this equation is still exact. The cell average values $\bar{u}$ are tracked and interpolated to find u locally as

$$u(x_i) \approx \sum_{j=a}^{b} c_j \bar{u}_j. \tag{2.21}$$

Once again, the coefficients c_j are not fully constrained and can be determined using machine learning.

2.2.3 Time stepping methods

Runge-Kutta methods are a family of numerical integration methods used to advance a differential equation $\frac{\partial u}{\partial t} = L(u)$ forward in time. In this thesis, we use SSPRK3 (Gottlieb and Shu, 1998)

$$\begin{aligned} u^{(1)} &= u^{(n)} + \Delta t L(u^{(n)}), \\ u^{(2)} &= \frac{3}{4}u^{(n)} + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)}), \\ u^{(n+1)} &= \frac{1}{3}u^{(n)} + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)}), \end{aligned} \tag{2.22}$$

a three-step, third-order accurate method that preserves the total variation diminishing (TVD) property $TV(u(t_{i+1})) \leq TV(u(t_i))$ of explicit Euler, where total variation is defined as

$$TV(u) = \sum_{i=1}^{N} |u(x_i) - u(x_{i-1})|. \tag{2.23}$$

A TVD method prevents the time-stepping method from adding spurious oscillations to the solution, which is a major concern for PDEs with discontinuous solutions, as they can lead to instabilities that crash the simulation in nonlinear PDEs such as the inviscid Burgers' equation.

2.2.4 Lax Equivalence Theorem

An important characteristic of any approximation method is whether or not it converges to the true solution. When solving a PDE numerically, one can use the Lax Equivalence theorem to determine whether or not a given simulation will converge. The theorem states that if the method is consistent and stable, then convergence is guaranteed (LeVeque, 1992).

Consistency is achieved if the FDM or FVM algorithm used for the simulation correctly approximates the spatial derivative terms. Consistency can be verified by examining the local truncation error associated with the spatial discretization method. For example, for the centered difference scheme

$$\frac{\partial u}{\partial x} \approx \frac{u_1 - u_{-1}}{2\Delta x} \tag{2.24}$$

we can perform a Taylor expansion of each term to get

$$\begin{aligned} u_1 &= u_0 + \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial^2 u}{\partial x^2} + \frac{\Delta x^3}{6}\frac{\partial^3 u}{\partial x^3} + \ldots \\ u_{-1} &= u_0 - \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2}\frac{\partial u}{\partial x^2} - \frac{\Delta x^3}{6}\frac{\partial^3 u}{\partial x^3} + \ldots \end{aligned} \tag{2.25}$$

Substituting these expressions into the difference scheme gives

$$\frac{\partial u}{\partial x} \approx \frac{\partial u}{\partial x} + \frac{\Delta x^2}{3}\frac{\partial^3 u}{\partial x^3} + \ldots \tag{2.26}$$

so one can see that as Δx approaches 0, the difference will converge to the derivative, making the method consistent. Because the leading order error term is proportional to Δx^2, this method is said to be second order accurate.

In the context of solving PDEs, stability refers to the accumulation of errors being bounded as the equation is stepped through time. In a linear PDE, this is achieved if the total variation of the numerical solution is bounded as Δt approaches 0. Stability can be analyzed for linear PDEs by Von Neumann stability analysis, but is nontrivial to analyze for nonlinear PDEs, and instead must be verified empirically for specific problems (LeVeque, 1992).

2.2.5 Modern shock capturing methods

Shock capturing methods aim to accomplish the difficult task of simulating PDEs with discontinuous solutions with no special treatment of the discontinuities. What this means is that the same algorithm can be applied everywhere, and one does not need to explicitly track the location of the discontinuity. This approach has the advantage of being simple and reliable, with the disadvantage of smearing the discontinuity over several cells.

One difference between classical and modern shock capturing methods is that modern methods tend to use upwind-biased stencils, while classical methods tend to use a symmetric stencil. By using a stencil that is biased in the direction of the flow, the algorithm uses information that is more physically relevant to determining the solution at the next time step, which can lead to improved accuracy and stability (Yee, 1987).

In order for a shock capturing method to be stable, some form of numerical dissipation must be added to damp out spurious oscillations. This detail gives rise to another important difference between classical and modern shock capturing methods: classical shock capturing methods apply a constant amount of artificially viscosity to the entire domain, while modern shock capturing methods can adapt this quantity based on local flow properties such as smoothness. This adaptivity allows modern shock capturing methods to maintain a high order of accuracy in smooth regions of the flow, which is why they are also sometimes referred to as high-resolution schemes.

An important example of a modern shock capturing method is ENO. The ENO algorithm works by computing the flux on several different sub-stencils, measuring the smoothness on each sub-stencil, and then selecting the flux on the sub-stencil where the solution is smoothest. An example of this for a five point stencil broken up into three three point sub stencils can be seen below (Harten, 1983).

First, the fluxes are computed on each stencil as

$$\begin{aligned}
\hat{f}^{(1)}_{j+1/2} &= \frac{1}{3}f(u_{j-2}) - \frac{7}{6}f(u_{j-1}) + \frac{11}{6}f(u_j) \\
\hat{f}^{(2)}_{j+1/2} &= -\frac{1}{6}f(u_{j-1}) + \frac{5}{6}f(u_j) + \frac{1}{3}f(u_{j+1}) \\
\hat{f}^{(3)}_{j+1/2} &= \frac{1}{3}f(u_j) + \frac{5}{6}f(u_{j+1}) - \frac{1}{6}f(u_{j+2})
\end{aligned} \tag{2.27}$$

and the corresponding smoothness indicators are computed as

$$\begin{aligned}
\hat{\beta}^{(1)} &= \frac{13}{12}(f(u_{j-2}) - 2f(u_{j-1}) + f(u_j))^2 + \frac{1}{4}(f(u_{j-2}) - 4f(u_{j-1}) + 3f(u_j))^2 \\
\hat{\beta}^{(2)} &= \frac{13}{12}(f(u_{j-1}) - 2f(u_j) + f(u_{j+1}))^2 + \frac{1}{4}(f(u_{j-1}) - f(u_{j+1}))^2 \\
\hat{\beta}^{(3)} &= \frac{13}{12}(f(u_j) - 2f(u_{j+1}) + f(u_{j+2}))^2 + \frac{1}{4}(3f(u_j) - 4f(u_{j+1}) + f(u_{j+2}))^2
\end{aligned} \tag{2.28}$$

The flux is then reported as $\hat{f}^{(k)}_{j+1/2}$, where

$$k = \underset{i\in\{1,2,3\}}{\arg\min} \quad \hat{\beta}^{(i)} \tag{2.29}$$

The ENO algorithm was then used as inspiration for creating the weighted ENO, or WENO-JS, algorithm (Jiang and Shu, 1996). Rather than simply selecting the flux with the smoothest local solution values, a weighted average of these fluxes is taken based on the smoothness indicators.

The smoothness indicators are used to compute nonlinear weights as

$$\begin{aligned} \hat{w}_k &= \frac{\gamma_k}{(\epsilon + \beta^{(k)})^2} \\ \gamma_1 &= \frac{1}{10} \\ \gamma_2 &= \frac{3}{5} \\ \gamma_3 &= \frac{3}{10} \end{aligned} \tag{2.30}$$

where ϵ is taken to be a small value, typically $1E - 6$. The nonlinear weights are then scaled such that they add to one as

$$w_k = \frac{\hat{w}_k}{\sum_{i=1}^{3} \hat{w}_i} \tag{2.31}$$

and the flux is finally computed as

$$\hat{f}_{j+1/2} = w^{(1)} \hat{f}^{(1)}_{j+1/2} + w^{(2)} \hat{f}^{(2)}_{j+1/2} + w^{(3)} \hat{f}^{(3)}_{j+1/2} \tag{2.32}$$

2.2.6 Lax-Friedrichs Flux Splitting

As mentioned above, modern shock capturing algorithms tend to use an upwind biased stencil. For example, in the ENO and WENO scheme presented previously information is used from three cells to the left of the location where the flux is to be computed, while only two cells are used from the right. Hence, depending on the direction of the characteristic waves the simulation may become unstable. One method that addresses this problem is Lax-Friedrichs flux splitting (LeVeque, 1992).

For a system of hyperbolic conservation laws with vector valued u

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \tag{2.33}$$

the Jacobian of $f(u)$ w.r.t. u must be positive semidefinite for an upwind scheme to remain stable. If that property is not satisfied automatically by the original hyperbolic conservation law, one can split the flux as

$$f(u) = f^+(u) + f^-(u) \tag{2.34}$$

where $f^+(u)$ has a positive semidefinite Jacobian and $f^-(u)$ has a negative semidefinite Jacobian. One simple method for computing $f^+(u)$ and $f^-(u)$ is to check the eigenvalues of the Jacobian of $f(u)$, which represent wave speeds of the characteristics, and then adding them to the flux as

$$\begin{aligned} f^+(u) &= \frac{1}{2}(f(u) + \lambda_{max}u) \\ f^-(u) &= \frac{1}{2}(f(u) - \lambda_{max}u) \end{aligned} \tag{2.35}$$

where λ_{max} is the highest magnitude eigenvalue of the Jacobian of $f(u)$. For many physical systems, these wavespeeds have simple closed form solutions. For example, for the 2D euler equations these wavespeeds are simply $v - c, v, v$ and $v + c$, where $v = \sqrt{u_x^2 + u_y^2}$.

One can then compute $f^+(u)$ exactly as described in the previous section with WENO-JS, and $f^-(u)$ by simply mirroring the algorithm about $x_{j+1/2}$, so that there are now 2 cells on the left and 3 on the right.

2.3 Machine Learning background

2.3.1 Regression and Neural Networks

In general, regression refers to the process of optimizing a model to estimate the relationships between dependent variables and independent variables to match a dataset. Given a dataset that has independent variables x_i and dependent variables y_j, we assume that these inputs can be mapped to the outputs via some parametric function $y = f(x; \theta)$ with parameters θ. To determine these parameters, we choose a loss function, which is often taken to be the square error $(y - f(x; \theta))^2$ and minimize its expectation $\mathbb{E}[(y - f(x; \theta))^2]$. In practice, this can be achieved using the given data and minimizing the sum square error as

$$\theta = \underset{\hat{\theta}\in\mathbb{R}^k}{\arg\min} \quad \textstyle\sum_{i=1}^{N}(y_i - f(x_i; \hat{\theta}))^2 \ . \tag{2.36}$$

One of the simplest examples of the regression function is linear regression. In this case, we simply take the inner product of the parameters and the inputs, and add a bias term

$$f(x; \theta) =< x, \theta > +\theta_0. \tag{2.37}$$

Another popular regression model is logistic regression, which is defined as

$$f(x;\theta) = \frac{1}{1 + \exp(< x, \theta > + \theta_0)}. \tag{2.38}$$

This idea of a regression function can then be extended to create neural networks, a popular and highly flexible class of regression functions. This is done by composing affine transformations with nonlinear 'activation functions', such as the sigmoid function seen above in logistic regression, as the series of operations

$$\begin{aligned} y_1 &= f_1(x;\Theta^{(1)}) = g(\Theta^{(1)}x + \Theta_0^{(1)}) \\ y_2 &= f_2(y_1;\Theta^{(2)}) = g(\Theta^{(2)}y_1 + \Theta_0^{(2)}) \\ y_3 &= f_3(y_2;\Theta^{(3)}) = g(\Theta^{(3)}y_2 + \Theta_0^{(3)}) \\ &\vdots \\ y &= f_n(y_{n-1};\Theta^{(n)}) = g(\Theta^{(n)}y_{n-1} + \Theta_0^{(n)}) \end{aligned} \tag{2.39}$$

where Θ^i represents a matrix of neural network weights, Θ_0^i represents a vector of neural network biases, and as mentioned above $g(.)$ is an activation function that is applied to each entry of vector $\Theta^{(i)}y_{i-1} + \Theta_0^{(i)}$. Two of the most popular activation functions are the sigmoid function

$$g(x) = \frac{1}{1 + \exp(x)} \tag{2.40}$$

and the ReLU function

$$g(x) = \max(x, 0). \tag{2.41}$$

Plots of these functions can be seen in Figure 2.4. The simulations in this thesis use the ReLU function as the activation function.

2.3.2 Other network architectures

The section above describes a feed-forward, fully connected neural network. However, this idea can be extended to other architectures that are better suited to problems that contain certain structures. Two architectures that will be used in this thesis are the fully convolutional neural network, and the LSTM network.

A convolutional neural network can be used to take advantage of problems that have spatially local structure. One of the canonical examples of this is image recognition,

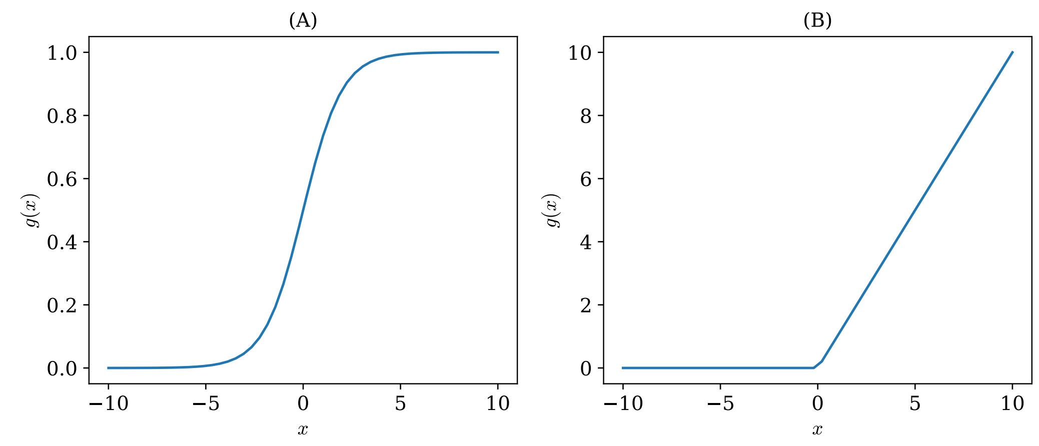


Figure 2.3: Examples of popular activation functions (A) sigmoid, and (B) ReLU

as intuitively, pixels form structures with other nearby pixels. PDEs also have a spatially local structure, which is already taken of advantage of by many standard numerical methods such as FVM. In fact, these methods are structurally identical to the fully convolutional network that is used in every model presented in this thesis. Conceptually, the fully convolutional network is very simple. Rather than taking the entire computational domain as an input to the neural network, the network only takes in several adjacent values, analogous to the computational stencil of a FVM or FDM. The network is then applied to each location in the domain independently. Expressed mathematically, the difference can be shown by considering a fully connected neural network to have the following input-output structure

$$\frac{\partial u_{1:N}}{\partial x} = f(u_{1:N}) \tag{2.42}$$

where every input is used to compute every output, while a fully convolutional neural network would evaluate the inputs as

$$\frac{\partial u_i}{\partial x} = f(u_{i-a:i+b}) \tag{2.43}$$

where only the local values are used to compute the output. This approach drastically reduces the number of parameters in the neural network, and allows the learned method to generalize to grids with an arbitrary number of nodes, cells or elements.

An LSTM network can be used to take advantage of problems that have temporal structure, which is present in time dependent PDEs. This architecture will be used in chapter 4 of this thesis for the FiniteNet algorithm. It is a type of recurrent neural network (RNN). An RNN is a neural network that propogates information forward in time, and accepts some type of time-dependent input. Compared to a standard neural network, in addition to the prediction, the network also outputs hidden information that is used as an input to the network at the next timestep. Mathematically, a simple example of this can be expressed as

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \\ y_t &= g(x_t, h_{t-1}) \end{aligned} \tag{2.44}$$

where h_t represents the hidden information at time t, x_t represents the input at time t and y_t represents the output at time t.

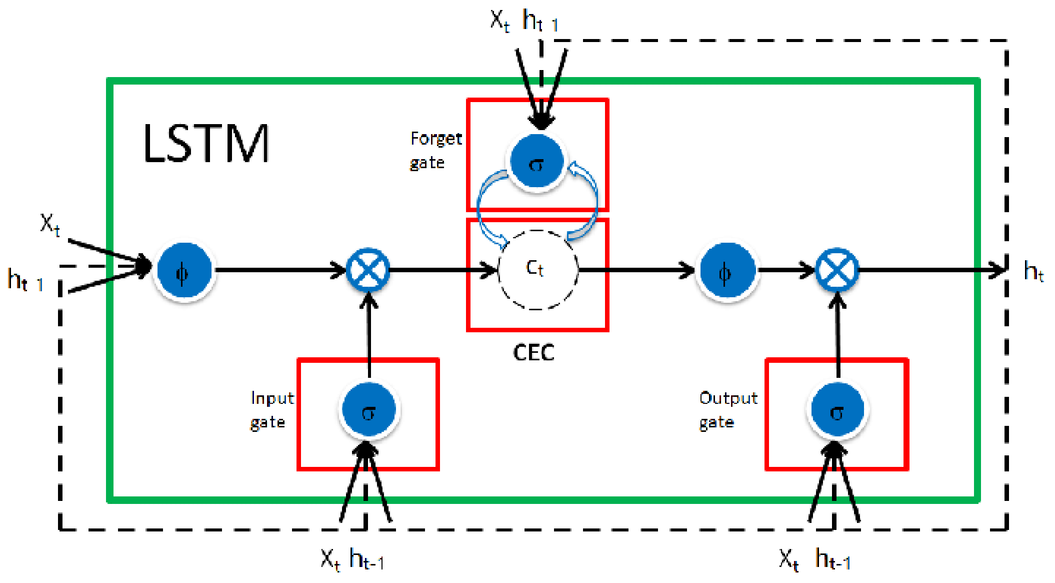


Figure 2.4: Diagram of LSTM cell (from Su and Kuo, 2019)

Unfortunately, classical RNNs such as the one shown in equation 2.44 run into practical problems with vanishing gradients when information must be propagated over long time horizons. LSTMs address this problem by allowing gradient information to be unchanged as it passes through the network. This property is achieved by including a ’cell’ which can remember information for an arbitrarily long time, as well as an input gate, output gate, and forget gate which regulate the flow of information in and out of the cell. Mathematically, an LSTM with a forget gate can be expressed as

$$\begin{aligned}
f_t &= \sigma_g(W_t x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &=< f_t, c_{t-1} > + < i_t, \tilde{c}_t > \\
h_t &=< o_t, \sigma_h(c_t) >
\end{aligned} \tag{2.45}$$

where f represents the forget gate, i represents the input gate, o represents the output gate, c represents the cell, W and U represent the network weights, b represents the network biases, σ represents the activation functions, and h represents the hidden information. This model can also be generalized to deep LSTMs, which have multiple layers.

2.3.3 Transfer Learning

Transfer learning does not refer to any specific neural network architecture or model, but rather the reuse of what has been learned in one task for another task (Torrey and Shavlik, 2010). For example, say one were to train a neural network to determine whether or not an image has a cat in it. This task would require a significant amount of data. Then, suppose that they wanted to train another neural network to determine whether or not an image has a dog in it. This task is very similar to determining if an image has a cat in it, so one could actually use the neural network as a starting point, and only retrain the last few layers on the dog dataset. This situation describes an example of transfer learning, as we are transferring what we learned about recognizing cat pictures to the slightly different task of recognizing dog pictures. This example can be seen visually in Figure 2.5.

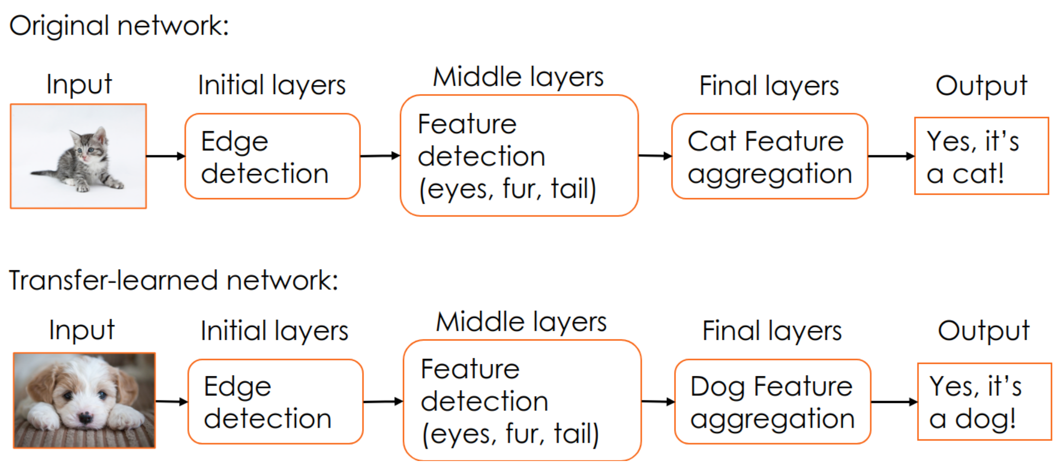


Figure 2.5: Example of transfer learning

Chapter 3

ENHANCEMENT OF SHOCK CAPTURING METHODS VIA MACHINE LEARNING

3.1 Introduction

For some initial-boundary value problems (IBVP) in fluid mechanics, the solution of the PDEs include discontinuous initial data or a discontinuity that forms in finite time, i.e. shockwaves. Numerical methods for solving these PDE must be specially tailored to properly resolve these discontinuities (LeVeque et al., 2002).

In this chapter, we attempt to train a neural network to improve WENO5-JS. Our goal is to get closer to the optimal nonlinear finite-volume coefficients while introducing a minimal amount of bias. Unlike other references, we do not directly change the smoothness indicators or nonlinear weights of the method. Instead, we use a neural network to perturb the finite-volume coefficients determined using the original smoothness indicators and nonlinear weights of WENO5-JS. We attempt to learn an optimal function for this perturbation using data generated from waveforms that are representative of solutions of PDEs. These modifications result in a finite-volume scheme that diffuses fine-scale flow features and discontinuities less severely than WENO5-JS. We start in the next section by giving a more detailed description of the proposed algorithm.

While many examples from WENO literature build off of each other rather than starting from WENO-JS, we will base our method on WENO-JS because our strategy for developing the method does not resemble other methods. However, our methodology could easily adopt various improvements that have been made to WENO-JS.

3.2 Numerical Methods

3.2.1 Description of WENO-NN

Although we focus on WENO5-JS in this chapter, our approach could generally be used to enhance any shock capturing method (or perhaps any numerical method). The proposed algorithm involves pre-processing the flow variables on a stencil using a conventional shock capturing method and feeding those results into a neural network. The neural network then perturbs the results of the shock capturing method. Post-processing is then applied to the output of the neural network to guarantee consistency (Bar-Sinai et al., 2019) (or, more generally, could be used to enforce other desirable properties). Hence, the augmented numerical scheme takes on many properties of the original. For example, applying the method to WENO5-JS results in an upwind-biased finite volume method with coefficients that depend on the local solution. The steps of the algorithm for enhancing WENO5-JS can be seen in algorithm 1.

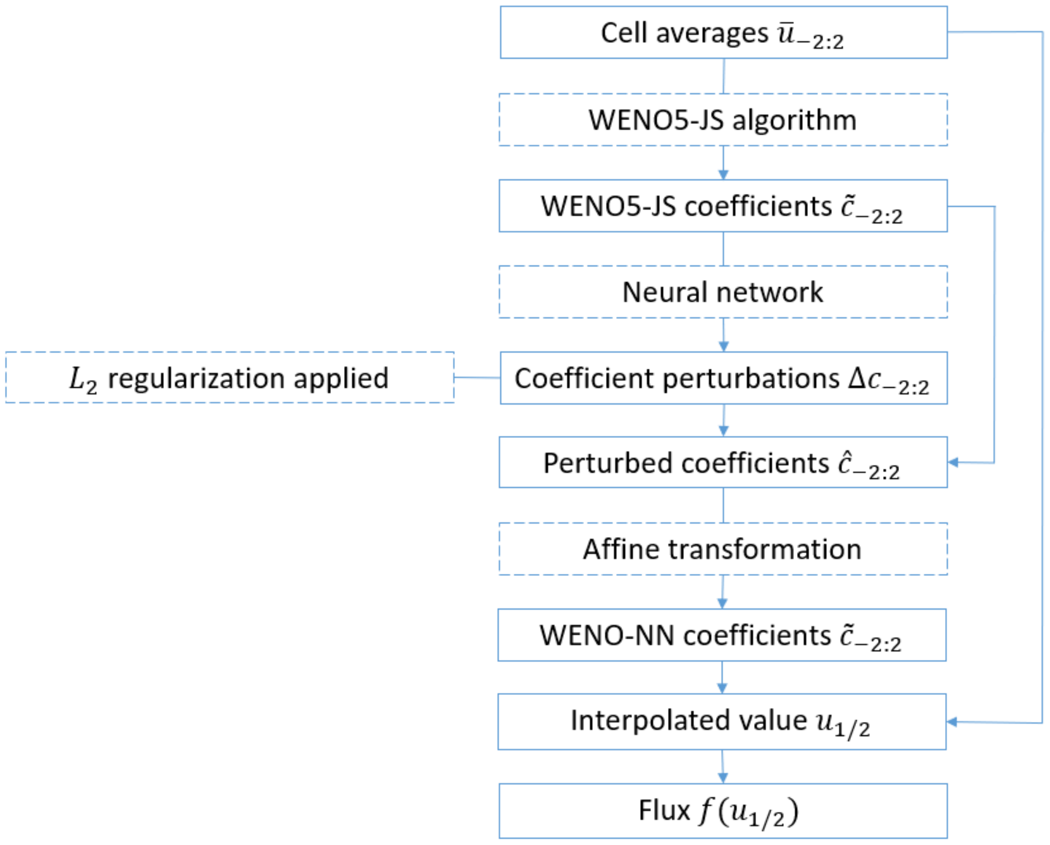

Cell averages $\bar{u}_{-2:2}$
WENO5-JS algorithm
WENO5-JS coefficients $\tilde{c}_{-2:2}$
Neural network
L_2 regularization applied
Coefficient perturbations $\Delta c_{-2:2}$
Perturbed coefficients $\hat{c}_{-2:2}$
Affine transformation
WENO-NN coefficients $\tilde{c}_{-2:2}$
Interpolated value $u_{1/2}$
Flux $f(u_{1/2})$

Figure 3.1: Diagram of WENO-NN algorithm

Algorithm 1 WENO-NN Algorithm

```
1: procedure WENONN
2:     Begin with cell averages $\bar{u}_{j-2:j+2}$
3:     Scale the cell averages
4:     Compute coefficients $\tilde{c}_{j-2:j+2}$ with WENO5-JS
5:     Compute change in coefficients $\Delta\tilde{c}_{j-2:j+2}$ with neural network
6:     Compute new coefficients $\hat{c}_{j-2:j+2} = \tilde{c}_{j-2:j+2} - \Delta\tilde{c}_{j-2:j+2}$
7:     Compute final coefficients $c_{j-2:j+2}$ by transforming $\hat{c}_{j-2:j+2}$
8:     Compute cell edge value $u_{j+1/2} = c_{j-2:j+2} \cdot \bar{u}_{j-2:j+2}$
```

We use WENO5-JS to pre-process the input data, so that the input to the neural network is the set of finite-volume coefficients found by WENO5-JS. We found that including this pre-processing step significantly improved performance. Once the nonlinear weights w_i are determined according to the WENO5-JS algorithm, the coefficients for each cell average are computed as

$$\begin{aligned}
\tilde{c}_{-2} &= \frac{1}{3}w_1, \\
\tilde{c}_{-1} &= -\frac{7}{6}w_1 - \frac{1}{6}w_2, \\
\tilde{c}_0 &= \frac{11}{6}w_1 + \frac{5}{6}w_2 + \frac{1}{3}w_3, \\
\tilde{c}_1 &= \frac{1}{3}w_2 + \frac{5}{6}w_3, \\
\tilde{c}_2 &= -\frac{1}{6}w_3.
\end{aligned} \tag{3.1}$$

These five coefficients are the inputs to the neural network, which outputs a change in each coefficient, $\Delta\tilde{c}_i$. Our neural network uses 3 hidden layers, each with 3 neurons. We deliberately make the network as small as possible to reduce the computational cost of evaluating it. We are able to use such a small network because assuming that the WENO5-JS coefficients are a useful model input is a strong prior, so WENO5-JS performs a significant amount of the required processing. Additionally, we noticed that empirically, increasing the network size did not lead to a meaningful accuracy improvement. L_2 regularization is applied to the output of the neural network to penalize deviations from WENO5-JS, which encourages the network to only change the answer supplied by WENO5-JS when an improved result is expected. The new coefficients are computed by subtracting the change in coefficients from the old coefficients.

Additionally, the size of the input space is reduced by scaling cell averages within the stencil as

$$\bar{\mathbf{u}}_s = \frac{\bar{\mathbf{u}} - \min \bar{\mathbf{u}}}{\max \bar{\mathbf{u}} - \min \bar{\mathbf{u}}}. \tag{3.2}$$

If all the cell averages have the same value, the scaling equation fails so the value at the cell edge is simply set to the cell average value.

To guarantee that WENO-NN is consistent, we apply an affine transformation to these coefficients that guarantees that they sum to one (Bar-Sinai et al., 2019). We derive this transformation by solving the optimization problem

$$\begin{aligned} \min_{\mathbf{c}\in\mathbb{R}^5} \quad & \textstyle\sum_{n=-2}^{2}(c_n - \hat{c}_n)^2 \\ \text{s.t.} \quad & \textstyle\sum_{n=-2}^{2}(c_n) = 1, \end{aligned} \tag{3.3}$$

which can be reformulated with the substitution $\Delta\mathbf{c} = \mathbf{c} - \hat{\mathbf{c}}$ to pose the problem as finding the minimum norm solution to an under-constrained linear system

$$\begin{aligned} \min_{\Delta\mathbf{c}\in\mathbb{R}^5} \quad & \textstyle\sum_{n=-2}^{2}(\Delta c_n)^2 \\ \text{s.t.} \quad & \textstyle\sum_{n=-2}^{2}(\hat{c}_n + \Delta c_n) = 1, \end{aligned} \tag{3.4}$$

which has the analytical solution

$$\Delta c_i = \frac{1 - \sum_{n=-2}^{2} \hat{c}_n}{5}. \tag{3.5}$$

One can use the same approach to enforce arbitrarily high orders of accuracy since the optimization problem has an analytical solution for any constraint matrix of sufficiently high rank

$$\begin{aligned} \min_{\Delta\mathbf{c}\in\mathbb{R}^5} \quad & \textstyle\sum_{n=-2}^{2}(\Delta c_n)^2 \\ \text{s.t.} \quad & A(\hat{\mathbf{c}} + \Delta\mathbf{c}) = \mathbf{b}. \end{aligned} \tag{3.6}$$

This optimization problem has analytical solution $\Delta\mathbf{c} = A^T(AA^T)^{-1}(\mathbf{b} - A\hat{\mathbf{c}})$ when AA^T is invertible.

We verify that our constraint is satisfied by looking at the convergence rate of WENO-NN for a smooth solution. For this test case, we will simply use WENO-NN, WENO5-JS, and WENO1 to take the derivative of $u(x) = \sin(4\pi x) + \cos(4\pi x)$, and compare the results to the analytical solution $\frac{\partial u}{\partial x}^* = -4\pi\sin(4\pi x) + 4\pi\cos(4\pi x)$ using the error metric

$$E = \sqrt{\frac{||\frac{\partial u}{\partial x} - \frac{\partial u}{\partial x}^*||_2}{N}}. \tag{3.7}$$

In Figure 3.2, we can see that WENO-NN achieves first order accuracy, which confirms that the constraint is satisfied. We also see that, as expected, WENO5-JS converges at fifth order and WENO1 converges at first order as $\Delta x \to 0$. However, when discontinuities are present it is not possible to achieve better than first order accuracy with any finite volume method (LeVeque et al., 2002). Despite this fact, it

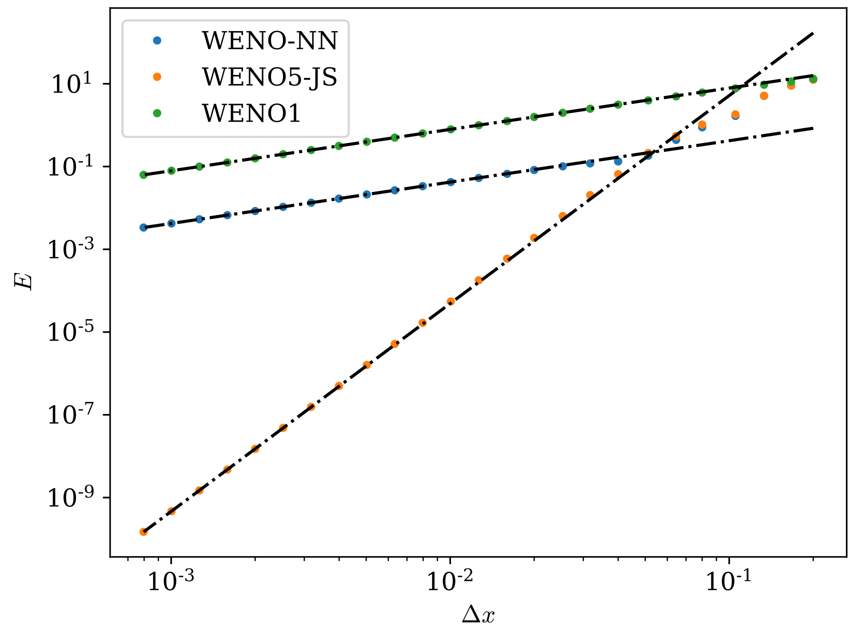


Figure 3.2: Convergence of WENO-NN, WENO5-JS, and WENO1 for smooth solutions

is advantageous to use WENO5-JS over WENO3-JS in such situations, as WENO5-JS still tends to give lower error in discontinuous problems (Shu, 1998), which is why we chose to use WENO5-JS for processing the cell average values despite the fact that WENO-NN ends up being first-order accurate. Similarly, we see that for some discontinuous problems, WENO-NN gives lower error than WENO5-JS. If a higher order of accuracy is desired in smooth regions of the flow, one could develop a hybrid method using WENO-NN and any high-order method.

3.2.2 Other Numerical Methods Used

For all simulations shown, we use a third-order TVD Runge-Kutta scheme (Gottlieb and Shu, 1998) as our time-stepping method

$$\begin{aligned} u^{(1)} &= u^{(n)} + \Delta t L(u^{(n)}), \\ u^{(2)} &= \frac{3}{4}u^{(n)} + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)}), \\ u^{(n+1)} &= \frac{1}{3}u^{(n)} + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)}). \end{aligned} \tag{3.8}$$

For flux-splitting, we use a Lax-Friedrichs flux splitting procedure (Shu, 2003)

$$f^{\pm}(u) = \tfrac{1}{2}(f(u) \pm \alpha u),$$
$$\alpha = \max_u |f'(u)|. \tag{3.9}$$

In this expression, $f(u)$ is defined as the flux of a 1-D hyperbolic conservation law $\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0$. When solving the 1-D Euler equations, we apply the flux splitting to the characteristic decomposition of the system. For our numerical Riemann solver, we use the Lax-Friedrichs method (Chu, 1979).

3.3 Machine Learning Methodology

We construct our training data directly from known functions that we expect to represent the waveforms that WENO-NN will encounter in practice. Note that the same dataset is used for every PDE, as we train only one network and use it for every WENO-NN result shown in this paper. However, one could develop a problem-specific dataset if desired. For each datapoint, we start with some function $u(x)$ and a discretized domain of n cells. The cell average is evaluated on each cell as

$$\bar{u}(x_i) = \frac{1}{\Delta x} \int_{x_i - \frac{\Delta x}{2}}^{x_i + \frac{\Delta x}{2}} u(x) dx, \tag{3.10}$$

and because we chose the form of $u(x)$ we can evaluate the cell average analytically. We also evaluate the function value on the cell boundary as $u(x_i + \Delta x/2)$ analytically. We then move along the domain and form the dataset based on the stencil size. So for WENO-NN, one datapoint involves 5 cell averages as the input with the function value on the cell boundary as the output. The functions we use when creating the dataset are step functions, sawtooth waves, hyperbolic-tangent functions, sinusoids, polynomials, and sums of the above. Sinusoids and polynomials broadly cover most smooth solutions that would be encountered in practice, and we specifically chose hyperbolic-tangent functions because they mimic smeared out discontinuities. Similarly, step functions and sawtooth waves mimic contact discontinuities and shocks. We included sums of these functions to mimic complex flow fields that may contain shocks within a turbulent flow. We found that including more data made the resulting scheme less diffusive and less oscillatory.

When adding a new entry to the dataset, we first check to see if it is close to other points already present in the L_2 sense. Sufficiently close points are not added to the dataset to prevent redundant data that will slow down the training process. The resulting dataset has 75241 entries.

When training the network, we use the Adam optimizer (Kingma and Ba, 2014), split the data into batches of 80 points to estimate the gradient, and optimize for 10 epochs using the Keras package in python (Chollet et al., 2015). We trained the network from many different randomly chosen initial guesses of the parameters, and chose the best one based on performance in simulating the linear advection of a step function. We apply L_2 regularization with a constant of $\lambda = 0.1$ to the neural network output, and find that when splitting the data into a training set of 80% of the data and a validation set of the other 20% of the data our in-sample error is 0.569 and the out-of-sample error is 0.571, averaged from 100 trials of training on the dataset, so overfitting within the generated dataset is not a concern. This difference is so small because the model we are training is of relatively low complexity, and is essentially underfitting the generated dataset. We use mean squared loss as our objective function to minimize.

Despite the fact that we do not see overfitting within the generated dataset, we still observe overfitting when we apply the method to an actual simulation. Figure 3.3 shows the average training error, average validation error, and average error when using the method to simulate a PDE for different regularization values λ of the neural network output. The training and validation error are computed using the mean square error,

$$E_{data} = \frac{\sum_{i=1}^{N}(y_i - y_i^*)^2}{N}, \tag{3.11}$$

while the simulation error is computed by using the learned numerical method to linearly advect a step function and computing the L_2 error at the end of the simulation,

$$E_{simulation} = \sqrt{\int_0^L (\bar{u}(x,T) - \bar{u}^*(x,T))^2 dx}. \tag{3.12}$$

One can see that adding regularization causes error to increase in both the training and validation datasets but decreases the error in the simulation results. Hence, we can see that we are overfitting to the training data, but because the validation data does not show this, we can conclude that the dataset does not exactly match the distribution we are trying to approximate.

The following paragraph describes our model development process, and can be skipped without loss of continuity. Initially, our model involved constant coefficients

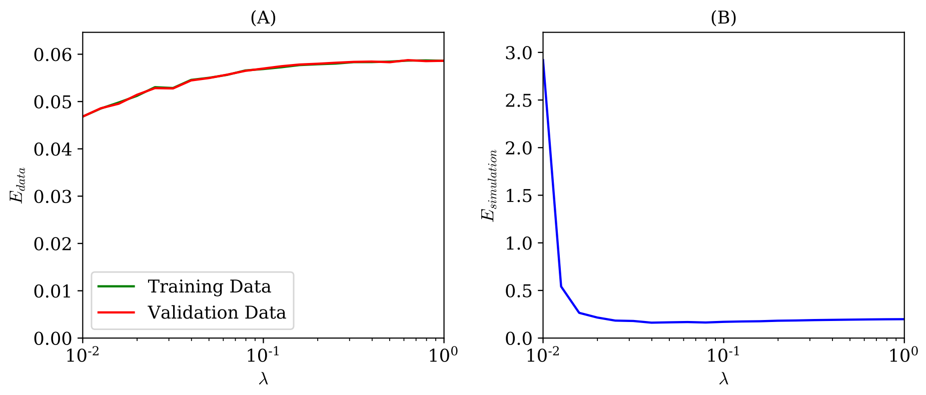


Figure 3.3: Comparing error trends between (A) exact generated data and (B) simulation results

rather than a neural network. We eventually found that the added flexibility of a neural network improved the performance of the numerical method. Initially, the neural network simply took the local solution values as the input and returned FVM coefficients as the output. Adding the affine transformation that guarantees consistency was found to significantly improve the performance of our numerical method. We initially enforced consistency by setting the neural network to output only four of the five coefficients, and choosing the last coefficient such that the method is consistent. We ultimately changed this constraint to the optimization framework seen in this paper, as it seemed more elegant. However, this change had very little effect on performance. Using the neural network to perturb the WENO5-JS coefficients was found to significantly improve performance over having the neural network directly output the coefficients. When the neural network directly outputs the coefficients, the trained numerical method is empirically unstable. To fix this issue, we applied L_2 regularization to the coefficients to add damping by biasing the coefficients to take on similar values. We then realized that we could instead bias these coefficients towards a scheme that we already know is stable, and modified the network architecture to perturb the 5th-order, constant coefficient method that WENO5-JS converges to in the presence of smooth solutions. We then decided to have the neural network perturb the WENO5-JS coefficients, which we found greatly improved the performance of the method. We also found that using these coefficients as the input to the neural network instead of the local solution values offered further improvement. Finally, we also experimented with the network size. We found that past a certain point, increasing the depth of the network and the number of nodes per layer did not improve performance, even with optimized regularization parameters.

In order to minimize computational cost, we chose the smallest network that offered maximum performance, as this network is small and further decreasing its size was found to rapidly harm performance while offering very little speedup.

3.4 Results

3.4.1 Advection Equation

These results will focus on comparing WENO5-JS to WENO-NN. Every WENO-NN result we show in this paper was generated using the same neural network with the same weights. As such, our numerical method is broadly applicable to problems not discussed in this paper, in contrast with many machine learning solutions that are problem-specific. No additional training is necessary to use this method for other PDEs. The first test case we look at is the linear advection of a step function on a periodic domain. Mathematically, this IBVP is posed as

$$\begin{gathered}\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \\ u(0,x) = \begin{cases} 1, & \text{if } x \geq L/2, \\ 0, & \text{otherwise,} \end{cases} \\ u(t,0) = u(t,L).\end{gathered} \tag{3.13}$$

For this simulation, we set $c = 1$ and $L = 2$. We split the domain into 100 cells, use a CFL number of 2/3, and run the simulation for 50 periods for a total time of $T = 100$. Figure 3.4 shows the solution of this PDE for WENO5-JS and WENO-NN at $t = 0, 20, 50$ and 100. The solution at $t = 0$ is also the exact solution at all the other times plotted.

One can see that the solution using WENO-NN provides a closer visual fit to the exact solution, as WENO5-JS diffuses the discontinuity more significantly than WENO-NN. WENO5-JS also introduces noticeable overshoot behind the discontinuity. The neural network has the interesting property that the waveform is nearly invariant to its propagation, while WENO5-JS continues to diffuse the solution. This behavior can be explained by examining the artificial fluid properties associated with the modified equation obtained by Taylor series expansion (assuming linearity of the scheme). The modified PDE is

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} + \delta\frac{\partial^3 u}{\partial x^3} - \sigma\frac{\partial^4 u}{\partial x^4} + \dots \tag{3.14}$$

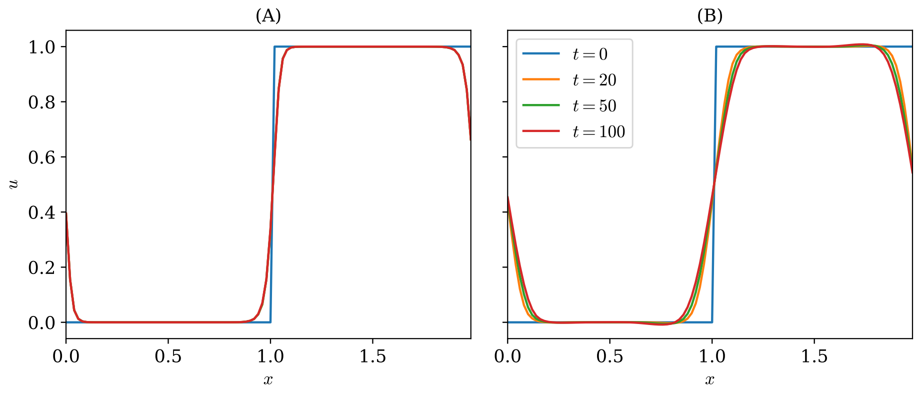


Figure 3.4: Numerical solutions of the advection equation at $t = 0, 20, 50$ and 100 using (A) WENO-NN and (B) WENO5-JS. Note that the curves in (A) for $t > 0$ are indistinguishable.

The expansions give expressions for the artificial viscosity, dispersion, and hyperviscosity, $\frac{\partial \bar{u}}{\partial t} + \frac{u(x+\frac{\Delta x}{2}) - u(x-\frac{\Delta x)}{2}}{\Delta x} = 0$ after making the substitutions $u(x+\frac{\Delta x}{2}) = \sum_{n=-2}^{2} c_n \bar{u}(x + n\Delta x)$ and $u(x - \frac{\Delta x}{2}) = \sum_{n=-2}^{2} c_n \bar{u}(x + (n-1)\Delta x)$, and are computed as

$$\begin{aligned}
\nu &= \Delta x \textstyle\sum_{n=-2}^{2} c_n \frac{(n-1)^2 - n^2}{2}, && (3.15)\\
\delta &= \Delta x^2 \textstyle\sum_{n=-2}^{2} c_n \frac{(n-1)^3 - n^3}{6}, && (3.16)\\
\sigma &= -\Delta x^3 \textstyle\sum_{n=-2}^{2} c_n \frac{(n-1)^4 - n^4}{24}. && (3.17)
\end{aligned}$$

Figure 3.5 shows these quantities for WENO5-JS. In order to estimate the contribution of each term, we approximated the higher-order spatial derivatives using standard finite-volume methods, and scale each by the magnitude of that derivative. For example, the influence of artificial viscosity is computed as

$$I_\nu(x) = \frac{\nu(x + \Delta x/2) + \nu(x - \Delta x/2)}{2} \left| \frac{u'(x + \Delta x/2) - u'(x - \Delta x/2)}{\Delta x} \right|. \qquad (3.18)$$

Hence, we ignore regions of the flow where the coefficient may signify that artificial viscosity (or other properties) are being added when they would have a negligible effect because the derivative is small.

One can see that for WENO-JS there is no viscosity or dispersion, as the method is designed such that on each substencil $\sum_{n=-2}^{2} c_n \frac{(n-1)^2 - n^2}{2} = 0$ and $\sum_{n=-2}^{2} c_n \frac{(n-1)^3 - n^3}{6} =$

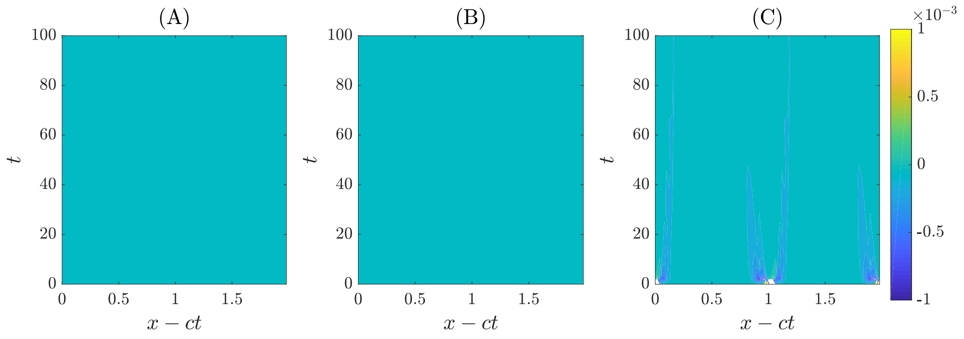


Figure 3.5: Influence of (A) artificial viscosity, (B) dispersion, and (C) hyperviscosity of WENO5-JS

0, so WENO5-JS applies only hyperviscosity. The method applies a small amount of negative hyperviscosity near the discontinuity. As time goes on and the discontinuity continues to diffuse, the influence of hyperviscosity decreases.

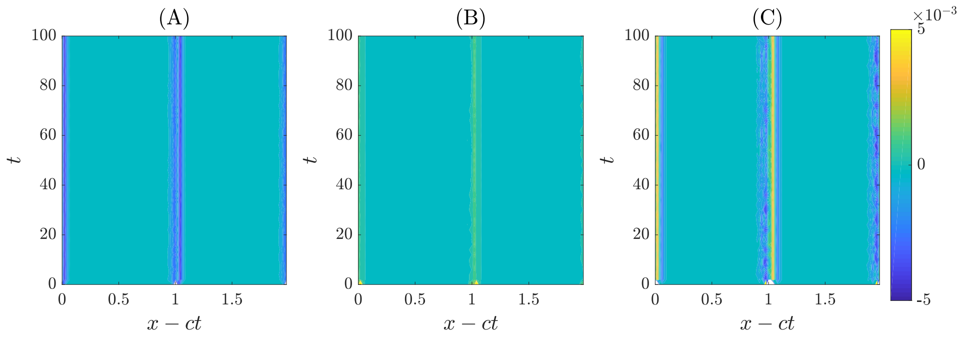


Figure 3.6: Influence of (A) artificial viscosity, (B) dispersion, and (C) hyperviscosity of WENO-NN

Figure 3.6 shows that unlike WENO5-JS, WENO-NN adds both artificial viscosity and dispersion to the solution. We see that near the discontinuity, negative viscosity is being added, which apparently provides the anti-diffusion that causes the discontinuity to retain its steepness, while hyperviscosity is applied to stabilize the solution. By analyzing physically how WENO-NN works, one could potentially learn how to develop an improved scheme that does not rely on a neural network, though this task is beyond the scope of this thesis.

We obtain a quantitative picture of the error in figure 3.7. We plot the L_2 error over time (measured to the exact solution), as well as the total variation, $TV = \sum_{i=1}^{N} |u(\Delta x i) - u(\Delta x(i-1))|$, to indicate when oscillations have been induced in the solution. We also measured the width over which the discontinuity is spread by counting the cells that have an error above a certain threshold (in this case chosen to be 0.01) and multiplying this number by $\Delta x/2$ since there are two discontinuities in the simulation.

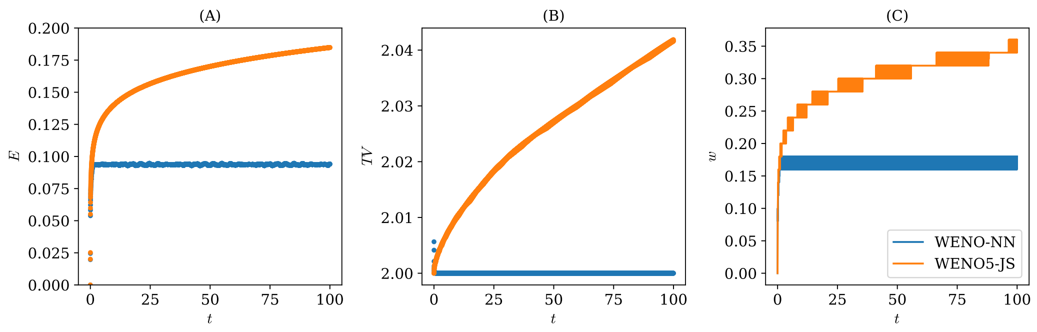


Figure 3.7: Comparing (A) L_2 error, (B) total variation, and (C) discontinuity width over time for WENO-NN and WENO-JS

The figure shows that WENO-NN decreases the error by almost a factor of 2 [1]. Although the total variation spikes at the start of the WENO-NN simulation, it is damped out and returns back to approximately the true value of 2, while the WENO5-JS total variation steadily climbs to above 2.04. We see a similar behavior in the discontinuity width, where WENO-NN reaches its steady value relatively quickly, while WENO5-JS continues to spread.

In order to determine how WENO-NN performs in different settings, the spatial and temporal discretizations were varied, and the L_2 error at the end of the simulation was measured. We again use a domain of length 2 and simulate for 50 periods. These results can be seen in figure 3.8.

We can see that WENO-NN tends to outperform WENO5-JS in regions where the spatial discretization is fine, but results in a larger L_2 error for coarse discretizations. To further compare the methods, figure 3.9 shows the error against the runtime for the two methods within a range of CFL values. We will only look at moderate CFL numbers, between 0.25 and 0.75, as stability becomes a concern for both methods above this range, and it becomes inefficient to run the simulation with CFL numbers below this range. We will also restrict the cell width to be below 0.025, as coarser meshes cause the final waveform to be unrecognizable compared to the exact

[1]Note that the error oscillates between two different values because in the exact solution the discontinuity switches between being on the edge of a cell and 1/3 of a cell width away from either the left or right of a cell edge since the CFL number is 2/3. To get a smooth curve, we apply a filter to the error and plot $E(i) = \frac{e(i)+e(i-1)+e(i-2)}{3}$

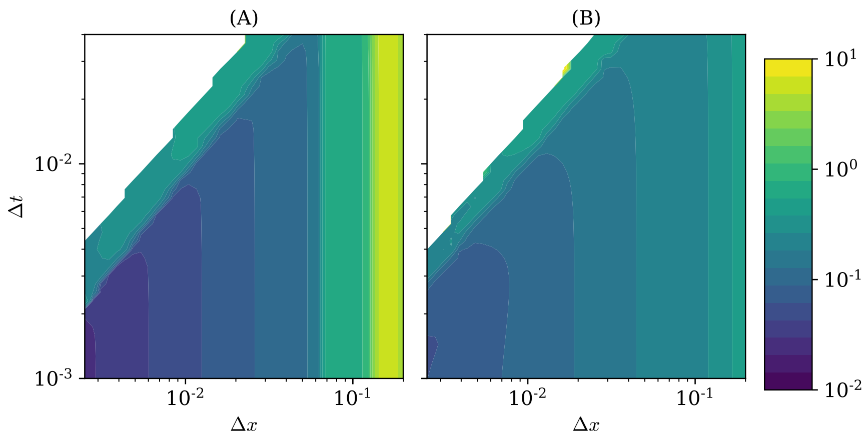


Figure 3.8: L_2 error at the end of the simulation for (A) WENO-NN and (B) WENO5-JS

solution for both methods, so the error comparison becomes meaningless. We see that when the CFL number is of a moderate value and the grid is sufficiently refined, WENO-NN typically achieves lower errors with smaller run time than WENO5-JS.

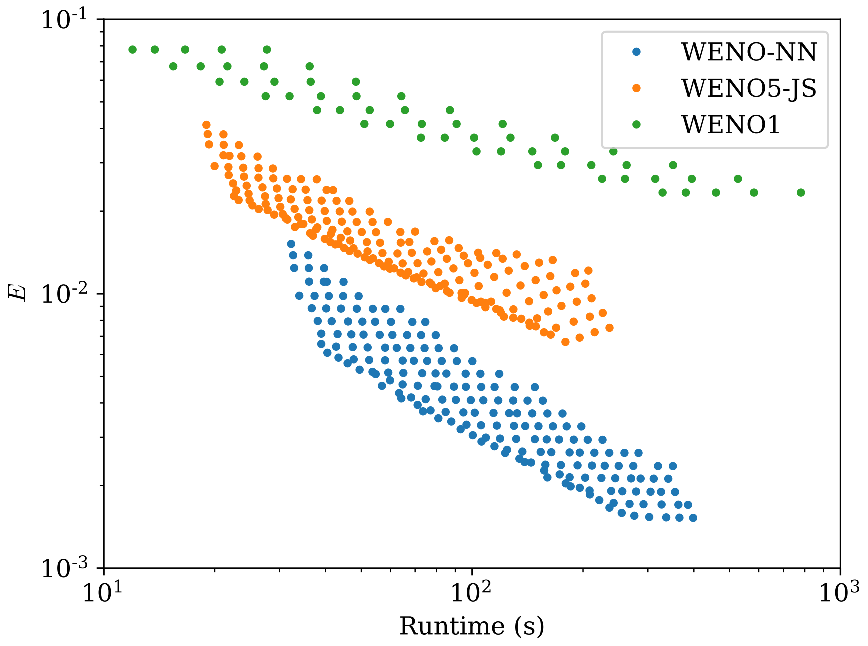


Figure 3.9: Comparing the L_2^2 error and runtime of WENO-NN, WENO5-JS and WENO1 for $0.25 < \text{CFL} < 0.75$ and $\Delta x < 0.025$

We also examine the convergence of each method for this problem in figure 3.10.

Here we fix CFL = 0.5 and measure the L_1 error of each numerical solution. We find that WENO1 achieves a slope of 0.5, WENO5-JS achieves a slope of 0.82, and WENO-NN achieves a slope of 1. Despite having a lower order of accuracy for smooth problems, WENO-NN is able to achieve a faster convergence rate for this discontinuous problem.

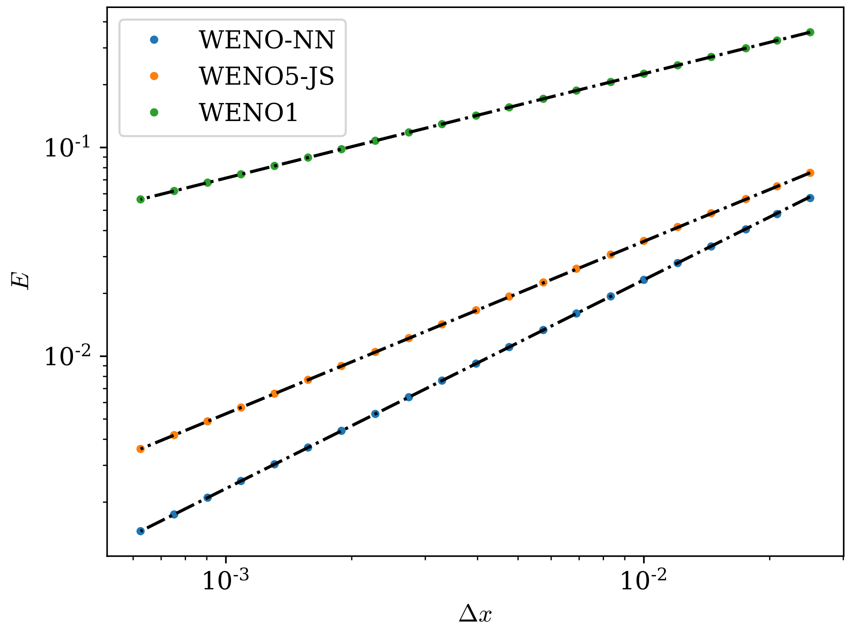


Figure 3.10: Comparing the convergence rates of WENO-NN, WENO5-JS, and WENO1 for advection of a step function

3.4.2 Inviscid Burgers' Equation

We next consider the inviscid Burgers' equation. Unlike the linear advection equation that included only contact (initial) discontinuities, the inviscid Burgers' equation results in shocks from smooth initial data. The distinction here is important: for a shock, the dynamics of the PDE will drive the solution towards a discontinuity, countering any diffusive effects associated with the numerics. We will again consider periodic boundary conditions, though we will start the simulation with a Gaussian as the initial condition. Hence, the IBVP is posed as

$$\frac{\partial u}{\partial t} + \frac{1}{2}\frac{\partial u^2}{\partial x} = 0, \tag{3.19}$$

$$u(0,x) = \exp(-k(x - \frac{L}{2})^2), \tag{3.20}$$

$$u(t,0) = u(t,L). \tag{3.21}$$

We simulate the problem for a time of $T = 4$ on a domain of length $L = 2$, and a value of $k = 20$. We first approximate the exact solution by solving this simulation with $\Delta x = 3.125 \cdot 10^{-4}$ and $\Delta t = 1.5625 \cdot 10^{-4}$ for a total of 6400 cells and 25601 timesteps. What we see is that the L_2 error is roughly the same for WENO5-JS and WENO-NN, as shown by Figure 3.11. Hence, we should expect the method to perform similarly to WENO5 in the presence of a shock.

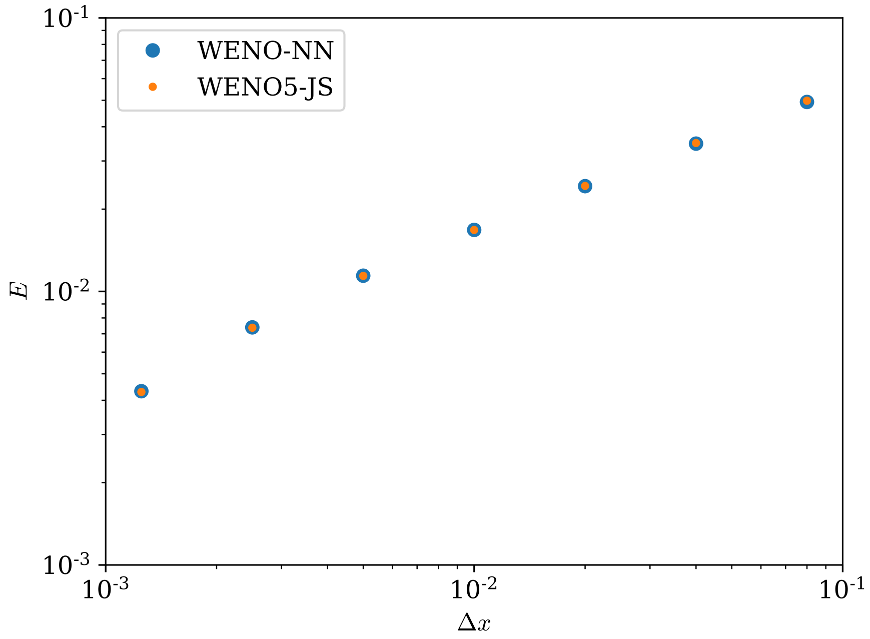


Figure 3.11: Comparing error vs. grid spacing of WENO-NN and WENO5-JS for the inviscid Burgers' equation

3.4.3 1-D Euler Equations

The last test case we will look at is the Shu-Osher problem, a test case involving the 1-D Euler equations. Note that the method was also tested on the Sod problem, but because this test case did not lead to any conclusions not drawn from either the advection equation or the inviscid Burgers' equation, these results have been omitted. The Shu-Osher problem is a model problem for turbulence-shockwave interactions. It involves the following equations and initial conditions

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u)}{\partial x} = 0, \tag{3.22}$$

$$\frac{\partial \rho u}{\partial t} + \frac{\partial (P + \rho u^2)}{\partial x} = 0, \tag{3.23}$$

$$\frac{\partial E}{\partial t} + \frac{\partial ((E + P)u)}{\partial x} = 0, \tag{3.24}$$

$$P = (\gamma - 1)(E - \frac{1}{2}\rho u^2), \tag{3.25}$$

$$\rho(0, x) = \begin{cases} 3.857143, & \text{if } x \leq 1 \\ 1 + \epsilon \sin(5x), & \text{otherwise} \end{cases}, \tag{3.26}$$

$$u(0, x) = \begin{cases} 2.629369, & \text{if } x \leq 1 \\ 0, & \text{otherwise} \end{cases}, \tag{3.27}$$

$$P(0, x) = \begin{cases} 10.33333, & \text{if } x \leq 1 \\ 1, & \text{otherwise} \end{cases}. \tag{3.28}$$

The simulation takes place on a domain of length $L = 10$ and is run until a final time of $T = 2$. ϵ is set to 0.2. We first obtain an approximately exact solution by discretizing the solution into 12800 cells and 10240 time-steps and use WENO5-JS for the simulation. This grid is fine enough to consider the solution exact. We then solve the problem using 300 cells and 240 time-steps using both WENO5-JS and WENO-NN, and compare the numerical results to the exact solution. Figure 3.12 shows the density, pressure, and velocity at the end of the simulation.

The most interesting aspect of the solution is the highly oscillatory section of the density profile, which is considered to behave similarly to turbulence. Figure 3.13 shows a zoomed in view of this section at different grid resolutions.

One can see that the neural network diffuses the oscillations significantly less than WENO5 for coarse grids, which is an encouraging result in terms of simulating actual turbulence. As the mesh is further refined, the WENO-NN appears to add too much anti-diffusion, which inflates fine features of the solution. On the very fine grid, both WENO5-JS and WENO-NN are similar (provided WENO-NN is stable, then it is constrained to converge as at least first-order).

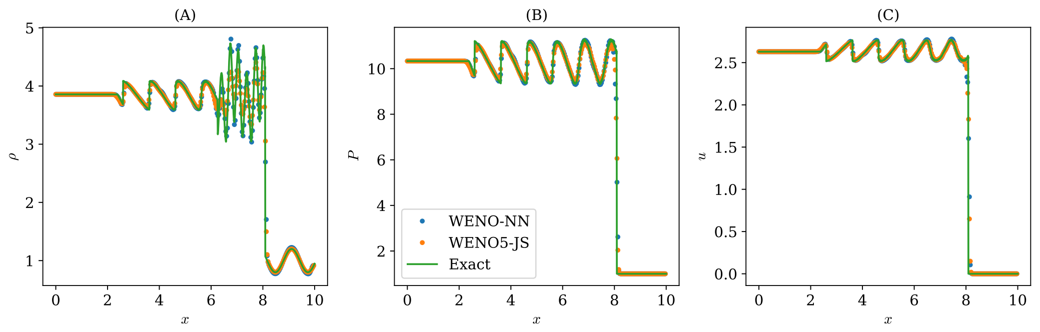


Figure 3.12: Comparing (A) density, (B) pressure, and (C) velocity of WENO-NN and WENO5-JS to the exact solution for the Shu-Osher problem

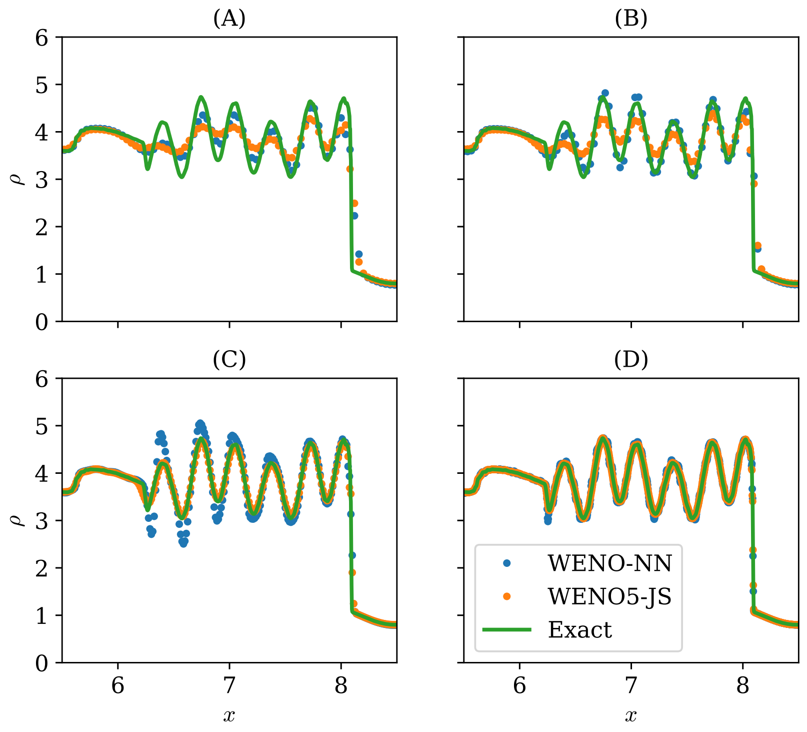


Figure 3.13: Zoomed in view of the turbulent section for different grid resolutions of (A) 250, (B) 300, (C) 800, and (D) 3200 cells for the Shu-Osher problem

3.5 Discussion and Conclusions

By training a neural network to process the outputs of the WENO5-JS algorithm, we were able to improve its accuracy, particularly in problems where the artificial diffusion introduced in WENO5-JS was excessive. While WENO-NN is more expensive per evaluation than WENO5-JS, it achieved lower errors on coarser grids, which indicates some potential to be useful more generally. We trace these performance improvements to increased flexibility in the neural network compared to WENO5-JS, as it can add artificial viscosity and dispersion while WENO5-JS coefficients are constrained to make these quantities zero. By analyzing the advection of a step function, we found that WENO-NN applies negative artificial viscosity near the discontinuity, which allows it to maintain its sharp profile (this takes place sometime into the simulation after the initial profile has been slightly smoothed due to artificial viscosity that prevents spurious oscillations). We then observe similar behavior in the Shu-Osher problem, where we see that WENO5-JS diffuses the fine features of the solution more than WENO-NN. However, we also found that at certain resolutions WENO-NN applies too much negative artificial viscosity, resulting in too much amplification of these fine scale features, though this amplification does not develop into an instability. For true shocks, as opposed to contact discontinuities, we found that our method performs very similarly to WENO5-JS.

One drawback of WENO-NN is that it does not inherit the high-order convergence of WENO5-JS. It would be an improvement to the method to be able to structure the network such that its coefficients more quickly converge to those of either WENO5-JS or the constant coefficient scheme that maximizes order of accuracy in the presence of smooth solutions. However, this must be done in a way that does not interfere with predictions in non-smooth regimes that benefit from low-order behavior, which is a non-trivial task. Until such a method is developed, one would need to use WENO-NN as part of a hybrid scheme if higher order convergence is desired in smooth regions (Li and Qiu, 2010). Another outstanding issue with machine-learned schemes is stability. The WENO-NN scheme used here seemed to inherit the stability of the underlying WENO5-JS scheme that it was based on, but this needn't have been the case, and we cannot offer proof or an estimate for the maximal CFL.

In future work, we aim to test the method on large-scale, multidimensional problems. We would expect the benefits seen in 1-D problems to be more significant when multiple spatial dimensions are present, as WENO-NN allows for a coarser mesh, so the improvement scales exponentially with the number of dimensions.

Chapter 4

FINITENET: A FULLY CONVOLUTIONAL LSTM NETWORK ARCHITECTURE FOR TIME-DEPENDENT PARTIAL DIFFERENTIAL EQUATIONS

4.1 Introduction

We try to simultaneously exploit the spatial and temporal structure of PDEs through the convolutional LSTM architecture of our neural network, while maintaining the computational structures used in FVM/FDM. While some papers use convolutional LSTMs or related architectures to predict future values of time-series that also possess the property of spatial locality (Mohan et al., 2019; Xingjian et al., 2015), none exploit the FVM/FDM structure that naturally discretizate PDEs. Other papers have developed coarse-graining models by learning from data and have embedded these into CNNs (Bar-Sinai et al., 2019), but none simultaneously used an LSTM structure to also exploit the temporal structure of PDEs. We also present a novel training approach of directly minimizing simulation error over long time-horizons by building upon ideas developed for PDE-Net (Long, Lu, Ma, et al., 2017). Our approach aims to push the field of data-driven scientific computing forward by combining these ideas in a sufficiently generic and efficient framework to permit extensions to many problems.

4.2 Methodologies

4.2.1 Network Architecture

We structure our network to utilize well-known methods from numerical PDEs described earlier (SSPRK, FDM, and FVM). The network architecture mimics the structure of a grid used to numerically solve a PDE, where the size of the filter of the convolutional layer corresponds to the stencil that selects which information is used to compute the derivative, and each output of the network corresponds to the solution of the PDE at time t_j and location x_i. This can be seen in Figure 4.1.

When looking at a specific realization of the stencil in the x-domain, we arrive at an LSTM network. The purpose of the LSTM is to transfer information about the solution over long time-horizons, which adds a feature to our method that is not present in traditional methods. The hidden information is transferred from substep to substep (t_{j-1} to $t_{j-2/3}$ to $t_{j-1/3}$, and then to the next step at the end of the current step $t_{j-1/3}$ to t_j).

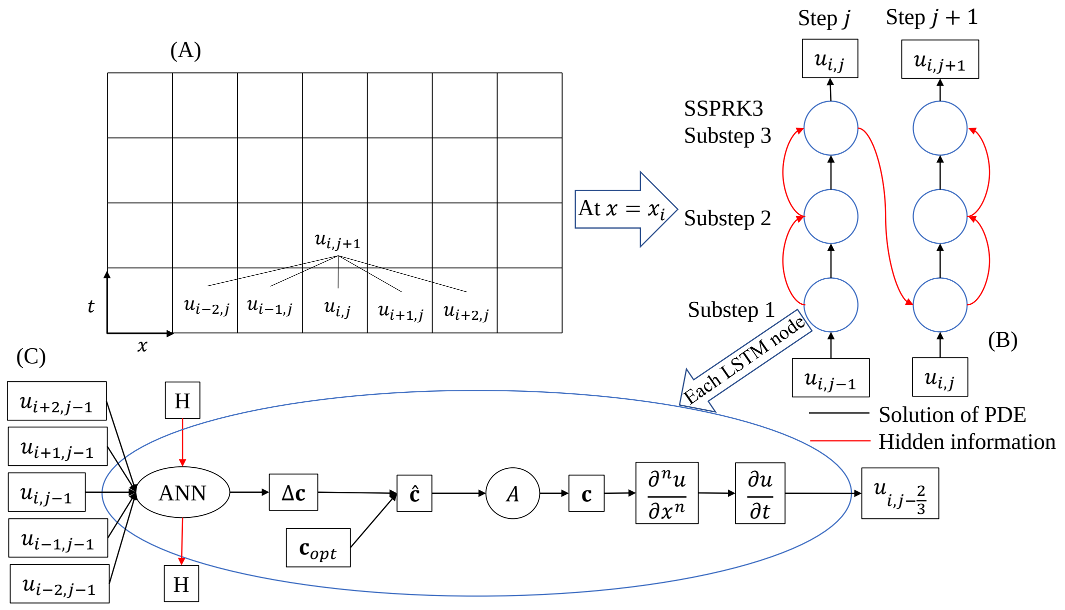


Figure 4.1: Network Architecture at (A) the top level, (B) LSTM at a specific x location, (C) each evaluation of the LSTM

Within each evaluation of the LSTM, the information is used to compute the solution at the next substep in a way that mimics traditional FDM or FVM. The solution values at the previous timestep $u_{i-2:i+2,j-1}$ and the hidden information with dimension 32 from the previous timestep are input to a neural network with 3 layers and 32 neurons per layer. Note that including the hidden information increases memory requirements, so the trade off between a more expressive model and using more gridpoint must be considered for larger problems. This network outputs the hidden information to the next substep, as well as a prediction of the FVM or FDM coefficients. The neural network does not directly predict these coefficients. Instead, it first computes a perturbation $\Delta\mathbf{c}$ to the coefficients $\mathbf{c}_{opt}$, with l_2 regularization applied to $\Delta\mathbf{c}$, similarly to a ResNet (He et al., 2016). c_opt is chosen based on the problem, and should be a baseline method such as a constant coefficient scheme that maximizes orders of accuracy or a shock capturing method such as WENO5-JS. We find that adding this step speeds up training and improves the performance of the model, as we are effectively biasing our estimate of the coefficients towards the coefficients that are optimal in the absence of prior knowledge about the solution. If the model is not confident about how perturbing $\mathbf{c}_{opt}$ will affect the accuracy of the derivative it can output values close to zero to default back to $\mathbf{c}_{opt}$. Once $\mathbf{c}_{opt}$ has been perturbed by $\Delta\mathbf{c}$ to obtain $\hat{\mathbf{c}}$, an affine transformation is performed on $\hat{\mathbf{c}}$

to guarantee that the coefficients are the desired order of accuracy (Bar-Sinai et al., 2019). This is one of the main benefits of using an FDM structure for our model, as we can prove that our model gives a solution that converges to the true solution at a known rate as the grid is refined. The details of this transformation can be seen in section 4.3.

After the affine transformation, the final coefficients **c** are known. The spatial derivatives are computed by taking the inner product of the coefficients $c_{i-2:i+2}$ with $u_{i-2:i+2,j-1}$. For equations with multiple spatial derivatives, a different set of coefficients is computed for each. These spatial derivatives are then used to compute the time derivative, which can finally be used to compute the solution at the next substep. This process is then repeated for the desired number of timesteps. Our network architecture allows us to train on exact solutions or data of the PDE end-to-end.

4.2.2 Training Algorithm

We train our network in a way that exactly mimics how it would be used to solve a PDE. More specifically, we start with some random initial condition and use the network to step the solution forward in time, and compare the result to the exact solution. For the linear advection equation, the analytical solution is known for arbitrary initial conditions. For the inviscid Burgers' equation and KS equation, the same simulation is also carried out on a fine grid using a baseline numerical method, which results in a solution that can be considered approximately exact. The inviscid Burgers' equation is solved using WENO5 FVM (Jiang and Shu, 1996), and the KS equation is solved using fourth-order FDM. The loss is computed by downsampling the exact solution onto the grid of the neural network, averaging over the square error at every point in time and space. We found this training strategy to be far superior to training on only 1 time-step at a time, as our approach is capable of minimizing long-term error accumulation and training the network to be numerically stable. In terms of computational complexity, our method is identical to backpropogation through time (BPTT) (Werbos, 1990).

Although this idea is not demonstrated in this thesis, one could also potentially train FiniteNet to minimize loss functions other than the L_2 error. In coarse graining simulations, getting the exact solution value at a specific point is typically not the main goal. For example, when using an LES model for turbulence, one would not be interested in what the pressure is at a specific point in time and space. Instead,

Algorithm 2 Train FiniteNet

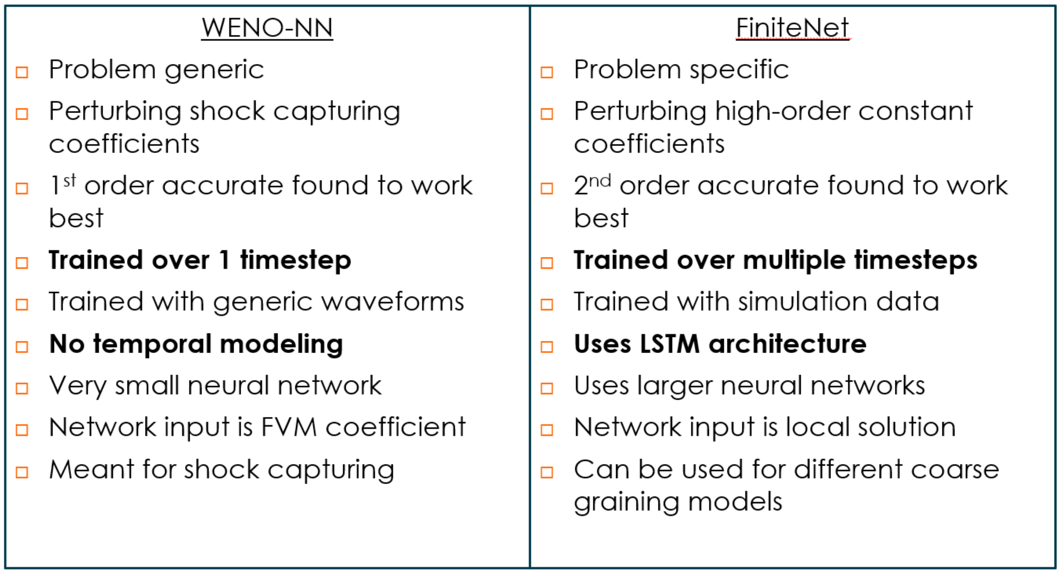
```
Select number of epochs $n_e$
Select minibatch size $n_m$
Select time-horizon $T$
for $i = 1$ to $n_e$ do
    Set total MSE to 0
    for $j = 1$ to $n_m$ do
        Select initial condition $u_j(x, 0)$
        Determine exact solution $u_j^*$
        Initialize hidden information $H(0)$ to 0
        for $k = 1$ to $T$ do
            Compute $u_j(x, t_k)$ and $H(t_k)$ with FiniteNet
        Add MSE between $u_j$ and $u_j^*$ to total MSE
    Compute gradient of simulation error w.r.t. FiniteNet parameters
    Update FiniteNet parameters with ADAM optimizer
```

one would be trying to accurately capture large scale properties of the flow. If one were to use FiniteNet to design airfoils, then the quantities of interest may be lift and drag. So perhaps one would want to design their loss function to be $L^2 + D^2$, where L represents lift and D represents drag.

FiniteNet has many similarities with the WENO-NN algorithm presented in Chapter 3. However, many of the details relating to the model and training process have been altered. A list summarizing these differences can be seen in Figure 4.2. In order to make the architecture more general, we now perturb coefficients from a high-order, constant scheme rather than a shock capturing method, though it would still be possible to do so. Because we are not necessarily using a scheme with varying coefficients, FVM coefficients are no longer a suitable input to the neural network, so we instead use the local solution values.

4.2.3 Accuracy Constraints

FiniteNet is structured such that the numerical method is guaranteed to satisfy a minimum order of accuracy n, $e = o(\Delta x^n)$ (Bar-Sinai et al., 2019). One can perform a Taylor series expansion on the approximations of the form of 2.26 to obtain linear constraint equations that the coefficients must satisfy for the method to achieve a desired order of accuracy. We take the coefficients that the neural network outputs, $\hat{\mathbf{c}}$, and find the minimal perturbation $\Delta\mathbf{c}$ that causes them to satisfy the constraint equations, which leads to the optimization problem



WENO-NN	FiniteNet
□ Problem generic	□ Problem specific
□ Perturbing shock capturing coefficients	□ Perturbing high-order constant coefficients
□ 1st order accurate found to work best	□ 2nd order accurate found to work best
□ **Trained over 1 timestep**	□ **Trained over multiple timesteps**
□ Trained with generic waveforms	□ Trained with simulation data
□ **No temporal modeling**	□ **Uses LSTM architecture**
□ Very small neural network	□ Uses larger neural networks
□ Network input is FVM coefficient	□ Network input is local solution
□ Meant for shock capturing	□ Can be used for different coarse graining models

Figure 4.2: List of the differences between WENO-NN and FiniteNet. Key differences have been bolded.

$$\begin{aligned} \min_{\Delta \mathbf{c} \in R^5} \quad & \textstyle\sum_{n=-2}^{2} (\Delta c_n)^2 \\ \text{s.t.} \quad & A(\hat{\mathbf{c}} + \Delta \mathbf{c}) = \mathbf{b}, \end{aligned} \tag{4.1}$$

which has analytical solution $\Delta \mathbf{c} = A^T(AA^T)^{-1}(\mathbf{b} - A\hat{\mathbf{c}})$, which can be expressed as an affine transformation on $\hat{\mathbf{c}}$, and can therefore be added to the network as a layer with no activation function and untrainable weights.

4.3 Simulation Results

4.3.1 Summary

We find that our method is capable of reducing the error relative to the baseline method for all three equations tested. These results show promise for generalization to other equations, as each PDE we examined has qualitatively different behavior. A table summarizing our results can be seen in Table 4.2. The variation is due to averaging results from randomly generated initial conditions. The error ratio e_r is computed as $e_r = \frac{e_F}{e_B}$ where e_F is the FiniteNet MSE and e_B is the MSE of the baseline method. We analyze $\log_{10} e_r$ because $\lim_{e_F \to 0} e_r \to 0$ and $\lim_{e_B \to 0} e_r \to \infty$. Hence, one case of the baseline method outperforming FiniteNet could skew the average and standard deviation of e_r significantly while the opposite would have very little effect, resulting in a bias. Additionally, the data empirically follows a log-normal distribution more closely than a normal distribution.

Table 4.1: Mean and standard deviation of $\log_{10} e_r$ for each PDE

Equation	$\log_{10} e_r$	Better?
Linear Advection	-0.27± 0.08	√
Inviscid Burgers'	-0.53± 0.08	√
Kuramoto-Sivashinsky	-0.62± 0.41	√

Hyperparameters were tuned to optimize performance on the linear advection equation. These same hyperparameters were then used as starting points for the other equations and tuned as necessary. For example, the time-horizon used in training was changed from 100 to 200 for the KS equation, as this helped FiniteNet track chaotic solutions over longer time-horizons. Additionally, it was found that increasing the l_2 regularization constant from $\lambda = 0.001$ to 0.1 for the KS equations improved performance. The inviscid Burgers' equation was trained for 500 epochs instead of 400 because the error was still decreasing,

All data generation, training, and testing was carried out on a desktop computer with 32 GB of RAM and a single CPU with 3 GHz processing speed. Hence, by scaling the computing resources one could scale our method to more challenging problems. Our code and trained models can be found here: `https://github.com/FiniteNetICML2020Code/FiniteNet`, which includes links to our data.

We also find that FiniteNet trains methods that are stable, which tends to be a challenge in the field of learned FDM. Although we cannot formally prove stability, we observe in Figure 4.3 that if the randomly initialized network weights lead to an unstable scheme, the method will quickly learn to become stable by minimizing accumulated error.

4.3.2 Linear Advection Equation

For the linear advection equation, we generate random discontinuous initial conditions, and compare the errors obtained using FiniteNet to errors obtained using WENO5. Each epoch involved generating 5 new initial conditions, computing the error against the exact solution, and updating the weights with the Adam optimizer (Kingma and Ba, 2014). We trained for 400 epochs.

After training was complete, we tested the model on 1000 more random initial conditions and computed the error ratio. We saw that FiniteNet outperformed WENO5 in 999 of the 1000 simulations. A probability mass function (PMF) of the

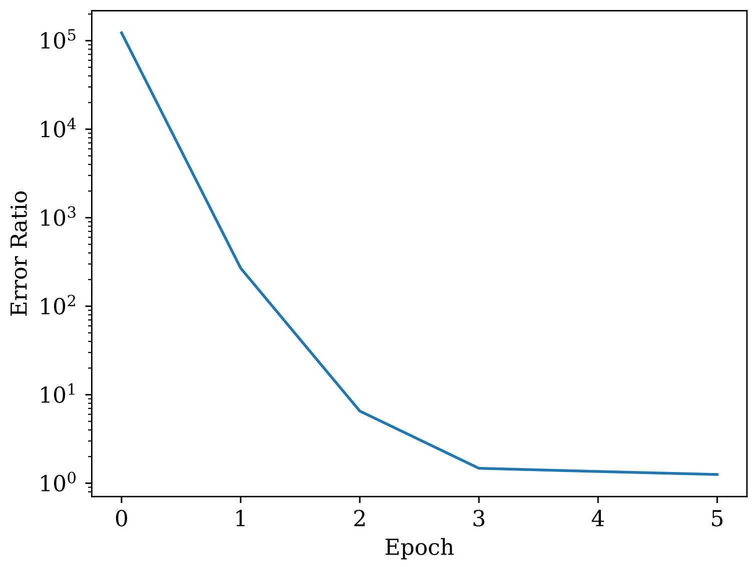


Figure 4.3: Training FiniteNet from unstable initial condition

error ratio can be seen in Figure 4.4.

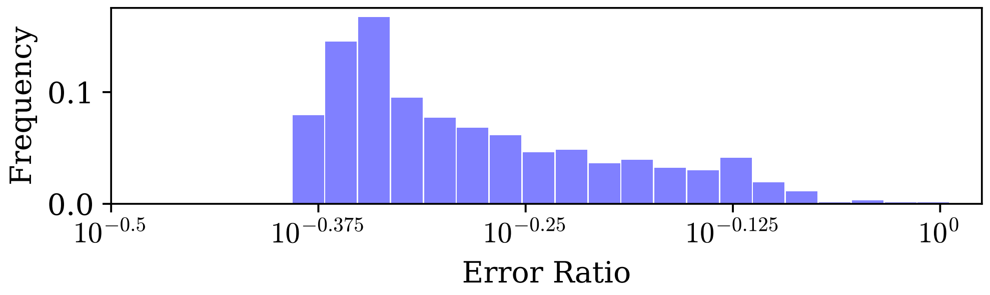


Figure 4.4: PMF of linear advection testing

We also plot a WENO5 solution and FiniteNet solution in Figure 4.5 to gain insight into how FiniteNet improves the solution. We see that FiniteNet more sharply resolves discontinuities at the cost of adding oscillations, which leads to a net reduction in error.

We can use this result to hypothesize that FiniteNet will further improve the solution when the discontinuity is larger. We verify our hypothesis by plotting error ratio against discontinuity width in Figure 4.6, and verifying that larger discontinuities lead to lower error ratios.

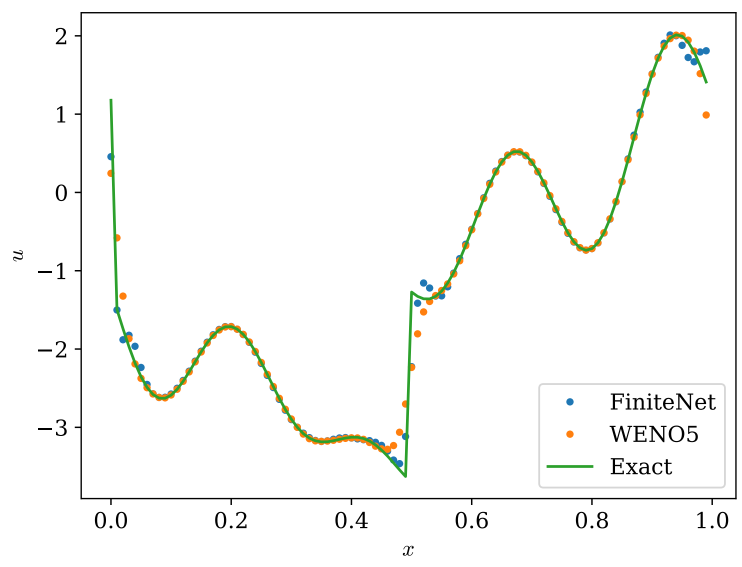


Figure 4.5: Solutions of linear advection equation

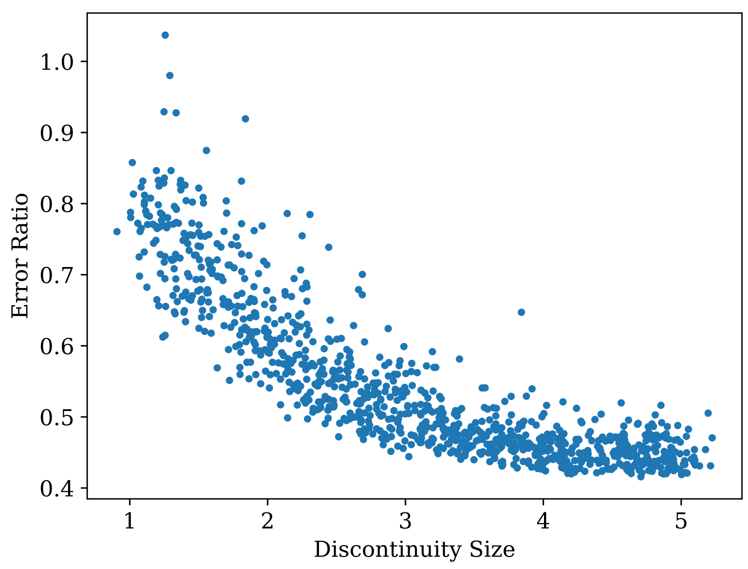


Figure 4.6: Discontinuity size vs error ratio for advection equation

4.3.3 Inviscid Burgers' Equation

We train the neural network to interpolate flux values for the inviscid Burgers' equation. Once again, each epoch involves generating five random initial conditions. In lieu of an exact solution, we use WENO5 on a 4x refined mesh to approximate the exact solution. We then ran 1000 simulations with the trained network and compared the results to WENO5. FiniteNet achieved a lower error on 998 of the 1000 test cases. A PMF showing the error ratio for these 1000 test cases can be seen in Figure 4.7.

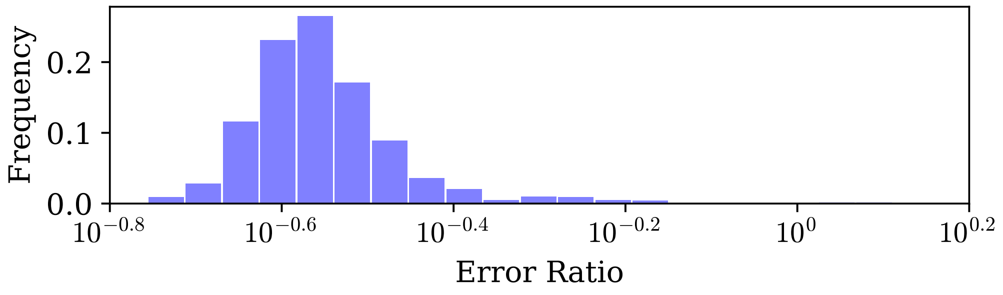


Figure 4.7: PMF of inviscid Burgers' testing

By examining the error induced when numerically solving this equation with WENO5 and the FiniteNet in Figure 4.8, we see that WENO5 accumulates higher error around shocks. This tells us that FiniteNet achieves it's error reduction by more sharply resolving the shocks.

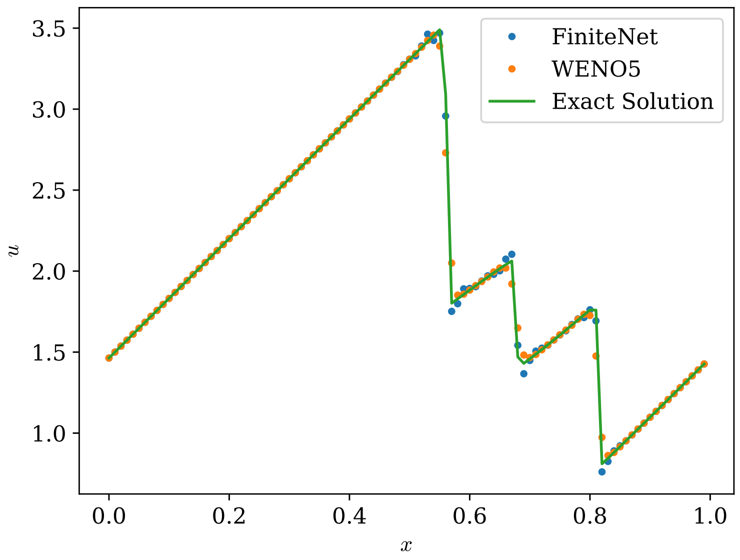


Figure 4.8: Solution of inviscid Burgers' equation

We can get a prediction of roughly how many and how large of shocks will develop in the solution by looking at the total variation of the initial condition. We plot the total variation of the initial condition and compare it to the error ratio in Figure 4.9.

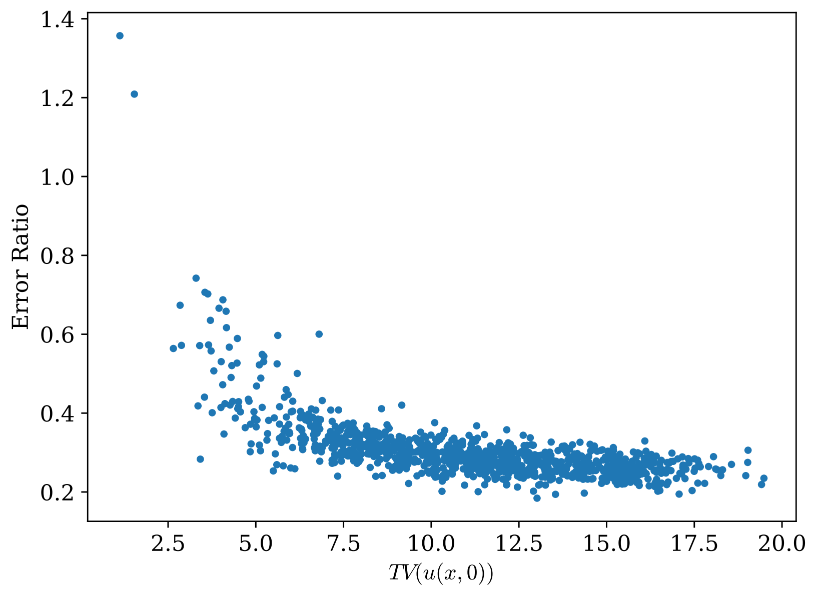


Figure 4.9: Trends between total variation and error ratio

This data shows that FiniteNet tends to do better relative to WENO5 when the total variation of the initial condition is higher, which helps confirm our result that FiniteNet offers the most improvement when many large shocks form.

4.3.4 Kuramoto-Sivashinsky Equation

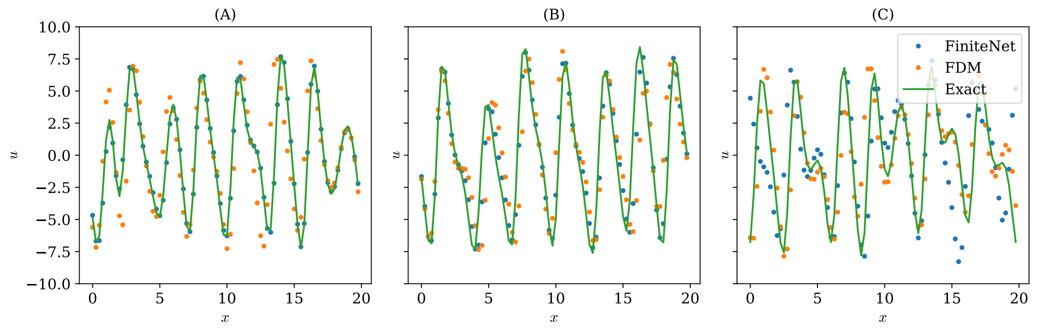


Figure 4.10: (A) Best case, (B) typical case, and (C) worst-case error ratio between FiniteNet and FDM for KS

The network is trained for the KS equation in the same way as was done for the inviscid Burgers' equation: by solving the equation on a 4x refined mesh to closely approximate the exact solution. We generate our random initial conditions for training by setting a random initial condition for the exact solution and simulating until the trajectory has reached the chaotic attractor, and start training from random sequences of the solution on the attractor so that we are learning a more consistent set of dynamics, as the initial transient tends to be less predictable.

After training, we use both FiniteNet and fourth-order FDM to solve the KS equations from 1000 new initial conditions. We find the FiniteNet achieves a lower error in 961 of the 1000 tests. Interestingly, we see that in some cases standard FDM is unable to track the chaotic evolution and results in a solution with no visual fit, while FiniteNet succeeds at tracking the chaotic trajectory over the time horizon tested. In the cases where FiniteNet leads to a larger error than FDM, we see that neither method could track the chaotic trajectory, so there is some small probability that FiniteNet may follow a worse trajectory than FDM. In order to determine the reliability of FiniteNet compared to FDM, we compute the statistics of the errors individually. FiniteNet has mean error 1.20 and standard deviation 1.89, while FDM has mean error 2.94 and standard deviation 3.01. So we see that FDM error has higher mean and variance, and can conclude that FiniteNet is more reliable.

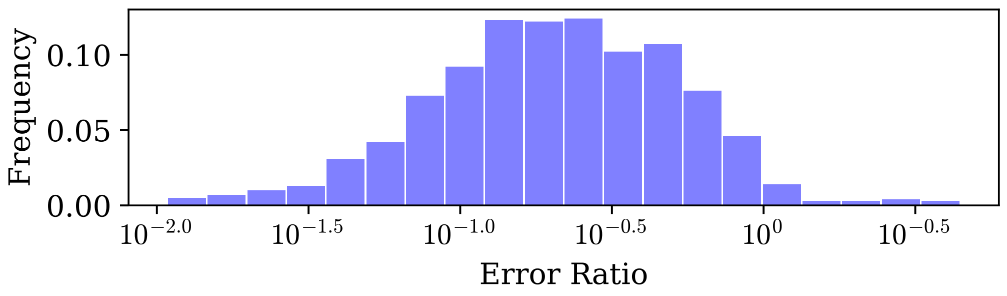


Figure 4.11: PMF of KS testing

4.3.5 Comparison with other temporal modeling techniques

We also compare FiniteNet to models that use techniques other than an LSTM to model temporal behavior. We look at two other simple temporal modeling approaches, including a basic RNN, and convolutions in time. The basic RNN is structurally similar to an LSTM, but simpler as it does not attempt to maintain long-term memory as the LSTM does. By convolutions in time, we simply refer to using information from multiple previous timesteps as inputs to the neural network. For this experiment, we use two additional previous timesteps. We also test a model

Table 4.2: Mean and standard deviation of $\log_{10} e_r$ for each model

Equation	$\log_{10} e_r$
LSTM	-0.27± 0.08
RNN	-0.18± 0.12
Convolutions in time	-0.22± 0.10
None	-0.17± 0.09

with no temporal modeling component as a baseline. We perform this experiment on the linear advection equation, with randomly generated initial conditions that were not seen during training. Additionally, we follow the same methodology for training the other temporal models as was used to train the LSTM model.

We see that the LSTM model achieves the largest error reduction out of the approaches tested, and that all models that included some sort of temporal modeling component outperformed the model that did not have any temporal modeling. Hence, we can conclude that attempting to model the temporal behavior of a PDE does improve the performance of the model.

4.4 Discussion

In this chapter, we have presented FiniteNet, a machine learning approach that can reduce the error when numerically solving a PDE. By combining the LSTM with well-understood and tested discretization schemes, we can significantly reduce the error for PDEs displaying a variety of behavior including chaotic and discontinuous solutions. The FiniteNet architecture mimics the structure of a numerical PDE solver, and builds the timestepping method, spatial discretization, and PDE into the network. We train FiniteNet by using it to simulate the PDE, and minimizing simulation error against a trusted solution. This training approach causes the resulting numerical scheme to be empirically stable, which has been a challenge for other approaches.

We compared numerical solutions obtained by FiniteNet to those of baseline methods. We saw that for the inviscid Burgers' equation and the linear advection equation, FiniteNet is more sharply resolving discontinuities at the cost of sometimes adding small oscillations. This result makes intuitive sense, as the global error is dominated by regions near discontinuities so FiniteNet reduces the error the most by improving performance in these areas. When examining the KS equations, we see that FiniteNet reduces the error by more accurately tracking the chaotic trajectory of the exact solution. The main challenge of chaotic systems is that small errors grow quickly over time, which can lead to solutions that have completely diverged from the exact solution. FiniteNet is able to significantly reduce error by preventing this from happening in many realizations of this PDE.

Further work could involve comparing the runtime vs. error of FiniteNet to baseline approaches to more directly characterize how much of an improvement FiniteNet can offer. Additionally, we have till now only examined problems in one spatial dimension with periodic boundary conditions. It will be interesting to test the method on a large-scale problem, such as a turbulent flow.

Chapter 5

APPLICATION OF TRANSFER LEARNING TO WENO SCHEMES

5.1 Introduction

In the third chapter of this thesis we presented the WENO-NN algorithm, which we showed is capable of giving more accurate results at the same level of computational cost as WENO5-JS by utilizing a small neural network trained on generic waveforms. We then showed in the fourth chapter that by training a larger neural network on simulation data, we can create equation specific methods that lead to a very significant error reduction. In this chapter we explore the intersection of these two ideas by applying transfer learning to WENO-NN integrated into a larger neural network, along with equation specific data.

5.2 WENO-TL Algorithm

The WENO-TL algorithm is similar to WENO-NN described in Chapter 3 of this thesis. Once trained, WENO-TL is essentially identical in structure to WENO-NN. The inputs to the algorithm are the cell averages $\bar{u}_{-2:2}$ selected by the FVM stencil, to which the WENO5-JS algorithm is then applied to get the WENO5-JS coefficients, $\tilde{c}_{-2:2}$. These coefficients are then inputs to a neural network with identical architecture as was used for WENO-NN, with the last two layers having been retrained for the problem of interest. This network again outputs perturbations to $\tilde{c}_{-2:2}$, as was done for WENO-NN. The main difference between WENO-TL and WENO-NN is that for some WENO-TL architectures tested, $\bar{u}_{-2:2}$ is also the input to an additional network added to make the model more flexible, whose output is an additional perturbation to $\tilde{c}_{-2:2}$. After these perturbations are applied and the perturbed coefficients $\hat{c}_{-2:2}$ are obtained, an affine transformation is applied to ensure that the method is consistent to obtain the final coefficients $c_{-1:2}$, and the interpolated value $u_{1/2}$ is finally obtained by taking the inner product of $c_{-2:2}$ and $\bar{u}_{-2:2}$ as $< \bar{u}_{-2:2}, c_{-2:2} >$. The algorithm is shown graphically in Figure 5.1.

The premise behind WENO-TL is that we use WENO-NN, whose training was fairly expensive, as a starting point for learning an equation specific numerical method via transfer learning. We then retrain a portion of the neural network used in WENO-NN, such as the last layer, on an equation specific dataset. We also

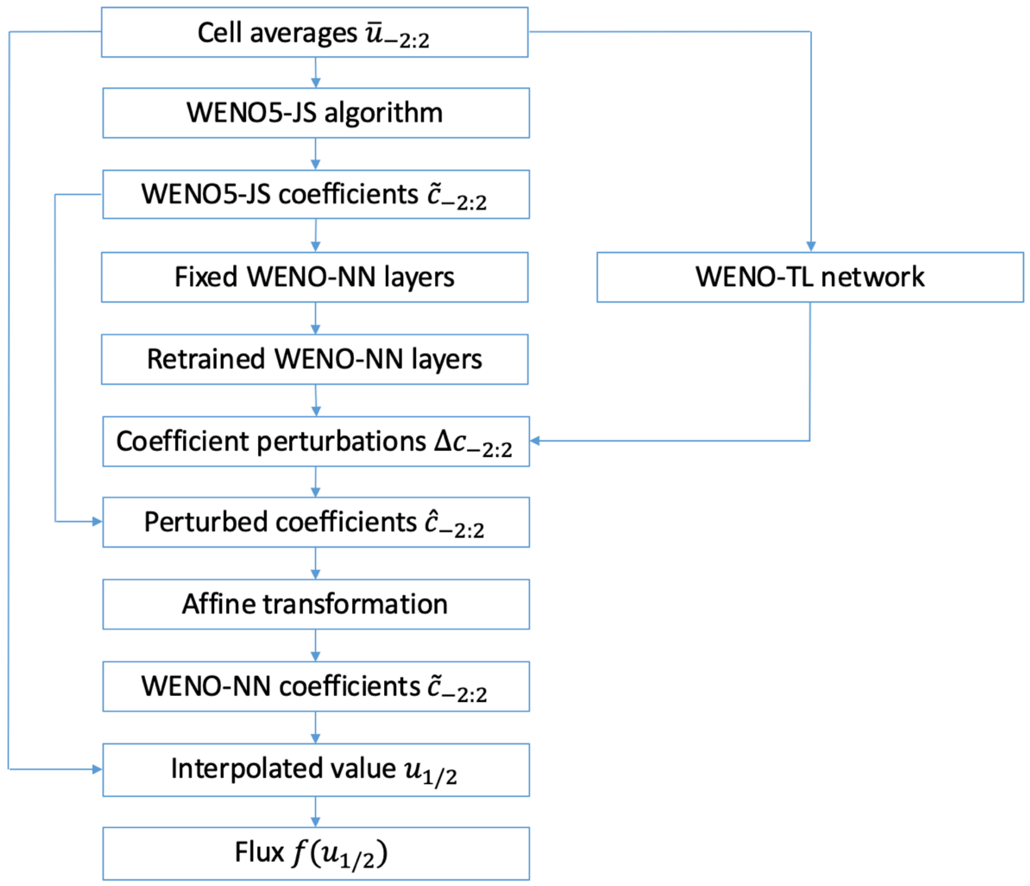

Cell averages $\bar{u}_{-2:2}$
WENO5-JS algorithm
WENO5-JS coefficients $\tilde{c}_{-2:2}$
Fixed WENO-NN layers
Retrained WENO-NN layers
Coefficient perturbations $\Delta c_{-2:2}$
Perturbed coefficients $\hat{c}_{-2:2}$
Affine transformation
WENO-NN coefficients $\tilde{c}_{-2:2}$
Interpolated value $u_{1/2}$
Flux $f(u_{1/2})$
WENO-TL network

Figure 5.1: Diagram of WENO-TL algorithm

experiment with adding additional components, such as linear regression models and additional layers to the network in order to make the model more flexible, which seems to be necessary to capture equation-specific patterns in a way that significantly reduces simulation error. We generate training data for WENO-TL by simulating the equation of interest with WENO5-JS on a fine grid.

5.2.1 Model

As mentioned above, we are building this model by applying transfer learning to WENO-NN. Hence, we use WENO-NN as a starting point for our model, and then can modify the network by retraining weights and/or adding more layers. We explore several different network architectures that use WENO-NN as a starting point.

First, we make no modifications to the WENO-NN architecture, and simply retrain the last layer of the neural network (WENO-TL4). We then add other network components that act in parallel with the WENO-NN network. This additional section of the network takes the cell average values directly as its inputs with no preprocessing, as was done for FiniteNet, and outputs an additional perturbation to the WENO5-JS coefficients. We construct several different models that follow this general architecture. The first simply applies linear regression from the cell averages to the perturbation (WENO-TL5). We also try adding one hidden layer with 32 neurons (WENO-TL6), and two hidden layers with 32 neurons each (WENO-TL7). Once again, the details of the model architecture can be seen in figure 5.1.

5.2.2 Training

We train WENO-TL using data specific to the equation of interest. This dataset can involve analytical solutions, high-fidelity simulation data, and experimental data. The training process is designed to be as similar to how the model will be used in practice as possible to maximize how well the model can generalize to new problems that it may be used for. Similar to FiniteNet described in Chapter 4, we generate a random initial condition, and solve the PDE on a very fine grid to get a solution with negligible error, or use an analytical solution if one exists. We then use WENO-TL to perform that same simulation on a coarse grid, compute the L_2 error of the WENO-TL solution, compute the gradient of that error with respect to the neural network parameters that are being retrained, update the parameters with the ADAM optimizer (Kingma and Ba, 2014), and repeat until training is complete. A flowchart showing the steps of the simulation that are carried out during the training process can be seen in the figure 5.2, which details the steps that take the initial data to the computation of the loss. Note that we once again apply regularization to the model by penalizing perturbations to WENO5-JS.

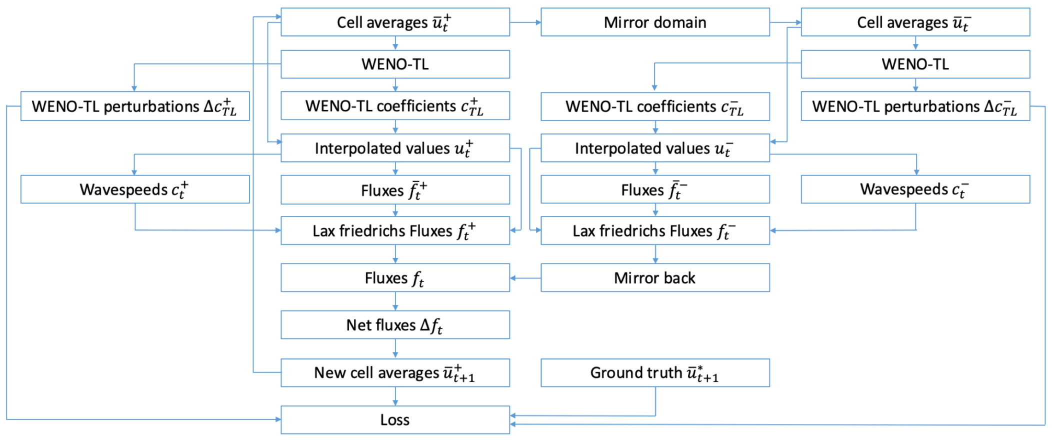

Cell averages $\bar{u}_t^+$
Mirror domain
Cell averages $\bar{u}_t^-$
WENO-TL
WENO-TL
WENO-TL perturbations Δc_{TL}^+
WENO-TL coefficients c_{TL}^+
WENO-TL coefficients c_{TL}^-
WENO-TL perturbations Δc_{TL}^-
Interpolated values u_t^+
Interpolated values u_t^-
Wavespeeds c_t^+
Fluxes $\bar{f}_t^+$
Fluxes $\bar{f}_t^-$
Wavespeeds c_t^-
Lax friedrichs Fluxes f_t^+
Lax friedrichs Fluxes f_t^-
Fluxes f_t
Mirror back
Net fluxes Δf_t
New cell averages $\bar{u}_{t+1}^+$
Ground truth $\bar{u}_{t+1}^*$
Loss

Figure 5.2: Diagram of simulation used in WENO-TL training procedure

The entire simulation is implemented in PyTorch so that the computational graph follows each step of the computation rather than only tracking steps that explicitly involve the neural network. By making the simulation end-to-end, we are able to build up the computational graph over multiple timesteps, which enhances the stability of the learned method because it account for how the numerical error evolves over longer time horizons. Hence, if an instability occurs, a very large error will be generated and the neural network will quickly learn to simulate the physical system in a stable way, resulting in a more robust training process.

5.3 Results

We perform several sets of experiments to assess the effectiveness of the WENO-TL algorithm in a variety of settings. We test the algorithm on the Euler equations in one and two dimensions. We examine the error reduction that each architecture is able to achieve on each equation, the convergence rates of each instance of the network, and the runtime required to achieve desired accuracy levels. We also study the ability of the network to generalize to higher dimensional versions of the same equation. For example, we train WENO-TL on solutions of the 1D Euler equations and demonstrate that it can then achieve performance improvements when solving the 2D Euler equations. Note that we do not train any of our models with 2D simulation data; all results shown for 2D problems were trained on data from 1D simulations.

5.3.1 Cases Examined

For the Euler equations, we consider two different families of initial conditions. The first will be referred to as the 'density bump' case. For this case, we have a randomly generated discontinuous initial density profile, zero initial velocity, and constant pressure. This is expressed mathematically as

$$\begin{aligned} \rho(0,x) &= \begin{cases} f_0(x), & \text{if } x \leq 1 \\ c + f_0(x), & \text{otherwise} \end{cases}, \\ u(0,x) &= 0 \\ P(0,x) &= 1, \end{aligned} \tag{5.1}$$

where $f_0(x)$ is a sum of sinusoids with random amplitude and phase, plus a constant to guarantee that $f_0(x) > 0 \ \forall x \in \mathbb{R}$ as is physically required. Additionally, c is a continuous random variable uniformly distributed between 1 and 6 to make the initial condition discontinuous. We choose to study this case because it has a known analytical solution for any random initial condition, where the solution should remain unchanged from the initial condition as the system is stepped forward in time, expressed mathematically as

$$\begin{aligned} \rho(t,x) &= \rho(0,x), \\ u(t,x) &= 0, \\ P(t,x) &= 1. \end{aligned} \tag{5.2}$$

However, simulating this problem with FVM introduces numerical errors. Hence, this case makes it easy to assess the accuracy of the method. The limitation of this case is that it only involves contact discontinuities rather than shockwaves or rarefactions that can be seen in other instances of the Euler equations. Because of the narrower scope of this problem, it is easier to tailor more accurate numerical methods to it.

We will also consider a case where both the density and pressure are randomly generated

$$\begin{aligned} \rho(0,x) &= \begin{cases} f_0(x), & \text{if } x \leq 1 \\ c_\rho + f_0(x), & \text{otherwise} \end{cases}, \\ u(0,x) &= 0 \\ P(0,x) &= \begin{cases} g_0(x), & \text{if } x \leq 1 \\ c_P + g_0(x), & \text{otherwise} \end{cases}. \end{aligned} \tag{5.3}$$

In this case, both $f_0(x)$ and $g_0(x)$ are sums of sinusoids with random amplitudes and phases, plus a constant to guarantee that $f_0(x), g_0(x) > 0 \; \forall x \in \mathbb{R}$. Also, both c_ρ and c_P are continuous random variables uniformly distributed between 1 and 6. This case generates much richer behaviors from the equations than the density bump test case, as not only are the solutions dynamic, but they are also capable of producing shockwaves and rarefactions in addition to contact discontinuities. Additionally, due to how general these initial conditions can be, the resulting numerical method should be applicable to essentially any initial conditions one may experience when solving the Euler equations.

5.3.2 Error reduction achieved by WENO-TL

We first attempt to quantify the accuracy improvements achieved by using WENO-TL. We do this by generating random initial conditions corresponding to the case that is being tested, and using each WENO-TL to step the IBVP forward in time. We generate these initial conditions from the same distributions used in training, but with parameters that were not seen during training such that we can see how well the new method generalizes to new problems.

5.3.2.1 Density bump for Euler equations

We first examine the density bump case for the 1D Euler equations. A summary of the statistics of the ratio of the error to WENO-JS, e_r for each network can be seen in table 5.1. The error ratio is computed by taking the L_2 error of each approximate solution compared to the exact solution and taking the ratio

$$e_r = \frac{||u_{TL} - u^*||_2}{||u_{JS} - u^*||_2}. \tag{5.4}$$

Table 5.1: e_r for each model for 1D density bump problems

Method	e_r	Better?
WENO-NN	0.98± 0.01	√
WENO-TL4	0.85± 0.03	√
WENO-TL5	0.94± 0.04	√
WENO-TL6	0.56± 0.07	√
WENO-TL7	0.49± 0.15	√

We see in this table that our starting point for the transfer learning, WENO-NN, barely manages to outperform WENO-JS. Then each model that transfer learning is applied to is able to get an even more accurate solution. We see a general trend that as model complexity increases, the error reduction and variance both increase as is expected in the bias-variance tradeoff. There is a notable exception in that WENO-TL4 actually reduces the error more significantly than WENO-TL5.

We also present PMFs for each model to get a more precise understanding of the distribution of these error ratios. This data can be seen in the plots below, and are generated by looking at 1000 newly generated initial conditions.

By analyzing these empirical PMFs, we can gain a deeper understanding of the ability of the learned numerical scheme to generalize to new problems. Each of

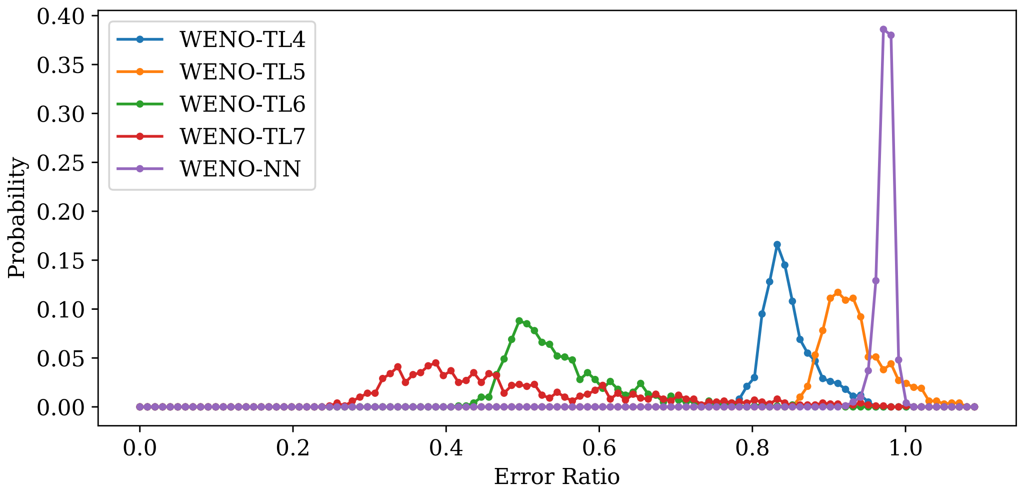


Figure 5.3: Probability mass function showing out of sample performance on 1D density bump problems for each model

these PMFs is constructed by using the model to simulate 1000 previously unseen initial conditions. We see that for WENO-TL4, WENO-TL6, and WENO-TL7, the hybrid model involving transfer learning always outperforms WENO5-JS, which indicates a promising ability to generalize to unseen problems as it seems to lead to an improvement in every case tested. However, we do not see this property for WENO-TL5, although it does tend to lead to an accuracy improvement for most initial conditions. We also see that each PMF is positively skewed, which indicates that relative to the most probable value of e_r, we expect to see a more modest accuracy improvement.

We also look at the performance of each model on the 2D Euler equations for a 2D analog of our density bump test case. This case extends the corresponding 1D case in an intuitive manner by applying the same process used to generate $f_0(x)$ to generate another function, $h_0(y)$ that instead varies in the y-direction, and another random discontinuity in the y-direction. These initial conditions are written as

$$\begin{aligned}
\rho(0,x,y) &= \begin{cases} f_0(x) + h_0(y), & \text{if } x \leq 1, y \leq 1 \\ c_x + f_0(x) + h_0(y), & \text{if } x > 1, y \leq 1 \\ c_y + f_0(x) + h_0(y), & \text{if } x \leq 1, y > 1 \\ c_x + c_y + f_0(x) + h_0(y), & \text{otherwise} \end{cases}, \qquad (5.5) \\
u_x(0,x,y) &= 0 \\
u_y(0,x,y) &= 0 \\
P(0,x,y) &= 1,
\end{aligned}$$

This test case also has the property that the exact solution is the same as the initial condition for all subsequent points in time, but numerical error occurs when stepping the solution forward in time with FVM and RK methods.

We again generate 1000 random initial conditions from this distribution, and use WENO-JS, WENO-NN, WENO-TL4, WENO-TL5, WENO-TL6, and WENO-TL7 to step the system of PDEs forward in time. The resulting statistics of e_r for each model can be seen in 5.2

Table 5.2: e_r for each model for 2D density bump problem

Method	e_r	Better?
WENO-NN	0.987 ± 0.003	√
WENO-TL4	0.85 ± 0.01	√
WENO-TL5	0.97 ± 0.02	√
WENO-TL6	0.60 ± 0.04	√
WENO-TL7	0.47 ± 0.09	√

These results are very similar to the error ratios seen in the 1D density bump case, and we even see better performance from WENO-TL7, in terms of both lower mean and variance. This is very promising for generalization, and a good indication that perhaps one could generate high fidelity training data at low computational cost using 1D simulations, and then deploy the trained model on larger 3D problems to get a large overall speedup and reduction in memory. We can once again look at the PMFs of the error ratios generated with each model, which are shown in Figure 5.4.

Overall these histograms look fairly similar to those generated from testing each model on the 1D density bump test case. WENO-TL5 occasionally does not outper-

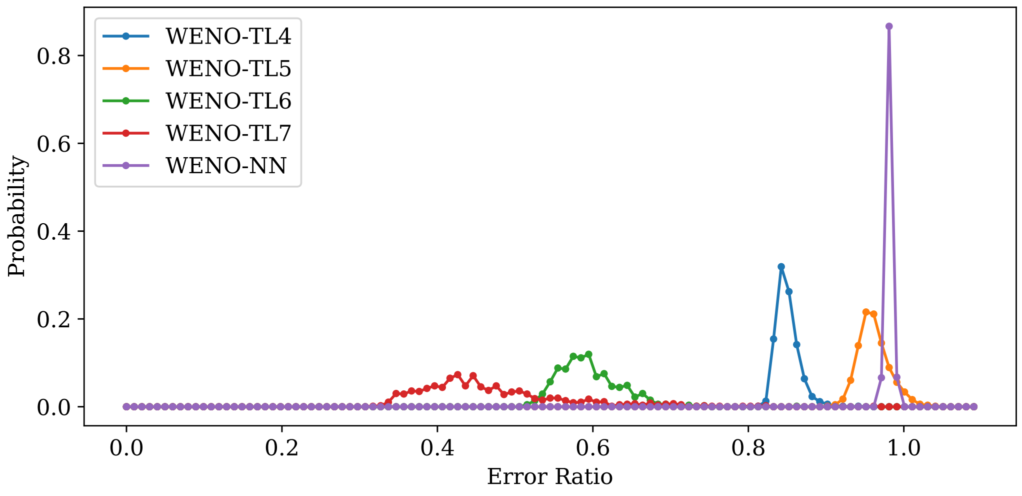


Figure 5.4: Probability mass function showing out of sample performance on 2D density bump problems for each model

form WENO-JS, but each of the other models do for each random initial condition tested.

5.3.2.2 Random initial conditions for Euler equations

We will now look at the case when the initial pressure and density are random and independently generated. Once again, we first show the statistics of e_r for each model architecture tested.

Table 5.3: e_r for each model for random initial conditions

Method	e_r	Better?
WENO-NN	0.93± 0.04	√
WENO-TL4	0.89± 0.07	√
WENO-TL5	0.86± 0.10	√
WENO-TL6	0.73± 0.13	√
WENO-TL7	0.68± 0.12	√

We once again see that each model tested is able to outperform WENO5-JS in terms of simulation accuracy measured in the L_2 norm, with WENO-NN giving a more significant error reduction than was seen in the density bump case, and a subsequent improvement on WENO-NN performance for each transfer learning model tested. We see that as model complexity increases, performance monotonically increases for this case. Variance also tends to increase, though WENO-TL6 has a slightly higher variance than WENO-TL7.

We also look at the PMFs of the error ratios for each model, once again by constructing 1000 previously unseen initial conditions to get a more detailed understanding of the out of sample performance of each model.

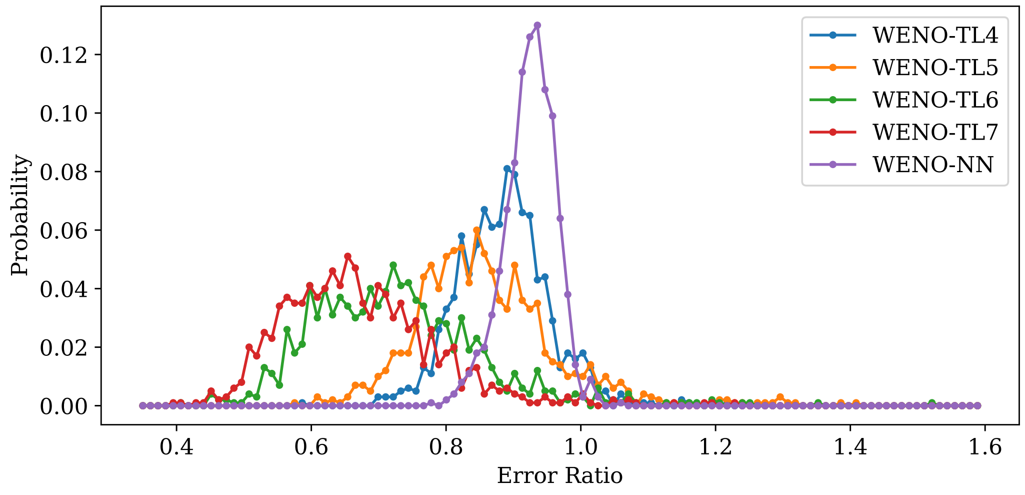


Figure 5.5: Probability mass function showing out of sample performance on 1D random initial condition problems for each model

Interestingly, we see that for the case where both the initial density and pressure are randomly generated, none of the hybrid models are always able to get a more accurate solution, though we do see that they are more accurate in the vast majority of cases.

5.3.3 Convergence Results

In this section, we look at the convergence rates of each model. This analysis helps us assess not just the rate at which the approximate solution approaches the exact solution as the grid is refined, but also the ability of each model to generalize beyond the resolution that the neural networks were trained for.

We first look at the 1D density bump case in Figure 5.6. We run the simulation on six different grid resolutions, logarithmically spaced between 0.002 and 0.05 with a constant CFL number of 0.1. We keep the initial condition the same for each resolution, and choose the initial density to be a square wave.

We see that WENO-JS and WENO-NN converge at a fairly constant rate, consistent

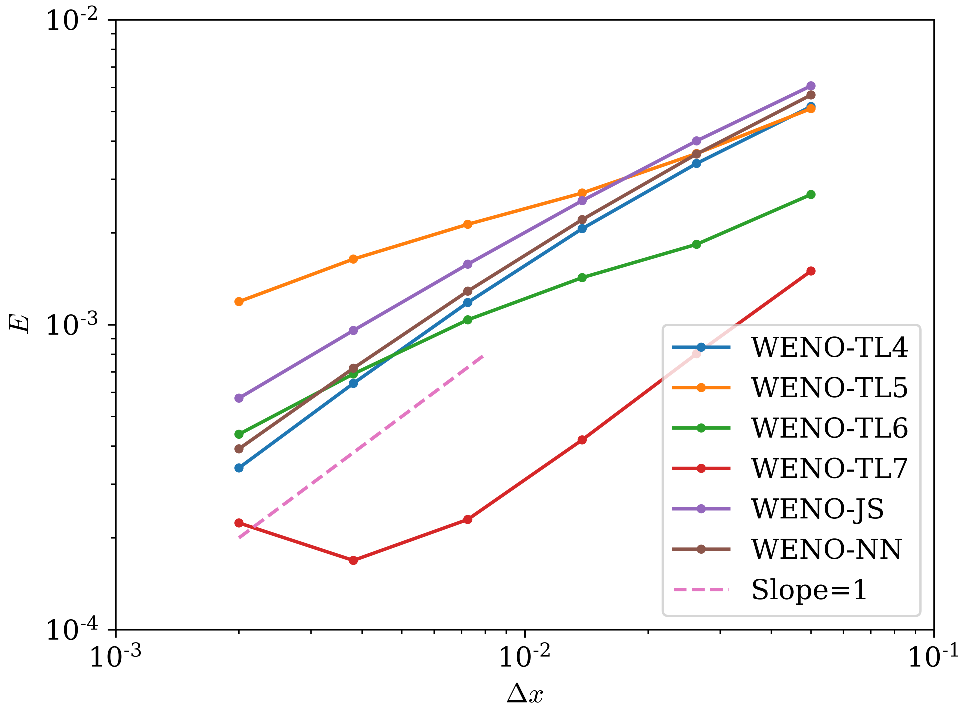


Figure 5.6: Convergence of each model on 1D density bump problem

with the convergence results from chapter 3. WENO-TL4 also shares this property, and maintains the faster convergence rate that WENO-NN is able to achieve. However, we see that WENO-TL7 does not continue to converge at resolutions much finer than the training resolution. This may seem unexpected because the scheme includes post-processing of the neural network outputs to make the scheme consistent. Hence, by using the Lax-Equivalence theorem, we conclude that there must be some loss of stability on finer grids, and that the more complicated model may have overfit to the training resolution.

We next look at the convergence properties for the 2D density bump case in Figure 5.7. For the 2D case, we keep $\Delta x = \Delta y$, and use six logarithmically spaced values between 0.005 and 0.05. For the initial density profile, we use the sum of two step functions in the x-direction and y-direction, or

$$\rho(0,x,y) = \begin{cases} 1, & \text{if } x \leq 1, y \leq 1 \\ 2, & \text{if } x > 1, y \leq 1 \\ 2, & \text{if } x \leq 1, y > 1 \\ 3, & \text{otherwise} \end{cases}, \tag{5.6}$$

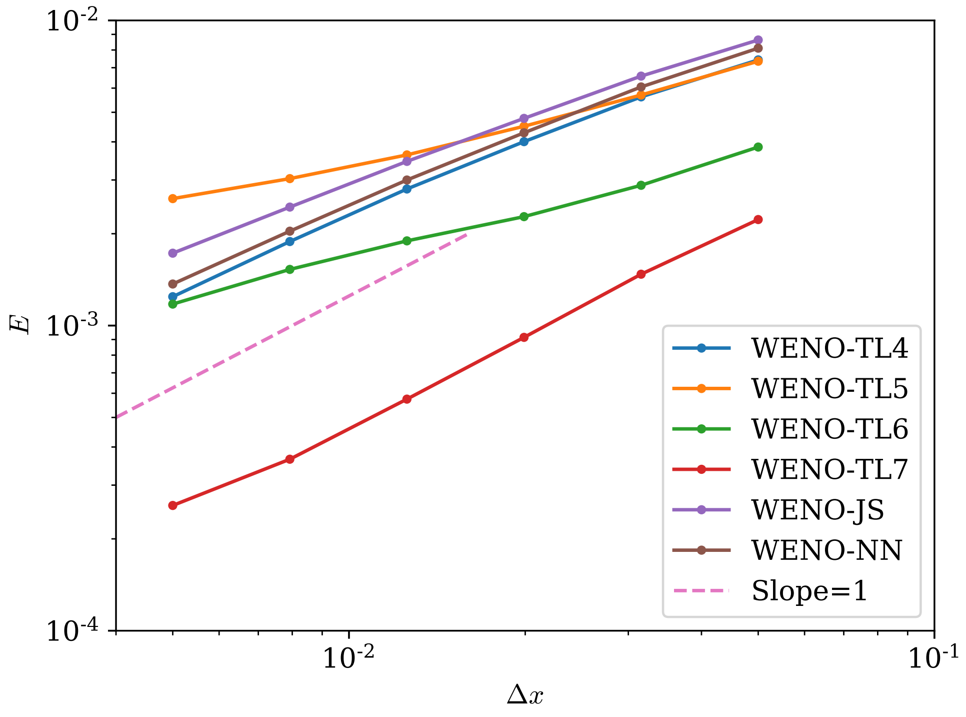


Figure 5.7: Convergence of each model on 2D density bump problem

In this study, the results are more in line with what one may expect, as we see the error E increases monotonically with discretization width Δx for each algorithm, though they do not quite appear to converge at a constant rate. We see much slower convergence rates for WENO-TL5 and WENO-TL6, with the other algorithms converging at similar speeds.

We also look at how quickly the simulation converges for each model for the 1D case with random initial conditions in Figure 5.8. For this case, we use an initial condition that is representative of what we saw in terms of accuracy for a typical case, solve the PDE on a very fine grid with WENO5-JS to get an 'exact' solution, and downsample by six different factors to see how error varies with discretization width.

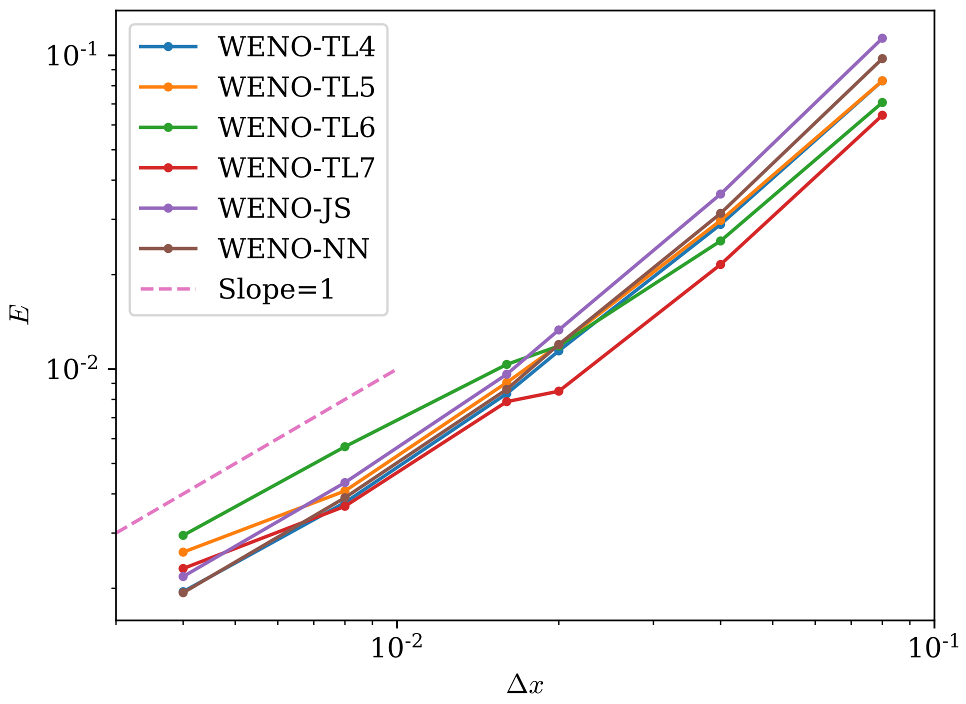


Figure 5.8: Convergence on 1D random initial conditions problems

We see that the curves are much more tightly bunched together for this case than they were for the density bump case. This result makes sense based on the accuracy results, as WENO-TL was not able to reduce the error as much for this case as we saw in the previous section. The results do appear to converge monotonically, though there is some jaggedness that is unexpected near the middle of the curve for WENO-TL6 and WENO-TL7. Overall we see fairly similar convergence rates between the different models, though the transfer learned models tend to perform poorly at finer resolutions.

Lastly, we also look at the convergence rates for each model for the 2D Euler equations with random initial conditions in Figure 5.9. Due to the large computational cost associated with performing this simulation on a fine grid, and wanting the 'exact' solution to be solved on a $10\times$ finer mesh than we analyze results for, we look only at fairly coarse grids. For the exact solution, we split up a 2×2 domain into an 800×800 grid and simulate for 2000 timesteps, which corresponds to a final time of $0.5s$ for a CFL number of 0.1. For the convergence study, we used the same domain and CFL number, and used grids with 80×80, 40×40, 20×20 and 10×10

cells.

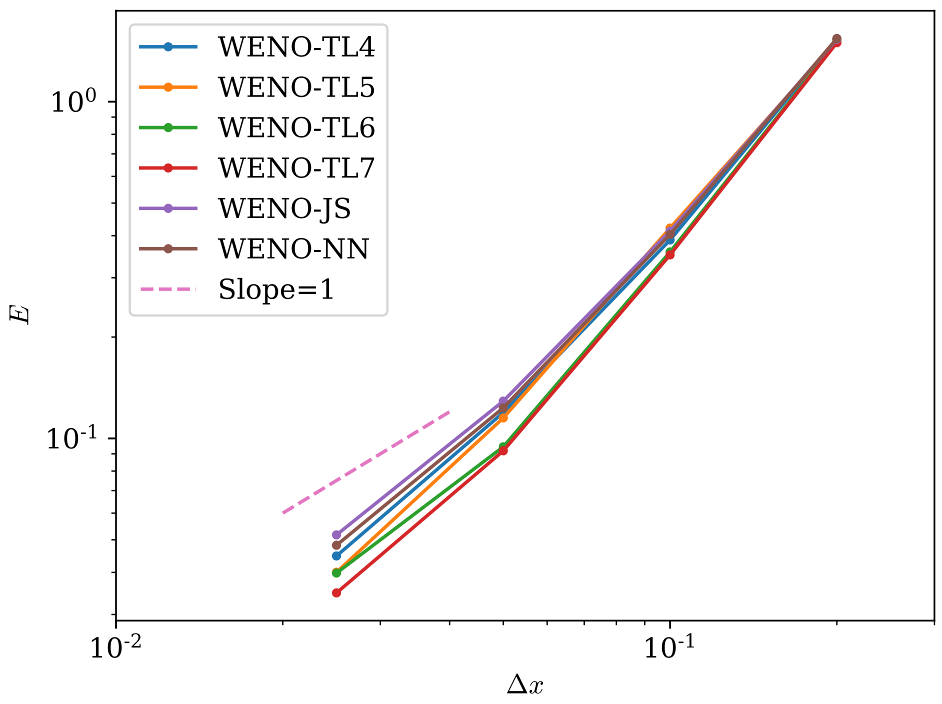


Figure 5.9: Convergence on 2D random initial conditions problems

We again see that the convergence curves are all fairly close to each other, especially on the coarsest grids. In fact, all of the algorithms produce a nearly identical error on a 10×10 grid, which may indicate that the grid is too fine for any of the algorithms tested and the simulation is dominated by numerical error.

5.3.4 Runtime Results

In this section, we examine the runtime of simulations using each model and the resulting error for a variety of different grids. We keep the initial condition the same throughout simulations, and instead vary the grid. We sweep over a range of values of Δx not seen in training to get a direct comparison between algorithms. We do not vary CFL number, as it was found that it has very little effect on the error ratio as long as each simulation is stable. We examine the runtime to gain a more direct understanding of the practicality of each method; a FVM scheme will not be useful if it can reduce error for one evaluation, but still ends up being more computationally expensive over the entire simulation. While this statement is true on some level for just about any computational problem, it is especially true for FVM schemes, as a lower error can easily be achieved with an FVM algorithm simply by refining the grid (with the caveat that one may run into memory issues for large problems, so it is not always possible to do this). Hence, it is especially important that a learned scheme is able to obtain competitive results in this type of study.

We can empirically measure the computational cost of one evaluation of each algorithm relative to WENO5-JS by simply taking ratio of the runtime required for the WENO5-JS simulation and the hybrid model being evaluated, though it will not be completely accurate because each simulation also requires other steps, such as the Lax-Friedrichs flux splitting. We call this quantity the time ratio and refer to it as T_r.

However, because this simulation is performed in PyTorch, there is a significant amount of overhead that would not be present in a version of the code that deployed the neural network in an optimal way. Hence, it is possible that these runtime values are not actually representative of the true potential of the enhanced methods. Rather than attempting to rewrite the codebase in a framework like NumPy, we will instead borrow code from the WENO-NN software which is already in NumPy because the model was not trained end-to-end. Isolating and comparing the runtimes of WENO5-JS and WENO-NN in this more comparable way gives a T_r value of 1.95, which is significantly smaller than the value of 2.8 that was obtained using the PyTorch software. In order to obtain similar corrected values for the transfer learned models, we hardcode the transfer learned models into the WENO-NN software using NumPy arrays and measure how long it takes to run each algorithm on 100,000 randomly generated datapoints to get a more direct comparison between the runtime of each algorithm programmed in a way that better reflects practical usecases and is not confounded by overhead associated with the rest of the simulation.

We see fairly dramatic differences in the values of the time ratios between the original PyTorch code and the optimized NumPy code. The most dramatic result can be seen in the time ratios associated with WENO-TL7, as we see a drop from 14 to 3.06, greater than a 4.5× reduction, which makes the algorithm seem very promising when deployed in an optimized program. When showing the results in this section, we will consider both the runtimes measured directly from running the PyTorch code, as well as an estimated NumPy runtime obtained by multiplying the WENO-JS runtime by the values of T_r obtained for each model using the NumPy code.

We first look at the runtime for the 1D density bump case in Figure 5.10. This data is generated from the same experiment as was used to generate Figure 5.6, with each simulation being timed in addition to looking at the numerical error.

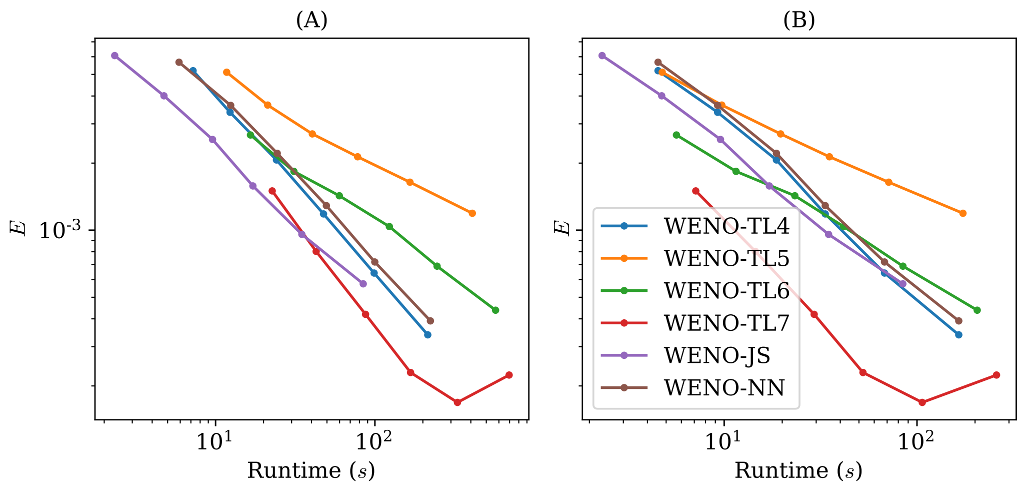


Figure 5.10: Runtime vs. error for each network at different grid resolutions for the 1D density bump problem. Shows (A) runtimes measured in PyTorch and (B) estimated NumPy runtimes

We see that WENO-JS and WENO-TL7 are very similar in terms of runtime and error for this density bump test case for the PyTorch code. Their curves actually intersect between the coarser WENO-TL7 and finer WENO-JS simulations, and due to the fast convergence of the learned algorithm we can see that we may actually be able to get a more accurate simulation at the same runtime on certain grid resolutions. Additionally, with the regions where this overlap occurs the memory requirements for the WENO-TL7 are much lower because of the coarser grid. For the points closest to overlapping, WENO-TL7 is using a 5× coarser grid, which of course corresponds to a 5× reduction in memory required. We see weaker performance from the other models tested. WENO-TL4 offers a small improvement over WENO-NN, with WENO-TL5 and WENO-TL6 performing relatively poorly. When looking at the estimated NumPy runtimes, the results become more promising. WENO-TL7 offers significant speedup potential over WENO-JS, and we see other algorithm such as WENO-TL6 and WENO-TL4 become competitive with WENO-JS.

We now measure runtime for the 2D density bump case. These results can be seen in Figure 5.11. Once again, these results were obtained from the same experiments used to generate the data in 5.7. Hence, it uses the same initial condition, domain, and discretizations.

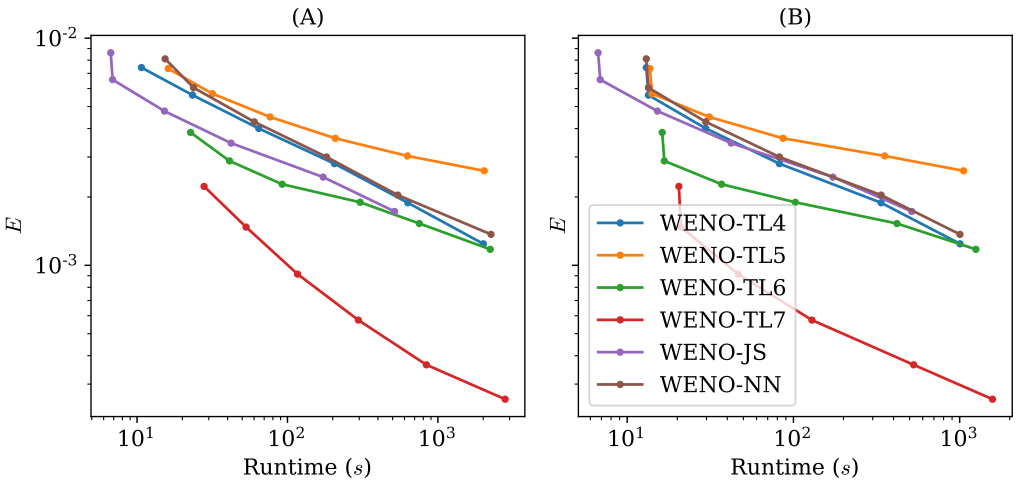


Figure 5.11: Runtime vs. error for each network at different grid resolutions for the 2D density bump problem. Shows (A) runtimes measured in PyTorch and (B) estimated NumPy runtimes

We see that WENO-TL7 performs very well relative to all other models tested in terms of runtime. In fact, one can compare certain specific datapoints to get an estimate of the speedup that WENO-TL7 is capable of delivering. The point $(44.72s, 0.0015)$ lies on the WENO-TL7 curve, and the point $(448.6s, 0.0017)$, which shows that in this case WENO-TL7 is capable of a more than $10\times$ speedup and delivers a lower error than WENO5-JS. Note that this simulation was performed in PyTorch, and an optimized code would most likely see a bigger speedup from WENO-TL7. To quantify memory reduction offered by WENO-TL7 for this problem, we can again look at how many cells each simulation used. The previously mentioned WENO-JS simulation used a 200×200 grid, while the WENO-TL7 simulation with comparable error used a 32×32 grid, which translates to a 97.44% reduction in memory required. WENO-TL6 also manages to outperform WENO-JS, but the other algorithms do not. Once again, we see that WENO-TL4 slightly outperforms WENO-NN, with WENO-TL5 performing worse than the other numerical schemes.

We also measure the runtime when solving the 1D Euler equations with random initial conditions. The results of this study can be seen in Figure 5.12. As for the previous two runtime studies, we again use the same simulation as the convergence study to obtain the data.

We see that the transfer learned schemes struggle to compete with WENO-JS for the

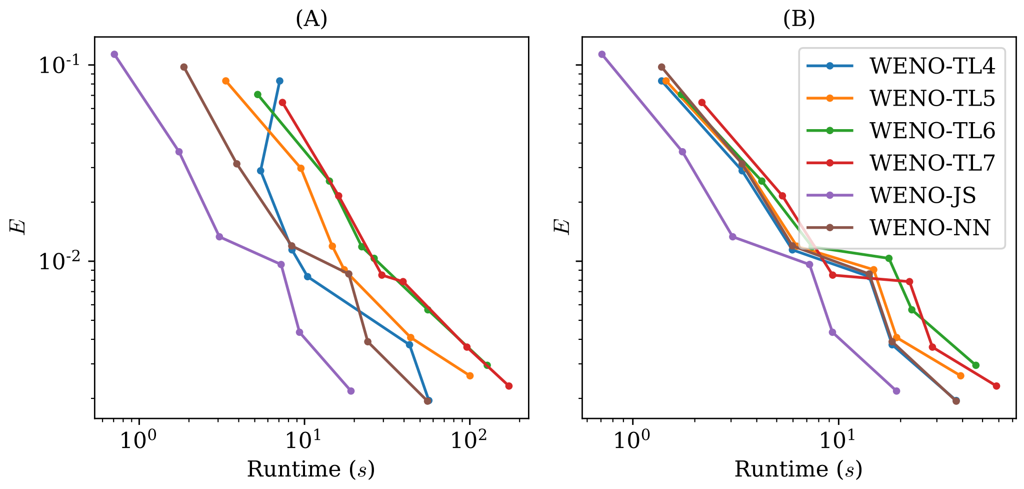


Figure 5.12: Runtime vs. error for each network at different grid resolutions for the 1D random initial condition problem. Shows (A) runtimes measured in PyTorch and (B) estimated NumPy runtimes

random initial condition case, even when using the NumPy estimates of runtime. It may be that this problem is too general for our framework to learn in an efficient way, or perhaps more training data would be required to learn numerical method that offers a faster runtime than WENO-JS. Interestingly, we see very different trends for this case than we saw for the density bump case. As model complexity increases, the efficiency of the scheme tends to decrease, with WENO-TL6 and WENO-TL7 performing the worst. This trend, along with the results of our previous experiments, seems to indicate that more training data and training the model for longer would be required to get a scheme that can perform better. We draw this conclusion because we saw in the density bump experiments increased model complexity leads to improved performance, and because this experiment involves a more complex/general problem, we should expect model complexity to again lead to improved model performance. However, because we do not see this, it would make sense that we simply did not train the network long enough for it to efficiently learn how to solve this class of problems. Lastly, one point that is worth mentioning is that these results were obtained with the PyTorch code, so perhaps we would see more competitive results from a more optimized software package.

Lastly, we perform the same experiment on the 2D Euler equations with random initial conditions, with the results of this study shown in Figure 5.13.

These results are similar to what we see from the 1D case, where the more complex

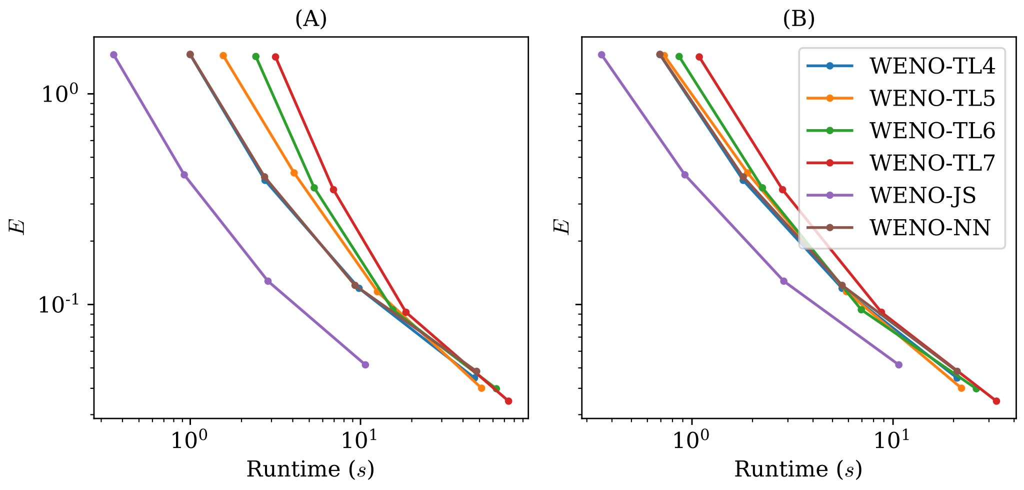


Figure 5.13: Runtime vs. error for each network at different grid resolutions for the 2D random initial condition problem. Shows (A) runtimes measured in PyTorch and (B) estimated NumPy runtimes

models tend to underperform the simpler models. WENO-JS runs the fastest, while WENO-TL7 is the slowest for most of the data. As this experiment involves the same models that were used for the previous experiments, the conclusions remain the same, where perhaps one would see better performance is the models were trained for longer or over more initial conditions. It may also be the case that because this case involves shocks rather than just contact discontinuities as was seen in the density bump case, WENO-TL is unable to offer much performance improvement, as was seen when testing WENO-NN on the inviscid Burgers' equations in Chapter 3.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusions

Throughout this thesis we have discussed the application of neural networks to finite volume methods. By combining machine learning with traditional ideas from FVM, we created several new frameworks for developing numerical schemes that can be tailored to specific families of problems, while also maintaining properties that are important for numerical schemes to have, such as consistency. After developing several new algorithms in this area, using them for simulations, and analyzing their performance, we now understand that these types of hybrid models present a promising avenue for reducing the computational cost of CFD simulations.

In Chapter 3, we developed a hybrid method called WENO-NN. By using a state of the art shock capturing method, WENO5-JS, to preprocess the local solution values and create inputs to a neural network that then perturbed the FVM coefficients of WENO5-JS, and then projecting those perturbed coefficients onto the space of consistent FVM schemes, we were able to use generic waveforms to train an improved shock capturing method. We showed that WENO-NN achieves faster convergence for discontinuous problems than WENO5-JS, at a rate of Δx^1, while WENO5-JS appears to only converge at a rate of $\Delta x^{0.82}$. In fact, it is impossible for an FVM scheme to converge faster than Δx^1, so our network is performing at the theoretical optimum in terms of convergence for these types of problems. We also show that WENO-NN is able to outperform WENO5-JS in terms of runtime for certain problems, such as when WENO5-JS applies too much numerical diffusion to the solution. More specifically, for a given error tolerance, a simulation using WENO-NN will take less time to run than a simulation using WENO5-JS. Also, because WENO-NN achieves this runtime improvement by running a more expensive algorithm per iteration on a coarser grid, WENO-NN also results in less memory required to run the simulation.

In Chapter 4, we developed a framework for learning and embedding coarse graining models into an FDM or FVM scheme called FiniteNet. While it has been popular in the FVM literature to utilize a fully convolutional model due to the spatial locality of the differential operators involved in PDEs, FiniteNet introduces the idea of also including an LSTM component to learn and track temporal patterns in PDEs and error accumulation present when solving them numerically. We demonstrated this framework on three PDEs that each feature different behavior, namely the advection equation which propagates contact discontinuities, the inviscid Burgers' equation which develops shockwaves, the Kuramoto-Sivashinsky equation which has chaotic solutions. We showed that for each equation, FiniteNet was able to significantly reduce the numerical error. We also show that using an LSTM leads to an error reduction over other temporal modeling techniques, which all outperform a model that does not contain any temporal component.

In Chapter 5, we apply transfer learning to WENO-NN to develop equation specific shock capturing methods, and name the resulting algorithm WENO-TL. By using the parameters of WENO-NN as the initial weights for the network in WENO-TL, the training results are more consistent, which alleviates the need for aggressive early stopping that was required when training WENO-NN. We then see that by training the network on equation specific data, we can get a larger error reduction than is achieved by WENO-NN. We observe that the resulting accuracy improvements are large enough to where WENO-TL7, which is more than 3× as expensive as WENO5-JS per evaluation, is able to obtain the same accuracy levels in less runtime by using a coarser grid on the density bump test case. Using a coarser grid has the added benefit of reducing the memory requirements of the simulation, which can by a major bottleneck in simulating large-scale turbulent flows. We also see very promising generalization behaviors from WENO-TL, in that we can train the network on solutions of the 1D Euler equations and see extremely similar results in terms of error relative to WENO5-JS performance when testing those same networks on the 2D Euler equations. However, we also see that if the space of initial conditions that we try to train the model on is too broad, the learned scheme does not perform as well.

6.2 Future Work

There remain many promising directions to investigate in the field of data-driven finite-volume methods. At this point, many different architectures and techniques have been developed for data-driven approaches to FVM and FDM. However, this problem setting has several interesting properties that perhaps warrant more detailed studies of how much of a performance improvement can be achieved in practical settings.

In the field of numerical PDEs, there is rigorous mathematical theory that can be used to derive coefficients with optimal convergence properties for smooth solutions, and human intuition is indeed still useful for developing what could be considered rule-based AI in shock-capturing methods such as WENO5-JS. This point leads to a key difference between FVMs and applications domains where such an approach is ineffective such as in image processing: the ML algorithm must outperform simple algorithms that are already highly effective. The simulation error can even be reduced to arbitrarily small levels by simply refining the grid. Hence, there exists an additional challenge in that it is not beneficial to simply make the model larger for marginal accuracy improvements, as increased evaluation costs must be carefully balanced with accuracy improvements. More specifically, traditional FVM and FDM algorithms themselves are not expensive. High order methods are simply an inner product between a constant vector of coefficients and the local solution, and even sophisticated algorithms such as WENO5-JS are not particularly expensive per iteration. Instead, the cost lies in needing to repeat these calculations many different times for a simulation that is being carried out on a fine grid for many timesteps. So the data driven method is replacing an algorithm that is fairly inexpensive, which leads to the constraint that the neural network size cannot be too large because the evaluation cost quickly becomes large compared to approaches that are not data-driven. So speed must be balanced carefully with accuracy. More detailed studies of the tradeoffs between model complexity and simulation runtime would be useful for accelerating the development of practical data-driven FVM schemes.

Additionally, there is much room for research studying the robustness of these schemes. Robustness is a very important feature for industrial CFD codes such as those developed by ANSYS, Inc. The algorithms used in these software packages must give accurate solutions on irregular meshes when analyzing flows involving complex geometries, which can be common in engineering applications. These meshes are typically generated algorithmically, which may produce poorly shaped cells that have high aspect ratios. While we looked at a large space of initial conditions in this thesis when assessing our algorithms, we looked at only a fixed cell width on a square domain. Robustness to practical concerns such as high aspect ratio elements remains unknown for the approaches developed in this thesis. Poorly shaped cells can cause the problem to be poorly conditioned, which can crash the simulation. Training an FVM scheme to be robust to poorly shaped cells could potentially avoid this issue, which would be extremely beneficial to engineers working in industry.

It would also be interesting to investigate the effectiveness of these algorithms on simulations with more complicated physics, such as multiphase or turbulent flows. Based on what we have observed in the experiments we have performed, these algorithms are very promising for simulating PDEs that have complicated solution structures, as we saw larger performance gains on the KS equation than on the inviscid Burgers' or advection equations when testing FiniteNet. Hence, WENO-TL seems very promising for large-scale turbulent flow problems, as one could attempt to train the network on 1D forced Navier-Stokes equation and then deploy the learned numerical scheme on a 3D turbulent flow to potentially obtain a significant speedup.

One other contribution that could accelerate development of data-driven PDE solvers would be the development of a standardized dataset and selection of model problems that could be used to more directly compare the performance of these types of algorithms. As it stands now, each paper that proposes a new framework for developing hybrid FVMs uses a unique dataset generally generated for that paper, and then tests their algorithm on problems based on the dataset they use. This approach makes it hard to get a direct comparison between different methods. If standardized experiments were developed for this field and it was expected that new algorithms would showcase their performance on them, it would highlight the usefulness and potential of specific algorithms in a more comparable way.

Lastly, there are a number of investigations specific to the work in this thesis that could be useful to explore. It would be interesting to perform further studies on WENO-TL. It seems likely that more training data would lead to better results on more general problems, such as the case of random initial conditions. It would also be interesting to see how specific one could make the training data relative to the problem of interest. For example, one of the main test cases we looked at when analyzing the performance of WENO-TL was the density bump case, and although we looked at a large space of initial conditions, the dynamics of the physical system are quite simple. The other case we examined had a very large space of initial conditions and complicated dynamics, but we do not have a test case that looks at a more restricted set of initial conditions with complicated dynamics, and this may be a promising class of problems to explore. It could also be useful to try training these algorithms on loss functions that are designed based on the problem of interest. For example, one may want to directly penalize oscillations when learning a shock capturing method, or minimize the error of a quantity of interest other than the L_2 error such as lift and drag error when designing a scheme to simulate airfoils.

Overall, using machine learning to solve PDEs appears to be a promising research direction. Many interesting approaches have been proposed, several of which are extremely different from one another. This variety of approaches reflects the diverse set of challenges faced in the field of PDEs. For example, the techniques presented in this thesis are fundamentally very different from the neural operators approach followed by other researchers (Li, Kovachki, et al., 2020c). I would expect to see the FVM/FDM approach shown in this thesis to deliver improvements in the area of large-scale computational physics, where the simulation is being carried out for the purpose of then investigating a physical phenomena. This approach has the potential to increase the space of solvable problems to allow researchers to investigate systems that were previously too expensive to simulate. The neural operators approach seems especially promising in the application domain of real-time flow control. The neural operators model learns how to solve the PDE directly, which could give a very large speedup when solving many perturbations of the same problem, such as slightly different configurations of an actively controlled airfoil.

BIBLIOGRAPHY

Bar-Sinai et al. (2019). “Learning data-driven discretizations for partial differential equations”. In: *Proceedings of the National Academy of Sciences* 116.31, pp. 15344–15349.

Bazilevs et al. (2008). “Isogeometric fluid-structure interaction: theory, algorithms, and computations”. In: *Computational mechanics* 43.1, pp. 3–37.

Bezgin, Schmidt, and Adams (2021). “A data-driven physics-informed finite-volume scheme for nonclassical undercompressive shocks”. In: *Journal of Computational Physics* 437, p. 110324.

Borges et al. (2008). “An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws”. In: *Journal of Computational Physics* 227.6, pp. 3191–3211.

Borggaard, Burns, and Zietsman (2004). “Computational challenges in control of partial differential equations”. In: *2nd AIAA Flow Control Conference*, p. 2526.

Brunton, Noack, and Koumoutsakos (2020). “Machine learning for fluid mechanics”. In: *Annual Review of Fluid Mechanics* 52, pp. 477–508.

Brunton, Proctor, and Kutz (2016). “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the national academy of sciences* 113.15, pp. 3932–3937.

Bryngelson and Colonius (2019). “Annular and spiral bubble nets: A simulation-focused analysis of humpback whale feeding strategies”. In: *The Journal of the Acoustical Society of America* 146.4, pp. 2771–2771.

– (2020). “Simulation of humpback whale bubble-net feeding models”. In: *The Journal of the Acoustical Society of America* 147.2, pp. 1126–1135.

Bungartz and Schäfer (2006). *Fluid-structure interaction: modelling, simulation, optimisation*. Vol. 53. Springer Science & Business Media.

Castro, Costa, and Don (2011). “High order weighted essentially non-oscillatory WENO-Z schemes for hyperbolic conservation laws”. In: *Journal of Computational Physics* 230.5, pp. 1766–1792.

Chen, Liu, and Beardsley (2003). “An unstructured grid, finite-volume, three-dimensional, primitive equations ocean model: application to coastal ocean and estuaries”. In: *Journal of atmospheric and oceanic technology* 20.1, pp. 159–186.

Chen, Patel, and Ju (1990). “Solutions of Reynolds-averaged Navier-Stokes equations for three-dimensional incompressible flows”. In: *Journal of Computational Physics* 88.2, pp. 305–336.

Chollet et al. (2015). *Keras*. `https://keras.io`.

Chu (1979). “Numerical methods in fluid dynamics”. In: *Advances in Applied Mechanics*. Vol. 18. Elsevier, pp. 285–331.

Davidson (2002). *An introduction to magnetohydrodynamics*.

Deville et al. (2002). *High-order methods for incompressible fluid flow*. Vol. 9. Cambridge university press.

DeVore (1991). “Flux-corrected transport techniques for multidimensional compressible magnetohydrodynamics”. In: *Journal of Computational Physics* 92.1, pp. 142–160.

Dissanayake and Phan-Thien (1994). “Neural-network-based approximations for solving partial differential equations”. In: *communications in Numerical Methods in Engineering* 10.3, pp. 195–201.

Dowell and Hall (2001). “Modeling of fluid-structure interaction”. In: *Annual review of fluid mechanics* 33.1, pp. 445–490.

Drikakis and Geurts (2006). *Turbulent flow computation*. Vol. 66. Springer Science & Business Media.

Fang, Li, and Lu (2013). “An optimized low-dissipation monotonicity-preserving scheme for numerical simulations of high-speed turbulent flows”. In: *Journal of Scientific Computing* 56.1, pp. 67–95.

Fukushima (1980). “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological cybernetics* 36.4, pp. 193–202.

Funaki (1995). “The scaling limit for a stochastic PDE and the separation of phases”. In: *Probability Theory and Related Fields* 102.2, pp. 221–288.

Gammie, McKinney, and Tóth (2003). “HARM: a numerical scheme for general relativistic magnetohydrodynamics”. In: *The Astrophysical Journal* 589.1, p. 444.

Gers, Schmidhuber, and Cummins (1999). “Learning to forget: Continual prediction with LSTM”. In:

Givoli (1991). “Non-reflecting boundary conditions”. In: *Journal of computational physics* 94.1, pp. 1–29.

Gottlieb and Shu (1997). “On the Gibbs phenomenon and its resolution”. In: *SIAM review* 39.4, pp. 644–668.

– (1998). “Total variation diminishing Runge-Kutta schemes”. In: *Mathematics of computation of the American Mathematical Society* 67.221, pp. 73–85.

Gustafsson and Holmgren (2010). “An implementation framework for solving high-dimensional PDEs on massively parallel computers”. In: *Numerical Mathematics and Advanced Applications 2009*. Springer, pp. 417–424.

Ha et al. (2013). “An improved weighted essentially non-oscillatory scheme with a new smoothness indicator”. In: *Journal of Computational Physics* 232.1, pp. 68–86.

Hagge et al. (2017). “Solving differential equations with unknown constitutive relations as recurrent neural networks”. In: *arXiv preprint arXiv:1710.02242*.

Halpern (1993). *Large eddy simulation of complex engineering and geophysical flows*. Cambridge University Press.

Harten (1983). “High resolution schemes for hyperbolic conservation laws”. In: *Journal of computational physics* 49.3, pp. 357–393.

Harten et al. (1987). “Uniformly high order accurate essentially non-oscillatory schemes, III”. In: *Upwind and high-resolution schemes*. Springer, pp. 218–290.

He et al. (2016). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hochreiter and Schmidhuber (1997a). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780.

– (1997b). “LSTM can solve hard long time lag problems”. In: *Advances in neural information processing systems*, pp. 473–479.

Hsieh et al. (2019). “Learning neural PDE solvers with convergence guarantees”. In: *arXiv preprint arXiv:1906.01200*.

Hyman and Nicolaenko (1986). “The Kuramoto-Sivashinsky equation: a bridge between PDE’s and dynamical systems”. In: *Physica D: Nonlinear Phenomena* 18.1-3, pp. 113–126.

Ishihara, Gotoh, and Kaneda (2009). “Study of high–Reynolds number isotropic turbulence by direct numerical simulation”. In: *Annual Review of Fluid Mechanics* 41, pp. 165–180.

Jang et al. (2012). “An analysis of dispersion and dissipation properties of Hermite methods and its application to direct numerical simulation of jet noise”. In: *18th AIAA/CEAS Aeroacoustics Conference (33rd AIAA Aeroacoustics Conference)*, p. 2240.

Jiang and Shu (1996). “Efficient implementation of weighted ENO schemes”. In: *Journal of computational physics* 126.1, pp. 202–228.

Johnsen and Colonius (2008). “Shock-induced collapse of a gas bubble in shockwave lithotripsy”. In: *The Journal of the Acoustical Society of America* 124.4, pp. 2011–2020.

Jordan and Colonius (2013). “Wave packets and turbulent jet noise”. In: *Annual review of fluid mechanics* 45, pp. 173–195.

Karniadakis et al. (2021). “Physics-informed machine learning”. In: *Nature Reviews Physics*, pp. 1–19.

Kharazmi, Zhang, and Karniadakis (2021). "hp-VPINNs: Variational physics-informed neural networks with domain decomposition". In: *Computer Methods in Applied Mechanics and Engineering* 374, p. 113547.

Kim, Ha, and Yoon (2016). "Modified non-linear weights for fifth-order weighted essentially non-oscillatory schemes". In: *Journal of Scientific Computing* 67.1, pp. 299–323.

Kim, Kim, and Choi (2001). "An immersed-boundary finite-volume method for simulations of flow in complex geometries". In: *Journal of computational physics* 171.1, pp. 132–150.

Kim and Lee (1996). "Optimized compact finite difference schemes with maximum resolution". In: *AIAA journal* 34.5, pp. 887–893.

Kingma and Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Kochkov et al. (2021). "Machine learning–accelerated computational fluid dynamics". In: *Proceedings of the National Academy of Sciences* 118.21.

Lagaris, Likas, and Fotiadis (1998). "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE transactions on neural networks* 9.5, pp. 987–1000.

LeCun, Boser, et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

LeCun, Bottou, et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lele (1992). "Compact finite difference schemes with spectral-like resolution". In: *Journal of computational physics* 103.1, pp. 16–42.

LeVeque (1992). *Numerical methods for conservation laws*. Vol. 132. Springer.

LeVeque et al. (2002). *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge university press.

LeVeque (2007). *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. Vol. 98. Siam.

Li, Kovachki, et al. (2020a). "Fourier neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2010.08895*.

– (2020b). "Multipole graph neural operator for parametric partial differential equations". In: *arXiv preprint arXiv:2006.09535*.

– (2020c). "Neural operator: Graph kernel network for partial differential equations". In: *arXiv preprint arXiv:2003.03485*.

Li and Qiu (2010). "Hybrid weighted essentially non-oscillatory schemes with different indicators". In: *Journal of Computational Physics* 229.21, pp. 8105–8129.

Li, Yu, et al. (2017). “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting”. In: *arXiv preprint arXiv:1707.01926*.

Linan and Williams (1993). “Fundamental aspects of combustion”. In:

Ling, Kurzawski, and Templeton (2016). “Reynolds averaged turbulence modelling using deep neural networks with embedded invariance”. In: *Journal of Fluid Mechanics* 807, pp. 155–166.

Liska and Colonius (2017). “A fast immersed boundary method for external incompressible viscous flows using lattice Green’s functions”. In: *Journal of Computational Physics* 331, pp. 257–279.

Liu (2013). “Globally optimal finite-difference schemes based on least squares”. In: *Geophysics* 78.4, T113–T132.

Long, Lu, and Dong (2019). “PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network”. In: *Journal of Computational Physics* 399, p. 108925.

Long, Lu, Ma, et al. (2017). “PDE-net: Learning PDEs from data”. In: *arXiv preprint arXiv:1710.09668*.

Maeda et al. (2018). “Energy shielding by cavitation bubble clouds in burst wave lithotripsy”. In: *The Journal of the Acoustical Society of America* 144.5, pp. 2952–2961.

Mahesh, Constantinescu, and Moin (2004). “A numerical method for large-eddy simulation in complex geometries”. In: *Journal of Computational Physics* 197.1, pp. 215–240.

Mao, Jagtap, and Karniadakis (2020). “Physics-informed neural networks for high-speed flows”. In: *Computer Methods in Applied Mechanics and Engineering* 360, p. 112789.

Mengaldo et al. (2017). “Immersed boundary lattice Green function methods for external aerodynamics”. In: *23rd AIAA computational fluid dynamics conference*, p. 3621.

Mohan et al. (2019). “Compressed convolutional LSTM: An efficient deep learning framework to model high fidelity 3D turbulence”. In: *arXiv preprint arXiv:1903.00033*.

Mohd-Yusof (1997). “For simulations of flow in complex geometries”. In: *Annual Research Briefs* 317, p. 35.

Molinaro et al. (2021). “Embedding data analytics and CFD into the digital twin concept”. In: *Computers & Fluids* 214, p. 104759.

Nelsen and Stuart (2020). “The random feature model for input-output maps between banach spaces”. In: *arXiv preprint arXiv:2005.10224*.

Novati, de Laroussilhe, and Koumoutsakos (2021). “Automating turbulence modelling by multi-agent reinforcement learning”. In: *Nature Machine Intelligence* 3.1, pp. 87–96.

Pang, Lu, and Karniadakis (2019). “fPINNs: Fractional physics-informed neural networks”. In: *SIAM Journal on Scientific Computing* 41.4, A2603–A2626.

Peer, Dauhoo, and Bhuruth (2009). “A method for improving the performance of the WENO5 scheme near discontinuities”. In: *Applied Mathematics Letters* 22.11, pp. 1730–1733.

Peters (2001). *Turbulent combustion*.

Pfaff et al. (2020). “Learning Mesh-Based Simulation with Graph Networks”. In: *arXiv preprint arXiv:2010.03409*.

Pfau et al. (2018). “Spectral Inference Networks: Unifying Deep and Spectral Learning”. In: *arXiv preprint arXiv:1806.02215*.

Pickering et al. (2020). “Resolvent-based jet noise models: a projection approach”. In: *AIAA Scitech 2020 Forum*, p. 0999.

Pishchalnikov et al. (2003). “Cavitation bubble cluster activity in the breakage of kidney stones by lithotripter shockwaves”. In: *Journal of endourology* 17.7, pp. 435–446.

Puckett and Stewart (1950). “The thickness of a shock wave in air”. In: *Quarterly of Applied Mathematics* 7.4, pp. 457–463.

Raissi (2018). “Deep hidden physics models: Deep learning of nonlinear partial differential equations”. In: *The Journal of Machine Learning Research* 19.1, pp. 932–955.

Raissi, Perdikaris, and 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.

Ranade, Hill, and Pathak (2021). “DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization”. In: *Computer Methods in Applied Mechanics and Engineering* 378, p. 113722.

Rathan and Raju (2018a). “A modified fifth-order WENO scheme for hyperbolic conservation laws”. In: *Computers & Mathematics with Applications* 75.5, pp. 1531–1549.

– (2018b). “Improved weighted ENO scheme based on parameters involved in nonlinear weights”. In: *Applied Mathematics and Computation* 331, pp. 120–129.

Reba et al. (2003). “A study of the role of organized structures in jet noise generation”. In: *9th AIAA/CEAS Aeroacoustics Conference and Exhibit*, p. 3314.

Ricardez-Sandoval (2011). “Current challenges in the design and control of multiscale systems”. In: *The Canadian Journal of Chemical Engineering* 89.6, pp. 1324–1341.

Rudd (2013). “Solving partial differential equations using artificial neural networks”. PhD thesis. Duke University.

Sahli et al. (2020). “Physics-informed neural networks for cardiac activation mapping”. In: *Frontiers in Physics* 8, p. 42.

Shin, Darbon, and Karniadakis (2020). “On the convergence and generalization of physics informed neural networks”. In: *arXiv preprint arXiv:2004.01806*.

Shu (1998). “Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws”. In: *Advanced numerical approximation of nonlinear hyperbolic equations*. Springer, pp. 325–432.

– (2003). “High-order finite difference and finite volume WENO schemes and discontinuous Galerkin methods for CFD”. In: *International Journal of Computational Fluid Dynamics* 17.2, pp. 107–118.

Sirignano and Spiliopoulos (2018). “DGM: A deep learning algorithm for solving partial differential equations”. In: *Journal of Computational Physics* 375, pp. 1339–1364.

Sommerfeld (1949). *Partial differential equations in physics*. Academic press.

Spalart (2010). “Reflections on RANS modelling”. In: *Progress in Hybrid RANS-LES Modelling*. Springer, pp. 7–24.

Strogatz (2001). “Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering (studies in nonlinearity)”. In:

Su, Yuanhang and C-C Jay Kuo (2019). “On extended long short-term memory and dependent bidirectional recurrent neural network”. In: *Neurocomputing* 356, pp. 151–161.

Tallec, Le and Mouro (2001). “Fluid structure interaction with large structural displacements”. In: *Computer methods in applied mechanics and engineering* 190.24-25, pp. 3039–3067.

Tam and Webb (1993). “Dispersion-relation-preserving finite difference schemes for computational acoustics”. In: *Journal of computational physics* 107.2, pp. 262–281.

Torrey and Shavlik (2010). “Transfer learning”. In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, pp. 242–264.

Veynante and Vervisch (2002). “Turbulent combustion modeling”. In: *Progress in energy and combustion science* 28.3, pp. 193–266.

Vlachas, Byeon, et al. (2018). “Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474.2213, p. 20170844.

Vlachas, Pathak, et al. (2020). “Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics”. In: *Neural Networks* 126, pp. 191–217.

Wan et al. (2018). “Data-assisted reduced-order modeling of extreme events in complex dynamical systems”. In: *PloS one* 13.5.

Wang and Chen (2001). “Optimized weighted essentially nonoscillatory schemes for linear waves with discontinuity”. In: *Journal of Computational Physics* 174.1, pp. 381–404.

Wang, Teng, and Perdikaris (2020). “Understanding and mitigating gradient pathologies in physics-informed neural networks”. In: *arXiv preprint arXiv:2001.04536*.

Werbos (1990). “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Wilcox (1998). *Turbulence modeling for CFD*. Vol. 2. DCW industries La Canada, CA.

Xingjian et al. (2015). “Convolutional LSTM network: A machine learning approach for precipitation nowcasting”. In: *Advances in neural information processing systems*, pp. 802–810.

Yang and Perdikaris (2019). “Adversarial uncertainty quantification in physics-informed neural networks”. In: *Journal of Computational Physics* 394, pp. 136–152.

Yang, Zafar, et al. (2019). “Predictive large-eddy-simulation wall modeling via physics-informed neural networks”. In: *Physical Review Fluids* 4.3, p. 034602.

Yee (1987). “Upwind and symmetric shock-capturing schemes”. In:

Yu, Dorschner, and Colonius (2020). “Multi-resolution lattice Green’s function method for incompressible flows”. In: *arXiv preprint arXiv:2010.13213*.

Yu, Hesthaven, and Yan (2018). *A data-driven shock capturing approach for discontinuous Galerkin methods*. Tech. rep.

Yu, Zheng, et al. (2017). “Long-term forecasting using tensor-train rnns”. In: *Arxiv*.

Zhang, Lu, et al. (2019). “Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems”. In: *Journal of Computational Physics* 397, p. 108850.

Zhang and Yao (2013). “Optimized explicit finite-difference schemes for spatial derivatives using maximum norm”. In: *Journal of Computational Physics* 250, pp. 511–526.

Zhiyin (2015). “Large-eddy simulation: Past, present and the future”. In: *Chinese journal of Aeronautics* 28.1, pp. 11–24.