ABSTRACT

Title of dissertation: ADVERSARIAL ROBUSTNESS AND ROBUST META-LEARNING FOR NEURAL NETWORKS

Micah Goldblum
Doctor of Philosophy, 2020

Dissertation directed by: Professor Wojciech Czaja
Department of Mathematics

Despite the overwhelming success of neural networks for pattern recognition, these models behave categorically different from humans. Adversarial examples, small perturbations which are often undetectable to the human eye, easily fool neural networks, demonstrating that neural networks lack the robustness of human classifiers. This thesis comprises a sequence of three parts. First, we motivate the study of defense against adversarial examples with a case study on algorithmic trading in which robustness may be critical for security reasons. Second, we develop methods for hardening neural networks against an adversary, especially in the low-data regime, where meta-learning methods achieve state-of-the-art results. Finally, we discuss several properties of the neural network models we use. These properties are of interest beyond robustness to adversarial examples, and they extend to the broad setting of deep learning.

ADVERSARIAL ROBUSTNESS AND ROBUST META-LEARNING FOR NEURAL NETWORKS

by

Micah Goldblum

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Acknowledgments

First and foremost, I would like to thank my advisor and co-advisor, Professors Wojciech Czaja and Tom Goldstein. Wojciech introduced me to machine learning, taught me the ins and outs of research, and walked me through navigating the bureaucracy of a university, which is no easy task. Tom opened my eyes to so many possibilities within deep learning. I owe the direction I've taken in research as well as my success as a graduate student to these two people.

Second, I want to thank my collaborators, without whom many projects would have been impossible. In addition to Wojciech and Tom, these collaborators include Tucker Balch, Valeria Cherepanova, Ping-Yeh Chiang, Naftali Cohen, Zeyad Emam, Liam Fowl, Jonas Geiping, Furong Huang, Ronny Huang, Michael Moeller, Renkun Ni, Ankit Patel, Steven Reich, Avi Schwarzschild, Ali Shafahi, and Justin Terry.

Finally, I'm grateful for the support of my friends and family without whom the long journey to this degree would have been unpleasant. Students in my cohort, including Steven Reich, Kasun Fernando, Pratima Hebbar, and Jerry Emidih, made being a student and taking classes much more enjoyable. My girlfriend Yimei has been there for me and has taken care of me when I was stressed out meeting deadlines. Nobody has expressed more faith in me than she has. Last but not least, my parents' undying support helped me to get through stressful moments and helped to guide me.

Table of Contents

Chapter 1: Introduction

Deep learning models have achieved state-of-the-art performance in settings ranging from computer vision to natural language processing to time-series forecasting to social network graph processing to speech recognition to reinforcement learning to recommendation systems. It is well-known that many of the biggest companies in existence are heavily dependent on neural networks. Nonetheless, little light has been shed on why neural networks make the decisions they do and how trustworthy they are. As a result of the strong empirical performance of these models, practitioners are satisfied to use them as black-boxes. Should we be alarmed by the widespread deployment of models whose behavior we do not understand? Recent work has shown that neural networks are vulnerable to *adversarial examples*, deliberately chosen tiny perturbations to inputs which do not change the semantic meaning but dramatically change the network's output. We first answer the question, "Are these perturbations harmless or should we be worried?"

In Chapter 3, we motivate the study of adversarial examples and defenses against these vulnerabilities by studying adversarial attacks in the setting of finance. High-frequency trading systems are often completely automated and make decisions on small time-scales with no human oversight. We demonstrate that a malicious trader can cause small perturbations to the input of neural networks which forecast future prices by placing their own buy and sell orders on a stock exchange. Using our algorithm and without knowing anything about their victims' models, an adversarial trader can fool the victim into consistently predicting that stock prices will always go down, for example. We further use this attack as a tool for interpreting the behavior of algorithmic trading models, and we find that they are sensitive to a specific type of pattern which closely resembles illegal "spoofing".

After motivating the field of adversarial attacks and defenses, we study methods for producing lightweight neural network models which are robust to adversarial attacks in Chapter 4. Previous work uses *knowledge distillation* to produce networks which are simultaneously lightweight and accurate by teaching the small "student" networks to imitate larger state-of-the-art "teacher" models. Other work has suggested that knowledge distillation from non-robust teacher models produces adversarially robust student models, but this claim has since been debunked. We show that knowledge distillation from adversarially robust teacher models produces robust student models, even when distillation is performed on only clean data. Further, we develop a method for improving the robustness of student models. Our method, *adversarially robust distillation* (ARD), improves on the performance of previous state-of-the-art defenses. We also introduce an accelerated version of ARD which has the same computational cost as (non-robust) knowledge distillation.

The rise of deep learning has been facilitated by the increasing availability of training data. Popular methods leverage enormous datasets, such as ImageNet, which contain over a million labeled images. In many applications, practitioners may not have such large datasets and instead adapt pre-trained models to their problems. *Few-shot learning* methods seek to produce models which adapt well to new tasks. In Chapter 5, we thoroughly investigate adversarial robustness properties of few-shot learning methods. We develop an algorithm for producing models which adapt on very little training data to new tasks while exhibiting a high level of adversarial robustness. We find that architectural features and pre-processing defenses, which have been used to improve robustness when vast training datasets are available, have no benefit for few-shot learning.

This work closes with a discussion of interesting properties of neural networks in Chapter 6. We empirically test predictions from deep learning theory and find that they often do not hold in the realistic setting. Previous work has suggested that the loss landscape for neural networks contains no suboptimal local minima. We present theoretical results which prove otherwise, and we use our constructive proof to empirically find suboptimal local minima on real datasets. Other previous theory suggests that networks with low-norm parameters perform better. We develop a new regularizer which improves performance over that of low-norm regularizers by increasing the norm of neural network parameters. We believe that practitioners have been led astray by theoretical results which operate in unrealistic settings. We encourage theoretical work which attempts to generalize previous results to settings which are relevant to real-world practice.

Chapter 2: Preliminaries

2.1 What are neural networks?

A *neural network* is a function which performs an alternating composition of affine and nonlinear operations. A common nonlinearity used in neural networks is $g(\mathbf{x}) = [\max(x_i, 0)]$, called the rectified linear unit (ReLU) [61]. Then, a simple one hidden layer network is $f(\mathbf{x}) = h_2(g(h_1(\mathbf{x})))$, where $h_i(\mathbf{x}) = A_i\mathbf{x} + \mathbf{b}_i$ is an affine function. We call $g \circ h_i$ a *hidden layer*, and we call the nonlinearity, g, an *activation function*. Layers in which all entries of the input are allowed to be used in computation of the output are called *fully-connected*. The entries in A_i and b_i are parameters of the network. Networks with all fully-connected layers are known as *multi-layer perceptrons* (MLP) [131]. See Figure 2.1 for a diagram of a multi-layer perceptron with two hidden layers.

Modern networks for computer vision harness translation invariance in images by employing discrete convolutions instead of fully-connected layers [48]. The parameters of convolutional layers are the entries of kernels against which inputs are convolved. Convolutions are linear and can be written in the matrix form above. For example, 2D discrete convolutions can be written as multiplication by a Toeplitz matrix.

In order to make computation efficient, modern neural networks use *pooling* operations between some or all layers. Average pooling can be written as a convolution operation with each entry of the kernel equal to $\frac{1}{n}$, where n is the number of entries in the kernel [112]. Maximum pooling is similar to average pooling, but instead of averaging elements within a window, the maximum is computed [168]. These pooling operations compress the information input into subsequent layers, reducing computational cost.

The authors of [80] hypothesized that as neural network parameters change during training, the distribution of inputs to any given intermediate layer changes since the parameters of the previous layer change. The authors devised a scheme called *batch-normalization* in which each batch of data during training, parameters are updated which normalize the outputs of convolutional layers channel-wise. Others have found that batch-normalization and other normalization schemes make training neural networks less dependent on hyperparameters of the training routine [159].

Another architectural feature of modern neural networks is skip connections, in which information is sent not only from a layer to the next layer but to layers several steps deeper as well [67, 76]. For example, in some networks, layers take as input not only the output of the previous layer but also the output of other previous layers. Another scheme, *residual connections*, involves adding the output of the previous layer to that of other previous layers before inputting to the upcoming layer. Using the notation of our previous construction, $f(\mathbf{x}) = h_3(g(h_2(g(h_1(\mathbf{x})) + \mathbf{x})))$ contains a residual connection.

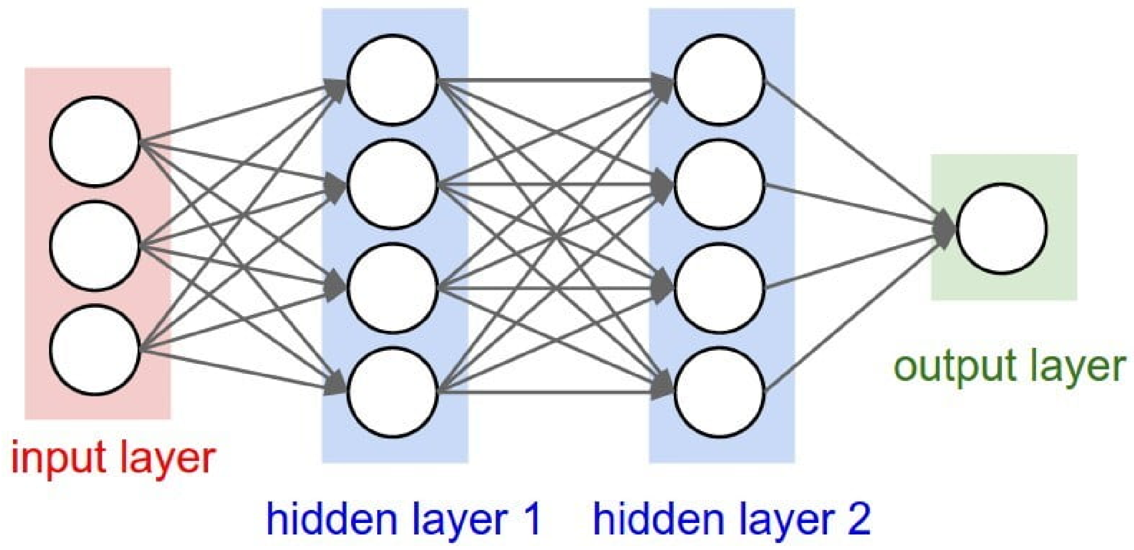


Figure 2.1: A two hidden layer multi-layer perceptron [39].

2.2 Training neural networks for classification

Training neural networks is framed as a minimization problem,

$$\min_{\theta} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \mathcal{L}(f_\theta(\mathbf{x}), y),$$

where f_θ is the network with parameters θ, $\mathcal{D}$ is a data distribution generating image-label pairs, and $\mathcal{L}$ is a loss function which is a proxy for error on the task. The loss function is carefully chosen, in combination with the activation function and the initialization for the network parameters, in order to yield a tractable optimization problem. In classification problems, a popular choice for loss functions is cross-entropy composed with the softmax function. The softmax function and cross-entropy functions are respectively written

$$s(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_{k=1}^{n} e^{\mathbf{x}_k}},\ c(\mathbf{x}, y) = -\log(\mathbf{x}_y),$$

where $\mathbf{x}$ has the number of entries corresponding to the number of classes in the classification problem, and the label y denotes the index corresponding to the correct class. Thus, classification networks output the same number of entries as there are classes in the problem, and the softmax function converts the network outputs to a probability distribution over the classes. The cross-entropy loss in turn penalizes the network for generating a probability far away from 1 for the correct class. Neural networks trained with softmax and cross-entropy, $\mathcal{L} = c \circ s$, generate estimates of posterior probabilities on the training data [17]. Another loss function used for training neural networks is KL-divergence, $\text{KL}(\mathbf{p}, \mathbf{q}) = \sum_i \mathbf{p}_i \log(\frac{\mathbf{p}_i}{\mathbf{q}_i})$, where $\mathbf{p}, \mathbf{q}$ are probability distributions. This loss function is used to encourage a network with softmax to output probabilities similar to those of a specified distribution.

We begin a training routine by randomly initializing the parameters, θ, of a network f [158]. We also consider a learning rate, α, which will govern the speed of training. Training is separated into epochs, periods in which the network sees all data in the data set exactly once. During each epoch, the data set is randomly split into batches of equal size. We loop through each batch and update the network's parameters according to a gradient descent step

$$\theta \leftarrow \theta - \alpha \sum_{i \in B} \nabla_\theta \mathcal{L}(f_\theta(\mathbf{x}_i), y_i),$$

where $\mathcal{L}$ denotes the loss function, and B is the set of indices corresponding to data in the current batch. This training algorithm is called stochastic gradient descent (SGD). Similar routines use modified update rules in order to accelerate training. For example, momentum [152] involves the following:

$$v \leftarrow \gamma v + \eta \sum_{i \in B} \nabla_\theta \mathcal{L}(f_\theta(\mathbf{x}_i), y_i),$$

$$\theta \leftarrow \theta - v.$$

Further additions to the training routine harness regularizers, penalty terms added to the loss function. A common regularizer is weight-decay, which penalizes the sum of squared parameters, $w(\theta) = \sum \theta_i^2$ [93]. During training, we measure both the training loss and training accuracy, the percentage of training samples classified correctly, in order to evaluate how well models fit the data. After training, we evaluate generalization by measuring accuracy on the test set. When models overfit the training data, we observe declining training loss while test accuracy remains constant or even increases.

2.3 Adversarial attacks

Adversarial examples are small perturbations to the input of a neural network which does not change the semantic meaning of the input but greatly changes the network's output [21]. A pathological example is a perturbation to a single pixel of an image of a car which causes the image to be classified as a person. Adversarial examples also exist in settings other than image classification. For example, an adversarial example for detection might involve a small perturbation that causes failure to detect a car or a person [164].

In the classification setting, an adversarial attack is a "small" perturbation to data which causes misclassified. We formalize the notion of an adversarial example below.

Definition 2.1 (Adversarial Example)**.** Denote the set of valid inputs by S and image similarity distance function by d. An input $\mathbf{x}' \in S$ is an adversarial example relative to model f, input label pair, $\mathbf{x}, y$, and attack budget ϵ if $d(\mathbf{x}, \mathbf{x}') < \epsilon$ and $\arg\max_i f(\mathbf{x})_i = y$, so that the perturbation is small and the original input is classified correctly by f and yet $\arg\max_i f(\mathbf{x}')_i \neq y$.

The most popular choice of distance function for constraining the attacker is the ℓ_∞ norm, in which case the attack is called an ℓ_∞ attack. A simple method for generating an ℓ_∞ adversarial attack, called the Fast Gradient Sign Method (FGSM), perturbs an input by $\delta = \epsilon\, \text{sign}[\nabla_{\mathbf{x}} \mathcal{L}(f_\theta(\mathbf{x}), y)]$ [57]. More powerful attacks use an iterative optimization procedure. An example of an iterative attack is the PGD attack in which $\delta_{n+1} = \pi_\epsilon[\delta_n + \alpha\, \text{sign}[\nabla_{\delta_n} \mathcal{L}(f_\theta(\mathbf{x}+\delta_n), y)]]$, where π_ϵ denotes projection onto the ℓ_∞-ball of radius ϵ centered at the origin [114].

A large body of work has developed around avoiding vulnerability to these attacks Some methods involve issuing certificates that a network does not admit adversarial examples within some radius of a data point [27]. Other methods involve creating networks whose loss function yields a difficult optimization problem for the attacker [7]. And yet other defenses sanitize adversarial data before feeding it into the network [59].

In the adversarial robustness literature, accuracy on non-adversarial data is called *natural accuracy* while accuracy on adversarial examples is called *robust accuracy*.

2.4 Datasets

In this work, we primarily consider natural image datasets. CIFAR-10 and CIFAR-100 are canonical natural image datasets from the Canadian Institute for Advanced Research (CIFAR) with ten and one hundred classes respectively [92]. These datasets each contain 60,000 labeled RGB images with spatial dimensions of 32×32. The ten classes for CIFAR-10 are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. These classes are non-overlapping. CIFAR-100 similarly contains non-overlapping classes of common objects and animals. Both CIFAR-10 and CIFAR-100 are split into training and test sets with 50000 and 10000 images respectively. Below is a panel with an image from each CIFAR-10 class.

In the few-shot setting, datasets are split into training and testing classes so that models can be trained on the training classes and tested by fine-tuning on the testing classes. One such dataset is CIFAR-FS which is constructed by choosing 64 classes of CIFAR-100 to be training classes, 16 for validation, and 20 for testing [12]. Mini-ImageNet is another few-shot dataset formed by choosing 600 images from each of 100 classes from the ImageNet 2012 classification dataset and downsampling

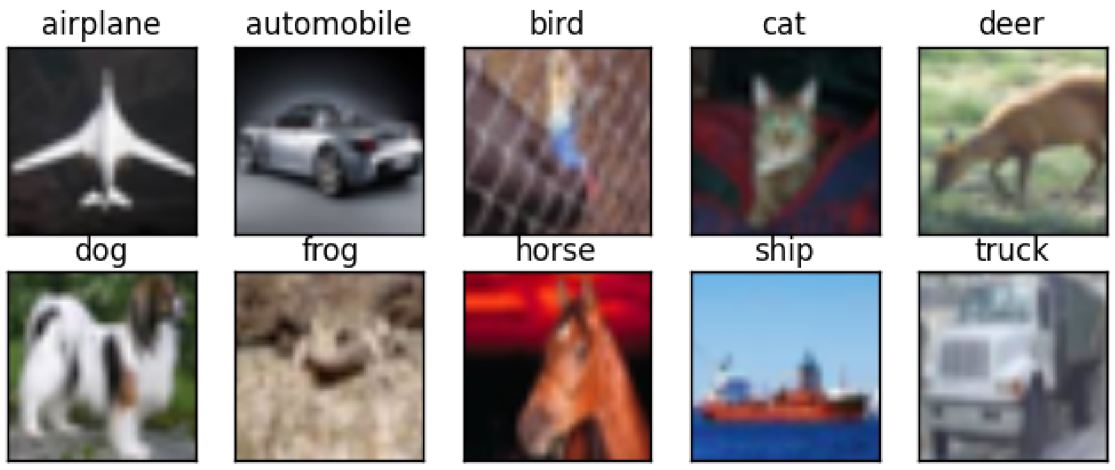


Figure 2.2: An example from each of the ten CIFAR-10 classes [92].

the images to be 84 × 84 × 3 [162]. The classes are then split using the same training/validation/testing proportions as CIFAR-FS.

In our experiments with stock data, we use assets from the Nasdaq exchange. We use order book snapshots provided by LOBSTER (Limit Order Book System - The Efficient Reconstructor) [77].

2.5 Mathematics in machine learning

Mathematical work has contributed to many facets of machine learning. Fundamental work on approximation theory attempts to better understand why neural networks work [32, 106]. Dimension reduction methods illuminate the shape and dimensionality of data and also to understand why some networks are more effective for few-shot classification [55, 124]. The scattering transform is a transform method which uses frames from harmonic analysis to model neural networks [18, 105, 127]. Similarly work has used shearlet transforms to introduce shearing and scale invariance into convolutional neural networks [50].

In scientific applications, mathematical contributions have helped to adapt deep learning methods to real applications. In optical communication, a CNN-based demultiplexing method has reduced hardware costs by rendering alignment unnecessary [40]. Denoising methods have enabled practitioners to remove artifacts in passive cavitation imaging [109].

The adversarial attack and robustness literature has seen advances from mathematics as well. Optimization work has proven that adversarial attacks can increase the loss of neural networks [41]. Other work has tested various basis function compression methods for adversarial robustness [148].

Chapter 3: Analyzing the Security of Machine Learning for Algorithmic Trading

Algorithmic trading systems are often completely automated, and deep learning is increasingly receiving attention in this domain. Nonetheless, little is known about the robustness properties of these models. We study valuation models for algorithmic trading from the perspective of adversarial machine learning. We introduce new attacks specific to this domain with size constraints that minimize attack costs. We further discuss how these attacks can be used as an analysis tool to study and evaluate the robustness properties of financial models. Finally, we investigate the feasibility of realistic adversarial attacks in which an adversarial trader fools automated trading systems into making inaccurate predictions. This work was conducted with Avi Schwarzschild, Naftali Cohen, Tucker Balch, Ankit Patel, and Tom Goldstein [56]. My contribution was conceiving, implementing, and testing the adversarial attacks and writing a substantial portion of the paper.

3.1 Introduction

Machine learning serves an increasingly large role in financial applications. Recent trends have seen finance professionals rely on automated machine learning systems for algorithmic trading [2], robo-advisers that allocate investments and rebalance portfolios [111, 118], fraud detection systems that identify illicit transactions [1, 13, 130], risk models that approve/deny loans [15], and high-frequency trading systems that make decisions on timescales that cannot be checked by humans [4, 5, 8, 16, 69, 90]. With the widespread use of such models, it is increasingly important to have tools for analyzing their security and reliability.

In the mainstream machine learning literature, it is widely known that many machine learning models are susceptible to *adversarial attacks* in which small but deliberately chosen perturbations to model inputs result in dramatic changes to model outputs [94]. It has already been demonstrated that adversarial attacks can be used to change someone's identity in a face recognition system [150], bypass copyright detection systems [132], and interfere with object detectors [46]. In this work, we investigate adversarial attacks on models for algorithmic and high-frequency trading.

Trading bots historically rely on simple ML models and increasingly leverage the cutting-edge performance of neural networks [5, 16]. Security and reliability issues are particularly relevant to high-frequency trading (HFT) systems, as actions are determined and executed in extremely short periods. These short time horizons make it impossible for human intervention to prevent deleterious behavior, and it is known that unstable behaviors and security vulnerabilities in such systems have contributed to major market events (e.g., the 2010 "flash crash" [89]).

We focus on adversarial attacks on the stock price prediction systems employed by trading bots. We look at adversarial attacks from two different perspectives.

Adversarial analysis of reliability and stability: After a price prediction model has been trained, it is important to understand the reliability of the model and any unseen instabilities or volatile behaviors it may have. A naive analysis method is backtesting; one feeds recent historical stock market data into a model and examines its outputs. However, models may have extreme behaviors and instabilities that arise during deployment that cannot be observed using historical data. This is especially true when complex and uninterpretable neural models are used [156]. Furthermore, market conditions can change rapidly, resulting in a domain shift that degrades model performance [34, 126].

We propose the use of adversarial attacks to reveal the most extreme behaviors a model may have. Our adversarial attacks generate synthetic market conditions under which models behave poorly, and determine whether instabilities exist in which small changes to input data result in extreme or undesirable behavior. This can be used as a tool to interrogate the reliability of a model before deployment or during postmortem analysis after a major event.

Adversarial attacks as a security vulnerability: We assess the potential for adversarial attacks to be used by malicious agents to manipulate trading bots. Several factors contribute to the potential vulnerability of stock price models. In mainstream adversarial literature, the attacker often cannot directly control model inputs; the adversary modifies a physical object (e.g., a stop sign) with the hopes that their perturbation remains adversarial after the object is imaged under unknown conditions. By contrast, our adversary is blessed with the ability to directly control model inputs. When order book data is used for price prediction, law-abiding traders and malicious actors alike receive identical market data from an exchange. The adversary can perturb the order book by placing (and canceling) their own orders. These adversarial orders quickly appear on the public exchange and are fed directly into victim models.

At the same time, there are challenges to attacks on order book data. An adversary's malicious orders must be bounded in their financial cost and detectability. Moreover, the attacker cannot know the future of the stock market, and so attackers must rely on *universal* attacks that remain adversarial under a wide range of stock market behaviors. An adversary's knowledge of the victim model is also limited, thus we assess the effectiveness of these universal attacks across model architectures as well.

Interestingly, when the attacks are deployed on historical market data, we observe that adversarial algorithms can *automatically* discover known manipulation strategies used by humans in "spoofing". These attacks are known to be effective and problematic, spawning their ban in the Dodd-Frank Act of 2010 [70].

3.2 Background

Before discussing new methods, we briefly review relevant existing work on algorithmic trading and adversarial attacks.

3.2.1 Machine Learning for High-Frequency Trading

High-frequency trading bots can profit from extremely short-term arbitrage opportunities in which a stock price is predicted from immediately available information, or by reducing the market impact of a large transaction block by breaking it into many small transactions over a trading day [4, 19, 69, 90].

High-frequency trading systems contain a complex pipeline that includes price prediction and execution logic. Price prediction models (often called "valuations") are a fundamental part of any trading system; they forecast the future value of equities and quantify uncertainty. Execution logic consists of proprietary buy/sell rules that account for firm-specific factors including transaction costs, price-time priority of orders, latency, and regulatory constraints [34].

Since the ability to anticipate trends is the core of algorithmic trading, we focus on price prediction and its vulnerability to attack. Price predictors use various streams of time-series data to predict the future value of assets. Traditional systems use linear models, such as autoregressive moving-average (ARMA) and support-vector machine (SVM), since linear models like these are fast for both training and inference [10, 87].

As hardware capabilities increase, training and inference are accelerated by advances in GPU, FPGA, and ASIC technology. Heavyweight models, like neural networks, have gathered interest as computations that were once intractable on a short timescale are becoming feasible [36]. Recent works use LSTM and temporal convolutional networks to predict future prices in a short time horizon [5, 16].

Common trading strategies, typical of technical analysis, rely on mean-reversion or momentum-following algorithms to identify an oversold or overbought asset. Mean-reversion methods assume that fluctuations in stock prices will eventually relax to their corresponding long-term average, whereas momentum-following methods aim at identifying and following persistent trends in asset prices. Simple trading algorithms identify an indicator of either strategy and act when prices cross a predefined threshold, thus signaling for a potentially profitable trade over the oversold or overbought asset [28, 119].] We use this type of thresholding for our trading systems. More details on our setup can be found in Section 3.3.2.

Historically, traders have engaged in *spoofing* in which an adversary places orders on either side of the best price to fake supply or demand, often with the intent of canceling the orders before execution [139]. Orders placed by a trader engaging in spoofing can be thought of as a naive hand-crafted adversarial perturbation. In this work, we use techniques from mathematical optimization to automate the construction of effective and efficient perturbations. Spoofing is now illegal under the Dodd-Frank Act of 2010. However, our perturbations, crafted with optimization techniques, may be less detectable than handcrafted versions and thus may present a problem for enforcement.

3.2.2 Adversarial Attacks

Neural networks are known to be vulnerable to *adversarial attacks*, small perturbations to inputs which dramatically change a network's output [94]. The most popular setting for studying adversarial attacks to date is image classification. Adversarial attacks in this domain perturb an image to fool a classifier while constraining the perturbation to be small in some norm.

While classical adversarial perturbations are designed for a single image, "universal" perturbations are crafted to fool the classifier when applied to nearly any image [116, 145]. Similar attacks exist for other vision tasks like detection and segmentation [165]. Other domains, such as natural language and graph structured data, have attracted the attention of adversarial attacks, but these attacks are impractical [3, 33]. In NLP, adversarial text may be nonsensical, and in graph structured data, attackers are weak.

3.2.3 The Order Book

The *order book* keeps track of buy orders and sell orders for a financial asset on an exchange. We use *limit* order book data from the Nasdaq Stock Exchange. Limit orders are orders in which an agent either offers to buy shares of a stock at or below a specified price or offers to sell shares of a stock at or above a specified price. Since agents can place, for example, buy orders at a low price, limit orders may not execute immediately and can remain in the order book for a long time. When the best buy order price (the highest price at which someone will buy) matches or exceeds the best sell order price (the lowest price at which someone is willing to sell) orders are filled and leave the book.

At any given time, the order book contains prices and their corresponding sizes. *Size* refers to the total number of shares being offered or demanded at a particular price. The size entries for a price level may contain orders of many agents, and these orders are typically filled in a first-in-first-out manner. More complex rules come into play when, e.g., orders require tie breaking, or limit orders have stipulations on their execution. A complete list of market practices and conduct standards can be found in the official Nasdaq equity rules [91].

3.3 Building valuation models

In this section, we describe simple valuation models that we will attack. Our models digest a time series of size-weighted average price levels over a 1 minute interval, and make a prediction about a stock price ten seconds in the future. The models used here were chosen for their simplicity – they are meant to form a testbed for adversarial attacks and not to compete with the proprietary state-of-the-art. Still, we verify that our models learn from patterns in the data and out-perform simple baselines.

3.3.1 Data

We use centisecond-resolution data from September and October 2019, and we choose to only use the first hour of trading each day since this period contains the largest volume of activity. Taking one month of data per asset, we use the first sixteen trading days for training and the last four for testing. We choose well-known stocks with high volatility relative to order book thickness: Ford (F), General Electric (GE), and Nvidia (NVDA). Order book data was furnished by LOBSTER, a financial data aggregator that makes historical order book data for Nasdaq assets available for academic use [77].

Consider that each row of the order book contains the ten best buy prices and their corresponding sizes, $\{(p_1^B, s_1^B), ..., (p_{10}^B, s_{10}^B)\}$, as well as the ten best sell prices and their corresponding sizes, $\{(p_1^S, s_1^S), ..., (p_{10}^S, s_{10}^S)\}$. These rows are each a snapshot of the book at a particular time. We process this data by creating the *size-weighted average* (SWA) of a row,

$$\text{SWA}([p_1^B, s_1^B, ..., p_{10}^B, s_{10}^B, p_1^S, s_1^S, ..., p_{10}^S, s_{10}^S])$$

$$= \frac{\sum_{i=1}^{10} p_i^B s_i^B + \sum_{i=1}^{10} p_i^S s_i^S}{\sum_{i=1}^{10} s_i^B + \sum_{i=1}^{10} s_i^S},$$

for each row. Movement in the size-weighted average may represent a shift in either or both price and size. The SWA is a univariate surrogate for the price of an asset.

After computing SWAs of the order book data, inputs to the models are 60 second time-series of this one-dimensional data. Since we use order book snapshots at intervals of 0.01 seconds, an input contains $60 \times 100 = 6,000$ SWA entries. See Figure 3.1 for a visual depiction of the SWA time-series.

We focus on three-class classification models for concreteness. The goal of the model is to classify an equity as likely to increase in price above a threshold, decrease below a threshold, or remain between the thresholds. Thresholds are chosen to be symmetric around the origin, and so that the default class (“no significant change”) contains one third of all events, making the classification problem approximately balanced.

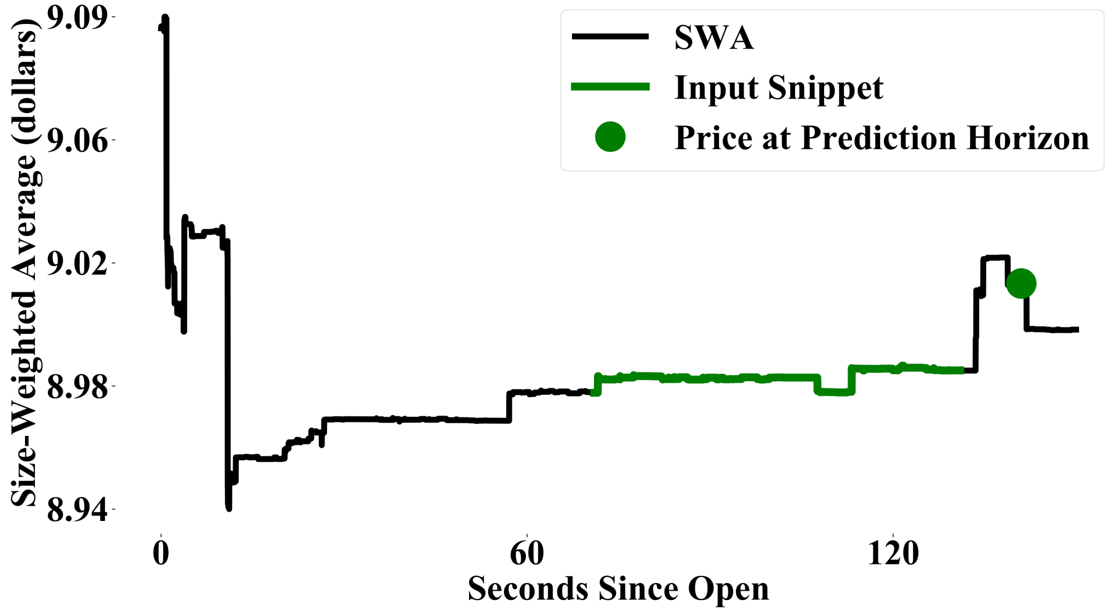


Figure 3.1: This sample of the SWA curve for GE data shows an example of an input snippet. The label for this input is determined by the change in price from the right end of the green snippet to the green dot.

3.3.2 Trading Models

On each asset, we train a linear classifier, a multi-layer perceptron (MLP), and an LSTM [74]. The MLP models have 4 hidden layers, each of width 8,000. The LSTM models have 3 layers and a hidden-layer size of 100.

We train each model with cross-entropy loss over the three classes. Since our data can be sampled by taking any 60 second snippet from the first hour of any of the trading days in our training set, we have a large number of data points. We randomly choose a batch of 60-second snippets from all of the training data and perform one iteration of SGD on this batch before resampling. These batches range in size from 2,000 to 3,000 data points depending on the model. Further details can be found in the Appendix of [56].

3.3.3 Comparison to baselines

The performance of our models is reported as classification accuracy on testing data. Since we have a large quantity of data, we randomly sample 10,000 snippets from the test set and form confidence intervals of radius one standard error from the accuracy estimations.

We verify that our models are indeed learning from the data by checking that they exceed simple baselines. See Table 3.1 for the numerical results. There are two natural baselines for this problem: a random guess, and a strategy that always predicts the label most commonly associated with that stock in the training data.[1] The latter baseline always performed worse than a random guess on the test data, so the most natural performance baseline accuracy is 33.33%. Tables 3.1 and 3.2 show accuracy measurements which should be read with this in mind. Note that price prediction is a difficult task, and profitable valuation models typically achieve performance only a few percentage points better than baselines [19].

3.4 Analyzing the Vulnerabilities of Trading Models

In this section, we develop a basic adversarial attack, and we use it to study the robustness properties of trading models as well as to compare the sensitivities of different networks. One might think that because these models are trained on noisy data, they are insensitive to small perturbations. We demonstrate that the

[1]Our classes are designed so that the middle class (little to no change in the SWA) accounts for one third of the training data. This leaves the 'up' and 'down' classes slightly off balance.

Ford (F)

Model	Test Accuracy(%)
Linear	34.66(±0.5)
MLP	36.56(±0.5)
LSTM	36.40(±0.5)

General Electric (GE)

Model	Test Accuracy (%)
Linear	36.41(±0.5)
MLP	36.87(±0.5)
LSTM	35.65(±0.5)

Nvidia (NVDA)

Model	Test Accuracy (%)
Linear	34.15(±0.5)
MLP	35.69(±0.5)
LSTM	37.11(±0.5)

Table 3.1: Accuracy of each model on test data with confidence intervals of radius one standard error. Note that all models are performing pattern recognition as described in Section 3.3.2. Furthermore, the neural network models outperform the linear ones.

networks are indeed vulnerable.

3.4.1 Rules for Perturbing the Order Book

A number of unique complexities arise when crafting adversarial attacks on order book data. We consider an adversary that places orders at the ten best buy and sell prices (orders at extreme price levels are typically discarded by analysts as they are unlikely to be executed). Equivalently, we can perturb the size entries upwards, but only to whole numbers, in raw order book data.

Note that using the size-weighted average as model inputs and only raising size entries prevents the adversary from greatly impacting the data seen by a model. Although these inputs lack the information content of raw order book data, without changing the price entries, an attacker can at most perturb the size-weighted average to the maximum or minimum price in the ten best buy/sell orders at a given time-stamp.

Consider the sequence of order book snapshots used for size-weighted average, $\{\mathbf{x}_i\}$, and the sequence of adversarial orders (on a per-row basis), $\{\mathbf{a}_i\}$. We cannot simply add these two quantities to compute the perturbed order book since the adversarial orders may remain on the book after being placed. On the other hand, we cannot blindly propagate the orders to the end of the snippet because transactions may occur in which adversarial orders are executed, removed from the book, and therefore be excluded from subsequent snapshots. Thus, denote the propagation function which accumulates adversarial orders and accounts for transactions, returning the sequence of perturbations to the snapshots, by p. Then, the perturbed sequence of order book snapshots is $\{\mathbf{x}_i + \delta_i\}$ where $\{\delta_i\} = p(\{\mathbf{a}_i\})$. Finally, the input to a model is $\mathrm{SWA}(\{\mathbf{x}_i + \delta_i\})$.

3.4.2 Quantifying Perturbation Size

In order to determine just how sensitive our models are, we must quantify the size of the perturbations being placed. We consider three possible metrics: cost, capital required, and relative size.

- We compute *cost* to the attacker as the change in the total value of his or her assets from the start of the snippet to the end. Assets include both cash and stocks. For this reason, cost accounts for changes in the price of stocks purchased when a buy order is filled (but we do not include transaction costs). We operate under the assumption that all attack orders are transacted when other orders are transacted at the same price level.

- *Capital required* is the hypothetical total cost of executing all orders placed by the attacker. This quantity is the total dollar value of orders in the perturbation.

- Finally, we quantify the *relative size* of our perturbation as the size (number of shares) of the adversary's propagated perturbation as a percentage of total size on the book during the snippet.

Since transactions occur infrequently at this time-scale, and prices go up roughly as much as they go down, the average cost across attacks is less than \$1.00

for each asset-model combination. For this reason, our crafting algorithms limit perturbation size in terms of capital required. We denote the capital required for an order sequence $\{\mathbf{a}_i\}$ as $C(\{\mathbf{a}_i\})$. We also measure cost and relative size.

3.4.3 Robustness to Random Noise

Before describing our attack, we establish a baseline by randomly inserting orders instead of using optimization methods, and we propagate the perturbations through the order book. We compare our attacks to this baseline to demonstrate that the optimization process is actually responsible for impairing classification. In fact, random attacks with a high budget barely impair classification at all. See Table 3.2 for accuracy against a random attacker. All random attacks have a budget over \$2,000,000, a sum far higher than any optimization-based attack we consider.

3.4.4 Crafting gradient-based attacks

We now introduce a simple adversarial attack for studying model sensitivity. This attack is untargeted, meaning that it degrades model performance by finding a small perturbation that causes misclassification to *any* wrong label.

Consider a model, f, and a sequence of snapshots, $\{\mathbf{x}_i\}$ with label y. Our attacker solves the maximization problem,

$$\max_{\{\mathbf{a}_i\}} \mathcal{L}(f(\{\mathrm{SWA}(\mathbf{x}_i + \delta_i)\}), y),$$

with capital constraint $C(\{\mathbf{a}_i\}) \leq c$, where $\{\delta_i\} = p(\{\mathbf{a}_i\})$. The adversary cannot perturb price (only size) in the snapshot, and all size entries must be integer-valued. To this end, we perform gradient ascent on the cross-entropy loss, $\mathcal{L}$, using a random learning rate $\alpha \sim \mathcal{U}(0, \alpha_0)$ and the update rule,

$$\{\mathbf{a}_i\}^k \leftarrow \{\mathbf{a}_i\}^{k-1} + \alpha \nabla \mathcal{L}(f(\{\text{SWA}(\mathbf{x}_i + \delta_i)\}), y).$$

The attacker runs through this iterative method with a small step size until either the model no longer predicts the correct label, y, or $C(R_r(\{\mathbf{a}_i\}))$ exceeds the capital constraint, c, where $R_r = \lceil \mathbf{x} + r \rceil$ is a ceiling operator that rounds size entry x to the greatest integer less than or equal to $x + r$ (i.e. $R_{0.3}(0.8) = 1$). When the attack terminates, we return the adversarial perturbation, $R_r(\{\mathbf{a}_i\})$. We choose c and r to be \$100,000 and 0.95, respectively, in our experiments. See the Appendix of [56] for a discussion on choice of attack hyperparameters.

Note. *In a standard PGD attack, attacks are randomly initialized in an ℓ_∞ ball, and perturbations are projected onto the ball each iteration. In contrast to ℓ_∞-bounded attacks, there is no natural distribution for random initialization within the set, $\{\{\mathbf{a}_i\} : C(\mathbf{a}_i) \leq c\}$. Moreover, there is no unique projection onto this set. Thus, we instead opt for a randomized learning rate in order to inject randomness into the optimization procedure [24].*

Table 3.2 depicts the effect of adversarial orders crafted in this manner on price prediction. Empirically, these trading models are not robust to low-budget perturbations. Moreover, gradient-based attacks are far more effective at fooling our models than randomly placed orders with far higher budget, indicating that our

Ticker	Model	$\mathcal{A}_{\text{test}}$(%)	$\mathcal{A}_{\text{rand}}$(%)	$\mathcal{A}_{\text{adv}}$(%)	Capital ($)	Size (%)
F	Linear	34.66	34.39	16.66	26,808	0.4
	MLP	36.56	33.12	7.22	24,055	0.4
	LSTM	36.40	37.26	21.76	10,340	< 0.1
	Model	$\mathcal{A}_{\text{test}}$(%)	$\mathcal{A}_{\text{rand}}$(%)	$\mathcal{A}_{\text{adv}}$(%)	Capital($)	Size (%)
GE	Linear	36.41	35.03	19.53	34,513	0.6
	MLP	36.87	33.65	9.87	23,853	0.4
	LSTM	35.65	33.01	20.59	12,580	< 0.1
	Model	$\mathcal{A}_{\text{test}}$(%)	$\mathcal{A}_{\text{rand}}$(%)	$\mathcal{A}_{\text{adv}}$(%)	Capital ($)	Size (%)
NVDA	Linear	34.15	37.47	28.98	53,805	2.5
	MLP	35.69	34.39	27.60	49,556	2.4
	LSTM	37.11	35.88	35.56	5,327	< 0.1

Table 3.2: Model performance on clean and perturbed inputs. We denote accuracy on the test data, accuracy on randomly perturbed data, and accuracy on adversarially perturbed data by $\mathcal{A}_{\text{test}}$, $\mathcal{A}_{\text{rand}}$, and $\mathcal{A}_{\text{adv}}$, respectively. We also report the average capital the adversary must have before making an effective perturbation, and the average relative size of successful attacks, both of which are computed as described in Section 3.4.2. The average cost to the attacker per asset is less than $1.00.

valuation models are robust to noise but not to cleverly placed orders. MLPs, while generally of higher natural accuracy than linear models, are also significantly less robust. In all experiments, MLPs achieve lower robust accuracy, even with smaller perturbations. On the other hand, LSTM models seemingly employ gradient masking as they are far more difficult to attack using gradient information. To further confirm the gradient masking hypothesis, we see in Section 3.5 that LSTM models are the most easily fooled models in the face of transferred universal perturbations.

3.4.5 The Effect of High Price-to-Size Ratio

We found that valuation models for equities with high price per share and low trading volume were difficult to attack. For such stocks, the increments by which an adversary can perturb the size-weighted average are coarse, so gradient-based methods with rounding are ineffective. For example, in the data we use, Nvidia stock costs $\sim$ \$180/share compared to $\sim$ \$9/share for Ford stock during the data range we considered. This fact, combined with roughly $5\times$ lower trading volume, makes it difficult to perform low cost attacks, or attacks with low relative size, on Nvidia. As a result of this quantization effect, we are almost unable to impair models on Nvidia. This observation leads us to believe that price prediction on assets with this property is more robust in general (at least using the class of SWA-based models studies in our setting). In the ensuing sections, we focus on Ford and General Electric stocks, which have relatively higher volume and lower cost per share.

3.5 Universal Transfer Attack: A More Feasible Threat Model

The adversarial attacks discussed above provide a useful basis for analyzing model robustness but do not pose any real security threat because of three unrealistic assumptions. First, the attacker is performing an untargeted attack, meaning that the attacker knows the class that a snippet belongs to (i.e., whether it will go up, down, or remain), and is only trying to cause the label to change to some other class. Furthermore, attacks are crafted while looking at every row of the order book simultaneously. For this reason, the attacker may use knowledge about the end of a snippet when deciding what orders to place in the book at the beginning of the snippet. Third, the attacker must know their victim's architecture and parameters. In short, the strategies displayed above require the attacker to know the future and their opponent's proprietary technology.

In this section, we consider making *universal perturbations* – we craft a single attack that works on a large number of historical training snippets with the hope that it transfers to unseen testing snippets. These attacks can be deployed in real time without knowing the future. Furthermore, we use targeted attacks, so that the attacker may anticipate the behavior of the victim. In order to add even more realism, we use transfer attacks in which the attacker does not know the parameters of the model they are attacking. In this situation, we train surrogate models and use universal perturbations generated on the surrogates to attack the victim.

We craft universal perturbations by solving

$$\min_{\delta} \sum_{i} \mathcal{L}(f(\{\mathrm{SWA}(\{\mathbf{x}_i + \delta_i)\}), y_t) + \gamma R(\delta),$$

where $\mathcal{L}$ is cross-entropy loss, $\{\delta_i\} = p(\{\mathbf{a}_i\})$, $\{x_i\}$ are the order book snapshots in the training data, y_t is a target label, δ is a universal perturbation to the size entries, and $R(\delta)$ is a penalty term which may be used to encourage the perturbation to require, for example, less size relative to the size on the book. The perturbation, δ, represents the size of orders placed by the adversary at particular price levels and time stamps. That is, the adversary must only consider how many levels a price is from the best offer rather than specific dollar amounts. We solve this problem with stochastic gradient descent by sampling training data and minimizing loss. See the Appendix of [56] for attack hyperparameters. We apply this pre-computed sequence of adversarial orders to the test data by propagating the orders through the order book for each individual input sequence and adding to the original clean snippet. We measure and report the success rate at moving correctly classified inputs from outside the target class into the target class.

We find that universal adversarial perturbations in this setting work well. In particular, we were able to find universal perturbations that are small relative to the order book and yet fool the model a significant percentage of the time. Moreover, we find that these attacks transfer between models, and our relative size penalty is effective at reducing the budget of an attacker while maintaining a high fool rate. See Table 3.3 for targeted universal perturbation transfer attack results.

These targeted universal perturbations are prototypical patterns for convincing a model to predict a particular SWA movement. For example, to force the victim to predict a downwards trend, the adversary places sell orders followed by buy orders and finally more sell orders to feign downwards momentum in the SWA (See Figure 3.2). We find that visualizing universal perturbations is helpful for interpreting why our models make the decisions they do.

	$\mathcal{U}_{\text{LINEAR}}$		$\mathcal{U}_{\text{MLP}}$	
MODEL	FOOLED	SIZE	FOOLED	SIZE
LINEAR	9.5%	1.0%	11.90%	0.8%
MLP	23.75%	1.1%	36.10%	0.8%
LSTM	31.21%	0.9%	40.46%	0.8%

Table 3.3: Model performance on universal attacks (Ford data). We denote the universal perturbations computed on the linear model and on the MLP by $\mathcal{U}_{\text{linear}}$ and $\mathcal{U}_{\text{MLP}}$, respectively. We craft universal adversarial perturbations on the linear model and the MLP and assess the performance of three models under these attacks. Both of these attacks were computed with a penalty on relative size.

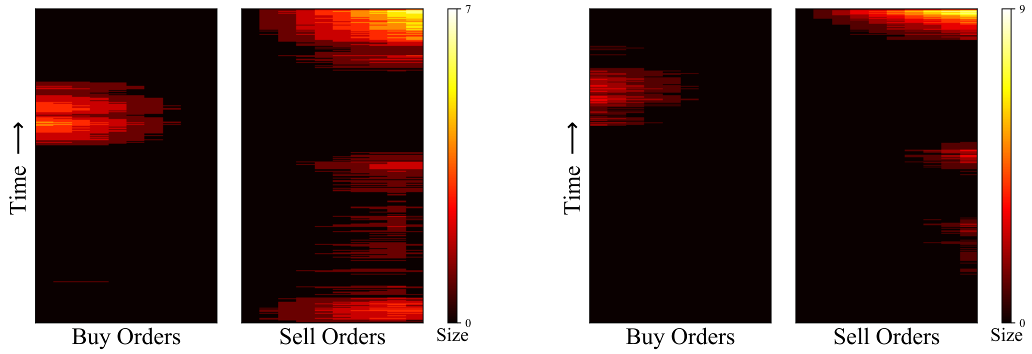


Figure 3.2: Examples of two targeted universal adversarial attacks computed on the same model (MLP trained on Ford data). The perturbation on the top was computed without constraint, and when transferred to test data and a different model, it fooled the victim on 157 inputs (out of 341 correctly classified) and accounts for a relative size on the book of 3.8%. The bottom perturbation is the result of the same process with an added penalty on relative size. The penalized attack shows a sparser perturbation with a relative size of 0.9% while causing misclassification of almost the same number of inputs (123).

3.6 Patterns in Adversarial Orders

We observe patterns in adversarial attacks that can help explain the behavior of classifiers. The non-universal attacks on particular inputs highlight the vulnerability of the valuation models. For example, see Figure 3.3, where the perturbation is so small compared to the size on the book that a human would not distinguish the attacked signal from the clean one. Yet, the attack is concentrated on the fringes of the order book, much like a spoofing attack. Figure 3.2 shows a case in which a universal adversary has learned an interpretable perturbation in which it creates a local minimum in the size-weighted average by alternately placing perturbations on opposite sides of the book. Our relative size penalty visibly decreases the size and concentration of the perturbation without decreasing effectiveness.

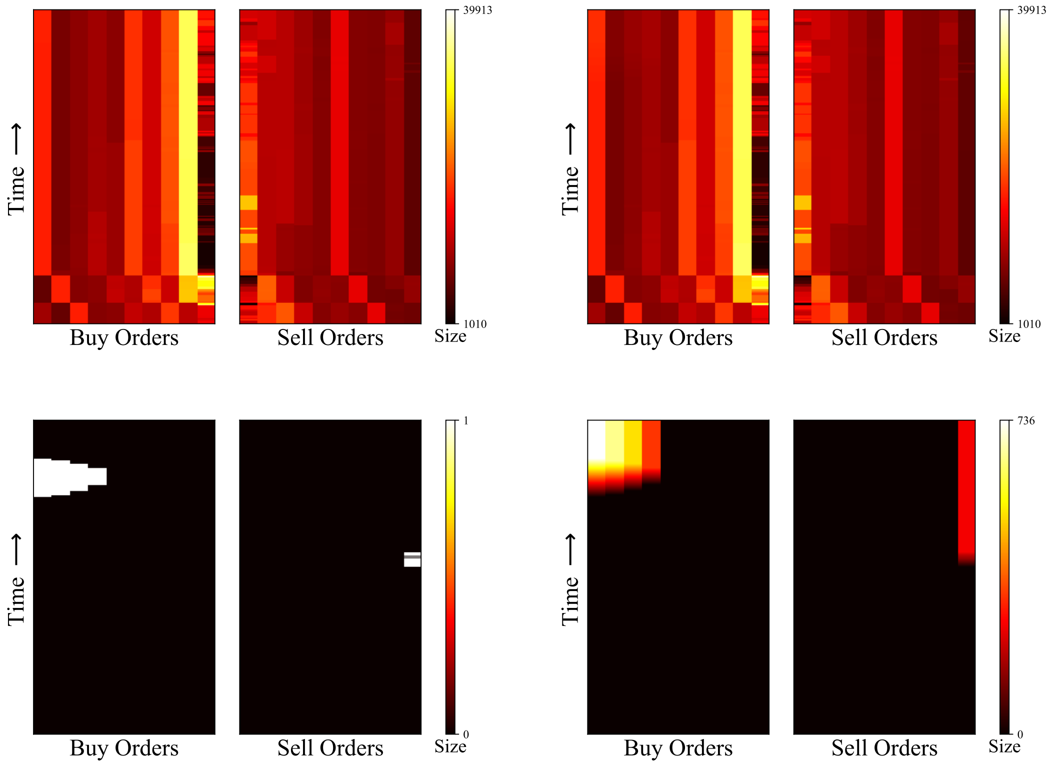


Figure 3.3: The unperturbed and perturbed order books (top left and top right, respectively). The bottom two images show the unpropagated and propagated adversarial perturbations (left and right, respectively). In this instance, the addition of the bottom right image to the clean signal in the top left yields the perturbed input in the top right. Note that the difference between the perturbed and unperturbed signals is not noticeable here, where the model was fooled into incorrectly classifying the SWA as going up.

Universal perturbations bring to light two important features of these price predictors. The first is that targeted universal perturbations, which are generally less effective, have a major impact on the performance of these models. The models are vulnerable even to these weak attacks. Second, the targeted universal attacks expose interpretable model behavior. In Figure 3.2, we see that the attacker creates local extrema in the order book which cause the model to anticipate mean reversion.

3.7 Discussion

This work introduces adversarial attacks to financial models both as a method for understanding model sensitivities and also as a potential threat. As seen in other applications of adversarial machine learning, there are robustness tradeoffs to be leveraged, and there is no free lunch. While neural network models perform better at pattern recognition in this setting than traditional linear valuations, we also find them to be less robust. We further notice that the same adversarial patterns that fool one model also fool others and that these patterns are highly interpretable to humans. The transferability of these attacks, combined with their ability to be effective with a small attack budget, suggests that they could possibly be leveraged by a malicious agent with limited knowledge of a victim.

Chapter 4: Adversarially Robust Distillation

Knowledge distillation is effective for producing small, high-performance neural networks for classification, but these small networks are vulnerable to adversarial attacks. This paper studies how adversarial robustness transfers from teacher to student during knowledge distillation. We find that a large amount of robustness may be inherited by the student even when distilled on only clean images. Second, we introduce *Adversarially Robust Distillation* (ARD) for distilling robustness onto student networks. In addition to producing small models with high test accuracy like conventional distillation, ARD also passes the superior robustness of large networks onto the student. In our experiments, we find that ARD student models decisively outperform adversarially trained networks of identical architecture in terms of robust accuracy, surpassing state-of-the-art methods on standard robustness benchmarks. Finally, we adapt recent fast adversarial training methods to ARD for accelerated robust distillation. This work was conducted with Liam Fowl, Soheil Feizi, and Tom Goldstein [51]. My contribution was conceiving of the central ideas of the project, implementing the algorithms, conducting most experiments from all sections, and writing most of the paper.

4.1 Introduction

State-of-the-art deep neural networks for many computer vision tasks have tens of millions of parameters, hundreds of layers, and require billions of operations per inference [68, 155]. However, networks are often deployed on mobile devices with limited compute and power budgets or on web servers without GPUs. Such applications require efficient networks with light-weight inference costs [30, 44, 135, 157].

Knowledge distillation was introduced as a way to transfer the knowledge of a large pre-trained teacher network to a smaller light-weight student network [73]. Instead of training the student network on one-hot class labels, distillation involves training the student network to emulate the outputs of the teacher. Knowledge distillation yields compact student networks that surpass the performance achievable by training from scratch without a teacher [73].

Classical distillation methods achieve high efficiency and accuracy but neglect security. Standard neural networks are easily fooled by *adversarial examples*, in which small perturbations to inputs cause mis-classification [57, 156]. This phenomenon leads to major security vulnerabilities for high-stakes applications like self-driving cars, medical diagnosis, and copyright control [102, 132, 137]. In such domains, efficiency and accuracy are not enough – networks must also be adversarially robust.

We study distillation methods that produce robust student networks. Unlike conventional adversarial training [114, 149], which encourages a network to output correct labels within an ϵ-ball of training samples, the proposed *Adversarially Robust Distillation* (ARD) instead encourages student networks to mimic their teacher's output within an ϵ-ball of training samples. See Figure 4.1 for a diagram of the ARD pipeline. Thus, ARD is a natural analogue of adversarial training but in the context of distillation. Formally, we solve the following optimization problem.

Definition 4.1 (Adversarially Robust Distillation)**.** Adversarially Robust Distillation (ARD) is defined as the minimax problem,

$$\min_{\theta} \mathbb{E}_{(X,y)\sim\mathcal{D}} \left[\alpha t^2 \underbrace{\mathrm{KL}(S_\theta^t(X+\delta_\theta), T^t(X))}_{\text{Adversarially Robust Distillation loss}} + (1-\alpha) \underbrace{\mathcal{L}(S_\theta^t(X), y)}_{\text{classification loss}} \right],$$

where $\delta_\theta = \arg\max_{\|\delta\|_p<\epsilon} \mathcal{L}(S_\theta^t(X+\delta), y)$, T and S are teacher and student networks, and $\mathcal{D}$ is the data generating distribution.

In this work, we focus on image datasets. See Section 4.5 for a more thorough description. Below, we summarize our contributions in this section:

- We show that knowledge distillation using only natural images can preserve much of the teacher's robustness to adversarial attacks. This property enables the production of efficient robust models without the expensive cost of adversarial training. See Section 4.4 for more details.

- We introduce Adversarially Robust Distillation (ARD) for producing small robust student networks. In our experiments, ARD students exhibit higher robust accuracy than adversarially trained models with identical architecture, and ARD often exhibits higher natural accuracy simultaneously. Interestingly, ARD students may exhibit even higher robust accuracy than their teacher. See Table 4.1 for the results of our experiments. 4.5

- We accelerate our method for efficient training by adapting fast adversarial training methods. 4.5.4

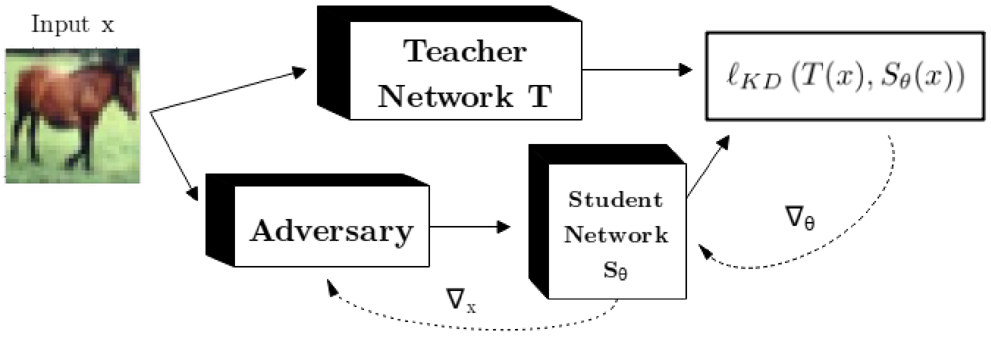


Figure 4.1: Adversarially Robust Distillation (ARD) works by minimizing discrepancies between the outputs of a teacher on natural images and the outputs of a student on adversarial images.

Model	Robust Accuracy ($\mathcal{A}_{adv}$)
AT ResNet18 teacher	44.46%
AT ResNet18 $\rightarrow$ MobileNetV2	38.21%
AT ResNet18 $\xrightarrow{\textbf{ARD}}$ MobileNetV2	**50.22%**

Table 4.1: Performance of an adv. trained (AT) teacher network and its student on CIFAR-10, where robust accuracy ($\mathcal{A}_{adv}$) is with respect to a 20-step PGD attack as in [114]. Here "$\rightarrow$" denotes "knowledge distillation onto" and "$\xrightarrow{\textbf{ARD}}$" denotes "adversarially robust distillation onto".

Table 4.1 shows that a student may learn robust behavior from a robust teacher, even if it only sees clean images during distillation. Our method, ARD, outperforms knowledge distillation for producing robust students. To gain initial intuition for the differences between these methods, we visualize decision boundaries for a toy problem in Figure 4.2. Randomly generated training data are depicted as colored dots with boxes showing the desired ℓ_∞ robustness radius. Background colors represent the classification regions of the networks (10-layer teacher and 5-layer student). In each case, the network achieves perfect training accuracy. A training point is vulnerable to attack if its surrounding box contains multiple colors. Results in Figure 4.2 are consistent with our experiments in Section 4.5 on more complex datasets. We see in these decision boundary plots that knowledge distillation from a robust teacher preserves some robustness, while ARD produces a student who closely mimics the teacher.

4.2 Related Work

Early schemes for compressing neural networks involved binarized weights to reduce storage and computation costs [30]. Other efforts focused on speeding up calculations via low-rank regularization and pruning weights to reduce computation costs [104, 157]. Knowledge distillation teaches a student network to mimic a more powerful teacher [73]. The student is usually a small, lightweight architecture like MobileNetV2 (MNV2) [135].

Knowledge distillation has also been adapted for robustness in a technique called *defensive distillation* [125]. In this setting, the teacher and student have identical architectures. An initial network is trained on class labels and then distilled at temperature t onto a network of identical architecture. Defensive distillation improves robustness to a certain ℓ_0 attack [125]. However, defensive distillation gains robustness due to gradient masking, and this defense has been broken using

ℓ_0, ℓ_∞, and ℓ_2 attacks [20, 21].

Various methods exist that modify networks to achieve robustness. Some model-specific variants of adversarial training utilize surrogate loss functions to minimize the difference in network output on clean and adversarial data [113, 178], feature denoising blocks [167], and logit pairing to squeeze logits from clean and adversarial inputs [84]. Still other methods, such as JPEG compression, pixel deflection, and image superresolution, are model-agnostic and minimize the effects of adversarial examples by transforming inputs [44, 120, 129].

Recently, there has been work on defensive-minded compression. The authors of [163] and [180] study the preservation of robustness under quantization. Defensive Quantization involves quantizing a network while minimizing the Lipschitz constant to encourage robustness [108]. While quantization reduces space complexity, it does not reduce the number of Multiply-Add (MAdd) operations needed for inference, although operations performed at lower precision may be faster depending on hardware. Moreover, Defensive Quantization is not evaluated against strong attackers. Another compression technique involves pruning [143]. Pruning does reduce the number of parameters in a network, but it does not decrease network depth and thus may not accelerate inference. Additionally, this work does not achieve high compression ratios and does not achieve competitive performance on CIFAR-10. Pruning maintains the same architectural framework and does not allow a user to compress a state-of-the-art large robust network into a lightweight architecture of their choice. Creating small robust models is also of interest for the few-shot setting. Adversarial querying approaches this problem from the meta-learning perspective

[52].

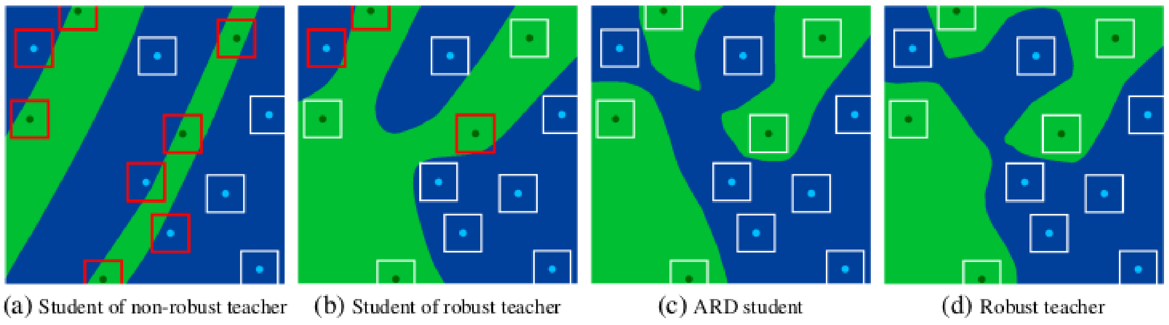


Figure 4.2: A student distilled from a robust teacher is more robust than a naturally trained network, but ARD produces a more robust network than either and closely mimics the teacher's decision boundary. Adversarially vulnerable training points have ℓ_∞ boxes outlined in red.

4.3 Problem Setup

Knowledge distillation employs a teacher-student paradigm in which a small student network learns to mimic the output of an often much larger teacher model [73]. Knowledge distillation entails the minimization problem,

$$\min_\theta \ell_{\mathrm{KD}}(\theta), \quad \ell_{\mathrm{KD}}(\theta) = \mathbb{E}_{X\sim\mathcal{D}}\left[\mathrm{KL}(S_\theta^t(X), T^t(X))\right], \tag{4.1}$$

where KL is KL divergence, S_θ is a student network with parameters θ, T is a teacher network, t is a positive temperature constant, and X is an input to the networks drawn from data generating distribution $\mathcal{D}$. In this work, we focus on distributions of natural images. The temperature constant refers to a number by which the logits are divided before being fed into the softmax function. Intuitively, knowledge distillation involves minimizing the average distance from the student's output to the teacher's output over data from a distribution. The softmax outputs of the teacher network, also referred to as soft labels, may be more informative than true data labels alone. In [73], the authors suggest using a linear combination of the loss function, $\ell_{\mathrm{KD}}(\theta)$, and the cross-entropy between the softmax output of the student network and the one-hot vector representing the true label in order to improve natural accuracy of the student model, especially on difficult datasets. In this case, we have the loss function

$$\ell_{\mathrm{KD}}(\theta) = \mathbb{E}_{(X,d)\sim\mathcal{D}}\big[\alpha t^2 \,\mathrm{KL}(S^t_\theta(X), T^t(X) + (1-\alpha)\mathcal{L}(S^t_\theta(X), y)\big], \tag{4.2}$$

where $\mathcal{L}$ is the standard cross-entropy loss, and y is the label. In our experiments, we use $\alpha = 1$ except where otherwise noted. We tune α (where $0 \leq \alpha \leq 1$) in select cases in order to trade robustness for natural accuracy to surpass some previous robustness methods in both robust and natural accuracy simultaneously. We investigate if students trained using knowledge distillation inherit their teachers' robustness to adversarial attacks. We also combine this method with adversarial training.

Adversarial training is another method for encouraging robustness to adversarial attacks during training [149]. Adversarial training involves the minimax optimization problem,

$$\min_\theta \mathbb{E}_{(X,y)\sim\mathcal{D}}\left[\max_{\|\delta\|_p<\epsilon} L_\theta(X+\delta, y)\right], \tag{4.3}$$

where $L_\theta(X+\delta, y)$ is the loss of network with parameters θ, input X perturbed by

δ, and label y. Adversarial training encourages a student to produce the correct label in an ϵ-ball surrounding data points. Virtual Adversarial Training (VAT) and TRADES instead use as a loss function a linear combination of cross-entropy loss and KL divergence between the network's softmax output from clean input and from adversarial input [113, 178]. The KL divergence term acts as a consistency regularizer which trains the neural network to produce identical output on a natural image and adversarial images generated from the natural image. As a result, this term encourages the neural network's output to be constant in ϵ-balls surrounding data points.

Knowledge distillation is useful for producing accurate student networks when highly accurate teacher networks exist. However, the resulting student networks may not be robust to adversarial attacks [21]. We combine the central ideas of knowledge distillation and adversarial training to similarly produce robust student networks when robust teacher networks exist.

We focus on adversarial robustness to ℓ_∞ attacks since these are pervasive in the robustness literature. Thus, we carry out both adversarial training and ARD with FGSM-based PGD ℓ_∞ attacks similarly to [114, 178]. We re-implemented the methods from these papers to perform adversarial training and to establish performance baselines. In our experiments, we use WideResNet (34-10) (WRN) and ResNet18 teacher models as well as MobileNetV2 (MNV2) students [68, 135, 172]. Adversarial training and TRADES teacher models are as described in [114, 178]. We aim to to use our combination of knowledge distillation and adversarial training to produce efficient networks robust to ℓ_∞ attacks.

4.4 Adversarial robustness is preserved under knowledge distillation

The softmax output of a neural network classifier trained with cross-entropy loss estimates the posterior distribution over class labels on the training data distribution [17]. Empirical study of distillation suggests that neural networks perform better when trained on the softmax output of a powerful teacher than when trained on only class labels [73]. The distribution of images generated by adversarial attacks with respect to a particular model and a data generating distribution, D, may differ from the distribution of natural images generated by D. Distillation from a naturally trained teacher is known to produce student models which are not robust to adversarial attacks [21]. Thus, we suspected that a non-robust teacher network trained on natural images would be poorly calibrated for estimating posterior probabilities on the distribution of images generated by adversarial attacks. On the other hand, an adversarially trained teacher network might provide a more accurate estimate of posterior probabilities on this distribution. We compare the robustness of student models distilled from both naturally trained and adversarially trained teachers. If we distill from an adversarially trained teacher, will the student inherit robustness?

If robustness transfers from teacher to student, we can harness state-of-the-art robust teacher networks to produce accurate, robust, and efficient student networks. Moreover, since adversarial training is slow due to the bottleneck of crafting adversarial samples for every batch, we could create many different student networks from one teacher, and adversarial training would only need to be performed once. This routine of training a robust teacher and then distilling onto robust students would be far more time efficient than training many robust student networks individually.

4.4.1 Non-robust teachers produce non-robust students

To establish a baseline for comparison, we distill a non-robust ResNet18 teacher and evaluate against a 20-step PGD attack as in [114]. We verify known results showing that defensive distillation is ineffective for producing adversarially robust students [21]. The authors of [21] noticed that the original defensive distillation paper defends against a particularly ill-suited attacker, and they introduce a stronger attacker. Since this work, adversarial attacks have become stronger and even more capable of breaking defensive distillation. In this section, we demonstrate that the strong attacker we use in our tests adequately cripples student networks distilled from a naturally trained teacher.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
ResNet18 teacher	94.75%	0.0%
ResNet18 $\rightarrow$ ResNet18	94.92%	0.0%
ResNet18 $\rightarrow$ MNV2	93.53%	0.0%

Table 4.2: Performance of a naturally trained teacher network and its students distilled (with $t = 30$) on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [114]. $\mathcal{A}_{nat}$ denotes natural accuracy.

4.4.2 Robust teachers can produce robust students, even distilling on only clean data

Next, we substitute in a robust adversarially trained ResNet18 teacher network and run the same experiments. We find that our new student networks are far more robust than students of the non-robust teacher. See Table 4.3 for empirical performance. In fact, the student networks acquire most of the teacher's robust accuracy. These results confirm that robust lightweight networks may indeed be produced cheaply through knowledge distillation without undergoing adversarial training.

Student Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
AT ResNet18 teacher	76.54%	44.46%
AT ResNet18 $\rightarrow$ ResNet18	76.13%	40.13%
AT ResNet18 $\rightarrow$ MNV2	76.86%	38.21%

Table 4.3: Performance of an adversarially trained ResNet18 teacher network and student networks of various sizes distilled on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [114].

4.4.3 Not all robust networks are good teachers, and robustness does not transfer on some datasets

In the previous experiments, we see that a student network may inherit a significant amount of robustness from a robust teacher network during knowledge distillation. However, some robust teachers are not conducive to this robustness transfer. We use robust WideResNet (34-10) models trained using adversarial training and TRADES to show that while these models do transfer robustness during knowledge distillation, they transfer less than the weaker ResNet18 teacher network from the previous section. See Table 4.4. Additionally, a robust WRN teacher model transfers almost no robustness under knowledge distillation against 20-step PGD untargeted attacks on CIFAR-100, a much harder dataset for robustness to untargeted attacks than CIFAR-10. See Table 4.5. Further experiments show that robustness transfer diminishes rapidly as we decrease α from our default value of 1.

For these reasons, we develop ARD for distilling a variety of teachers in order to produce robust students. Using ARD, robustness is preserved on architectures and datasets that do not transfer robustness under vanilla knowledge distillation.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
AT WRN teacher	84.41%	45.75%
TRADES WRN teacher	84.92%	56.61%
AT WRN $\rightarrow$ MNV2	92.49%	5.46%
TRADES WRN $\rightarrow$ MNV2	85.6%	21.69%

Table 4.4: Robust WRN teacher models and their students on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [114].

4.5 Improving the robustness of student models with Adversarially Robust Distillation (ARD)

We combine the central machinery from knowledge distillation, adversarial training, and TRADES/VAT to produce small robust student models from much larger robust teacher models using a method we call *Adversarially Robust Distillation.* ARD not only produces more robust students than knowledge distillation, but ARD also works for teachers and datasets on which knowledge distillation is ineffective for transferring robustness. Our procedure is a natural analogue of adversarial

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
AT WRN teacher	59.9%	28.36%
AT WRN $\rightarrow$ MNV2	25.54%	1.30%
AT WRN $\rightarrow$ MNV2 ($\alpha = 0.95$)	75.94%	0.02%
AT WRN $\rightarrow$ MNV2 ($\alpha = 0.93$)	76.38%	0.00%

Table 4.5: Robust teacher network and its students on CIFAR-100, where robust accuracy is with respect to a 20-step PGD attack as in [114].

training but in a distillation setting. During standard adversarial training, we encourage a network to produce the ground truth label corresponding to a clean input when the network is exposed to an adversary. Along the same lines, our method treats the teacher network's softmax output on clean data as the ground truth and trains a student network to reproduce this ground truth when exposed to adversarial examples. We start with a robust teacher model, T, and we train the student model S_θ by solving the following optimization problem (Definition 4.1 revisited):

$$\min_\theta \mathbb{E}_{(X,y)\sim\mathcal{D}}\big[\alpha t^2\,\mathrm{KL}(S^t_\theta(X+\delta_\theta), T^t(X)) + (1-\alpha)\mathcal{L}(S^t_\theta(X), y)\big],$$

where $\delta_\theta = \arg\max_{\|\delta\|_p<\epsilon} \mathcal{L}(S^t_\theta(X+\delta), y)$, $\mathcal{L}$ is cross-entropy loss, and we divide the logits of both student and teacher models by temperature term t during training. The t^2 term is used as in [73] since dividing the logits shrinks the gradient. The cross-entropy loss, which encourages natural accuracy, is a standard training loss. Thus, α is a hyperparameter that prioritizes similarity to the teacher over natural accuracy. In our experiments, we set $\alpha = 1$ except where otherwise noted, eliminating the cross-entropy term except when additional natural accuracy is needed for comparison with baseline models. We find that lower values of α are useful for improving performance on harder classification tasks. Our training routine involves the following procedure:

Algorithm 1: Adversarially Robust Distillation (ARD)

```
Require: Student and teacher networks $S$ and $T$, learning rate $\gamma$, dataset
  $\{(\mathbf{x}_i, y_i)\}$, number of steps, K, per PGD attack, and $\epsilon$ maximum attack
  radius.
Initialize $\theta$, the weights $S$;
for Epoch = 1,...,$N_{epochs}$ do
    for Batch = 1,...,$N_{batches}$ do
        Construct adv. example $\mathbf{x}'_i$ for each $\mathbf{x}_i \in$ Batch by maximizing
          cross-entropy between $S_\theta(\mathbf{x}'_i)$ and $\mathbf{y}_i$ constrained to $\|\mathbf{x}_i - \mathbf{x}'_i\|_p < \epsilon$
          using K-step PGD.
        Compute $\nabla_\theta \ell_{\text{ARD}}(\{\mathbf{x}_i\}, \theta) = \sum_i \nabla_\theta[\alpha t^2 \,\text{KL}(S^t_\theta(\mathbf{x}'_i), T^t(\mathbf{x}_i))$
          $+(1-\alpha)\mathcal{L}(S^t_\theta(X), \mathbf{y})]$, over the current batch.
        $\theta \leftarrow \theta - \gamma \nabla_\theta \ell_{\text{ARD}}(\{\mathbf{x}_i\}, \theta)$
```

4.5.1 ARD works with teachers that fail to transfer robustness under knowledge distillation

In Section 4.4, we saw that that some teacher networks do not readily transfer robustness. We see through our experiments in Table 4.6 that ARD is able to create robust students from teachers whose robustness failed to transfer during knowledge distillation. TRADES and adversarially trained MobileNetV2 (MNV2) models are each outperformed in both natural and robust accuracy simultaneously by an ARD variant. In these experiments, the WideResNet teacher model contains 20 times as many parameters and performs 70 times as many MAdd operations as the MobileNetV2 student.

4.5.2 ARD works on datasets where knowledge distillation fails

In Section 4.4, we saw that a student network inherited little robustness from an adversarially trained teacher network on CIFAR-100. This dataset is very difficult

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
TRADES WRN teacher	84.92%	56.61%
AT MNV2	80.50%	46.90%
TRADES MNV2	83.59%	44.79%
TRADES WRN $\xrightarrow{\textbf{ARD}}$ MNV2	82.63%	**50.42%**
TRADES WRN $\xrightarrow{\textbf{ARD}}$ MNV2 ($\alpha = 0.95$)	**84.70%**	46.28%

Table 4.6: Performance on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [114].

to protect from untargeted attacks because it contains many classes that are similar in appearance. In Table 4.7, we see that a MobileNetV2 student model trained on CIFAR-100 using ARD from an adversarially trained WideResNet is significantly more robust than an adversarially trained MobileNetV2. In fact, the ARD model is nearly as robust as its teacher.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
AT WRN teacher	59.90%	28.36%
AT MNV2	55.62%	22.80%
AT WRN $\xrightarrow{\textbf{ARD}}$ MNV2 ($\alpha = 0.93$)	55.47%	**27.64%**

Table 4.7: Performance on CIFAR-100, where robust accuracy is with respect to a 20-step PGD attack as in [114].

4.5.3 ARD can produce networks more robust than their teacher

In some experiments, ARD student networks are more robust than their teacher. Interestingly, this behavior does not depend on distilling from a high-capacity network to a low capacity network; distilling ResNet18 onto itself and MobileNetV2 onto itself using ARD results in far better robustness than adversarial training alone.

We seek to understand if this increased robustness is caused by differences between the student and teacher architectures, and so we use ARD to distill a teacher network onto a student network with an identical architecture. In our experiments, ARD boosted the robustness of both the ResNet18 and MobileNetV2 models. See Table 4.8.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
AT ResNet18	76.54%	44.46%
AT MNV2	80.50%	46.90%
AT ResNet18 $\xrightarrow{\textbf{ARD}}$ ResNet18	79.49%	**51.21%**
AT MNV2 $\xrightarrow{\textbf{ARD}}$ MNV2	81.22%	47.95%
AT ResNet18 $\xrightarrow{\textbf{ARD}}$ MNV2	79.47%	50.22%

Table 4.8: Performance on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [114].

4.5.4 Accelerating ARD using fast adversarial training methods

The ARD procedure described above takes approximately the same amount of time as adversarial training. Adversarial training is slow since it requires far more gradient calculations than natural training. Several methods have been proposed recently for accelerating adversarial training [144, 175]. We similarly accelerate performance for ARD by adapting "free" adversarial training to distillation. This version, *Fast-ARD*, described in Algorithm 2, is equally fast to knowledge distillation. See Table 4.10 for a list of training times. During training, we replay each mini-batch several times in a row. On each replay, we simultaneously compute the gradient of the loss with respect to the image and parameters using the same backward pass. Then, we update the adversarial attack and the network's parameters simultaneously. Empirically, Fast-ARD produces less robust students than the full

ARD above, but it produces higher robust accuracy compared to models with identical architectured trained using existing accelerated free adversarial training methods as seen in Table 4.9. Furthermore, Fast-ARD from a TRADES WideResNet onto MobileNetV2 produces a more robust student than our most robust MobileNetV2 produced during vanilla knowledge distillation and in the same amount of training time. Our accelerated algorithm is detailed in Algorithm 2.

Algorithm 2: Fast-ARD with free adversarial training

```
Requires: Student and teacher networks $S$ and $T$, learning rate $\gamma$, norm $p$,
  dataset $\{(\mathbf{x}_i, y_i)\}$, and attack step-size $r$ and radius $\epsilon$
Initialize $\theta$, the weights of network $S$, and set $\boldsymbol{\delta} = 0$.
for Epoch = 1,..., $\frac{N_{epochs}}{m}$ do
    for Batch = 1,...,$N_{batches}$ do
        for j = 1,..., m do
            For $\mathbf{x}_i \in$ Batch, find new perturbation $\delta_i'$ by maximizing
              cross-entropy between $S_\theta(\mathbf{x}_i + \delta_i + \delta_i')$ and $\mathbf{y}_i$ over $\delta_i'$,
              constrained to $\|\delta_i + \delta_i'\|_p < \epsilon$, using a 1-step PGD attack.
            Compute the gradient of the loss function $\nabla_\theta \ell_{\text{ARD}}(\{\mathbf{x}_i\}, \theta) =$
              $\sum_i \nabla_\theta [\alpha t^2 \, \text{KL}(S_\theta^t(\mathbf{x}_i'), T^t(\mathbf{x}_i)) + (1-\alpha)\mathcal{L}(S_\theta^t(X), \mathbf{y})]$, over
              current batch.
            $\delta_i \leftarrow \delta_i + \delta_i'$
            $\theta \leftarrow \theta - \gamma \nabla_\theta \ell_{\text{ARD}}(\{\mathbf{x}_i\}, \theta)$
```

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
Free trained MNV2 (m=4)	82.63%	23.13%
TRADES WRN $\xrightarrow{\textbf{F-ARD}}$ MNV2 (m=4)	**83.51%**	**37.07%**
Free trained MNV2 (m=8)	72.30%	27.96%
TRADES WRN $\xrightarrow{\textbf{F-ARD}}$ MNV2 (m=8)	76.38%	36.85%

Table 4.9: Performance of MobileNetV2 classifiers, free trained and Fast-ARD, on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [114]. "$\xrightarrow{\textbf{F-ARD}}$" denotes "Fast-ARD onto".

Model	Time (hrs)
AT MNV2	41.09
TRADES WRN $\rightarrow$ MNV2	5.13
TRADES WRN $\xrightarrow{\textbf{ARD}}$ MNV2	41.06
TRADES WRN $\xrightarrow{\textbf{F-ARD}}$ MNV2 (m=4)	5.15
TRADES WRN $\xrightarrow{\textbf{F-ARD}}$ MNV2 (m=8)	**5.10**

Table 4.10: Training times for adversarial training, clean distillation, ARD, and Fast-ARD. Each model was trained with 200 parameter updates for each training image (equivalent of 200 epochs). All models were trained on CIFAR-10 with a single RTX 2080 Ti GPU and identical batch sizes. The AT model and the ARD model were trained with a 10-step PGD attack. "$\xrightarrow{\textbf{F-ARD}}$" denotes "Fast-ARD onto".

4.5.5 ARD and Fast-ARD models are more robust than their adversarially trained counterparts

While 20-step PGD is a powerful attack, we also test ARD against other ℓ_∞ attackers including Momentum Iterative Fast Gradient Sign Method [37], DeepFool [117], 1000-step PGD, and PGD with random restarts. We find that ARD and Fast-ARD outperform adversarial training and free training respectively across all attacks we tried. See Table 4.11. A common criticism of early robustness papers is that they do not test their methods against sufficiently powerful attacks. In order to assuage these concerns, we test our models against some of the most powerful known attacks.

4.6 Space and time efficiency of student and teacher models

We perform our experiments for ARD with WideResNet (34-10) and ResNet18 teacher models as well as a MobileNetV2 student model. We consider two network

Model	MI-FGSM$_{20}$	DeepFool	1000-PGD	20-PGD 100-restarts
AT MNV2	50.82%	57.74%	46.51%	46.79%
$\xrightarrow{\textbf{ARD}}$ MNV2	**55.16%**	**64.61%**	**49.98%**	**50.30%**
Free trained MNV2 (m=4)	30.60%	41.09%	22.23%	22.94%
$\xrightarrow{\textbf{F-ARD}}$ MNV2(m=4)	**44.78%**	**60.03%**	**36.01%**	**36.88%**

Table 4.11: Robust validation accuracy of adversarially trained and free trained MobileNetV2 and TRADES WRN ARD (and Fast-ARD) onto MobileNetV2 on CIFAR-10 under various attacks. All attacks use $\epsilon = \frac{8}{255}$.

qualities for quantifying compression. First, we study space efficiency by counting the number of parameters in a network. Second, we study time complexity. To this end, we compute the multiply-add (MAdd) operations performed during a single inference. The real time elapsed during this inference will vary as a result of implementation and deep learning framework, so we use MAdd, which is invariant under implementation and framework, to study time complexity.

The WideResNet and ResNet18 teachers we employ contain $\sim$ 46.2M and $\sim$ 11.2M parameters respectively, while the MobileNetV2 student contains $\sim$ 2.3M parameters. A forward pass through the WRN and ResNet18 teacher models takes $\sim$ 13.3B MAdd operations and $\sim$ 1.1B MAdd operations respectively, while a forward pass through the student model takes $\sim$ 187M MAdd operations. To summarize these network traits, compared to a WideResNet (34-10) teacher, the MobileNetV2 student model:

- Contains $\sim$ 5% as many parameters
- Performs $\sim$ 1.4% as many MAdd operations during a forward pass

4.7 Experimental details

We train our models for 200 epochs with SGD, momentum of 0.9, and weight-decay of $2e-4$. Fast-ARD models are trained for $\frac{200}{m}$ epochs so that they make the same number of gradient computations as ARD. We use an initial learning rate of 0.1, and we decrease the learning rate by a factor of 10 on epochs 100 and 150 (epochs $\frac{100}{m}$ and $\frac{150}{m}$ for Fast-ARD). We use a temperature term of 30 for CIFAR-10 and 5 for CIFAR-100. To craft adversarial examples during training, we use FGSM-based PGD with 10 steps, ℓ_∞ attack radius of $\epsilon = \frac{8}{255}$, a step size of $\frac{2}{255}$, and a uniformly distributed random start within the ℓ_∞ radius. A PyTorch implementation of ARD can be found at https://github.com/goldblum/AdversariallyRobustDistillation

4.8 The effects of hyperparameters and data augmentation for knowledge distillation of robust teacher models

We show the results of experiments in Table 4.12 and Table 4.13, varying temperature and α. We see that only very low temperature terms are not conducive to robustness preservation, but knowledge distillation is not highly sensitive to temperature, and a wide array of temperature terms are effective for producing robust students. On the other hand, robustness transfer decays rapidly when α decreases. This effect produces a robustness-accuracy tradeoff.

Data augmentation yields dramatic improvements for robustness transfer as the student gets to see the teacher's behavior at more data points, as can be seen in

Table 4.14. A natural idea is to teach the student the teacher's behavior at adversarial points as well. However, this data augmentation technique greatly decreases training speed and does not provide significant improvement.

Temperature	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
1	77.37%	35.44%
10	78.1%	38.55%
30	76.86%	38.21%
50	76.28%	38.5%
100	76.44%	38.65%

Table 4.12: Performance of adversarially trained ResNet18 teacher network distilled onto MobileNetV2 (MN) with different temperature terms on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [15].

α	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
1.00	76.86%	38.21%
0.99	82.93%	28.36%
0.95	91.58%	9.65%
0.90	92.34%	7.96%
0.70	92.33%	7.54%
0.50	92.14%	2.73%

Table 4.13: Performance of adversarially trained ResNet18 teacher network distilled onto MobileNetV2 (MN) with different α terms on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [15].

Data Augmentation	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
No augmentation	75.76%	32.05%
Horizontal flips	76.24%	35.39%
Random crops	76.31%	37.31%
Both flips and crops	76.86%	38.21%
Adv. examples generated from teacher	76.76%	38.33%

Table 4.14: Performance of adversarially trained WideResNet distilled onto MobileNetV2 (MN) with various data augmentation on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [15].

4.9 Notes on loss functions for ARD

We tried other versions of the loss function with the addition of a (non-adversarial) knowledge distillation KL divergence term, which increases natural accuracy slightly but sharply decreases robust accuracy, and the addition of cross-entropy between the softmax of the adversarially attacked student and the one-hot label vectors, which is redundant in combination with the ARD loss and empirically harms robustness. We also explored using $T^t(\mathbf{x}_i')$ instead of $T^t(\mathbf{x}_i)$ in the KL divergence term in the loss function, where $\mathbf{x}_i'$ is an adversarial example generated by the input, $\mathbf{x}_i$. However, the accuracy, both natural and robust, decreased with this loss function. Additionally, passing an adversarial example through the teacher model will slow down training and sharply increase memory consumption if the teacher is much larger than the student. Our method does not require forward passes through the teacher during training as long as logits of the training data are stored ahead of time. We explored generating adversarial attacks during training by maximizing KL divergence instead of cross-entropy. This technique lowers natural accuracy without improving robust accuracy.

In our training routine, we employ data augmentation in the form of random crops and horizontal flips. We intend to explore adaptive data augmentation methods tailored specifically for robust distillation.

4.10 ARD with naturally trained teacher models

ARD encourages a student to produce, for all images within an ϵ-ball of a data point, the teacher's output at that data point. Thus, it seems reasonable to try using ARD with a naturally trained teacher model. As evident in Table 4.15, naturally trained teachers produce robust students, but these students may be less robust than those of robust teachers and less robust than adversarially trained models with identical architecture.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
ResNet18 $\xrightarrow{\textbf{ARD}}$ MobileNetV2	84.18%	44.61%
WRN $\xrightarrow{\textbf{ARD}}$ MobileNetV2	84.43%	42.51%

Table 4.15: Performance of MobileNetV2 students distilled from naturally trained teachers on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [15].

4.11 Improving the speed of ARD by reducing the number of attack steps

Another way to improve the speed of ARD training is to reduce the number of attack steps in order to reduce the number of gradient calculations. In our experiments shown in Table 4.16, reducing the number of attack steps improves natural accuracy while decreasing robust accuracy. This procedure has a similar effect to reducing α. We suggest this strategy over reducing α since it has the added benefit of accelerating training.

Attack Steps (training)	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
4	84.28%	46.49%
6	83.38%	48.35%
10	82.63%	50.42%

Table 4.16: Performance of TRADES WideResNet distilled onto MobileNetV2 using ARD with adversaries generated using different numbers of attack steps on CIFAR-10. Robust accuracy is with respect to a 20-step PGD attack as in [15].

4.12 Sensitivity of ARD to hyperparameters

Compared to knowledge distillation for preserving robustness, ARD is less sensitive to the temperature parameter. We found that varying the temperature parameter does not significantly impact either the natural or robust accuracy of the resulting student. See Table 4.17. Like with knowledge distillation, varying α presents an accuracy-robustness tradeoff. See Table 4.18. However, unlike with knowledge distillation, we find that under ARD, robust accuracy decays far slower as α decreases so that ARD is less sensitive to this parameter.

Temperature	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
1	81.41%	49.57%
5	82.24%	48.77%
10	82.33%	48.93%
30	82.63%	50.42%
50	82.14%	48.97%

Table 4.17: Performance of TRADES WideResNet distilled onto MobileNetV2 using ARD with different temperature terms on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [15].

α	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
1.00	82.63%	50.42%
0.95	84.7%	46.28%
0.90	86.58%	41.16%
0.70	90.57%	25.18%
0.50	83.00%	13.09%

Table 4.18: Performance of TRADES WideResNet distilled onto MobileNetV2 using ARD with different values of α on CIFAR-10, where robust accuracy is with respect to a 20-step PGD attack as in [15].

4.13 Discussion

We find that knowledge distillation allows a student network to absorb a large amount of a teacher network's robustness to adversarial attacks, even when the student is only trained on clean data. However, in some cases, a distilled student model is still far less robust than the teacher. To improve student robustness, we introduce Adversarially Robust Distillation (ARD). In our experiments, student models trained using our method outperform similar networks trained using adversarial training in robust and often natural accuracy. Our models exceed state-of-the-art performance on CIFAR-10 and CIFAR-100 benchmarks. Furthermore, we develop a free adversarial training variant of ARD and demonstrate appreciably accelerated performance.

Recent work on distillation has produced significant improvements over vanilla knowledge distillation [22]. We believe that Knowledge Distillation with Feature Maps could improve both natural and robust accuracy of student networks. Adaptive data augmentation like AutoAugment [31] may also improve performance of both normal knowledge distillation for robust teachers and ARD. Finally, with the recent publication of fast adversarial training methods [144, 175], we hope to further accelerate ARD.

Chapter 5: Robust Few-Shot Learning: A Meta-Learning Approach

Previous work on adversarially robust neural networks requires large training sets and computationally expensive training procedures. On the other hand, few-shot learning methods are highly vulnerable to adversarial examples. The goal of our work is to produce networks which both perform well at few-shot tasks and are simultaneously robust to adversarial examples. We adapt adversarial training for meta-learning, we adapt robust architectural features to small networks for meta-learning, we test pre-processing defenses as an alternative to adversarial training for meta-learning, and we investigate the advantages of robust meta-learning over robust transfer-learning for few-shot tasks. This work provides a thorough analysis of adversarially robust methods in the context of meta-learning, and we lay the foundation for future work on defenses for few-shot tasks. This work was conducted with Liam Fowl and Tom Goldstein [53]. My contribution was conceiving of the central ideas of the project, implementing the algorithms, conducting most experiments from all sections, and writing most of the paper.

5.1 Introduction

For safety-critical applications like facial recognition, traffic sign detection, and copyright control, adversarial attacks pose an actionable threat [45, 132, 181]. Conventional adversarial training and pre-processing defenses aim to produce networks that resist attack [115, 134, 179], but such defenses rely heavily on the availability of large training datasets. In applications that require *few-shot learning*, such as face recognition from few images, recognition of a video source from a single clip, or recognition of a new object from few example photos, the conventional robust training pipeline breaks down.

When data is scarce or new classes arise frequently, neural networks must adapt quickly [43, 83, 128, 161]. In these situations, *meta-learning* methods achieve few-shot learning by creating networks that learn quickly from little data and with computationally cheap fine-tuning. While state-of-the-art meta-learning methods perform well on benchmark few-shot classification tasks, these naturally trained neural networks are highly vulnerable to adversarial examples. In fact, we will see below that even robust classifiers, when adapted to a new task, fail to resist attacks unless appropriate measures are taken.

We study robust few-shot image classification by meta-learning. We begin by exploring several obvious defenses for few shot learning: adversarial training, robust architectural features, and pre-processing defenses, and find that all three provide relatively weak security in the few-shot setting. Specifically, feature denoising layers, architectural features that achieve state-of-the-art adversarial robustness on

ImageNet, are not effective on the lightweight architectures used by meta-learning algorithms, and pre-processing defenses, such as DefenseGAN and image superresolution, dramatically decrease natural accuracy without achieving robustness.

We propose a new approach, called *adversarial querying*, in which the network is exposed to adversarial attacks during the query step of meta-learning. This algorithm-agnostic method produces a feature extractor that is robust, even without adversarial training during fine-tuning. In the few-shot setting, we show that adversarial querying out-performs standard defenses by a wide margin in terms of both clean accuracy and robustness.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
AT transfer learning (R2-D2 backbone)	39.13%	25.33%
ADML	47.75%	18.49%
Naturally Trained R2-D2	72.59%	0.00%
AQ R2-D2 (ours)	57.87%	**31.52%**

Table 5.1: The R2-D2 meta-learning method, adversarially trained transfer learning (ADML), and our adversarially queried (AQ) R2-D2 classifier on 5-shot Mini-ImageNet. The transfer learning model was trained on all training data (except the hold-out classes) simultaneously, and then fine-tuned on few-shot classes. All R2-D2 models are fine-tuned with a ridge regression head as in [12], and we re-implement ADML from [169]. Natural accuracy is denoted $\mathcal{A}_{nat}$, and robust accuracy, $\mathcal{A}_{adv}$, is computed with respect to a 20-step PGD attack as in [115] with $\epsilon = \frac{8}{255}$.

5.2 Related Work

5.2.1 Learning with less data

Before the emergence of meta-learning, a number of approaches existed to cope with few-shot data. One simple approach is *transfer learning*, in which pre-trained feature extractors are created using large datasets, and then fine-tuned on new tasks using less data [11]. Metric learning methods avoid overfitting to the small number of training examples in new classes by instead performing classification using nearest-neighbors in feature space with a feature extractor that is trained on a large corpus of data and not re-trained when classes are added [49, 110, 151]. Metric learning methods are computationally efficient when adding many low-shot classes, since the feature extractor network is not re-trained.

Meta-learning algorithms create a "base" model that quickly adapts to new tasks by fine-tuning. This model is created using a set of training tasks $\{\mathcal{T}_i\}$ that can be sampled from a task distribution. Each task comes with *support* data, $\mathcal{T}_i^s$, and *query* data, $\mathcal{T}_i^q$. In practice, each task is taken to be a classification problem involving only a small subset of classes in a large many-class dataset. The number of examples per class in the support set is called the *shot*, so that fine-tuning on five support examples per class is 5-shot learning.

An iteration of training begins by sampling tasks $\{\mathcal{T}_i\}$ from the task distribution. The base model is fine-tuned on the support data for the sampled tasks, and then used to make predictions on the query data. Then, the base model parameters are updated to improve the accuracy of the resulting fine-tuned model. This requires backpropagation through the fine-tuning steps. See Algorithm 3 for a formal treatment.

Note that the fine-tuned parameters, $\theta_i = A(\theta, \mathcal{T}_i^s)$, in the above algorithm, are a function of the base model's parameters so that the gradient computation in the outer loop may backpropagate through A. For validation after training, the base

Algorithm 3: The meta-learning framework

```
Require: Base model, $F_\theta$, fine-tuning algorithm, $A$, learning rate, $\gamma$, and
  distribution over tasks, $p(\mathcal{T})$.
Initialize $\theta$, the weights of $F$;
while not done do
    Sample batch of tasks, $\{\mathcal{T}_i\}_{i=1}^n$, where $\mathcal{T}_i \sim p(\mathcal{T})$ and $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$.
    for $i = 1, ..., n$ do
        Fine-tune model on $\mathcal{T}_i$ (inner loop). New network parameters are
          written $\theta_i = A(\theta, \mathcal{T}_i^s)$.
        Compute gradient $g_i = \nabla_\theta \mathcal{L}(F_{\theta_i}, \mathcal{T}_i^q)$.
    Update base model parameters (outer loop): $\theta \leftarrow \theta - \frac{\gamma}{n} \sum_i g_i$
```

model is fine-tuned on the support set of hold-out tasks, and accuracy on the query set is reported. In this work, we report performance on OmniGlot, Mini-ImageNet, and CIFAR-FS [12, 95, 162].

We focus on four meta-learning algorithms: MAML, R2-D2, MetaOptNet, and ProtoNet. [12, 47, 103, 151]. During fine-tuning, MAML uses SGD to update all parameters, minimizing cross-entropy loss. Since unrolling SGD steps into a deep computation graph is expensive, a first-order variants ignore second-order derivatives. We use the original MAML formulation. R2-D2 and MetaOptNet, on the other hand, only update the final linear layer during fine-tuning, leaving the "backbone network" that extracts these features frozen at test time. R2-D2 replaces SGD with a closed-form differentiable solver for regularized ridge regression, while MetaOptNet achieves its best performance when replacing SGD with a solver for SVM. Because the objective of these linear problems is convex, differentiable convex optimizers can be efficiently deployed to find optima, and differentiate these optima with respect to the backbone features at train time. ProtoNet takes an approach inspired by metric learning. It constructs class prototypes as the centroids in feature space for each task. These centroids are then used to classify the query set in the outer loop of training. Because each class prototype is a simple geometric average of feature representations, it is easy to differentiate through the fine-tuning step.

5.2.2 Robust learning with less data

Several authors have tried to learn robust models in the data scarce regime. The authors of [146] study robustness properties of transfer learning. They find that retraining earlier layers of the network during fine-tuning impairs the robustness of the network, while only retraining later layers can largely preserve robustness. ADML is the first attempt at achieving robustness through meta-learning. ADML is a MAML variant, specifically designed for robustness, which employs adversarial training [169]. However, this method for robustness is only compatible with MAML, an outdated meta-learning algorithm. Moreover, ADML is computationally expensive, and the authors only test their method against a weak attacker. We implement ADML and test it against a strong attacker. We show that our methods achieve both higher robustness and natural accuracy.

Sample results comparing baseline robust learning methods are shown in Table 5.1, which shows that clean meta-learning and a direct application of adversarial training to meta-learning (the ADML method) achieve low levels of robustness. While simple robust transfer learning achieves more robustness, the adversarial querying procedure does significantly better in terms of both clean and robust accuracy.

5.3 Evaluating the robustness of existing few-shot methods

In this section, we benchmark existing methods for robust learning with scarce data in terms of both natural and robust accuracy. Following standard practices, we assess the robustness of models by attacking them with ℓ_∞-bounded perturbations. We craft image perturbations using the projected gradient descent attack (PGD) since it has proven to be one of the most effective algorithms both for attacking as well as for adversarial training [115]. This attack is a more powerful version of the one-step attack used in ADML [169]. A detailed description of the PGD attack can be found in Algorithm 4. We consider perturbations of ℓ_∞ radius of $\frac{8}{255}$, and a step size of $\frac{2}{255}$ as described by [115].

Adversarial training is the industry standard for creating robust models that maintain good clean-label performance [115]. This method involves replacing clean examples with adversarial examples during the training routine. A simple way to harden models to attack is *adversarial training* which we define formally.

Definition 5.1 (Adversarial Training)**.** Adversarial training involves solving the minimax optimization problem,

$$\min_{\theta} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}} \left[\max_{\|\delta\|_p<\epsilon} \mathcal{L}_\theta(\mathbf{x}+\delta, y) \right], \tag{5.1}$$

where $\mathcal{L}_\theta(\mathbf{x}+\delta, y)$ is the loss function of a network with parameters θ, $\mathbf{x}$ is an input image with label y, and δ is an adversarial perturbation.

Adversarial training finds network parameters that keep the loss low (and class labels correct) even when adversarial perturbations are added.

```
Algorithm 4: PGD Attack
Require: network, $F_\theta$, input data, $(\mathbf{x}, y)$, perturbation, $\delta$, number of steps,
  $n$, step size, $\gamma$, and attack bound, $\epsilon$.
Initialize $\delta \in \mathcal{B}_\epsilon(\mathbf{x})$ randomly;
for $i = 1, ..., n$ do
    Compute $g = \text{sign}\left(\nabla_{\mathbf{x}} \mathcal{L}_\theta\left(\mathbf{x} + \delta, y\right)\right)$.
    Update $\delta = \delta + \gamma g$.
    If $\|\delta\|_p > \epsilon$, then project $\delta$ onto the surface of $\mathcal{B}_\epsilon(\mathbf{x})$.
return perturbed image $\mathbf{x} + \delta$
```

5.3.1 Naturally trained meta-learners are not robust

Similarly to classically trained classifiers, we expect that few-shot learners are highly vulnerable to attack when adversarial defenses are not employed. We test prominent meta-learning algorithms against a 20-step PGD attack as in [115]. Table 5.2 contains 5-shot natural and robust accuracy on the Mini-ImageNet and CIFAR-FS datasets [12, 162].

Model	$\mathcal{A}_{nat}$ MI	$\mathcal{A}_{adv}$ MI	$\mathcal{A}_{nat}$ CIFAR-FS	$\mathcal{A}_{adv}$ CIFAR-FS
ProtoNet	70.23%	0.00%	79.66%	0.00%
R2-D2	73.02%	0.00%	82.81%	0.00%
MetaOptNet	78.12%	0.00%	84.11%	0.00%

Table 5.2: 5-shot MiniImageNet (MI) and CIFAR-FS results comparing naturally trained meta-learners. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack.

We find that these algorithms are completely unable to resist the attack. Interestingly, MetaOptNet uses SVM for fine-tuning, which is endowed with a wide margins property. The failure of even SVM to express robustness during testing suggests that using robust fine-tuning methods on naturally trained meta-learners is insufficient for robust performance. To further examine this, we consider MAML, which updates the entire network during fine-tuning. We use a naturally trained MAML model and perform adversarial training during fine-tuning (see Table 5.3). Adversarial training is performed with 7-PGD as in [115]. If adversarial fine-tuning yielded robust classification, then we could avoid expensive adversarial training variants during meta-learning.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$	$\mathcal{A}_{nat(adv-tuned)}$	$\mathcal{A}_{adv(adv-tuned)}$
1-shot Mini-ImageNet	45.04%	0.03%	33.18%	0.20%
5-shot Mini-ImageNet	60.25%	0.03%	32.45%	1.55%
1-shot Omniglot	91.50%	68.46%	91.60%	74.66%
5-shot Omniglot	97.12%	82.28%	97.71%	87.94%
5-shot Omniglot AQ	97.27%	**95.85%**	97.51%	**96.14%**

Table 5.3: MAML models on Mini-ImageNet and Omniglot datasets. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack. $\mathcal{A}_{nat(adv-tuned)}$ and $\mathcal{A}_{adv(adv-tuned)}$ are natural and robust test accuracy with 7-PGD training during fine-tuning. The bottom row is an adversarially queried model for comparison.

While clean trained MAML models with adversarial fine-tuning are slightly more robust than their naturally fine-tuned counterparts, they achieve almost no robustness on Mini-ImageNet even with adversarial fine-tuning. Omniglot is an easier dataset for robustness, so we include an adversarially queried (AQ) MAML model for comparison. The adversarially queried model achieves far superior robustness. We conclude from these experiments that naturally trained meta-learners are vulnerable to adversarial examples, and an analysis of robust techniques for few-shot learning is in order.

5.3.2 Transfer learning from adversarially trained models is less robust than robust meta-learning

We have observed that few-shot learning methods with a non-robust feature extractor break under attack. But what if we use a robust feature extractor? In the following section, we consider both transfer learning and meta-learning with a robust feature extractor.

In order to compare transfer learning and meta-learning, we train the backbone networks from meta-learning algorithms on all training data simultaneously in the fashion of standard adversarial training using 7-PGD (not meta-learning). We then fine-tune using the head from a meta-learning algorithm on top of the transferred feature extractor. We compare the performance of these feature extractors to that of those trained using adversarially queried meta-learning algorithms with the same backbones and heads. This experiment provides a direct comparison of feature extractors produced by transfer learning and robust meta-learning (see Table 5.4). Meta-learning exhibits far superior robustness than transfer learning on all algorithms we test.

Model	$\mathcal{A}_{nat}$ Transfer	$\mathcal{A}_{adv}$ Transfer	$\mathcal{A}_{nat}$ Meta	$\mathcal{A}_{adv}$ Meta
MAML	32.79%	18.03%	**33.45%**	**23.07%**
ProtoNet	31.14%	22.31%	**52.04%**	**27.99%**
R2-D2	39.13%	25.33%	**57.87%**	**31.52%**
MetaOptNet	50.23%	22.45%	**60.71%**	**28.08%**

Table 5.4: Adversarially trained transfer learning and adversarially queried meta-learning on 5-shot Mini-ImageNet. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack.

5.4 Adversarial Querying: a robust meta-learning technique

We now adapt adversarial training to the meta-learning paradigm by introducing the query data, but not support data, to adversarial attack (see Algorithm 5). This approach yields fast performance during deployment, as adversarial training (which is roughly 10X slower than standard training) is not required to adapt to a new task. Adversarial querying is algorithm agnostic. We test this method on the MAML, ProtoNet, R2-D2, and MetaOptNet algorithms on the Mini-ImageNet and CIFAR-FS datasets (see Table 5.5).

Algorithm 5: Adversarial Querying

```
Require: Base model, $F_\theta$, fine-tuning algorithm, $A$, learning rate, $\gamma$, and
  distribution over tasks, $p(\mathcal{T})$.
Initialize $\theta$, the weights of $F$;
while not done do
    Sample batch of tasks, $\{\mathcal{T}_i\}_{i=1}^n$, where $\mathcal{T}_i \sim p(\mathcal{T})$ and $\mathcal{T}_i = (\mathcal{T}_i^s, \mathcal{T}_i^q)$.
    for $i = 1, ..., n$ do
        Fine-tune model on $\mathcal{T}_i$. New network parameters are written
          $\theta_i = A(\theta, \mathcal{T}_i^s)$.
        Construct adversarial query data, $\widehat{\mathcal{T}_i^q}$, by maximizing $\mathcal{L}(F_{\theta_i}, \widehat{\mathcal{T}_i^q})$
          constrained to $\|\widehat{\mathbf{x}_j^q} - \mathbf{x}_j^q\|_p < \epsilon$ for query examples, $\mathbf{x}_j^q$, and their
          associated adversaries, $\widehat{\mathbf{x}_j^q}$.
        Compute gradient $g_i = \nabla_\theta \mathcal{L}(F_{\theta_i}, \widehat{\mathcal{T}_i^q})$.
    Update base model parameters: $\theta \leftarrow \theta - \frac{\gamma}{n}\sum_i g_i$
```

In our tests, R2-D2 outperforms MetaOptNet in robust accuracy despite having a less powerful backbone architecture. In Section 5.4.2, we dissect the effects of backbone architecture and classification head on the disparity between R2-D2 and MetaOptNet in robust performance. In Section 5.4.4, we verify that adversarial querying generates networks robust to a wide array of strong attackers.

Model	$\mathcal{A}_{nat}$ MI	$\mathcal{A}_{adv}$ MI	$\mathcal{A}_{nat}$ CIFAR-FS	$\mathcal{A}_{adv}$ CIFAR-FS
ProtoNet AQ	52.04%	27.99%	63.53%	40.11%
R2-D2 AQ	57.87%	**31.52%**	69.25%	**44.80%**
MetaOptNet AQ	60.71%	28.08%	71.07%	43.79%

Table 5.5: Comparison of adversarially queried (AQ) meta-learners on 5-shot Mini-ImageNet (MI) and CIFAR-FS. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack.

Adversarial querying can also be used to construct meta-learning analogues for other variants of adversarial training. We explore this by substituting the cross-entropy loss for the TRADES loss [179]. We refer to this method as meta-TRADES. While meta-TRADES can marginally outperform our initial adversarial querying method in robust accuracy with a careful hyperparameter choice, λ, we find that networks trained with meta-TRADES severely sacrifice natural accuracy (see Table 5.6).

Model	$\mathcal{A}_{nat}$ MI	$\mathcal{A}_{adv}$ MI	$\mathcal{A}_{nat}$ CIFAR-FS	$\mathcal{A}_{adv}$ CIFAR-FS
R2-D2 Adversarial Queried	**57.87%**	31.52%	**69.25%**	44.80%
R2-D2 TRADES ($1/\lambda = 1$)	56.02%	30.96%	66.29%	45.59%
R2-D2 TRADES ($1/\lambda = 3$)	51.51%	**32.30%**	61.41%	**46.54%**
R2-D2 TRADES ($1/\lambda = 6$)	34.29%	22.04%	58.32%	45.89%

Table 5.6: 5-shot Mini-ImagNet (MI) and CIFAR-FS results comparing meta-TRADES to adversarial querying. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack.

5.4.1 For better natural and robust accuracy, only fine-tune the last layer.

High performing meta-learning models, like MetaOptNet and R2-D2, fix their feature extractor and only update their last linear layer during fine-tuning. In the setting of transfer learning, robustness is a feature of early convolutional layers, and re-training these early layers leads to a significant drop in robust test accuracy [146]. We verify that re-training only the last layer leads to improved natural and robust accuracy in adversarially queried meta-learners by training a MAML model but only updating the final layer during fine-tuning including during the inner loop of meta-learning. We find that the model trained by only fine-tuning the last layer decisively outperforms the traditional MAML algorithm (AQ) in both natural and robust accuracy (see Table 5.7).

Layers updated	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$	$\mathcal{A}_{nat(adv-tuned)}$	$\mathcal{A}_{adv(adv-tuned)}$
All layers	33.45%	23.07%	33.03%	23.29%
FC Only	40.06%	25.15%	39.94%	25.32%

Table 5.7: Adversarially queried MAML compared with a MAML variant with only the last layer re-trained during fine-tuning on 5-shot Mini-ImageNet. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack. $\mathcal{A}_{nat(adv-tuned)}$ and $\mathcal{A}_{adv(adv-tuned)}$ are natural and robust test accuracy respectively with 7-PGD training during fine-tuning. Layers are fine-tuned for 10 steps with a learning rate of 0.01.

5.4.2 The R2-D2 head, not embedding, is responsible for superior robust performance.

The naturally trained MetaOptNet algorithm outperforms R2-D2 in natural accuracy, but previous research has found that performance discrepancies between meta-learning algorithms might be an artifact of different backbone networks [23]. On natural meta-learning, we confirm that MetaOptNet with the R2-D2 backbone performs similarly to R2-D2 (see Table 5.8). In our adversarial querying experiments, we saw that MetaOptNet was less robust than R2-D2. This discrepancy remains when we train MetaOptNet with the R2-D2 backbone (see Table 5.9). We conclude that MetaOptNet's backbone is not responsible for its inferior robustness. These experiments suggest that ridge regression may be a more effective fine-tuning technique than SVM for robust performance. ProtoNet with R2-D2 backbone also performs worse than the other two adversarially queried models with the same backbone architecture.

Model	1-shot MI	5-shot MI	1-shot CIFAR	5-shot CIFAR
R2-D2	55.22%	73.02%	68.36%	82.81%
MetaOptNet	**60.65%**	**78.12%**	**70.99%**	**84.11%**
MetaOptNet (R2-D2 backbone)	55.78%	73.15%	68.37%	82.71%

Table 5.8: Natural test accuracy of naturally trained R2-D2, MetaOptNet, and the MetaOptNet head with R2-D2 backbone on the Mini-ImageNet (MI) and CIFAR-FS (CIFAR) datasets.

Model	1-shot MI	5-shot MI	1-shot CIFAR	5-shot CIFAR
R2-D2	**20.59%**	**31.52%**	**32.33%**	**44.80%**
MetaOptNet	18.37%	28.08%	30.74%	43.79%
MetaOptNet (R2-D2 backbone)	18.81%	24.68%	29.57%	41.90%
ProtoNet (R2-D2 backbone)	18.24%	28.39%	26.48%	40.59%

Table 5.9: Robust test accuracy of adversarially queried R2-D2, MetaOptNet, and the MetaOptNet and heads with R2-D2 backbone on Mini-ImageNet (MI) CIFAR-FS (CIFAR) datasets. Robust accuracy is computed with respect to a 20-step PGD attack.

5.4.3 Enhancing robustness with robust architectural features

In addition to adversarial training, architectural features have been used to enhance robustness [166]. Feature denoising blocks pair classical denoising operations with learned 1×1 convolutions to reduce the feature noise in feature maps at various stages of a network, and thus reduce the success of adversarial attacks. Massive architectures with these blocks have achieved state-of-the-art robustness against targeted adversarial attacks on ImageNet. However, when deployed on small networks for meta-learning, we find that denoising blocks do not improve robustness. We deploy denoising blocks identical to those in [166] after various layers of the R2-D2 network. The best results for the denoising experiments are achieved by adding a denoising block after the fourth layer in the R2-D2 embedding network (See Table 5.10).

5.4.4 Resistance to other attacks

We test our method by exposing our adversarially queried R2-D2 model to a variety of powerful adversarial attacks. We implement the momentum iterated fast

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
R2-D2	73.02%	0.00%
R2-D2 AQ	57.87%	**31.52%**
R2-D2 AQ Denoising	57.68%	31.14%

Table 5.10: 5-shot MiniImageNet results for our highest performing R2-D2 with feature denoising blocks. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack.

gradient sign method (MI-FGSM), DeepFool, and 20-step PGD with 20 random restarts [38, 115, 117]. Our adversarially queried model indeed is nearly as robust against the strongest ℓ_∞ bounded attacker as it is against the 20-step PGD attack with a single random start we tested against previously. Note that DeepFool is not ℓ_∞ bounded and thus the perturbed images are outside of the robustness radius enforced during adversarial querying.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{DF}$	$\mathcal{A}_{MI}$	$\mathcal{A}_{20-PGD}$
R2-D2	73.02%	7.91%	0.01%	0.0%
R2-D2 AQ	57.87%	**14.45%**	**31.87%**	**30.31%**
R2-D2 AT (Transfer Learning)	39.13%	0.42%	24.01%	19.75%

Table 5.11: 5-shot MiniImageNet results against DeepFool (DF) (2 iteration) ℓ_∞ attack, MI-FGSM (MI) ($\epsilon = 8/255$) attack, and PGD attack with 20 random restarts (20-PGD). We compare R2-D2 trained with adversarial-querying (AQ) to the transfer learning R2-D2 as in section 5.4.

5.5 Pre-processing defenses as an alternative to adversarial training

Recent works have proposed pre-proccessing defenses for sanitizing adversarial examples before feeding them into a naturally trained classifier. If successful, these methods would avoid the expensive adversarial querying procedure during training. While this approach has found success in the mainstream literature, we find that it is ineffective in the few-shot regime.

In DefenseGAN, a GAN trained on natural images is used to sanitize an adversarial example by replacing (possible corrupted) test images with the nearest image in the output range of the GAN [134]. Unfortunately, GANs are not expressive enough to preserve the integrity of testing images on complex datasets involving high-res natural images, and recent attacks have critically compromised the performance of this defense [7, 79]. We found the expressiveness of the generator architecture used in the original DefenseGAN setup to be insufficient for even CIFAR-FS, so we substitute a stronger ProGAN generator to model the CIFAR-100 classes [85].

The supperesolution defense first denoises data with sparse wavelet filters and then performs superresolution [120]. This defense is also motivated by the principle of projecting adversarial examples onto the natural image manifold. We test the superresolution defense using the same wavelet filtering and superresolution network (SRResNet) used by [120] and first introduced by [99]. Like with the generator for DefenseGAN, we train the SRResNet on the entire CIFAR-100 dataset before applying the superresolution defense.

We find that these methods are not well suited to the few-shot domain, in which the generative model or superresolution network may not be able to train on the little data available. Morever, even after training the generator on all CIFAR-100 classes, we find that DefenseGAN with a naturally trained R2-D2 meta-learner performs significantly worse in both natural and robust accuracy than an adversarially queried meta-learner of the same architecture. Similarly, the superresolution defense achieves little robustness. The results of these experiments can be found in Table 5.12.

Model	$\mathcal{A}_{nat}$	$\mathcal{A}_{adv}$
R2-D2	83.30%	0.00%
R2-D2 AQ	69.25%	**44.80%**
R2-D2 with SR defense	35.15%	23.00%
R2-D2 with DefenseGAN	35.15%	28.05%

Table 5.12: 5-shot CIFAR-FS results comparing the superresolution defense (SR defense) and DefenseGAN. $\mathcal{A}_{nat}$ and $\mathcal{A}_{adv}$ are natural and robust test accuracy respectively, where robust accuracy is computed with respect to a 20-step PGD attack. Both methods perform worse than their adversarially queried counterpart.

5.6 Discussion & Conclusion

Naturally trained networks for few-shot learning are vulnerable to adversarial attacks, and existing robust transfer learning methods do not perform well on few-shot tasks. Naturally trained networks suffer from adversarial vulnerability even when adversarially trained during fine-tuning. We thus identify the need for an investigation into robust few-shot methods. We particularly study robustness in the context of meta-learning. We develop an algorithm-agnostic method, called adversarial querying, for hardening meta-learning models. We find that meta-learning models are most robust when the feature extractor is fixed, and only the last layer is retrained during the fine tuning stage. We further identify that choice of classification head matters for robustness. We hope that this work serves as a starting point for developing new adversarially robust methods for few-shot applications.

Chapter 6: Some Interesting Properties of Neural Networks

We empirically evaluate common assumptions about neural networks that are widely held by practitioners and theorists alike. In this work, we: (1) prove the widespread existence of suboptimal local minima in the loss landscape of neural networks, and we use our theory to find examples; (2) show that small-norm parameters are not optimal for generalization; (3) demonstrate that ResNets do not conform to wide-network theories, such as the neural tangent kernel, and that the interaction between skip connections and batch normalization plays a role; (4) find that rank does not correlate with generalization or robustness in a practical setting. This work was done in collaboration with Jonas Geiping, Avi Schwarzschild, Michael Moeller, and Tom Goldstein [54]. My role was leading the project, conceiving of and proving the theorem, conceiving of the norm-bias regularizer as well as conducting the related experiments, and doing a large portion of writing.

6.1 Introduction

Modern deep learning methods are descendent from such long-studied fields as statistical learning, optimization, and signal processing, all of which were built on mathematically rigorous foundations. In statistical learning, principled kernel methods have vastly improved the performance of SVMs and PCA [141, 153], and boosting theory has enabled weak learners to generate strong classifiers [140]. Optimizers in deep learning are borrowed from the field of convex optimization , where momentum optimizers [121] and conjugate gradient methods provably solve ill-conditioned problems with high efficiency [72]. Deep learning harnesses foundational tools from these mature parent fields.

Despite its rigorous roots, deep learning has driven a wedge between theory and practice. Recent theoretical work has certainly made impressive strides towards understanding optimization and generalization in neural networks. But doing so has required researchers to make strong assumptions and study restricted model classes.

In this work, we seek to understand whether deep learning theories accurately capture the behaviors and network properties that make realistic deep networks work. Following a line of previous work, such as [154], [174], [9] and [138], we put the assumptions and conclusions of deep learning theory to the test using experiments with both toy networks and realistic ones. We focus on the following important theoretical issues:

- Local minima: Numerous theoretical works argue that all local minima of neural loss functions are globally optimal or that all local minima are nearly optimal. In practice, we find highly suboptimal local minima in realistic neural loss functions, and we discuss reasons why suboptimal local minima exist in the loss surfaces of deep neural networks in general.

- Weight decay and parameter norms: Research inspired by Tikhonov regularization suggests that low-norm minima generalize better, and for many, this is an intuitive justification for simple regularizers like weight decay. Yet for neural networks, it is not at all clear which form of ℓ_2-regularization is optimal. We show this by constructing a simple alternative: biasing solutions toward a non-zero norm still works and can even measurably improve performance for modern architectures.

- Neural tangent kernels and the wide-network limit: We investigate theoretical results concerning neural tangent kernels of realistic architectures. While stochastic sampling of the tangent kernels suggests that theoretical results on tangent kernels of multi-layer networks may apply to some multi-layer networks and basic convolutional architectures, the predictions from theory do not hold for practical networks, and the trend even reverses for ResNet architectures. We show that the combination of skip connections and batch normalization is critical for this trend in ResNets.

- Rank: Generalization theory has provided guarantees for the performance of low-rank networks. However, we find that regularization which encourages high-rank weight matrices often outperforms that which promotes low-rank matrices. This indicates that low-rank structure is not a significant force behind generalization in practical networks. We further investigate the adversarial robustness of low-rank networks, which are thought to be more resilient to attack, and we find empirically that their robustness is often lower than the baseline or even a purposefully constructed high-rank network.

6.2 Local minima in loss landscapes: Do suboptimal minima exist?

It is generally accepted that "in practice, poor local minima are rarely a problem with large networks." [98]. However, exact theoretical guarantees for this statement are elusive. Various theoretical studies of local minima have investigated spin-glass models [26], deep linear models [86, 97], parallel subnetworks [60], and dense fully connected models [123] and have shown that either all local minima are global or all have a small optimality gap. The apparent scarcity of poor local minima has lead practitioners to develop the intuition that bad local minima ("bad" meaning high loss value and suboptimal training performance) are practically non-existent.

To further muddy the waters, some theoretical works prove the *existence* of local minima. Such results exist for simple fully connected architectures [154], single-layer networks [107, 170], and two-layer ReLU networks [133]. For example, [171] show that local minima exist in single-layer networks with univariate output and unique datapoints. The crucial idea here is that all neurons are activated for all datapoints at the suboptimal local minima. Unfortunately, these existing analyses of neural loss landscapes require strong assumptions (e.g. random training data, linear activation functions, fully connected layers, or extremely wide network widths) — so strong, in fact, that it is reasonable to question whether these results have any bearing on practical neural networks or describe the underlying cause of good optimization performance in real-world settings.

In this section, we investigate the existence of suboptimal local minima from a theoretical perspective and an empirical one. If suboptimal local minima exist, they are certainly hard to find by standard methods (otherwise training would not work). Thus, we present simple theoretical results that inform us on how to construct non-trivial suboptimal local minima, concretely generalizing previous constructions, such as those by [171]. Using experimental methods inspired by theory, we easily find suboptimal local minima in the loss landscapes of a range of classifiers.

Trivial local minima are easy to find in ReLU networks – consider the case where bias values are sufficiently low so that the ReLUs are "dead" (i.e. inputs to ReLUs are strictly negative). Such a point is trivially a local minimum. Below, we make a more subtle observation that multilayer perceptrons (MLPs) must have non-trivial local minima, provided there exists a linear classifer that performs worse than the neural network (an assumption that holds for virtually any standard benchmark problem). Specifically, we show that MLP loss functions contain local minima where they behave identically to a linear classifier on the same data.

We now define a family of low-rank linear functions which represent an MLP. Let "rank-s affine function" denote an operator of the form $G(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$ with $\text{rank}(A) = s$.

Definition 6.1. Consider a family of functions, $\{F_\phi : \mathbb{R}^m \to \mathbb{R}^n\}_{\phi \in \mathbb{R}^P}$ parameterized by ϕ. We say this family has *rank-s affine expression* if for all rank-s affine functions $G : \mathbb{R}^m \to \mathbb{R}^n$ and finite subsets $\Omega \subset \mathbb{R}^m$, there exists ϕ with $F_\phi(\mathbf{x}) = G(\mathbf{x})$, $\forall \mathbf{x} \in \Omega$. If $s = \min(n, m)$ we say that this family has *full affine expression.*

We investigate a family of L-layer MLPs with ReLU activation functions, $\{F_\phi : \mathbb{R}^m \to \mathbb{R}^n\}_{\phi \in \Phi}$, and parameter vectors ϕ, i.e., $\phi = (A_1, \mathbf{b}_1, A_2, \mathbf{b}_2, \ldots, A_L, \mathbf{b}_L)$,

$F_\phi(\mathbf{x}) = H_L(f(H_{L-1}...f(H_1(\mathbf{x}))))$, where f denotes the ReLU activation function and $H_i(\mathbf{z}) = A_i\mathbf{z} + \mathbf{b}_i$. Let $A_i \in \mathbb{R}^{n_i \times n_{i-1}}$, $\mathbf{b}_i \in \mathbb{R}^{n_i}$ with $n_0 = m$ and $n_L = n$.

Lemma 6.1. *Consider a family of L-layer multilayer perceptrons with ReLU activations $\{F_\phi : \mathbb{R}^m \to \mathbb{R}^n\}$ and let $s = \min_i n_i$ be the minimum layer width. Then this family has rank-s affine expression.*

Proof. Let G be a rank-s affine function, and $\Omega \subset \mathbb{R}^m$ be a finite set. Let $G(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$ with $A = U\Sigma V$ being the singular value decomposition of A with $U \in \mathbb{R}^{n\times s}$ and $V \in \mathbb{R}^{s\times m}$.

We define

$$A_1 = \begin{bmatrix} \Sigma V \\ \\ \mathbf{0} \end{bmatrix}$$

where $\mathbf{0}$ is a (possibly void) $(n_1 - s) \times m$ matrix of all zeros, and $b_1 = c\mathbf{1}$ for $c = \max_{\mathbf{x}_i \in \Omega, 1\leq j \leq n_1} |(A_1\mathbf{x}_i)_j| + 1$ and $\mathbf{1} \in \mathbb{R}^{n_1}$ being a vector of all ones. We further choose $A_l \in \mathbb{R}^{n_l \times n_{l-1}}$ to have an $s \times s$ identity matrix in the upper left, and fill all other entries with zeros. This choice is possible since $n_l \geq s$ for all l. We define $\mathbf{b}_l = \begin{bmatrix} \mathbf{0} & c\,\mathbf{1} \end{bmatrix}^T \in \mathbb{R}^{n_l}$ where $\mathbf{0} \in \mathbb{R}^{1\times s}$ is a vector of all zeros and $\mathbf{1} \in \mathbb{R}^{1\times(n_l - s)}$ is a (possibly void) vector of all ones.

Finally, we choose $A_L = \begin{bmatrix} U & \mathbf{0} \end{bmatrix}$, where now $\mathbf{0}$ is a (possibly void) $n \times (n_{L-1} - s)$ matrix of all zeros, and $\mathbf{b}_L = -cA_L\mathbf{1} + \mathbf{b}$ for $\mathbf{1} \in \mathbb{R}^{n_{L-1}}$ being a vector of all ones.

Then one readily checks that $F_\phi(\mathbf{x}) = G(\mathbf{x})$ holds for all $x \in \Omega$. Note that all entries of all activations are greater or equal to $c > 0$, such that no ReLU ever maps an entry to zero. □

The ability of MLPs to represent linear networks allows us to derive a theorem which implies that arbitrarily deep MLPs have local minima at which the performance of the underlying model on the training data is equal to that of a (potentially low-rank) linear model. In other words, neural networks inherit the local minima of elementary linear models.

Theorem 6.2. *Consider a training set,* $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$*, a family* $\{F_\phi\}$ *of MLPs with* $s = \min_i n_i$ *being the smallest width. Consider the training of a rank-s linear classifier* $G_{A,\mathbf{b}}$*, i.e.,*

$$\min_{A,\mathbf{b}} \mathcal{L}(G_{A,\mathbf{b}}; \{(\mathbf{x}_i, y_i)\}_{i=1}^N), \qquad \textit{subject to rank}(A) \leq s, \tag{6.1}$$

for any continuous loss function $\mathcal{L}$*. Then for each local minimum,* $(A', \mathbf{b}')$*, of the above training problem, there exists a local minimum,* ϕ'*, of* $\mathcal{L}(F_\phi; \{(\mathbf{x}_i, y_i)\}_{i=1}^N)$ *with the property that* $F_{\phi'}(\mathbf{x}_i) = G_{A',\mathbf{b}'}(\mathbf{x}_i)$ *for* $i = 1, 2, ..., N$*.*

Proof. Based on the definition of a local minimium, there exists an open ball D around $(A', \mathbf{b}')$ such that

$$\mathcal{L}(G_{A',\mathbf{b}'}; \{(\mathbf{x}_i, y_i)\}_{i=1}^N) \leq \mathcal{L}(G_{A,\mathbf{b}}; \{(\mathbf{x}_i, y_i)\}_{i=1}^N) \quad \forall (A, \mathbf{b}) \in D \text{ with rank}(A) \leq s. \tag{6.2}$$

First, we use the same construction as in the proof of Lemma 6.1 to find a function $F_{\phi'}$ with $F_{\phi'}(\mathbf{x}_i) = G_{A',\mathbf{b}'}(\mathbf{x}_i)$ for all training example $\mathbf{x}_i$. Because the mapping $\phi \mapsto F_\phi(\mathbf{x}_i)$ is continuous (not only for the entire network F but also for all subnetworks), and because all activations of $F_{\phi'}$ are greater or equal to $c > 0$, there exists an open ball $B(\phi', \delta_1)$ around ϕ' such that the activations of F_ϕ remain positive for all $\mathbf{x}_i$ and all $\phi \in B(\phi', \delta_1)$.

Consequently, the restriction of F_ϕ to the training set remains affine linear for $\phi \in B(\phi', \delta_1)$. In other words, for any $\phi \in B(\phi', \delta_1)$ we can write

$$F_\phi(\mathbf{x}_i) = A(\phi)\mathbf{x}_i + \mathbf{b}(\phi) \qquad \forall \mathbf{x}_i,$$

by defining $A(\phi) = A_L A_{L-1} \dots A_1$ and $\mathbf{b}(\phi) = \sum_{l=1}^{L} A_L A_{L-1} \dots A_{l+1} \mathbf{b}_l$. Note that due to $s = \min_i n_i$, the resulting $A(\phi)$ satisfies $\operatorname{rank}(A(\phi)) \leq s$.

After restricting ϕ to an open ball $B(\phi', \delta_2)$, for $\delta_2 \leq \delta_1$ sufficiently small, the above $(A(\phi), \mathbf{b}(\phi))$ satisfy $(A(\phi), \mathbf{b}(\phi)) \in D$ for all $\phi \in B(\phi', \delta_2)$. On this set, we, however, already know that the loss can only be greater or equal to $\mathcal{L}(F_{\phi'}; \{(\mathbf{x}_i, y_i)\}_{i=1}^N)$ due to (6.2). Thus, ϕ' is a local minimum of the underlying loss function. $\square$

The proof of the above theorem constructs a network in which all activations of all training examples are positive, generalizing previous constructions of this type such as [171] to more realistic architectures and settings. Another paper has employed a similar construction concurrently to our own work [64]. We do expect that the general problem in expressivity occurs every time the support of the activations coincides for all training examples, as the latter reduces the deep network to an affine linear function (on the training set), which relates to the discussion in [9]. We test this hypothesis below by initializing deep networks with biases of high variance.

Remark (CNN and more expressive local minima). Note that the above constructions of Lemma 6.1 and Theorem 6.2 are not limited to MLPs and could be extended to convolutional neural networks with suitably restricted linear mappings G_ϕ by using the convolution filters to represent identities and using the bias to avoid any negative activations on the training examples. Moreover, shallower MLPs can similarly be embedded into deeper MLPs recursively by replicating the behavior of each linear layer of the shallow MLP with several layers of the deep MLP. Linear classifiers, or even shallow MLPs, often have higher training loss than more expressive networks. Thus, we can use the idea of Theorem 1 to find various suboptimal local minima in the loss landscapes of neural networks. We confirm this with subsequent experiments.

We find that initializing a network at a point that approximately conforms to Theorem 1 is enough to get trapped in a bad local minimum. We verify this by training a linear classifier on CIFAR-10 with weight decay, (which has a test accuracy of 40.53%, loss of 1.57, and gradient norm of 0.00375 w.r.t to the logistic regression objective). We then initialize a multilayer network as described in Lemma 6.1 to approximate this linear classifier and recompute these statistics on the full network (see Table 6.1). When training with this initialization, the gradient norm drops futher, moving parameters even closer to the linear minimizer. The final training result still yields positive activations for the entire training dataset.

Moreover, any isolated local minimum of a linear network results in many local minima of an MLP $F_{\phi'}$, as the weights ϕ' constructed in the proof of Theorem 6.2 can undergo transformations such as scaling, permutation, or even rotation without changing $F_{\phi'}$ as a function during inference, i.e. $F_{\phi'}(\mathbf{x}) = F_\phi(\mathbf{x})$ for all $\mathbf{x}$ for an

Table 6.1: Local minima for MLPs generated via various initializations. We show loss, euclidean norm of the gradient vector, and minimum eigenvalue of the Hessian before and after training. We use 500 iterations of the power method on a shifted Hessian matrix computed on the full dataset to find the minimum eigenvalue. The experiment in the last row is trained with no momentum (NM).

	At Initialization			After training		
Init. Type	Loss	Grad.	Min. EV	Loss	Grad.	Min. EV
Default	4.5963	0.5752	-1.5549	0.0061	0.0074	0.0007
Lemma 6.1	1.5702	0.0992	0.03125	1.5699	0.0414	0.0156
Bias+20	31.204	343.99	-1.7421	2.3301	0.0090	0.0005
Bias $\in \mathcal{U}(-50, 50)$	51.445	378.36	-430.49	2.3153	0.0048	0.0000
Bias $\in \mathcal{U}(-10, 10)$ NM	12.209	42.454	-47.733	0.2198	0.0564	0.0013

infinite set of parameters ϕ, as soon as F has at least one hidden layer.

While our first experiment initializes a deep MLP at a local minimum it inherited from a linear one to empirically illustrate our findings of Theorem 6.2, Table 6.1 also illustrates that similarly bad local minima are obtained when choosing large biases (third row) and choosing biases with large variance (fourth row) as conjectured above. To significantly reduce the bias, however, and still obtain a sub-par optimum, we need to rerun the experiment with SGD without momentum, as shown in the last row, reflecting common intuition that momentum is helpful to move away from bad local optima.

Remark (Sharpness of sub-optimal local optima). An interesting additional property of minima found using the previously discussed initializations is that they are "sharp". Proponents of the sharp-flat hypothesis for generalization have found that minimizers with poor generalization live in sharp attracting basins with low volume and thus low probability in parameter space [78, 88], although care has to be taken to correctly measure sharpness [35]. Accordingly, we find that the maximum eigenvalue of the Hessian at each suboptimal local minimum is significantly higher than those at near-global minima. For example, the maximum eigenvalue of the initialization by Lemma 1 in Table 6.1 is estimated as $113,598.85$ after training, whereas that of the default initialization is only around 24.01. While our analysis has focused on sub-par local optima in training instead of global minima with sub-par generalization, both the scarcity of local optima during normal training and the favorable generalization properties of neural networks seem to correlate with their sharpness.

In light of our finding that neural networks trained with unconventional initialization reach suboptimal local minima, we conclude that poor local minima can readily be found with a poor choice of hyperparameters. Suboptimal minima are less scarce than previously believed, and neural networks avoid these because good initializations and stochastic optimizers have been fine-tuned over time. Fortunately, promising theoretical directions may explain good optimization performance while remaining compatible with empirical observations. The approach followed by [42] analyzes the loss trajectory of SGD, showing that it avoids bad minima. While this work assumes (unrealistically) large network widths, this theoretical direction is compatible with empirical studies, such as [58], showing that the training trajectory of realistic deep networks does not encounter significant local minima.

6.2.1 Additional comments regarding Theorem 1

Note that our theoretical and experimental results do not contradict theoretical guarantees for deep linear networks [86, 97] which show that all local minima are global. A deep linear network with $s = \min(n, m)$ is equivalent to a linear classifier, and in this case, the local minima constructed by Theorem 6.2 are global. However, this observation shows that Theorem 6.2 characterizes the gap between deep linear and deep nonlinear networks; the global minima predicted by linear network theories are inherited as (usually suboptimal) local minima when ReLU's are added. Thus, linear networks do not accurately describe the distribution of minima in non-linear networks.

6.2.2 Additional results concerning suboptimal local optima

Table 6.2 shows more experiments. As above in the previous experiment, we use gradient descent to train a full ResNet-18 architecture on CIFAR-10 until convergence from different initializations. We find that essentially the same results appear for the deeper architecture, initializing with very high bias leads to highly non-optimal solutions. In this case even solutions that are equally bad as a zero-norm initialization.

Further results on CIFAR-100 are shown in Tables 6.3 and 6.4. These experiments with MLP and ResNet-18 show the same trends as explained above, thus confirming that the results are not specific to the CIFAR-10 dataset.

Table 6.2: Local minima for ResNet-18 and CIFAR-10 generated via initialization and trained by vanilla gradient descent, showing loss, euclidean norm of the gradient vector.

	At Initialization		After training	
Init. Type	Loss	Grad.	Loss	Grad.
Default	2.30312	0.05000	0.00014	0.01410
Zero	2.30258	0.00025	2.30259	0.00013
Bias+20	12.95754	590.12170	2.30658	0.00004
Bias $\in \mathcal{U}(-10, 10)$	12.96790	214.68600	2.30260	0.00123
Bias $\in \mathcal{U}(-50, 50)$	84.67800	1190.23500	2.30260	0.00702

Table 6.3: Local minima for ResNet-18 and CIFAR-100 generated via initialization and trained by vanilla gradient descent, showing loss, euclidean norm of the gradient vector

	At Initialization		After training	
Init. Type	Loss	Grad.	Loss	Grad.
Default	4.60591	0.02346	0.00030	0.00466
Zero	4.60517	0.00019	4.60517	0.00003
Bias+20	34.37053	655.51569	4.60517	0.00015
Bias $\in \mathcal{U}(-100, 100)$	178.74391	2615.72534	4.60517	0.00003

Table 6.4: Local minima for MLP and CIFAR-100 generated via initialization and trained by vanilla gradient descent, showing loss, euclidean norm of the gradient vector.

	At Initialization		After training	
Init. Type	Loss	Grad.	Loss	Grad.
Default	4.60670	0.16154	0.02579	0.01482
Zero	4.60517	0.00019	4.60517	0.00011
Bias+10	15.77286	359.65710	4.60517	0.00079
Bias $\in \mathcal{U}(-5, 5)$	8.69149	63.59983	2.15917	0.09718
Bias $\in \mathcal{U}(-10, 10)$	13.02693	158.78347	2.58368	0.09233

6.3 Weight decay: Are small ℓ_2-norm solutions better?

Classical learning theory advocates regularization for linear models, such as SVM and linear regression. For SVM, ℓ_2 regularization endows linear classifiers with a wide-margin property [29], and recent work on neural networks has shown that minimum norm neural network interpolators benefit from over-parametrization [63] . Following the long history of explicit parameter norm regularization for linear models, weight decay is used for training nearly all high performance neural networks [25, 65, 76, 136].

In combination with weight decay, all of these cutting-edge architectures also employ batch normalization after convolutional layers [80]. With that in mind, [160] shows that the regularizing effect of weight decay is counteracted by batch normalization, which removes the effect of shrinking weight matrices. [176] argue that the synergistic interaction between weight decay and batch norm arises because weight decay plays a large role in regulating the effective learning rate of networks, since scaling down the weights of convolutional layers amplifies the effect of each optimization step, effectively increasing the learning rate. Thus, weight decay increases the effective learning rate as the regularizer drags the parameters closer and closer towards the origin. The authors also suggest that data augmentation and carefully chosen learning rate schedules are more powerful than explicit regularizers like weight decay.

Other work echos this sentiment and claims that weight decay and dropout have little effect on performance, especially when using data augmentation [71]. [75]

further study the relationship between weight decay and batch normalization, and they develop normalization with respect to other norms. [147] instead suggest that minimum norm solutions may not generalize well in the over-parametrized setting.

We find that the difference between performance of standard network architectures with and without weight decay is often statistically significant, even with a high level of data augmentation, for example, horizontal flips and random crops on CIFAR-10 (see Tables 6.5 and 6.6). But is weight decay the most effective form of ℓ_2 regularization? Furthermore, is the positive effect of weight decay because the regularizer promotes small norm solutions? We generalize weight decay by biasing the ℓ_2 norm of the weight vector towards other values using the following regularizer, which we call *norm-bias*:

$$R_\mu(\phi) = \left| \left(\sum_{i=1}^{P} \phi_i^2 \right) - \mu^2 \right| . \tag{6.3}$$

R_0 is equivalent to weight decay, but we find that we can further improve performance by biasing the weights towards higher norms (see Tables 6.5 and 6.6). In our experiments on CIFAR-10 and CIFAR-100, networks are trained using weight decay coefficients from their respective original papers. ResNet-18 and DenseNet are trained with $\mu^2 = 2500$ and norm-bias coefficient 0.005, and MobileNetV2 is trained with $\mu^2 = 5000$ and norm-bias coefficient 0.001. μ is chosen heuristically by first training a model with weight decay, recording the norm of the resulting parameter vector, and setting μ to be slightly higher than that norm in order to avoid norm-bias leading to a lower parameter norm than weight decay. While we find that weight decay improves results over a non-regularized baseline for all three models, we also find that models trained with large norm bias (i.e., large μ) outperform models trained with weight decay.

These results lend weight to the argument that explicit parameter norm regularization is in fact useful for training networks, even deep CNNs with batch normalization and data augmentation. However, the fact that norm-biased networks can outperform networks trained with weight decay suggests that any benefits of weight decay are unlikely to originate from the superiority of small-norm solutions.

To further investigate the effect of weight decay and parameter norm on generalization, we also consider models without batch norm. In this case, weight decay directly penalizes the norm of the linear operators inside a network, since there are no batch norm coefficients to compensate for the effect of shrinking weights. Our goal is to determine whether small-norm solutions are superior in this setting where the norm of the parameter vector is more meaningful.

In our first experiment without batch norm, we experience improved performance training an MLP with *norm-bias* (see Table 6.6). In a state-of-the-art setting, we consider ResNet-20 with Fixup initialization, a ResNet variant that removes batch norm and instead uses a sophisticated initialization that solves the exploding gradient problem [177]. We observe that weight decay substantially improves training over SGD with no explicit regularization — in fact, ResNets with this initialization scheme train quite poorly without explicit regularization and data normalization. Still, we find that *norm-bias* with $\mu^2 = 1000$ and norm-bias coefficient 0.0005 achieves better results than weight decay (see Table 6.6). This once again refutes the theory that small-norm parameters generalize better and brings into doubt any relationship between classical Tikhonov regularization and weight decay in neural networks. See Section 6.3.1 for a discussion concerning the final parameter norms of Fixup networks as well as additional experiments on CIFAR-100, a harder image classification dataset.

Table 6.5: ResNet-18, DenseNet-40, and MobileNetV2 models trained on non-normalized CIFAR-10 data with various regularizers. Numerical entries are given by $\overline{m}(\pm s)$, where $\overline{m}$ is the average accuracy over 10 runs, and s represents standard error.

Model	No weight decay (%)	Weight decay (%)	Norm-bias (%)
ResNet	93.46 (±0.05)	94.06 (±0.07)	**94.86** (±0.05)
DenseNet	89.26 (±0.08)	92.27 (±0.06)	**92.49** (±0.06)
MobileNetV2	92.88 (±0.06)	92.88 (±0.09)	**93.50** (±0.09)

Table 6.6: ResNet-18, DenseNet-40, MobileNetV2, ResNet-20 with Fixup initialization, and a 4-layer multi-layer perceptron (MLP) trained on normalized CIFAR-10 data with various regularizers. Numerical entries are given by $\overline{m}(\pm s)$, where $\overline{m}$ is the average accuracy over 10 runs, and s represents standard error.

Model	No weight decay (%)	Weight decay (%)	Norm-bias (%)
ResNet	93.40 (±0.04)	94.76 (±0.03)	**94.99** (±0.05)
DenseNet	90.78 (±0.08)	92.26 (±0.06)	**92.46** (±0.04)
MobileNetV2	92.84 (±0.05)	**93.64** (±0.05)	**93.64** (±0.03)
ResNet Fixup	10.00 (±0.00)	91.42 (±0.04)	**91.55** (±0.07)
MLP	58.88 (±0.10)	58.95 (±0.07)	**59.13** (±0.09)

6.3.1 Details concerning low-norm regularization experiments

Our experiments comparing regularizers all run for 300 epochs with an initial learning rate of 0.1 and decreases by a factor of 10 at epochs 100, 175, 225, and 275. We use the SGD optimizer with momentum 0.9.

We also tried negative weight decay coefficients, which leads to ResNet-18

CIFAR-10 performance above 90% while blowing up parameter norm, but this performance is still suboptimal and is not informative concerning the optimality of minimum norm solutions. One might wonder if high norm-bias coefficients lead to even lower parameter norm than low weight decay coefficients. This question may not be meaningful in the case of networks with batch normalization. In the case of ResNet-20 with Fixup, which does not contain running mean and standard deviation, the average parameter ℓ_2 norm after training with weight decay is 24.51 while that of models trained with norm-bias is 31.62. Below, we perform the same tests on CIFAR-100, a substantially more difficult dataset. Weight decay coefficients are chosen to be ones used in the original paper for the corresponding architecture. Norm-bias coefficient/μ^2 is chosen to be 8100/0.005, 7500/0.001, and 2000/0.0005 for ResNet-18, DenseNet-40, and ResNet-20 with Fixup, respectively, using the same heuristic as described in the main body.

Table 6.7: ResNet-18, DenseNet-40, and ResNet-20 with Fixup initialization trained on normalized CIFAR-100 data with various regularizers. Numerical entries are given by $\overline{m}(\pm s)$, where $\overline{m}$ is the average accuracy over 10 runs, and s represents standard error.

Model	No weight decay (%)	Weight decay (%)	Norm-bias (%)
ResNet	71.73 (±0.25)	74.66 (±0.17)	**75.90** (±0.16)
DenseNet	65.61 (±0.33)	68.98 (±0.25)	**69.24** (±0.11)
ResNet Fixup	1.000 (±0.00)	65.08 (±0.30)	**65.58** (±0.17)

6.4 Kernel theory and the infinite-width limit

In light of the recent surge of works discussing the properties of neural networks in the infinite-width limit, in particular, connections between infinite-width deep neural networks and Gaussian processes, see [100], several interesting theoretical works have appeared. The wide network limit and Gaussian process interpretations have inspired work on the neural tangent kernel [81], while [101] and [14] have used wide network assumptions to analyze the training dynamics of deep networks. The connection of deep neural networks to kernel-based learning theory seems promising, but how closely do current architectures match the predictions made for simple networks in the large-width limit?

We focus on the Neural Tangent Kernel (NTK), developed in [81]. Theory dictates that, in the wide-network limit, the neural tangent kernel remains nearly constant as a network trains. Furthermore, neural network training dynamics can be described as gradient descent on a convex functional, provided the NTK remains nearly constant during training [101]. In this section, we experimentally test the validity of these theoretical assumptions.

Fixing a network architecture, we use $\mathcal{F}$ to denote the function space parametrized by $\phi \in \mathbb{R}^p$. For the mapping $F : \mathbb{R}^P \to \mathcal{F}$, the NTK is defined by

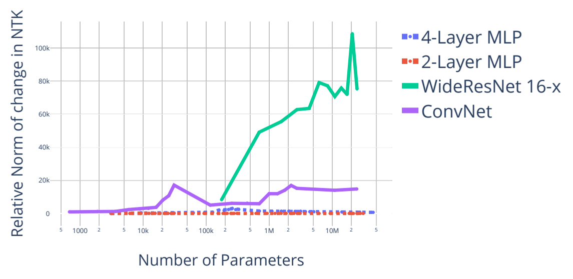
$$\Phi(\phi) = \sum_{p=1}^{P} \partial_{\phi_p} F(\phi) \otimes \partial_{\phi_p} F(\phi), \tag{6.4}$$

where the derivatives $\partial_{\phi_p} F(\phi)$ are evaluated at a particular choice of ϕ describing a neural network. The NTK can be thought of as a similarity measure between images; given any two images as input, the NTK returns an $n \times n$ matrix, where n is the dimensionality of the feature embedding of the neural network. We sample entries from the NTK by drawing a set of N images $\{x_i\}$ from a dataset, and computing the entries in the NTK corresponding to all pairs of images in our image set. We do this for a random neural network $f : \mathbb{R}^m \to \mathbb{R}^n$ and computing the tensor $\Phi(\phi) \in \mathbb{R}^{N \times N \times n \times n}$ of all pairwise realizations, restricted to the given data:

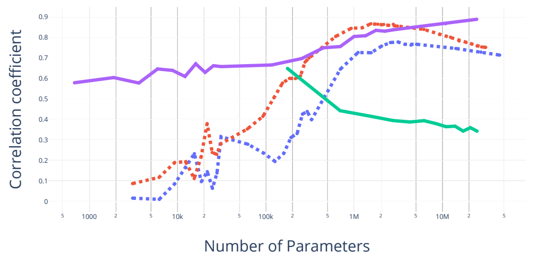
$$\Phi(\phi)_{ijkl} = \sum_{p=1}^{P} \partial_{\phi_p} f(\mathbf{x}_i, \phi)_k \cdot \partial_{\phi_p} f(\mathbf{x}_j, \phi)_l \tag{6.5}$$

By evaluating (6.5) using automatic differentiation, we compute slices from the NTK before and after training for a large range of architectures and network widths. We consider image classification on CIFAR-10 and compare a two-layer MLP, a four-layer MLP, a simple 5-layer ConvNet, and a ResNet. We draw 25 random images from CIFAR-10 to sample the NTK before and after training. We measure the change in the NTK by computing the correlation coefficient of the (vectorized) NTK before and after training. We do this for many network widths, and see what happens in the wide network limit. For MLPs we increase the width of the hidden layers, for the ConvNet (6-Layer, Convolutions, ReLU, MaxPooling), we increase the number of convolutional filters, for the ResNet we consider the WideResnet [173] architecture, where we increase its width parameter. We initialize all models with uniform He initialization as discussed in [66], departing from specific Gaussian initializations in theoretical works to analyze the effects for modern architectures and methodologies.

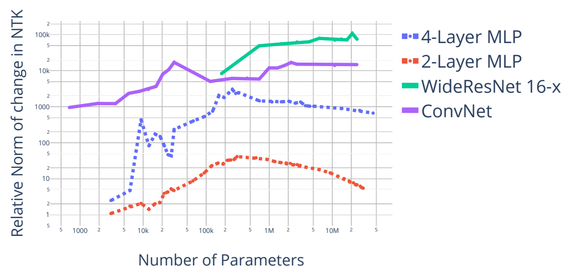
The results are visualized in Figure 6.1, where we plot parameters of the NTK for these different architectures, showing how the number of parameters impacts the relative change in the NTK ($||\Phi_1 - \Phi_0|| / ||\Phi_0||$, where Φ_0/Φ_1 denotes the sub-sampled



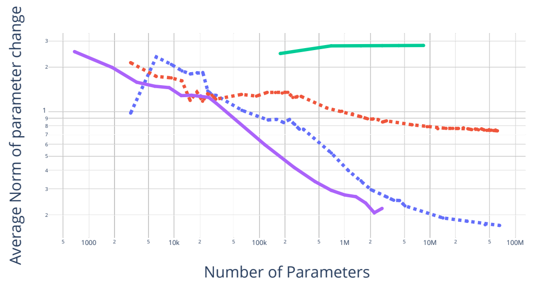
(a)



(b)



(c)



(d)

Figure 6.1: (a) The relative norm of the neural tangent kernel as a function of the number of parameters is shown for several networks. This figure highlights the difference between the behavior of ResNets and other architectures. Figure 6.1c visualizes the same data in a logarithmic scale. (b) The correlation of the neural tangent kernel before and after training. We expect this coefficient to converge toward 1 in the infinite-width limit for multi-layer networks as in [81]. We do not observe this trend for ResNets as is clear from the curve corresponding to the WideResNet. (d) The average norm of parameter change decreases for simple architectures but stays nearly constant for the WideResNet.

NTK before/after training) and correlation coefficient $(\mathrm{Cov}(\Phi_1, \Phi_0)/\sigma(\Phi_1)/\sigma(\Phi_0))$. [81] predicts that the NTK should change very little during training in the infinite-width limit.

At first glance, it might seem that these expectations are hardly met for our (non-infinite) experiments. Figure 6.1a and Figure 6.1c show that the relative change in the NTK during training (and also the magnitude of the NTK) is rapidly increasing with width and remains large in magnitude for a whole range of widths of convolutional architectures. The MLP architectures do show a trend toward small changes in the NTK, yet convergence to zero is slower in the 4-Layer case than in the 2-Layer case.

However, a closer look shows that almost all of the relative change in the NTK seen in Figure 6.1c is explained by a simple linear re-scaling of the NTK. It should be noted that the scaling of the NTK is strongly effected by the magnitude of parameters at initialization. Within the NTK theory of [100], a linear rescaling of the NTK during training corresponds simply to a change in learning rate, and so it makes more sense to measure similarity using a scale-invariant metric.

Measuring similarity between sub-sampled NTKs using the scale-invariant correlation coefficient, as in Figure 6.1b, is more promising. Surprisingly, we find that, as predicted in [81], the NTK changes very little (beyond a linear rescaling) for the wide ConvNet architectures. For the dense networks, the predicted trend toward small changes in the NTK also holds for most of the evaluated widths, although there is a dropoff at the end which may be an artifact of the difficulty of training these wide networks on CIFAR-10. For the Wide Residual Neural Networks, however, the general trend toward higher correlation in the wide network limit is completely reversed. The correlation coefficient decreases as network width increases, suggesting that the neural tangent kernel at initialization and after training becomes qualitatively more different as network width increases. The reversal of the correlation trend seems to be a property which emerges from the interaction of batch normalization and skip connections. Removing either of these features from the architecture leads to networks which have an almost constant correlation coefficient for a wide range of network widths, see Figure 6.5, calling for the consideration of both properties in new formulations of the NTK.

In conclusion, we see that although the NTK trends towards stability as the width of simple architectures increases, the opposite holds for the highly performant Wide ResNet architecture. Even further, neither the removal of batch normalization or the removal of skip connections fully recover the positive NTK trend. While we have hope that kernel-based theories of neural networks may yield guarantees for realistic (albeit wide) models in the future, current results do not sufficiently describe state-of-the-art architectures. Moreover, the already good behavior of models with unstable NTKs is an indicator that good optimization and generalization behaviors do not fundamentally hinge on the stability of the NTK.

6.4.1 Details on the Neural Tangent Kernel experiment

For further reference, we include details on the NTK sampling during training epochs in Figure 6.2. We see that the parameter norm (Right) behaves normally (all of these experiments are trained with a standard weight decay parameter of 0.0005), yet the NTK norm (Left) rapidly increases. Most of this increase, however is scaling of the kernel, as the correlation plot (Middle) is much less drastic. We do see that most change happens in the very first epochs of training, whereas the kernel only changes slowly later on.

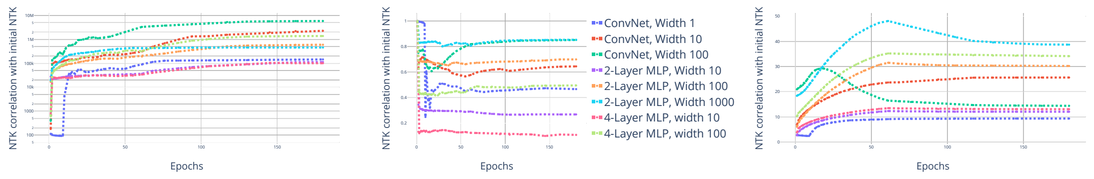


Figure 6.2: Plotting the evolution of NTK parameters during training epochs. Left: Norm of the NTK Tensor, Middle: Correlation of current NTK iterate versus initial NTK. Right: Reference plot of the network parameter norms.

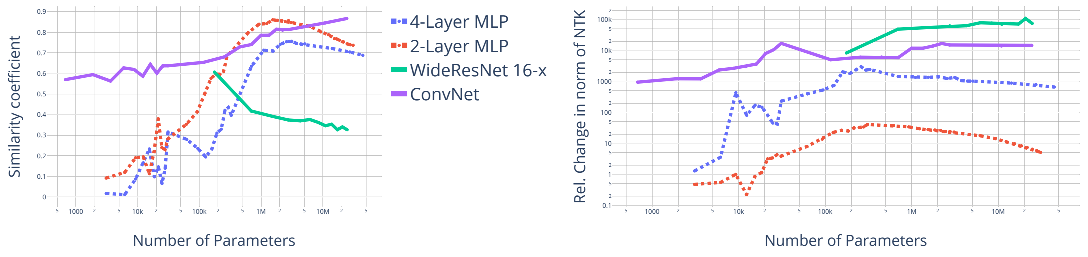


Figure 6.3: The similarity coefficient of the neural tangent kernel after training with its initialization. We expect this coefficient to converge toward 1 in the infinite-width limit for multi-layer networks. Also shown is the direct relative difference of the NTK norms, which behaves similarly to the normalized direct difference from figure 6.1.

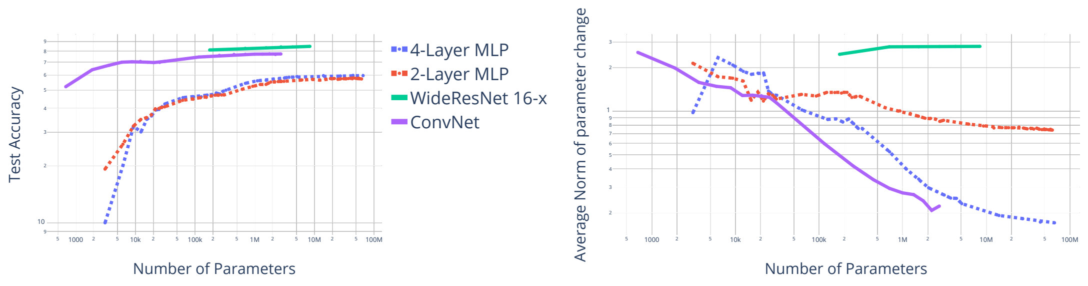


Figure 6.4: For reference we record the test accuracy of all models from 6.1 in the left plot and the relative change in parameters in the right plot.

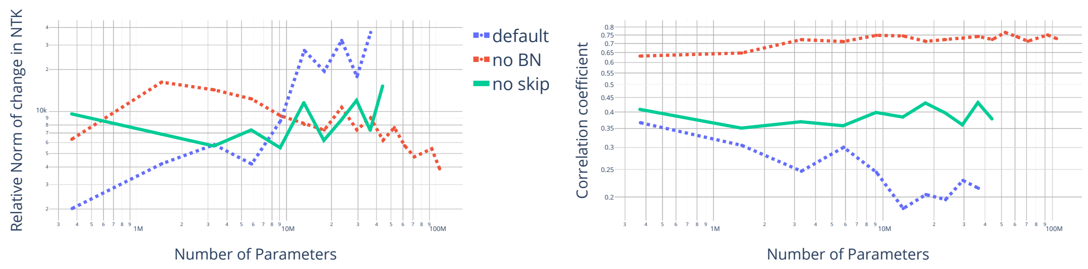


Figure 6.5: The correlation coefficient of the neural tangent kernel after training with its initialization for different WideResNet variants - namely WideResNet without batch normalizations and WideResNet without skip connections. We interestingly find that removing either of both properties, which are widely regarding as beneficial for neural network training, stabilizes the trend seen in the default WideResNet. However both variants hardly converge toward 1, even when sampling very wide ResNets.

6.5 Rank: Do networks with low-rank layers generalize better?

State-of-the-art neural networks are highly over-parameterized, and their large number of parameters is a problem both for learning theory and for practical use. In the theoretical setting, rank has been used to tighten bounds on the generalization gap of neural networks. Generalization bounds from [62] are improved under conditions of low rank and high sparsity [122] of parameter matrices, and the compressibility of low-rank matrices (and other low-dimensional structure) can be directly exploited to provide even stronger bounds [6]. Further studies show a tendency of stochastic gradient methods to find low-rank solutions [82]. The tendency of SGD to find low-rank operators, in conjunction with results showing generalization bounds for low-rank operators, might suggest that the low-rank nature of these operators is important for generalization.

[96] claim that low-rank networks, in addition to generalizing well to test data, are more robust to adversarial attacks. Theoretical and empirical results from the aforementioned paper lead the authors to make two major claims. First, the authors claim that networks which undergo adversarial training have low-rank and sparse matrices. Second, they claim that networks with low-rank and sparse parameter matrices are more robust to adversarial attacks. We find in our experiments that neither claim holds up in practical settings, including ResNet-18 models trained on CIFAR-10.

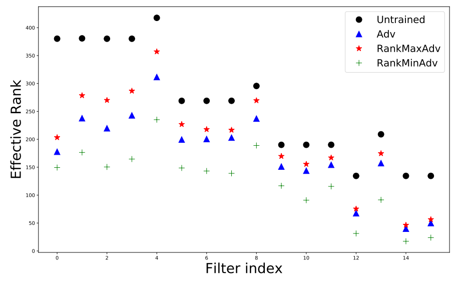
We test the generalization and robustness properties of neural networks with low-rank and high-rank operators by promoting low-rank or high-rank parameter matrices in late epochs. We employ the regularizer introduced in [142] to create the protocols RankMin, to find low-rank parameters, and RankMax, to find high-rank parameters. RankMin involves fine-tuning a pre-trained model by replacing linear operators with their low-rank approximations, retraining, and repeating this process. Similarly, RankMax involves fine-tuning a pre-trained model by clipping singular values from the SVD of parameter matrices in order to find high-rank approximations. We are able to manipulate the rank of matrices without strongly affecting the performance of the network. We use both natural training and 7-step projected gradient descent (PGD) adversarial training routines [115]. The goal of the experiment is to observe how the rank of weight matrices impacts generalization and robustness. We start by attacking naturally trained models with the standard PGD adversarial attack with $\epsilon = 8/255$. Then, we move to the adversarial training setting and test the effect of manipulating rank on generalization and on robustness.

Model	Training method	Clean Test Accuracy (%)	Robust (%) $\epsilon = 8/255$	Robust (%) $\epsilon = 1/255$
ResNet-18	Natural	94.66	0.00	31.98
	RankMax	93.66	0.00	22.01
	RankMin	94.44	0.00	31.53
	Adversarial	79.37	35.38	74.27
	RankMaxAdv	80.00	35.55	74.92
	RankMinAdv	78.34	33.68	73.19
ResNet-18 w/o skips	Natural	92.95	0.01	31.34
	RankMax	91.71	0.00	18.81
	RankMin	92.42	0.00	30.37
	Adversarial	79.57	35.95	74.88
	RankMaxAdv	79.43	36.45	74.87
	RankMinAdv	78.52	33.97	73.64

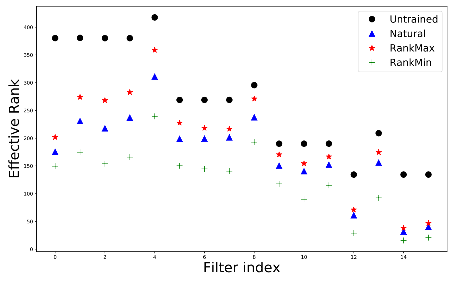
Table 6.8: Result presented here are from experiments with CIFAR-10 data and two of the architectures we studied. Robust accuracy is measured with 20-step PGD attacks with the ϵ values specified at the top of the column.

In order to compare our results with [96], we borrow the notion of effective rank, denoted by $r(W)$ for some matrix W. This continuous relaxation of rank is defined as follows. $r(W) = \frac{\|W\|_*}{\|W\|_F}$ where $\|\cdot\|_*$, $\|\cdot\|_1$, and $\|\cdot\|_F$ are the nuclear norm, the 1-norm, and the Frobenius norm, respectively. Note that the singular values of convolution operators can be found quickly with a method from [142], and that method is used here.

In our experiments we investigate two architectures, ResNet-18 and ResNet-18 without skip connections. We train on CIFAR-10 and CIFAR-100, both naturally and adversarially. Table 6.8 shows that RankMin and RankMax achieve similar generalization on CIFAR-10. More importantly, when adversarially training, a setting when robustness is undeniably the goal, we see the RankMax outperforms both



(a) Effective rank of naturally trained models.



(b) Effective rank of adversarially trained models.

Figure 6.6: This plot shows the effective rank of each filter for the ResNet-18 models. The filters are indexed on the x-axis, so moving to the right is like moving through the layers of the network. Our routines designed to manipulate the rank have exactly the desired effect as shown here.

RankMin *and* standard adversarial training in robust accuracy. Figure 6.6 confirms that these two training routines do, in fact, control effective rank. Experiments with CIFAR-100 yield similar results and are presented in Section 6.5.1. It is clear that increasing rank using an analogue of rank minimizing algorithms does not harm performance. Moreover, we observe that adversarial robustness does not imply low-rank operators, nor do low-rank operators imply robustness. The findings in [82] are corroborated here as the black dots in Figures 6.6 show that initializations are higher in rank than the trained models. Our investigation into what useful intuition in practical cases can be gained from the theoretical work on the rank of CNNs and from the claims about adversarial robustness reveals that rank plays little to no role in the performance of CNNs in the practical setting of image classification.

6.5.1 Details on RankMin and RankMax

We employ routines to promote both low-rank and high-rank parameter matrices. We do this by computing approximations to the linear operators at each layer. Since convolutional layers are linear operations, we know that there is a matrix whose dimensions are the number of parameters in the input to the convolution and the number of parameters in the output of the convolution. In order to compute low-rank approximations of these operators, one could write down the matrix corresponding to the convolution, and then compute a low-rank approximation using a singular value decomposition (SVD). In order to make this problem computationally tractable we used the method for computing singular values of convolution operators derived in [142]. We were then able to do low-rank approximation in the classical sense, by setting each singular value below some threshold to zero. In order to compute high-rank operators, we clipped the singular values so that when mulitplying the SVD factors, we set each singular value to be equal to the minimum of some chosen constant and the true singular value. It is important to note here that these approximations to the convolutional layers, when done naively, can return convolutions with larger filters. To be precise, an $n \times n$ filter will map to a $k \times k$ filter through our rank modifications, where $k \geq n$. We follow the method in [142], where these filters are pruned back down by only using $n \times n$ entries in the output.

When naturally training ResNet-18 and Skipless ResNet-18 models, we train with a batch size of 128 for 200 epochs with the learning rate initiated to 0.01 and decreasing by a factor of 10 at epochs 100, 150, 175, and 190 (for both CIFAR-

10 and CIFAR-100). When adversarially training these two models on CIFAR-10 data, we use the same hyperparameters. However, in order to adversarially train on CIFAR-100, we train ResNet-18 with a batch size of 256 for 300 epochs with an initial learning rate of 0.1 and a decrease by a factor of 10 at epochs 200 and 250. For adversarially training Skipless ResNet-18 on CIFAR-100, we use a batch size of 256 for 350 epochs with an initial learning rate of 0.1 and a decrease by a factor of 10 at epochs 200, 250, and 300. Adversarial training is done with an ℓ_∞ 7-step PGD attack with a step size of 2/255, and $\epsilon = 8/255$. For all of the training described above we augment the data with random crops and horizontal flips.

During 15 additional epochs of training we manipulate the rank as follows. RankMin and RankMax protocols are employed periodically in the last 15 epochs taking care to make sure that the loss remains small. For these last epochs, the learning rate starts at 0.001 and decreases by a factor of 10 after the third and fifth epochs of the final 15 epochs. As shown in Table 6.10, we test the accuracy of each model on clean test data from the corresponding dataset, as well as on adversarial examples generated with 20-step PGD with $\epsilon = 8/255$ (with step size equal to 2/255) and $\epsilon = 1/255$ (with step size equal to .25/255).

When training multi-layer perceptrons on CIFAR-10, we train for 100 epochs with learning rate initialized to 0.01 and decreasing by a factor of 10 at epochs 60, 80 and 90. Then, we train the network for 8 additional epochs, during which RankMin and RankMax networks undergo rank manipulation.

Table 6.9: Results from rank experiments with a multi-layer perceptron and CIFAR-10.

MLP and CIFAR-10

Training method	Training Accuracy (%)	Clean Accuracy (%)	Robust (%) $\epsilon = 8/255$	Robust (%) $\epsilon = 1/255$
Naturally Trained	100.00	58.79	3.76	28.94
RankMax	99.97	58.19	3.72	26.63
RankMin	100.00	58.06	3.76	28.48

Table 6.10: Results from rank experiments on CIFAR-100. Robust accuracy is measured with 20-step PGD attacks with the ϵ values specified at the top of the column.

ResNet-18 and CIFAR-100

Training method	Training Accuracy (%)	Clean Accuracy (%)	Robust (%) $\epsilon = 8/255$	Robust (%) $\epsilon = 1/255$
Naturally Trained	99.97	73.08	0.00	17.5
RankMax	99.90	72.67	0.00	16.95
RankMin	99.92	72.57	0.00	17.63
Adversarially Trained	99.92	50.88	17.81	45.99
RankMaxAdv	99.73	51.04	16.80	45.74
RankMinAdv	99.91	50.22	16.64	45.03

ResNet-18 w/o skip connections and CIFAR-100

Training method	Training Accuracy (%)	Clean Accuracy (%)	Robust (%) $\epsilon = 8/255$	Robust (%) $\epsilon = 1/255$
Naturally Trained	99.96	72.13	0.01	13.7
RankMax	99.82	71.35	0.04	11.74
RankMin	99.90	71.28	0.00	13.53
Adversarially Trained	99.92	50.47	17.62	45.18
RankMaxAdv	99.90	50.93	17.72	45.78
RankMinAdv	99.91	49.37	16.77	44.41

6.6 Conclusion

This work highlights the gap between deep learning theory and observations in the real-world setting. We underscore the need to carefully examine the assumptions of theory and to move past the study of toy models, such as deep linear networks or single-layer MLPs, whose traits do not describe those of the practical realm. First, we show that realistic neural networks on realistic learning problems contain suboptimal local minima. Second, we show that low-norm parameters may not be optimal for neural networks, and in fact, biasing parameters to a non-zero norm during training improves performance on several popular datasets and a wide range of networks. Third, we show that the wide-network trends in the neural tangent kernel do not hold for ResNets and that the interaction between skip connections and batch normalization play a large role. Finally, we show that low-rank linear operators and robustness are not correlated, especially for adversarially trained models.

Bibliography

[1] Abdallah, A., Maarof, M. A., and Zainal, A. (2016). Fraud detection system: A survey. *Journal of Network and Computer Applications*, 68:90–113.

[2] Almgren, R. and Chriss, N. (2001). Optimal execution of portfolio transactions. *Journal of Risk*, 3:5–40.

[3] Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.-J., Srivastava, M., and Chang, K.-W. (2018). Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*.

[4] Angel, J. J. and McCabe, D. (2013). Fairness in financial markets: The case of high frequency trading. *Journal of Business Ethics*, 112(4):585–595.

[5] Arévalo, A., Niño, J., Hernández, G., and Sandoval, J. (2016). High-frequency trading strategy based on deep neural networks. In *International conference on intelligent computing*, pages 424–436. Springer.

[6] Arora, S., Ge, R., Neyshabur, B., and Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, pages 254–263.

[7] Athalye, A., Carlini, N., and Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*.

[8] Avramovic, A., Lin, V., and Krishnan, M. (2017). We're all high frequency traders now. *Credit Suisse Market Structure White Paper*.

[9] Balduzzi, D., Frean, M., Leary, L., Lewis, J. P., Ma, K. W.-D., and McWilliams, B. (2017). The Shattered Gradients Problem: If resnets are the answer, then what is the question? *arXiv:1702.08591 [cs, stat]*.

[10] Beck, A., Kim, Y. S. A., Rachev, S., Feindt, M., and Fabozzi, F. (2013). Empirical analysis of ARMA-GARCH models in market risk estimation on high-frequency US data. *Studies in Nonlinear Dynamics and Econometrics*, 17(2):167–177.

[11] Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36.

[12] Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. (2018). Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*.

[13] Bhattacharyya, S., Jha, S., Tharakunnel, K., and Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3):602–613.

[14] Bietti, A., Mialon, G., Chen, D., and Mairal, J. (2018). A Kernel Perspective for Regularizing Deep Neural Networks. *arXiv:1810.00363 [cs, stat]*.

[15] Binns, R. (2017). Fairness in machine learning: Lessons from political philosophy. *arXiv preprint arXiv:1712.03586*.

[16] Borovkova, S. and Tsiamas, I. (2019). An ensemble of LSTM neural networks for high-frequency stock market classification. *Journal of Forecasting*, 38(6):600–619.

[17] Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer.

[18] Bruna, J. and Mallat, S. (2013). Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886.

[19] Byrd, D. and Balch, T. H. (2019). Intra-day equity price prediction using deep learning as a measure of market efficiency. *arXiv preprint arXiv:1908.08168*.

[20] Carlini, N. and Wagner, D. (2016). Defensive distillation is not robust to adversarial examples. *arXiv preprint arXiv:1607.04311*.

[21] Carlini, N. and Wagner, D. (2017). Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE.

[22] Chen, W.-C., Chang, C.-C., Lu, C.-Y., and Lee, C.-R. (2018). Knowledge distillation with feature maps for image classification. *arXiv preprint arXiv:1812.00660*.

[23] Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., and Huang, J.-B. (2019). A closer look at few-shot classification. In *International conference on learning representations*.

[24] Chiang, P.-Y., Geiping, J., Goldblum, M., Goldstein, T., Ni, R., Reich, S., and Shafahi, A. (2019). Witchcraft: Efficient pgd attacks with random step size. *arXiv preprint arXiv:1911.07989*.

[25] Chollet, F. (2016). Xception: Deep Learning with Depthwise Separable Convolutions. *arXiv:1610.02357 [cs].*

[26] Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2014). The Loss Surfaces of Multilayer Networks. *arXiv:1412.0233 [cs].*

[27] Cohen, J. M., Rosenfeld, E., and Kolter, J. Z. (2019a). Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918.*

[28] Cohen, N., Balch, T., and Veloso, M. (2019b). Trading via image classification. *arXiv preprint arXiv:1907.10046.*

[29] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[30] Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830.*

[31] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2018). Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501.*

[32] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[33] Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., and Song, L. (2018). Adversarial attack on graph structured data. *arXiv preprint arXiv:1806.02371.*

[34] De Prado, M. L. (2018). *Advances in financial machine learning.* John Wiley & Sons.

[35] Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. (2017). Sharp Minima Can Generalize For Deep Nets. *arXiv:1703.04933 [cs].*

[36] Dixon, M. (2018). Sequence classification of the limit order book using recurrent neural networks. *Journal of computational science*, 24:277–286.

[37] Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. (2018a). Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9185–9193.

[38] Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., and Li, J. (2018b). Boosting adversarial attacks with momentum. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[39] Dormehl, L. (2019). Artificial neural network : Beginning of the ai revolution. *Digital Trends.*

[40] Doster, T. and Watnik, A. T. (2017). Machine learning approach to oam beam demultiplexing via convolutional neural networks. *Applied optics*, 56(12):3386–3396.

[41] Dou, Z., Osher, S. J., and Wang, B. (2018). Mathematical analysis of adversarial attacks. *arXiv preprint arXiv:1811.06492*.

[42] Du, S., Lee, J., Li, H., Wang, L., and Zhai, X. (2019). Gradient Descent Finds Global Minima of Deep Neural Networks. In *International Conference on Machine Learning*, pages 1675–1685.

[43] Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098.

[44] Dziugaite, G. K., Ghahramani, Z., and Roy, D. M. (2016). A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853*.

[45] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2017). Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*.

[46] Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018). Robust physical-world attacks on deep learning visual classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634.

[47] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

[48] Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

[49] Gidaris, S. and Komodakis, N. (2018). Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375.

[50] Goldblum, M., Fowl, L., and Czaja, W. (2019a). Sheared multi-scale weight sharing for multi-spectral superresolution. In *Algorithms, Technologies, and Applications for Multispectral and Hyperspectral Imagery XXV*, volume 10986, page 109860X. International Society for Optics and Photonics.

[51] Goldblum, M., Fowl, L., Feizi, S., and Goldstein, T. (2019b). Adversarially robust distillation. *arXiv preprint arXiv:1905.09747*.

[52] Goldblum, M., Fowl, L., and Goldstein, T. (2019c). Adversarially robust few-shot learning: A meta-learning approach. *arXiv preprint arXiv:1910.00982*.

[53] Goldblum, M., Fowl, L., and Goldstein, T. (2019d). Robust few-shot learning with adversarially queried meta-learners. *arXiv preprint arXiv:1910.00982*.

[54] Goldblum, M., Geiping, J., Schwarzschild, A., Moeller, M., and Goldstein, T. (2019e). Truth or backpropaganda? an empirical investigation of deep learning theory. *arXiv preprint arXiv:1910.00359*.

[55] Goldblum, M., Reich, S., Fowl, L., Ni, R., Cherepanova, V., and Goldstein, T. (2020a). Unraveling meta-learning: Understanding feature representations for few-shot tasks. *arXiv preprint arXiv:2002.06753*.

[56] Goldblum, M., Schwarzschild, A., Cohen, N., Balch, T., Patel, A. B., and Goldstein, T. (2020b). Adversarial attacks on machine learning systems for high-frequency trading. *arXiv preprint arXiv:2002.09565*.

[57] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014a). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

[58] Goodfellow, I. J., Vinyals, O., and Saxe, A. M. (2014b). Qualitatively characterizing neural network optimization problems. *arXiv:1412.6544 [cs, stat]*.

[59] Guo, C., Rana, M., Cisse, M., and Van Der Maaten, L. (2017). Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*.

[60] Haeffele, B. D. and Vidal, R. (2017). Global Optimality in Neural Network Training. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4390–4398.

[61] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951.

[62] Harvey, N., Liaw, C., and Mehrabian, A. (2017). Nearly-tight vc-dimension bounds for piecewise linear neural networks. *CoRR*, abs/1703.02930.

[63] Hastie, T., Montanari, A., Rosset, S., and Tibshirani, R. J. (2019). Surprises in high-dimensional ridgeless least squares interpolation. *arXiv preprint arXiv:1903.08560*.

[64] He, F., Wang, B., and Tao, D. (2020). Nonlinearities in activations substantially shape the loss surfaces of neural networks. *International Conference of Learning Representations*.

[65] He, K., Zhang, X., Ren, S., and Sun, J. (2015a). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*.

[66] He, K., Zhang, X., Ren, S., and Sun, J. (2015b). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *arXiv:1502.01852 [cs]*.

[67] He, K., Zhang, X., Ren, S., and Sun, J. (2016a). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[68] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[69] Hendershott, T., Jones, C. M., and Menkveld, A. J. (2011). Does algorithmic trading improve liquidity? *The Journal of finance*, 66(1):1–33.

[70] Henning, P. J. (2018). The problem with prosecuting spoofing. *The New York Times*.

[71] Hernández-García, A. and König, P. (2018). Do deep nets really need weight decay and dropout? *arXiv:1802.07042 [cs]*.

[72] Hestenes, M. R. and Stiefel, E. (1952). *Methods of conjugate gradients for solving linear systems*, volume 49. NBS Washington, DC.

[73] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[74] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.

[75] Hoffer, E., Banner, R., Golan, I., and Soudry, D. (2018). Norm matters: Efficient and accurate normalization schemes in deep networks. *arXiv:1803.01814 [cs, stat]*.

[76] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[77] Huang, R. and Polak, T. (2011). Lobster: Limit order book reconstruction system. *Available at SSRN 1977207*.

[78] Huang, W. R., Emam, Z., Goldblum, M., Fowl, L., Terry, J. K., Huang, F., and Goldstein, T. (2019). Understanding generalization through visualizations. *arXiv preprint arXiv:1906.03291*.

[79] Ilyas, A., Jalal, A., Asteri, E., Daskalakis, C., and Dimakis, A. G. (2017). The robust manifold defense: Adversarial training using generative models. *arXiv preprint arXiv:1712.09196*.

[80] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[81] Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural Tangent Kernel: Convergence and Generalization in Neural Networks. *arXiv:1806.07572 [cs, math, stat]*.

[82] Ji, Z. and Telgarsky, M. (2018). Gradient descent aligns the layers of deep linear networks. *arXiv preprint arXiv:1810.02032*.

[83] Kaiser, Ł., Nachum, O., Roy, A., and Bengio, S. (2017). Learning to remember rare events. *arXiv preprint arXiv:1703.03129*.

[84] Kannan, H., Kurakin, A., and Goodfellow, I. (2018). Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*.

[85] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation.

[86] Kawaguchi, K. (2016). Deep Learning without Poor Local Minima. *arXiv:1605.07110 [cs, math, stat]*.

[87] Kercheval, A. N. and Zhang, Y. (2015). Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8):1315–1329.

[88] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv:1609.04836 [cs, math]*.

[89] Kirilenko, A., Kyle, A. S., Samadi, M., and Tuzun, T. (2017). The flash crash: High-frequency trading in an electronic market. *The Journal of Finance*, 72(3):967–998.

[90] Klaus, T. and Elzweig, B. (2017). The market impact of high-frequency trading systems and potential regulation. *Law and Financial Markets Review*, 11(1):13–19.

[91] Kluwer, W. (2020). The Nasdaq stock market LLC rules. *http://nasdaq.cchwallstreet.com/*.

[92] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

[93] Krogh, A. and Hertz, J. A. (1992). A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957.

[94] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

[95] Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

[96] Langenberg, P., Balda, E. R., Behboodi, A., and Mathar, R. (2019). On the effect of low-rank weights on adversarial robustness of neural networks. *CoRR*, abs/1901.10371.

[97] Laurent, T. and Brecht, J. (2018). Deep linear networks with arbitrary loss: All local minima are global. In *International Conference on Machine Learning*, pages 2908–2913.

[98] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[99] Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., and et al. (2017). Photo-realistic single image super-resolution using a generative adversarial network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[100] Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017a). Deep Neural Networks as Gaussian Processes. *arXiv:1711.00165 [cs, stat]*.

[101] Lee, J., Xiao, L., Schoenholz, S. S., Bahri, Y., Novak, R., Sohl-Dickstein, J., and Pennington, J. (2019a). Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *arXiv:1902.06720 [cs, stat]*.

[102] Lee, J.-G., Jun, S., Cho, Y.-W., Lee, H., Kim, G. B., Seo, J. B., and Kim, N. (2017b). Deep learning in medical imaging: general overview. *Korean journal of radiology*, 18(4):570–584.

[103] Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019b). Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665.

[104] Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2016). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

[105] Li, W. (2018a). *Topics in Harmonic Analysis, Sparse Representations, and Data Analysis*. PhD thesis.

[106] Li, Y. (2018b). *Feature extraction in image processing and deep learning*. PhD thesis.

[107] Liang, S., Sun, R., Li, Y., and Srikant, R. (2018). Understanding the Loss Surface of Neural Networks for Binary Classification. *arXiv:1803.00909 [cs, stat]*.

[108] Lin, J., Gan, C., and Han, S. (2019). Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*.

[109] Liu, Y., Tracey, B., Aeron, S., Miller, E., Sun, T., McDannold, N., and Murphy, J. (2019). Artifact suppression for passive cavitation imaging using u-net cnns with uncertainty quantification. In *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, pages 1037–1042. IEEE.

[110] Mensink, T., Verbeek, J., Perronnin, F., and Csurka, G. (2012). Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *European Conference on Computer Vision*, pages 488–501. Springer.

[111] Meola, A. (2017). Robo advisors: Online financial advisors that fit in your pocket. *Business Insider Magazine*.

[112] Mittal, S. (2018). A survey of fpga-based accelerators for convolutional neural networks. *Neural computing and applications*, pages 1–31.

[113] Miyato, T., Maeda, S.-i., Ishii, S., and Koyama, M. (2018). Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*.

[114] Mądry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017a). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

[115] Mądry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017b). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

[116] Moosavi-Dezfooli, S.-M., Fawzi, A., Fawzi, O., and Frossard, P. (2017). Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773.

[117] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.

[118] Moyer, L. (2015). Putting Robo advisors to the test. *The Wall Street Journal*.

[119] Murphy, J. J. (1999). *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin.

[120] Mustafa, A., Khan, S. H., Hayat, M., Shen, J., and Shao, L. (2019). Image super-resolution as a defense against adversarial attacks. *arXiv preprint arXiv:1901.01677*.

[121] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547.

[122] Neyshabur, B., Bhojanapalli, S., McAllester, D., and Srebro, N. (2017). A pac-bayesian approach to spectrally-normalized margin bounds for neural networks. *CoRR*, abs/1707.09564.

[123] Nguyen, Q., Mukkamala, M. C., and Hein, M. (2018). On the loss landscape of a class of deep neural networks with no bad local valleys. *arXiv:1809.10749 [cs, stat]*.

[124] Njeunje, F. O. N. (2018). *Computational methods in machine learning: transport model, Haar wavelet, DNA classification, and MRI*. PhD thesis.

[125] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE.

[126] Pedersen, L. H. (2019). *Efficiently inefficient: how smart money invests and market prices are determined*. Princeton University Press.

[127] Pekala, M. (2018). *Harmonic Analysis and Machine Learning*. PhD thesis.

[128] Pfister, T., Charles, J., and Zisserman, A. (2014). Domain-adaptive discriminative one-shot learning of gestures. In *European Conference on Computer Vision*, pages 814–829. Springer.

[129] Prakash, A., Moran, N., Garber, S., DiLillo, A., and Storer, J. (2018). Deflecting adversarial attacks with pixel deflection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8571–8580.

[130] Randhawa, K., Loo, C. K., Seera, M., Lim, C. P., and Nandi, A. K. (2018). Credit card fraud detection using adaboost and majority voting. *IEEE access*, 6:14277–14284.

[131] Rosenblatt, F. (1961). Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY.

[132] Saadatpanah, P., Shafahi, A., and Goldstein, T. (2019). Adversarial attacks on copyright detection systems. *arXiv preprint arXiv:1906.07153*.

[133] Safran, I. and Shamir, O. (2017). Spurious Local Minima are Common in Two-Layer ReLU Neural Networks. *arXiv:1712.08968 [cs, stat]*.

[134] Samangouei, P., Kabkab, M., and Chellappa, R. (2018). Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*.

[135] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018a). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

[136] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. (2018b). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

[137] Santana, E. and Hotz, G. (2016). Learning a driving simulator. *arXiv preprint arXiv:1608.01230*.

[138] Santurkar, S., Tsipras, D., Ilyas, A., and Mądry, A. (2018). How Does Batch Normalization Help Optimization? *arXiv:1805.11604 [cs, stat]*.

[139] Sar, M. (2017). Dodd-frank and the spoofing prohibition in commodities markets. *Fordham J. Corp. & Fin. L.*, 22:383.

[140] Schapire, R. E. (1990). The strength of weak learnability. *Machine learning*, 5(2):197–227.

[141] Schölkopf, B., Smola, A., and Müller, K.-R. (1997). Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer.

[142] Sedghi, H., Gupta, V., and Long, P. M. (2018). The singular values of convolutional layers. *arXiv preprint arXiv:1805.10408*.

[143] Sehwag, V., Wange, S., Mittal, P., and Jana, S. (2019). Towards compact and robust deep neural networks. *arXiv preprint arXiv:1906.06110*.

[144] Shafahi, A., Najibi, M., Ghiasi, A., Xu, Z., Dickerson, J., Studer, C., Davis, L. S., Taylor, G., and Goldstein, T. (2019a). Adversarial training for free! *arXiv preprint arXiv:1904.12843*.

[145] Shafahi, A., Najibi, M., Xu, Z., Dickerson, J., Davis, L. S., and Goldstein, T. (2018). Universal adversarial training. *arXiv preprint arXiv:1811.11304*.

[146] Shafahi, A., Saadatpanah, P., Zhu, C., Ghiasi, A., Studer, C., Jacobs, D., and Goldstein, T. (2019b). Adversarially robust transfer learning. *arXiv preprint arXiv:1905.08232*.

[147] Shah, V., Kyrillidis, A., and Sanghavi, S. (2018). Minimum norm solutions do not always generalize well for over-parameterized problems. *arXiv preprint arXiv:1811.07055*.

[148] Shaham, U., Garritano, J., Yamada, Y., Weinberger, E., Cloninger, A., Cheng, X., Stanton, K., and Kluger, Y. (2018a). Defending against adversarial images using basis functions transformations. *arXiv preprint arXiv:1803.10840*.

[149] Shaham, U., Yamada, Y., and Negahban, S. (2018b). Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 307:195–204.

[150] Sharif, M., Bhagavatula, S., Bauer, L., and Reiter, M. K. (2016). Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pages 1528–1540.

[151] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087.

[152] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147.

[153] Suykens, J. A. and Vandewalle, J. (1999). Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300.

[154] Świrszcz, G., Czarnecki, W. M., and Pascanu, R. (2016). Local minima in training of neural networks. *arXiv:1611.06310 [cs, stat]*.

[155] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

[156] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

[157] Tai, C., Xiao, T., Zhang, Y., Wang, X., et al. (2015). Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*.

[158] Thimm, G. and Fiesler, E. (1995). Neural network initialization. In *International Workshop on Artificial Neural Networks*, pages 535–542. Springer.

[159] Van Laarhoven, T. (2017). L2 regularization versus batch and weight normalization. *arXiv preprint arXiv:1706.05350*.

[160] van Laarhoven, T. (2017). L2 Regularization versus Batch and Weight Normalization. *arXiv:1706.05350 [cs, stat]*.

[161] Vartak, M., Thiagarajan, A., Miranda, C., Bratman, J., and Larochelle, H. (2017). A meta-learning perspective on cold-start recommendations for items. In *Advances in neural information processing systems*, pages 6904–6914.

[162] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638.

[163] Wijayanto, A. W., Jin, C. J., Madhawa, K., and Murata, T. (2018). Robustness of compressed convolutional neural networks. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4829–4836. IEEE.

[164] Wu, Z., Lim, S.-N., Davis, L., and Goldstein, T. (2019). Making an invisibility cloak: Real world adversarial attacks on object detectors. *arXiv preprint arXiv:1910.14667.*

[165] Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., and Yuille, A. (2017). Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1369–1378.

[166] Xie, C., Wu, Y., Maaten, L. v. d., Yuille, A. L., and He, K. (2019). Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 501–509.

[167] Xie, C., Wu, Y., van der Maaten, L., Yuille, A., and He, K. (2018). Feature denoising for improving adversarial robustness. *arXiv preprint arXiv:1812.03411.*

[168] Yamaguchi, K., Sakamoto, K., Akabane, T., and Fujimoto, Y. (1990). A neural network for speaker-independent isolated word recognition. In *First International Conference on Spoken Language Processing.*

[169] Yin, C., Tang, J., Xu, Z., and Wang, Y. (2018). Adversarial meta-learning. *arXiv preprint arXiv:1806.03316.*

[170] Yun, C., Sra, S., and Jadbabaie, A. (2018). Small nonlinearities in activation functions create bad local minima in neural networks. *arXiv:1802.03487 [cs, math, stat].*

[171] Yun, C., Sra, S., and Jadbabaie, A. (2019). Small nonlinearities in activation functions create bad local minima in neural networks. In *International Conference on Learning Representations.*

[172] Zagoruyko, S. and Komodakis, N. (2016a). Wide residual networks. *arXiv preprint arXiv:1605.07146.*

[173] Zagoruyko, S. and Komodakis, N. (2016b). Wide Residual Networks. *arXiv:1605.07146 [cs].*

[174] Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization. *arXiv:1611.03530 [cs].*

[175] Zhang, D., Zhang, T., Lu, Y., Zhu, Z., and Dong, B. (2019a). You only propagate once: Painless adversarial training using maximal principle. *arXiv preprint arXiv:1905.00877.*

[176] Zhang, G., Wang, C., Xu, B., and Grosse, R. (2018). Three Mechanisms of Weight Decay Regularization. *arXiv:1810.12281 [cs, stat].*

[177] Zhang, H., Dauphin, Y. N., and Ma, T. (2019b). Fixup Initialization: Residual Learning Without Normalization. *arXiv:1901.09321 [cs, stat].*

[178] Zhang, H., Yu, Y., Jiao, J., Xing, E. P., El Ghaoui, L., and Jordan, M. I. (2019c). Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv: 1901.08573.*

[179] Zhang, H., Yu, Y., Jiao, J., Xing, E. P., Ghaoui, L. E., and Jordan, M. I. (2019d). Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573.*

[180] Zhao, Y., Shumailov, I., Mullins, R., and Anderson, R. (2018a). To compress or not to compress: Understanding the interactions between adversarial attacks and neural network compression. *arXiv preprint arXiv:1810.00208.*

[181] Zhao, Y., Zhu, H., Shen, Q., Liang, R., Chen, K., and Zhang, S. (2018b). Practical adversarial attack against object detector. *arXiv preprint arXiv:1812.10217.*