Exploiting Structure for Scalable and Robust Deep Learning

Thesis by
Stephan Tao Zheng

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

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2018
Defended April 23, 2018

ACKNOWLEDGEMENTS

In this universe, the two most fascinating objects are black holes and the human brain. Both represent extremes in our world: black holes are the most exotic space-time objects that we currently know about, while the human brain is the most advanced form of intelligence that we have observed so far. In the past four years, after having worked in theoretical physics, I have been fortunate to have had the opportunity to do research on machine learning / artificial intelligence and study mathematical recognition and decision-making models that have the potential to emulate and surpass human brains and enable artificial general intelligence. Hence, contributing to the field and completing this thesis has been a tremendously enriching and gratifying experience.

This journey and thesis would not have been possible without the help and support of many people, whiteboard markers, lines of code, and cups of coffee.

First and foremost, I want to thank Yisong Yue, for being my advisor and mentor in the past four years, and for giving me the opportunity to work on one of the most exciting areas of science in our time. Our many meetings, shared insights, discussions and speculation have made the research process exciting and enjoyable for me, and encouraged me to move forward in this field. Without your support and numerous small and large pieces of advice, this thesis would not have been possible.

Second, I want to thank several organizations for supporting my research: the Powell Foundation, the National Science Foundation, Bloomberg, Activision/Blizzard and Northrop Grumman. In particular, I want to thank STATS for providing sports tracking data.

I am very grateful to my collaborators that I have had the pleasure to work with (in no particular order): Patrick Lucey, Yang Song, Thomas Leung, Ian Goodfellow, Rose Yu, Anima Anandkumar, Eric Zhan, and Sumanth Dathathri. I have learned many valuable lessons from our interactions.

I also want to thank my PhD committee, Pietro Perona, Frank Porter and Ilya Sutskever, for your time and advice. Your help has greatly improved my work.

I am indebted to many: Bassam, Karthik, Petr, Minyoung, Minfeng, Ramya, Welmar, for all the coffees and dinners filled with stimulating discussions, ranging from the merits of semi-classical quantum gravity versus topological string theory (it’s a tie) to blockchain-based economies (why not?). It was always fun to discuss the latest ideas and problems in machine learning with my labmates: Hoang, Jialin, Eric, Yuxin, Yanan, Rose: thank you for all our interactions. Furthermore, I want to thank the many people at Caltech and the Physics and CMS departments who I have interacted with and who provided a unique atmosphere in which I have learned so much.

I am especially thankful to my family, my parents and sister Sarah, for all their support and patience while I was away overseas to pursue science.

And finally, a very special thanks to Rose, whose persistent support and encouragement are singular and unmatched.

ABSTRACT

Deep learning has seen great success training deep neural networks for complex prediction problems, such as large-scale image recognition, short-term time-series forecasting, and learning behavioral models for games with simple dynamics. However, neural networks have a number of weaknesses: 1) they are not sample-efficient and 2) they are often not robust against (adversarial) input perturbations. Hence, it is challenging to train neural networks for problems with exponential complexity, such as multi-agent games, complex long-term spatiotemporal dynamics, or noisy high-resolution image data.

This thesis contributes methods to improve the sample efficiency, expressive power, and robustness of neural networks, by exploiting various forms of low-dimensional *structure*, such as spatiotemporal hierarchy and multi-agent coordination. We show the effectiveness of this approach in multiple learning paradigms: in both the supervised learning (e.g., imitation learning) and reinforcement learning settings.

First, we introduce hierarchical neural networks that model both short-term actions and long-term goals from data, and can learn human-level behavioral models for spatiotemporal multi-agent games, such as basketball, using imitation learning.

Second, in reinforcement learning, we show that behavioral policies with a hierarchical latent structure can efficiently learn forms of multi-agent coordination, which enables a form of structured exploration for faster learning.

Third, we showcase tensor-train recurrent neural networks that can model high-order mutliplicative structure in dynamical systems (e.g., Lorenz dynamics). We show that this model class gives state-of-the-art long-term forecasting performance with very long time horizons for both simulation and real-world traffic and climate data.

Finally, we demonstrate two methods for neural network robustness: 1) stability training, a form of stochastic data augmentation to make neural networks more robust, and 2) neural fingerprinting, a method that detects adversarial examples by validating the network's behavior in the nieghborhood of any given input.

In sum, this thesis takes a step to enable machine learning for the next scale of problem complexity, such as rich spatiotemporal multi-agent games and large-scale robust predictions.

PUBLISHED CONTENT AND CONTRIBUTIONS

Dathathri, Sumanth et al. (2018). "Detecting Adversarial Examples via Neural Fingerprinting". In: *arXiv preprint arXiv:1803.03870*. S.T.Z. participated in the conception of the project, analyzed the method and experimental results, provided theoretical analyses and participated in the writing of the manuscript.

Zhan, Eric et al. (2018). "Generative Multi-Agent Behavioral Cloning". In: *arXiv preprint arXiv:1803.07612*. S.T.Z. participated in the conception of the project and formulation of the method, analyzed the method and experimental results, and participated in the writing of the manuscript.

Yu, Rose et al. (2017). "Long-term forecasting using tensor-train RNNs". In: *arXiv preprint arXiv:1711.00073*. S.T.Z. participated in the formulation of the project, analyzed the method and all experiments, and participated in the writing of the manuscript.

Zheng, Stephan, Yang Song, et al. (2016). "Improving the robustness of deep neural networks via stability training". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. S.T.Z. participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted all experiments and participated in the writing of the manuscript., pp. 4480–4488. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Zheng_Improving_the_Robustness_CVPR_2016_paper.pdf.

Zheng, Stephan, Yisong Yue, and Patrick Lucey (2016). "Generating long-term trajectories using deep hierarchical networks". In: *Advances in Neural Information Processing Systems*. S.T.Z. participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted all experiments and user studies, and participated in the writing of the manuscript., pp. 1543–1551. URL: https://papers.nips.cc/paper/6520-generating-long-term-trajectories-using-deep-hierarchical-networks.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION: CHALLENGES TOWARDS SCALABLE AND ROBUST MACHINE LEARNING

In recent years, the rapid increase in availability of large-scale data and computational power has fueled a great surge in the development and application of *machine learning*. Broadly speaking, the goal of machine learning is to understand and develop models and algorithms that can discover, describe, and leverage useful *structure in data*. There are numerous successful applications of this approach in a wide range of domains, such as computer vision (e.g., image classifiers), language processing (e.g., translation), recommendation systems, and many others.

In this work, we primarily consider machine learning in the setting of *sequential decision problems*, in which an intelligent *agent* interacts with its environment, which could contain other agents. In particular, we consider complex problem settings where the environment dynamics are highly nonlinear (e.g., agents move along piecewise smooth trajectories of high curvature), possess non-trivial long-term correlations (e.g., processes with long memory), or are exponentially large (e.g., high-dimensional input spaces due to high-resolution images, or many agents). Figure 1.1 shows two example multi-agent settings: professional basketball and synthetic predator-prey games.

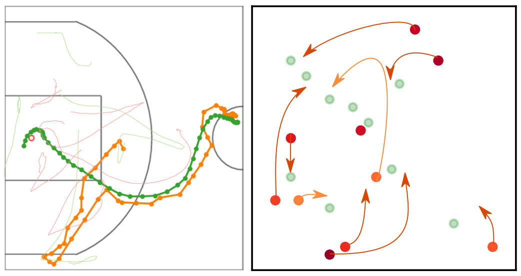


Figure 1.1: Visualization of data from multi-agent environments, Left: basketball tracking data. A player (green) has possession of the ball (orange) and passes the ball to team-mates (faint green) while running to the basket (center-left). Defenders are faint red. Right: synthetic predator-prey game. Predators are orange-red, prey are green. The predators' goal is to capture all the prey, arrows indicate the prey that each predator aims to capture.

Specifically, Figure 1.1 shows an example of multi-agent tracking data from professional NBA players (Zheng, Yue, and Lucey, 2016). Here, a human player (agent) interacts with the ball, his teammates and opponents in an environment with complex spatiotemporal dynamics (ball and agent movements) that are a result of the game rules, the physics, and decision-making of the agents.

A key characteristic of learning a good decision-making model (*policy*) in such domains, is that models with good performance often require exponentially many samples to learn. In particular, we will focus on *deep learning* which focuses on *deep neural networks*, a very powerful class of functions that have seen great success in many domains. Hence, the first contribution of this thesis is to improve the sample efficiency of deep neural networks in complex spatiotemporal problems.

A second focus of this thesis is *robustness*: complex policies, such as deep neural networks, are often unstable when the input space is very high-dimensional, e.g., for image, video and audio input. Such instability is caused by *adversarial examples* (Christian Szegedy et al., 2013), e.g., an adversary can add (imperceptible) visual perturbations that fool neural networks. Figure 1.2 shows a (strong) adversarial example: adversarial perturbations can change the neural network's correct output on the clean image, to an incorrect output on the altered image.

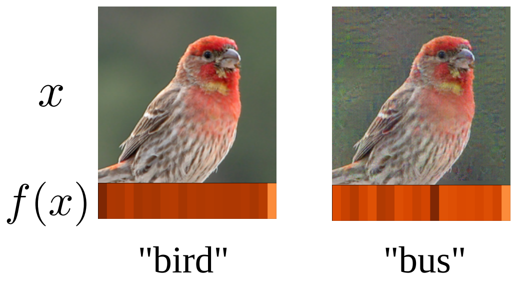


Figure 1.2: Adversarial examples can a state-of-the-art neural network $f(x)$ to misclassify images: the image x on the left is classified as a "bird", while the adversarial image on the right is classified as "bus".

In this thesis, we focus on 1) "weak" adversarial examples: a wide class of natural perturbations that can cause neural networks to be unstable, such as JPEG image compression, and 2) "strong" adversarial examples: imperceptible perturbations as in Figure 1.2. The main contribution of this thesis in this problem setting are techniques to stabilize neural networks and detect adversatial examples.

In sum, the central challenge we address in this work is to develop machine learning methods that can *learn efficiently* (e.g., requiring few data samples) and yield *expressive and robust* decision-making models that are *optimal*. Here, "expressive" models are those that are applicable in complex environments and “robust” models are those that are stable under (adversarial) input or environment noise. Informally, optimality of an agent’s decision-making model is defined with respect to a metric, such as maximizing the score that the agent can achieve during a game under the model.

We next present a succinct and more formal definition of the sequential decision-making setting and several learning approaches and models that play a central role in this thesis.

1.1 Sequential Decision-Making in a Markov Decision Process

Formally, such sequential decision-making problems can be modeled as a *Markov Decision Process* (MDP), which is the common setting for this work. In an MDP, the interaction between agent and environment can be decomposed into a cycle of three phases. At each time t, the agent executes:

- Perception: the agent observes the state s_t of itself and the environment, and any environment feedback, such as a scalar reward r,
- Decision: the agent uses a *policy* model π to determine its next action,
- Transition: the agent and environment transitions to the next state s_{t+1} under dynamics $p\left(s_{t+1} \mid s_t, a_t\right)$.

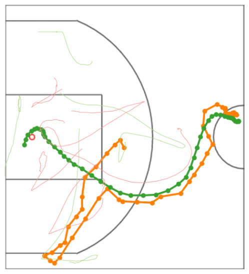


Figure 1.3: Visualization of a Markov Decision Process. An agent interacts with its environment by iteratively observing a state s, executing an action a and receiving a reward r.

Formally, this is captured by the tuple:

$$(S, A, r, p), \tag{1.1}$$

where S is the space of states, A is the space of actions, r is the environment reward, and p is the environment transition function (dynamics). In this work, we typically work with both discrete or continuous state-action spaces. This formulation admits many generalizations that also model e.g., partial observability of the state (e.g., POMDPs (Braziunas, 2003; Shani, Pineau, and Kaplow, 2013; Ross, Pineau, et al., 2008)) or multi-agent generalizations (Dec-POMDPs (Oliehoek, 2012)).

The learning problem now is to learn an *optimal policy*, which is a map $\pi : s \mapsto a$, that "solves" the MDP. An optimal policy is one that maximizes the total expected reward:

$$\max_{\pi} \mathbb{E}\left[\sum_t r_t \mid a_t \sim \pi, s_{t+1} \sim p\left(s_{t+1} \mid s_t, a_t\right)\right]. \tag{1.2}$$

In this work, we will only consider parametric policies $\pi\left(s; \theta\right)$, which are conditional distributions over actions $p(a \mid s)$. Typically, we will assume that the environment transitions p is not perfectly known.

There are two general paradigms to find the optimal policy: learning from interaction with a simulator (**"reinforcement learning"**) and learning interactively from expert data (**"imitation learning"**).

Imitation Learning

In this setting, the agent learns the optimal policy from a fixed dataset of expert demonstrations, and / or can dynamically query experts that can provide demonstrations (Ross, Gordon, and Bagnell, 2011). The former setting is also known as *behavioral cloning*. Here, the experts are humans or other models that are assumed to have good performance in the MDP, i.e., achieve high reward.

A key challenge in imitation learning is *error propagation due to model drift from the data*. In an MDP, in a sequence of states and actions, the states are not iid samples. Rather, the next state is determined via the actions of the agent and the environment dynamics. Hence, if a learned policy does not exactly match the expert policy and the data coverage of the state space is low, the next state is *very likely not in the data*. Hence, an imperfect model that does not generalize well can cause compounding errors by executing suboptimal actions in the unfamiliar states. This causes the learned policy to drift far from the data, which is typical in the imitation learning setting.

In this thesis, we will focus on how to learn expressive policy models in the setting of *behavioral cloning*, with relative small amounts of data. We are specifically interested in learning policy models that can generalize well to *long time horizons*, where error propagation becomes a particularly salient issue. We show in Chapter 2 how to train *hierarchical policy models* that learn long-term planning structure from expert data.

Reinforcement Learning

In the reinforcement learning setting, the agent can freely interact with the environment and collects its own training data by executing policies and observing the empirical state transitions and reward. It then uses this experience to perform *credit assignment*: the agent has to learn which sequences of states and actions lead to high reward. A key challenge in this setting is the exploration-exploitation tradeoff: during training the agent can sample actions from a (fixed) distribution ("exploration"), or execute actions according to its best known policy ("exploitation"). How to balance between these two modes strongly influences whether the agent can discover states with high reward, the statistical properties of the collected trainingd data, and how accurate future reward estimates are. In effect, reinforcement learning is often a highly challenging learning setting. For an extensive introduction into reinforcement learning, see Sutton and Barto, 1998.

In this thesis, we study domains where the action space is high-dimensional, such as in multi-agent settings (e.g., the full action space is the product space of the individual agent action spaces, which is exponentially large). In Chapter 4, we describe an approach to *structured exploration*, which improves the sample efficiency of reinforcement learning in this setting by learning a low-dimensional coordinated policy subspace that contains the optimal multi-agent policy.

1.2 Deep Learning

In this work, we primarily focus on learning models $f : x \mapsto y$, such as policies π, that are instantiated using *deep neural networks*. Recently, the field of *deep learning* that develops and applies these models has grown significantly in scope and innovation. Deep neural networks have transformed computer vision (Russakovsky et al., 2015), speech recognition (Amodei et al., 2016), speech generation (Oord et al., 2016; Wang et al., 2017), machine translation (Wu et al., 2016), and many other areas. Their effectiveness stems from their typically large model capacity, while still being trainable using various forms of stochastic training algorithms, e.g., stochastic gradient descent.

Two classes of neural networks that are particularly relevant in this thesis are *deep sequence models* and *deep probabilistic models*, which we will survey below. For an extended introduction into the subject, see I. Goodfellow, Y. Bengio, and Courville, 2016.

Sequence Modeling with Recurrent Neural Networks

For sequential prediction problems, a general class of neural networks are *recurrent neural networks (RNN)*. RNNs model functions $f : x_{0:T} \mapsto y_{0:T'}$ mapping sequences $x_{0:T}$ to sequences $y_{0:T'}$ with a hidden state h_t that summarizes the information in the first $t-1$ timesteps:

$$h_t = \varphi(x_t, h_{t-1}), \quad y_t = \psi(h_t), \tag{1.3}$$

where φ, ψ are nonlinear functions, such as the ReLU ($\max(0, x)$) or tanh-functions. For instance, a standard RNN with tanh-activation computes the next hidden state using a linear map of input and previous hidden state:

$$\varphi(x_t, h_{t-1}) = \tanh\left(Ux_t + Vh_t + b\right). \tag{1.4}$$

Such standard RNNs with simple output distributions often struggle to capture complex relationships in highly variable and structured sequential data, such as long-term dependencies. More complex alternative RNNs exist, such as LSTMs (Hochreiter and Schmidhuber, 1997) or GRUs (Cho et al., 2014), although these often still do not fully capture complex sequential dependencies.

Learning Deep Probabilistic Models using Implicit Optimization

Another key application of neural networks is to model complex probability distributions ("generative models") $p(x)$ over the data x. This is a challenging problem, as 1) sampling the full distribution is often intractable for real data, 2) $p(x)$ often has complicated structure and 3) real data often has many nuisance or noise factors.

One approach that will recur in this thesis is to learn *latent factor models* that decompose

$$p(x) = \int d\lambda p(x, \lambda) \tag{1.5}$$

over high-dimensional data x using latent variables λ. The idea is that the λ can learn to parametrize a low-dimensional structure in the data x. For instance, x could be an image or a sequence of actions $a_{0:T}$, and λ could parametrize the different classes of objects or actions.

However, for flexible distributions $p(x)$ (i.e., non-Gaussian) and a fixed $p(\lambda)$, learning the exact distribution $p(x \mid \lambda)$ is intractable as computing integrals over λ becomes intractable. An end-to-end learning solution is to use *implicit optimization*. A salient example is the variational autoencoder (VAE) (Kingma and Welling, 2014), which introduces an inference network $q_\phi(\lambda \mid x)$ parametrized by ϕ to approximate the true posterior $p(\lambda \mid x)$, and a learned generation model $p_\theta(x \mid \lambda)$. The inference model $q_\phi(\lambda \mid x)$ and generative model $p_\theta(x \mid \lambda)$ are commonly implemented with neural networks.

The key idea is that q_ϕ can be learned via back-propagation by constructing a differentiable stack consisting of the inference network q_ϕ, followed by a learned generative model $p_\theta(x \mid \lambda)$. The learning objective is to maximize the evidence lower-bound (ELBO) of the log-likelihood with respect to the model parameters:

$$\log p(x) \geq \mathbb{E}_{q_\phi(\lambda|x)} \left[\log p_\theta(x \mid \lambda)\right] + D_{KL} \left[q_\phi(\lambda \mid x) \mid\mid p(\lambda)\right]. \tag{1.6}$$

The first term is known as the reconstruction term and can be approximated with Monte Carlo sampling. The second term is the Kullback-Leibler divergence between the approximate posterior and the prior, and can be evaluated analytically (i.e., if both distributions are Gaussian with diagonal covariance).

As the objective 1.6 approximates (e.g., lower-bounds) the true objective, this approach is an instance of *implicit optimization*. Another recent popular approach is to learn $p(x)$ using Generative Adversarial Networks (I. Goodfellow, Pouget-Abadie, et al., 2014), although evaluating GANs for our problem setting is beyond the scope of this thesis.

1.3 Thesis Structure and Contributions

In this work, we address several shortcomings of standard deep neural networks when applied to complex sequential decision-making problems: deep policy models are typically “flat” and can be “brittle”. These limitations imply that learning policies for complex problems using imitation learning or reinforcement learning can be highly sample inefficient and lead to unstable solutions. As such, this thesis is broadly organized around two themes.

Theme 1: From Flat to Hierarchical Models

The first major theme of this work is the benefit of using models that have *hierarchical structure*, rather than being “flat”. We use hierarchy in two distinct ways: spatiotemporal hierarchy and agent-team hierarchy. In general, such approaches offer an efficient, low-dimensional representation of the data or environment that enables learning expressive policy models.

We first study structured methods in the supervised learning domain, where we assume a labeled dataset of expert agent behavior or environment dynamics is given, and our goal is to learn a policy or dynamics models from that data. In Chapter 2 we demonstrate how modeling spatiotemporal hierarchies leads to realistic basketball player behavior models implemented as hierarchical neural networks. Hence, we model temporal hierarchies, e.g., two networks predict both short-term actions and long-term goals, as well as spatial hierarchies: goals are sub-regions of the basketball court, and can be seen as subsets of the raw state space (e.g., the positions of all the players). We showcase such hierarchical policy networks first for a single-agent policy, and also for a central controller for 5 agents on offense.

In Chapter 3, we present Tensor-Train Recurrent Neural Networks, powerful sequential prediction models that can capture higher-order structure in environment dynamics, e.g., multiplicative interactions. Such dynamics give rise to non-trivial short-term (e.g., highly oscillatory) and long-term (e.g., chaos) behavior, which is characteristic of many time-series data, such climate and traffic network data.

Next, we turn to the reinforcement learning setting, where instead of a fixed dataset, the learning agent has access to a simulator, in which it has to collect training data using exploration (i.e., iterative trial-and-error). We show in Chapter 4 that learning the low-dimensional coordination patterns between many agents leads to an effective dimension reduction which enables much more efficient exploration, as the collective learns to only explore well-coordinated actions.

Theme 2: Towards Robust Deep Learning

The second theme in this thesis, is how to make neural networks more robust against various forms of input perturbations, which is a significant challenge when the input space X is high-dimensional, e.g., natural images or multi-agent state space. (I. J. Goodfellow, Shlens, and C. Szegedy, 2014; Christian Szegedy et al., 2013) have shown in the computer vision domain that a wide class of imperceptible small perturbations can significantly change the output of prediction models, making them vulnerable to malicious attackers that craft *adversarial examples* that deliberately confuse models such as neural networks. Similar examples of adversarial attacks have been demonstrated in other domains, e.g., natural language processing (Jia and Liang, 2017).

In Chapter 5, we demonstrate two methods that improve the robustness of neural networks against such input perturbations. First, in Section 5.2 we demonstrate

Stability Training, a method to stabilize network outputs against weak adversarial perturbations, such as image compression artefacts. Second, in Section 5.5 we showcase *NeuralFingerprinting*, a method to detect strong adversarial examples, which change model outputs by only changing the model input by small, imperceptible amounts.

In Chapter 6 we conclude with interesting directions for future research.

Chapter 2

LONG-TERM PLANNING VIA HIERARCHICAL POLICY NETWORKS

Zheng, Stephan, Yisong Yue, and Patrick Lucey (2016). "Generating long-term trajectories using deep hierarchical networks". In: *Advances in Neural Information Processing Systems*. S.T.Z. participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted all experiments and user studies, and participated in the writing of the manuscript., pp. 1543–1551. URL: https://papers.nips.cc/paper/6520-generating-long-term-trajectories-using-deep-hierarchical-networks.

Zhan, Eric et al. (2018). "Generative Multi-Agent Behavioral Cloning". In: *arXiv preprint arXiv:1803.07612*. S.T.Z. participated in the conception of the project and formulation of the method, analyzed the method and experimental results, and participated in the writing of the manuscript.

Summary We study the problem of modeling spatiotemporal trajectories over long time horizons using expert demonstrations. For instance, in sports, agents often choose action sequences with long-term goals in mind, such as achieving a certain strategic position. Conventional policy models are often "flat" and can only model short-term decision-making, and hence fail at efficiently learning policies that scale well to problems with long time horizons. We instead propose a hierarchical policy class that automatically reasons about both long-term and short-term goals, which we instantiate as a hierarchical neural network. We showcase our approach in case studies for both single-agent and multi-agent behavioral cloning, i.e., learning to imitate demonstrated basketball trajectories, and show that it generates significantly more realistic trajectories compared to non-hierarchical baselines as judged by professional sports analysts.

2.1 Introduction

Modeling long-term behavior is a key challenge in many learning problems that require complex decision-making. Consider a sports player determining a movement trajectory to achieve a certain strategic position. The space of such trajectories is prohibitively large, and precludes conventional approaches, such as those based on simple Markovian dynamics.

The study of such fine-grained behavior is enabled by the ongoing explosion of recorded tracking data. Beyond sports Miller et al., 2014; Yue et al., 2014b; Zheng, Yue, and Lucey, 2016; Le et al., 2017, examples include video games Ross, Gordon, and Bagnell, 2011, video & motion capture Suwajanakorn, Seitz, and Kemelmacher-Shlizerman, 2017; Taylor et al., 2017; Xue et al., 2016, navigation & driving Ziebart et al., 2009; J. Zhang and Cho, 2017; Li, Song, and Ermon, 2017, laboratory animal behaviors Johnson et al., 2016; Eyjolfsdottir et al., 2017, and tele-operated robotics Abbeel and Ng, 2004; Lin et al., 2006. One popular research direction, which we also study, is to learn a policy that imitates demonstrated behavior, also known as imitation learning Abbeel and Ng, 2004; Ziebart et al., 2008; Daumé, Langford, and Marcu, 2009; Ross, Gordon, and Bagnell, 2011; Ho and Ermon, 2016. The arguably simplest form of imitation learning is known as behavioral cloning, where the goal is to mimic a batch of pre-collected demonstration data, e.g., from human experts.

Many decision problems can be naturally modeled as requiring high-level, long-term *macro-goals*, which span time horizons much longer than the timescale of low-level *micro-actions* (cf. (He, Brunskill, and Roy, 2010; Hausknecht and Stone, 2016)). A natural example for such macro-micro behavior occurs in spatiotemporal games, such as basketball where players execute complex trajectories. The micro-actions of each agent are to move around the court and, if they have the ball, dribble, pass, or shoot the ball. These micro-actions operate at the centisecond scale, whereas their macro-goals, such as "maneuver behind these 2 defenders towards the basket", span multiple seconds. Figure 2.1 depicts an example from a professional basketball game, where the player must make a sequence of movements (micro-actions) in order to reach a specific location on the basketball court (macro-goal).

Intuitively, agents need to trade-off between short-term and long-term behavior: often sequences of individually reasonable micro-actions do not form a cohesive trajectory towards a macro-goal. For instance, in Figure 2.1 the player (green) takes a highly non-linear trajectory towards his macro-goal of positioning near the basket. As such, conventional approaches are not well suited for these settings, as they generally use a single (low-level) state-action policy, which is only successful when myopic or short-term decision-making leads to the desired behavior.

In this chapter, we propose a novel class of *hierarchical policy models*, which we instantiate using recurrent neural networks, that can simultaneously reason about

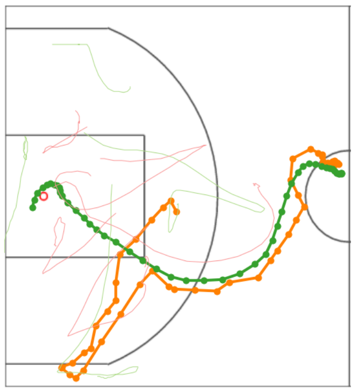


Figure 2.1: The player (green) has two macro-goals: 1) pass the ball (orange) and 2) move to the basket.

both macro-goals and micro-actions. Our model utilizes an attention mechanism through which the macro-policy guides the micro-policy. Our model is further distinguished from previous work on hierarchical policies by dynamically predicting macro-goals instead of following fixed goals, which gives additional flexibility to our model class that can be fitted to data (rather than having the macro-goals be specifically hand-crafted).

Single-agent case. We showcase our approach in a case study on behavioral cloning, i.e., learning to imitate demonstrated behavior in professional basketball. Our primary result is that our approach generates significantly more realistic player trajectories compared to non-hierarchical baselines, as judged by professional sports analysts. We also provide a comprehensive qualitative and quantitive analysis, e.g., showing that incorporating macro-goals can actually improve 1-step micro-action prediction accuracy.

Multi-agent case. We then generalize our hierarchical approach to the multi-agent case, and introduce a novel problem setting for behavioral cloning, where the optimal agent policy is probabilistic and multi-modal in nature, and must reason over the behavior of multiple interacting agents. Here, we introduce a hierarchical generative policy class that can use macro-goals to capture coordination between agents, as well as long-term intent. We show how to derive a principled learning approach that utilizes amortized variational inference, and that this model generates significantly more realistic long-term trajectories than conventional baselines.

2.2 Behavioral Cloning

We are interested in learning policies that can produce high quality trajectories, where quality is some global measure of the trajectory (e.g., realistic trajectories as in Figure 2.1). We first set notation:

- At time t, an agent i is in state $s_t^i \in S$ and takes action $a_t^i \in A$. The full state and action are $s_t = \left\{s_t^i\right\}_{\text{players } i}$, $a_t = \left\{a_t^i\right\}_{\text{players } i}$. The history of events is $\tau = \{(s_u, a_u)\}_{0 \leq u < t}$.
- Let $e_{\leq T} = \{e_t\}_{1 \leq t \leq T}$ denote a demonstration, where

$$e_t = (\mathbf{s}_t, \mathbf{a}_t) = (\{\mathbf{s}_t^k\}_{\text{agents } k}, \{\mathbf{a}_t^k\}_{\text{agents} k}). \quad \mathbf{s}_t^k \in S, \mathbf{a}_t^k \in A \tag{2.1}$$

  are the state, action of agent k at time t.
- Let π_E denote the (multi-agent) expert stochastic policy that generated the data $\mathcal{D}$, and $e_{\leq T} \sim \pi_E$ to denote that $e_{\leq T}$ was generated from policy π_E.
- Macro policies also use a goal space G, e.g., regions in the court that a player should reach.
- Let $\pi(s_t, \tau)$ denote a policy that maps state and history to a distribution over actions $P(a_t | s_t, \tau)$. If π is deterministic, the distribution is peaked around a specific action. We also abuse notation to sometimes refer to π as deterministically taking the most probable action $\pi(s_t, \tau) = \text{argmax}_{a \in A} P(a | s_t, \tau)$ – this usage should be clear from context.
- Let $\mathcal{M}(\mathbf{s}_t, \mathbf{a}_t)$ denote a (possibly probabilistic) transition function on states: $\mathbf{s}_{t+1} \sim p_{\mathcal{M}}(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$.

Learning Objective

The goal in behavioral cloning is to find a policy that behaves like the expert demonstrations $\mathcal{D}$, e.g., by solving an optimization problem with respect to a loss function ℓ:

$$\begin{aligned} \theta^* &= \text{argmin}_\theta \, \mathbb{E}_{e_{\leq T} \sim \pi_E} \left[\sum_{t=1}^{T} \ell\big(\mathbf{a}_t, \pi_\theta(\mathbf{s}_t, \tau_{t-1})\big) \right] \\ &\approx \text{argmin}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \ell\big(\mathbf{a}_t, \pi_\theta(\mathbf{s}_t, \tau_{t-1})\big). \end{aligned} \tag{2.2}$$

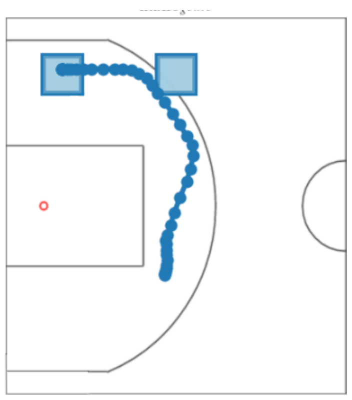


Figure 2.2: Depicting two macro-goals (blue boxes) as an agent moves to the top left.

2.3 Long-Term Trajectory Planning: Single-Agent Policy

Our main research question is how to design a policy class that can capture the salient properties of how expert agents execute trajectories. In particular, we present a general policy class that utilizes a goal space G to guide its actions to create such trajectory histories. We show in Section 2.4 how to instantiate this policy class as a hierarchical network that uses an attention mechanism to combine macro-goals and micro-actions. In our case study on modeling basketball behavior (Section 2.5), we train such a policy to imitate expert demonstrations using a large dataset of tracked basketball games.

Incorporating Macro-Goals

Our main modeling assumption is that a policy should *simultaneously optimize behavior hierarchically on multiple well-separated timescales*. We consider two distinct timescales (*macro* and *micro*-level), although our approach could in principle be generalized to even more timescales. During an episode $[t_0, t_1]$, an agent i executes a sequence of micro-actions $\left(a_t^i\right)_{t\geq 0}$ that leads to a macro-goal $g_t^i \in G$. We do not assume that the start and end times of an episode are fixed. For instance, macro-goals can change before they are reached. We finally assume that macro-goals are relatively static on the timescale of the micro-actions, that is: $dg_t^i/dt \ll 1$.

Figure 2.2 depicts an example of an agent with two unique macro-goals over a 50-frame trajectory. At every timestep t, the agent executes a micro-action a_t^i, while the macro-goals g_t^i change more slowly.

We model the interaction between a micro-action a_t^i and a macro-goal g_t^i through a raw micro-action $u_t^i \in A$ that is independent of the macro-goal. The micro-policy is

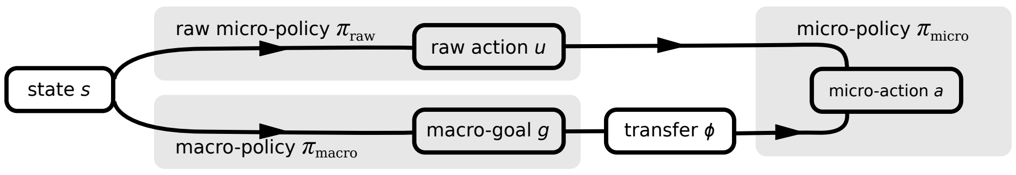

raw micro-policy π_{raw}
raw action u
micro-policy π_{micro}
state s
micro-action a
macro-goal g
transfer ϕ
macro-policy π_{macro}

Figure 2.3: The general structure of a 2-level hierarchical policy that consists of 1) a raw micro-policy π_{raw} 2) a macro-policy π_{macro} and 3) a transfer function ϕ. For clarity, we suppressed the indices i, t in the image. The raw micro-policy learns optimal short-term policies, while the macro-policy is optimized to achieve long-term rewards. The macro-policy outputs a macro-goal $g_t^i = \pi_{\text{macro}}(s_t^i, h_t^i)$, which guides the raw micro-policy $u_t^i = \pi_{\text{raw}}(s_t^i, h_t^i)$ in order for the hierarchical policy π_{micro} to achieve a long-term goal g_t^i. The hierarchical policy $\pi_{\text{micro}} = \psi(u_t^i, \phi(g_t^i))$ uses a transfer function ϕ and synthesis functon ψ, see (2.5) and Section 2.4.

then defined as:

$$a_t^i = \pi_{\text{micro}}(s_t, \tau) = \text{argmax}_a P^{\text{micro}}(a|s_t, \tau) \tag{2.3}$$

$$P^{\text{micro}}(a_t^i|s_t, \tau) = \int dudg P(a_t^i|u, g, s_t, \tau) P(u, g|s_t, \tau). \tag{2.4}$$

Here, we model the conditional distribution $P(a_t^i|u, g, s_t, \tau)$ as a non-linear function of u, g:

$$P(a_t^i|u_t^i, g_t^i, s_t, \tau) = \psi(u_t^i, \phi(g_t^i)), \tag{2.5}$$

where ϕ, ψ are transfer and synthesis functions respectively that we make explicit in Section 2.4. Note that (2.5) does not explicitly depend on s_t, τ: although it is straightforward to generalize, this did not make a significant difference in our experiments. This decomposition is shown in Figure 2.3 and can be generalized to multiple scales l using multiple macro-goals g^l and transfer functions ϕ^l.

2.4 Hierarchical Policy Network

Figure 2.3 depicts a high-level overview of our hierarchical policy class for generating long-term spatiotemporal trajectories. Both the raw micro-policy and macro-policy are instantiated as recurrent convolutional neural networks, and the raw action and macro-goals are combined via an attention mechanism, which we elaborate on below.

Discretization and deep neural architecture. In general, when using continuous latent variables g, learning the model (2.3) is intractable, and one must resort to approximation methods. We choose to discretize the state-action and latent spaces. In the basketball setting, a state $s_t^i \in S$ is naturally represented as a 1-hot occupancy vector of the basketball court. We then pose goal states g_t^i as sub-regions of the court that i wants to reach, defined at a coarser resolution than S. As such, we instantiate the macro and micro-policies as convolutional recurrent neural networks, which can capture both predictive spatial patterns and non-Markovian temporal dynamics.

Attention mechanism for integrating macro-goals and micro-actions. We model (2.5) as an attention, i.e., ϕ computes a softmax density $\phi(g_t^i)$, over the *output* action space A and ψ is an element-wise (Hadamard) product. Suppressing indices i, t and s, h for clarity, the distribution (2.5) becomes

$$\phi_k(g) = \frac{\exp h_\phi(g)_k}{\sum_j \exp h_\phi(g)_j}, \quad P(a_k|u, g) \propto P^{\text{raw}}(u_k|s, h) \cdot \phi_k(g), \quad k = 1 \dots |A|, \tag{2.6}$$

where $h_\phi(g)$ is computed by a neural network that takes $P^{\text{macro}}(g)$ as an input. Intuitively, this structure captures the trade-off between the macro- and raw micro-policy. On the one hand, the raw micro-policy π_{raw} aims for short-term optimality. On the other hand, the macro-policy π_{macro} can attend via ϕ to sequences of actions that lead to a macro-goal and bias the agent towards good long-term behavior. The difference between u and $\phi(g)$ thus reflects the trade-off that the hierarchical policy learns between actions that are good for either short-term or long-term goals.

Multi-stage learning. Given a set D of sequences of state-action tuples $(s_t, \hat{a}_t)$, where the $\hat{a}_t$ are 1-hot labels (omitting the index i for clarity), the hierarchical policy network can be trained via

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_D \sum_{t=1}^T L_t(s_t, \tau, \hat{a}_t; \theta). \tag{2.7}$$

Given the hierarchical structure of our model class, we decompose the loss L_t (omitting the index t):

$$L(s, \tau, \hat{a}; \theta) = L_{\text{macro}}(s, \tau, g; \theta) + L_{\text{micro}}(s, \tau, \hat{a}; \theta) + R(\theta), \tag{2.8}$$

$$L_{\text{micro}}(s, \tau, \hat{a}; \theta) = \sum_{k=1}^A \hat{a}_k \log \left[P^{\text{raw}}(u_k|s, \tau; \theta) \cdot \phi_k(g; \theta)\right], \tag{2.9}$$

where $R_t(\theta)$ regularizes the model weights θ and k indexes A discrete action-values. Although we have ground truths $\hat{a}_t$ for the observable micro-actions, in general we may not have labels for the macro-goals g_t that induce optimal long-term planning. As such, one would have to appeal to separate solution methods to compute the posterior $P(g_t|s_t, \tau)$ which minimizes $L_{t,\text{macro}}(s_t, \tau, g_t; \theta)$.

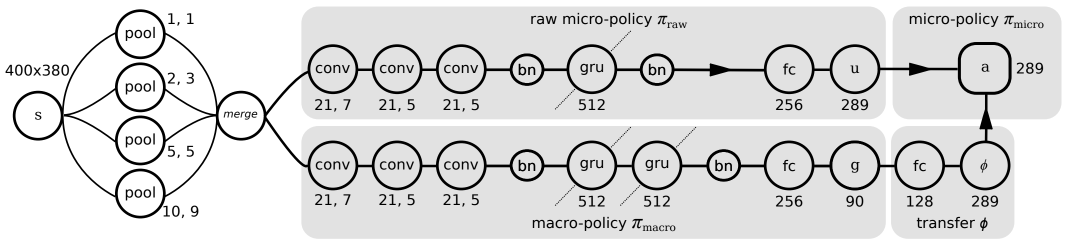


Figure 2.4: Network architecture and hyperparameters of the hierarchical policy network. For clarity, we suppressed the indices i, t in the image. Max-pooling layers (numbers indicate kernel size) with unit stride upsample the sparse tracking data s_t. The policies $\pi_{\text{raw}}, \pi_{\text{macro}}$ use a convolutional (kernel size, stride) and GRU memory (number of cells) stack to predict u_t^i and g_t^i. Batch-normalization "bn" ((Ioffe and Szegedy, 2015)) is applied to stabilize training. The output attention ϕ is implemented by 2 fully-connected layers (number of output units). Finally, the network predicts the final output $\pi_{\text{micro}}(s_t, h_t) = \pi_{\text{raw}}(s_t, h_t) \odot \phi(g_t^i)$.

To reduce complexity and given the non-convexity of (2.9), we instead follow a multi-stage learning approach with a set of *weak labels* $\hat{g}_t, \hat{\phi}_t$ for the macro-goals g_t and attention masks $\phi_t = \phi(g_t)$. We assume access to such weak labels and only use them in the initial training phases. Here, we first train the raw micro-policy, macro-policy, and attention individually, freezing the other parts of the network. The policies $\pi_{\text{micro}}, \pi_{\text{macro}}$ and attention ϕ can be trained using standard cross-entropy minimization with the labels $\hat{a}_t$, $\hat{g}_t$, and $\hat{\phi}_t$, respectively. In the final stage we fine-tune the entire network on objective (2.7), using only $L_{t,\text{micro}}$ and R. We found this approach capable of finding a good initialization for fine-tuning and generating high-quality long-term trajectories.[1] Another advantage of this approach is that the network can be trained using gradient descent during all stages.

2.5 Case Study on Modeling Basketball Behavior

We applied our approach to modeling basketball behavior data. In particular, we focus on imitating the players' movements, which is a challenging problem in the spatiotemporal planning setting.

[1]As u_t and $\phi(g_t)$ enter symmetrically into the objective (2.9), it is hypothetically possible that the network converges to a symmetric phase where the predictions u_t and $\phi(g_t)$ become identical along the entire trajectory. However, our experiments suggest that our multi-stage learning approach separates timescales well between the micro- and macro policy and prevents the network from settling in such a redundant symmetric phase.

Experimental Setup

We validated the hierarchical policy network (HPN) by learning a movement policy of individual basketball players that predicts as the micro-action the *instantaneous velocity* $v_t^i = \pi_{\text{micro}}(s_t, \tau)$.

Training data. We trained the HPN on a large dataset of tracking data from professional basketball games ((Yue et al., 2014a)). The dataset consists of *possessions* of variable length: each possession is a sequence of tracking coordinates $s_t^i = \left(x_t^i, y_t^i\right)$ for each player i, recorded at 25 Hz, where one team has continuous possession of the ball. Since possessions last between 50 and 300 frames, we sub-sampled every 4 frames and used a fixed input sequence length of 50 to make training feasible. Spatially, we discretized the left half court using 400×380 cells of size 0.25ft $\times$ 0.25ft. For simplicity, we modeled every player identically using a single policy network. The resulting input data for each possession is grouped into 4 channels: the ball, the player's location, his teammates, and the opposing team. After this pre-processing, we extracted 130,000 tracks for training and 13,000 as a holdout set.

Labels. We extracted micro-action labels $\hat{v}_t^i = s_{t+1}^i - s_t^i$ as 1-hot vectors in a grid of 17×17 unit cells. Additionally, we constructed a set of weak macro-labels $\hat{g}_t, \hat{\phi}_t$ by heuristically segmenting each track using its stationary points. The labels $\hat{g}_t$ were defined as the next stationary point. For $\hat{\phi}_t$, we used 1-hot velocity vectors $v_{t,\text{straight}}^i$ along the straight path from the player's location s_t^i to the macro-goal g_t^i.

Model hyperparameters. To generate smooth rollouts while sub-sampling every 4 frames, we simultaneously predicted the next 4 micro-actions $a_t, \ldots, a_{t+3}$. A more general approach would model the dependency between look-ahead predictions as well, e.g., $P(\pi_{t+\Delta+1}|\pi_{t+\Delta})$. However, we found that this variation did not outperform baseline models. We selected a network architecture to balance performance and feasible training-time. The macro and micro-policy use GRU memory cells (Chung, Gülçehre, et al., 2015) and a memory-less 2-layer fully-connected network as the transfer function ϕ, as depicted in Figure 2.4.

Baselines. We compared the HPN against two natural baselines.

1. A policy with only a raw micro-policy and memory (GRU-CNN) and without memory (CNN).

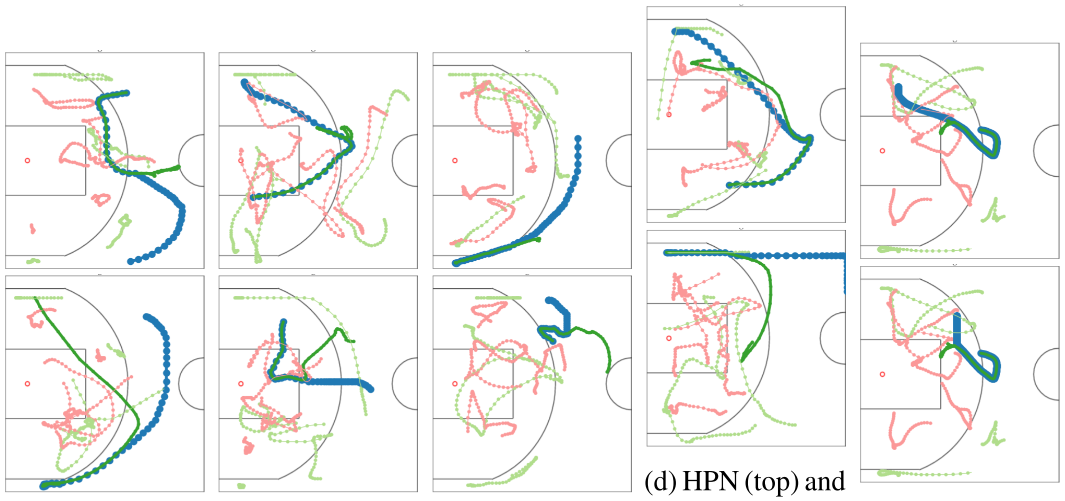

(a) HPN rollouts (b) HPN rollouts (c) HPN rollouts (d) HPN (top) and failure case (bottom) (e) HPN (top), baseline (bottom)

Figure 2.5: Rollouts generated by the HPN (columns a, b, c, d) and baselines (column e). Each frame shows an offensive player (dark green), a rollout (blue) track that extrapolates after 20 frames, the offensive team (light green) and defenders (red). Note we do not show the ball, as we did not use semantic basketball features (i.e "currently has the ball") during training. The HPN rollouts do not memorize training tracks (column a) and display a variety of natural behavior, such as curving, moving towards macro-goals and making sharp turns (c, bottom). We also show a failure case (d, bottom), where the HPN behaves unnaturally by moving along a straight line off the right side of the court – which may be fixable by adding semantic game state information. For comparison, a hierarchical baseline without an attention model produces a straight-line rollout (column e, bottom), whereas the HPN produces a more natural movement curve (column e, top).

2. A hierarchical policy network H-GRU-CNN-CC without an attention mechanism, which instead learns the final output from a concatenation of the raw micro- and macro-policy.

Rollout evaluation. To evaluate the quality of our model, we generated rollouts $(s_t; \tau)$ with *burn-in period* r_0, These are generated by 1) feeding a ground truth sequence of states $\tau = \left(s_0, \ldots, s_{r_0}\right)$ to the policy network and 2) for $t > r_0$, predicting a_t as the mode of the network output (2.3) and updating the game-state $s_t \longrightarrow s_{t+1}$, using ground truth locations for the other agents.

How Realistic are the Generated Trajectories?

The most holistic way to evaluate the trajectory rollouts is via visual analysis. Simply put, would a basketball expert find the rollouts by HPN more realistic than those by

Model comparison	Experts		Non-Experts		All	
	W/T/L	Avg Gain	W/T/L	Avg Gain	W/T/L	Avg Gain
VS-CNN	21 / 0 / 4	0.68	15 / 9 / 1	0.56	21 / 0 / 4	0.68
VS-GRU-CNN	21 / 0 / 4	0.68	18 / 2 / 5	0.52	21 / 0 / 4	0.68
VS-H-GRU-CNN-CC	22 / 0 / 3	0.76	21 / 0 / 4	0.68	21 / 0 / 4	0.68
VS-GROUND TRUTH	11 / 0 / 14	-0.12	10 / 4 / 11	-0.04	11 / 0 / 14	-0.12

Table 2.1: Preference study results. We asked basketball experts and knowledgeable non-experts to judge the relative quality of policy rollouts. We compare HPN with ground truth and 3 baselines: a memory-less (CNN) and memory-full (GRU-CNN) micro-policy and a hierarchical policy without attention (GRU-CNN -CC). For each of 25 test cases, HPN wins if more judges preferred the HPN rollout over a competitor. Average gain is the average signed vote (1 for always preferring HPN , and -1 for never preferring). We see that the HPN is preferred over all baselines (all results against baselines are significant at the 95% confidence level). Moreover, HPN is competitive with ground truth, indicating that HPN generates realistic trajectories within our rollout setting.

the baselines? We begin by first visually analyzing some rollouts, and then present our human preference study results.

Visualization. Figure 2.5 depicts example rollouts for our HPN approach and one example rollout for a baseline. Every rollout consists of two parts: 1) an initial ground truth phase from the holdout set and 2) a continuation by either the HPN or ground truth. We note a few salient properties. First, the HPN does not memorize tracks, as the rollouts differ from the tracks it has trained on. Second, the HPN generates rollouts with a high dynamic range, e.g., they have realistic curves, sudden changes of directions and move over long distances across the court towards macro-goals. For instance, HPN tracks do not move towards macro-goals in unrealistic straight lines, but often take a curved route, indicating that the policy balances moving towards macro-goals with short-term responses to the current state (see also Figures 2.5a-2.5e). In contrast, the baseline model often generates more constrained behavior, such as moving in straight lines or remaining stationary for long periods of time.

Human preference study. Our primary empirical result is a preference study eliciting judgments on the relative quality of rollout trajectories between HPN and baselines or ground truth. We recruited seven experts (professional sports analysts) and eight knowledgeable non-experts (e.g., college basketball players) as judges.

Because all the learned policies perform better with a "burn-in" period, we first animated with the ground truth for 20 frames (after subsampling), and then extrapolated with a policy for 30 frames. During extrapolation, the other nine players do not animate.[2] For each test case, the judges were shown an animation of two rollout extrapolations of a specific player's movement: one generated by the HPN, the other by a baseline or ground truth. The judges then chose which rollout looked more realistic.

Table 2.1 shows the preference study results. We tested 25 scenarios (some corresponding to scenarios in Figures 2.5a-2.5e). HPN won the vast majority of comparisons against the baselines using expert judges, with slightly weaker but still very positive results using non-expert judgments. HPN was also competitive with ground truth. These results suggest that HPN can generate high-quality player trajectories that are significant improvements over baselines, and approach the ground truth quality in this comparison setting.

Analyzing Macro- and Micro-policy Integration

Our model integrates the macro- and micro-policy by converting the macro-goal into an attention mask on the micro-action output space, which intuitively guides the micro-policy towards the macro-goal. We now inspect our macro-policy and attention mechanism to verify this behavior.

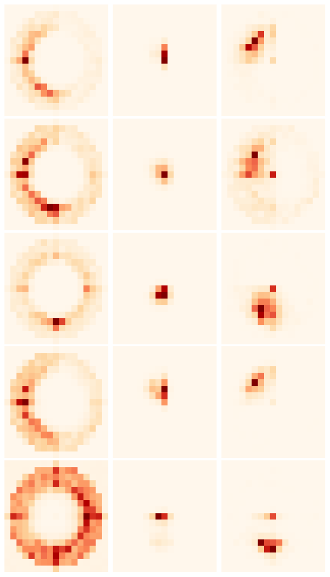
Figure 2.6a depicts how the macro-policy π_{macro} guides the micro-policy π_{micro} through the attention ϕ, by attending to the direction in which the agent can reach the predicted macro-goal. The attention model and micro-policy differ in semantic behavior: the attention favors a wider range of velocities and larger magnitudes, while the micro-policy favors smaller velocities.

Figures 2.5a-2.5e depicts predicted macro-goals by HPN along with rollout tracks. In general, we see that the rollouts are guided towards the predicted macro-goals. However, we also observe that the HPN makes some inconsistent macro-goal predictions, which suggests there is still room for improvement.

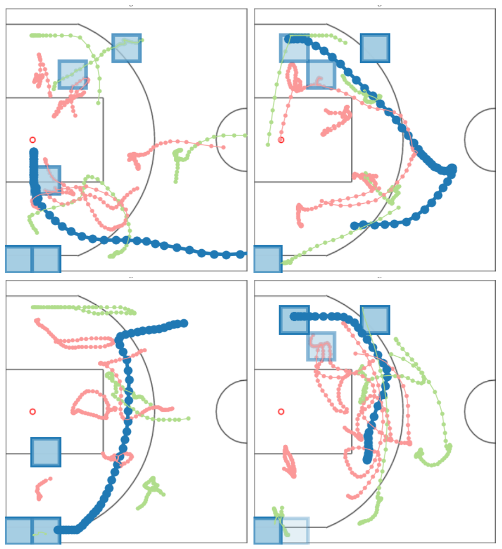
Benchmark Analysis

We finally evaluated using a number of benchmark experiments, with results shown in Table 2.2. These experiments measure quantities that are related to overall quality, albeit not holistically. We first evaluated the 4-step look-ahead accuracy of the HPN for micro-actions $a^i_{t+\Delta}, \Delta = 0, 1, 2, 3$. On this benchmark, the HPN outperforms all baseline policy networks when not using weak labels $\hat{\phi}$ to train the attention

[2] We chose this preference study design to focus the qualitative comparison on the plausibility of individual movements (e.g., how players might practice alone), as opposed to strategically coordinated team movements.



(a) Predicted distributions for attention masks $\phi(g)$ (left column), velocity (micro-action) π_{micro} (middle) and weighted velocity $\phi(g) \odot \pi_{\text{micro}}$ (right) for basketball players. The center corresponds to 0 velocity.



(b) Rollout tracks and predicted macro-goals g_t (blue boxes). The HPN starts the rollout after 20 frames. Macro-goal box intensity corresponds to relative prediction frequency during the trajectory.

Figure 2.6: **Left:** Visualizing how the attention mask generated from the macro-policy interacts with the micro-policy π_{micro}. Row 1 and 2: the micro-policy π_{micro} decides to stay stationary, but the attention ϕ goes to the left. The weighted result $\phi \odot \pi_{\text{micro}}$ is to move to the left, with a magnitude that is the average. Row 3) π_{micro} wants to go straight down, while ϕ boosts the velocity so the agent bends to the bottom-left. Row 4) π_{micro} goes straight up, while ϕ goes left: the agent goes to the top-left. Row 5) π_{micro} remains stationary, but ϕ prefers to move in any direction. As a result, the agent moves down. **Right:** The HPN dynamically predicts macro-goals and guides the micro-policy in order to reach them. The macro-goal predictions are stable over a large number of timesteps. The HPN sometimes predicts inconsistent macro-goals. For instance, in the bottom right frame, the agent moves to the top-left, but still predicts the macro-goal to be in the bottom-left sometimes.

mechanism, which suggests that using a hierarchical model can noticeably improve the short-term prediction accuracy over non-hierarchical baselines.

We also report the prediction accuracy on weak labels $\hat{g}$, $\hat{\phi}$, although they were only used during pre-training, and we did not fine-tune for accuracy on them. Using weak labels one can tune the network for both long-term and short-term planning, whereas all non-hierarchical baselines are optimized for short-term planning only. Including the weak labels $\hat{\phi}$ can lower the accuracy on short-term prediction, but increases the quality of the on-policy rollouts. This trade-off can be empirically set by tuning the number of weak labels used during pre-training.

Model	$\Delta = 0$	$\Delta = 1$	$\Delta = 2$	$\Delta = 3$	Macro-goals g	Attention ϕ
CNN	21.8%	21.5%	21.7%	21.5%	-	-
GRU-CNN	25.8%	25.0%	24.9%	24.4%	-	-
H-GRU-CNN-CC	31.5%	29.9%	29.5%	29.1%	10.1%	-
H-GRU-CNN-STACK	26.9%	25.7%	25.9%	24.9%	9.8%	-
H-GRU-CNN-ATT	**33.7%**	**31.6%**	**31.0%**	**30.5%**	10.5%	-
H-GRU-CNN-AUX	31.6%	30.7%	29.4%	28.0%	10.8%	19.2%

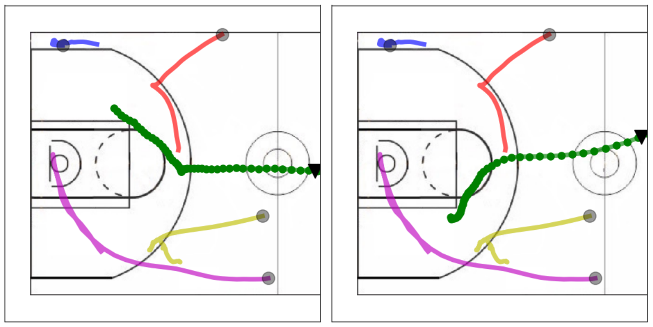
Table 2.2: Benchmark Evaluations. Δ-step look-ahead prediction accuracy for micro-actions $a^i_{t+\Delta} = \pi(s_t)$ on a holdout set, with $\Delta = 0, 1, 2, 3$. H-GRU-CNN-STACK is an HPN where predictions are organized in a feed-forward stack, with $\pi(s_t)_t$ feeding into $\pi(s_t)_{t+1}$. Using attention (H-GRU-CNN-ATT) improves on all baselines in micro-action prediction. All hierarchical models are pre-trained, but not fine-tuned, on macro-goals $\hat{g}$. We report prediction accuracy on the weak labels $\hat{g}$, $\hat{\phi}$ for hierarchical models.H-GRU-CNN-AUX is an HPN that was trained using $\hat{\phi}$. As $\hat{\phi}$ optimizes for optimal long-term behavior, this lowers the micro-action accuracy.

2.6 Long-term Trajectory Planning: Multi-Agent Policy

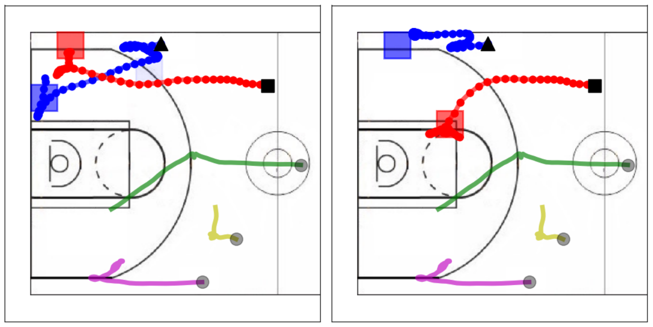
We now generalize the single-agent approach to learning a multi-agent policy. In particular, we are interested to capture the fact that expert multi-agent policies are inherently non-deterministic in many domains. Consider Figure 2.7, which depicts several scenarios of offensive player behavior in basketball. On offense, players typically behave non-deterministically and the distribution over possible trajectories is multi-modal (Figure 2.7a). Furthermore, all players display coherent team coordination over long time horizons; in Figure 2.7b, we observe that knowing each other's macro-goals (boxes that represent long-term intent) allows the red and blue players to avoid going to the same location. We aim to learn a policy that can capture all these aspects. One challenge is that the space of multi-agent trajectories is naively exponentially large. Beyond team sports, other domains include modeling social interactions in laboratory animals Eyjolfsdottir et al., 2017, team-based video games, and music generation (each instrument is an "agent") Thickstun, Harchaoui, and Kakade, 2017.

We now study the problem of *generative multi-agent behavioral cloning*, whereby the desired policy must map input states to distributions over multi-agent action spaces. We generalize the hierarchical policy class to be 1) stochastic and 2) model coordination between multiple agents. This policy class is also compatible with existing variational methods for training deep generative models, which we can adapt to the behavioral cloning setting.

While there has been some work in multi-agent imitation learning Chernova and



(a) Players have multi-modal behavior. For instance, in the above examples the green player (▼) moves to either the top or bottom.



(b) Players are coordinated. **Left**: The red player (■) moves to the top-left corner of the court. **Right**: The blue player (▲) moves to the top-left corner so the red player goes elsewhere. Knowing each other's macro-goals (boxes) is crucial for team coordination.

Figure 2.7: Player behaviors for offense in basketball (ball not shown). Black symbols indicate starting positions. An interactive demo can be found at: http://basketball-ai.com/.

Veloso, 2007; Le et al., 2017 and imitation learning with stochastic polices Ziebart et al., 2008; Ho and Ermon, 2016; Li, Song, and Ermon, 2017, no previous work has focused on learning generative polices as a core research direction, not to mention simultaneously addressing generative and multi-agent imitation learning. For instance, all the experiments in Ho and Ermon, 2016 lead to policies with highly peaked distributions, and the stochastics are essentially used to help with learning (i.e., not get stuck in local optima). In contrast, we are interested in settings where the desired behavior is a complex multi-modal distribution that reflects the non-determinism inherent in the true policy.

We showcase our approach on modeling team offense in basketball. We show that our approach can learn to generate high-quality trajectories of multi-agent gameplay. Our approach also allows for conditional inference such as grounding the macro-goal variables to manipulate agent behavior.

Simplifying assumptions. In our problem setting of human motion tracking, the transition $\mathcal{M}$ is deterministic:

$$\mathbf{s}_{t+1} = \mathbf{s}_t + \mathbf{a}_t. \tag{2.10}$$

As such, we can simplify the problem by absorbing $\mathcal{M}$ into the policy π_θ and predict $\mathbf{s}_t + \mathbf{a}_t$ directly:

- Since actions are now implicitly tied into the state, we can denote each demonstration as $\mathbf{s}_{\leq T} = \{\mathbf{s}_t\}_{1\leq t\leq T}$, and the history of states as $\tau_t = \mathbf{s}_{\leq t}$.
- Similarly, the stochastic policy $\pi_\theta(\mathbf{s}_t, \tau_{t-1}) = \pi_\theta(\tau_t)$ now samples the next state directly from $p_\theta(\mathbf{s}_{t+1}|\tau_t)$. The policy is implicitly sampling an action.

For example, the initial state $\mathbf{s}_1$ of the green player in Figure 2.7a are marked by ▼. The player's action $\mathbf{a}_1$ is to move left, which results in the next state $\mathbf{s}_2 = \mathbf{s}_1 + \mathbf{a}_1$.

Our learning objective becomes:

$$\theta^* = \text{argmin}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \ell\big(\mathbf{s}_t, \pi_\theta(\tau_{t-1})\big). \tag{2.11}$$

If the policy is deterministic, i.e., probability peaked around a single action, then we can define ℓ to be an L_2 reconstruction loss:

$$\ell\big(\mathbf{s}_t, \pi_\theta(\tau_{t-1})\big) = \|\mathbf{s}_t - \pi_\theta(\tau_{t-1})\|_2^2. \tag{2.12}$$

For a stochastic policy that returns parameters of a distribution, ℓ can be the negative log-likelihood and we can re-write the objective as a maximization problem:

$$\ell\big(\mathbf{s}_t, \pi_\theta(\tau_{t-1})\big) = -\log p_\theta\big(\mathbf{s}_t|\tau_{t-1}\big), \tag{2.13}$$

$$\theta^* = \text{argmax}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \log p_\theta\big(\mathbf{s}_t|\tau_{t-1}\big). \tag{2.14}$$

Eq. (2.14) is exactly the objective for sequential generative models that maximize the log-likelihood of data $\mathcal{D} = \{\mathbf{s}_{\leq T}\}$ by factorizing the joint distribution of the sequence:

$$\begin{aligned}\theta^* &= \text{argmax}_\theta \sum_{\mathcal{D}} \log p_\theta(\mathbf{s}_{\leq T}) \\ &= \text{argmax}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \log p_\theta(\mathbf{s}_t|\mathbf{s}_{<t}).\end{aligned} \tag{2.15}$$

As we empirically verify in Section 2.8, models trained with Eq. (2.15) have difficulty learning representations of the data that generalize well over long time horizons. Our solution is to introduce macro-goals (as seen in Figure 2.7) as an effective means in learning low-dimensional representations of the data that extend in time and space for multiple agents.

2.7 Generative Multi-Agent Policy Class

We now present our generative multi-agent policy class. Our policy class is hierarchical and incorporates macro-goals within the higher layer of the hierarchy.

We first assume conditional independence between the agent states $\mathbf{s}_t^k$ given history $\tau_{t-1} = \mathbf{s}_{<t}$. This lets us decompose the loss ℓ and policy π_θ in Eq. (2.11):

$$\begin{aligned}\theta^* &= \operatorname{argmin}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \ell\big(\mathbf{s}_t, \pi_\theta(\tau_{t-1})\big)\\ &= \operatorname{argmin}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \sum_{k=1}^{K} \ell\big(\mathbf{s}_t^k, \pi_\theta^k(\tau_{t-1})\big)\\ &= \operatorname{argmax}_\theta \sum_{\mathcal{D}} \sum_{t=1}^{T} \sum_{k=1}^{K} \log p_\theta^k(\mathbf{s}_t^k|\mathbf{s}_{<t}). \end{aligned} \tag{2.16}$$

For our experiments, we model our policies π_θ^k with VRNNs using stochastic latent variables $\mathbf{z}_t^k$ for each agent:

$$\pi_\theta^k(\tau_{t-1}) \sim p_\theta^k(\mathbf{s}_t^k|\mathbf{s}_{<t}) = \varphi^k(\mathbf{z}_t^k, \mathbf{h}_{t-1}^k), \tag{2.17}$$

$$\mathbf{h}_t^k = f^k(\mathbf{s}_t^k, \mathbf{z}_t^k, \mathbf{h}_{t-1}^k). \tag{2.18}$$

Variational RNNs.

VRNNs combine VAEs and RNNs by conditioning the VAE on a hidden state $\mathbf{h}_t$ (see Figure 2.9a):

$$p_\theta(\mathbf{z}_t|\mathbf{s}_{<t}, \mathbf{z}_{<t}) = \varphi_{\text{prior}}(\mathbf{h}_{t-1}) \qquad \text{(prior)} \tag{2.19}$$

$$q_\phi(\mathbf{z}_t|\mathbf{s}_{\le T}, \mathbf{z}_{<t}) = \varphi_{\text{enc}}(\mathbf{s}_t, \mathbf{h}_{t-1}) \qquad \text{(inference)} \tag{2.20}$$

$$p_\theta(\mathbf{s}_t|\mathbf{z}_{\le t}, \mathbf{s}_{<t}) = \varphi_{\text{dec}}(\mathbf{z}_t, \mathbf{h}_{t-1}) \qquad \text{(generation)} \tag{2.21}$$

$$\mathbf{h}_t; = f(\mathbf{s}_t, \mathbf{z}_t, \mathbf{h}_{t-1}). \qquad \text{(recurrence)} \tag{2.22}$$

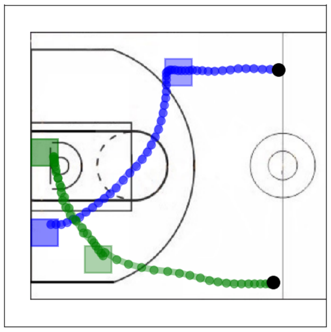


Figure 2.8: Showing macro-goals (boxes) for two players.

VRNNs are also trained by maximizing the ELBO, which in this case can be interpreted as the sum of the ELBOs over each timestep of the sequence:

$$\mathbb{E}_{q_\phi(\mathbf{z}_{\leq T}|\mathbf{s}_{\leq T})}\left[\sum_{t=1}^{T}\log p_\theta(\mathbf{s}_t \mid \mathbf{z}_{\leq T}, \mathbf{s}_{<t}) - D_{KL}\Big(q_\phi(\mathbf{z}_t \mid \mathbf{s}_{\leq T}, \mathbf{z}_{<t})||p_\theta(\mathbf{z}_t \mid \mathbf{s}_{<t}, \mathbf{z}_{<t})\Big)\right] \tag{2.23}$$

Note that the prior distribution of latent variable $\mathbf{z}_t$ depends on the history of states and latent variables (Eq. (2.19)). This temporal dependency of the prior allows VRNNs to model complex sequential data like speech and handwriting.

Hierarchical Policy with Macro-goals

Next, we introduce macro-goal variables $\mathbf{g}_t$ for policies π_θ^k:

$$\pi_\theta^k(\tau_{t-1}) \sim p_\theta(\mathbf{s}_t^k|\mathbf{s}_{<t}) = \varphi^k(\mathbf{z}_t^k, \mathbf{h}_{t-1}^k, \mathbf{g}_t). \tag{2.24}$$

The motivations behind introducing a hierarchical structure with macro-goals are 1) to provide a tractable way to capture coordination between agents, and 2) to encode long-term intents of agents and enable long-term planning at a higher-level timescale. The space of all possible combinations of trajectories for multiple agents is exponentially large. A hierarchical decomposition using macro-goals can compactly represent some low-dimension structure in the trajectory space that is easier to learn and represent.

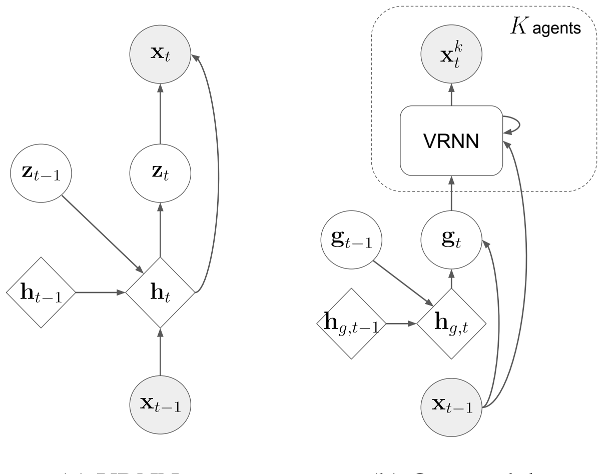
Figure 2.8 illustrates the macro-goals for two players. The blue player moves from the top-right to bottom-left of the court, while the green player moves from the bottom-right to the middle-left. At each timestep t, the players move towards their current macro-goals $\mathbf{g}_t^k$. The macro-goals change only once in 50 timesteps.

Our modeling assumptions for macro-goals are:

- subsequences of agent states $\{\mathbf{s}_t^k\}$ in an episode $[t_1, t_2]$ lead to some macro-goal $\mathbf{g}_t^k$,
- the start and end times of an episode can vary,
- macro-goals change slowly over time relative to the agent states: $d\mathbf{g}_t^k/dt \ll 1$.
- due to their reduced dimensionality, we can model (near-)arbitrary dependencies between macro-goals (e.g., coordination) via black box learning.

Capturing coordination. The last assumption motivates us to capture coordination by learning a single black-box model to predict the macro-goals of all the players. In contrast, the models in Eq. (2.17) have separate hidden states $\mathbf{h}_t^k$, so the agents are uncoordinated because their policies are independent. We induce coordination between agents by sharing the macro-goal variable $\mathbf{g}_t$ among all agents in Eq. (2.24). Intuitively, this means that all agents share a common macro-goal. In our case where the macro-goal decomposes for individual agents, each agent knows the macro-goals of all other agents. This can help the players move more cohesively (see Figure 2.7b).

Capturing long-term intent. The above assumptions also imply that macro-goals can encode long-term intent. We aim to leverage these macro-goals to ensure more consistent behavior over long horizons.



(a) VRNN (b) Our model

Figure 2.9: Computation graph of VRNN Chung, Kastner, Dinh, Goel, Aaron C. Courville, et al., 2015 and our model. Circles are stochastic variables whereas diamonds are deterministic states. Macro-goal $\mathbf{g}_t$ is shared among all agents.

Modeling Macro-goals

Ultimately, we want our model to generate macro-goals rather than depend on conditional input, so we train a policy modeled by a RNN to sample macro-goals:

$$p(\mathbf{g}_t|\mathbf{g}_{<t}) = \varphi_g(\mathbf{h}_{g,t-1}, \mathbf{s}_{t-1}), \tag{2.25}$$

$$\mathbf{h}_{g,t} = f_g(\mathbf{g}_t, \mathbf{h}_{g,t-1}). \tag{2.26}$$

We also choose to condition the macro-goal policy on the previous states $\mathbf{s}_{t-1}$ in Eq. (2.25). Then we generate trajectories by first sampling a macro-goal $\mathbf{g}_t$, and then sampling $\mathbf{s}_t^k$ conditioned on $\mathbf{g}_t$ (see Figure 2.9b for full graphical model). In addition, macro-goals that are interpretable and can be manipulated to control an agent's behavior (see Section 2.8).

In this paper, we let $\mathbf{g}_t$ be the concatenation of macro-goals $\mathbf{g}_t^k$ for each agent k. We further assume that macro-goals are available in the demonstration data, or can be easily extracted. Interesting future directions include richer macro-goals spaces and learning from less supervision.

Multi-stage Training.

Our agent and macro-goal policies can be trained independently. For our macro-goal policy, we directly maximize the log-likelihood of macro-goals $\mathbf{g}_{\leq T}$. For each of our agent policies, we maximize the ELBO for VRNNs from Eq (2.23):

$$\mathbb{E}_{q^k(\mathbf{z}^k_{\leq T}|\mathbf{s}^k_{\leq T},\mathbf{g}_{\leq T})}\Bigg[\sum_{t=1}^{T} \log p_\theta^k(\mathbf{s}_t^k|\mathbf{z}^k_{\leq T}, \mathbf{s}^k_{<t}, \mathbf{g}_{\leq T}) - D_{KL}\Big(q_\phi^k(\mathbf{z}_t^k|\mathbf{s}^k_{\leq T}, \mathbf{z}^k_{<t}, \mathbf{g}_{\leq T})\Big\|p_\theta^k(\mathbf{z}_t^k|\mathbf{s}^k_{<t}, \mathbf{z}^k_{<t}, \mathbf{g}_{<t})\Big)\Bigg]. \tag{2.27}$$

2.8 Experiments

We apply our approach to modeling team basketball gameplay on offense (team with possession of the ball). We present both quantitative and qualitative experimental results. Our quantitative results include a user study comparison with professional sports analysts, who significantly preferred rollouts by our approach to standard baselines. Our qualitative results demonstrate the ability of our approach to generate high-quality rollouts under various conditions. An interactive demo is available at http://basketball-ai.com/.

Experimental Setup

Training data. Each demonstration in our data contains trajectories of $K = 5$ players on the left half-court, recorded for $T = 50$ timesteps at 6 Hz. The offensive team has possession of the ball for the entire sequence. $\mathbf{s}_t^k$ contains 2 dimensions: the coordinates of player k at time t on the court (50×94 feet). We normalize and mean-shift the data. Players are ordered based on their relative positions, similar to the role assignment in Lucey et al., 2013. Overall, there are 107,146 training and 13,845 test examples.

For simplicity, we ignore the defensive players to focus on capturing the coordination of the offensive team. In addition, the defense is usually reactionary whereas the offense takes the initiative and will tend to have more multi-model behavior. In principle, we can provide the defensive positions as conditional input for our model and update the defensive positions using methods such as Le et al., 2017. We also ignore the ball since the dynamics of the ball are difficult to learn (e.g., oscillations indicate dribbling while straight lines indicate passing).

Weak macro-goal labels. We extract weak macro-goals labels $\hat{\mathbf{g}}_t^k$ for each player k as done in Zheng, Yue, and Lucey, 2016. We segment the left half-court into a 10×9 grid of 5ft $\times$5ft cells. The weak macro-goal $\hat{\mathbf{g}}_t^k$ at time t is a 1-hot encoding of dimension 90 of the next cell in which player k is stationary (i.e., speed $\|\hat{\mathbf{v}}_t^k\|_2 = \|\mathbf{s}_{t+1}^k - \mathbf{s}_t^k\|_2$ below a threshold). Macro-goals change slowly over time relative to player positions (see Figure 2.8). Figure 2.12 shows the distribution of extracted weak macro-goal labels for each player.

Model details. We model each latent variable $\mathbf{z}_t^k$ as a multivariate Gaussian with diagonal covariance of dimension 16 (so the KL-term in Eq. (2.27) can be computed analytically). All policies (and inference/prior functions for VRNNs, where applicable) are implemented with memory-less 2-layer fully-connected neural networks with a hidden layer of size 200. Our agent policies sample from a multivariate Gaussian with diagonal covariance while our macro-goal policies sample from a multinomial distribution over the macro-goals. All hidden states $(\mathbf{h}_{g,t}, \mathbf{h}_t^1, \ldots \mathbf{h}_t^K)$ are modeled with 200 2-layer GRU memory cells each. We maximize the log-likelihood/ELBO with stochastic gradient descent using the Adam optimizer Kingma and Ba, 2014 and a learning rate of 0.0001.[3]

[3]Code for our experiments will be available at `https://github.com/ezhan94/gen-MA-BC`.

Model	Log-Likelihood	# Parameters
RNN-gauss	1931	7,620,820
VRNN-single	$\geq$ 2302	8,523,140
VRNN-indep	$\geq$ 2360	4,367,340
Ours	$\geq$ **2362**	4,372,190

Table 2.3: We report the average log-likelihood per sequence in the test set as well as the number of trainable parameters for each model. ”$\geq$” indicates a lower bound on the log-likelihood.

Baselines. We compare our approach with 3 baselines that do not use a hierarchy of macro-goals:

1. **RNN-gauss:** RNN model Eq. (1.3) with 900 2-layer GRU cells for the hidden state.
2. **VRNN-single:** We concatenate all player positions together and use model Eq. (2.17-2.18) for $K = 1$ with 900 2-layer GRU cells for the hidden state and a 80-dimensional latent variable.
3. **VRNN-indep:** Model Eq. (2.17-2.18) with 250 2-layer GRU cells for the hidden states and 16-dimensional latent variables. We also provide the previous positions of all players as conditional input for each policy, so Eq. (2.17) becomes $p_\theta^k(\mathbf{s}_t^k | \mathbf{s}_{<t}) = \varphi^k(\mathbf{z}_t^k, \mathbf{h}_{t-1}^k, \mathbf{s}_{t-1})$.

Quantitative Evaluation

Log Likelihood. Table 2.3 reports the ELBO (log-likehood for RNN-gauss) on the test data. Our approach outperforms RNN-gauss and VRNN-single using fewer parameters and is comparable with VRNN-indep. However, higher log-likelihoods do not necessarily indicate higher quality of generated samples Theis, van den Oord, and Bethge, 2015. As such, we also conduct a human preference study to assess the relative quality of generated rollouts.

Human preference study. We recruited 14 professional sports analysts as judges to compare the quality of rollouts. Each comparison animates two rollouts, one from our model and another from a baseline. Both rollouts are burned-in for 10 timesteps with the same ground-truth states from the test set, and then generated for the next 40 timesteps. Judges decide which of the two rollouts looks more realistic.

Table 2.4 shows the results from the preference study. We tested our model against two baselines, VRNN-single and VRNN-indep, with 25 comparisons for each. All

Model Comparison	Win/Tie/Loss	Avg Gain
vs. VRNN-single	25/0/0	0.57
vs. VRNN-indep	15/4/6	0.23

Table 2.4: Preference study results. We asked 14 professional sports analysts to judge the relative quality of the generated rollouts. Judges are shown 50 comparisons that animate one rollout from our model and another from a baseline. Win/Tie/Loss indicates how often our model is preferred over baselines. Gain scores are computed by scoring +1 when our model is preferred and -1 otherwise. The average gain is computed over the total number of comparisons (25 per baseline) and judges. Our results are 98% significant using a one-sample t-test.

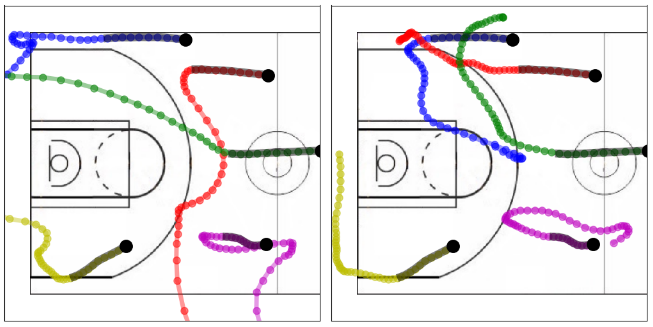
judges preferred our model over the baselines with 98% statistical significance. These results suggest that our model generates rollouts of significantly higher quality than the baselines.

Qualitative Evaluation of Generated Rollouts

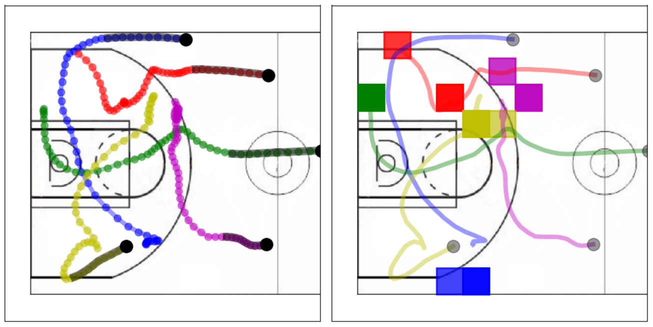
We next conduct a qualitative visual inspection of rollouts. Figure 2.10 shows rollouts generated by VRNN-single, VRNN-indep, and our model. Rollouts are generated by sampling states for 40 timesteps after an initial burn-in period of 10 timesteps with ground truth states from the test set. An interactive demo to generate more rollouts from our model can be found at: `http://basketball-ai.com/`.

Common problems in baseline rollouts (Figure 2.10a) include players moving out of bounds or in the wrong direction. These issues tend to occur at later timesteps, suggesting that the baselines do not perform well over long horizons. One possible explanation is due to compounding errors Ross, Gordon, and Bagnell, 2011: if the policy makes a mistake and deviates from the states seen during training, it will make more mistakes in the future thus leading to poor generalization.

On the other hand, generated rollouts from our model (Figure 2.10b) are more robust to the types of errors made by the baselines. Generated macro-goals also allow us to intepret the intent of each individual player as well as a global team strategy (e.g., setting up a specific formation on the court). We highlight that our model learns a multi-modal generating distribution, as repeated rollouts with the same burn-in result in a dynamic range of generated trajectories, as seen in Figure 2.11 Left. Furthermore, Figure 2.11 Right demonstrates that we can ground macro-goals manually instead of sampling them to control agent behavior.



(a) Baseline rollouts of representative quality. **Left**: VRNN-single. **Right**: VRNN-indep. Common problems in baseline rollouts include players moving out of bounds or in the wrong direction. Players do not appear to behave cohesively as a team.



(b) **Left**: Rollout from our model. All players remain in bounds. **Right**: Corresponding macro-goals for left rollout. Macro-goal generation is stable and suggests that the team is creating more space for the blue player (perhaps setting up an isolation play).

Figure 2.10: Rollouts from baselines and our model starting from black dots, generated for 40 timesteps after an initial burn-in period of 10 timesteps (marked by dark shading). An interactive demo of our model is available at: `http://basketball-ai.com/`.

Analysis of Models

Output distribution for states. The agent policies in all our models (including baselines) sample from a multivariate Gaussian with diagonal covariance. We also experimented with sampling from a mixture of 2, 3, 4, and 8 Gaussian components, but discovered that the models would always learn to assign all the weight on a single component and ignore the others. The variance of the active component is also very small. This is intuitive because sampling with a large variance at every timestep would result in noisy trajectories and not the smooth ones that we see in Figure 2.10.

Consequently, each agent's inidividual policy is effectively deterministic when

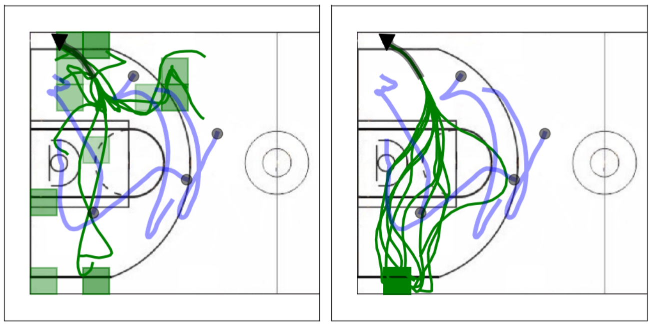


Figure 2.11: 10 rollouts of the green player (▼) overlayed on top of each other. A burn-in period of 20 timesteps is applied. Blue trajectories (●) are ground truth and black symbols indicate starting positions. **Left**: The model generates macro-goals. **Right**: We ground the macro-goals at the bottom-left. In both cases, we observe a multi-modal generating distribution of trajectories.

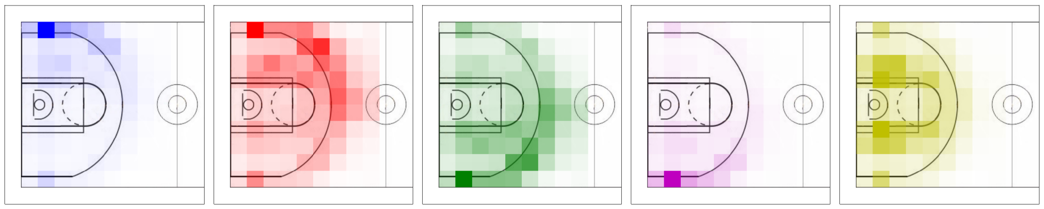


Figure 2.12: Distribution of weak macro-goal labels extracted for each player from the training data. Color intensity corresponds to frequency of macro-goal label. Players are ordered by their relative positions on the court, which can be seen from the macro-goals.

conditioned on its two inputs, one of which is deterministic (hidden state $\mathbf{h}_t^k$), while the other is stochastic (latent variable $\mathbf{z}_t^k$). So the variability of generated trajectories that we observe in Figure 2.11 must come from the only remaining source of randomness: the latent variable $\mathbf{z}_t^k$. We conclude that our model does not suffer from the Dying Units problem C. Zhang et al., 2017; Xi Chen et al., 2016, where the output policy model is sometimes sufficiently expressive on its own that the latent variables fail to encode anything meaningful.

Model for macro-goal policy. We chose to model our macro-goal policy in Eq. (2.25-2.26) with an RNN. In principle, we can also use more expressive models, like VRNNs, to model macro-goal policies over richer macro-goal spaces. In our case, we found that an RNN was sufficient in capturing the distribution of macro-goals shown in Figure 2.12. The RNN learns multinomial distributions over macro-goals that are peaked at a single macro-goal and relatively static through time, which is consistent with the behavior of macro-goals that we extracted from the data. Latent variables in a VRNN had little effect on the multinomial distribution.

Hidden state for macro-goal policy model. In our work, we defined a macro-goal $\mathbf{g}_t$ that consists of individual macro-goals $\mathbf{g}_t^k$ for each player. For our macro-goal policy, we compared a RNN model with a shared hidden state in Eq. (2.26), with a RNN model with independent hidden states, similar to Eq. (2.18). Intuitively, we expect the shared hidden state model to be better at capturing coordination.

To provide a more quantitative comparison, we computed the frequency that two or more players had the same individual macro-goal $\mathbf{g}_t^k$ at a given time, with the assumption that coordinated players do not have coinciding macro-goals very often. In the training data, 5.7% of all timesteps had coinciding macro-goals. In 10,000 rollouts from our macro-goal policy, 8.5% and 15.2% of all timesteps had coinciding macro-goals for the shared and independent hidden state models respectively. As a result, we used a RNN with a shared hidden state to model the macro-goal policy.

2.9 Related Work

Imitation Learning & Behavioral Cloning. Broadly speaking, one can decompose imitation learning along two dimensions. The first dimension spans learning to mimic batched pre-collected demonstrations, also known as behavioral cloning (Abbeel and Ng, 2004; Ziebart et al., 2008; Ho and Ermon, 2016), versus actively querying an oracle for feedback during the learning process (Daumé, Langford, and Marcu, 2009; Ross, Gordon, and Bagnell, 2011). Along this dimension, behavioral cloning is often regarded as the simplest form of imitation learning.

The second dimension spans learning value functions, also known as inverse reinforcement learning (Abbeel and Ng, 2004; Ziebart et al., 2008), versus direct policy learning (Daumé, Langford, and Marcu, 2009; Ross, Gordon, and Bagnell, 2011; Ho and Ermon, 2016). In value function learning, one assumes that there exists an unknown value function, and demonstrations are rational with respect to that value function (i.e., the demonstrated actions maximize the value). In a sense, the value function approach imposes a certain model structure, whereas direct policy learning is essentially model-free.

Our single-agent approach shares affinity with behavioral cloning. One difference with previous work is that we do not learn a reward function that induces such behavior (cf. (Muelling et al., 2014). Another related line of research aims to develop efficient policies for factored MDPs (Guestrin et al., 2003), e.g., by learning value functions over factorized state spaces for multi-agent systems. It may be possible that such approaches are also applicable for learning our hierarchical policy.

Our multi-agent work can be viewed as learning via behavioral cloning while imposing new forms of model structure. By learning via behavioral cloning, we simplify the complexity along the first dimension to focus our research on the modeling challenges that arise from learning generative multi-agent policies.

As mentioned in the introduction, there have been some prior work in multi-agent imitation learning (Chernova and Veloso, 2007; Le et al., 2017) as well as learning stochastic policies (Ziebart et al., 2008; Ho and Ermon, 2016; Li, Song, and Ermon, 2017). As also discussed before, no previous work has focused on learning generative polices as a core research direction, not to mention simultaneously addressing generative and multi-agent imitation learning. For instance all the experiments in (Ho and Ermon, 2016) led to highly peaked distributions, while (Li, Song, and Ermon, 2017) captures multi-modal distributions by assigning each demonstration to one of a fixed number of experts and learning policies for each expert that turn out to be unimodal. In contrast, we are interested in learning mappings between input states and complex multi-modal distributions over multi-agent action spaces.

Hierarchical Models. The reinforcement learning community has largely focused on non-hierarchical policies such as those based on Markovian or linear dynamics (cf. (Ziebart et al., 2008; Mnih et al., 2015; Hausknecht and Stone, 2016). By and large, such policy classes are shown to be effective only when the optimal action can be found via short-term planning. Previous research has instead focused on issues such as how to perform effective exploration, plan over parameterized action spaces, or deal with non-convexity issues from using deep neural networks. In contrast, we focus on developing hierarchical policies that can effectively generate realistic long-term plans in complex settings such as basketball gameplay.

The use of hierarchical models to decompose macro-goals from micro-actions is relatively common in the planning community (cf. (Richard S. Sutton, Precup, and Singh, 1999; He, Brunskill, and Roy, 2010; Bai, Wu, and Xiaoping Chen, 2015). For instance, the winning team in 2015 RoboCup Simulation Challenge (Bai, Wu, and Xiaoping Chen, 2015) used a manually constructed hierarchical policy to solve MDPs with a set of fixed sub-tasks, while (Konidaris et al., 2012) segmented demonstrations to construct a hierarchy of static macro-goals. In contrast, we study how one can *learn* a hierarchical policy from a large amount of expert demonstrations that can adapt its policy in non-Markovian environments with dynamic macro-goals.

Our model shares conceptual similarities to the Dual Process framework (Evans and Stanovich, 2013), which decomposes cognitive processes into fast, unconscious behavior (System 1) and slow, conscious behavior (System 2). This separation reflects our policy decomposition into a macro and micro part. Other related work in neuroscience and cognitive science include hierarchical models of learning by imitation (Byrne and Russon, 1998).

Attention. Attention models for deep networks have mainly been applied to natural language processing, image recognition and combinations thereof (Xu et al., 2015). In contrast to previous work which focuses on attention models of the input, our attention model is applied to the *output* by integrating control from both the macro-policy and the micro-policy.

Long-term planning. Another issue that our work addresses is long-term planning. In this regard, the closest prior work is Zheng, Yue, and Lucey, 2016, which also reasoned over long sequences using macro-goals. However, their approach was only for a single agent and used relatively simple stochastics. Beyond imitation learning, designing hierarchical policies is a topic of both historical and contemporary interest in reinforcement learning (Dayan and Hinton, 1993; Richard S Sutton, Precup, and Singh, 1999; Kulkarni et al., 2016). From that perspective, one can view our work as developing generative policies to capture complex non-deterministic behaviors.

Deep generative models. The study of deep generative models is an increasingly popular research area, due to their ability to inherit both the flexibility of deep learning and the probabilistic semantics of generative models. Broadly speaking, there are two ways that one can incorporate stochastics into deep models. The first approach is to model an explicit distribution over actions in the output layer, e.g., via logistic regression (L.-C. Chen et al., 2015; Oord, Dieleman, et al., 2016; Oord, Kalchbrenner, and Kavukcuoglu, 2016; Zheng, Yue, and Lucey, 2016; Eyjolfsdottir et al., 2017). The second approach is to use deep neural nets to define a transformation from a simpler distribution (e.g., unit Gaussian) to the distribution of interest (Goodfellow et al., 2014; Kingma and Welling, 2014; Rezende, Mohamed, and Wierstra, 2014). This second approach cannot explicitly specify the output distribution (i.e., one can typically only sample from the implicitly defined distribution), but can more readily be extended to incorporate additional structure, such as a hierarchy of random variables (Ranganath, Tran, and Blei, 2016) or dynamics (Johnson et al., 2016; Chung, Kastner, Dinh, Goel, Aaron C. Courville, et al., 2015; Krishnan, Shalit, and Sontag, 2017; Fraccaro et al., 2016). Our framework can incorporate both variants as appropriate.

Recent work on generative models for sequential data (Chung, Kastner, Dinh, Goel, Aaron C Courville, et al., 2015), such as handwriting generation, have combined latent variables with an RNN's hidden state to capture temporal variability in the input. In our work, we instead aim to learn semantically meaningful latent variables that are external to the RNN and reason about long-term behavior and goals.

2.10 Discussion

We have presented a hierarchical memory network for generating long-term spatiotemporal trajectories. Our approach simultaneously models macro-goals and micro-actions and integrates them using a novel attention mechanism. We demonstrated significant improvement over non-hierarchical baselines in a case study on modeling basketball player behavior.

There are several notable limitations to our HPN model. First, we did not consider all aspects of basketball gameplay, such as passing and shooting. We also modeled all players using a single policy whereas in reality player behaviors vary (although the variability can be low-dimensional (Yue et al., 2014a)). We only modeled offensive players: an interesting direction is modeling defensive players and integrating adversarial reinforcement learning (Panait and Luke, 2005) into our approach. These issues limited the scope of our preference study, and it would be interesting to consider extended settings.

In order to focus on the HPN model class, we only used the imitation learning setting. More broadly, many planning problems require collecting training data via exploration (Mnih et al., 2015), which can be more challenging. One interesting scenario is having two adversarial policies learn to be strategic against each other through repeatedly game-play in a basketball simulator. Furthermore, in general it can be difficult to acquire the appropriate weak labels to initialize the macro-policy training.

From a technical perspective, using attention in the output space may be applicable to other architectures. More sophisticated applications may require multiple levels of output attention masking.

Furthermore, we studied the problem of generative multi-agent behavioral cloning, where the goal is to learn a multi-agent policy that can tractably map input states to distributions over multi-agent action spaces. We proposed a hierarchical policy class that incorporates macro-goals to effectively reason over agent-coordination and ensure consistent behavior over long time horizons. We showcased our approach on modeling team offense in basketball, and demonstrated that our approach generates significantly higher-quality rollouts than non-hierarchical baselines.

The macro-goals used in our experiments are relatively simple. For instance, rather than simply using location-based macro-goals, we could also incorporate interactions such as “pick and roll”. Another direction for future work is to explore how to adapt our approach to different domains. For example, instead of defining macro-goals for each agent, one can imagine learning a macro-goal representing “argument” for a dialogue between two agents, or a macro-goal representing “refrain” for music generation.

Another limitation is the need for macro-goal annotations or heuristics to extract weak macro-goal labels. Although high-level annotations in many datasets is feasible, such as in music (Thickstun, Harchaoui, and Kakade, 2017), an interesting future direction is to develop algorithms to learn macro-goals in a semi-supervised or unsupervised setting, which has emerged in prior work. For example in STRAW, the agent learns to commit to a plan of future actions and is penalized for changing its plan (Vezhnevets et al., 2016). Incorporating such concepts is an interesting direction for future work.

References

Abbeel, Pieter and Andrew Y Ng (2004). “Apprenticeship learning via inverse reinforcement learning”. In: *ICML*.

Bai, Aijun, Feng Wu, and Xiaoping Chen (2015). “Online planning for large markov decision processes with hierarchical decomposition”. In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 6.4, p. 45.

Byrne, Richard W and Anne E Russon (1998). “Learning by imitation: A hierarchical approach”. In: *Behavioral and brain sciences* 21.05, pp. 667–684.

Chen, Liang-Chieh et al. (2015). “Learning deep structured models”. In: *ICML*.

Chen, Xi et al. (2016). “Variational Lossy Autoencoder”. In: *CoRR* abs/1611.02731.

Chernova, Sonia and Manuela Veloso (2007). “Multiagent collaborative task learning through imitation”. In: *International Symposium on Imitation in Animals and Artifacts*.

Chung, Junyoung, Çaglar Gülçehre, et al. (2015). “Gated Feedback Recurrent Neural Networks”. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 2067–2075.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C. Courville, et al. (2015). “A Recurrent Latent Variable Model for Sequential Data”. In: *NIPS*.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, et al. (2015). “A Recurrent Latent Variable Model for Sequential Data”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., pp. 2980–2988.

Daumé, Hal, John Langford, and Daniel Marcu (2009). “Search-based structured prediction”. In: *Machine learning* 75.3, pp. 297–325.

Dayan, Peter and Geoffrey E Hinton (1993). “Feudal reinforcement learning”. In: *NIPS*.

Evans, Jonathan St B. T. and Keith E. Stanovich (2013). “Dual-Process Theories of Higher Cognition Advancing the Debate”. en. In: *Perspectives on Psychological Science* 8.3, pp. 223–241. ISSN: 1745-6916, 1745-6924. DOI: 10.1177/1745691612460685.

Eyjolfsdottir, Eyrun et al. (2017). “Learning recurrent representations for hierarchical behavior modeling”. In: *ICLR*.

Fraccaro, Marco et al. (2016). “Sequential Neural Models with Stochastic Layers”. In: *NIPS*.

Goodfellow, Ian et al. (2014). “Generative adversarial nets”. In: *Advances in neural information processing systems*, pp. 2672–2680.

Guestrin, Carlos et al. (2003). “Efficient Solution Algorithms for Factored MDPs”. In: *J. Artif. Int. Res.* 19.1, pp. 399–468. ISSN: 1076-9757.

Hausknecht, Matthew and Peter Stone (2016). “Deep Reinforcement Learning in Parameterized Action Space”. In: *Proceedings of the International Conference on Learning Representations (ICLR)*. San Juan, Puerto Rico.

He, Ruijie, Emma Brunskill, and Nicholas Roy (2010). “PUMA: Planning Under Uncertainty with Macro-Actions”. en. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Ho, Jonathan and Stefano Ermon (2016). “Generative Adversarial Imitation Learning”. In: *NIPS*.

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: pp. 448–456.

Johnson, Matthew et al. (2016). “Composing graphical models with neural networks for structured representations and fast inference”. In: *NIPS*.

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

Kingma, Diederik P. and Max Welling (2014). “Auto-Encoding Variational Bayes”. In: *ICLR*.

Konidaris, George et al. (2012). “Robot learning from demonstration by constructing skill trees”. en. In: *The International Journal of Robotics Research* 31.3, pp. 360–375. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364911428653.

Krishnan, Rahul G., Uri Shalit, and David Sontag (2017). “Structured Inference Networks for Nonlinear State Space Models”. In: *AAAI*.

Kulkarni, Tejas D et al. (2016). “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation”. In: *NIPS*.

Le, Hoang Minh et al. (2017). “Coordinated Multi-Agent Imitation Learning”. In: *ICML*.

Li, Yunzhu, Jiaming Song, and Stefano Ermon (2017). “Infogail: Interpretable imitation learning from visual demonstrations”. In: *NIPS*.

Lin, Henry C et al. (2006). “Towards automatic skill evaluation: Detection and segmentation of robot-assisted surgical motions”. In: *Computer Aided Surgery* 11.5, pp. 220–230.

Lucey, Patrick et al. (2013). “Representing and discovering adversarial team behaviors using player roles”. In: *CVPR*.

Miller, Andrew et al. (2014). “Factorized point process intensities: A spatial analysis of professional basketball”. In: *ICML*.

Mnih, Volodymyr et al. (2015). “Human-level control through deep reinforcement learning”. en. In: *Nature* 518.7540, pp. 529–533. ISSN: 0028-0836. DOI: 10.1038/nature14236.

Muelling, Katharina et al. (2014). “Learning strategies in table tennis using inverse reinforcement learning”. ENG. In: *Biological Cybernetics* 108.5, pp. 603–619. ISSN: 1432-0770. DOI: 10.1007/s00422-014-0599-1.

Oord, Aaron van den, Sander Dieleman, et al. (2016). “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499*.

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu (2016). “Pixel recurrent neural networks”. In: *ICML*.

Panait, Liviu and Sean Luke (2005). “Cooperative multi-agent learning: The state of the art”. In: *Autonomous Agents and Multi-Agent Systems* 11.3, pp. 387–434.

Ranganath, Rajesh, Dustin Tran, and David Blei (2016). “Hierarchical variational models”. In: *ICML*.

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra (2014). “Stochastic backpropagation and approximate inference in deep generative models”. In: *ICML*.

Ross, Stéphane, Geoffrey J. Gordon, and J. Andrew Bagnell (2011). “No-Regret Reductions for Imitation Learning and Structured Prediction”. In: *AISTATS*.

Sutton, Richard S., Doina Precup, and Satinder Singh (1999). “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial Intelligence* 112.1–2, pp. 181–211. ISSN: 0004-3702. DOI: 10.1016/S0004-3702(99)00052-1.

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial intelligence* 112.1-2, pp. 181–211.

Suwajanakorn, Supasorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman (2017). “Synthesizing obama: learning lip sync from audio”. In: *ACM Transactions on Graphics (TOG)* 36.4, p. 95.

Taylor, Sarah et al. (2017). “A deep learning approach for generalized speech animation”. In: *SIGGRAPH*.

Theis, L., A. van den Oord, and M. Bethge (2015). “A note on the evaluation of generative models”. In: *ArXiv e-prints*. arXiv: 1511.01844 [stat.ML].

Thickstun, John, Zaid Harchaoui, and Sham Kakade (2017). “Learning Features of Music from Scratch”. In: *ICLR*.

Vezhnevets, Alexander et al. (2016). “Strategic Attentive Writer for Learning Macro-Actions”. In: *CoRR* abs/1606.04695.

Xu, Kelvin et al. (2015). “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”. In: *arXiv:1502.03044 [cs]*. arXiv: 1502.03044.

Xue, Tianfan et al. (2016). “Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks”. In: *NIPS*.

Yue, Yisong et al. (2014a). “Learning Fine-Grained Spatial Models for Dynamic Sports Play Prediction”. In: *IEEE International Conference on Data Mining (ICDM)*.

– (2014b). “Learning fine-grained spatial models for dynamic sports play prediction”. In: *ICDM*. IEEE.

Zhang, Cheng et al. (2017). “Advances in Variational Inference”. In: *CoRR* abs/1711.05597.

Zhang, Jiakai and Kyunghyun Cho (2017). “Query-Efficient Imitation Learning for End-to-End Simulated Driving.” In: *AAAI*.

Zheng, Stephan, Yisong Yue, and Patrick Lucey (2016). “Generating Long-term Trajectories Using Deep Hierarchical Networks”. In: *NIPS*.

Ziebart, Brian D et al. (2008). “Maximum Entropy Inverse Reinforcement Learning.” In: *AAAI*, pp. 1433–1438.

– (2009). “Human Behavior Modeling with Maximum Entropy Inverse Optimal Control.” In: *AAAI Spring Symposium: Human Behavior Modeling*, p. 92.

Chapter 3

LONG-TERM FORECASTING USING TENSOR-TRAIN RNNS

Yu, Rose et al. (2017). "Long-term forecasting using tensor-train RNNs". In: *arXiv preprint arXiv:1711.00073*. S.T.Z. participated in the formulation of the project, analyzed the method and all experiments, and participated in the writing of the manuscript.

Summary We present Tensor-Train RNN (`TT-RNN`), a novel family of neural sequence architectures for multivariate forecasting in environments with nonlinear dynamics. Long-term forecasting in such systems is highly challenging, since there exist long-term temporal dependencies, higher-order correlations, and sensitivity to error propagation. Our proposed tensor recurrent architecture addresses these issues by learning the nonlinear dynamics directly using higher order moments and high-order state transition functions. Furthermore, we decompose the higher-order structure using the tensor-train (TT) decomposition to reduce the number of parameters while preserving the model performance. We theoretically establish the approximation properties of Tensor-Train RNNs for general sequence inputs, and such guarantees are not available for usual RNNs. We also demonstrate significant long-term prediction improvements over general RNN and LSTM architectures on a range of simulated environments with nonlinear dynamics, as well on real-world climate and traffic data.

3.1 Introduction

One of the central questions in science is forecasting: given the past history, how well can we predict the future? In many domains with complex multivariate correlation structures and nonlinear dynamics, forecasting is highly challenging since the system has long-term temporal dependencies and higher-order dynamics. Examples of such systems abound in science and engineering, from biological neural network activity, fluid turbulence, to climate and traffic systems (see Figure 3.1). Since current forecasting systems are unable to faithfully represent the higher-order dynamics, they have limited ability for accurate *long-term* forecasting.

Therefore, a key challenge is accurately modeling nonlinear dynamics and obtaining stable long-term predictions, given a dataset of realizations of the dynamics. Here,

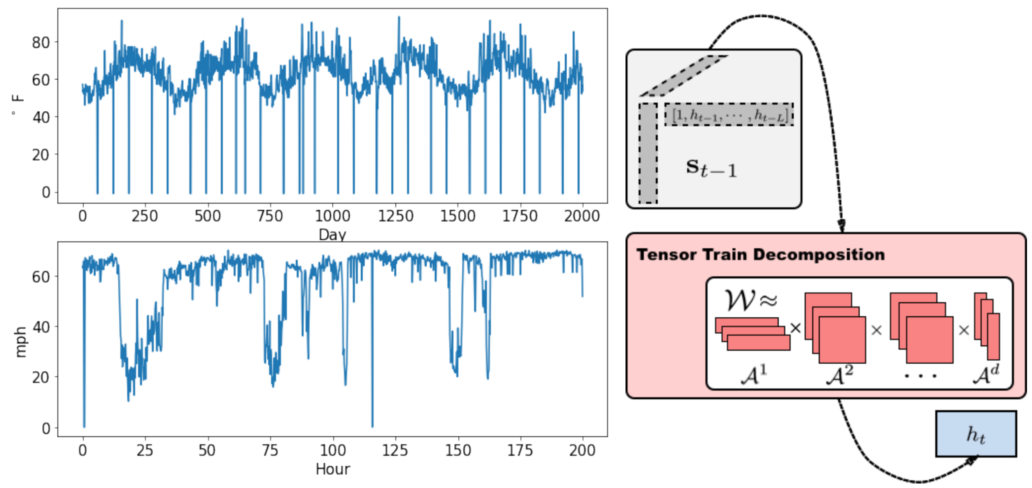


Figure 3.1: Left: climate and traffic time series per location. The time-series exhibits long-term temporal correlations, and can be viewed as a realization of highly nonlinear dynamics. Right: tensor-train RNN unit encodes high-order dynamics and factorizes hidden states with tensor train decomposition.

the forecasting problem can be stated as follows: how can we efficiently learn a model that, given only few initial states, can reliably predict a sequence of future states over a long horizon of T time-steps?

Common approaches to forecasting involve linear time series models such as auto-regressive moving average (ARMA), state space models such as hidden Markov model (HMM), and deep neural networks. We refer readers to a survey on time series forecasting by (Box et al., 2015) and the references therein. A recurrent neural network (RNN), as well as its memory-based extensions such as the LSTM, is a class of models that have achieved good performance on sequence prediction tasks from demand forecasting (Flunkert, Salinas, and Gasthaus, 2017) to speech recognition (Soltau, Liao, and Sak, 2016) and video analysis (LeCun, Bengio, and G. Hinton, 2015). Although these methods can be effective for short-term, smooth dynamics, neither analytic nor data-driven learning methods tend to generalize well to capturing long-term nonlinear dynamics and predicting them over longer time horizons.

To address this issue, we propose a novel family of tensor-train recurrent neural networks that can learn stable long-term forecasting. These models have two key features: they 1) *explicitly model the higher-order dynamics*, by using a longer history of previous hidden states and high-order state interactions with multiplicative memory units; and 2) they are scalable by using *tensor trains*, a structured low-rank tensor decomposition that greatly reduces the number of model parameters, while mostly preserving the correlation structure of the full-rank model.

In this work, we analyze Tensor-Train RNNs theoretically, and also experimentally validate them over a wide range of forecasting domains. Our contributions can be summarized as follows:

- We describe how `TT-RNNs` encode higher-order non-Markovian dynamics and high-order state interactions. To address the memory issue, we propose a tensor-train (TT) decomposition that makes learning tractable and fast.
- We provide theoretical guarantees for the representation power of `TT-RNNs` for nonlinear dynamics, and obtain the connection between the target dynamics and `TT-RNN` approximation. In contrast, no such theoretical results are known for standard recurrent networks.
- We validate `TT-RNNs` on simulated data and two real-world environments with nonlinear dynamics (climate and traffic). Here, we show that `TT-RNNs` can forecast more accurately for significantly longer time horizons compared to standard RNNs and LSTMs.

3.2 Forecasting using Tensor-Train RNNs

Forecasting Nonlinear Dynamics Our goal is to learn an efficient model f for *sequential multivariate forecasting* in environments with nonlinear dynamics. Such systems are governed by *dynamics* that describe how a system state $\mathbf{s}_t \in \mathbb{R}^d$ evolves using a set of *nonlinear* differential equations:

$$\left\{\xi^i\left(\mathbf{s}_t, \frac{d\mathbf{s}}{dt}, \frac{d^2\mathbf{s}}{dt^2}, \ldots; \phi\right) = 0\right\}_i, \tag{3.1}$$

where ξ^i can be an arbitrary (smooth) function of the state $\mathbf{s}_t$ and its derivatives. Continous time dynamics are usually described by differential equations while difference equations are employed for discrete time. In continuous time, a classic example is the first-order Lorenz attractor, whose realizations showcase the "butterfly-effect", a characteristic set of double-spiral orbits. In discrete-time, a non-trivial example is the 1-dimensional Genz dynamics, whose difference equation is:

$$s_{t+1} = \left(c^{-2} + (s_t + w)^2\right)^{-1}, \quad c, w \in [0, 1], \tag{3.2}$$

where s_t denotes the system state at time t and c, w are the parameters. Due to the nonlinear nature of the dynamics, such systems exhibit higher-order correlations, long-term dependencies and sensitivity to error propagation, and thus form a challenging setting for learning.

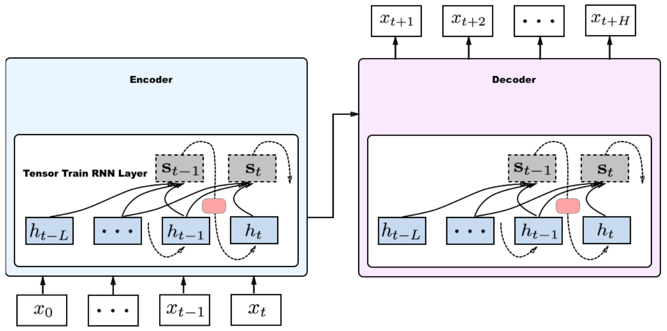


Figure 3.2: Tensor-train recurrent cells within a seq2seq model. Both encoder and decoder contain tensor-train recurrent cells. The augmented state (grey) takes in past L hidden states (blue) and forms a high-order tensor. Tensor-train cell (red) factorizes the tensor and outputs the next hidden state.

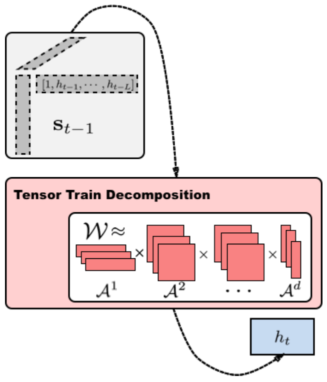


Figure 3.3: A tensor-train cell. The augmented state (grey) forms a high-order tensor. Tensor-train cell factorizes the tensor and outputs the next hidden state.

Given a sequence of initial states $\mathbf{s}_0 \ldots \mathbf{s}_t$, the forecasting problem aims to learn a model f

$$f : (\mathbf{s}_0 \ldots \mathbf{s}_t) \mapsto (\mathbf{y}_t \ldots \mathbf{y}_T), \quad \mathbf{y}_t = \mathbf{s}_{t+1}, \tag{3.3}$$

that outputs a sequence of future states $\mathbf{s}_{t+1} \ldots \mathbf{s}_T$. Hence, accurately approximating the dynamics ξ is critical to learning a good forecasting model f and accurately predicting for long time horizons.

First-order Markov Models In deep learning, common approaches for modeling dynamics usually employ first-order hidden-state models, such as recurrent neural networks (RNNs). An RNN with a single cell recursively computes a hidden state $\mathbf{h}_t$ using the most recent hidden state $\mathbf{h}_{t-1}$, generating the output $\mathbf{y}_t$ from the hidden state $\mathbf{h}_t$:

$$\mathbf{h}_t = f(\mathbf{s}_t, \mathbf{h}_{t-1}; \theta), \quad \mathbf{y}_t = g(\mathbf{h}_t; \theta), \tag{3.4}$$

where f is the state transition function, g is the output function and θ are model parameters. A common parametrization scheme for (3.4) is a nonlinear activation function applied to a linear map of $\mathbf{s}_t$ and $\mathbf{h}_{t-1}$ as:

$$\mathbf{h}_t = f(W^{hx}\mathbf{s}_t + W^{hh}\mathbf{h}_{t-1} + \mathbf{b}^h), \tag{3.5}$$

$$\mathbf{s}_{t+1} = W^{xh}\mathbf{h}_t + \mathbf{b}^x, \tag{3.6}$$

where the state transition f can be sigmoid, tanh, etc., W^{hx}, W^{xh} and W^{hh} are transition weight matrices and $\mathbf{b}^h, \mathbf{b}^x$ are biases.

RNNs have many different variations, including LSTMs (Hochreiter and Schmidhuber, 1997) and GRUs (Chung, Gulcehre, et al., 2014). For instance, LSTM cells use a memory-state, which mitigate the "exploding gradient" problem and allow RNNs to propagate information over longer time horizons. Although RNNs are very expressive, they compute the hidden state $\mathbf{h}_t$ using only the previous state $\mathbf{h}_{t-1}$ and input $\mathbf{s}_t$. Such models do not explicitly model higher-order dynamics and only implicitly model long-term dependencies between all historical states $\mathbf{h}_0 \dots \mathbf{h}_t$, which limits their forecasting effectiveness in environments with nonlinear dynamics.

Tensorized Recurrent Neural Networks

To effectively learn nonlinear dynamics, we propose Tensor-Train RNNs, or `TT-RNNs`, a class of higher-order models that can be viewed as a higher-order generalization of RNNs. We developed `TT-RNNs` with two goals in mind: explicitly modeling 1) L-order Markov processes with L steps of temporal memory and 2) polynomial interactions between the hidden states $\mathbf{h}_\cdot$ and $\mathbf{s}_t$.

First, we consider longer "history": we keep length L historic states: $\mathbf{h}_t, \cdots, \mathbf{h}_{t-L}$:

$$\mathbf{h}_t = f(\mathbf{s}_t, \mathbf{h}_{t-1}, \cdots, \mathbf{h}_{t-L}; \theta), \tag{3.7}$$

where f is an activation function. In principle, early work (Giles et al., 1989) has shown that with a large enough hidden state size, such recurrent structures are capable of approximating any dynamics.

Second, to learn the nonlinear dynamics ξ efficiently, we also use higher-order moments to approximate the state transition function. Concatenate the L-lag hidden state as an augmented state $\mathbf{H}_{t-1}$:

$$\mathbf{H}_{t-1}^T = [1 \ \mathbf{h}_{t-1}^\top \ \dots \ \mathbf{h}_{t-L}^\top].$$

For every hidden dimension, we construct a P-dimensional transition *weight tensor* by modeling a degree P polynomial interaction between hidden states:

$$[\mathbf{h}_t]_\alpha = f(W_\alpha^{hx}\mathbf{s}_t + \sum_{i_1,\cdots,i_p} \mathcal{W}_{\alpha i_1 \cdots i_P} \underbrace{\mathbf{H}_{t-1;i_1} \otimes \cdots \otimes \mathbf{H}_{t-1;i_p}}_{P}),$$

where α indices the hidden dimension, $i_\cdot$ indices the high-order terms and P is the polynomial order. We included the bias unit 1 in $\mathbf{H}_{t-1}$ to account for the first order term, so that $\mathbf{H}_{t-1;i_1} \otimes \cdots \otimes \mathbf{H}_{t-1;i_p}$ can model all possible polynomial expansions up to order P.

The `TT-RNN` with LSTM cell, or "TLSTM", is defined analogously as:

$$[\mathbf{i}_t, \mathbf{g}_t, \mathbf{f}_t, \mathbf{o}_t]_\alpha = \sigma(W_\alpha^{hx}\mathbf{s}_t + \sum_{i_1,\cdots,i_p} \mathcal{W}_{\alpha i_1 \cdots i_P} \underbrace{\mathbf{H}_{t-1;i_1} \otimes \cdots \otimes \mathbf{H}_{t-1;i_P}}_{P}), \tag{3.8}$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} \circ \mathbf{f}_t + \mathbf{i}_t \circ \mathbf{g}_t, \qquad \mathbf{h}_t = \mathbf{c}_t \circ \mathbf{o}_t,$$

where $\circ$ denotes the Hadamard product. Note that the bias units are again included.

`TT-RNN` is a basic unit that can be incorporated in most of the existing recurrent neural architectures such as convolutional RNN (Xingjian et al., 2015) and hierarchical RNN (Chung, Ahn, and Bengio, 2016). In this work, we use `TT-RNN` as a module for sequence-to-sequence (Seq2Seq) framework (Sutskever, Vinyals, and Le, 2014) in order to perform long-term forecasting.

As shown in Figure 3.2, sequence-to-sequence consists of an encoder-decoder pair. Encoder takes an input sequence and learns a hidden representation. Decoder initializes with this hidden representation and generates a output sequence. Both contains multiple layers of tensor-train recurrent cells (color coded in red). The augmented state $\mathbf{H}_{t-1}$ (color coded in grey) concatenates the past L hidden states. And the tensor-train cell takes $\mathbf{H}_{t-1}$ and outputs the next hidden state. The encoder encodes the initial states $s_0, \ldots, s_t$ and the decoder predicts $s_{t+1}, \ldots, s_T$. For each timestep t, the decoder uses its own previous prediction $\mathbf{y}_t$ as an input.

Tensor-train Networks

Unfortunately, due to the "curse of dimensionality", the number of parameters in $\mathcal{W}_\alpha$ with hidden size H grows exponentially as $O(HL^P)$, which makes the high-order model prohibitively large to train. To overcome this difficulty, we utilize *tensor networks* to approximate the weight tensor. Such networks encode a structural decomposition of tensors into low-dimensional components and have been shown to provide the most general approximation to smooth tensors (Orús, 2014). The most commonly used tensor networks are *linear tensor networks* (LTN), also known as *tensor-trains* in numerical analysis or *matrix-product states* in quantum physics (I. V. Oseledets, 2011).

A tensor train model decomposes a P-dimensional tensor $\mathcal{W}$ into a network of sparsely connected low-dimensional tensors $\{\mathcal{A}^p \in \mathbb{R}^{r_{p-1} \times n_p \times r_p}\}$ as:

$$\mathcal{W}_{i_1 \cdots i_P} = \sum_{\alpha_1 \cdots \alpha_{P-1}} \mathcal{A}^1_{\alpha_0 i_1 \alpha_1} \mathcal{A}^2_{\alpha_1 i_2 \alpha_2} \cdots \mathcal{A}^P_{\alpha_{P-1} i_P \alpha_P}$$

with $\alpha_0 = \alpha_P = 1$, as depicted in Figure (3.3). When $r_0 = r_P = 1$ the $\{r_p\}$ are called the tensor-train rank. With tensor-train, we can reduce the number of parameters of `TT-RNN` from $(HL+1)^P$ to $(HL+1)R^2P$, with $R = \max_p r_p$ as the upper bound on the tensor-train rank. Thus, a major benefit of tensor-train is that they *do not* suffer from the curse of dimensionality, which is in sharp contrast to many classical tensor decomposition models, such as the Tucker decomposition.

3.3 Approximation Results for Tensor-Train RNNs

A significant benefit of using `TT-RNN` is that we can theoretically characterize its expressiveness for approximating the underlying dynamics. The main idea is to analyze a class of functions that satisfies certain regularity conditions. For such functions, tensor-train representations preserve the weak differentiability and yield a compact representation.

The following theorem characterizes the representation power of `TT-RNN`, viewed as a one-layer hidden neural network, in terms of 1) the regularity of the target function f, 2) the dimension of the input space, 3) the tensor train rank and 4) the order of the tensor:

Theorem 1. *Let the state transition function $f \in \mathcal{H}^k_\mu$ be a Hölder continuous function defined on a input domain $\mathcal{I} = I_1 \times \cdots \times I_d$, with bounded derivatives up to order k and finite Fourier magnitude distribution C_f. A single layer `TT-RNN` with h hidden units can approximate f with approximation error ϵ at most:*

$$\epsilon \le \frac{1}{h}\left(C_f^2 \frac{d-1}{(k-1)(r+1)^{k-1}} + C(k)p^{-k}\right), \tag{3.9}$$

where $C_f = \int |\omega|_1 |\hat{f}(\omega) d\omega|$, d is the dimension of the function, i.e., the size of the state space, r is the tensor-train rank, p is the degree of the higher-order polynomials i.e., the order of the tensor, and $C(k)$ is the coefficient of the spectral expansion of f.

Remarks: The result above shows that the number of weights required to approximate the target function f is dictated by its regularity (i.e., its Hölder-continuity order k). The expressiveness of `TT-RNN` is driven by the selection of the rank r and the polynomial degree p; moreover, it improves for functions with increasing regularity. Compared with "first-order" regular RNNs, `TT-RNN`s are exponentially more powerful for large rank: if the order p increases, we require fewer hidden units h.

Proof. We provide theoretical guarantees for the proposed `TT-RNN` model by analyzing a class of functions that satisfy some regularity condition. For such functions, tensor-train decompositions preserve weak differentiability and yield a compact representation. We combine this property with neural network estimation theory to bound the approximation error for `TT-RNN` with one hidden layer, in terms of: 1) the regularity of the target function f, 2) the dimension of the input space, and 3) the tensor train rank.

In the context of `TT-RNN`, the target function $f(\mathbf{s})$ with $\mathbf{s} = \mathbf{H} \otimes \ldots \otimes \mathbf{H}$, is the system dynamics that describes state transitions, as in (3.8). Let us assume that $f(\mathbf{s})$ is a Sobolev function: $f \in \mathcal{H}^k_\mu$, defined on the input space $\mathcal{I} = I_1 \times I_2 \times \cdots I_d$, where each I_i is a set of vectors. The space $\mathcal{H}^k_\mu$ is defined as the set of functions that have bounded derivatives up to some order k and are L_μ-integrable:

$$\mathcal{H}^k_\mu = \left\{ f \in L^2_\mu(I) : \sum_{i \le k} \|D^{(i)} f\|^2 < +\infty \right\}, \tag{3.10}$$

where $D^{(i)} f$ is the i-th weak derivative of f and $\mu \ge 0$.[1]

Any Sobolev function admits a Schmidt decomposition: $f(\cdot) = \sum_{i=0}^{\infty} \sqrt{\lambda(i)} \gamma(\cdot; i) \otimes \phi(i; \cdot)$, where $\{\lambda\}$ are the eigenvalues and $\{\gamma\}, \{\phi\}$ are the associated eigenfunctions. Hence, we can decompose the target function $f \in \mathcal{H}^k_\mu$ as:

$$f(\mathbf{s}) = \sum_{\alpha_0, \cdots, \alpha_d = 1}^{\infty} \mathcal{A}^1(\alpha_0, s_1, \alpha_1) \cdots \mathcal{A}^d(\alpha_{d-1}, s_d, \alpha_d), \tag{3.11}$$

where $\{\mathcal{A}^d(\alpha_{d-1}, \cdot, \alpha_d)\}$ are basis functions

$$\{\mathcal{A}^d(\alpha_{d-1}, s_d, \alpha_d)\} = \sqrt{\lambda_{d-1}(\alpha_{d-1})} \phi(\alpha_{d-1}; s_d)\}, \tag{3.12}$$

satisfying $\langle \mathcal{A}^d(i, \cdot, m), \mathcal{A}^d(i, \cdot, m) \rangle = \delta_{mn}$. We can truncate Eqn 3.13 to a low dimensional subspace ($\mathbf{r} < \infty$), and obtain the *functional tensor-train (FTT)* approximation of the target function f:

$$f_{TT}(\mathbf{x}) = \sum_{\alpha_0, \cdots, \alpha_d = 1}^{\mathbf{r}} \mathcal{A}^1(\alpha_0, s_1, \alpha_1) \cdots \mathcal{A}^d(\alpha_{d-1}, s_d, \alpha_d) \tag{3.13}$$

.

FTT approximation in Eqn 3.13 projects the target function to a subspace with finite basis. And the approximation error can be bounded using the following Lemma:

[1] A weak derivative generalizes the derivative concept for (non)-differentiable functions and is implicitly defined as: e.g., $v \in L^1([a, b])$ is a weak derivative of $u \in L^1([a, b])$ if for all smooth φ with $\varphi(a) = \varphi(b) = 0$: $\int_a^b u(t)\varphi'(t) = -\int_a^b v(t)\varphi(t)$.

Lemma 2 (FTT Approximation Bigoni, Engsig-Karup, and Marzouk, 2016)**.** *Let $f \in \mathcal{H}_\mu^k$ be a Hölder continuous function, defined on a bounded domain $\mathbf{I} = I_1 \times \cdots \times I_d \subset \mathbb{R}^d$ with exponent $\alpha > 1/2$, the FTT approximation error can be upper bounded as*

$$\|f - f_{TT}\|^2 \leq \|f\|^2 (d-1) \frac{(r+1)^{-(k-1)}}{(k-1)} \tag{3.14}$$

for $r \geq 1$ and

$$\lim_{r \to \infty} \|f_{TT} - f\|^2 = 0 \tag{3.15}$$

for $k > 1$

Lemma 2 relates the approximation error to the dimension d, tensor-train rank r,and the regularity of the target function k. In practice, `TT-RNN` implements a polynomial expansion of the input states $\mathbf{H}$, using powers $[\mathbf{H}, \mathbf{H}^{\otimes 2}, \cdots, \mathbf{H}^{\otimes p}]$ to approximate f_{TT}, where p is the degree of the polynomial. We can further use the classic spectral approximation theory to connect the `TT-RNN` structure with the degree of the polynomial, i.e., the order of the tensor. Let $I_1 \times \cdots \times I_d = \mathbf{I} \subset \mathbb{R}^d$. Given a function f and its polynomial expansion P_{TT}, the approximation error is therefore bounded by:

Lemma 3 (Polynomial Approximation)**.** *Let $f \in \mathcal{H}_\mu^k$ for $k > 0$. Let P be the approximating polynomial with degree p, Then*

$$\|f - P_N f\| \leq C(k) p^{-k} |f|_{k,\mu}$$

Here $|f|_{k,\mu}^2 = \sum_{|i|=k} \|D^{(i)} f\|^2$ is the semi-norm of the space $\mathcal{H}_\mu^k$. $C(k)$ is the coefficient of the spectral expansion. By definition, $\mathcal{H}_\mu^k$ is equipped with a norm $\|f\|_{k,\mu}^2 = \sum_{|i| \leq k} \|D^{(i)} f\|^2$ and a semi-norm $|f|_{k,\mu}^2 = \sum_{|i|=k} \|D^{(i)} f\|^2$. For notation simplicity, we muted the subscript μ and used $\|\cdot\|$ for $\|\cdot\|_{L_\mu}$.

So far, we have obtained the tensor-train approximation error with the regularity of the target function f. Next we will connect the tensor-train approximation and the estimation error of neural networks with one layer hidden units. Given a neural network with one hidden layer and sigmoid activation function, following Lemma describes the classic result of describes the error between a target function f and the single hidden-layer neural network that approximates it best:

Lemma 4 (NN Approximation Barron, 1993)**.** *Given a function f with finite Fourier magnitude distribution C_f, there exists a neural network of n hidden units f_n, such that*

$$\|f - f_n\| \leq \frac{C_f}{\sqrt{n}} \tag{3.16}$$

where $C_f = \int |\omega|_1 |\hat{f}(\omega)| d\omega$ with Fourier representation $f(x) = \int e^{i\omega x} \hat{f}(\omega) d\omega$.

We can now generalize Barron's approximation lemma 4 to `TT-RNN`. The target function we are approximating is the state transition function $f() = f(\mathbf{H} \otimes \cdots \otimes \mathbf{H})$. We can express the function using FTT, followed by the polynomial expansion of the states concatenation P_{TT}. The approximation error of `TT-RNN`, viewed as one hidden layer, is:

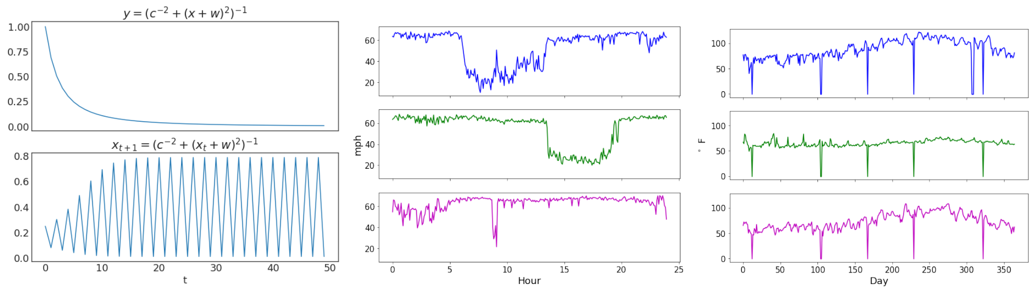
$$\begin{aligned} \|f - P_{TT}\| &\leq \|f - f_{TT}\| + \|f_{TT} - P_{TT}\| \\ &\leq \|f\| \sqrt{(d-1)\frac{(r+1)^{-(k-1)}}{(k-1)}} + C(k)p^{-k}|f_{TT}|_k \\ &\leq \|f - f_n\| \sqrt{(d-1)\frac{(r+1)^{-(k-1)}}{(k-1)}} \\ &\quad + C(k)p^{-k} \sum_{i=k} \|D^{(i)}(f_{TT} - f_n)\| + o(\|f_n\|) \\ &\leq \frac{C_f^2}{\sqrt{n}} \Big(\sqrt{(d-1)\frac{(r+1)^{-(k-1)}}{(k-1)}} + C(k)p^{-k} \sum_{i=k} \|D^{(i)} f_{TT}\| \Big) + o(\|f_n\|), \end{aligned}$$

where p is the order of tensor and r is the tensor-train rank. As the rank of the tensor-train and the polynomial order increase, the required size of the hidden units become smaller, up to a constant that depends on the regularity of the underlying dynamics f.

3.4 Experiments

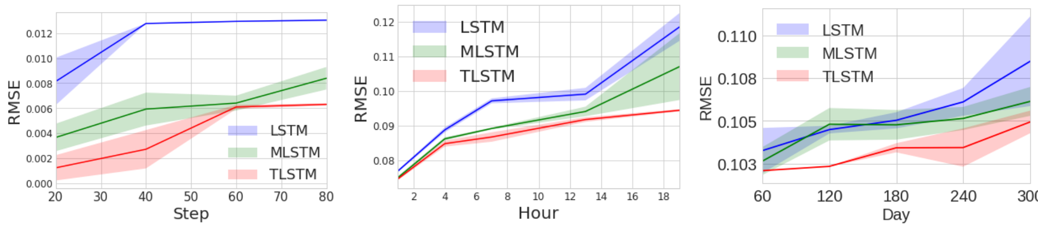
We validated the accuracy and efficiency of `TT-RNN` on one synthetic and two real-world datasets, as described below; we performed missing data imputation and used rolling window to extract input-output subsequences.

Genz dynamics The Genz "product peak" (see Figure 3.4 a) is one of the Genz functions (Genz, 1984), which are often used as a basis for high-dimensional function



(a) *Genz dynamics* (b) *Traffic* daily : 3 sensors (c) *Climate* yearly: 3 stations

Figure 3.4: Data visualizations: (3.4a) Genz dynamics, (3.4b) traffic data, (3.4c) climate data.



(a) *Genz dynamics* (b) *Traffic* (c) *Climate*

Figure 3.5: Forecasting RMSE for *Genz dynamics* and real world *traffic*, *climate* time series for varying forecasting horizon for LSTM, MLSTM, and TLSTM.

approximation. In particular, (Bigoni, Engsig-Karup, and Marzouk, 2016) used them to analyze tensor-train decompositions. We generated 10, 000 samples of length 100 using (3.2) with $w = 0.5, c = 1.0$ and random initial points.

Traffic The traffic data (see Figure 3.4 b) of Los Angeles County highway network is collected from California department of transportation `http://pems.dot.ca.gov/`. The prediction task is to predict the speed readings for 15 locations across LA, aggregated every 5 minutes. After upsampling and processing the data for missing values, we obtained 8, 784 sequences of length 288.

Climate The climate data (see Figure 3.4 c) is collected from the U.S. Historical Climatology Network (USHCN) (`http://cdiac.ornl.gov/ftp/ushcn_daily/`). The prediction task is to predict the daily maximum temperature for 15 stations. The data spans approximately 124 years. After preprocessing, we obtained 6, 954 sequences of length 366.

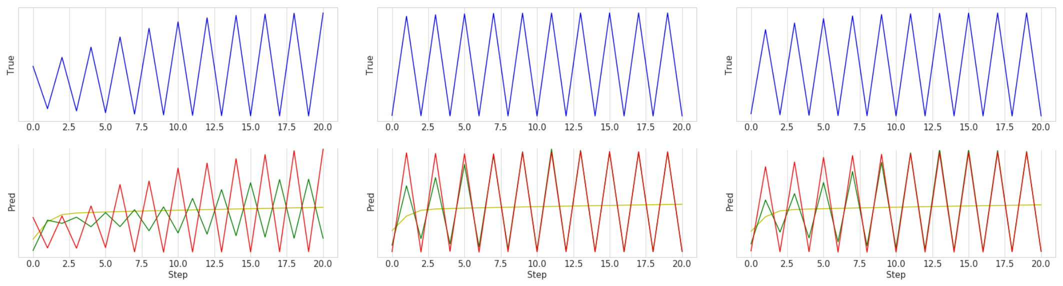


Figure 3.6: Model prediction for three realizations with different intiial conditions for Genz dynamics "product peak". Top (blue): ground truth. Bottom: model predictions for LSTM (green) and TLSTM (red). TLSTM perfectly captures the Genz oscillations, whereas the LSTM fails to do so (left) or only approaches the ground truth towards the end (middle and right).

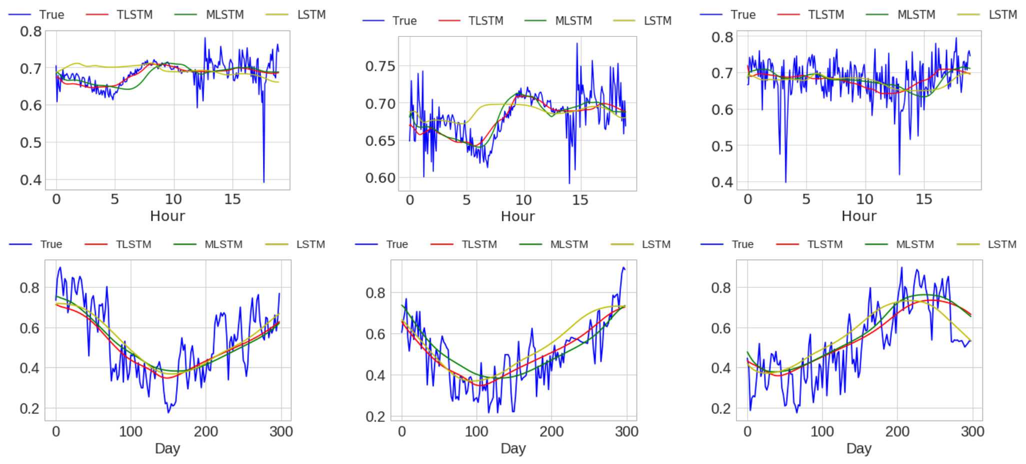


Figure 3.7: Top: 18 hour ahead predictions for hourly *traffic* time series given 5 hour as input for LSTM, MLSTM, and TLSTM. Bottom: 300 days ahead predictions for daily *climate* time series given 2 month observations as input for LSTM, MLSTM, and TLSTM.

Long-term Forecasting Evaluation

Experimental Setup To validate that TT-RNNs effectively perform long-term forecasting task in (3.3), we experiment with a seq2seq architecture with TT-RNN using LSTM as recurrent cells (TLSTM). For all experiments, we used an initial sequence of length t_0 as input and varied the forecasting horizon T. We trained all models using stochastic gradient descent on the length-T sequence regression loss $L(y, \hat{y}) = \sum_{t=1}^{T} ||\hat{y}_t - y_t||_2^2$, where $y_t = s_{t+1}, \hat{y}_t$ are the ground truth and model prediction respectively.

We compared TT-RNN against 2 set of natural baselines: 1st-order RNN (vanilla

RNN, LSTM), and matrix RNNs (vanilla MRNN, MLSTM), which use matrix products of multiple hidden states without factorization (Soltani and Jiang, 2016)). We observed that `TT-RNN` with RNN cells outperforms vanilla RNN and MRNN, but using LSTM cells performs best in all experiments. We also evaluated the classic ARIMA time series model with AR lags of $1 \sim 5$, and MA lags of $1 \sim 3$. We observed that it consistently performs $\sim 5\%$ worse than LSTM.

Training and Hyperparameter Search We trained all models using the RMS-prop optimizer and employed a learning rate decay of 0.8 schedule. We performed an exhaustive search over the hyper-parameters for validation. Table 3.1 reports the hyper-parameter search range used in this work.

Table 3.1: Hyper-parameter search range statistics for `TT-RNN` experiments and the best performing model size for all models.

Hyper-parameter Range

learning rate	tensor rank	hidden size
$10^{-1} \sim 10^{-5}$	$1 \sim 16$	$8 \sim 128$
# of lags	# of orders	# of layers
$1 \sim 6$	$1 \sim 3$	$1 \sim 3$

Best Performing Model Size

TLSTM	MLSTM	LSTM
7.2 k	9.7k	8.7 k

For all datasets, we used a $80\% - 10\% - 10\%$ train-validation-test split and train for a maximum of $1e^4$ steps. We compute the moving average of the validation loss and use it as an early stopping criteria. We also did not employ scheduled sampling, as we found training became highly unstable under a range of annealing schedules.

The number of parameters of best performing models are listed in Table 3.1. The TLSTM model is comparable with that of MLSTM and LSTM. More parameters would cause overfitting. TLSTM is more flexible than other methods, which gives us better control of the model complexity.

Long-term Accuracy For *traffic*, we forecast up to 18 hours ahead with 5 hours as initial inputs. For *climate*, we forecast up to 300 days ahead given 60 days of initial observations. For *Genz dynamics*, we forecast for 80 steps given 5 initial steps. All results are averages over 3 runs.

We now present the long-term forecasting accuracy of TLSTM in nonlinear systems. Figure 3.5 shows the test prediction error (in RMSE) for varying forecasting horizons for different datasets. We can see that TLSTM notably outperforms all baselines on all datasets in this setting. In particular, TLSTM is more robust to long-term error propagation. We observe two salient benefits of using `TT-RNN`s over the unfactorized models. First, MRNN and MLSTM can suffer from overfitting as the number of weights increases. Second, on *traffic*, unfactorized models also show considerable instability in their long-term predictions. These results suggest that tensor-train neural networks learn more stable representations that generalize better for long-term horizons.

Visualization of Predictions To get intuition for the learned models, we visualize the best performing TLSTM and baselines in Figure 3.6 for the Genz function "corner-peak" and the state-transition function. We can see that TLSTM can almost perfectly recover the original function, while LSTM and MLSTM only correctly predict the mean. These baselines cannot capture the dynamics fully, often predicting an incorrect range and phase for the dynamics.

In Figure 3.7 we show predictions for the real world traffic and climate dataset. We can see that the TLSTM corresponds significantly better with ground truth in long-term forecasting. As the ground truth time series is highly chaotic and noisy, LSTM often deviates from the general trend. While both MLSTM and TLSTM can correctly learn the trend, TLSTM captures more detailed curvatures due to the inherent high-order structure.

Speed Performance Trade-off We now investigate potential trade-offs between accuracy and computation. Figure 3.5b displays the validation loss with respect to the number of steps, for the best performing models on long-term forecasting. We see that `TT-RNN`s converge faster than other models, and achieve lower validation-loss. This suggests that `TT-RNN` has a more efficient representation of the nonlinear dynamics, and can learn much faster as a result.

Hyper-parameter Analysis The TLSTM model is equipped with a set of hyper-parameters, such as tensor-train rank and the number of lags. We perform a random grid search over these hyper-parameters and showcase the results in Table 3.2. In the top row, we report the prediction RMSE for the largest forecasting horizon w.r.t tensor ranks for all the datasets with lag 3. When the rank is too low, the model does

TLSTM Prediction Error (RMSE $\times 10^{-2}$)

Tensor rank r	2	4	8	16
Genz ($T = 95$)	**0.82**	0.93	1.01	1.01
Traffic ($T = 67$)	9.17	**9.11**	9.32	9.31
Climate ($T = 360$)	10.55	**10.25**	10.51	10.63

TLSTM Traffic Prediction Error (RMSE $\times 10^{-2}$)

Number of lags L	2	4	5	6
$T = 12$	**7.38**	7.41	7.43	7.41
$T = 84$	**8.97**	9.31	9.38	9.01
$T = 156$	9.49	9.32	9.48	**9.31**
$T = 228$	10.19	9.63	**9.58**	9.94

Table 3.2: TLSTM performance for various tensor-train hyperparameters. Top: varying tensor rank r with $L = 3$. Bottom: varying number of lags L and prediction horizon T.

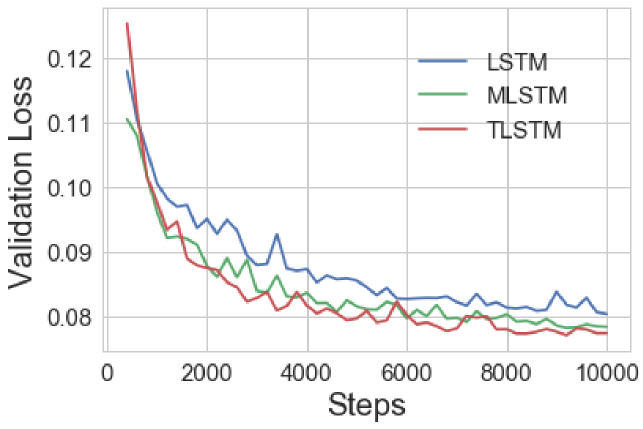
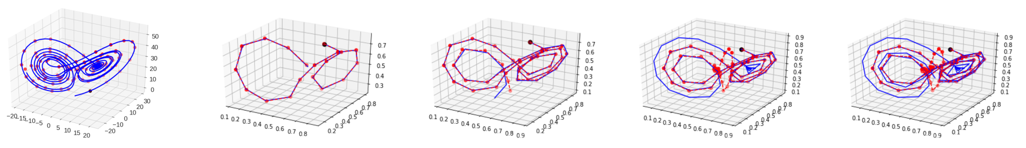


Figure 3.8: Training speed evaluation: validation loss versus steps for the models with the best long-term forecasting accuracy.

not have enough capacity to capture non-linear dynamics. When the rank is too high, the model starts to overfit. In the bottom row, we report the effect of changing lags (degree of orders in Markovian dynamics). For each setting, the best r is determined by cross-validation. For different forecasting horizon, the best lag value also varies.

Chaotic Nonlinear Dynamics We have also evaluated `TT-RNN` on long-term forecasting for *chaotic* dynamics, such as the Lorenz dynamics (see Figure 3.93.9a). Such dynamics are highly sensitive to input perturbations: two close points can



(a) $T = 0$ (b) $T = 20$ (c) $T = 40$ (d) $T = 60$ (e) $T = 80$ 6

Figure 3.9: 3.9a Lorenz attractor with dynamics (blue) and sampled data (red). 3.9b, 3.9c, 3.9d ,3.9e TLSTM long-term predictions for different forecasting horizons T versus the ground truth (blue). TLSTM shows consistent predictions over increasing horizons T.

move exponentially far apart under the dynamics. This makes long-term forecasting highly challenging, as small errors can lead to catastrophic long-term errors. Figure 3.9 shows that `TT-RNN` can predict up to $T = 40$ steps into the future, but diverges quickly beyond that. We have found no state-of-the-art prediction model is stable beyong 40 time stamps in this setting.

3.5 Related Work

Classic work in time series forecasting has studied auto-regressive models, such as the ARMA or ARIMA model (Box et al., 2015), which model a process $x(t)$ linearly, and so do not capture nonlinear dynamics. Recent development in RNNs has led to forecasting models such as deep AutoRegressive (Flunkert, Salinas, and Gasthaus, 2017) and Predictive State Representation (Downey, Hefny, and Gordon, 2017). However, RNNs only use the most recent hidden state and can be restrictive in modeling higher-order dynamics. Our method contrasts with this by explicitly modeling higher-order dependencies. Using neural networks to model time series has a long history. More recently, they have been applied to room temperature prediction, weather forecasting, traffic prediction and other domains. We refer to (Schmidhuber, 2015) for a detailed overview of the relevant literature.

From a modeling perspective, (Giles et al., 1989) considers a *high-order RNN* to simulate a deterministic finite state machine and recognize regular grammars. This work considers a second order mapping from inputs $x(t)$ and hidden states $h(t)$ to the next state. However, this model only considers the most recent state and is limited to two-way interactions. (Sutskever, Martens, and G. E. Hinton, 2011) proposes *multiplicative RNN* that allow each hidden state to specify a different factorized hidden-to-hidden weight matrix. A similar approach also appears in (Soltani and Jiang, 2016), but without the factorization. Moreover, hierarchical RNNs have been used to model sequential data at multiple resolutions, e.g., to learn both short-term and long-term human behavior (Zheng, Yue, and Lucey, 2016). Our method can be seen as an efficient generalization of these works where we model the high-order interactions using a hidden-to-hidden tensor.

Tensor methods have tight connections with neural networks. For example, (Novikov et al., 2015; Stoudenmire and Schwab, 2016) employs tensor-train as model compression tool to reduce the number of weights in neural networks. (Yang, Krompass, and Tresp, 2017) further extends this idea to RNNs by reshaping the inputs into a tensor. The focus of these work is model compression whereas `TT-RNN` aims to learn a high-order hidden states transition function. Theoretically, (Cohen, Sharir, and Shashua, 2016) shows convolutional neural networks have equivalence to hierarchical tensor factorizations. Mostly recently, (Khrulkov, Novikov, and I. Oseledets, 2017) provides expressiveness analysis for shallow network with tensor train models. This work however, to the best of our knowledge, is the first work to consider tensor networks in RNNs for sequential prediction tasks for learning in environments with nonlinear dynamics.

3.6 Discussion

In this work, we considered long-term forecasting under nonlinear dynamics. We propose a novel class of RNNs – `TT-RNN` that directly learns the nonlinear dynamics. We provide the first approximation guarantees for it representation power. We demonstrate the benefits of `TT-RNN` to forecast accurately for significantly longer time horizon in both synthetic and real-world multivariate time series data.

As we observed, chaotic dynamics still present a significant challenge to any sequential prediction model. Hence, it would be interesting to study how to learn robust models for chaotic dynamics.

In other sequential prediction settings, such as natural language processing, there does not (or is not known to) exist a succinct analytical description of the data-generating process. It would be interesting to go beyond forecasting and further investigate the effectiveness of `TT-RNNs` in such domains as well.

References

Barron, Andrew R (1993). "Universal approximation bounds for superpositions of a sigmoidal function". In: *IEEE Transactions on Information theory* 39.3, pp. 930–945.

Bigoni, Daniele, Allan P Engsig-Karup, and Youssef M Marzouk (2016). “Spectral tensor-train decomposition”. In: *SIAM Journal on Scientific Computing* 38.4, A2405–A2439.

Box, George EP et al. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Chung, Junyoung, Sungjin Ahn, and Yoshua Bengio (2016). “Hierarchical multiscale recurrent neural networks”. In: *arXiv preprint arXiv:1609.01704*.

Chung, Junyoung, Caglar Gulcehre, et al. (2014). “Empirical evaluation of gated recurrent neural networks on sequence modeling”. In: *arXiv preprint arXiv:1412.3555*.

Cohen, Nadav, Or Sharir, and Amnon Shashua (2016). “On the expressive power of deep learning: a tensor analysis”. In: *29th Annual Conference on Learning Theory*, pp. 698–728.

Downey, Carlton, Ahmed Hefny, and Geoffrey Gordon (2017). “Practical Learning of Predictive State Representations”. In: *arXiv preprint arXiv:1702.04121*.

Flunkert, Valentin, David Salinas, and Jan Gasthaus (2017). “DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks”. In: *arXiv preprint arXiv:1704.04110*.

Genz, Alan (1984). “Testing Multidimensional Integration Routines”. In: *Proc. Of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation*. Paris, France: Elsevier North-Holland, Inc., pp. 81–94. ISBN: 0-444-87570-0. URL: http://dl.acm.org/citation.cfm?id=2837.2842.

Giles, C Lee et al. (1989). “Higher order recurrent networks and grammatical inference.” In: *NIPS*, pp. 380–387.

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

Khrulkov, Valentin, Alexander Novikov, and Ivan Oseledets (2017). “Expressive power of recurrent neural networks”. In: *arXiv preprint arXiv:1711.00811*.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *Nature* 521.7553, pp. 436–444.

Novikov, Alexander et al. (2015). “Tensorizing neural networks”. In: *Advances in Neural Information Processing Systems*, pp. 442–450.

Orús, Román (2014). “A practical introduction to tensor networks: Matrix product states and projected entangled pair states”. In: *Annals of Physics* 349, pp. 117–158.

Oseledets, Ivan V (2011). “Tensor-train decomposition”. In: *SIAM Journal on Scientific Computing* 33.5, pp. 2295–2317.

Schmidhuber, Jürgen (2015). “Deep learning in neural networks: An overview”. In: *Neural networks* 61, pp. 85–117.

Soltani, Rohollah and Hui Jiang (2016). "Higher order recurrent neural networks". In: *arXiv preprint arXiv:1605.00064*.

Soltau, Hagen, Hank Liao, and Hasim Sak (2016). "Neural speech recognizer: Acoustic-to-word LSTM model for large vocabulary speech recognition". In: *arXiv preprint arXiv:1610.09975*.

Stoudenmire, Edwin and David J Schwab (2016). "Supervised Learning with Tensor Networks". In: *Advances in Neural Information Processing Systems*, pp. 4799–4807.

Sutskever, Ilya, James Martens, and Geoffrey E Hinton (2011). "Generating text with recurrent neural networks". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

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

Yang, Yinchong, Denis Krompass, and Volker Tresp (2017). "Tensor-Train Recurrent Neural Networks for Video Classification". In: *International Conference on Machine Learning*, pp. 3891–3900.

Zheng, Stephan, Yisong Yue, and Patrick Lucey (2016). "Generating long-term trajectories using deep hierarchical networks". In: *Advances in Neural Information Processing Systems*. S.T.Z. participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted all experiments and user studies, and participated in the writing of the manuscript., pp. 1543–1551. URL: https://papers.nips.cc/paper/6520-generating-long-term-trajectories-using-deep-hierarchical-networks.

Chapter 4

STRUCTURED EXPLORATION VIA HIERARCHICAL VARIATIONAL POLICY NETWORKS

Summary Reinforcement learning is challenging in environments with large state-action spaces, as exploration can be highly inefficient. Even if the dynamics are simple, the optimal policy can be combinatorially hard to discover. In this work, we propose a hierarchical approach to structured exploration to improve the sample efficiency of on-policy exploration in large state-action spaces. The key idea is to model a stochastic policy as a hierarchical latent variable model, which can learn low-dimensional structure in the state-action space, and to define exploration by sampling from the low-dimensional latent space. This approach enables lower sample complexity, while preserving the expressiveness of the policy class. To make learning tractable, we derive a joint learning and exploration strategy by combining hierarchical variational inference with actor-critic learning. The benefits of our learning approach are that it is principled, simple to implement, scalable to settings with many actions, and composable with existing deep learning approaches. We evaluate our approach on learning a deep centralized multi-agent policy, as multi-agent environments naturally have an exponentially large state-action space. We demonstrate that our approach can more efficiently learn optimal policies in challenging multi-agent games *with a large number (~ 20) of agents*, compared to conventional baselines.

4.1 Introduction

Reinforcement learning in environments with large state-action spaces is challenging, as exploration can be highly inefficient in high-dimensional spaces. Hence, even if the environment dynamics are simple, the optimal policy can be combinatorially hard to discover. However, for many large-scale environments, the high-dimensional state-action space has (often hidden or implicit) low-dimensional structure which can be exploited.

Many natural examples are in collaborative multi-agent problems, whose state-action space is exponentially large in the number of agents, but have a low-dimensional coordination structure. For instance, consider a simple variant of the Hare-Hunters problem (see Figure 4.1). In this game, $N = 2$ identical hunters need to capture

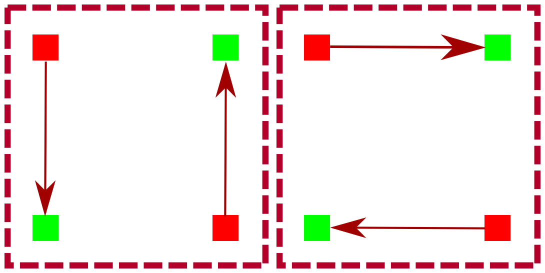


Figure 4.1: Equivalent solutions in a 2-hunter 2-prey game.

$M = 2$ identical *static* prey within T time-steps, and exactly $H = 1$ hunter is needed to capture each prey. T is set such that no hunter can capture both preys. There are two equivalent solutions: hunter 1 captures prey 1 and hunter 2 captures prey 2, or vice versa. There are also two suboptimal choices: both hunters choose the same prey. Hence, the hunters must coordinate *over a (large) number of time-steps* to maximize their reward. This implies the solution space has low-dimensional structure that can be used to accelerate exploration and training.

In this work, we propose a principled approach to structured exploration to improve sample complexity in large state-action spaces, by learning deep probabilistic hierarchical policies with a latent structure. As a high-level intuition, consider a tabular multi-agent policy, which maps discrete (joint) states to action probabilities. For N agents with S states and A actions each, this policy has $O((S \cdot A)^N)$ weights. However, the low-dimensional coordination structure can be captured by a factorized, low-rank matrix, where the factorization can be learned and, for instance, only has $O(NK(S + A))$ weights. Similarly, our approach both 1) learns a low-dimensional "factorization" of the policy distribution and 2) defines exploration by also sampling from the low-dimensional latent space. For instance, in the multi-agent setting, we can learn a centralized multi-agent policy with a latent structure that encodes (a distribution of possible) coordination between agents and biases exploration towards policies that (likely) encode "good" coordination.

The key ideas of our approach are: 1) to utilize a shared stochastic latent variable model that defines the structured exploration policy, and 2) to employ a principled variational method to learn the posterior distribution over the latents jointly with the optimal policy. Our approach has several desirable properties. First we do not incorporate any form of prior domain knowledge, but rather discover the coordination structure purely from empirical experience during learning. Second, our variational learning method enables fully differentiable end-to-end training of the entire policy class. Finally, by utilizing a hierarchical policy class, our approach can easily scale to large action spaces (e.g., a large number of coordinating agents). Our approach can also be seen as a deep hierarchical generalization of Thompson sampling, which is a historically popular way to capture correlations between actions (e.g., in the bandit setting (Agrawal and Goyal, 2012)).

To summarize, our contributions in this work are as follows:

- We introduce a structured probabilistic policy class that uses a hierarchy of stochastic latent variables.
- We propose an efficient and principled algorithm using variational methods to train the policy end-to-end.
- To validate our learning framework, we introduce several challenging synthetic multi-agent environments that explicitly require team coordination, and feature competitive pressures that are characteristic of many coordinated decision problems.
- We empirically verify that our approach improves sample complexity on coordination games with a large number ($N \sim 20$) of agents.
- We show that learned latent structures correlate with meaningful coordination patterns, which implies that our latent structure can recover meaningful low-dimensional structure in the state-action space.

4.2 Problem Setup & Approach

In this paper, we use centralized multi-agent environments to showcase the efficacy of our approach to structured exploration, as they naturally exhibit exponentially large state-action spaces. More generally, our approach can be applied to any reinforcement learning setting where one can posit the existence of a low-dimensional structure within a high dimensional state-action space.

In centralized multi-agent RL, agents sequentially interact within an environment defined by the tuple: $\mathcal{E} \equiv (\mathbf{S}, \mathbf{A}, \mathbf{r}, f_P)$. Each agent i starts in an initial state s_0^i, and at each time t observes a state $\boldsymbol{s}_t \in \mathbf{S}$ and executes an action a_t^i chosen by a (stochastic) policy $a_t^i \sim P\left(a_t^i|\boldsymbol{s}_t\right)$. Each agent then receives a reward $r^i\left(\boldsymbol{s}_t, \boldsymbol{a}_t\right)$, and the environment transitions to a new state $\boldsymbol{s}_{t+1}$ with probability $f_P\left(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t\right)$. We define the joint state and actions as $\boldsymbol{s}_t = \{s_t^i \in \mathbf{S}\}$ and $\boldsymbol{a}_t = \{a_t^i \in \mathbf{A}\}$, where $i \in \mathcal{I}$

indexes the agents. Note that the rewards for each agent r^i can depend on the full joint state and actions.

In this work, we restrict to fully cooperative MDPs that are fully observable, deterministic and episodic.[1] Each agent can see the full state $\boldsymbol{s}$, f_P is deterministic and each episode $\tau = (\boldsymbol{s}_t, \boldsymbol{a}_t)_{0 \le t \le T}$ ends when the agent encounters a terminal state and the MDP resets.

In the fully cooperative case, the goal for each agent is to learn its optimal policy $P^* \left(a^i_t | \boldsymbol{s}_t, \boldsymbol{\theta}\right)$ that maximizes the total reward $R(\tau) = \sum_i R^i(\tau) = \sum_i \sum_t r^i(\boldsymbol{s}_t, \boldsymbol{a}_t)$:

$$\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \sum_{i \in I} J^i(\boldsymbol{\theta}), \tag{4.1}$$

$$J^i(\boldsymbol{\theta}) = \mathbb{E}\left[R^i(\tau) \middle| \boldsymbol{a}_t \sim P(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}) \right] \tag{4.2}$$

To optimize, we can apply gradient descent with policy gradient estimators $\hat{g}_{\boldsymbol{\theta}}$ Williams, 1992

$$\begin{aligned} g_{\boldsymbol{\theta}} &= \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}\left[\nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{a}_t | \boldsymbol{s}_t; \boldsymbol{\theta}) R(\tau) | \boldsymbol{a}_t, \boldsymbol{s}_t \right] \\ &\approx \frac{1}{M} \sum_{k=1}^{M} \sum_t \nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{a}^k_t | \boldsymbol{s}^k_t; \boldsymbol{\theta}) R(\tau^k), \end{aligned} \tag{4.3}$$

where we sample M rollouts τ^k by sampling actions from the policy that is being learned.

Challenges in Exploration. A central issue in reinforcement learning is the *exploration-exploitation* trade-off: how can the policy sample rollouts and learn efficiently? In particular, when the state-action space is exponentially large (as in multi-agent RL), discovering a good policy becomes intractable as the number of agents N grows. Hence, exploration in large state-action spaces poses a significant challenge. We consider the general policy gradient setting where exploration is driven by the stochastics of the policy (similar to Thompson Sampling Agrawal and Goyal, 2012 in the bandit setting). Thus, it is important for the policy to maintain *expressive low-dimensional* distributions that can make exploration more efficient.

[1] More generally, multi-agent problems can be generalized along many dimensions, e.g., one can learn decentralized policies in partial-information settings. For an overview, see Busoniu, Babuska, and De Schutter, 2008.

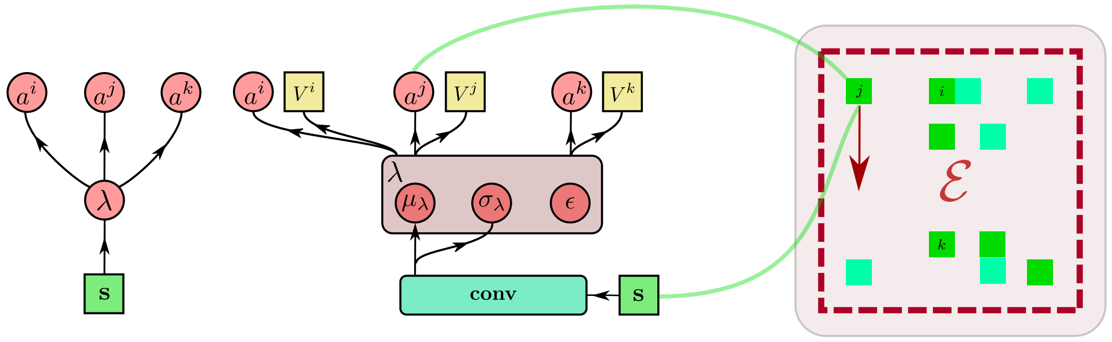


Figure 4.2: Structured latent variable model of a centralized multi-agent policy (*actor*, left) and instance of the actor-critic interacting in the environment $\mathcal{E}$ (right). The policy contains two stacked layers of stochastic latent variables (red), and deterministically receives states and computes actions (green). The actions for individual agents are independent conditional on latent variables λ, which implies a "factorization" of the state-action space. On the right, a neural network instance of the actor-critic uses the reparametrization trick and receives the environment state, samples actions from the policy (for all agents) and computes the value function $V^{\bullet}$.

Hierarchical Variational Policy

We now formulate (4.1) using a stochastic hierarchical policy class that enables structured exploration. Our approach builds upon two complementary approaches:

- Encode structured exploration using a latent variable that "factorizes" the state-action space. In the multi-agent setting, the factorization is per-agent and the latent variable encode coordination encodes coordination between agents.
- Use a variational approach to derive and optimize a lower bound on the objective (4.1).

Hierarchical Latent Model. For simplicity, we assume a factorization of the state-action space into state-action spaces for individual agents in the multi-agent RL setting. More generally, one could consider other factorization structures as well. To encode coordination between agents, we assume the individual per-agent sub-policies have shared structure, encoded by a latent variable $\lambda_t \in \mathbb{R}^n$ for all t, where n is the dimension of the latent space. This leads to a hierarchical policy model $P(\boldsymbol{a}_t, \lambda_t|\boldsymbol{s}_t)$, as shown in Figure 4.2. We first write the joint policy for a single time-step as:

$$
\begin{aligned}
P(\boldsymbol{a}_t|\boldsymbol{s}_t) &= \int d\lambda_t P(\boldsymbol{a}_t, \lambda_t|\boldsymbol{s}_t) = \int d\lambda_t \prod_{i=1}^{N} P(a_t^i, \lambda_t|\boldsymbol{s}_t) \\
&= \int d\lambda_t \prod_{i=1}^{N} P(a_t^i|\lambda_t, \boldsymbol{s}_t) P(\lambda_t|\boldsymbol{s}_t), \qquad (4.4)
\end{aligned}
$$

where we introduced the conditional priors $P(\lambda_t|\boldsymbol{s}_t)$. The latent variables λ_t introduce dependencies among the $\boldsymbol{a}_t$, hence this policy is more flexible compared to standard fully factorized policies Ranganath, Tran, and Blei, 2015. Note that this approach supports centralized learning and decentralized execution, by sharing a random seed amongst agents to sample λ_t and actions $\boldsymbol{a}_t$ during execution.

Computing the integral in the optimal policy (4.4) is hard, because the unknown distribution $P(a_t^i|\lambda_t, \boldsymbol{s}_t)$ can be highly complex. Hence, to make learning (4.4) tractable, we will use a variational approach.

Hierarchical Variational Lower Bound. We next derive a tractable learning algorithm using variational methods. Instead of directly optimizing (4.1), we cast it as a probabilistic inference problem, as in Levine and Koltun, 2013; Vlassis et al., 2009, and instead optimize a lower bound.

To do so, we model R^i as a *random variable*, whose distribution $P(R^i|\tau) \propto \exp R^i(\tau)$. For clarify, we will suppress the index i hereafter. Maximizing expected reward (4.1) can then be seen as probabilistic inference:

$$
\max_{\boldsymbol{\theta}} \mathbb{E}\left[R(\tau)\,|\tau \sim \pi(\boldsymbol{s}_t;\boldsymbol{\theta})\right] = \max_{\boldsymbol{\theta}} P(R|\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \log P(R|\boldsymbol{\theta}), \qquad (4.5)
$$

where the distribution $P(R|\boldsymbol{\theta})$ can be written as:

$$
P(R|\boldsymbol{\theta}) = \int d\tau P(R|\tau) P(\tau;\boldsymbol{\theta}), \qquad (4.6)
$$

$$
P(\tau;\boldsymbol{\theta}) = P(\boldsymbol{s}_0) \prod_{t=0}^{T} P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) P(\boldsymbol{a}_t|\boldsymbol{s}_t;\boldsymbol{\theta}). \qquad (4.7)
$$

We first introduce latent variables λ_t by applying (4.4) at each timestep t:

$$
P(\tau;\boldsymbol{\theta}) \equiv \int d\lambda_{0:T} P(\tau, \lambda_{0:T};\boldsymbol{\theta}) \propto \prod_{t=0}^{T} \int d\lambda_t P(\boldsymbol{a}_t, \lambda_t|\boldsymbol{s}_t;\boldsymbol{\theta}), \qquad (4.8)
$$

where we summarized the terms in (4.8) using shorthands $\lambda_{0:T} \equiv \prod_{t=0}^{T} \lambda_t$ and $\int d\lambda_{0:T} \equiv \prod_{t=0}^{T} \int d\lambda_t$.

We now apply the variational approach to the inference problem (4.5), by maximizing a lower bound on its log-likelihood. This lower bound is derived using Jensen's inequality:

$$\begin{aligned}\log P(R|\boldsymbol{\theta}) &= \log \int d\tau d\lambda_{0:T} P(R|\tau) P(\tau, \lambda_{0:T}; \boldsymbol{\theta}) \\ &\geq \int d\tau d\lambda_{0:T} Q(\tau, \lambda_{0:T}; \boldsymbol{\phi}) \log \left(\frac{P(R|\tau) P(\tau, \lambda_{0:T}; \boldsymbol{\theta})}{Q(\tau, \lambda_{0:T}; \boldsymbol{\phi})} \right) \\ &\equiv \texttt{ELBO}\,(\boldsymbol{\theta}, \boldsymbol{\phi}) \end{aligned} \tag{4.9}$$

where $Q\,(\tau, \lambda_{0:T}; \boldsymbol{\phi})$ is a variational distribution with parameters $\boldsymbol{\phi}$, and we used (4.6) and (4.8). This lower bound is commonly called the Evidence Lower Bound (ELBO). To simplify (4.9), we can separate out reward and transition probabilities in Q:

$$Q(\tau, \lambda_{0:T}; \boldsymbol{\phi}) = P(R|\tau) P(\boldsymbol{s}_0) \prod_{t=0}^{T} P(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) \hat{Q}(\lambda_t|\boldsymbol{s}_t; \boldsymbol{\phi}), \tag{4.10}$$

which simplifies the ELBO:

$$\texttt{ELBO}\,(\boldsymbol{\theta}, \boldsymbol{\phi}) = \int d\tau d\lambda_{0:T} Q(\tau, \lambda_{0:T}; \boldsymbol{\phi}) \sum_{t=0}^{T} \left(\log P(\boldsymbol{a}_t|\lambda_t, \boldsymbol{s}_t; \boldsymbol{\theta}) + \log \frac{P(\lambda_t|\boldsymbol{s}_t)}{\hat{Q}(\lambda_t|\boldsymbol{s}_t, \boldsymbol{\phi})} \right). \tag{4.11}$$

Here, we assumed the policy prior $P(\lambda_t|\boldsymbol{s}_t)$ is *fixed* and does not depend on $\boldsymbol{\theta}$.

The standard choice for the prior $P(\lambda_t|\boldsymbol{s}_t)$ is to use maximum-entropy standard-normal priors: $P(\lambda_t|\boldsymbol{s}_t) = \mathcal{N}(\mathbf{0}, \mathbf{1})$. We can then maximize (4.11) using e.g., stochastic gradient ascent. Formally, the policy gradient is:

$$\begin{aligned} g_{\boldsymbol{\theta}} \approx g_{\boldsymbol{\theta},Q} &= \nabla_{\boldsymbol{\theta}} \texttt{ELBO}(\boldsymbol{\theta}, \boldsymbol{\phi}) \\ &= \int d\tau d\lambda_{0:T} Q(\tau, \lambda_{0:T}; \boldsymbol{\phi}) \sum_{t'=0}^{T} \nabla_{\boldsymbol{\theta}} \log P(\boldsymbol{a}_{t'}|\lambda_{t'}, \boldsymbol{s}_{t'}; \boldsymbol{\theta}), \end{aligned} \tag{4.12}$$

which is an approximation of the true policy gradient (4.3), and can be estimated using roll-outs τ^k of the policy P^{π}:

$$g_{\boldsymbol{\theta},Q} \approx \hat{g}_{\boldsymbol{\theta},Q} = \frac{1}{M} \sum_{k=1}^{M} \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log P\left(\boldsymbol{a}_t^k | \lambda_t^k, \boldsymbol{s}_t^k; \boldsymbol{\theta} \right) R(\tau^k). \tag{4.13}$$

During a rollout τ^k, we sample $\lambda \sim Q$, observe rewards $R \sim P(R|\tau)$ and transitions $s_{t+1} \sim P(s_{t+1}|.)$, and use these to compute (4.13). We can similarly compute $g_{\boldsymbol{\phi},Q} = \nabla_{\boldsymbol{\phi}} \texttt{ELBO}(Q, \boldsymbol{\theta}, \boldsymbol{\phi})$, the gradient for the variational posterior Q.

Preys are	**Frozen**		**Moving**		**Frozen**		**Moving**	
Samples (x100k)	5	10	5	10	5	10	5	10
`Hare-Hunters`	**10-vs-10**				**20-vs-20**			
`Cloned`	85.0	178.3	465.0	912.5	20.0	70.0	706.7	1401.7
`Shared`	65.0	155.0	457.5	923.8	65.0	105.0	491.4	962.9
`HLPN`	**240.0**	**580.0**	**662.7**	**1381.8**	**200.0**	**393.3**	**1260.0**	**2344.0**
`Stag-Hunters`	**10-vs-10**				**20-vs-20**			
`Cloned`	1229.2	2482.9	2079.4	4171.2	3224.5	6219.9	5934.5	11429.2
`Shared`	1214.5	2423.7	2005.7	4144.3	3150.7	6379.8	6344.4	12196.8
`HLPN`	**1515.2**	**3047.3**	**2275.7**	**4610.7**	**3799.3**	**7158.1**	**6880.7**	**13358.6**

Table 4.1: Total terminal reward (averaged over 5 best runs) for N agents for set # of training samples. Our `HLPN` approach outperforms baselines by up to 10x.

Actor-Critic and Bias-Variance. Estimating policy gradients $g_{\boldsymbol{\theta}}$ using empirical rewards can suffer from high variance. It is useful to consider more general objectives F^i:

$$J^i(\boldsymbol{\theta}) = \mathbb{E}\left[F^i(\tau)\middle|\, \boldsymbol{a}_t \sim P^{\pi_t}(\boldsymbol{a}_t|\boldsymbol{s}_t;\boldsymbol{\theta})\right], \qquad (4.14)$$

such that the variance in $\hat{g}$ is reduced.[2] In practice, we find that using (4.13) with more general F, such as generalized advantages Schulman et al., 2015, performs quite well.

4.3 Model and Experimental Setup

Multi-agent Environments

To validate our approach, we created two grid-world games, depicted in Figure 4.2, inspired by the classic Predator-Prey and Stag-Hunt games Shoham and Leyton-Brown, 2008. In both, the world is periodic and the initial positions of the hunters and prey are randomized. We consider two instances for both games: either the prey are moving or fixed.

Hare-Hunters. Predator-Prey is a classic test environment for multi-agent learning, where 4 predators try to capture a prey by boxing it in. We consider a variation defined by the settings (N, M, H, T): N hunters and M prey. Each prey can be captured by exactly H hunters: to capture the prey, a hunter gets next to it, after which the hunter is frozen. Once a prey has had H hunters next to it, it is frozen and

[2]Note that if the total reward R is bounded, we can define R-weighted probabilities by shifting and normalizing R. In general, the derivation applies if F is similarly bounded.

cannot be captured by another hunter. The terminal rewards are:

$$R^i = \begin{cases} 1, & \text{if } \textit{all} \text{ prey are captured } H \text{ times before the} \\ & \text{time limit } T \\ 0, & \text{otherwise.} \end{cases} \tag{4.15}$$

The challenge of the game is for the agents to inactivate all prey within a finite time T. Due to the time limit, the optimal strategy is for the agents to distribute targets efficiently, which can be challenging due to the combinatorially large number of possible hunter-to-prey assignments.

Stag-Hunters. The Stag-Hunt is another classic multi-agent game designed to study coordination. In this game, hunters have a choice: either they capture a hare for low reward, or, together with another hunter, capture a stag for a high reward. We extend this to the multi-agent (N, M, H, T)-setting: N hunters hunt M prey ($M/2$ stags and $M/2$ hares). Each stag has H hit-points, while hares and hunters have 1 hit-point. Capturing is as in `Hare-Hunters`. The spatial domain is similar to the `Hare-Hunters` game and we also use a time limit T. The terminal reward is now defined as:

$$R^i = \begin{cases} 1, & \text{if } i \text{ captured a live stag that became inactive} \\ & \text{before the time limit } T \\ 0.1, & \text{if } i \text{ captured a live hare} \\ & \text{before the time limit } T \\ 0, & \text{otherwise} \end{cases} \tag{4.16}$$

The challenge for the agents here is to discover that choosing to capture the same prey can yield substantially higher reward, but this requires coordinating with another hunter.

Neural Coordination Model

For experiments, we instantiated our policy class (as in Figure 4.2) with deep neural networks. For simplicity, we only used *reactive policies* without memory, although it is straightforward to apply our approach using policies with memory (e.g., LSTMs). The model takes a joint state $\boldsymbol{s}_t$ as input and computes features $\phi(\boldsymbol{s})$ using a 2-layer convolutional neural network. To compute the latent variable $\lambda \in \mathbb{R}^d$, we use the reparametrization trick (Kingma and Welling, 2013) to learn the variational distribution (e.g., $Q(\lambda \mid \boldsymbol{s})$), sampling λ via $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and distribution parameters

μ, σ (omitting t):

$$\mu(\boldsymbol{s}) = W_\mu \phi(\boldsymbol{s}) + b_\mu, \quad \log\sigma(\boldsymbol{s})^2 = W_\sigma \phi(\boldsymbol{s}) + b_\sigma, \\ \lambda = \mu(\boldsymbol{s}) + \sigma(\boldsymbol{s}) \odot \epsilon. \tag{4.17}$$

Given λ, the model then computes the policies $P(a^i \mid, \boldsymbol{s})$ and value functions $V^i(\boldsymbol{s})$ as (omitting t):

$$P(a^i \mid, \boldsymbol{s}) = \texttt{softmax}\left(W^i_\pi[\lambda \ \phi(\boldsymbol{s})] + b^i_\pi\right), \\ V^i(\boldsymbol{s}) = W^i_V \phi(\boldsymbol{s}) + b^i_V, \tag{4.18}$$

where $\texttt{softmax}(\boldsymbol{x}) = \exp \boldsymbol{x} / \sum_j \exp x^j$. In this way, the model can be trained end-to-end.

Training. We used A3C Mnih et al., 2016 with KL-controlled policy gradients (4.13), generalized advantage as F Schulman et al., 2015. and policy-entropy regularization. The loss for the value function at each state $\mathbf{s}_t$ is the standard L_2-loss between the observed total rewards for each agent i and its value estimate:

$$\alpha \sum_t \sum_i \left(V^i(\mathbf{s}_t) - R^i(\mathbf{s}_t, \mathbf{a}_t)\right)^2. \tag{4.19}$$

In addition, in line with other work using actor-critic methods, we found that adding a small entropy regularization on the policy can sometimes positively influence performance, but this does not seem to be always required for our testbeds. The entropy regularization is:

$$H(P) = -\beta \sum_t \sum_{\mathbf{a}_t} P(\mathbf{a}_t \mid \mathbf{s}_t; \boldsymbol{\theta}) \log P(\mathbf{a}_t \mid \mathbf{s}_t; \boldsymbol{\theta}). \tag{4.20}$$

A3C additionally defines training minibatches in terms of a fixed number of environment steps L: a smaller L gives faster training with higher variance and vice versa.

For all experiments, we performed a hyper-parameter search and report the best 5 runs seen.

Baselines. We compared against two natural baselines:

- `Shared` (shared actor-critic): agents share a deterministic hidden layer, but maintain individual weights θ^i for their (stochastic) policy $P(a \mid, \boldsymbol{s}; \theta^i)$ and value function $V^i(\boldsymbol{s}; \boldsymbol{\theta})$. The key difference is that this model does not sample from the shared hidden layer.

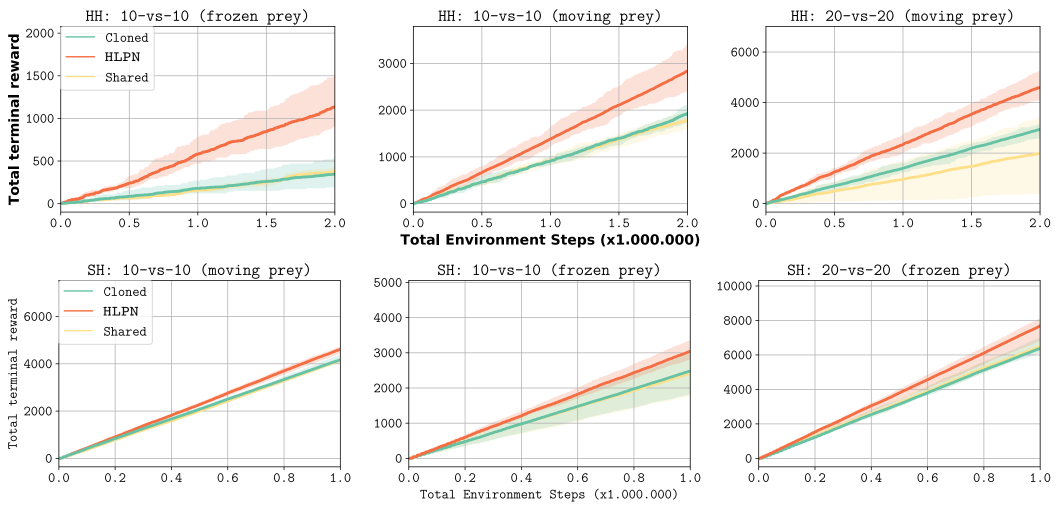


Figure 4.3: Train-time cumulative terminal reward for N agents in $(N, M, 1, T)$ `Hare-Hunters` (upper, $T = 2000$) and `Stag-Hunters` (lower, $T = 1000$) on a 50×50 gridworld, for **10-vs-10** or **20-vs-20** agents; randomly moving or fixed preys. Average, minimal and maximal rewards for the best 5 runs for each model are shown. Our `HLPN` approach accumulates increasingly higher rewards compared to the baselines, by 1) achieving higher terminal reward per episode and 2) finishing episodes faster (see Figure 4.4). For 10-10 `Stag-Hunters` with frozen prey, average reward per-episode is 4.64 (`Cloned`), 6.22 (`Shared`), 6.61 (`HLPN`) after 1 million samples.

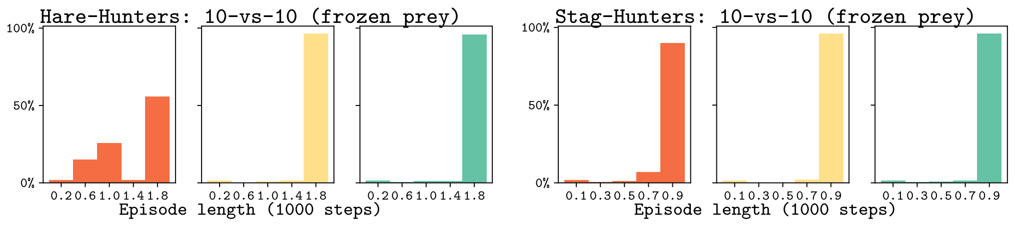


Figure 4.4: Train-time episode lengths during 1 million steps for **10-vs-10** `Hare-Hunters` (left) and `Stag-Hunters` (right), with fixed preys. Our `HLPN` approach (orange) finishes an episode successfully before the time limit more often than the baselines (`Cloned` (blue) and `Shared` (yellow)).

- `Cloned` (actor-critic): each agent uses an identical policy and value function with shared weights. There is shared information between the agents, and actions are sampled according to the agents' own policies.

4.4 Quantitative Analysis

We now validate the efficacy of our approach by showing our method scales to environments with a large number of agents. We ran experiments for both

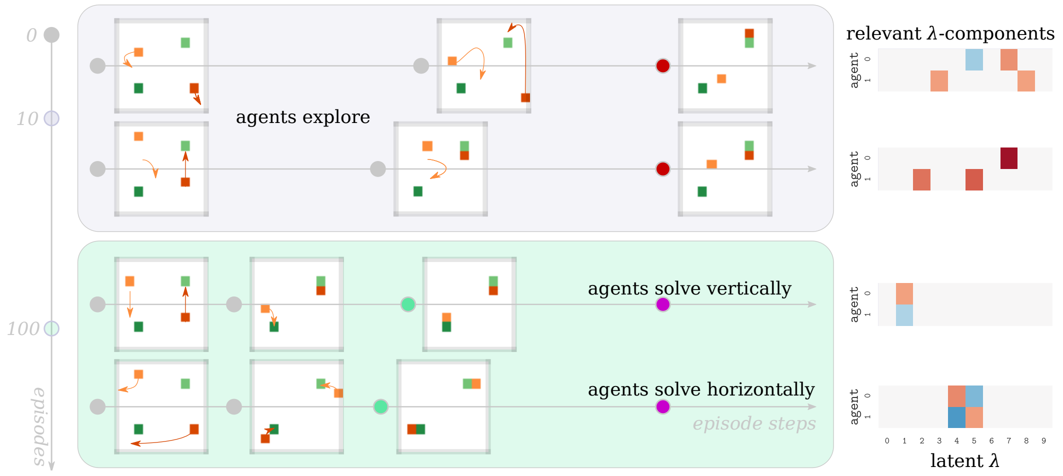


Figure 4.5: Predators (red) and prey (green) during training for 2v2 `Hare-Hunters` for 100 episodes. Arrows show where agents move to in the next frame. Top: at the start, predators explore via λ, but do not succeed before the time limit T (red dot). Bottom: after convergence agents succeed consistently (green dot) before the time limit (purple dot) and λ encodes the two strategies from Figure 4.1. Highlighted λ-components correlate with rollout under a 2-sided t-test at $\alpha = 0.1$ significance.

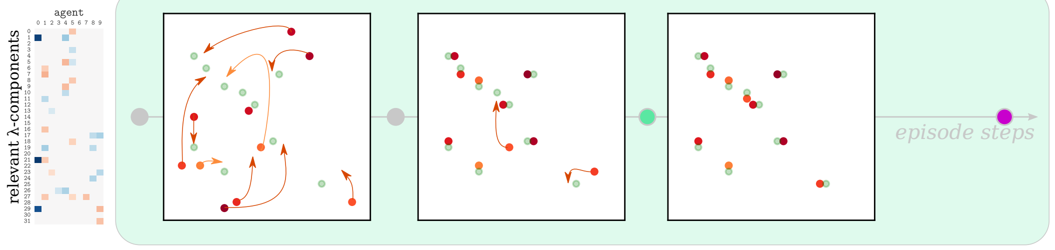


Figure 4.6: Visualization of our `HLPN` approach for a (10, 10, 1, 1000) `Hare-Hunters` game in a 30×30 world. Left: components of the latent code λ that significantly correlate with sampled actions (computed as in Figure 4.5). Right: Three episode snapshots: at the start, middle, and end. Red: predators; green: prey. Arrows indicate where agents move to in the next snapshot. The hunters solve the game (green dot) before the time limit $T = 1000$ (purple dot), by distributing targets amongst themselves.

`Hare-Hunters` and `Stag-Hunters` for $N = M = 10, 20$ in a spatial domain of 50×50 grid cells. We refer to our approach as `HLPN` (Hierarchical Latent Policy Network).

Sample complexity. In Table 4.1 we show the achieved rewards after a fixed number of training samples, and Figure 4.3 shows the corresponding learning curves. We see that `HLPN` achieves up to 10× reward compared to the baselines. Figure 4.4 shows the corresponding distribution of training episode lengths. We see that

`HLPN` solves game instances more than 20% faster than baselines in 50% (10%) of `Hare-Hunters` (`Stag-Hunters`) episodes. In particular, `HLPN` learns to coordinate for higher reward more often: it achieves the highest average reward per-episode (e.g., for 10-10 `Stag-Hunters` with frozen prey, average rewards are 4.64 (`Cloned`), 6.22 (`Shared`), 6.61 (`HLPN`)). Hence, `HLPN` coordinates successfully more often to capture the stags. Together, these results show `HLPN` enables more efficient learning.

Using the ELBO. A salient difference between (4.13) and (4.3) is the KL-regularization, which stems from the derivation of the ELBO. Since we use a more general objective F, c.f. (4.14), we also investigated the impact of using the KL-regularized policy gradient (4.13) versus the standard (4.3). We ran several instances of the above experiments both with and without KL-regularization. We found that without KL-regularization, training is unstable and prone to mode collapse: the variance σ of the variational distribution can go to 0, leading to essentially 0 achieved reward for any reasonable hyperparameter settings.

Impact of dynamics and T. Inspecting training performance, we see the relative difficulty of capturing moving or randomly moving prey. Capturing moving prey is easier to learn than capturing fixed preys, as comparing rewards in Table 4.1 shows. This shows a feature of the game dynamics: the expected distance between a hunter and an uncaptured prey are lower when the preys are randomly moving, resulting in an easier game. Comparing `Hare-Hunters` and `Stag-Hunters`, we also see the impact of the time limit T. Since we use terminal rewards only, as T gets larger, the reward becomes very sparse and models need more samples to discover good policies.

4.5 Model Inspection

Beyond training benefits, we now demonstrate empirical evidence that suggest efficacy and meaningfulness of our approach to structured exploration. We start by inspecting the behavior of the latent variable λ for a simple $N = M = 2$ `Hare-Hunters` game, which enables semantic inspection of the learned policies, as in Figure 4.5. We make a number of observations. First, λ is relevant: many components are statistically significantly correlated with the agents' actions. This suggests the model does indeed use the latent λ: it (partly) controls the coordination between agents.[3] Second, the latent λ shows strong correlation during all phases of training. This suggests that the model indeed is performing a form of structured exploration. Third, the components

[3]This is parallel to the discussion in Chen et al., 2016, which investigates the effectiveness of latent codes λ.

of λ are correlated with semantic meaningful behavior. We show a salient example in the bottom 2 rows in Figure 4.5: the correlated components of λ are disjoint and each component correlates with both agents. The executed policies are exactly the two equivalent ways to assign 2 hunters to 2 preys, as illustrated in Figure 4.1.

Coordination with a large N. In the large $N = M = 10$ case, the collective dynamics are generalize the $N = M = 2$ case. There are now redundancies in multi-agent hunter-prey assignments that are analogous to the $N = M = 2$ case that are prohibitively complex to analyze due to combinatorial complexity. However our experiments strongly suggest (see e.g., Figure 4.6) the latent code is again correlated with agents' behavior during all phases of training, hence λ induces meaningful multi-agent coordination.

4.6 Related Work

Deep Structured Inference. Recent works have focused on learning structured representations using expressive distributions, which enable more powerful probabilistic inference. For instance, Johnson et al., 2016 has proposed combining neural networks with graphical models, while Ranganath, Tran, and Blei, 2015 learn hierarchical latent distributions. Our work builds upon these approaches to learn structured policies in the reinforcement learning setting. In the multi-agent setting, the RL problem has also been considered as an inference problem in e.g., (Liu, Amato, Liao, et al., 2015; Wu, Zilberstein, and Jennings, 2013; Liu, Amato, Anesta, et al., 2016).

Variational methods in RL. Neumann, 2011; Furmston and Barber, 2010 discuss variational approaches for RL problems, but did not consider end-to-end trainable models. Levine and Koltun, 2013 used variational methods for guided policy search. Houthooft et al., 2016 learned exploration policies via information gain using variational methods. However, these only consider 1 agent.

Coordination in RL. Multi-agent coordination has been studied in the RL community (e.g., Guestrin, Lagoudakis, and Parr, 2002; Kapetanakis and Kudenko, 2002; Chalkiadakis and Boutilier, 2003, for instance, as a method to reduce the instability of multiple agents learning simultaneously using RL. The benefit of coordination was already demonstrated in simple multi-agent settings in e.g., Tan, 1993. The shared latent variable λ of our structured policy can also be interpreted as a learned *correlation device* (see Bernstein, 2005 for an example in the decentralized setting), which can be used to e.g., break ties between alternatives or induce coordination between agents. More generally, they can be used to achieve correlated equilibria

(Greenwald and Hall, 2003), a more general solution concept than Nash equilibria. However, previous methods learned hand-crafted models and do not scale well to complex state spaces and many agents. In contrast, our method learns coordination end-to-end via on-policy methods, *learns* the multi-agent exploration policy and scales well to many agents via its simple hierarchical structure.

Communication Models. Recently, end-to-end learning of communication models has been studied for shared broadcast channels Sukhbaatar, Szlam, and Fergus, 2016, sequential communication Peng et al., 2017, heuristic multi-agent exploration Usunier et al., 2016 and bit-channels Foerster et al., 2016. These works show that communication protocols can be learned through back-propagation or heuristic stabilization methods, but often do not scale well to a large number of agents. Our hierarchical approach is complementary, learns via variational methods, and can scale to large N.

Multi-task learning. Hierarchical models have been studied for multi-task learning, e.g., Daume III, 2014 learns latent hierarchies via EM in a supervised learning setting. Instead, we study flexible end-to-end trainable latent hierarchies in the reinforcement learning setting.

4.7 Discussion

We proposed a hierarchical deep policy network for conducting structured exploration in complex state-action spaces. We further derive a variational learning approach to train our policy in the actor-critic framework. We validate our approach on a collection of challenging multi-agent predator-prey style games, and demonstrate significant improvements over conventional baselines. We finally show that the latent variables in our hierarchical policy capture meaningful coordination between the agents, which suggests that the low-dimensional latent variable bottleneck in the hierarchical structure does indeed enable more efficient exploration.

In a sense, we studied the simplest setting that can benefit from structured exploration, in order to isolate the contribution of our work. Our hierarchical model and variational approach are a simple way to implement multi-agent coordination, and easily combine with existing actor-critic methods. Moving forward, there are many ways to expand on our work. Firstly, for complex (partial-information) environments, instead of using reactive policies with simple priors $P \sim \mathcal{N}(0, 1)$, memoryfull policies with flexible priors (Chen et al., 2016) may be needed. Secondly, within the multi-agent setting, our approach is complementary to richer forms of communication between agents. Our hierarchical structure can be interpreted as a broadcast channel, where agents are passive receivers of the message λ. Richer communication protocols could be encoded by policies with more complex inter-agent structure. Finally, there are other interesting combinatorial structures beyond multi-agent settings, and it would be interesting to investigate how to learn in those situations as well.

References

Agrawal, Shipra and Navin Goyal (2012). "Analysis of Thompson sampling for the multi-armed bandit problem". In: *Conference on Learning Theory*, pp. 39–1.

Bernstein, Daniel S (2005). "Bounded Policy Iteration for Decentralized POMDPs". In: *International Joint Conference on Artificial Intelligence*. Citeseer.

Busoniu, Lucian, Robert Babuska, and Bart De Schutter (2008). "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*.

Chalkiadakis, Georgios and Craig Boutilier (2003). "Coordination in Multiagent Reinforcement Learning: A Bayesian Approach". In: *International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '03. New York, NY, USA.

Chen, Xi et al. (2016). "Variational Lossy Autoencoder". In: *arXiv:1611.02731 [cs, stat]*. (Visited on 05/14/2017).

Daume III, Hal (2014). "Bayesian Multitask Learning with Latent Hierarchies". In: *arXiv:1408.2032 [cs, stat]*. (Visited on 05/01/2017).

Foerster, Jakob N. et al. (2016). "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *arXiv:1605.06676 [cs]*. arXiv: 1605.06676.

Furmston, T. and D. Barber (2010). "Variational methods for reinforcement learning". In: *Journal of Machine Learning Research* 9, pp. 241–248. ISSN: 1533-7928. (Visited on 05/16/2017).

Greenwald, Amy and Keith Hall (2003). "Correlated Q-Learning". In: *International Conference on Machine Learning (ICML-03)*, pp. 242–249.

Guestrin, Carlos, Michail Lagoudakis, and Ronald Parr (2002). "Coordinated reinforcement learning". In: *ICML*.

Houthooft, Rein et al. (2016). "VIME: Variational Information Maximizing Exploration". In: *arXiv:1605.09674 [cs, stat]*. (Visited on 05/16/2017).

Johnson, M. J. et al. (2016). "Composing graphical models with neural networks for structured representations and fast inference". In: *ArXiv e-prints*. arXiv: 1603.06277 [stat.ML].

Kapetanakis, Spiros and Daniel Kudenko (2002). “Reinforcement Learning of Coordination in Cooperative Multi-agent Systems”. In: *National Conference on Artificial Intelligence*. Menlo Park, CA, USA, pp. 326–331. (Visited on 05/17/2017).

Kingma, Diederik P and Max Welling (2013). “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114*.

Levine, Sergey and Vladlen Koltun (2013). “Variational Policy Search via Trajectory Optimization”. In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., pp. 207–215. (Visited on 05/15/2017).

Liu, Miao, Christopher Amato, Emily P Anesta, et al. (2016). “Learning for Decentralized Control of Multiagent Systems in Large, Partially-Observable Stochastic Environments.” In:

Liu, Miao, Christopher Amato, Xuejun Liao, et al. (2015). “Stick-Breaking Policy Learning in Dec-POMDPs.” In: *IJCAI*, pp. 2011–2018.

Mnih, Volodymyr et al. (2016). “Asynchronous Methods for Deep Reinforcement Learning”. In: *arXiv:1602.01783 [cs]*. arXiv: 1602.01783.

Neumann, Gerhard (2011). “Variational inference for policy search in changing situations”. en. In: *International Conference on Machine Learning, ICML 2011*. (Visited on 05/15/2017).

Peng, Peng et al. (2017). “Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games”. In: *arXiv:1703.10069 [cs]*. arXiv: 1703.10069. (Visited on 05/16/2017).

Ranganath, Rajesh, Dustin Tran, and David M. Blei (2015). “Hierarchical Variational Models”. In: *arXiv:1511.02386 [cs, stat]*. (Visited on 05/12/2017).

Schulman, John et al. (2015). “High-Dimensional Continuous Control Using Generalized Advantage Estimation”. In: *arXiv:1506.02438 [cs]*. (Visited on 01/16/2017).

Shoham, Yoav and Kevin Leyton-Brown (2008). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. New York, NY, USA: Cambridge University Press. ISBN: 9780521899437.

Sukhbaatar, Sainbayar, Arthur Szlam, and Rob Fergus (2016). “Learning Multiagent Communication with Backpropagation”. In: *arXiv:1605.07736 [cs]*. arXiv: 1605.07736.

Tan, Ming (1993). “Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents”. In: *International Conference on Machine Learning*, pp. 330–337.

Usunier, Nicolas et al. (2016). “Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks”. In: *arXiv:1609.02993 [cs]*. arXiv: 1609.02993.

Vlassis, Nikos et al. (2009). “Learning model-free robot control by a Monte Carlo EM algorithm”. en. In: *Autonomous Robots* 27.2, pp. 123–130. ISSN: 0929-5593, 1573-7527. (Visited on 05/15/2017).

Williams, Ronald J (1992). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3-4, pp. 229–256.

Wu, Feng, Shlomo Zilberstein, and Nicholas R Jennings (2013). “Monte-Carlo expectation maximization for decentralized POMDPs”. In: *International Joint Conference on Artificial Intelligence*. AAAI Press, pp. 397–403.

Chapter 5

IMPROVING THE ROBUSTNESS OF DEEP NEURAL NETWORKS

Zheng, Stephan et al. (2016). "Improving the robustness of deep neural networks via stability training". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. S.T.Z. participated in the conception of the project, formulated, implemented and analyzed the method, prepared the data, conducted all experiments and participated in the writing of the manuscript., pp. 4480–4488. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Zheng_Improving_the_Robustness_CVPR_2016_paper.pdf.

Dathathri, Sumanth et al. (2018). "Detecting Adversarial Examples via Neural Fingerprinting". In: *arXiv preprint arXiv:1803.03870*. S.T.Z. participated in the conception of the project, analyzed the method and experimental results, provided theoretical analyses and participated in the writing of the manuscript.

5.1 Introduction

Deep neural networks learn feature embeddings of the input data that enable state-of-the-art performance in a wide range of computer vision tasks, such as visual recognition Krizhevsky, Sutskever, and Hinton, 2012; Christian Szegedy, Liu, et al., 2015 and similar-image ranking Wang et al., 2014. Due to this success, neural networks are now routinely applied to vision tasks on large-scale *un-curated* visual datasets that, for instance, can be obtained from the Internet. Such un-curated visual datasets often contain small distortions that are undetectable to the human eye, due to the large diversity in formats, compression, and manual post-processing that are commonly applied to visual data in the wild. These lossy image processes do not change the correct ground truth labels and semantic content of the visual data, but can significantly confuse feature extractors, including deep neural networks. Namely, when presented with a pair of indistinguishable images, state-of-the-art feature extractors can produce two significantly different outputs.

In fact, current feature embeddings and class labels are not robust to a large class of small perturbations. Recently, it has become known that intentionally engineered imperceptible perturbations of the input can change the class label output by the model

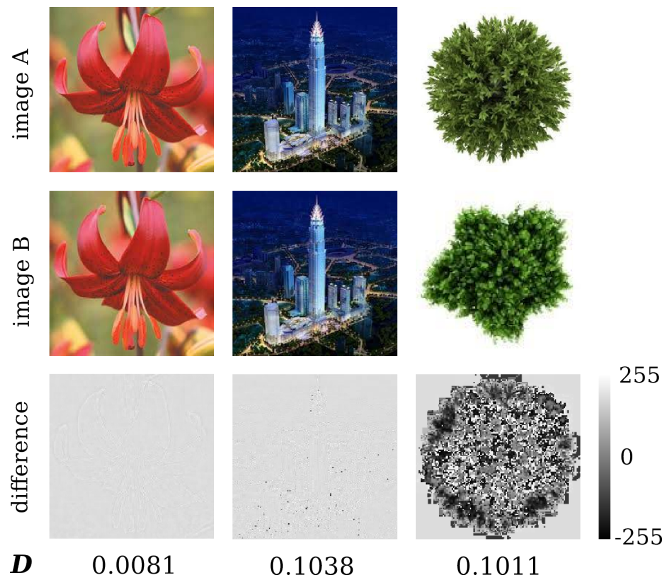


Figure 5.1: Near-duplicate images can confuse state-of-the-art neural networks due to feature embedding instability. Left and middle columns: near-duplicates with small (left) and large (middle) feature distance. Image A is the original, image B is a JPEG version at quality factor 50. Right column: a pair of dissimilar images. In each column we display the pixel-wise difference of image A and image B, and the feature distance *D* Wang et al., 2014. Because the feature distances of the middle near-duplicate pair and the dissimilar image pair are comparable, near-duplicate detection using a threshold on the feature distance will confuse the two pairs.

I. J. Goodfellow, J. Shlens, and C. Szegedy, 2014; Christian Szegedy, Zaremba, et al., 2013a **("strong adversarial perturbations")**. A scientific contribution in this chapter is the demonstration that these imperceptible perturbations can also occur without being contrived and widely occur due to compression, resizing, and cropping corruptions in visual input **("weak adversarial perturbations")**.

As such, output instability poses a significant challenge for the large-scale application of neural networks because high performance at large scale requires robust performance on noisy visual inputs. Feature instability complicates tasks such as near-duplicate detection, which is essential for large-scale image retrieval and other applications. In near-duplicate detection, the goal is to detect whether two given images are visually similar or not. When neural networks are applied to this task, there are many failure cases due to output instability. For instance, Figure 5.1 shows a case where a state-of-the-art deep network cannot distinguish a pair of near-duplicates Wang et al., 2014 and a pair of dissimilar images.

Analogously, class label instability introduces many failure cases in large-scale clas-

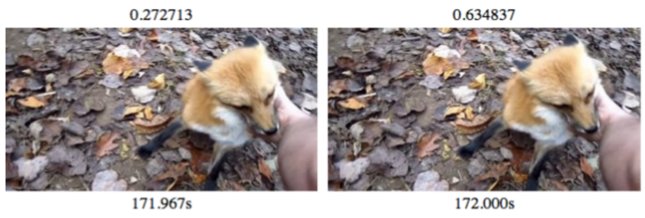


Figure 5.2: Visually similar video frames can confuse state-of-the-art classifiers: two neighboring frames are visually indistinguishable, but can lead to very different class predictions. The class score for ‘fox’ is significantly different for the left frame (27%) and right frame (63%), which causes only the fox in the right image to be correctly recognized, using any reasonable confidence threshold (e.g., > 50%).

sification and annotation. For example, unstable classifiers can classify neighboring video-frames inconsistently, as shown in Figure 5.2. In this setting, output instability can cause large changes in label scores of a state-of-the-art convolutional neural network on consecutive video-frames that are indistinguishable.

In this chapter we present two methods to make neural networks more robust.

***Stability Training*: defending against weak adversarial perturbations**

First, we showcase *Stability Training*: a general approach to stabilize machine learning models, in particular deep neural networks, and make them more robust to visual perturbations. These perturbations can be viewed as *weak adversarial perturbations*.

To this end, we introduce a fast and effective technique that makes the output of neural networks significantly more robust, *while maintaining or improving state-of-the-art performance on the original task*. Our method is fast in practice and can be used at a minimal additional computational cost.

To do so, *Stability Training* operates through two mechanisms: 1) introducing an additional *Stability Training* objective and 2) training on a large class of distorted copies of the input. The goal of this approach is to force the prediction function of the model to be more constant around the input data, while preventing underfitting on the original learning objective.

We validate our method by stabilizing state-of-the-art classification and ranking networks based on the Inception architecture Christian Szegedy, Liu, et al., 2015; Wang et al., 2014. We evaluate on three tasks: near-duplicate image detection, similar-image ranking, and image classification.

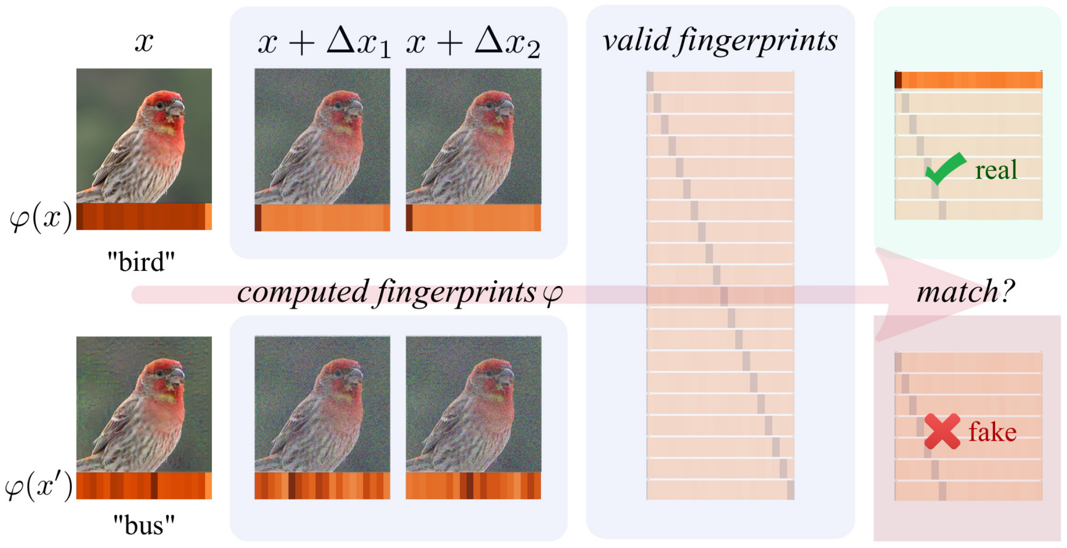


Figure 5.3: Detecting adversarial examples using *NeuralFP* with $N = 2$ fingerprints, for K-class classification. *NeuralFP* separates real data x (top) from adversarial examples $x' = x + \eta$ (bottom) by 1) adding N perturbations Δx^i, $(i = 1, \ldots, N)$ to each input, and 2) comparing model features $\varphi(x + \Delta x^i)$ with K sets of *valid fingerprints* $\varphi(x) + \Delta y^{i,j}$, $(j = 1 \ldots K)$. If a match is found, it classifies the input as "real" (x), and if not, flags it "fake" (x').

Furthermore, we show the impact of *Stability Training* by visualizing what perturbations the model has become robust to. Finally, we show that stabilized networks offer robust performance and significantly outperform unstabilized models on noisy and corrupted data.

NeuralFingerprinting: detecting strong adversarial perturbations

Second, we discuss *NeuralFingerprinting* (*NeuralFP*), a fast, secure and effective method to detect *strong adversarial perturbations*.

DNNs are also vulnerable to contrived adversarial examples: an attacker can add small perturbations to real input data, that maximally change the model's output (Christian Szegedy, Zaremba, et al., 2013b; Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, 2014). To make DNNs robust against such adversarial examples, we propose *NeuralFingerprinting* (*NeuralFP*): a fast, secure and effective method to detect adversarial examples.

The key intuition for *NeuralFP* is that we can encode fingerprint patterns into the behavior of a neural network around the input data. This pattern characterizes the network's expected behavior around real data and can thus be used to reject fake data, where the model outputs are not consistent with the expected fingerprint outputs. This process is shown in Figure 5.3. This approach is attractive as encoding fingerprints is feasible and simple to implement during training, and evaluating fingerprints is computationally cheap. Furthermore, *NeuralFP* does not require knowledge of the adversary's attack method, and differs from state-of-the-art methods (Meng and Chen, 2017; Ma et al., 2018) that use auxiliary classifiers.

We theoretically characterize the feasibility and effectiveness of *NeuralFP*, and experimentally validate that 1) *NeuralFP* achieves almost perfect detection AUC scores against state-of-the-art adversarial attacks on various datasets and 2) adaptive attackers with knowledge of the fingerprints fail to craft successful attacks. To summarize, our key contributions are:

- We present *NeuralFP*: a simple and secure method to detect adversarial examples that does not rely on knowledge of the attack mechanism.
- We formally characterize the effectiveness of *NeuralFP* for linear classification.
- We empirically show that *NeuralFP* achieves state-of-the-art near-perfect AUC-scores on detecting and separating unseen test data and the strongest known adversarial attacks.
- We empirically show that the performance of *NeuralFP* is robust to the choice of fingerprints and is effective for a wide range of choices of hyperparameters.
- We also show that *NeuralFP* can be robust even in the adaptive-whitebox-attack setting, where an adaptive attacker has knowledge of the fingerprint data.

5.2 Stability training

We now present our *Stability Training* approach, and how it can be applied to learn robust feature embeddings and class label predictions.

Stability objective

Our goal is to stabilize the output $f(x) \in \mathbb{R}^m$ of a neural network $\mathcal{N}$ against small natural perturbations to a natural image $x \in [0,1]^{w \times h}$ of size $w \times h$, where we normalize all pixel values. Intuitively, this means that we want to formulate a training objective that flattens f in a small neighborhood of any natural image x: if a perturbed copy x' is close to x, we want $f(x)$ to be close to $f(x')$, that is

$$\forall x' : d(x, x') \text{ small} \Leftrightarrow D\left(f(x), f(x')\right) \text{ small.} \tag{5.1}$$

Here d is the distance on $[0,1]^{w\times h}$ and D is an appropriate distance measure in feature space.

Given a training objective L_0 for the original task (e.g., classification, ranking), a reference input x and a perturbed copy x', we can implement the stability objective (5.1) as:

$$L(x, x'; \theta) = L_0(x; \theta) + \alpha L_{\text{stability}}(x, x'; \theta), \tag{5.2}$$

$$L_{\text{stability}}(x, x'; \theta) = D\left(f(x), f(x')\right), \tag{5.3}$$

where α controls the strength of the stability term and θ denotes the weights of the model $\mathcal{N}$. The stability objective $L_{\text{stability}}$ forces the output $f(x)$ of the model to be similar between the original x and the distorted copy x'. Note that *our approach differs from data augmentation*: we do not evaluate the original loss L on the distorted inputs x'. This is required to achieve both output stability and performance on the original task, as we explain in 5.2.

Given a training dataset $\mathcal{D}$, *Stability Training* now proceeds by finding the optimal weights θ^* for the training objective (5.2), that is, we solve

$$\theta^* = \operatorname*{argmin}_{\theta} \sum_{x_i \in \mathcal{D}, d(x_i, x_i') < \epsilon} L(x_i, x_i'; \theta). \tag{5.4}$$

To fully specify the optimization problem, we firstly need a mechanism to generate, for each training step, for each training sample x_i, a random perturbed copy x_i'. Secondly, we need to define the distance D, which is task-specific.

Sampling perturbed images x'

Sampling using Gaussian noise. During training, at every training step we need to generate perturbed versions x' of a clean image x to evaluate the stability objective (5.3).

A natural approach would be to augment the training data with examples with explicitly chosen classes of perturbation that the model should be robust against. However, it is hard to obtain general robustness in this way, as there are many classes of perturbations that cause output instability, and model robustness to one class of perturbations does not confer robustness to other classes of perturbations.

Therefore, we take a general approach and use a sampling mechanism that adds pixel-wise uncorrelated Gaussian noise ϵ to the visual input x. If k indexes the raw

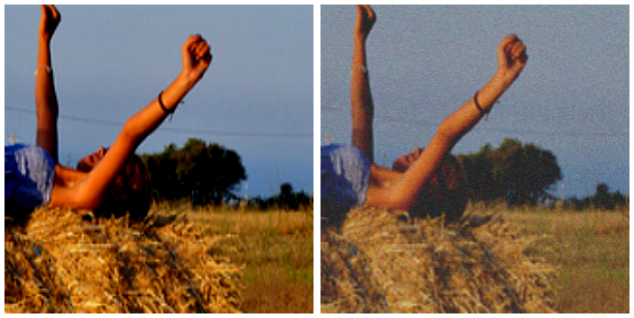


Figure 5.4: Examples of reference and distorted training images used for *Stability Training*. Left: an original image x. Right: a copy x' perturbed with pixel-wise uncorrelated Gaussian noise with $\sigma = 0.06$, in normalized pixel values. During *Stability Training*, we use dynamically sampled copies x' together with the stability loss (5.3) to flatten the prediction function f around the original image x.

Gaussian noise strength σ	0.0	0.1	0.2
Triplet ranking score @ top-30	7,312	6,300	5,065

Table 5.1: Underfitting by data augmentation with Gaussian noise on an image ranking task (higher score is better), see section 5.4 for details. The entry with $\sigma = 0.0$ is the model without data augmentation.

pixels, a new sample is given by:

$$x'_k = x_k + \epsilon_k, \;\; \epsilon_k \sim \mathcal{N}\left(0, \sigma_k^2\right), \;\; \sigma_k > 0, \tag{5.5}$$

where σ_k^2 is the variance of the Gaussian noise at pixel k. In this work, we use uniform sampling $\sigma_k = \sigma$ to produce unbiased samples of the neighborhood of x, using the variance σ^2 as a hyper-parameter to be optimized.

Preventing underfitting. Augmenting the training data by adding uncorrelated Gaussian noise can potentially simulate many types of perturbations. Training on these extra samples could in principle lead to output robustness to many classes of perturbations. However, we found that training on a dataset augmented by Gaussian perturbation leads to underfitting, as shown in Table 5.1. To prevent such underfitting, we do *not* evaluate the original loss L_0 on the perturbed images x' in the full training objective (5.2), but only evaluate the stability loss (5.3) on both x and x'. This approach differs from data augmentation, where one would evaluate L_0 on the extra training samples as well. It enables achieving both output stability and maintaining high performance on the original task, as we validate empirically.

Stability for feature embeddings

We now show how *Stability Training* can be used to obtain stable feature embeddings. In this work, we aim to learn feature embeddings for robust similar-image detection. To this end, we apply *Stability Training* in a ranking setting. The objective for similar-image ranking is to learn a feature representation $f(x)$ that detects visual image similarity Wang et al., 2014. This learning problem is modeled by considering a *ranking triplet* of images (q, p, n): a *query* image q, a *positive* image p that is visually similar to q, and a *negative* image n that is less similar to q than p is.

The objective is to learn a feature representation f that respects the triplet ranking relationship in feature space, that is,

$$D(f(q), f(p)) + g < D(f(q), f(n)), \;\; g > 0, \tag{5.6}$$

where g is a margin and D is the distance. We can learn a model for this objective by using a hinge loss:

$$L_0(q, p, n) = \max\left(0, g + D(f(q), f(p)) - D(f(q), f(n))\right). \tag{5.7}$$

In this setting, a natural choice for the similarity metric D is the L_2-distance. The stability loss is,

$$L_{\text{stability}}(x, x') = ||f(x) - f(x')||_2. \tag{5.8}$$

To make the feature representation f stable using our approach, we sample triplet images (q', p', n') close to the reference (q, p, n), by applying (5.5) to each image in the triplet.

Stability for classification

We also apply *Stability Training* in the classification setting to learn stable prediction labels for visual recognition. For this task, we model the likelihood $P(\mathbf{y}|x;\theta)$ for a labeled dataset $\left\{(x_i, \hat{\mathbf{y}}_i)\right\}_{i\in\mathcal{I}}$, where $\hat{\mathbf{y}}$ represents a vector of ground truth binary class labels and i indexes the dataset. The training objective is then to minimize the standard cross-entropy loss

$$L_0(x;\theta) = -\sum_j \hat{y}_j \log P\left(y_j|x;\theta\right), \tag{5.9}$$

where the index j runs over classes. To apply *Stability Training*, we use the KL-divergence as the distance function D:

$$L_{\text{stability}}(x, x';\theta) = -\sum_j P\left(y_j|x;\theta\right) \log P\left(y_j|x';\theta\right), \tag{5.10}$$

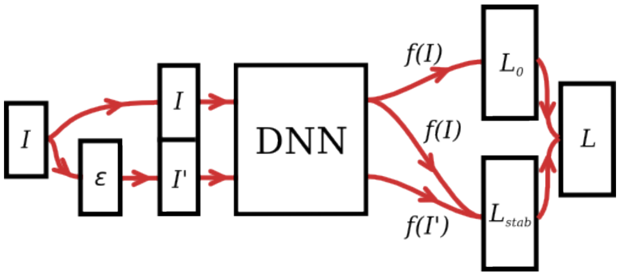


Figure 5.5: The architecture used to apply *Stability Training* to any given deep neural network. The arrows display the flow of information during the forward pass. For each input image I, a copy I' is perturbed with pixel-wise independent Gaussian noise ϵ. Both the original and perturbed version are then processed by the neural network. The task objective L_0 is only evaluated on the output $f(I)$ of the original image, while the stability loss $L_{\text{stability}}$ uses the outputs of both versions. The gradients from both L_0 and $L_{\text{stability}}$ are then combined into the final loss L and propagated back through the network. For triplet ranking training, three images are processed to compute the triplet ranking objective.

which measures the correspondence between the likelihood on the natural and perturbed inputs.

5.3 Implementation

Network

Base network. In our experiments, we use a state-of-the-art convolutional neural network architecture, the Inception network Christian Szegedy, Liu, et al., 2015 as our base architecture. Inception is formed by a deep stack of composite layers, where each composite layer output is a concatenation of outputs of convolutional and pooling layers. This network is used for the classification task and as a main component in the triplet ranking network.

Triplet ranking network. Triplet ranking loss (5.7) is used train feature embeddings for image similarity and for near duplicate image detection, similar to Wang et al., 2014. This network architecture uses an Inception module (while in Wang et al., 2014, a network like Krizhevsky, Sutskever, and Hinton, 2012 is used) to process every input image x at full resolution and uses 2 additional low-resolution towers. The outputs of these towers map into a 64-dimensional L_2-normalized embedding feature $f(x)$. These features are used for the ranking task: for each triplet of images (q, p, n), we use the features $(f(q), f(p), f(n))$ to compute the ranking loss and train the entire architecture.

Stability training. It is straightforward to implement *Stability Training* for any given neural network by adding a Gaussian perturbation sampler to generate perturbed copies of the input image x and an additional stability objective layer. This setup is depicted in Figure 5.5.

Distortion types

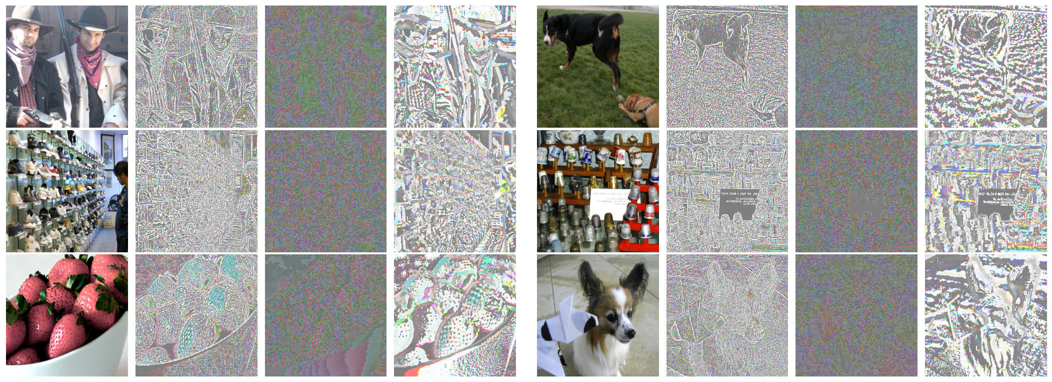


Figure 5.6: Examples of natural distortions that are introduced by common types of image processing. From left to right: original image (column 1 and 5), pixel-wise differences from the original after different forms of transformation: thumbnail downscaling to 225×225 (column 2 and 6), JPEG compression at quality level 50% (column 3 and 7) and random cropping with offset 10 (column 4 and 8). For clarity, the JPEG distortions have been up-scaled by $5\times$. Random cropping and thumbnail resizing introduce distortions that are structured and resemble the edge structure of the original image. In contrast, JPEG compression introduces more unstructured noise.

To demonstrate the robustness of our models after *Stability Training* is deployed, we evaluate the ranking, near-duplicate detection and classification performance of our stabilized models on both the original and transformed copies of the evaluation datasets. To generate the transformed copies, we apply visual perturbations that widely occur in real-world visual data and that are a result of lossy image processes.

JPEG compression. JPEG compression is a commonly used lossy compression method that introduces small artifacts in the image. The extent and intensity of these artifacts can be controlled by specifying a quality level q. In this work, we refer to this as JPEG-q.

Thumbnail resizing. Thumbnails are smaller versions of a reference image and obtained by downscaling the original image. Because convolutional neural networks use a fixed input size, both the original image and its thumbnail have to be rescaled to fit the input window. Downscaling and rescaling introduces small differences between the original and thumbnail versions of the network input. In this work we refer to this process as THUMB-A, where we downscale to a thumbnail with A pixels, preserving the aspect ratio.

Random cropping. We also evaluated the performance on perturbations coming from random crops of the original image. This means that we take large crops with window size $w' \times h'$ of the original image of size $w \times h$, using an offset $o > 0$ to define $w' = w - o, h' = h - o$. The crops are centered at random positions, with the constraint that the cropping window does not exceed the image boundaries. Due to the fixed network input size, resizing the cropped image and the original image to the input window introduces small perturbations in the visual input, analogous to thumbnail noise. We refer to this process as CROP-o, for crops with a window defined by offset o.

Optimization

To perform *Stability Training*, we solved the optimization problem (5.2) by training the network using mini-batch stochastic gradient descent with momentum, dropout Srivastava et al., 2014, RMSprop and batch normalization Ioffe and Christian Szegedy, 2015. To tune the hyper-parameters, we used a grid search, where the search ranges are displayed in Table 5.2.

Hyper-parameter	Start range	End range
Noise standard deviation σ	0.01	0.4
Regularization coefficient α	0.001	1.0
Learning rate λ	0.001	0.1

Table 5.2: Hyper-parameter search range for the *Stability Training* experiments.

As *Stability Training* requires a distorted version of the original training example, it effectively doubles the training batch-size during the forward-pass, which introduces a significant extra computational cost. To avoid this overhead, in our experiments we first trained the network on the original objective $L_0(x;\theta)$ only and started *Stability Training* with $L(x, x';\theta)$ only in the fine-tuning phase. Additionally, when applying *Stability Training*, we only fine-tuned the final fully-connected layers of the network. Experiments indicate that this approach leads to the same model performance as applying *Stability Training* right from the beginning and training the whole network during *Stability Training*.

5.4 Evaluation of Stability Training

Here we present experimental results to validate our *Stability Training* method and characterize stabilized models.

- Firstly, we evaluate stabilized features on near-duplicate detection and similar-image ranking tasks.
- Secondly, we validate our approach of stabilizing classifiers on the ImageNet classification task.

We use training data as in Wang et al., 2014 to train the feature embeddings for near-duplicate detection and similar-image ranking. For the classification task, training data from ImageNet are used.

Near-duplicate detection

Detection criterion. We used our stabilized ranking feature to perform near-duplicate detection. To do so, we define the detection criterion as follows: given an image pair (a, b), we say that

$$a, b \text{ are near-duplicates} \iff ||f(a) - f(b)||_2 < T, \tag{5.11}$$

where T is the near-duplicate detection threshold.

Near-duplicate evaluation dataset. For our experiments, we generated an image-pair dataset with two parts: one set of pairs of near-duplicate images (true positives) and a set of dissimilar images (true negatives).

We constructed the near-duplicate dataset by collecting 650,000 images from randomly chosen queries on Google Image Search. In this way, we obtained a representative sample of un-curated images. We then combined every image with a copy perturbed with the distortion(s) from section 5.3 to construct near-duplicate pairs. For the set of dissimilar images, we collected 900,000 random image pairs from the top 30 Image Search results for 900,000 random search queries, where the images in each pair come from the *same* search query.

Experimental results

Precision-recall performance. To analyze the detection performance of the stabilized features, we report the near-duplicate precision-recall values by varying the detection threshold in (5.11). Our results are summarized in Figure 5.7. The

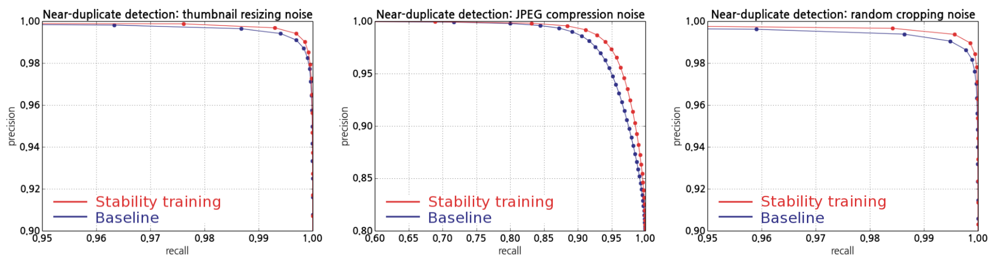


Figure 5.7: Precision-recall performance for near-duplicate detection using feature distance thresholding on deep ranking features. We compare Inception-based deep ranking features (blue), and the same features with *Stability Training* applied (red). Every graph shows the performance using near-duplicates generated through different distortions. Left: THUMB-50k. Middle: JPEG-50. Right: CROP-10. Across the three near-duplicate tasks, the stabilized model significantly improves the near-duplicate detection precision over the baseline model.

stabilized deep ranking features outperform the baseline features for all three types of distortions, for all levels of fixed recall or fixed precision. Although the baseline features already offer very high performance in both precision and recall on the near-duplicate detection task, the stabilized features significantly improve precision across the board. For instance, recall increases by 1.0% at 99.5% precision for thumbnail near-duplicates, and increases by 3.0% at 98% precision for JPEG near-duplicates. This improved performance is due to the improved robustness of the stabilized features, which enables them to correctly detect near-duplicate pairs that were confused with dissimilar image pairs by the baseline features, as illustrated in Figure 5.1.

Feature distance distribution. To analyze the robustness of the stabilized features, we show the distribution of the feature distance $D(f(x), f(x'))$ for the near-duplicate evaluation dataset in Figure 5.8, for both the baseline and stabilized deep ranking feature. Stability training significantly increases the feature robustness, as the distribution of feature distances becomes more concentrated towards 0. For instance, for the original feature 76% of near-duplicate image pairs has feature distance smaller than 0.1, whereas this is 86% for the stabilized feature, i.e., the stabilized feature is significantly more similar for near-duplicate images.

Stabilized feature distance. We also present our qualitative results to visualize the improvements of the stabilized features over the original features. In Figure 5.9 we show pairs of images and their JPEG versions that were confusing for the un-stabilized features, i.e., that lay far apart in feature space, but whose stabilized features are significantly more close. This means that they are correctly detected as

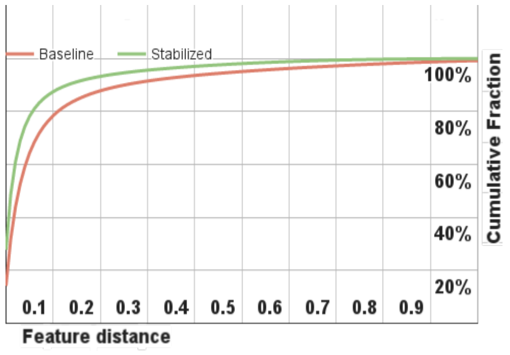


Figure 5.8: Cumulative distribution of the deep ranking feature distance $D(f(x_i), f(x_i')) = ||f(x_i) - f(x_i')||_2$ for near-duplicate pairs (x_i, x_i'). Red: baseline features, 76% of distribution < 0.1. Green: stabilized features using *Stability Training* with $\alpha = 0.1, \sigma = 0.2$, 86% of distribution < 0.1. The feature distances are computed over a dataset of 650,000 near-duplicate image pairs (reference image and a JPEG-50 version). Applying *Stability Training* makes the distribution of $D(f(x), f(x'))$ more concentrated towards 0 and hence makes the feature f significantly more stable.

near-duplicates for much more aggressive, that is, lower detection thresholds by the stabilized feature, whereas the original feature easily confuses these as dissimilar images. Consistent with the intuition that Gaussian noise applies a wide range of types of perturbations, we see improved performance for a wide range of perturbation types. Importantly, this includes even localized, structured perturbations that do not resemble a typical Gaussian noise sample.

Similar image ranking

The stabilized deep ranking features (see section 5.2) are evaluated on the similar image ranking task. Hand-labeled triplets from Wang et al., 2014[1] are used as evaluation data. There are 14,000 such triplets. The ranking score-at-top-K ($K = 30$) is used as evaluation metric. The ranking score-at-top-K is defined as

$$\begin{aligned}&\text{ranking score @top-}K = \\ &\text{\# correctly ranked triplets} - \text{\# incorrectly ranked triplets},\end{aligned} \tag{5.12}$$

where only triplets whose positive or negative image occurs among the closest K results from the query image are considered. This metric measures the ranking performance on the K most relevant results of the query image. We use this evaluation metric because it reflects better the performance of similarity models in practical image retrieval systems as users pay most of their attentions to the results on the first few pages.

[1] https://sites.google.com/site/imagesimilaritydata/.

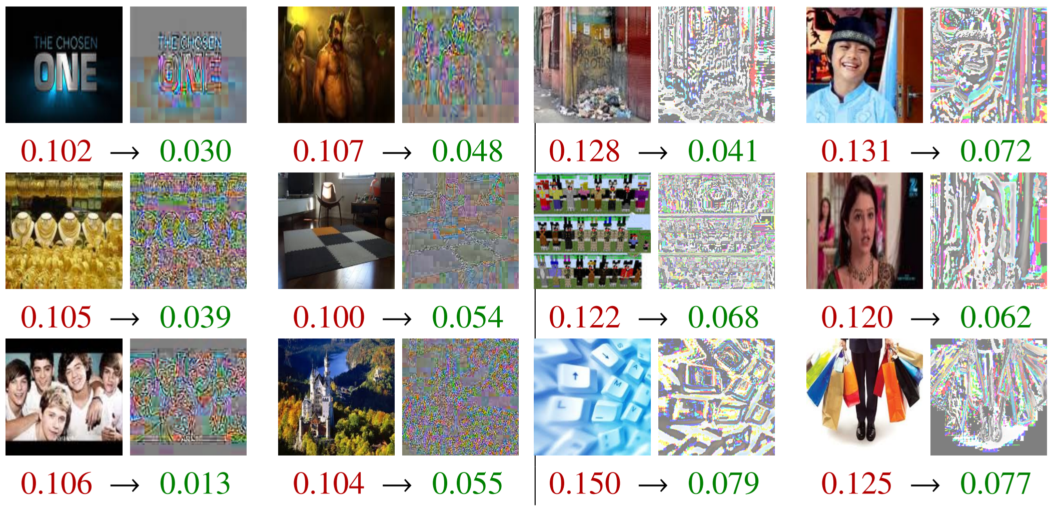


Figure 5.9: Examples of near-duplicate image pairs that are robustly recognized as near-duplicates by stabilized features (small feature distance), but easily confuse un-stabilized features (large feature distance). Left group: using JPEG-50 compression corruptions. Right group: random cropping CROP-10 corruptions. For each image pair, we display the reference image x, the difference with its corrupted copy $x - x'$, and the distance in feature space $D(f(x), f(x'))$ for the un-stabilized (red) and stabilized features (green).

Experimental results.

Our results for triplet ranking are displayed in Table 5.3. The results show that applying *Stability Training* improves the ranking score on both the original and transformed versions of the evaluation dataset. The ranking performance of the baseline model degrades on all distorted versions of the original dataset, showing that it is not robust to the input distortions. In contrast, the stabilized network achieves ranking scores that are higher than the ranking score of the baseline model on the *original* dataset.

Image classification

In the classification setting, we validated *Stability Training* on the ImageNet classification task Russakovsky et al., 2015, using the Inception network Christian Szegedy, Liu, et al., 2015. We used the full classification dataset, which covers 1,000 classes and contains 1.2 million images, where 50,000 are used for validation. We evaluated the classification precision on both the original and a JPEG-50 version of the validation set. Our benchmark results are in Table 5.4.

Applying *Stability Training* to the Inception network makes the class predictions of the network more robust to input distortions. On the original dataset, both the

Distortion	Deep ranking	Deep ranking + ST
Original	7,312	**7,368**
JPEG-50	7,286	**7,360**
THUMB-30k	7,160	**7,172**
CROP-10	7,298	**7,322**

Table 5.3: Ranking score @top-30 for the deep ranking network with and without *Stability Training* (higher is better) on distorted image data. Stability training increases ranking performance over the baseline on all versions of the evaluation dataset. We do not report precision scores, as in Wang et al., 2014, as the ranking score @top-30 agrees more with human perception of practical similar image retrieval.

Precision @top-5	Original	JPEG-50	JPEG-10
Szegedy et al Christian Szegedy, Liu, et al., 2015	93.3%		
Inception	**93.9%**	92.4%	83.0%
Stability training	93.6%	**92.7%**	**88.3%**
Precision @top-1			
Inception	77.8%	75.1%	61.1%
Stability training	**77.9%**	**75.7%**	**67.9%**

Table 5.4: Classification evaluation performance of Inception with *Stability Training*, evaluated on the original and JPEG versions of ImageNet. Both networks give similar state-of-the-art performance on the original evaluation dataset (note that the performance difference on the original dataset is within the statistical error of 0.3% Russakovsky et al., 2015). However, the stabilized network is significantly more robust and outperforms the baseline on the distorted data.

baseline and stabilized network achieve state-of-the-art performance. However, the stabilized model achieves higher precision on the distorted evaluation datasets, as the performance degrades more significantly for the baseline model than for the stabilized model. For high distortion levels, this gap grows to 5% to 6% in top-1 and top-5 precision.

Robust classification on noisy data. We also evaluated the effectiveness of *Stability Training* on the classification performance of Inception on the ImageNet evaluation dataset with increasing JPEG corruption. In this experiment, we collected the precision @top-1 scores at convergence for a range of the training hyper-parameters: the regularization coefficient α and noise standard deviation σ. A summary of these results is displayed in Figure 5.10.

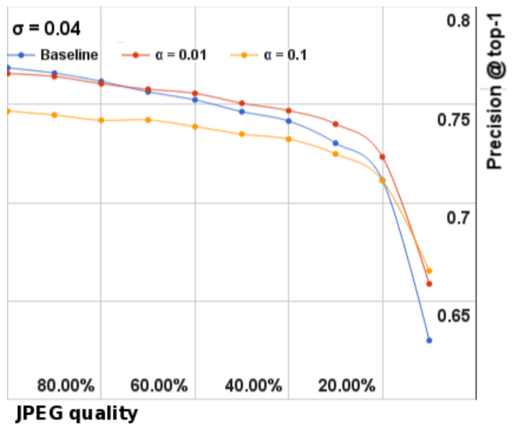


Figure 5.10: A comparison of the precision @ top-1 performance on the ImageNet classification task for different *Stability Training* hyper-parameters α, using JPEG compressed versions of the evaluation dataset at decreasing quality levels, using a fixed $\sigma = 0.04$. At the highest JPEG quality level, the baseline and stabilized models perform comparably. However, as the quality level decreases, the stabilized model starts to significantly outperform the baseline model.

At the highest JPEG quality level, the performance of the baseline and stabilized models are comparable, as the visual distortions are small. However, as the JPEG distortions become stronger, the stabilized model starts to significantly outperform the baseline model. This qualitative behavior is visible for a wide range of hyper-parameters, for instance, using $\alpha = 0.01$ and $\sigma = 0.04$ results in better performance already below the 80% quality level.

5.5 Fingerprinting for Detection of Adversarial Examples

We now consider how to defend against *strong adversarial examples* by detecting such attacks with *neural fingerprinting*.

We consider supervised classification, where we aim to learn a model $f(x;\theta)$ from labeled data $\left\{(x^i, y^{*i})\right\}_{i\in\mathcal{I}}$, where $x \in \mathbb{R}^l$ and y is a 1-hot label vector $y \in \{0,1\}^K$ (K classes) corresponding to y^*. The model f predicts class probabilities $P(y|x;\theta)$:

$$f(x;\theta)_j = \frac{\exp h(x;\theta)_j}{\sum_l \exp h(x;\theta)_l}, \tag{5.13}$$

with logits $h(x;\theta) \in \mathbb{R}^K$ and can be learned via a loss function $L(x,y;\theta)$, e.g. cross-entropy loss. This data is generated by sampling from a *data-generating distribution* $P_{data}(x,y)$. This distribution characterizes "real" data (x,y), for which $P_{data}(x,y) > 0$. In this spirit, we define *fake* data to be inputs that are not from the underlying data-distribution from which the training data was sampled from, i.e.

Algorithm 1 *NeuralFP*

```
1: Input: example $x$, comparison function $D$ (see Eqn 5.17), threshold $\tau > 0$, $\xi^{i,j}$
   (see Eqn 5.16), model $f$.
2: Output: accept / reject.
3: if $\exists j : D(x, f, \xi^{i,j}) \leq \tau$ then
4:     Return: accept # x is real
5: else
6:     Return: reject # x is fake
7: end if
```

inputs where the model prediction cannot be relied on. This includes outliers and adversarially crafted inputs. Formally, for a distribution $P_{data}(x, y)$, we define *fake* data as input-label pairs (x', y') for which $P_{data}(x', y') = 0$.

Recently, there has been increased focus on adversarial attacks that produce small perturbations, exploiting the behavior of a neural network at an input point x. A (carefully) crafted adversarial perturbation η causes a large change in model output, i.e. for $\delta, \rho > 0$:

$$||\eta|| \leq \delta, \quad ||f(x + \eta) - f(x)|| > \rho, \tag{5.14}$$

such that the class predicted by the model changes:

$$k' = \operatorname*{argmax}_j f(x + \eta)_j \neq \operatorname*{argmax}_j f(x)_j = k. \tag{5.15}$$

For instance, the Fast-Gradient Sign Method (Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, 2014) perturbs along the gradient: $\eta \propto \left(\text{sign } \frac{\partial L(x,y;\theta)}{\partial x}\right)$.

Neural Fingerprinting. We propose *NeuralFP* as a defense against adversarial examples: a method that detects whether an input-output pair $(x, \hat{y})$ is consistent with the data distribution ("real"), or is adversarial ("fake"). This algorithm is summarized in Algorithm 1 and Figure 5.3. The key idea of *NeuralFP* is to detect adversarial inputs by checking if the network's sensitivity in *specific directions* (Δx) around x matches a set of *output-perturbations* (Δy) that can be *chosen by the defender*. These chosen output-changes are encoded into the network during training, and can then be used to detect *fake* (outlier) inputs.

We will discuss strategies to choose the fingerprints $(\Delta x, \Delta y)$ in Section 5.5.

Notation. Formally, we define a fingerprint ξ as the tuple $\xi \triangleq (\Delta x, \Delta y)$. For K-class classifcation, we define a set of fingerprints:

$$\xi^{i,j} = (\Delta x^i, \Delta y^{i,j}), \;\; i = 1, \dots, N, j = 1, \dots, K, \tag{5.16}$$

where $\xi^{i,j}$ is the i^{th} fingerprint for class j. Here, the Δx^i ($\Delta y^{i,j}$) are input (output) perturbations that are chosen by the defender. Note that across classes $j = 1, 2 \dots, K$, we use the same directions Δx^i, and that $\Delta y^{i,j}$ can be either discrete or continuous depending on the signature of $f(x;\theta)$. To characterize sensitivity, we define $F(x, \Delta x^i)$ to measure the change in model output. A simple choice could be $F(x, \Delta x^i) = f(x + \Delta x^i) - f(x)$ (although we will use variations hereafter). To compare $F(x, \Delta x^i)$ with the reference output-perturbation Δy^i, we use the comparison function D:

$$D(x, f, \xi^{i,j}) \triangleq \frac{1}{N} \sum_{i=1}^{N} ||F(x, \Delta x^i) - \Delta y^{i,j}||_2. \tag{5.17}$$

Encoding Fingerprints. Once a defender has constructed a set of desired fingerprints (5.16), the chosen fingerprints can be embedded into the network's response by adding a fingerprint regression loss during training. Given a classification model (5.13) with logits $h(x;\theta) \in \mathbb{R}^K$, the fingerprint loss is:

$$L_{\text{fp}}(x, y, \xi; \theta) = \frac{1}{N} \sum_{i=1}^{N} ||F(x, \Delta x^i) - \Delta y^{i,k}||_2^2, \tag{5.18}$$

where k is the ground truth class for example x and $\Delta y^{i,k}$ are the fingerprint outputs. Note that we *only train on the fingerprints for the ground truth class*. The total training objective then is:

$$\min_{\theta} \sum_{(x,y)} L_0(x, y; \theta) + \alpha L_{\text{fp}}(x, y, \xi; \theta), \tag{5.19}$$

where L_0 is a loss function for the task (e.g. cross-entropy loss for classification) and α a positive scalar. In the hereafter we will use $\alpha = 1$, but in practice, we choose α such that it balances the task and fingerprint losses. The procedure trains the model so that the function D has low values around the data-distribution. We then exploit this characterization to detect for outliers using D.

Testing for Outliers *NeuralFP* classifies a new input x' as *real* if the change in model output is close to the $\Delta y^{i,j}$ *for some class* j, for all i. Here, we use a comparison

θ	`NFP`	
$\times$	$\times$	blackbox-attack
✓	$\times$	partial-whitebox-attack
✓	✓	adaptive-whitebox-attack

Table 5.5: Threat models: attacker knows θ and / or `NFP`.

function D and threshold $\tau > 0$ to define the level of agreement required, i.e. we declare x' *real* when D is below a threshold τ.

$$x' \text{ is real} \Leftrightarrow \exists j : D(x, f, \xi^{\cdot,j}) \leq \tau. \tag{5.20}$$

Hence, the *NeuralFP* test is defined by the data: `NFP` $= (\xi, D, \tau)$.

Complexity. Extra computation comes from Algorithm 1 which requires $O(NK)$ forward passes to compute the differences $F(x, \Delta x^i)$. A straightforward implementation is to check (5.20) iteratively for all classes, and stop whenever an agreement is seen or all classes have been exhausted. However, this can be parallelized and performed in minibatches for real-time applications.

Threat Model Analysis

We study *NeuralFP* under various threat models (Table 5.5), where the attacker has varying levels of knowledge of `NFP` and model $f(x;\theta)$.

In the **partial-whitebox-attack** (PWA) setting, the attacker has access to θ, can query $f(x;\theta)$ and its derivatives, *but does not know `NFP`*. We evaluate under this setting in Section 5.6. Most defenses reported in Section 5.6 study attacks under this threat model.

If the attacker can either reverse engineer or has access to the fingerprints, he can mount a adapative-whitebox-attack (discussed below) that adapts to the fingerprints and tries to fool the detector. Reverse engineering the `NFP` by brute-force search can be combinatorially hard. To see this, consider a simple setting where only the $\Delta y^{i,j}$ are unknown, and that the attacker knows that each fingerprint is discrete, i.e. each component $\Delta y_k^{i,j} = \pm 1$. Then the attacker would have to search over combinatorially ($O(2^{NK})$) many Δy to find the subset of Δy that satisfy the detection criterion in equation (5.20). Further, smaller τs reduce the volume of inputs accepted as *real*.

In the **adaptive-whitebox-attack** (AWA) setting, the adversary has perfect knowledge of `NFP` in addition to the information available under PWA. While a key part of the

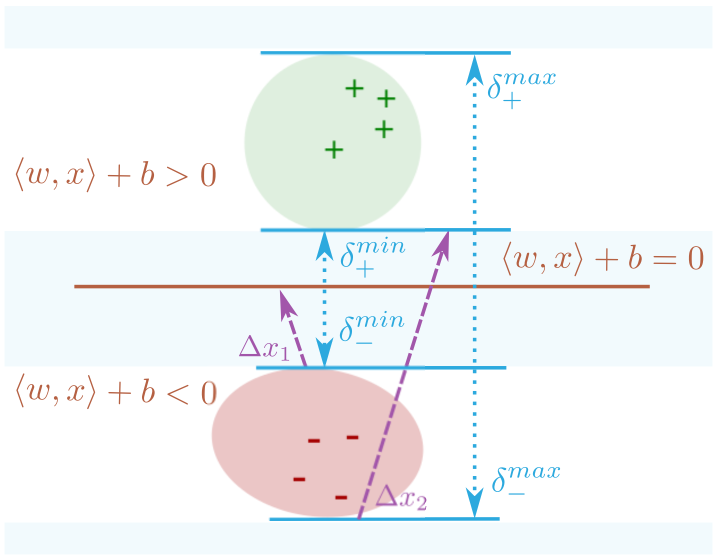


Figure 5.11: Geometry of fingerprints for SVMs with linearly separable data. Let $d(x)$ be the distance of x to the decision boundary (see Thm 1). $\delta_\pm^{max}$ ($\delta_\pm^{min}$) denote the maximal (minimal) distances of the positive (x_+) and negative (x_-) examples to the separating hyperplane $\langle w, x\rangle + b = 0$. The fingerprint Δx^1 with $\langle \Delta x^1, e\rangle = \delta_-^{min}$ will have $f(x_- + \Delta x) < 0$ and $f(x_-) < 0$ for all x_- in the data distribution (red region). Hence, Δx^1 will flag all x' in the regions $-\delta_-^{min} < d(x') < 0$ as "fake" ($d(x')$ is the signed distance of x' from the boundary), since for those x' it will *always* see a change in predicted class. Similarly, Δx^2 with $\langle \Delta x^2, e\rangle = \delta_-^{max}$ always sees a class change for real x_-, thus flagging all x' with $d(x') < -\delta_-^{max}$ as "fake".

defense relies on the fingerprints being hidden from the attacker, we also study the vulnerability of the defense when the attacker has access to `NFP`. For this setting, first, we characterize the region of inputs that will be classified as "real" for a given set of fingerprints $\xi^{i,j}$ and the correspondence with the support of the data distribution $P_{data}(x, y)$ (see Theorem 1) for binary classification with linear models. Second, we empirically show the robustness of *NeuralFP* (for DNNs) to adaptive attacks in Section 5.6. Ma et al., 2018; Xu, Evans, and Qi, 2018 are other recent defenses that investigate the robustness to such adaptive attacks.

The **blackbox-attack setting** is the weakest setting, where the adversary has no knowledge of the model parameters or `NFP`. We evaluate *NeuralFP* under this setting in Appedix 5.6. Additionally, we define the notion of **whitebox-defense** (the defender is aware of the attack mechanism) and **blackbox-defense** (defender has no knowledge about attacker).

Choosing Fingerprints and Characterizing Model Response – Linear Models

We first analyze *NeuralFP* for SVMs – binary classification with a linear model

$$f(x) = \langle w, x\rangle + b, \quad \hat{y} = \text{sign } f(x) \in \{-1, 1\}, \tag{5.21}$$

on inputs $x \in \mathbb{R}^n$, where $n \gg 1$ (e.g. $n = 900$ for MNIST). The SVM defines a hyperplane $f(x) = 0$ to separate positive ($\hat{y} = +1$) from negative examples ($\hat{y} = -1$). We will assume that the positive and negative examples are linearly separable by a hyperplane defined by a normal $e = \frac{w}{||w||}$, $||e|| = 1$. We define the minimal and maximal distance from the examples to the hyperplane along e as:

$$\delta_{\pm}^{min} = \min_{a:y^a=\pm 1} |\langle x^a, e\rangle|, \ \delta_{\pm}^{max} = \max_{a:y^a=\pm 1} |\langle x^a, e\rangle|. \tag{5.22}$$

In this setting, the set of x classified as "real" by fingerprints is determined by the geometry of $f(x)$. Here, for detection we measure the *exact change in predicted class* using

$$F(x, \Delta x^i) = \text{sign}\left(\langle w, x + \Delta x^i\rangle + b\right) - \text{sign}\left(\langle w, x\rangle + b\right) \in \{-2, 0, 2\}, \tag{5.23}$$

use $\tau = 0$ and $D(x, \Delta x)$ as in (5.17). The following Theorem characterizes fingerprints for SVMs:

Theorem 1 (Fingerprint Detection for SVM)**.** *For an SVM with* $e = \frac{w}{||w||}$ *and separable data,*

$$(\Delta x^1 = \delta_{-}^{min} e, \Delta y^{1,-} = 0), \tag{5.24}$$
$$(\Delta x^2 = \delta_{-}^{max} e, \Delta y^{2,-} = +2), \tag{5.25}$$
$$(\Delta x^3 = -\delta_{+}^{max} e, \Delta y^{3,+} = -2), \tag{5.26}$$
$$(\Delta x^4 = -\delta_{+}^{min} e, \Delta y^{4,+} = 0). \tag{5.27}$$

will detect adversarial inputs $x' = x_{\pm} + \eta$ *as "fake" for which one the following hold:*

$$d(x') > \delta_{+}^{max}, \quad 0 < d(x') < \delta_{+}^{min}, \quad d(x') < -\delta_{-}^{max}, \quad -\delta_{-}^{min} < d(x') < 0, \tag{5.28}$$

where $d(x') = \frac{\langle x', w\rangle + b}{||w||}$ *represents the signed distance of* x' *from the separating hyperplane. This choice of fingerprints is optimal, i.e., excludes the largest area of "fake" data.*

The proof for two fingerprints is shown in Figure 5.11. The full proof now follows.

Proof. Consider any perturbation $\eta = \lambda e$ that is positively aligned with w, and has $\langle \eta, e\rangle = \delta_{-}^{min}$. Then for any negative example $(x_{-}, -1)$ (except for the support vectors that lie exactly δ_-^{min} from the hyperplane), adding the perturbation η does not change the class prediction:

$$\text{sign } f(x_-) = -1, \quad \text{sign } f(x_- - \eta) = -1. \tag{5.29}$$

The fingerprint in (5.24) is an example of such an η. However, if λ is large enough, that is:

$$\langle \eta, e \rangle = \delta_-^{max}, \tag{5.30}$$

(e.g. the fingerprint in (5.25)), for *all* negative examples $(x_-, -1)$ the class prediction will *always* change (except for the x_- that lie exactly δ_-^{max} from the hyperplane):

$$\text{sign } f(x_-) = -1, \quad \text{sign } f(x_- + \eta) = +1, \tag{5.31}$$

Note that if η has a component smaller (or larger) than $\delta_\pm^{min}$, it will exclude *fewer* (more) examples, e.g. those that lie closer to (farther from) the hyperplane. Similar observations hold for fingerprints (5.26) and (5.27) and the positive examples x_+. Hence, it follows that for any x that lies too close to the hyperplane (closer than $\delta_\pm^{min}$), or too far (farther than $\delta_\pm^{max}$), the model output after adding the four fingerprints will never perfectly correspond to their behavior on examples x from the data distribution. For instance, for any x that is closer than δ_+^{min} to the hyperplane, (5.27) will always cause a change in class, while none was expected. Similar observations hold for the other regions in (5.28). Since the SVM is translation invariant parallel to the hyperplane, the fingerprints can only distinguish examples based on their distance perpendicular to the hyperplane. Hence, this choice of λs is optimal. $\square$

Theorem 1 by itself *does not* prevent attacks parallel to the decision boundary; an adversary could in principle add a (large) perturbation η that pushes a negative example x_- across the decision boundary to a region where the data distribution $P_{data}(x_- + \eta, y) = 0$, but is classified as positive. However, this can be prevented by checking the distance to the nearest example in the dataset.

Choosing Fingerprints and Characterizing Model Response – Nonlinear Models

In contrast to the linear setting, *NeuralFP* utilizes a *softer* notion of fingerprint matching by checking whether the model outputs match (a pattern of) *changes in normalized-logits*. Specifically, for K-class classification $f(x;\theta)$ is a distribution

$P(y|x;\theta)$ ($y \in \mathbb{R}^K$) with logits $h(x;\theta)$, F is defined as:

$$F(x, \Delta x^i) \triangleq \varphi(x + \Delta x^i) - \varphi(x), \quad \varphi(x) \triangleq \frac{h(x;\theta)}{||h(x;\theta)||}, \tag{5.32}$$

where φ are the normalized logits. The logits are normalized so the DNN does not fit the Δy by making the weights arbitrarily large. Here, we use $D(x, \Delta x^i)$ as in (5.17). Note that here $\Delta y^{i,j} \in \mathbb{R}^K$.

However, for nonlinear models (e.g., DNNs), the best fingerprint choice is not obvious, as it is hard to prove optimality. To overcome this, we propose a straightforward extension from the linear case, using randomly sampled fingerprints. Randomization minimizes structural assumptions that might make *NeuralFP* exploitable. We empirically found that this still provides effective detection.

Choosing Δx For all experiments, we sampled the fingerprint directions Δx^i from a uniform distribution ($\Delta x^i \sim \mathcal{U}(-\varepsilon, \varepsilon)^l$), where l is the input dimension, and each pixel is uniformly randomly sampled in the range $[-\varepsilon, \varepsilon]$. Our experiments (see Figure 5.15) suggest that *NeuralFP* is *not sensitive* to the random values sampled. This also suggests that the Δx^i could be chosen based on a different distribution/alternative approaches.

Choosing Δy For our experiments, we choose the Δy so that the normalized-logit of the true class either increases or decreases along the Δx^i (analogous to the linear case). For e.g., for a 10-class classification task, if x is in class k we set

$$\Delta y^k_{l \neq k} = -0.235, \quad \Delta y^k_{l=k} = 0.73, \quad k = 1, \ldots, 10, \tag{5.33}$$

, with $||\Delta y|| = 1$. More generally, we evaluated using random $\Delta y^{i,j}$: for each Δx^i:

$$\Delta y^{i,k}_{l \neq k} = -0.235(2p - 1), \quad \Delta y^{i,k}_{l=k} = 0.7(2p - 1), \quad p \sim \text{Bern}\left(\frac{1}{2}\right). \tag{5.34}$$

Here $p \in \{0, 1\}$ is a Bernoulli random variable, that is resampled for each Δx^i, For this `NFP`, we achieved AUC-ROCs of $> 95\%$ across attacks with $N = 30, \varepsilon = 0.05$, without extensive tuning. This suggests that *NeuralFP* is effective with (randomly sampled) fingerprints as well.

Visualizing Fingerprints. To understand the behavior of fingerprinted non-linear models we trained a neural network (2 hidden layers with 200 ReLU nodes each) to distinguish between two Gaussian balls in a 2D space (Figure 5.12, left). Without

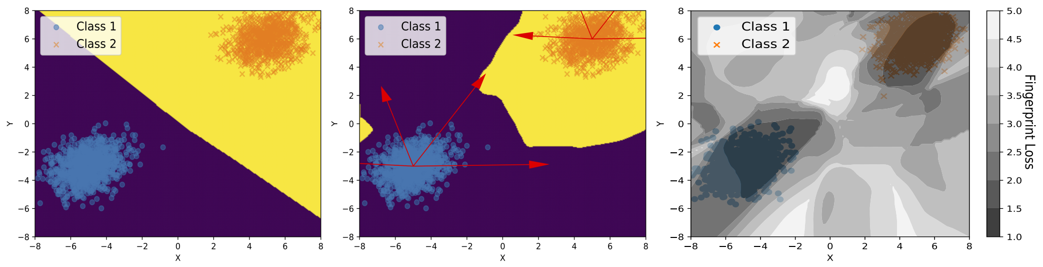


Figure 5.12: Left: decision boundary without fingerprints. Center: with fingerprints, red arrows indicate fingerprint-directions. The decision boundary is significantly more non-linear. Right: contour plot of fingerprint loss. *NeuralFP* detects dark regions as “real”, while lighter regions are “fake” (tunable through τ). Fingerprinting create valleys of low-loss delineating the data-distribution from outliers.

fingerprints, the model learns an almost linear boundary separating the two balls (compare with Figure 5.11). When we train to encode the fingerprints (5.33), we observe that *NeuralFP* causes the model to learn a highly non-linear boundary (Figure 5.12, center) forming pockets of low fingerprint-loss characterizing the data-distribution (Figure 5.12,right). In this simple setting, *NeuralFP* learns to delineate the data-distribution, where the darker regions are accepted as “real” and the rest is rejected as “fake”. Figure 5.13 further confirms this intuition where adversarial images (“fake”, outliers) have high fingerprint-loss.

5.6 Evaluating the Detection of Adversarial Attacks

We now empirically validate the effectiveness of *NeuralFP*, as well as analyze the behavior and robustness of *NeuralFP*. First, we evaluate *NeuralFP* on distinguishing between unseen *real* images and adversarial images, for data and models of varying scales. We also study its sensitivity to varying hyperparameters. Lastly, we study the robustness of *NeuralFP* under an adaptive-whitebox-attack.

Detection Performance. We report the AUC-ROC of *NeuralFP* on MNIST, CIFAR-10 and MiniImagenet-20 against four state-of-the art **untargeted**[2] partial-whitebox attacks (Table 5.6):

- *Fast Gradient Method* (FGSM) (Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy, 2014) and *Basic Iterative Method* (BIM) (Kurakin, Ian J. Goodfellow, and Bengio, 2016) are both gradient based attacks with BIM

[2]Untargeted attacks can cause misclassification to any class; harder to defend against than a targeted attack.

Data	Attack	Success Rate	Bound on Adversarial Perturbation η
MNIST	FGSM	88.13%	$\|\|\eta\|\|_\infty \leq 0.4$
	BIM-a	100%	$\|\|\eta\|\|_\infty \leq 0.4$
	BIM-b	100%	$\|\|\eta\|\|_\infty \leq 0.4$
CIFAR	FGSM	89.61 %	$\|\|\eta\|\|_\infty \leq 0.05$
	BIM-a	100%	$\|\|\eta\|\|_\infty \leq 0.05$
	BIM-b	100%	$\|\|\eta\|\|_\infty \leq 0.05$
MiniImagenet	FGSM	100%	$\|\|\eta\|\|_\infty \leq 16/255$
	BIM-b	100%	$\|\|\eta\|\|_\infty \leq 16/255$

Table 5.6: Parameters and success rates of evaluated attacks for different datasets. CW-L_2 and JSMA attacks are unbounded. The bounds are relative to images with pixel intensities in [0, 1].

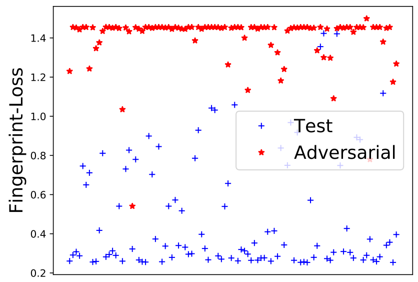


Figure 5.13: Fingerprint losses on 100 random test (blue) and adversarial (red) CIFAR-10 images. We see a clear separation in loss, illustrating that *NeuralFP* is effective across many thresholds τ.

being an iterative variant of FGSM. We consider both BIM-a (iterates until misclassification has been achieved) and BIM-b (iterates 50 times).

- *Jacobian-based Saliency Map Attack* (JSMA) (Papernot et al., 2015) perturbs pixels using a saliency map.

- *Carlini-Wagner Attack* (CW-L_2): an optimization-based attack, is one of the strongest known attacks (N. Carlini and D. Wagner, 2016; Nicholas Carlini and D. A. Wagner, 2017a), and optimizes to minimize the perturbation needed for misclassification.

Following Ma et al., 2018, for each dataset we consider a randomly sampled *pre-test*-set of 1328 test-set images, and discard misclassified pre-test images. For the *test-set* of remaining images, we generate adversarial perturbations by applying each of the above mentioned attacks. We report AUC-ROC on sets composed in equal measures of the *test-set* and *test-set* adversarial examples. The AUC-ROC is computed by varying the threshold τ.

Experimental Setup

For the CW-adaptive attack, we used the published code from `https://github.com/carlini/nn_robust_attacks`. Code for the other attacks was obtained from the paper (Ma et al., 2018).

Baselines. The baselines are LID Ma et al., 2018, a recent detection based defense; KD; BU Feinman et al., 2017; all trained on FGSM, as in Ma et al., 2018. We use published code for the attacks and code from Ma et al., 2018 for the baselines.

Data	Method	FGSM	JSMA	BIM-a	BIM-b	CW-L_2
MNIST	LID	99.68	96.36	99.05	99.72	98.66
	KD	81.84	66.69	99.39	99.84	96.94
	BU	27.21	12.27	6.55	23.30	19.09
	KD+BU	82.93	47.33	95.98	99.82	85.68
	NeuralFP	**100.0**	**99.97**	**99.94**	**99.98**	**99.74**
CIFAR-10	LID	82.38	89.93	82.51	91.61	93.32
	KD	62.76	84.54	69.08	89.66	90.77
	BU	71.73	84.95	82.23	3.26	89.89
	KD+BU	71.40	84.49	82.07	1.1	89.30
	NeuralFP	**99.96**	**99.91**	**99.91**	**99.95**	**98.87**

Table 5.8: Detection AUC-ROC of blackbox-defenders (defender does not know attack strategy) against *partial-whitebox-attackers* (know model $f(x;\theta)$, but not fingerprints; see Section 5.5), on MNIST, CIFAR-10 on test-set ("real") and corresponding adversarial ("fake") samples (1328 *pre-test* samples each). *NeuralFP* outperforms the baselines (LID, KD, BU) on MNIST and CIFAR-10 across attacks.

Layer	Parameters
Convolution + ReLU + BatchNorm	$5 \times 5 \times 32$
MaxPool	2×2
Convolution + ReLU + BatchNorm	$5 \times 5 \times 64$
MaxPool	2×2
Fully Connected + ReLU + BatchNorm	200
Fully Connected + ReLU + BatchNorm	200
Softmax	10

Table 5.9: MNIST Model Used

Layer	Parameters
Convolution + ReLU + BatchNorm	$3 \times 3 \times 32$
Convolution + ReLU + BatchNorm	$3 \times 3 \times 64$
MaxPool	2×2
Convolution + ReLU + BatchNorm	$3 \times 3 \times 128$
Convolution + ReLU + BatchNorm	$3 \times 3 \times 128$
MaxPool	2×2
Fully Connected + ReLU + BatchNorm	256
Fully Connected + ReLU + BatchNorm	256
Softmax	10

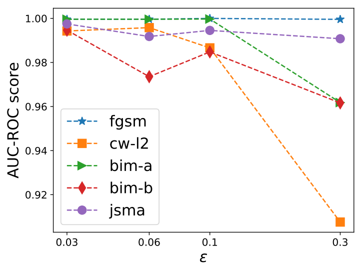
Table 5.10: CIFAR Model Used

Layer	Parameters
Convolution + ReLU + BatchNorm	$11 \times 11 \times 64$
MaxPool	3×3
Convolution + ReLU + BatchNorm	$5 \times 5 \times 192$
MaxPool	3×3
Convolution + ReLU + BatchNorm	$3 \times 3 \times 384$
MaxPool	3×3
Convolution + ReLU + BatchNorm	$3 \times 3 \times 256$
MaxPool	3×3
Convolution + ReLU + BatchNorm	$3 \times 3 \times 156$
MaxPool	3×3
Fully Connected + ReLU + BatchNorm	3072
Dropout	-
Fully Connected + ReLU + BatchNorm	1024
Dropout	-
Softmax	20

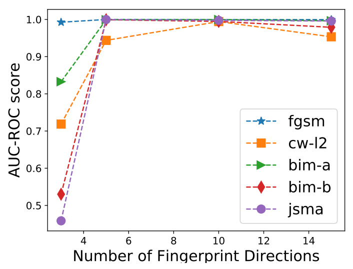
Table 5.11: MiniImagenet-20 Model Used

Experimental Results

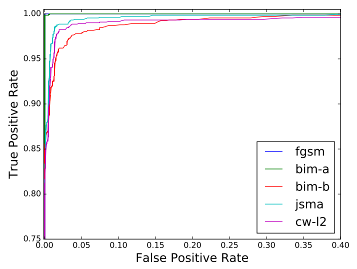
MNIST. We trained a 5-layer ConvNet to $99.2 \pm 0.1\%$ test-accuracy, see Table 5.9. The set of $\Delta x^i \in \mathbb{R}^{28\times 28}$ are chosen at random, with each pixel perturbation chosen uniformly in $[-\varepsilon, \varepsilon]$. For each Δx^i, if x is of label-class k, $\Delta y^{i,k} \in \mathbb{R}^{10}$ is chosen to be such that $\Delta y^{i,k}_{l\neq k} = -0.235$ and $\Delta y^{i,k}_{l=k} = 0.73$, with $\|\Delta y\|_2 = 1$. The AUC-ROCs for the best N and ε using grid-search are reported in Table 5.8. We see that *NeuralFP* achieves near-perfect detection with AUC-ROC of $99 - 100\%$ across all attacks.



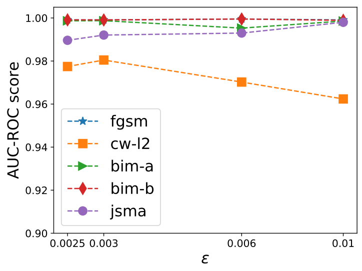
(a) Varying ε ($N = 10$)



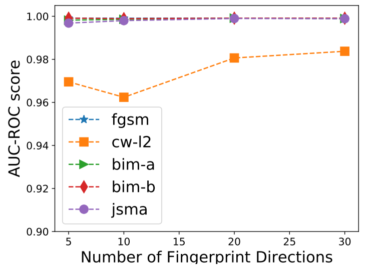
(b) Varying N ($\varepsilon = 0.03$)



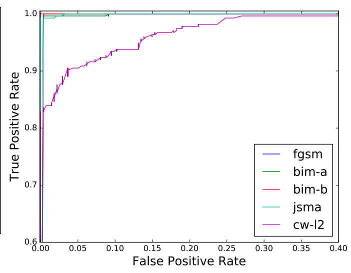
(c) ROC Curve



(d) Varying fingerprint magnitude ε ($N = 10$)



(e) Varying no. fingerprints (N) ($\varepsilon = 0.01$)



(f) ROC Curve, Varying Threshold (τ)

Figure 5.14: AUC-ROC for different hyperparameters (left, middle) and ROC curves (right) on MNIST (top) and CIFAR-10 (bottom). *NeuralFP* is robust across attacks & hyperparameters with an AUC-ROC between 95 – 100%. Increasing N improves performance, indicating more fingerprints are harder to fool. Increasing the magnitude ε decreases AUC on CW-L_2 only, suggesting that as adversarial perturbations become of smaller magnitude, *NeuralFP* requires smaller detection ranges.

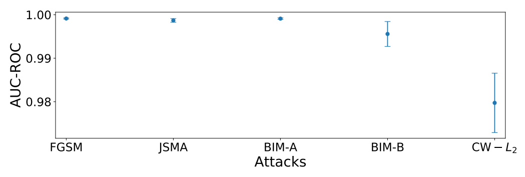


Figure 5.15: AUC-ROC mean μ and standard-deviation σ for 32 randomly sampled fingerprints (including randomizing N) for CIFAR-10. The AUC-ROC across all attacks varies little ($\sigma < 1\%$), with σ highest for CW-L_2.

CIFAR-10. For CIFAR-10, we trained a 7-layer ConvNet (see Table 5.10, similar to (N. Carlini and D. Wagner, 2016)) to 85 ± 1% accuracy. The Δx^i and $\Delta y^{i,j}$ are chosen similarly as for MNIST. Across attacks *NeuralFP* outperforms LID on average by **11.77%** and KD+BU, KD, BU even more substantially (Table 5.8). Even compared to LID-whitebox, *NeuralFP* outperforms LID-whitebox on average by **8.0%** (see Table 5.13).

Data	FGSM	BIM-b
MiniImagenet-20	99.96	99.68

Table 5.12: Detection AUC-ROC of *NeuralFP* vs *partial-whitebox-attacks* on MiniImagenet-20, $N = 20, \varepsilon = 0.05$.

MiniImagenet-20. We also evaluate on *MiniImagenet*-20 with 20 classes[3] randomly chosen from the 100 MiniImagenet classes (Vinyals et al., 2016). For this, we trained an AlexNet network (see Table 5.11) on 10,600 images (not downsampled) with 91.1% top-1 accuracy. We generated test-set adversarial examples using BIM-b with 50 steps (*NIPS 2017: Non-targeted Adversarial Attack* n.d.) and FGSM. Here, *NeuralFP* achieves AUC-ROCs of > 99.5% (Table 5.12)[4].

Remarks. Surprisingly, FGSM is more successful against LID (Tables 5.8, 5.13), compared to the stronger, iterative attacks. Athalye, Nicholas Carlini, and David Wagner, 2018 suggests that such behavior is indicative of an obfuscated gradient and potentially explains why it is harder for LID to defend against FGSM. Athalye, Nicholas Carlini, and David Wagner, 2018 goes on to reduce the accuracy for the LID defense to 0% using a high-confidence attack, oblivious to the defense. In contrast to LID and other defenses that were brokenBuckman et al., 2018; Song et al., 2018, *NeuralFP* does marginally worse against iterative attacks compared to the non-iterative attacks, and does well against blackbox-attacks (5.6), indicating that the defense's functioning is not based on obfuscated gradients.

Visualization. Figure 5.13 shows that fingerprint-loss differs significantly for most test and adversarial samples (across the 5 attacks in Table 5.6), enabling *NeuralFP* to achieve close to 100% AUC-ROC.

Sensitivity and Efficiency Analysis. Next, we study the effect of changing N (number of fingerprint directions) and ε on the AUC-ROC for MNIST and CIFAR-10, see Figure 5.14 shows that *NeuralFP* performs well across a wide range of hyperparameters and is robust to variation in the hyperparameters. With increasing ε,

[3]We use images from the following 20 ImageNet classes for our experiments: `n01558993, n02795169, n03062245, n03272010, n03980874, n04515003 n02110063, n02950826, n03146219, n03400231, n04146614, n04612504, n02443484, n02981792, n03207743, n03476684, n04443257, n07697537.`

[4] We could not obtain results for JSMA and CW-L_2, which are computationally very expensive for tasks of this size. Baselines numbers are not reported because of time constraints & unavailable baseline implementations.

the AUC-ROC for CW-L_2 decreases. A possible explanation is that CW-L_2 produces smaller adversarial perturbations than other attacks, and for larger ε, fingerprints are less sensitive to those small adversarial perturbations. However, the degradation in performance is not substantial ($\sim 4-8\%$) as we increase ε over an order of magnitude. With increasing N, the AUC-ROC generally increases across attacks. We conjecture that a larger set of fingerprints can detect perturbations in more directions and hence result in better detection.

Figure 5.15 shows that *NeuralFP* achieves mean AUC-ROC of $98\% - 100\%$ against all attacks, with standard deviation $< 1\%$. This suggests that *NeuralFP* is *not very sensitive to the chosen fingerprints*. Furthermore, the test accuracy with *NeuralFP* is $85 \pm 1\%$ on CIFAR-10 and $99.2 \pm 0.1\%$ on MNIST. This matches the accuracy for the same models when trained without fingerprints.

Robustness to Adaptive Whitebox-Attackers. We consider an adaptive attack that has full knowledge of the defense, similar to (Nicholas Carlini and D. A. Wagner, 2017a). The attacker (Adaptive-CW-L_2) tries to find an adversarial sample x' for input x that also minimizes the fingerprint-loss, attacking model trained with *NeuralFP*. In effect, the CW-L_2 objective is modified as:

$$\min_{x'} \|x - x'\|_2 + \gamma \left(L_{\mathrm{CW}}(x') + L_{\mathrm{fp}}(x', y^*, \xi; \theta) \right) \tag{5.35}$$

Here, y^* is the label-vector, $\gamma \in [10^{-3}, 10^6]$ is a scalar found through a bisection search, L_{fp} is the fingerprint-loss we trained on and L_{CW} is an objective encouraging misclassification. Under this threat model, *NeuralFP* achieves an AUC-ROC of **98.79%** for the CIFAR-10 task against Adaptive-CW-L_2, with $N = 30$ and $\varepsilon = 0.006$ for a set of unseen test-samples (1024 *pre-test*) and the corresponding adversarial examples. In contrast to other defenses that are vulnerable to Adaptive-CW-L_2 (Nicholas Carlini and D. A. Wagner, 2017a), we find that *NeuralFP* is robust even under this adaptive whitebox-attack threat model.

***NeuralFP* vs Whitebox-LID.** We also compare LID against *NeuralFP* in the setting where LID is aware of the attack mechanism and trains against the specific attack mechanism (*whitebox-LID*, while *NeuralFP* still does not use any attacker information.

In this *unfair* setting, we see whitebox-LID performs comparably on CW-L2, and *NeuralFP* outperforms whitebox-LID against all other attacks, see Table 5.13.

Data	Method	FGM	JSMA	BIM-a	BIM-b	CW-L_2
MNIST	Whitebox-LID	99.68	98.67	99.61	99.90	99.55
	NeuralFP	**100.0**	**99.97**	**99.94**	**99.98**	**99.74**
CIFAR-10	Whitebox-LID	82.38	95.87	82.30	99.78	**98.94**
	NeuralFP	**99.96**	**99.91**	**99.91**	**99.95**	98.87

Table 5.13: Detection AUC-ROC for *NeuralFP*, whitebox-LID against *whitebox-attackers* (know model $f(x;\theta)$, but not fingerprints; see Section 5.5), on MNIST, CIFAR-10 tasks on test-set ("real") and corresponding adversarial ("fake") samples (1328 *pre-test* samples each). *NeuralFP* outperforms the baselines (LID, KD, BU) on MNIST and CIFAR-10 across all attacks, except CW-L2 where it performs comparably. A possibly explanation for LID's improved performance against stronger, iterative attacks is gradient masking (Athalye, Nicholas Carlini, and David Wagner, 2018).

Data	Method	FGM
MNIST	*NeuralFP*	**99.96**
CIFAR-10	*NeuralFP*	**99.92**

Table 5.14: Detection AUC-ROC for *NeuralFP* against *blackbox-attackers* (know dataset but not model or fingerprints), on MNIST, CIFAR-10 tasks on test-set ("real") and corresponding blackbox adversarial ("fake") samples (1328 *pre-test* samples each). *NeuralFP* achieves near perfect AUC-ROC scores in this setting against FGM. Iterative attacks are known to not transfer well in the blackbox setting. For CIFAR-10, the hyperparameters are $N = 30, \varepsilon = 0.003$ and for MNIST, the hyperparameters are $N = 10, \varepsilon = 0.03$. We did not tune the parameters because this setting in itself achieved near perfect detection rates.

Athalye, Nicholas Carlini, and David Wagner, 2018 provides a possible explanation for this behavior for whitebox-LID, where defenses relying on obfuscated gradients perform better against stronger attacks (CW-L2) compared to one-step weaker attacks.

Robustness to Blackbox Attackers. In Athalye, Nicholas Carlini, and David Wagner, 2018, the authors indicate that testing against blackbox attacks is useful to gauge if the defense is relying on gradient masking. To explore this aspect of our defense, we construct adversarial examples for models trained without fingerprinting and then test the ability of fingerprinted models to distinguish these from unseen test-data. We find that the performance does not degrade from the whitebox-attack setting, in contrast to other defenses evaluated in Athalye, Nicholas Carlini, and David Wagner, 2018, where the performance degrades against blackbox-attacks.

5.7 Related work

Adversarial examples. Recently, several machine learning algorithms were found to have extreme instability against *contrived* input perturbations (Christian Szegedy, Zaremba, et al., 2013a) called adversarial examples. An open question remained as to whether such small perturbations that change the class label could occur without intentional human intervention. In this work, we document that they do in fact occur. Previous work has shown that training a classifier to resist adversarial perturbation can improve its performance on both the original data and on perturbed data (I. J. Goodfellow, J. Shlens, and C. Szegedy, 2014; Miyato et al., 2015). We extend this approach by training our feature embeddings to resist the naturally occurring perturbations that are far more common in practice.

Furthermore, our work differs drastically from Nguyen, Yosinski, and Clune, 2015, which is about how a model responds to intentionally contrived inputs that don't resemble the original data at all. In contrast, in this paper we consider the stability to practically widely occurring perturbations.

Defense via Data augmentation. A natural strategy to improve label stability is to augment the training data with *hard positives*, which are examples that the prediction model does not classify correctly with high confidence, but that are visually similar to easy positives. Finding such hard positives in video data for data augmentation has been used in Misra, Shrivastava, and Hebert, 2015; Kuznetsova et al., 2015; Prest et al., 2012 and has been found to improve predictive performance and consistency. As such, data augmentation with hard positives can confer output stability on the classes of perturbations that the hard positives represent. However, *Stability Training* differs from data augmentation in two ways. Firstly, we take a general approach by proposing a method that intends to make model performance more robust to various types of natural perturbations. Secondly, our proposed method does not use the extra generated samples as training examples for the original prediction task, but only for the stability objective.

Related Work. Several forms of defense to adversarial examples have been proposed, including adversarial training, detection and reconstructing images using adversarial networks (Meng and Chen, 2017). However, Nicholas Carlini and D. A. Wagner, 2017a; Nicholas Carlini and D. A. Wagner, 2017b showed many defenses are still vulnerable. Madry et al., 2017 employs robust-optimization techniques to minimize the maximal loss the adversary can achieve through first-order attacks.

Raghunathan, Steinhardt, and Liang, 2018; Kolter and Wong, 2017 train on convex relaxations of the network to maximize robustness. Although these works are complementary to *NeuralFP*, they do not scale very well to large neural networks or high-resolution inputs. Several other recent defenses attempt to make robust predictions by relying on randomization (Xie et al., 2018), introducing non-linearity that is not differentiable (Buckman et al., 2018) and by relying on Generative Adversarial Networks (Song et al., 2018; Pouya Samangouei, 2018) for denoising images. Instead, we focus on *detecting* adversarial attacks.

Amongst defenses that study the detection of adversarial examples, Ma et al., 2018 detect adversarial samples using an auxiliary classifier trained to use an expansion-based measure, *local intrinsic dimensionality (LID)*. A similar approach to detection is based on Kernel Density (KD) and Bayesian-Uncertainty (BU) using artifacts from pre-trained networks Feinman et al., 2017. In contrast with these methods, *NeuralFP* encodes information into the network response during training, and does not depend on auxiliary detectors.

5.8 Discussion

Our experiments suggest that *NeuralFP* is an effective method for safe-guarding against the strongest known state-of-the-art attacks, and the high AUC-ROC indicates that the fingerprints generalizes well to the test-set, but not to adversarials. Open questions are if there are attacks that can fool *NeuralFP* or if it is provably robust to certain attacks. Learning the fingerprints during training, and studying *NeuralFP* within a robust optimization framework are other interesting directions.

References

Athalye, Anish, Nicholas Carlini, and David Wagner (2018). “Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*. URL: https://arxiv.org/abs/1802.00420.

Buckman, Jacob et al. (2018). “Thermometer Encoding: One Hot Way To Resist Adversarial Examples”. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S18Su--CW.

Carlini, N. and D. Wagner (2016). “Towards Evaluating the Robustness of Neural Networks”. In: *ArXiv e-prints*. arXiv: 1608.04644 [cs.CR].

Carlini, Nicholas and David A. Wagner (2017a). “Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods”. In: *CoRR* abs/1705.07263. arXiv: 1705.07263. URL: http://arxiv.org/abs/1705.07263.

Carlini, Nicholas and David A. Wagner (2017b). “MagNet and "Efficient Defenses Against Adversarial Attacks" are Not Robust to Adversarial Examples”. In: *CoRR* abs/1711.08478. arXiv: 1711.08478. URL: http://arxiv.org/abs/1711.08478.

Feinman, R. et al. (2017). “Detecting Adversarial Samples from Artifacts”. In: *ArXiv e-prints*. arXiv: 1703.00410 [stat.ML].

Goodfellow, I. J., J. Shlens, and C. Szegedy (2014). “Explaining and Harnessing Adversarial Examples”. In: *arXiv:1412.6572 [cs, stat]*.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (2014). “Explaining and Harnessing Adversarial Examples”. In: *CoRR* abs/1412.6572.

Ioffe, Sergey and Christian Szegedy (2015). “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: pp. 448–456.

Kolter, J. Zico and Eric Wong (2017). “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *CoRR* abs/1711.00851. arXiv: 1711.00851. URL: http://arxiv.org/abs/1711.00851.

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

Kurakin, Alexey, Ian J. Goodfellow, and Samy Bengio (2016). “Adversarial examples in the physical world”. In: *CoRR* abs/1607.02533. arXiv: 1607.02533. URL: http://arxiv.org/abs/1607.02533.

Kuznetsova, A. et al. (2015). “Expanding object detector’s Horizon: Incremental learning framework for object detection in videos”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ma, Xingjun et al. (2018). “Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality”. In: *International Conference on Learning Representations*. accepted as oral presentation. URL: https://openreview.net/forum?id=B1gJ1L2aW.

Madry, A. et al. (2017). “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *ArXiv e-prints*. arXiv: 1706.06083 [stat.ML].

Meng, Dongyu and Hao Chen (2017). “MagNet: a Two-Pronged Defense against Adversarial Examples”. In: *CoRR* abs/1705.09064. arXiv: 1705.09064. URL: http://arxiv.org/abs/1705.09064.

Misra, Ishan, Abhinav Shrivastava, and Martial Hebert (2015). “Watch and Learn: Semi-Supervised Learning for Object Detectors From Video”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Miyato, Takeru et al. (2015). “Distributional Smoothing with Virtual Adversarial Training”. In: *arXiv:1507.00677 [cs, stat]*. URL: http://arxiv.org/abs/1507.00677.

Nguyen, Anh, Jason Yosinski, and Jeff Clune (2015). “Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

NIPS 2017: Non-targeted Adversarial Attack (n.d.). Accessed: 2017-02-07. URL: %5Curl%7Bhttps://www.kaggle.com/c/nips-2017-non-targeted-adversarial-attack/%7D.

Papernot, Nicolas et al. (2015). “The Limitations of Deep Learning in Adversarial Settings”. In: *CoRR* abs/1511.07528. arXiv: 1511.07528. URL: http://arxiv.org/abs/1511.07528.

Pouya Samangouei Maya Kabkab, Rama Chellappa (2018). “Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models”. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=BkJ3ibb0-.

Prest, A. et al. (2012). “Learning object class detectors from weakly annotated video”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Raghunathan, A., J. Steinhardt, and P. Liang (2018). “Certified Defenses against Adversarial Examples”. In: *ArXiv e-prints*. arXiv: 1801.09344 [cs.LG].

Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Song, Yang et al. (2018). “PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples”. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rJUYGxbCW.

Srivastava, Nitish et al. (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15, pp. 1929–1958.

Szegedy, Christian, Wei Liu, et al. (2015). “Going Deeper with Convolutions”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. URL: http://arxiv.org/abs/1409.4842.

Szegedy, Christian, Wojciech Zaremba, et al. (2013a). “Intriguing Properties of Neural Networks”. In: *arXiv:1312.6199 [cs]*. URL: http://arxiv.org/abs/1312.6199.

– (2013b). “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199*.

Vinyals, Oriol et al. (2016). “Matching Networks for One Shot Learning”. In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., pp. 3630–3638. URL: http://papers.nips.cc/paper/6385-matching-networks-for-one-shot-learning.pdf.

Wang, Jiang et al. (2014). “Learning Fine-grained Image Similarity with Deep Ranking”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Xie, Cihang et al. (2018). “Mitigating Adversarial Effects Through Randomization”. In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Sk9yuql0Z.

Xu, Weilin, David Evans, and Yanjun Qi (2018). “Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks”. In: *Network and Distributed Systems Security Symposium (NDSS) 2018*. Vol. abs/1704.01155.

Chapter 6

CONCLUSION: TOWARDS ROBUST AND GENERAL INTELLIGENCE

In this thesis we demonstrated new methods for exploiting structure for more expressive and efficient model learning. Spatiotemporal hierarchical reasoning and coordination are examples of complex low-dimensional structure in the state-action space that can lead to highly expressive and efficient learning. Furthermore, we demonstrated methods to improve the robustness of neural networks and how to detect adversarial examples.

However, this work inspires a significant number of follow-up questions; we outline a number of exciting future research directions and goals below.

Safety, Communication and Multi-agent Dynamics

A fruitful research direction is to generalize structured machine learning methods to richer forms of structure, such as safety and multi-agent communication that could be *learned* or *implemented as structural constraints*. Here, structure not only could improve the sample efficiency of learning, but also prevent unwanted behavior or inject domain knowledge into models.

Consider the future application of a multi-armed surgery robot that can operate autonomously on humans. Here, each arm can be thought of as an agent. It is clear that training such a system to operate safely on humans requires the system to have an understanding of proper coordination between arms and understanding of which coordinated actions are safe (e.g., do not harm the human patients).

However, there are significant challenges towards such applications.

First, it is unclear how to learn or impose safety constraints in the policy space or shape rewards, such that the effectiveness and consistency of the learned optimal policy is not adversely affected.

Second, multi-agent learning is most natural in the decentralized learning setting, where agents cannot access the full state of the environment and other agents. For instance, communicating the state of each robot arm to other parts of the system could be computationally costly and introduce too much lag and overhead, especially when power efficiency and robustness constraints are considered. However, decentralized multi-agent learning with partial information in cooperative and adversarial settings is an unsolved challenge.

Here, it would be interesting to generalize our structured learning approach to include more expressive spatiotemporal priors that can efficiently model safety and richer forms of structural constraints.

Unsupervised Structure Learning

In Chapter 2 we demonstrated that a meaningful hierarchical structure (e.g., long-term goals) can be learned from weak labels, extracted from the data using a heuristic. However, the drawback of this approach is that it is not always a-priori clear what the right hierarchical structure is to predict (e.g., for basketball other heuristic definitions than stationary points could be effective). In lieu of structure labels, our results suggest that simple "flat" models do not suffice, and that naive model class generalizations (e.g., hierarchical models) fail to capture structural concepts such as long-term goals.

Therefore, it is an open challenge to learn complex low-level structures, such as spatiotemporal hierarchies, in an unsupervised fashion, i.e., without labels. Some approaches could be to study more flexible, temporally extended priors on the latent distributions, that preserve tractability of learning and inference.

Model-Based Learning

Our overall goal of improving the sample efficiency and generalization of learning points to a general learning challenge towards broadly applicable smart systems.

Consider the following problem: *how can a bipedal robot autonomously learn to run on solid surfaces?*

There are several exciting research directions that have great potential towards efficient and robust learning for this problem.

First, the recent success in reinforcement learning methods largely have come from *model-free methods*, where the environment dynamics are implicitly modeled through rollout samples. However, this is highly inefficient, especially when the state and action space are continuous and / or high-dimensional. For example, the dynamics for a bipedal robot running on solid surfaces include hard contact dynamics between feet and surface, and nonlinear reaction forces due to friction and tension throughout the robot body.

A promising long-term approach is *model-based learning*, that also leverages a (parametric) model of the environment dynamics (also referred to as just the “model”).[1] Such methods could employ *planning methods*, that use forms of forward search to accurately estimate value function estimates and infer optimal actions. However, when only an imperfect model is known, e.g., when learning has not converged yet, combining imperfect models with suboptimal policies leads to highly unstable learning. An open challenge therefore is to formulate stable model-based learning approaches. Here, a fruitful approach is to use more powerful model classes, such as the `TT-RNN` introduced in Chapter 3, as environment models, and/or to use blackbox environment models indirectly by using fictitious rollouts as features for the policy and value functions.

Transfer Learning, Meta-Learning

A follow-up problem is: *given a robot that can stably run on solid surfaces, how can it quickly learn to run on sand?*

This is an instance of *transfer learning*, where we aim to develop learning algorithms that can quickly generalize models to new problem settings (“tasks”). Exploiting structure between tasks is key to effective transfer learning. For instance, the dynamics of solid and flexible surfaces are related by deforming the material “stress tensor”, and modeling such structure in the dynamics could aid policies to efficiently adapt to smooth transformations in the dynamics.

Detecting Strong Adversarial Examples

In Chapter 5 we showed two methods to stabilize neural networks and detect adversarial examples. In these works, we focused on the image domain. However, an open question is how effective these methods are against adversarial attacks in different data domains, such as sequential data, such as natural language, audio and video.

More generally, a central issue in the security domain, is whether one can *guarantee* certain levels of safety or robustness of the learned models. For instance, can we certify that a neural network is provably robust against adversarial examples with a given magnitude?

Until now, a strong theoretical understanding of the effectiveness of these methods is still lacking for complex models such as neural networks.

[1]Perfect knowledge of the dynamics reduces reinforcement learning to dynamic programming for finite state-action spaces.

For instance, for binary classification with linear models, we can characterize the vulnerable regigon exactly. However, for nonlinear models, we do not have an explicit description of the input regions that are vulnerable to an attacker with knowledge of our chosen fingerprints. Theoretical understanding of this issue requires *understanding and developing tools to control the geometry of neural networks*.

BIBLIOGRAPHY

Amodei, Dario et al. (2016). “Deep speech 2: End-to-end speech recognition in english and mandarin”. In: *International Conference on Machine Learning*, pp. 173–182.

Braziunas, Darius (2003). “POMDP solution methods”. In: *University of Toronto*.

Cho, KyungHyun et al. (2014). “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *CoRR* abs/1409.1259. arXiv: 1409.1259.

Goodfellow, I. J., J. Shlens, and C. Szegedy (2014). “Explaining and Harnessing Adversarial Examples”. In: *arXiv:1412.6572 [cs, stat]*.

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

Goodfellow, Ian, Jean Pouget-Abadie, et al. (2014). “Generative adversarial nets”. In: *Advances in neural information processing systems*, pp. 2672–2680.

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

Jia, Robin and Percy Liang (2017). “Adversarial examples for evaluating reading comprehension systems”. In: *arXiv preprint arXiv:1707.07328*.

Kingma, Diederik P. and Max Welling (2014). “Auto-Encoding Variational Bayes”. In: *ICLR*.

Oliehoek, Frans A (2012). “Decentralized pomdps”. In: *Reinforcement Learning*. Springer, pp. 471–503.

Oord, Aaron van den et al. (2016). “Wavenet: A generative model for raw audio”. In: *arXiv preprint arXiv:1609.03499*.

Ross, Stéphane, Geoffrey Gordon, and Drew Bagnell (2011). “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635.

Ross, Stéphane, Joelle Pineau, et al. (2008). “Online planning algorithms for POMDPs”. In: *Journal of Artificial Intelligence Research* 32, pp. 663–704.

Russakovsky, Olga et al. (2015). “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Shani, Guy, Joelle Pineau, and Robert Kaplow (2013). “A survey of point-based POMDP solvers”. In: *Autonomous Agents and Multi-Agent Systems* 27.1, pp. 1–51.

Sutton, Richard S and Andrew G Barto (1998). *Introduction to reinforcement learning*. Vol. 135. MIT press Cambridge.

Szegedy, Christian et al. (2013). "Intriguing Properties of Neural Networks". In: *arXiv:1312.6199 [cs]*. URL: http://arxiv.org/abs/1312.6199.

Wang, Yuxuan et al. (2017). "Tacotron: A fully end-to-end text-to-speech synthesis model". In: *arXiv preprint arXiv:1703.10135*.

Wu, Yonghui et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation". In: *arXiv preprint arXiv:1609.08144*.

Zheng, Stephan, Yisong Yue, and Patrick Lucey (2016). "Generating Long-term Trajectories Using Deep Hierarchical Networks". In: *NIPS*.