University of Oxford

Fourth Year Project

Optimal Loss Functions for Distributionally Robust Optimization of Neural Networks

Long Thanh Pham

Keble College

supervised by Prof. Marta Kwiatkowska and Dr. Wenjie Ruan

A thesis submitted for the degree of

Master of Science in Computer Science

Trinity 2019

Abstract

Recent years have seen a surge in the attention drawn by deep learning as its presence gets stronger in numerous application domains, including safety-critical systems such as autonomous vehicles. Although neural networks have superb generalization ability in object recognition, they are surprisingly susceptible to adversarial perturbations that are often imperceptible to the human eye and yet can trick the neural networks into a misclassification. One promising approach to improving the robustness of neural networks is adversarial training, whereby neural networks are trained using adversarially perturbed examples as well as the original training data.

In evaluating the robustness of a neural network, it is a common practice to measure the zero-one loss of the neural network with respect to adversarially perturbed examples. However, because the zero-one loss function is not smooth, other loss functions are instead used to train neural networks. This raises the question of what loss function should be used in the training in order to provide the highest degree of robustness to neural networks. Focusing on a concrete variant of adversarial training called distributionally robust optimization (DRO), this work attempts to empirically answer this question.

The present work has found out that optimal loss functions for DRO vary from DRO algorithm to DRO algorithm. In addition, the cross entropy loss function, which is widely used in training neural networks, yields acceptable robustness in all three DRO algorithms considered in this work. Therefore, as far as the robustness of neural networks in DRO is concerned, there is no need to reconsider the standard use of the cross entropy loss function in training neural networks.

Contents

Chapter 1

Introduction

1.1 Background

Inspired by the structure of the human brain, artificial neural networks (often abbreviated to neural networks) were invented as one of the paradigms for machine learning. Neural networks have recently garnered substantial attention from the research community and industry due to the neural networks' noteworthy performance in a wide range of machine learning tasks where humans have traditionally been superior to computers. Notable examples include

1. Computer vision: Krizhevsky et al. [2012] won the ImageNet Large Scale Visual Recognition Challenge (LSVRC) in 2012 by a significant margin, demonstrating that the neural network is a promising approach to computer vision.

2. Natural language processing: Wu et al. [2016] incorporated neural networks into Google Translate, successfully improving the quality of statistical machine translation.

3. Playing games: Silver et al. [2016] used a neural network to create the program AlphaGo that outperformed pre-existing programs for the game of Go.

As neural networks become increasingly widely employed in safety-critical systems (e.g. autonomous vehicles), it is essential to ensure safety of systems with machine learning components. Despite the stellar performance of neural networks, they have been found to be vulnerable to so-called adversarial perturbations in compute vision [Szegedy et al., 2013]. That is, it is relatively easy to trick neural networks into a misclassification by introducing a small and often imperceptible perturbation to an input image. A sample adversarial example (i.e. an adversarially perturbed image) is given in Figure 1.1.

Since the discovery of adversarial perturbations in neural networks, adversarial perturbations have drawn much attention in the research community. Although adversarial perturbations have been studied extensively, no definitive solution has been found.

One promising approach to enhancing the robustness of neural networks is adversarial training (AT) [Goodfellow et al., 2015], which trains neural networks using not only the

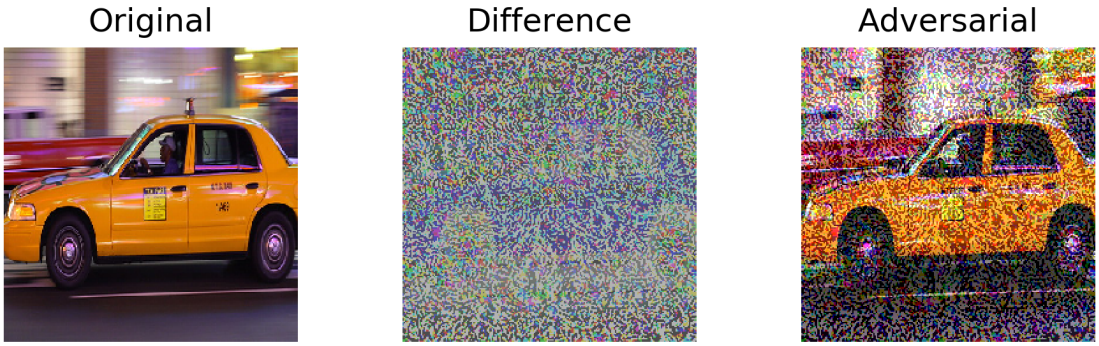

Figure 1.1: Adversarial example of a yellow cab generated using FGSM [Goodfellow et al., 2015]. The original image is correctly classified as a cab with the confidence of 0.999, whilst the adversarial example is incorrectly classified as a jigsaw puzzle with the confidence of 0.629. The adversarial perturbation in this example is discernible, but [Szegedy et al., 2013; Kurakin et al., 2017] contain examples of adversarial perturbations that are hardly recognizable by the human eye. The original image of the yellow cab comes from the ImageNet dataset [Deng et al., 2009], which is publicly available at http://www.image-net.org/.

original training data but also adversarial examples. In conventional AT, a user-specified limit is imposed on the size of a perturbation for each individual datapoint from the training dataset. Thus, conventional AT considers "pointwise" adversarial perturbations. By contrast, a variant of adversarial training called distributionally robust optimization (DRO) considers distributional adversarial perturbations, where a limit is imposed on the overall size of a perturbation on the entire probability distribution (as opposed to perturbations on individual datapoints) [Staib and Jegelka, 2017; Sinha et al., 2018].

A key advantage of DRO over conventional AT is that DRO is a generalization of conventional adversarial training. Consequently, whilst conventional AT assumes that all pointwise adversarial perturbations are equally likely, DRO does not [Staib and Jegelka, 2017].

To execute adversarial training, we need to compute adversarial examples as a subproblem. Intuitively, an adversarial example can be devised by searching for an example where there is a significant difference between the correct result and the actual prediction result of a neural network. More formally, computation of an adversarial example can be formulated as the maximization problem of a loss function, which quantifies the discrepancy between the correct and actual prediction results. The literature abounds with loss functions, and the choice of loss functions depends on a specific machine learning task/algorithm.

In their highly influential paper, Carlini and Wagner [2017] empirically investigate the relationship between loss functions and the robustness of neural networks in the setting of adversarial attacks. Based on the optimal loss function they have identified, they devise a new adversarial attack that successfully circumvents some of the defenses

that had not been compromised previously.

In the literature of adversarial training (including DRO), the robustness of a neural network is measured by first creating adversarial examples from a dataset and then calculating the proportion of (adversarial) examples that the neural network misclassifies [Staib and Jegelka, 2017; Sinha et al., 2018]. In other words, the goal of adversarial training is minimization of the zero-one loss function, which returns 1 if the prediction is different from the correct result and otherwise returns 0. However, since most regions in the zero-one loss function is flat, it is unsuitable to be used in a training algorithm (i.e. backpropagation algorithm) for neural networks. This is because in order to train neural networks, it is necessary to compute gradients of a loss function, and the zero-one loss function rarely gives non-zero gradients. Hence, we need to use a "smoother loss function" in practice when training neural networks, although the objective of adversarial training is minimization of the "zero-one loss function". This begs the question: what loss function is the most suitable in adversarial training where robustness is evaluated in terms of the zero-one loss?

Inspired by [Carlini and Wagner, 2017], in this work, I apply their idea to the setting of DRO, investigating which loss function will offer the highest degree of robustness to neural networks. This research is worthwhile for two reasons:

1. From a theoretical viewpoint, it is interesting to see whether the empirical findings on optimal loss functions by Carlini and Wagner [2017] will carry over from the setting of adversarial attacks to the setting of DRO, which relies on adversarial attacks. It is reasonable to expect that being able to find better adversarial examples in the process of adversarial training will improve the robustness of adversarially trained neural networks. Accordingly, it is also reasonable to expect optimal loss functions in [Carlini and Wagner, 2017] to be optimal in the setting of adversarial training (DRO in particular) as well.

2. From an empirical viewpoint, it is useful to know what loss function(s) will give the highest degree of robustness and whether the cross entropy loss function, which is widely adopted, produces satisfactory robustness in DRO.

1.2 Related work

1.2.1 Properties of adversarial perturbations

Adversarial examples were first reported by Biggio et al. [2013] and Szegedy et al. [2013]. Szegedy et al. [2013] additionally observed the transferability of adversarial examples: a number of adversarial examples carry over from one deep neural network to another network with different hyper-parameters (e.g. architectures and initial weights).

Goodfellow et al. [2015] maintain that adversarial examples are prevalent due to the (locally) linear nature of deep neural networks. For instance, ReLU, which is a popular choice for activation functions because it can prevent gradients from vanishing/exploding, is a piecewise linear function and hence can make neural networks (locally) linear.

Bubeck et al. [2018] construct a classification problem that is easy to learn with a high degree of robustness information-theoretically but is computationally difficult to learn with a high degree of robustness under the statistical query (SQ) learning model. As a result, they show that the difficulty of creating robust neural networks may be due to computational constraints rather than information-theoretic ones.

Jacobsen et al. [2019] investigate invariance-based adversarial examples (as opposed to perturbation-based adversarial examples), reporting that the standard cross entropy loss function commonly used in training neural networks is partially responsible for vulnerability of neural networks to invariance-based adversarial perturbations.

The relationship between loss functions and properties of neural networks (e.g. test accuracy and convergence rate of empirical loss to expected loss) is investigated in [Rosasco et al., 2004; Janocha and Czarnecki, 2017]. Saito and Roy [2018] empirically study the influence of loss functions on the robustness of neural networks to adversarial perturbations. In [Saito and Roy, 2018], neural networks are trained using the standard backpropagation algorithm (and hence not adversarial training). On the other hand, in the present work, I use neural networks that are trained by distributionally robust optimization (DRO), which is a variant of adversarial training.

1.2.2 Adversarial attacks and training

Szegedy et al. [2013] suggest a method to generate adversarial examples based on an optimization method L-BFGS. Exploiting the local linearity of deep neural networks, Goodfellow et al. [2015] use one-step optimization method called the fast gradient sign method (FGSM) to generate adversarial examples. Goodfellow et al. [2015] also experiment with adversarial training using FGSM-generated adversarial examples, reporting that adversarial training makes neural networks more robust.

Madry et al. [2017] propose robust optimization (RO) as a unifying framework of adversarial training. This RO-based formulation of adversarial training has been extended to one based on distributionally robust optimization (DRO) [Staib and Jegelka, 2017; Sinha et al., 2018]. In DRO, we consider perturbations on entire probability distributions of datapoints (instead of perturbations on individual points as in conventional adversarial training). DRO with the Wasserstein distance (in a context outside adversarial training of neural networks) is also investigated in [Blanchet et al., 2016; Gao and Kleywegt, 2016; Mohajerin Esfahani and Kuhn, 2018]. DRO with different metrics (e.g. f-divergences) are studied in [Duchi et al., 2016; Ben-Tal et al., 2013].

Papernot et al. [2016b] introduce defensive distillation to make neural networks more robust. This defense yields promising results in protecting against adversarial examples crafted by four adversarial attacks: L-BFGS, FGSM, Deepfool [Moosavi-Dezfooli et al., 2016], and JSMA [Papernot et al., 2016a]. However, by empirically examining different loss functions for adversarial attacks, Carlini and Wagner [2017] show that solving optimization problems with carefully designed loss functions can foil defensive distillation.

1.3 Contributions

The first contribution of this project is identifying two issues with the current literature on distributionally robust optimization (DRO): (i) mismatch between loss functions used in practice and loss functions in the theoretical formulation of DRO; (ii) the use of pointwise adversarial attacks instead of distributional attacks to evaluate the robustness of neural networks (See Section 3.3).

The second contribution is proving that the DRO algorithm developed by Sinha et al. [2018], which is named WRM, can be viewed as adversarial training in which adversarial examples are computed using the framework of [Carlini and Wagner, 2017] (See Section 4.3). Building on this, the third contribution of the project is empirically studying the relationship between loss functions and the robustness of neural networks trained by three distinct DRO algorithms (including WRM), thereby addressing the first issue mentioned above. I systematically evaluate seven different loss functions from [Carlini and Wagner, 2017] in terms of the resulting robustness in the DRO setting of [Staib and Jegelka, 2017; Sinha et al., 2018] (See Chapter 5).

1.4 Outline of the report

This document is structured as follows.

In Chapter 2, basic terminologies and concepts necessary to understand distributionally robust optimization (DRO) are presented. Specifically, I will introduce robust optimization, supervised learning, neural networks, and adversarial attacks/training.

In Chapter 3, I will provide a formulation of DRO and describe previous research work by Staib and Jegelka [2017] and Sinha et al. [2018] on DRO algorithms. Chapter 4 discusses the empirical studies by Carlini and Wagner [2017] on the relationship between loss functions and robustness of neural networks in the setting of adversarial attacks. The highlight of this chapter is the analysis showing that WRM, which is a DRO algorithm developed by Sinha et al. [2018], can be viewed as adversarial training where adversarial examples are computed using the framework for adversarial attacks in [Carlini and Wagner, 2017].

This is followed by Chapter 5, where the first part describes the experiments conducted to confirm some of the results in the previous research work, and the second part describes the experiments for comparing loss functions in terms of their resulting robustness in three DRO algorithms.

Finally, Chapter 6 concludes the report. Basics of optimization are presented in Appendix A. Appendix B provides details on adversarial attacks and training. Additional analysis and experiment results are available in Appendix C and Appendix D, respectively.

Chapter 2

Preliminaries

2.1 Robust optimization

Robust optimization concerns optimization under uncertainty in data and seeks to produce an optimized solution that is resistant to perturbation in data [Ben-Tal et al., 2009]. To illustrate robust optimization, consider linear programming:

$$\begin{aligned} \text{minimize} \quad & c^\intercal x \\ \text{subject to} \quad & Ax \leq b, \end{aligned}$$

where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$. This can be more succinctly written as

$$\min_x \{c^\intercal x \mid Ax \leq b\}.$$

Here, $c^\intercal x$ is an objective value, and the function $x \mapsto c^\intercal x$ is called an objective function. If x satisfies $Ax \leq b$, then x is said to be a feasible solution. If $x = \arg\min_y \{c^\intercal y \mid Ay \leq b\}$, x is called an optimal solution.

Suppose an optimal solution to the above optimization problem is x^*. If the entries of A possess uncertainty, they may cause x^* to violate the constraint $Ax^* \leq b$. In some application scenarios of optimization, violation of constraints is unacceptable and must be avoided at all costs. In such a situation, it is critical to ensure that a solution remains feasible even when it is subject to perturbations within a certain degree of uncertainty. Thus, in robust optimization, we aim to compute x that optimizes the "worst-case" objective value under uncertainty.

Formally, let $\mathcal{U}$ be the set of possible values for entries in A, b, and c under uncertainty. This $\mathcal{U}$ is called an uncertainty set. The above optimization problem is transformed into the following robust optimization problem [Ben-Tal et al., 2009]:

$$\min_x \left\{ \sup_{(A,b,c) \in \mathcal{U}} c^\intercal x \mid \forall (A, b, c) \in \mathcal{U}. Ax \leq b \right\}.$$

2.2 Empirical risk minimization (ERM)

This section will establish a formal setting of supervised learning, which is one of machine learning paradigms, and will introduce empirical risk minimization (ERM). The presentation style of this section follows that of [Mohri et al., 2012]. Let $\mathcal{X}$ be an instance space (i.e. a set of possible of inputs) and $\mathcal{Y}$ be a set of possible outputs. Because I will focus on classification problems, $\mathcal{Y}$ is finite. Throughout this document, I refer to $x \in \mathcal{X}$ as an instance/example.

A "concept" is a function $c : \mathcal{X} \to \mathcal{Y}$. The goal of supervised learning is to recover c (or its approximation) using a training sample generated by c; i.e. a set $\{(x_1, c(x_1)), \ldots, (x_m, c(x_m))\}$, where $x_i \in \mathcal{X}$ for all i. The output of a machine learning algorithm is a function $h : \mathcal{X} \to \mathcal{Y}$ (called a hypothesis) that approximates c. The distinction between concepts and hypotheses is that concepts are the mapping that generates the data and that we would like to learn, whereas hypotheses are the output mappings of machine learning algorithms.

In a noise-free setting such as the PAC (probably approximately correct) learning model [Valiant, 1984], learning algorithms aim to learn a target concept c that generates training data. In the real-world applications, on the other hand, it is more reasonable to assume that training data contain noise. Therefore, unlike in PAC learning where we work with distributions over instance space $\mathcal{X}$, in this document, we work with distributions $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

Let $\ell : \Theta \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ denote a loss function, where Θ is the set of all possible parameters of a machine learning model (e.g. neural networks). The loss $\ell(\theta, x, y)$ quantifies the discrepancy between y (which is usually the ground-truth label) and the prediction result $F_\theta(x)$. Here, F_θ is a machine learning model with parameter(s) θ and (x, y) is drawn from $\mathcal{D}$ (denoted by $(x, y) \sim \mathcal{D}$). A loss function is usually a hyper-parameter specified by a user.

When we disregard adversarial perturbations, the goal of training a model is to evaluate an approximate minimizer θ for

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[\ell(\theta, x, y)],$$

which is called an expected risk/error/loss.

In practice, $\mathcal{D}$ is unknown, and we are instead given a training sample drawn from $\mathcal{D}$. Given a sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, the empirical loss of a hypothesis θ with respect to this sample is

$$\frac{1}{m}\sum_{i=1}^{m} \ell(\theta, x_i, y_i).$$

Empirical risk minimization (ERM) refers to finding parameters in machine learning models that (approximately) minimize empirical risks.

Under the assumption that each instance in a sample is drawn randomly from $\mathcal{D}$, we

have

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\frac{1}{m}\sum_{i=1}^{m}\ell(\theta,x_i,y_i)\right]=\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[\ell(\theta,x,y)].$$

By the weak law of large numbers, as long as we have sufficiently many examples in the training data, empirical risk minimization will produce a hypothesis whose expected risk is close to the smallest expected risk attainable. However, the rate of convergence may be painfully slow, and consequently, the amount of necessary training data may be impractically large. The relationship between empirical and expected risks is one of the primary questions addressed by computational/statistical learning theory [Vapnik, 1999].

2.3 Neural networks

An artificial neural network is composed of layers of nodes, each of which plays the role of a neuron in the human brain. Although some neural networks are also called "deep" neural networks, the word "deep", which is usually interpreted as having multiple layers of nodes, is not rigorously defined in the literature. The technique of using deep neural networks to solve machine learning tasks is known as deep learning [LeCun et al., 2015].

In this document, we will focus on feedforward neural networks, in which information only flows in one direction (hence "feedforward"). Other architectures of neural networks include recurrent neural networks, which process sequences of input data.

2.3.1 Feedforward neural networks

Definition Suppose that an instance space is $\mathcal{X}=\mathbb{R}^{d_0}$ and the set of outputs is $\mathcal{Y}=\{1,\dots,k\}$. A "vanilla" feedforward neural network is a function $F:\mathcal{X}\to\mathcal{Y}$ of the form

$$F:=F_n\circ\cdots\circ F_1,$$

where each $F_i:\mathbb{R}^{d_{i-1}}\to\mathbb{R}^{d_i}$ for $1\le i\le n$ is

$$F_i(x):=\rho_i(A_ix+b_i).$$

Here, both $A_i\in\mathbb{R}^{d_i\times d_{i-1}}$ and $b_i\in\mathbb{R}^{d_i}$ store trainable/learnable parameters, and each ρ_i is a non-linear activation function. For consistency, we should have $k=d_n$.

Activation functions Non-linear activation functions are crucial as they provide non-linearity to neural networks, thereby enabling neural networks to express a wide range of non-linear functions in addition to linear functions. Four common activation functions are

- tanh: $x\mapsto\frac{e^x-e^{-x}}{e^x+e^{-x}}$
- Sigmoid: $x\mapsto\frac{1}{1+e^{-x}}$

- Rectified linear unit (ReLU): $x \mapsto \max(x, 0)$
- Exponential linear unit (ELU):

$$x \mapsto \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{otherwise,} \end{cases}$$

  where α is a user-specified hyper-parameter and is usually 1 by default.

These activation functions are applied to an input vector in a pointwise fashion. For the last function F_n, the activation function is usually the softmax function, which, given $x \in \mathbb{R}^k$, returns $y \in \mathbb{R}^k$ such that

$$y_i := \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}.$$

This function squeezes each x_i into $y_i \in [0, 1]$ in such a way that $\sum_{j=0}^{k} y_j = 1$ and $\arg\max_{1 \leq i \leq k} x_i = \arg\max_{1 \leq i \leq k} y_i$. Thus, y_i can be interpreted as the probability (also called confidence) that the correct class label is i.

Beyond vanilla feedforward neural networks The feedforward neural networks used in practice are more involved than the vanilla feedforward neural network described above. For instance, various types of layers have been invented and used in practice; e.g. convolutional layers and dropout layers. In Chapter 5, we will use convolutional layers for experiments, but it is not necessary to understand the details of convolutional layers. Hence, I will omit an explanation on convolutional layers.

2.3.2 Training by backpropagation

Cross entropy loss function Training a neural network amounts to determining trainable parameters such that an empirical loss of the neural network is (approximately) minimized. For computer vision, a common loss function is the cross entropy loss function:

$$\ell(\theta, x, t) := -\ln(F_\theta(x)_t),$$

where x is an instance, t is the correct class label, and $F_\theta(x)_t$ is the confidence for class t in the output of the neural network F_θ.

Backpropagation Suppose we are given a training dataset $\{(x_i, y_i) \mid 1 \leq i \leq m\}$ and we are to use a loss function ℓ. Training of neural networks is done by the backpropagation algorithm [LeCun et al., 2015], which calculates the gradient $\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} \ell(\theta, x_i, y_i)$. Then θ is adjusted by making one step in the direction of the loss's steepest descent.

In theory, it is expected to calculate a gradient using all instances in a training dataset. To speed up the computation of gradients, GPUs, which are capable of efficiently handling matrices, are employed. However, since it is impractical to fit an entire dataset

in the memory of a GPU, the dataset is split into batches whose size is usually 128, 256, or 512. This split is usually done randomly in the training phase. Once the dataset is split into batches, we visit them sequentially, using each batch to calculate a gradient and updating the weights of the neural network accordingly. When we have gone through all batches, we divide the dataset randomly into batches again and repeat the same process. Each round of traversing the entire dataset is called an epoch.

Optimizers A component within the backpropagation algorithm responsible for updating the neural network's weights based on a gradient is called an optimizer. Amongst popular optimizers for deep learning are stochastic gradient descent (SGD), RMSprop [Tieleman and Hinton, 2012], and Adam [Kingma and Ba, 2014].

2.4 Adversarial attack and training

2.4.1 Adversarial attacks

Adversarial examples and perturbations can be informally defined as follows.

Definition 2.4.1 (Adversarial examples and perturbations)**.** *Let $F : \mathcal{X} \to \{1, \ldots, k\}$ be a classifier and let $c : \mathcal{X} \to \mathcal{Y}$ be a target concept to be learned. An adversarial example is $x + \delta \in \mathcal{X}$ that is obtained by applying an adversarial perturbation δ to an original example x, where $(x, c(x)) \sim \mathcal{D}$ and $F(x) = c(x)$, such that $F(x + \delta) \neq c(x)$.*

It is unnecessary to consider the case of $F(x) \neq c(x)$ (i.e. the prediction result of F is already wrong even without a perturbation), because we are concerned with the robustness of a neural network only when its accuracy is high.

In most cases, additional requirements are imposed on the above definition of adversarial examples. One common constraint is an upper limit on the size of perturbations δ. This is called an "adversarial budget" and denoted by ϵ. In order for a perturbation δ to be adversarial, we expect δ to be relatively small; otherwise, if we were allowed to inject as much perturbation as we wish, any reasonable machine learning model could be tricked and hence would never be robust. However, an upper limit on the size of δ is determined by users and hence is problem dependent.

Regarding distance functions that we use to measure the size of perturbations, common distances in computer vision include the L_2 and L_∞ distances. However, none of the distance functions that have been used in computer vision faithfully reflects the human perception of similarity between images [Carlini and Wagner, 2017].

Another constraint is the target class of adversarial attacks; i.e. $F(x+\delta)$. If a target class is specified, the attack is called targeted; otherwise, it is said to be untargeted.

The literature has a large body of knowledge on approaches to finding adversarial examples [Akhtar and Mian, 2018], and some of them are presented in Appendix B.1.

2.4.2 Adversarial training

A family of defenses called adversarial training incorporates adversarial examples into training data, thereby improving neural networks' classification accuracy on adversarial examples. Although we know that adversarial training increases the robustness of deep neural networks, it remains unknown to what the extent adversarial training is effective. Many of early research results on adversarial training are focused on specific training schemes and tend to be of experimental nature rather than theoretical nature. Therefore, it is necessary to develop a theoretical framework in which we can investigate adversarial training rigorously and holistically (as in computational learning theory).

Recognizing this need, Madry et al. [2017] suggest robust optimization (RO) as a theoretical framework for adversarial training. The use of RO in the setting of deep neural networks first appeared in [Shaham et al., 2015; Huang et al., 2015; Lyu et al., 2015].

As adversarial training is not strictly defined, it encompasses a broad array of algorithms to train neural networks, whilst the RO framework for adversarial training is rigorously defined. Therefore, not all adversarial training algorithms can be faithfully described in the RO framework. Nonetheless, RO is instrumental in providing a theoretical model of adversarial training that captures the intent of adversarial training. Therefore, this situation is analogous to statistical learning theory, where we have a number of different learning models such as PAC learning and statistical query learning, all of which capture the main intent of machine learning whilst using different assumptions.

I now introduce adversarial training in the form of robust optimization using the presentation style of Madry et al. [2017]. Consider a set $\mathcal{S}$ of possible perturbations around any instance $x \in \mathcal{X}$. Adversarial training can be viewed as the following robust optimization problem:

$$\min_{\theta} \rho(\theta), \tag{2.4.1}$$

where $\rho(\theta)$ (adversarial loss) is given by

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} \ell(\theta, x+\delta, y) \right]. \tag{2.4.2}$$

An optimization problem of the shape (2.4.1) is called a saddle-point problem because it has a maximization problem embedded inside a minimization problem [Madry et al., 2017].

In (2.4.1), the inner maximization problem is interpreted as computing adversarial examples with respect to parameter θ of a neural network. The outer minimization problem seeks to find θ such that $\rho(\theta)$ is minimized.

In practice, adversarial training proceeds by first computing the (approximately) worst adversarial example within $\mathcal{S}$ around some x. This step can be achieved using, for example, FGSM and PGD (See Appendix B.1). Using the resulting adversarial examples, we run the backpropagation algorithm to adjust θ such that the neural network classifies these adversarial examples correctly with high confidence. A mathematical justification behind this workflow of adversarial training is discussed in Appendix B.2.

Chapter 3

Distributionally robust optimization (DRO)

3.1 Formalization

In distributionally robust optimization (DRO), entire probability distributions, instead of individual datapoints, are perturbed. Whilst robust optimization (RO) in neural networks is primarily motivated by adversarial attacks, distributionally robust optimization is motivated by not only adversarial attacks but also changes in data distribution [Sinha et al., 2018]. In real-world applications of deep learning, the probability distribution generating training data may not coincide with the probability distribution that generates data for testing or prediction.

We will work with the Wasserstein distance as a distance between probability distributions. I will follow the presentation style of Staib and Jegelka [2017]. Let $d(\cdot,\cdot) : Z \times Z \to \mathbb{R}$ be a metric/distance on some metric space Z (See Appendix A.1 for precise definitions of metric spaces and distances). Using this metric, we define the p-Wasserstein distance as follows:

Definition 3.1.1 (p-Wasserstein distance)**.** *The p-Wasserstein distance is defined as*

$$W_p(D, \tilde{D}) := \inf \left\{ \int d(z, \tilde{z})^p \, \mathrm{d}\gamma(z, \tilde{z}) \mid \gamma \in \Pi(D, \tilde{D}) \right\}^{\frac{1}{p}},$$

where $\Pi(D, \tilde{D})$ is the set of couplings of probability distributions D and $\tilde{D}$ over Z.

Villani [2003] provides an alternative definition:

Definition 3.1.2 (Wasserstein distance)**.** *Given a cost function $c : Z \times Z \to \mathbb{R}$, the Wasserstein distance is defined as*

$$W(D, \tilde{D}) := \inf \left\{ \int c(z, \tilde{z}) \, \mathrm{d}\gamma(z, \tilde{z}) \mid \gamma \in \Pi(D, \tilde{D}) \right\},$$

where $\Pi(D, \tilde{D})$ is the set of couplings of probability distributions D and $\tilde{D}$ over Z. The cost function c does not need to be a metric; however, c is required to be measurable and non-negative [Villani, 2003]. Also, since c is sometimes allowed to return ∞, its codomain can be $\mathbb{R}^{\geq 0} \cup \{\infty\}$.

There is no need to understand what it means for a function to be measurable in Definition 3.1.2—it suffices to remember that measurable functions are not necessarily metrics/distances.

The above two definitions of the Wasserstein distance are equivalent because $W_p(\mathcal{D}, \tilde{\mathcal{D}})$ (according Definition 3.1.1) is equal to the $W^{\frac{1}{p}}(\mathcal{D}, \tilde{\mathcal{D}})$ (in Definition 3.1.2), where the cost function is $d^p(\cdot, \cdot)$. To distinguish between the two definitions, I will write W_p for the first definition and W_1 for the second one.

In [Villani, 2008], couplings of two probability distributions are defined as follows.

Definition 3.1.3 (Coupling)**.** *Let (Ω_1, μ) and (Ω_2, ν) be two probability spaces, where Ω_1 and Ω_2 are outcome spaces and μ and ν are probability measures. Coupling μ and ν means constructing two random variables X and Y on probability space $(\Omega, \mathbb{P})$ such that law$(X) = \mu$ and law$(Y) = \nu$. Ω is commonly set to $\Omega_1 \times \Omega_2$.*

Also, a coupling of two random variables A and B is a pair (X, Y) of random variables on the same outcome space such that the probability distributions of X and Y are identical to those of A and B, respectively. Hence, the joint probability distribution of a coupling specifies how to "transport" the probability distribution of X to that of Y. The Wasserstein distance therefore represents the minimum cost of transporting $\mathcal{D}$ to $\tilde{\mathcal{D}}$.

The goal of DRO is to find parameter θ of a deep neural network that satisfy the following:

$$\min_{\theta} \max_{\tilde{\mathcal{D}}: W_p(\mathcal{D}, \tilde{\mathcal{D}}) \leq \epsilon} \mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}} [\ell(\theta, x, y)]. \tag{3.1.1}$$

The metric $d_{\mathcal{X},\mathcal{Y}}$ on $\mathcal{X} \times \mathcal{Y}$ that is used to define W_p is given by

$$d_{\mathcal{X},\mathcal{Y}}((x, y), (\tilde{x}, \tilde{y})) := d(x, \tilde{x}) + \infty \cdot \mathbb{1}_{y \neq \tilde{y}},$$

where the base metric d is chosen by the user. The distance $d_{\mathcal{X},\mathcal{Y}}$ imposes an infinite penalty if $y \neq \tilde{y}$ because we assume that the adversary can only perturb instances and not labels [Staib and Jegelka, 2017; Sinha et al., 2018]. Consequently, if $W_p(\mathcal{D}, \tilde{\mathcal{D}}) < \infty$, any coupling $\pi \in \Pi(\mathcal{D}, \tilde{\mathcal{D}})$ with the optimal cost is not allowed to assign a positive probability to the transport from (x, y) to $(\tilde{x}, \tilde{y})$, where $y \neq \tilde{y}$.

One advantage of DRO over robust optimization (RO) is that DRO encompasses RO. Hence, if a neural network can be trained to be robust (i.e. the distributional adversarial loss is small) in the DRO framework, it automatically implies that the neural network also has a small adversarial loss in RO. This is made precise by the following theorem:

Lemma 3.1.1 (Staib and Jegelka [2017])**.** *Fix parameters θ and a metric d on $\mathcal{X}$. For any $p \in [1, \infty]$, we have*

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\max_{\tilde{x} \sim B_\epsilon(x)} \ell(\theta \tilde{x}, y) \right] \leq \max_{\tilde{\mathcal{D}}: W_p(\mathcal{D}, \tilde{\mathcal{D}}) \leq \epsilon} \mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}} [\ell(\theta, x, y)], \tag{3.1.2}$$

where $B_\epsilon(x) := \{\tilde{x} \mid d(x, \tilde{x}) \leq \epsilon\}$. Equality holds for $p = \infty$.

In (3.1.2), when $p = \infty$, the Wasserstein distance $W_\infty(\mathcal{D}, \tilde{\mathcal{D}})$ represents the the largest distance (measured using d) between any $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. This is analogous to the L_∞-distance of vectors, which measures the larges component-wise distance. This is why equality holds in in (3.1.2) when $p = \infty$.

Another related advantage of DRO is that DRO does not assume that all perturbations are equally likely, whilst RO does. This is because we impose an upper bound on the total perturbation across entire probability distributions instead of individual point-wise perturbations. However, this means that DRO considers distribution-wise perturbations in which individual point-wise perturbations are unrealistically large. This should not be a serious problem, since if some point-wise perturbation is large, its probability should be relatively low (such that the total distribution-wise perturbation remains bounded by ϵ). Consequently, as long as $\ell(\theta, \cdot, y)$ is reasonably proportional to the distance measured by d, we should not have an issue.

According to Staib and Jegelka [2017], another (practical) advantage of DRO over RO is that it is unknown how well RO generalizes (i.e. the extent to which a small empirical adversarial loss in RO will translate to a small expected adversarial loss), whereas the theoretical foundation of DRO is known better. The generalization guarantee of DRO with the Wasserstein distance is studied in [Blanchet et al., 2016; Gao and Kleywegt, 2016; Mohajerin Esfahani and Kuhn, 2018]. Personally, since DRO subsumes RO, I wonder why we cannot use the generalization guarantee of DRO to derive one for RO.

3.2 Approaches to solving distributionally robust optimization

3.2.1 Introduction

Distributionally robust optimization (DRO) with W_p is formulated as (3.1.1). Let OPT denote the inner maximization problem of (3.1.1):

$$\text{OPT} := \max_{\tilde{\mathcal{D}}: W_p(\mathcal{D}, \tilde{\mathcal{D}}) \leq \epsilon} \mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}} [\ell(\theta, x, y)]. \tag{3.2.1}$$

Let us fix θ so that we can focus on the inner maximization problem. In (3.2.1), the true instance distribution $\mathcal{D}$ is unknown. Thus, at best we can only (approximately) solve the following new inner problem:

$$\text{OPT}_{\text{empirical}} := \max_{\tilde{\mathcal{D}}: W_p(\mathcal{D}_n, \tilde{\mathcal{D}}) \leq \epsilon} \mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}} [\ell(\theta, x, y)],$$

where $\mathcal{D}_n$ is an empirical distribution; i.e. an instance distribution constructed from the training dataset. As suggested in [Staib and Jegelka, 2017], this new optimization problem can be approximately solved by considering

$$\begin{aligned} \text{MIX} := \max_{\tilde{x}_1, \ldots, \tilde{x}_n} \; & \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, \tilde{x}_i, y_i) \\ \text{subject to } & \frac{1}{n} \sum_{i=1}^{n} d^p(\tilde{x}_i, x_i) \leq \epsilon^p. \end{aligned} \tag{3.2.2}$$

Note that (3.2.2) perturbs $\mathcal{D}_n$ by moving individual examples such that the total transport cost is bounded above by ϵ. Hence, the probability distribution considered in (3.2.2) has the Wasserstein distance at most ϵ from $\text{OPT}_{\text{empirical}}$. This implies $\text{MIX} \leq \text{OPT}_{\text{empirical}}$.

Additionally, the following theorem, which is originally Corollary 2 (iv) of Gao and Kleywegt [2016] and is subsequently simplified by Staib and Jegelka [2017], establishes another connection between MIX and $\text{OPT}_{\text{empirical}}$:

Theorem 3.2.1 (Gao and Kleywegt [2016] and Staib and Jegelka [2017])**.** *Fix θ and suppose that $\ell(\theta, \cdot, y)$ is an L-Lipschitz function for all y. Then we have $OPT \leq MIX + \frac{LD}{n}$, where D bounds the maximum deviation of a single point.*

In summary, we obtain

$$\text{OPT}_{\text{empirical}} - \frac{LD}{n} \leq \text{MIX} \leq \text{OPT}_{\text{empirical}}.$$

This gives a bound on $|\text{MIX} - \text{OPT}_{\text{empirical}}|$. The convergence of the distance between OPT and $\text{OPT}_{\text{empirical}}$ is discussed in Appendix C.1.

3.2.2 Distributional projected gradient descent

If the distance function for $\mathcal{X}$ is $d = L_q$, the optimization problem (3.2.2) can be expressed as

$$\begin{aligned} &\max_{\tilde{X}} \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, \tilde{x}_i, y_i) \\ &\text{subject to } \left\| \tilde{X} - X \right\|_{q,p} \leq n^{\frac{1}{p}} \epsilon, \end{aligned} \tag{3.2.3}$$

where X and $\tilde{X}$ are matrices in which each column is a datapoint. The matrix norm $L_{q,p}$ of $A \in \mathbb{R}^{d \times n}$ is defined as

$$\|A\|_{q,p} := \left(\sum_{j=1}^{n} \left(\sum_{i=1}^{d} \|A_{i,j}\|^q \right)^{\frac{p}{q}} \right)^{\frac{1}{p}}.$$

The constraint in (3.2.3) is called a mixed-norm ball since it contains two different norms.

One plausible approach to solving this optimization problem is projected gradient descent (PGD), which in each iteration projects an intermediate solution to the feasible region specified by the constraint. For details on PGD, the reader is referred to Appendix A.2.2.

To apply PGD, it is necessary to compute the projection of $\tilde{X}$ on the feasible region specified by $\left\| \tilde{X} - X \right\|_{q,p} \leq n^{\frac{1}{p}} \epsilon$. The feasible region is rather conceptually simple—it is an $L_{q,p}$-ball. However, the projection operator used in PGD is defined using the

Euclidean distance (equivalently, the L_2 or $L_{2,2}$ norm), and there is no general and straightforward way to compute Euclidean projection on an $L_{q,p}$-ball. Research results on efficient projection algorithms are available for some special cases; e.g. when $p = 1$ and $q \geq 1$ [Sra, 2012]. Also, if we have $p = q$, the $L_{p,q}$ norm is reduced to the vector norm L_p. When $p = q = 2$ or $p = q = \infty$, projection is trivial. When $p = q = 1$, the algorithm by Duchi et al. [2008] can be employed.

In order to avoid confusion, I will henceforth refer to this DRO algorithm as "distributional" projected gradient descent (PGD), distinguishing this from PGD attacks.

3.2.3 Frank-Wolfe method with stochastic block coordinate descent

Staib and Jegelka [2017] suggest using the Frank-Wolfe method together with stochastic block coordinate descent (BCD) to approximately solve DRO. The Frank-Wolfe method for convex optimization is explained in Appendix A.2.4, and coordinate descent is explained in Appendix A.2.5. For brevity, this Frank-Wolfe method based approach to solving DRO is abbreviated to FWDRO or even FW.

In this approach, we start with an initial value at which the $L_{q,p}$ constraint is tight. In the t-th iteration, we first randomly choose a block of coordinates (specifically, multiple columns of $\tilde{X}$ in this case). Let X denote the matrix composed of training examples and $\tilde{X}$ denote the matrix containing all variables to be optimized. Further, I write X_S and $\tilde{X}_S$ for the projection of the corresponding matrices on the columns we have selected. For the projection of matrices on those columns that were not selected, I use the subscript of N, which is short for "not selected". The original constraint on $\tilde{X}$ is $\left\|\tilde{X} - X\right\|_{q,p}^p \leq n\epsilon^p$. It is easy to see

$$\left\|\tilde{X} - X\right\|_{q,p}^p = \left\|\tilde{X}_S - X_S\right\|_{q,p}^p + \left\|\tilde{X}_N - X_N\right\|_{q,p}^p.$$

Therefore, the (local) constraint imposed on $\tilde{X}_S$ is

$$\left\|\tilde{X}_S - X_S\right\|_{q,p}^p \leq n\epsilon^p - \left\|\tilde{X}_N - X_N\right\|_{q,p}^p. \tag{3.2.4}$$

The Frank-Wolfe method is then used to update the variables in $\tilde{X}_S$ by solving the following linear optimization problems in terms of Y:

$$\begin{aligned} \text{maximize} \quad & \left(\nabla_{\tilde{X}_S}\left(\frac{1}{n}\sum_{i=1}^{n}\ell(\theta, \tilde{x}_i, y_i)\right)\right)^\intercal Y \\ \text{subject to} \quad & \|Y - X_S\|_{q,p}^p \leq n\epsilon^p - \left\|\tilde{X}_N - X_N\right\|_{q,p}^p. \end{aligned}$$

In the objective function above, we need to flatten the two matrices before taking their inner product. This optimization problem can be cast into

$$\begin{aligned} \text{maximize} \quad & C^\intercal Y \\ \text{subject to} \quad & \|Y - D\|_{q,p} \leq \delta, \end{aligned}$$

where C, D, and δ take appropriate constants. Introducing a dummy variable $Z := Y - D$, we can express the above optimization problem as

$$\begin{aligned} \text{maximize} \quad & C^\intercal (Z + D) \\ \text{subject to} \quad & \|Z\|_{q,p} \leq \delta, \end{aligned}$$

which can be transformed into

$$\begin{aligned} \text{maximize} \quad & C^\intercal Z \\ \text{subject to} \quad & \|Z\|_{q,p} \leq \delta, \end{aligned}$$

because D is a constant. Note that the optimal solution Z to this problem is not necessarily a scalar product of vector C. For example, in an L_∞-ball in the two-dimensional space, the optimal solution to the above problem is usually a vector from the origin to one of the corners of the square; hence, it does not always have the same direction as C. Details on how to solve this optimization problem are provided in Appendix B of [Staib and Jegelka, 2017].

The pseudocode for FWDRO is in Algorithm 1.

Algorithm 1 Distributionally robust optimization by the FW method and stochastic BCD

Require: Dataset $\{(x_i, y_i)\}_{i=1}^n$, batch size k, budget ϵ, step-size sequence $(\alpha_t > 0)_{t=0}^{T-1}$

1: Initialise $\epsilon_i \leftarrow \epsilon$ for all $i = 1, \ldots, n$
2: **for** $t = 0, \ldots, T-1$ **do**
3: Randomly draw a subset $S \subseteq \{1, \ldots, n\}$ of size k
4: Compute the budget for the local constraint $\epsilon_S := \|X_S\|_{q,p}$ using (3.2.4) and ϵ_i for $i \notin S$
5: To compute $\tilde{X}_S$, use the Frank-Wolfe method to solve

$$\begin{aligned} \max_{\tilde{X}_S} \quad & \frac{1}{k} \sum_{i \in S} \ell(\theta, \tilde{x}_i, y_i) \\ \text{subject to} \quad & \left\| \tilde{X}_S - X_S \right\|_{q,p} \leq \epsilon_S \end{aligned}$$

▷ Apply clipping at the end of each iteration in the FW method if necessary
6: $\epsilon_i \leftarrow \|\tilde{x}_i\|_q$ for all $i \in S$
7: $\theta \leftarrow \theta - \alpha_t \nabla_\theta \left(\frac{1}{k} \sum_{i \in S} \ell(\theta, \tilde{x}_i, y_i) \right)$

In this algorithm, we do not store the adversarial examples computed in line 5 for future use. Instead, we only store the amount of perturbation, namely ϵ_i, for each $i \in S$. This is because we can use ϵ_i to calculate $\left\| \tilde{X}_N - X_N \right\|_{q,p}^p$ in (3.2.4). Hence, it is unnecessary to store the entire $\tilde{X}$ for the purpose of computing $\tilde{X}_N$.

The values of ϵ_i are initialized to ϵ (as opposed to 0, for instance) for all $1 \leq i \leq n$ in line 1. This is because in BCD we start with a point where the constraint is tight, and this can be attained by setting $\epsilon_i = \epsilon$ for all $1 \leq i \leq n$.

In my implementation of FWDRO, to further reduce memory usage, I use ϵ instead of ϵ_S in line 4 in every iteration.

3.2.4 Lagrangian relaxation

Sinha et al. [2018] use Lagrangian relaxation to approximately solve distributionally robust optimization (DRO) and term this approach WRM (it is probably short for Wasserstein Robust Method). In their paper, they use Definition 3.1.2 in lieu of Definition 3.1.1 for the Wasserstein distance. Basics of general Lagrangian relaxation are available in Appendix A.3.1.

Recall that DRO can be formulated as

$$\min_{\theta\in\Theta}\max_{\tilde{\mathcal{D}}:W_1(\tilde{\mathcal{D}},\mathcal{D})}\mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}}[\ell(\theta,x,y)],$$

where $\mathcal{D}$ is a fixed instance distribution (e.g. empirical instance distribution) and Θ is the set of possible model parameters θ. This expression is obtained by replacing W_p in (3.1.1) with W_1 (in Definition 3.1.2).

By Lagrangian relaxation, the inner problem is transformed into

$$\sup_{\tilde{\mathcal{D}}}\left[\mathop{\mathbb{E}}_{(x,y)\in\tilde{\mathcal{D}}}[\ell(\theta,x,y)]-\lambda W_1(\tilde{\mathcal{D}},\mathcal{D})\right],$$

where $\gamma\geq 0$ is fixed. Note that since the inner problem is a maximization problem, the penalty term in the above Lagrangian is negated, unlike in Section A.3.1, where the objective was minimization.

The following result (originally proved by Blanchet and Murthy [2016], and an alternative proof is provided by Sinha et al. [2018]) establishes strong duality.

Theorem 3.2.2 (Proposition 1 in [Sinha et al., 2018] and Theorem 1 in [Blanchet and Murthy, 2016])**.** *Suppose loss function $\ell:\Theta\times\mathcal{X}\times\mathcal{Y}\to\mathbb{R}$ and cost function $c:(\mathcal{X}\times\mathcal{Y})\times(\mathcal{X}\times\mathcal{Y})\to\mathbb{R}^{\geq 0}\cup\{\infty\}$ are continuous. Then for any distribution Q and any $\epsilon>0$, we have*

$$\sup_{P:W_1(P,Q)\leq\epsilon}\mathop{\mathbb{E}}_{(x,y)\sim P}[\ell(\theta,x,y)]=\inf_{\lambda\geq 0}\left[\lambda\epsilon+\mathop{\mathbb{E}}_{(x,y)\sim Q}[\phi_\lambda(\theta,x,y)]\right], \tag{3.2.5}$$

where

$$\phi_\lambda(\theta,x,y):=\sup_{(x',y')\in\mathcal{X}\times\mathcal{Y}}\left[\ell(\theta,x',y')-\lambda c((x,y),(x',y'))\right]. \tag{3.2.6}$$

This is called a robust surrogate. Furthermore, for any $\lambda\geq 0$, we have

$$\sup_{P}\left[\mathop{\mathbb{E}}_{(x,y)\sim P}[\ell(\theta,x,y)]-\lambda W_1(P,Q)\right]=\mathop{\mathbb{E}}_{(x,y)\sim Q}[\phi_\lambda(\theta,x,y)]. \tag{3.2.7}$$

In (3.2.5), the left hand side is a constrained optimization problem, and the right hand side can be rewritten as

$$\inf_{\lambda \geq 0} \left[\sup_{P} \mathop{\mathbb{E}}_{(x,y)\sim P} [\ell(\theta, x, y)] - \lambda(W_1(P, Q) - \epsilon) \right],$$

where $W_1(P, Q) - \epsilon \leq 0$ is the constraint of DRO. Hence, the left hand side of (3.2.5) is in accord with the definition of Lagrangian duality given in Section A.3.1.

(3.2.7) is useful because on the left hand side is the relaxed inner maximization problem of (3.1.1), which takes a supremum over probability distributions, and on the left hand side is an expression that takes a supremum over individual points rather than over probability distributions. Consequently, by (3.2.7), the Lagrangian relaxation of DRO by Sinha et al. [2018] becomes

$$\min_{\theta \in \Theta} \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}} [\phi_\lambda(\theta, x, y)],$$

or equivalently

$$\min_{\theta \in \Theta} \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}} \left[\sup_{(x',y')} \left[\ell(\theta, x', y') - \lambda c((x, y), (x', y')) \right] \right] \tag{3.2.8}$$

by the definition of the robust surrogate in (3.2.6). We can notice that (3.2.8) closely resembles

$$\min_{\theta \in \Theta} \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}} \left[\max_{\tilde{x} \in B_\epsilon(x)} \ell(\theta, \tilde{x}, y) \right],$$

that is, robust optimization (RO) for adversarial training. In fact, (3.2.8) is a Lagrangian relaxation of RO as well as a Lagrangian relaxation of DRO.

(3.2.8) is then solved by stochastic (projected) gradient descent (GD) in [Sinha et al., 2018]. In their paper, Sinha et al. [2018] also provide a theoretical result on the convergence of stochastic (projected) GD under several assumptions on ℓ and c. Two of the major assumptions are

1. ℓ is smooth (See Definition A.2.2) in the first argument. For this reason, they recommend the use of smooth loss functions when their approach to DRO is used. Additionally, this is why they use the exponential linear unit (ELU) in place of the rectified linear unit (ReLU) in their experiments.

2. γ is sufficiently large. Due to the duality result in Theorem 3.2.2, larger γ means smaller ϵ (i.e. adversarial budget). This inverse relationship between γ and ϵ is explained in more detail in Section C.2.

The pseudocode of WRM is in Algorithm 2.

Algorithm 2 Distributionally robust optimization by Lagrangian relaxation and SGD

Require: Training data distribution $\mathcal{D}_0$, penalty parameter λ, parameter space Θ, step-size sequence $\{\alpha_t > 0\}_{t=0}^{T-1}$

for $t = 0, \ldots, T-1$ **do**

Sample $(x, y) \sim \mathcal{D}_0$ and find an approximate maximizer $\tilde{x}_t$ of $\ell(\theta, \tilde{x}_t, y) - \lambda d(x, \tilde{x}_t)$

$\theta_{t+1} \leftarrow \mathsf{Proj}_\Theta(\theta_t - \alpha_t \nabla_\theta \ell(\theta_t, \tilde{x}_t, y))$ ▷ Apply clipping if necessary

3.3 Issues with the previous research work on DRO

I have discovered that the previous research work on DRO by Staib and Jegelka [2017] and Sinha et al. [2018] suffers from the following issues:

1. In the typical formulation of DRO, users have a choice over a loss function. However, in many experimental studies, success/error rates on defenses against adversarial examples are used to measure performance of adversarial training procedures. This suggests that instead of allowing users to select loss functions by themselves, we should fix the loss function used in DRO's formulation to be the zero-one loss function:

$$\ell_{0,1}(\theta, x, y) := \mathbb{1}_{F_\theta(x) \neq y}, \tag{3.3.1}$$

where F_θ is a neural network with parameter θ.

However, to train neural networks using the bakpropagation algorithm, it is necessary to use loss functions that are smoother than $\ell_{0,1}$. This raises the question of what loss functions gives the highest level of robustness in DRO where the robustness is evaluated using $\ell_{0,1}$. This will be one of the main goals of the empirical studies in Chapter 5.

2. DRO is more suitable than RO when we impose a limit on the total perturbation on the entire probability distribution rather than the perturbation of individual points. However, in many empirical studies comparing adversarial training procedures (including RO and DRO), adversarial attacks are launched on individual points rather than on the entire probability distributions. For instance, in [Staib and Jegelka, 2017], although the authors are aware that DRO and RO make different assumptions on the nature of adversaries, they only use pointwise adversarial attacks to evaluate DRO algorithms.

Appendix D.1 addresses this issue, showing the experiment results of the use of distributional PGD as a distributional adversarial attack (instead of as a DRO algorithm). However, this approach suffers a significantly lower attack success rate than pointwise PGD for an unknown reason. Hence, this needs further investigation in future work.

Chapter 4

Tuning of loss functions

Carlini and Wagner [2017] develop an adversarial attack that successfully breaks defensive distillation, a defense against adversarial perturbations proposed by Papernot et al. [2016b]. As one of the strongest adversarial attacks to date, the attack developed by Carlini and Wagner [2017] is commonly used as a benchmark [Madry et al., 2017].

This chapter describes the research work of Carlini and Wagner [2017] and discusses the theoretical connection between their formulation of adversarial attacks and the formulation of distributionally robust optimization (DRO).

4.1 Adversarial attacks with tuned loss functions

Given a neural network whose last layer is the softmax function, let $F : \mathcal{X} \to \mathbb{R}^n$ and $Z : \mathcal{X} \to \mathbb{R}^n$, where $n = |\mathcal{Y}|$, represent the neural network with and without the softmax function, respectively. In addition, define $C : \mathcal{X} \to \mathcal{Y}$ as $C(x) := \arg\max_{y \in \mathcal{Y}} F_y(x)$, where $F_i(x)$ denotes the i-th confidence value in $F(x)$.

Carlini and Wagner [2017] consider the following problem of crafting targeted adversarial examples:

$$\begin{aligned} \text{minimize} \quad & d(x, x+\delta) \\ \text{such that} \quad & C(x+\delta) = t \\ & x + \delta \in \mathcal{X}. \end{aligned} \tag{4.1.1}$$

Here, $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a user-specified distance metric, $C : \mathcal{X} \to \mathcal{Y}$ is a classifier, and t is the target class of an adversarial example.

If we introduce a function f such that $C(x+\delta) = t \iff f(x+\delta) \leq 0$, (4.1.1) can be equivalently expressed as

$$\begin{aligned} \text{minimize} \quad & d(x, x+\delta) \\ \text{such that} \quad & f(x+\delta) \leq 0 \\ & x + \delta \in \mathcal{X}. \end{aligned} \tag{4.1.2}$$

(4.1.2) is then converted to

$$\begin{aligned} &\text{minimize} && d(x, x+\delta) + cf(x+\delta) \\ &\text{such that} && x+\delta \in \mathcal{X}, \end{aligned} \tag{4.1.3}$$

where $c > 0$ is a fixed penalty parameter that will be chosen appropriately.

We can make two remarks at this point:

1. The formulation of adversarial attacks in (4.1.3) can be considered as either the penalty method (See Appendix A.3.2) or Lagrangian relaxation, depending on the property of f. If $f(x+\delta) = 0$ holds whenever $C(x+\delta) = t$, (4.1.3) is the penalty-method formulation derived form (4.1.1) (i.e. the original problem formulation of crafting adversarial attacks). Conversely, if $f(x+\delta) = 0 \iff C(x+\delta) = t$ does not hold, (4.1.3) is regarded as the Lagrangian relaxation of (4.1.2), which is equivalent to (4.1.1).

2. Another important observation is that (4.1.3) can be seen as not only the Lagrangian relaxation of (4.1.2) but also the Lagrangian relaxation of

$$\max_{\delta \in \mathcal{X}} \ell(\theta, x+\delta, y),$$

which appears inside (2.4.2). Here, $\mathcal{X}$ is a ball around x with the distance function d, and we have $f(\cdot) := \ell(\theta, \cdot, y)$, which is a partially applied loss function. This means that in a sense (4.1.3) serves as a bridge connecting (4.1.2) and (2.4.2), which are two distinct formulations of adversarial attacks.

In order to choose appropriate c, by viewing (4.1.3) as the penalty method, we can test multiple values for c until we obtain an optimal solution x^* that satisfies $f(x^*) \leq 0$. Although this is a simple strategy, it is not guaranteed that such c exists (See Appendix A.3.2).

Carlini and Wagner [2017] observe that it empirically works well to choose the smallest c whose optimal solution satisfies $f(x+\delta^*) = 0$. Presumably, they have always succeeded in eventually finding c such that the resulting optimal solution δ^* satisfies $f(x+\delta^*) = 0$. The reason why it is generally sensible to choose a small c is that large c can drastically distort the landscape of the search space by creating steep hills, thereby making it more difficult for numerical optimization algorithms to find accurate approximations to the true optimal solutions.

In [Carlini and Wagner, 2017], the metrics L_0, L_2, and L_∞ are specifically considered. Different candidates for f are examined, and the best one is selected for each of the L_0, L_2, and L_∞ distances. In the present project, I will focus on the L_2 distance.

4.2 Loss functions

Carlini and Wagner [2017] examine the following seven loss functions:

$$
\begin{aligned}
f_1(x) &:= \text{loss}_{F,t}(x) - 1 \\
f_2(x) &:= \left(\max_{i \neq t} F_i(x) - F_t(x)\right)^+ \\
f_3(x) &:= \text{softplus}\left(\max_{i \neq t} F_i(x) - F_t(x)\right) - \ln 2 \\
f_4(x) &:= (0.5 - F_t(x))^+ \\
f_5(x) &:= \ln(2 - 2F_t(x)) \\
f_6(x) &:= \left(\max_{i \neq t} Z_i(x) - Z_t(x)\right)^+ \\
f_7(x) &:= \text{softplus}\left(\max_{i \neq t} Z_i(x) - Z_t(x)\right) - \ln 2.
\end{aligned}
$$

Here, $(x)^+$ is short for $\max(x, 0)$, which is equivalent to ReLU, and $\text{softplus}(x) := \ln(1 + e^x)$, which resembles ReLU. $\text{loss}_{F,t}(x)$ is the cross entropy loss function of x with the correct label being t; i.e. $\text{loss}_{F,t}(x) := -\log_2(F_t(x))$. Some of these formulas are adjusted by the addition of constants such that $C(x + \delta) = t \iff f(x + \delta) \leq 0$ are satisfied.

In f_1, f_4, and f_5, Carlini and Wagner [2017] seem to assume that the confidence values for correct labels must be at least $\frac{1}{2}$ whenever prediction is correct. However, if $n > 2$ (i.e. the number of classes is more than 2), the confidence value for a correct label can be strictly less than $\frac{1}{2}$ whilst still being the highest confidence value. Thus, $C(x + \delta) = t \iff f(x + \delta) \leq 0$ does not always hold in f_1, f_4, and f_5.

Nonetheless, all of f_1, f_4, and f_5 measure a deviation from the correct prediction, each in a different way. Thus, they can be properly interpreted as loss functions.

4.3 Adversarial attacks and WRM

This section proves that WRM can be viewed as adversarial training where adversarial examples are computed using the adversarial attacks of [Carlini and Wagner, 2017]. As explained in [Staib and Jegelka, 2017], the original formulation of DRO in (3.1.1) can be approximated by (3.2.2), which is reproduced below for convenience:

$$
\begin{aligned}
\text{MIX} := \max_{\tilde{x}_1, \ldots, \tilde{x}_n} \; & \frac{1}{n} \sum_{i=1}^{n} \ell(\theta, \tilde{x}_i, y) \\
\text{subject to } & \frac{1}{n} \sum_{i=1}^{n} d(\tilde{x}_i, x_i)^p \leq \epsilon^p.
\end{aligned}
$$

Further, Lagrangian relaxation can transform MIX into

$$\frac{1}{n}\sum_{i=1}^{n}\max_{\tilde{x}_i}\{\ell(\theta,\tilde{x}_i,y_i)-\lambda d(\tilde{x}_i,x_i)^p\}. \tag{4.3.1}$$

This coincides with WRM proposed in [Sinha et al., 2018]. In their formulation, the inner maximization problem of DRO can be approximated by

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_n}[\phi_\lambda(\theta,x,y)], \tag{4.3.2}$$

where

$$\phi_\lambda(\theta,x,y)=\sup_{(x',y')}\{\ell(\theta,x',y')-\lambda c((x,y),(x',y'))\}. \tag{4.3.3}$$

Here, $c:(\mathcal{X}\times\mathcal{Y})\times(\mathcal{X}\times\mathcal{Y})\to\mathbb{R}^{\geq 0}\cup\{\infty\}$ is a cost function. Since Sinha et al. [2018] use the W_1 distance instead of the general W_p distance, we have $c((x,y),(x',y'))=d(x,x')^p+\infty\cdot\mathbb{1}_{y\neq y'}$. It is straightforward to see that (4.3.1) is an empirical approximation of (4.3.2).

Now we can notice the similarity between (4.1.3) and (4.3.3) (or (4.3.1)). To compare these two expressions more easily, we rename variables and modify the objective of optimization (i.e. minimization or maximization) in (4.1.3). As a result, (4.1.3) becomes

$$\max_{\tilde{x}_i}\{\ell(\theta,\tilde{x}_i,y_i)-\lambda d(\tilde{x}_i,x_i)\}. \tag{4.3.4}$$

Here, ℓ in (4.3.4) corresponds to f in (4.1.3).

For (4.3.3), since c returns ∞ whenever $y\neq y'$, we obtain

$$\max_{\tilde{x}_i}\{\ell(\theta,\tilde{x}_i,y_i)-\lambda c(\tilde{x}_i,x_i)\}. \tag{4.3.5}$$

Thus, if we view d in (4.3.4) as a cost function, (4.3.4) and (4.3.5) are identical. Indeed, the L_2-adversarial attack in [Carlini and Wagner, 2017] uses $(x,x')\mapsto\|x-x'\|_2^2$, which is exactly what Sinha et al. [2018] use for c as well. Therefore, the inner maximization problem of WRM (i.e. (4.3.3)) is identical to the adversarial attack's formulation in [Carlini and Wagner, 2017] (i.e. (4.1.3)).

A crucial difference between WRM and the adversarial attack setting of [Carlini and Wagner, 2017] is in the original problems that (4.3.4) and (4.3.5) seek to solve. In [Carlini and Wagner, 2017], the problem of crafting adversarial examples is originally formulated as (4.1.1), which can be generalized to

$$\begin{aligned}&\text{minimize}\quad c(\tilde{x},x)\\&\text{such that}\quad \ell_{0,1}(\theta,\tilde{x},y)=1,\end{aligned} \tag{4.3.6}$$

where c is a cost function and $\ell_{0,1}$ is the one-zero loss function (which is defined in (3.3.1)).

On the other hand, for DRO, the aim of (4.3.5) is to solve

$$\begin{aligned} &\max_{\tilde{x}_1,\ldots,\tilde{x}_n} \frac{1}{n}\sum_{i=1}^{n} \ell_{0,1}(\theta, \tilde{x}_i, y_i) \\ &\text{subject to } \frac{1}{n}\sum_{i=1}^{n} c(\tilde{x}_i, x_i) \leq \epsilon^p. \end{aligned} \tag{4.3.7}$$

Comparing (4.3.6) and (4.3.7), we can see that the former seeks to optimize the cost function (that is, perturbation size), whilst the latter seeks to optimize the loss.

Due to this theoretical connection between WRM and the adversarial attacks in [Carlini and Wagner, 2017], it is reasonable to expect the optimal loss function(s) identified by Carlini and Wagner [2017] for adversarial attacks to be optimal in the setting of WRM as well. This is based on the assumption that if we can find better adversarial examples for WRM's inner maximization problem, the robustness of WRM-trained neural networks will be higher.

4.4 Empirical evaluation of loss functions in adversarial attacks

Carlini and Wagner [2017] use Adam as an optimizer because they find it to converge more quickly than stochastic gradient descent (SGD) and the SGD's variant with momentum. To solve the optimization problem (4.1.3), they test multiple values of the Lagrangian parameter c. They conduct binary search (on log scale) on $c \in [10^{-2}, 10^2]$ using 20 iterations, and for each fixed value of c, they run 10^4 iterations of gradient descent.

As for datasets, they use MNIST and CIFAR-10, both of which produce identical results on the relative performance of loss functions. For experiment results, they report

- the quality of adversarial examples in terms of the size of δ;
- the success rate of adversarial attacks.

Regarding loss functions for the L_2-norm based attacks, [Carlini and Wagner, 2017] report that the best one is f_6, which consistently performs well in most cases. Although they also claim that the worst-performing loss function is f_1, it does not seem to be uniquely the worst. For example, f_1 can always find adversarial examples, whereas f_2, f_3, and f_4 do not. In this regard, f_1 is not the worst-performing.

Chapter 5

Empirical studies

This chapter describes the experiments[1] I conducted for this project. The first section provides an overview of the experiment setup used in all experiments, and subsequent sections present experiment results.

The experiments conducted for this project fall into two categories. The first category (Section 5.2) is the experiments conducted with the goal of reproducing some of the experiment results in [Staib and Jegelka, 2017; Sinha et al., 2018] (in a similar but not identical setting). The second category (Section 5.3) is the experiments for identifying an optimal loss function for distributionally robust optimization (DRO).

5.1 Basic experiment setup

Dataset Since both [Staib and Jegelka, 2017] and [Sinha et al., 2018] use the MNIST dataset, I will use it as well. This contains $6 \cdot 10^4$ images of (black-white) hand-written digits for training and 10^4 images for testing, where each image has the dimension of 28×28. In many major deep learning frameworks, each MNIST image is represented by a vector from $[0, 1]^{1 \times 28 \times 28}$ (to be more precise, each pixel value should be a multiple of $\frac{1}{255}$, but this detail is inconsequential).

Neural network In [Staib and Jegelka, 2017] and [Sinha et al., 2018], the same convolutional neural network (CNN) architecture is used to empirically evaluate adversarial training algorithms. The layer structure of the CNN used in [Staib and Jegelka, 2017; Sinha et al., 2018] is presented in Table 5.1. It is not important to know the details of convolutional neural layers and what kernels mean—Table 5.1 is provided only for interested readers.

All convolutional layers use the same activation function, which is either the rectified linear unit (ReLU) or the exponential linear unit (ELU). The linear layer uses softmax

[1] The source code for my experiment is available at https://github.com/LongPham7/Distributionally-Robust-Optimization.

Layer type	Output shape	Kernel size	Stride size
Input	(1, 28, 28)	N/A	N/A
2D convolutional	(64, 14, 14)	(8, 8)	(2, 2)
2D convolutional	(128, 5, 5)	(6, 6)	(2, 2)
2D convolutional	(128, 1, 1)	(5, 5)	(1, 1)
Linear	(10)	N/A	N/A

Table 5.1: Architecture of the neural network in [Staib and Jegelka, 2017] and [Sinha et al., 2018]

for activation. The learnable parameters in neural networks are initialized using the Glorot/Xavier method.

Regarding activation functions, ELU is recommended by Sinha et al. [2018] due to its smoothness, which is necessary for the theory behind their DRO algorithm (See Section 3.2.4). Staib and Jegelka [2017], on the other hand, use ELU only for WRM and use ReLU for all the other training procedures.

Programming environment The code for experiments is written in Python, which is a predominant programming language in machine learning and data science. As for deep learning frameworks for Python, I use PyTorch, which is one of the major deep learning frameworks along with TensorFlow. The primary reason for selecting PyTorch is that it is easier to learn than TensorFlow.

For adversarial attacks, I use an open-source library developed by IBM and called ART (adversarial robustness toolbox) [Nicolae et al., 2018].

Training algorithms The training algorithms for neural networks that we study in this chapter include empirical risk minimization (ERM) and three DRO algorithms: distributional PGD, FWDRO, and WRM. With regard to DRO, I will focus on the W_2 distance with the underlying distance function of L_2. The hyper-parameters for ERM and the three DRO algorithms are summarized in Table 5.2.

Hyper-parameters	ERM	Distributional PGD	FWDRO	WRM
Adversarial budget	$\epsilon = 0$	$\epsilon = 2.8$	$\epsilon = 2.8$	$\gamma \in \{0.0001, \ldots, 3.0\}$
Batch size	128	128	128	128
Epochs	25	25	25	25
Iterations for inner problems	N/A	15	15	N/A
Activation	ReLU/ELU	ReLU/ELU	ReLU/ELU	ReLU/ELU
Optimizer	Adam/SGD	Adam	Adam	Adam
Loss function	N/A	$\{f_1, \ldots, f_7\}$	$\{f_1, \ldots, f_7\}$	$\{f_1, \ldots, f_7\}$

Table 5.2: Hyper-parameters of ERM and DRO algorithms

In this table, the row "iterations for inner problems" refers to the number of iterations for solving the inner maximization problem of (3.1.1). This is only applicable to distributional PGD and FWDRO.

The reason for selecting $\epsilon = 2.8$ for distributional PGD and FWDRO is that if each pixel in an MNIST image (whose dimension is 28×28) is perturbed by 0.1, then the overall size of a perturbation in the L_2 norm (or more precisely $L_{2,2}$ matrix norm) is 2.8. The perturbation size of 0.1 for each pixel comes from [Staib and Jegelka, 2017], where the W_2 distance and the underlying distance function of L_∞ (as opposed to L_2 as in this project) are used.

Details on the range of γ for WRM and definitions of $f_1, \ldots, f_7$ will be given in later sections (See Sections 5.2.2 and 5.3).

Adversarial attacks The adversarial attacks we use to evaluate the robustness of neural networks are FGSM and PGD. In fact, I have also experimented with the use of distributional PGD to produce distributional adversarial perturbations (as opposed to pointwise perturbations). In this attack, we use PGD to solve (3.2.3). However, it does not perform well in terms of attack success rates. This is discussed in detail in Section D.1. Table 5.3 summarizes the hyper-parameters of FGSM and PGD attacks.

Hyper-parameters	FGSM	PGD
Norm	L_∞, L_2	L_∞, L_2
Adversarial budget	$\epsilon \in [0, 0.4], [0, 4.0]$	$\epsilon \in [0, 0.4], [0, 4.0]$
Iterations	N/A	15
Loss function	cross entropy	cross entropy

Table 5.3: Hyper-parameters of adversarial attacks

Regarding the adversarial budget of FGSM and PGD, the range of ϵ is $[0, 0.4]$ for the L_∞ norm and $[0, 4.0]$ for the L_2 norm. For both ranges, twenty-one evenly separated samples of ϵ are collected.

To evaluate the robustness of neural networks, for each fixed value of ϵ, the entire test dataset of MNIST, which contain 10,000 images, are used to generate adversarial examples. The number of misclassified examples are calculated, and the proportion of this to the total number of images in the test data is referred to as a "adversarial attack success rate". A line graph of adversarial attack success rates at various values of ϵ is then created.

Note that those adversarial examples whose original images are already misclassified also count towards an adversarial attack success rate. In fact, because the the test accuracy (in the absence of adversarial perturbations) of neural networks is usually very close to 1, it does not matter whether we count those adversarial examples whose original images are misclassified or not in the calculation of adversarial attack success rates.

5.2 Reproducing previous empirical studies

In this section, the experiments for reproducing some of the experiment results in [Staib and Jegelka, 2017; Sinha et al., 2018] are described. Specifically, the following three experiments are conducted:

1. An experiment to test the robustness of ERM-trained neural networks, that is, neural networks that do not go through adversarial training.
2. An experiment to determine an optimal γ in WRM such that subsequent experiments can focus on this optimal γ.
3. An experiment to determine the best-performing DRO algorithm among distributional PGD, FWDRO, and WRM.

5.2.1 Empirical risk minimization (ERM)

Overview

This experiment aims to reproduce the performance of ERM-trained neural networks reported in [Staib and Jegelka, 2017] and [Sinha et al., 2018]. Since Staib and Jegelka [2017] use ReLU in their neural networks and Sinha et al. [2018] use ELU, I have tested both activation functions.

Additionally, with respect to optimizers, Staib and Jegelka [2017] use SGD in their experiments (apart from WRM, for which they use Adam). On the other hand, Sinha et al. [2018] use Adam in their experiments according to their source code[2]. However, their source code only contains the code for WRM and does not contain any code for other training algorithms (e.g. ERM). Hence, it is unknown what optimizer they used to generate the data for ERM-trained neural networks presented in their paper. Thus, for completeness, I have tested both Adam and SGD.

Results

The robustness results of ERM-trained neural networks are presented in Figures 5.1 and 5.2.

To evaluate robustness, adversarial attacks (FGSM or PGD) are mounted using all of the 10000 test images from MNIST. The success rates of the adversarial attacks are then used as a measure of robustness.

Figure 5.1 is for the L_∞ norm, and Figure 5.2 is for the L_2 norm. The graphs on the left hand side are for FGSM, and those on the right hand side are for PGD.

Looking at Figures 5.1 and 5.2, we can make the following observations:

1. SGD generally produces less robust neural networks than Adam.

[2]The source code for WRM by Sinha et al. [2018] is publicly available at https://github.com/duchi-lab/certifiable-distributional-robustness.

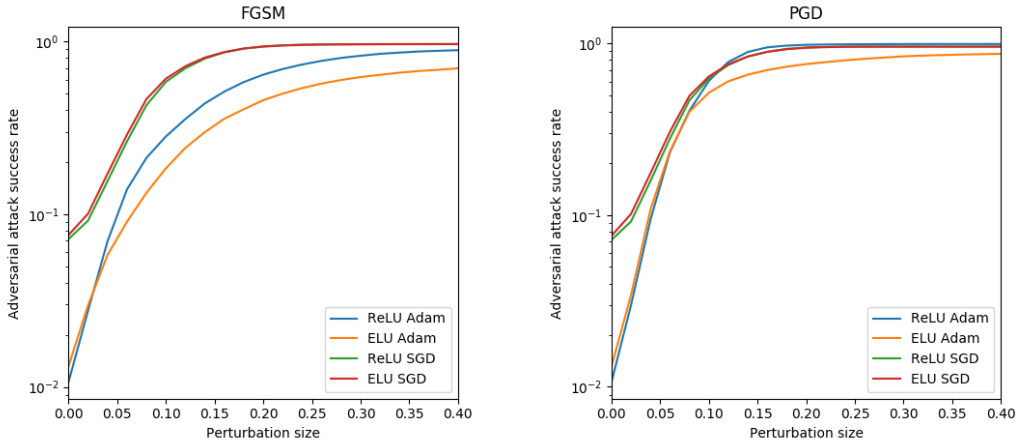

Figure 5.1: Robustness of ERM-trained models with the L_∞ norm

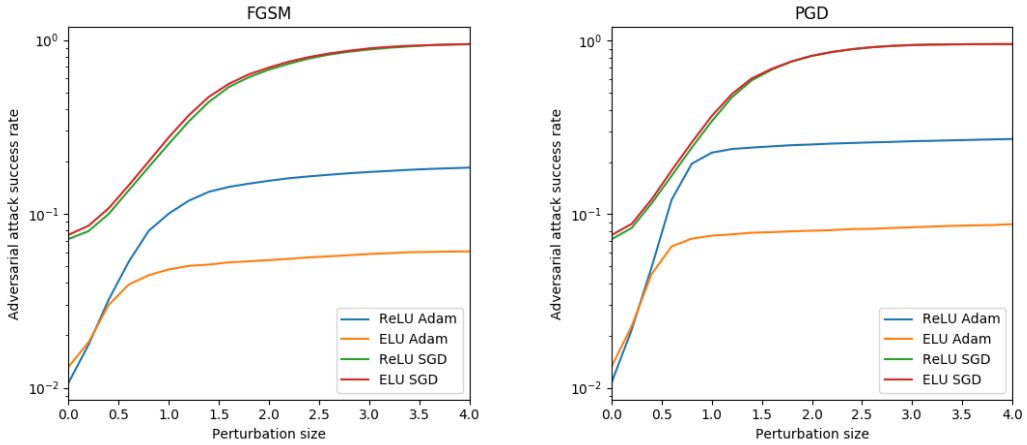

Figure 5.2: Robustness of ERM-trained models. The L_2 norm is used in adversarial attacks.

2. If SGD is used in training, there is no significant difference in robustness between ReLU and ELU.

3. If Adam is used, ELU produces more robust neural networks than ReLU, although the prediction accuracy on the test data (i.e adversarial attack with the budget of 0) is slightly but consistently higher with ReLU than with ELU.

These robustness results are consistent with what Staib and Jegelka [2017] (c.f. Figure 1 (a) and Figure 3) and Sinha et al. [2018] (c.f. Figure 3) report. Moreover, I deduce that Sinha et al. [2018] use SGD for ERM. In Figure 3 of [Sinha et al., 2018], the adversarial attack success rate of PGD with the L_2 norm on an ERM-trained neural network is close to 1 when $\epsilon = 2.0$, and according to my experiment results, this cannot be the

case if Adam is used.

5.2.2 Optimal value of γ for WRM

Overview

The formulation of WRM developed by Sinha et al. [2018] contains a Lagrangian parameter denoted by γ. As explained in Section C.2, γ and ϵ are inversely related by (C.2.3). However, we do not have a closed-form formula available to compute a corresponding value of ϵ from a given value of γ. In this experiment, I test different values of $\gamma \in \{0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0\}$ for two purposes:

1. to select an optimal γ to be used in subsequent experiments;
2. to check whether the optimal value of $\gamma = 0.001$ reported by Staib and Jegelka [2017] is plausible.

In line with the experiment setup of [Sinha et al., 2018], Adam is used in this experiment to train neural networks.

Results

The results for FGSM attacks with the L_∞ and L_2 norms are displayed in Figures 5.3 and 5.4, respectively. The results for PGD are available in Figures D.2 and D.3, both of which are in Appendix D.2.

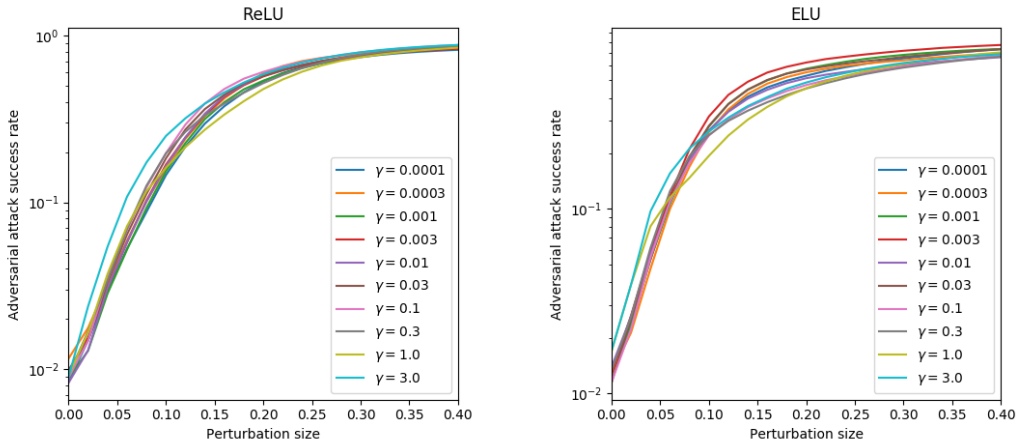

Figure 5.3: Robustness of WRM-trained models to FGSM with the L_∞ norm

The following observations can be made on these results:

1. Figure 5.3 suggests that although $\gamma = 1.0$ results in the highest robustness for some regions in ϵ, the difference is marginal. Likewise, in Figure D.2, all values of γ produce similar results, and $\gamma = 1.0$ is not optimal by any means. Thus, if we use the L_∞ norm, no value of γ uniquely stands out.

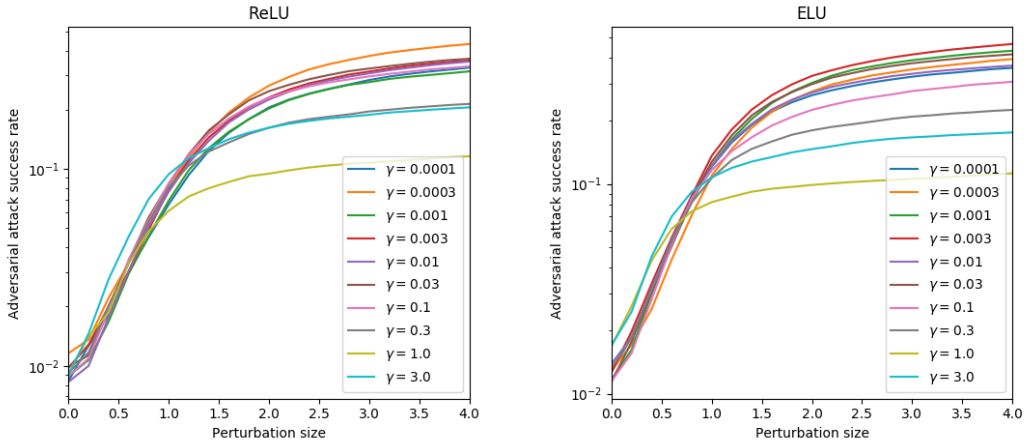

Figure 5.4: Robustness of WRM-trained models to FGSM with the L_2 norm

2. Figure 5.4 suggests that $\gamma = 1.0$ is markedly optimal in terms of the resulting robustness when ϵ is sufficiently large. The same remark applies to Figure D.3. However, when ϵ is small, $\gamma = 1.0$ is not optimal. This is in accord with the finding by Tsipras et al. [2019] that robustness and accuracy may be conflicting.

In respect of consistency with the results obtained by Staib and Jegelka [2017] and Sinha et al. [2018], we remark:

1. Staib and Jegelka [2017] obtained the optimal value of $\gamma = 0.001$. They use PGD with the L_∞ norm (equivalently, IFGSM) to evaluate robustness, and Figure D.2 shows that all neural networks exhibit closely clustered lines of robustness. Thus, their result of $\gamma = 0.001$ being optimal is plausible.

2. Figure 1 (a) in [Staib and Jegelka, 2017] contains the robustness result for a WRM-trained model with $\gamma = 0.001$, where the attack method is IFGSM (i.e. PGD with the L_∞ norm). Their line graph looks similar to Figure D.2. One difference is that at $\epsilon = 0.1$, Figure D.2 exhibits a higher attack success rate (i.e. less robustness) than Figure 1 (a) in [Staib and Jegelka, 2017].

3. Figure 2 in [Staib and Jegelka, 2017] displays the robustness of a WRM-trained model, where $\gamma = 0.001$, with respect tot PGD with the L_2 norm. Again, their line graph looks similar to Figure D.2. However, at $\epsilon = 1.0$, their graph indicates the attack success rate is less than 0.1, whereas Figure D.2 tells us that the attack success rate is higher than 0.1.

4. Figures 6 (c) and (d) in [Sinha et al., 2018], where they use $\gamma = 0.37$, give slightly lower attack success rates than Figures D.3 and D.2, respectively. Nonetheless, overall Figure 6 of [Sinha et al., 2018] is consistent with my experiment results.

5.2.3 Comparison of DRO-trained models

Overview

This experiment aims to compare three DRO algorithms, distributional DRO, FWDRO, and WRM, where we use the W_2 distributional distance and the L_2 distance for the underlying metric.

I will use $\gamma = 1.0$ for WRM since it is uniquely optimal if we use the L_2 norm in FGSM and PGD attacks. Also, $\gamma = 1.0$ does not significantly deviate from the optimal value when FGSM and PGD use the L_∞ norm. Notice that $\gamma = 1.0$ is larger than $\gamma = 0.37$, which Sinha et al. [2018] use for MNIST. Thus, there should not be an issue of $\gamma = 1.0$ being too small and consequently violating an underlying assumption of WRM.

Now that we have selected an optimal γ, we are ready to compare three DRO algorithms: WRM, the Frank-Wolfe method, and PGD (which is only valid for the W_2 distance with the underlying distance metric of L_2). For the FW method and PGD, I use $\epsilon = 2.8$ in training.

Results

The results are presented in Figures 5.5 and 5.6.

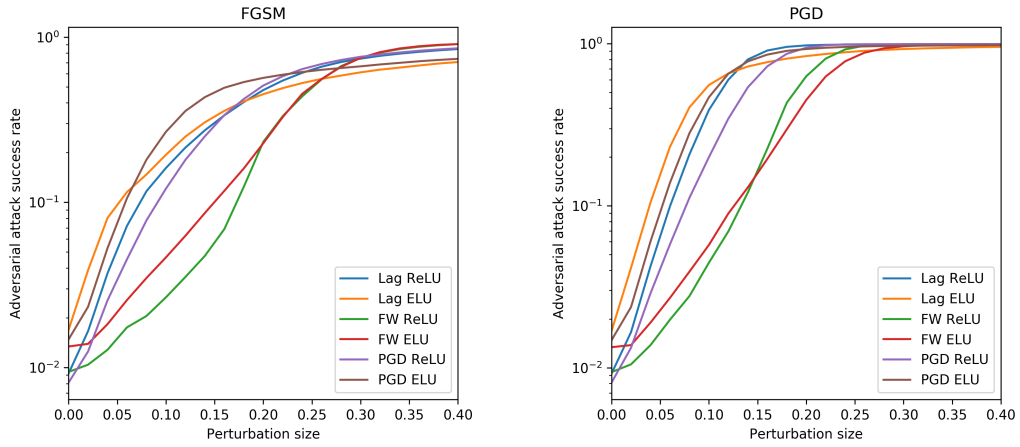

Figure 5.5: Robustness of models trained by various DRO algorithms. The L_∞ norm is used in adversarial attacks. The label "Lag" refers to the Lagrangian relaxation based DRO, that is, WRM.

Irrespective of the norm in use, the following trend emerges: the Frank-Wolfe method with ReLU is the most robust if ϵ is sufficiently small; otherwise, WRM with ELU is the most robust.

This experiment result is consistent with what Staib and Jegelka [2017] report: FWDRO is better than WRM for all $\epsilon \in [0, 0.20]$ in the IFGSM attack (i.e. PGD with the L_∞ norm). Note that their experimental setup differs from mine in that they use the underlying distance function of L_∞, whilst I use the L_2 distance.

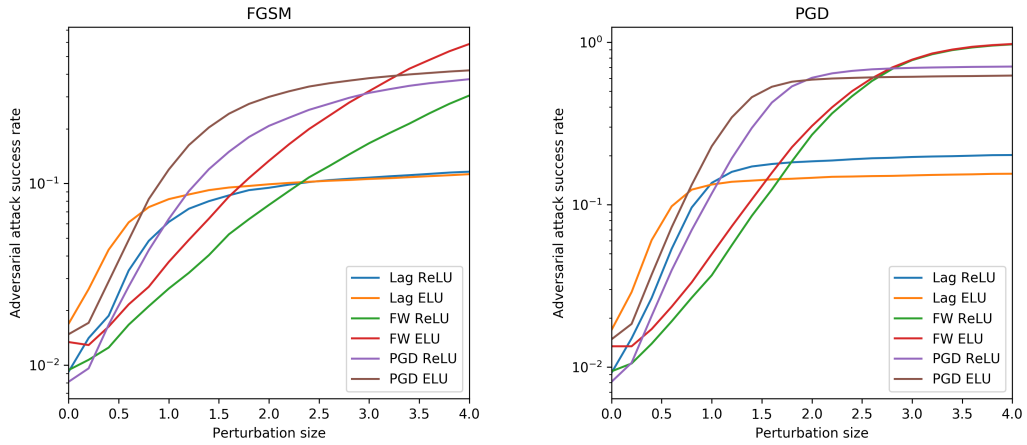

Figure 5.6: Robustness of models trained by various DRO algorithms. The L_2 norm is used in adversarial attacks. The label "Lag" refers to the Lagrangian relaxation based DRO, that is, WRM.

It is surprising that distributional PGD does not perform as well as other DRO algorithms because intuitively, distributional PGD reflects the theoretical formulation of DRO (See (3.2.2)) the most faithfully among the three DRO algorithms. In contrast to distributional PGD, FWDRO employs a more complicated optimization method (viz. the Frank-Wolfe method) so that it can deal with not only the L_2 norm but also the L_∞ norm for the underlying distance function. Likewise, WRM makes use of Lagrangian relaxation, which is an approximation technique.

Further, the relatively poor performance of distributional PGD as a training algorithm ties in with the poor performance of distributional PGD as an adversarial attack (See Appendix D.1).

5.3 Comparison of loss functions

This section compares loss functions in terms of the robustness of the resulting neural networks. The following DRO settings are considered:

1. WRM with ReLU and $\gamma = 1.0$;
2. FWDRO with ReLU and $\epsilon = 2.8$;
3. Distributional PGD with ReLU and $\epsilon = 2.8$.

The loss functions I examine are obtained by dropping constant terms from the seven loss functions in [Carlini and Wagner, 2017] listed in Section 4.2. As a result, the loss

functions become:

$$
\begin{aligned}
f_1(x) &:= \text{loss}_{F,t}(x) \\
f_2(x) &:= \left(\max_{i \neq t} F_i(x) - F_t(x)\right)^+ \\
f_3(x) &:= \text{softplus}\left(\max_{i \neq t} F_i(x) - F_t(x)\right) \\
f_4(x) &:= (0.5 - F_t(x))^+ \\
f_5(x) &:= \log_2(2.125 - 2F_t(x)) \\
f_6(x) &:= \left(\max_{i \neq t} Z_i(x) - Z_t(x)\right)^+ \\
f_7(x) &:= \text{softplus}\left(\max_{i \neq t} Z_i(x) - Z_t(x)\right).
\end{aligned}
$$

The original paper defines $f_5(x) := \ln(2 - 2F_t(x))$. However, I use $f_5 := \log_2(2.125 - 2F_t(x))$ instead so as to avoid arithmetic overflow (the base is changed from e to 2 for convenience). The original function $\ln(2 - 2F_t(x))$ approaches $-\infty$ as $F_t(x)$ approaches 1. Consequently, we obtain an NaN for the loss, which then leads to all weights in a neural network becoming NaN's.

5.3.1 WRM-trained neural networks

Overview

This section presents the experiment that compares the performance of $f_1, \ldots, f_7$ on WRM. In addition to the primary purpose of identifying an optimal loss function, this experiment serves another purpose: to check if the experiment results of [Carlini and Wagner, 2017] apply to not only adversarial attacks but also adversarial training (specifically WRM). As noted at the end of Section 4.3, WRM is essentially adversarial training where the inner maximization problem (i.e. the problem of finding adversarial examples) is solved by the adversarial attacks in [Carlini and Wagner, 2017]. Hence, it is reasonable to expect that the results of [Carlini and Wagner, 2017] will transfer to the setting of WRM.

There are three differences between the experiment setup in my experiment and that in [Carlini and Wagner, 2017]:

1. The Lagrangian parameter in WRM is fixed (my experiment uses $\gamma = 1.0$), whilst in the adversarial attacks of [Carlini and Wagner, 2017], we ought to first search for an optimal Lagrangian parameter (i.e. c in (4.1.3)).

2. Carlini and Wagner [2017] use a different architecture of neural networks. Their neural networks have a couple of max pooling layers and more convolutional layers than the architecture used in [Staib and Jegelka, 2017] and [Sinha et al., 2018].

3. Carlini and Wagner [2017] consider targeted adversarial attacks, whilst in WRM, the inner maximization problem of DRO is solved using untargeted attacks.

4. Carlini and Wagner [2017] evaluate loss functions mainly by the average size of perturbations rather than adversarial attack success rates. This is because their adversarial attack problem given in (4.1.1) is an optimization problem for the perturbation size.

The activation function for neural networks in this experiment is ReLU. Since the experiment results in Section 5.2.3 indicate that ReLU and ELU yield comparable robustness, I could use ELU instead. However, in view of the fact that one purpose of this experiment is to see if the experiment results of Carlini and Wagner [2017] carry over to the setting of WRM, I decided to use ReLU because this is what Carlini and Wagner [2017] use.

Carlini and Wagner [2017] report that f_6 is the best-performing loss function. Hence, if their results carry over from adversarial attacks to adversarial training despite the above two differences in the experimental setups, f_6 should also be the best-performing loss function.

Results

As indicated by Figures 5.7 and 5.8, f_6 is not an optimal loss function.

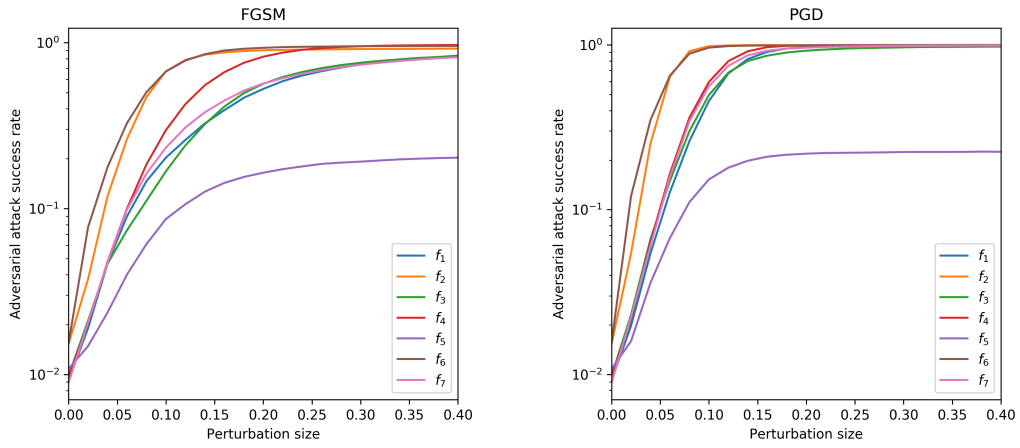

Figure 5.7: Robustness of WRM-trained neural networks with various loss functions. The L_∞ norm is used in adversarial attacks.

We can make the following remarks on these graphs:

1. Regardless of whether we use the L_∞ or L_2 norm in adversarial attacks, f_5 is uniquely the best-performing loss function. When the L_2 norm is used in attacks, the test accuracy of f_5 is worse than those of other loss functions. However, as

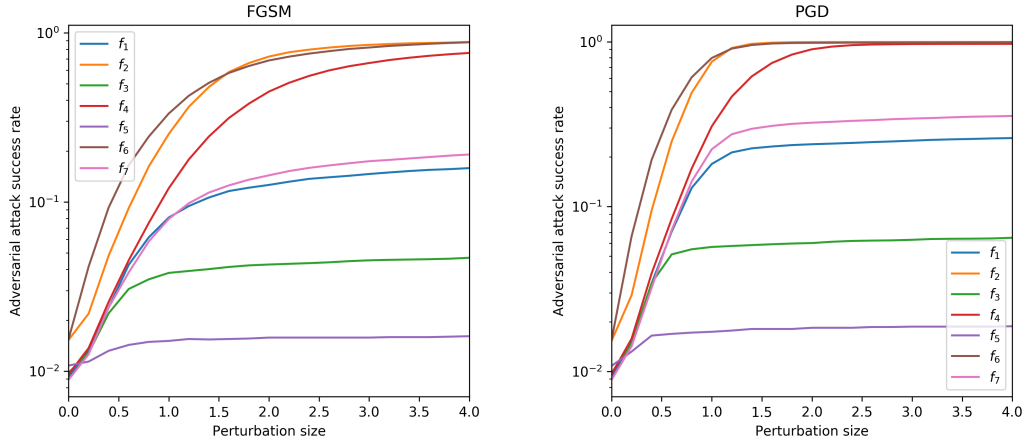

Figure 5.8: Robustness of WRM-trained neural networks with various loss functions. The L_2 norm is used in adversarial attacks.

soon as we start introducing adversarial perturbations to input images, f_5 quickly emerges with the highest degree of robustness to adversarial perturbations.

2. As for the next best-performing loss functions, if we use the L_∞ norm in adversarial attacks (See Figure 5.7), f_1 and f_3 are close contenders. For the L_2 norm, on the other hand, f_3 is uniquely the second best-performing,

3. The performance of f_6 is fairly poor—in fact, it is one of the worst-performing loss functions.

In conclusion, of the seven loss functions tested in the experiment, f_5 yields the most robust neural networks. Also, f_1 and f_3 perform fairly well, whereas f_6 performs poorly. The high degree of robustness arising from f_5 ties in with the good performance of f_5 reported in [Carlini and Wagner, 2017] (although their experiment results indicate f_6 is marginally better than f_5). However, the results on the performance of f_1 and f_6 in my experiment stand in contrast with the results in [Carlini and Wagner, 2017] since they conclude that f_1 is one of the worst-performing and f_6 is the best-performing.

5.3.2 FW-trained neural networks

This section empirically compares $f_1, \ldots, f_7$ on FWDRO. The activation function for this experiment is ReLU because the results in Section 5.2.3 show that overall ReLU gives more robustness than ELU when FWDRO is used.

The results are displayed in Figure 5.9 and 5.10.

We can observe the following:

1. Irrespective of adversarial attack types and norms, f_1 gives the highest degree of robustness. The closest contender of f_1 is f_7.

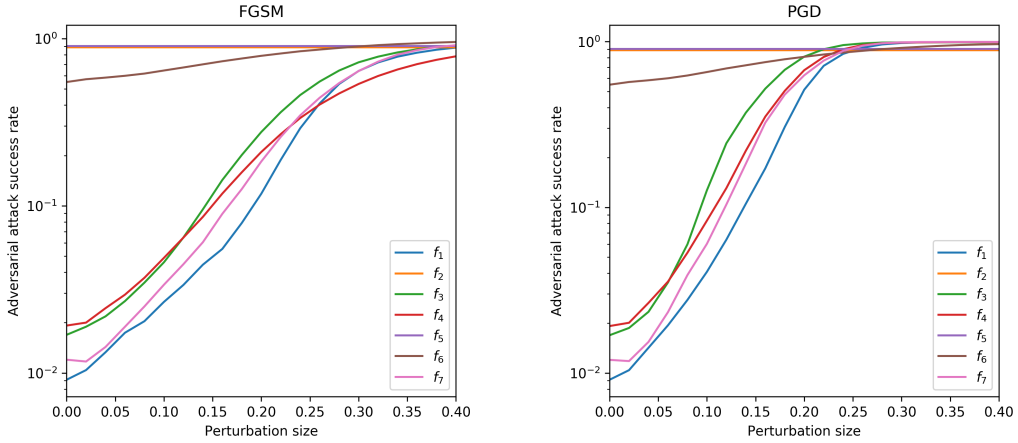

Figure 5.9: Robustness of FW-trained neural networks with various loss functions. The L_∞ norm is used in adversarial attacks.

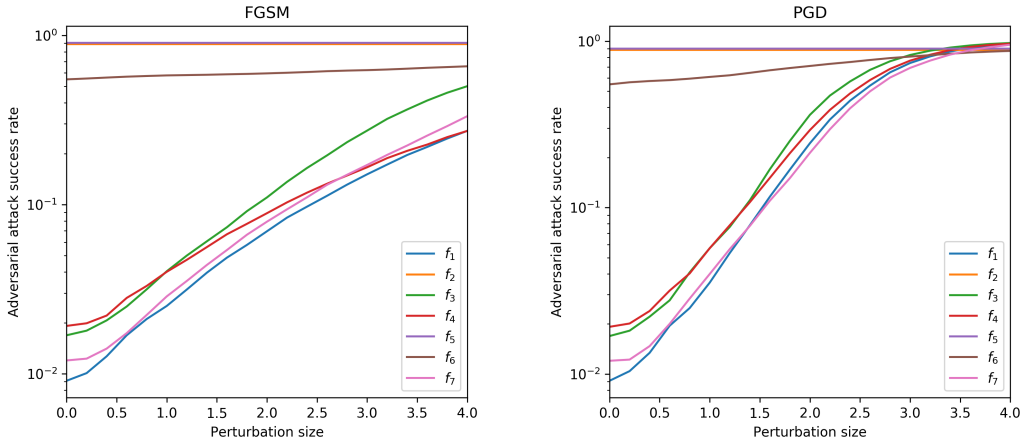

Figure 5.10: Robustness of FW-trained neural networks with various loss functions. The L_2 norm is used in adversarial attacks.

2. Some loss functions exhibit lines that are nearly horizontal. f_2 and f_5 result in considerably low test accuracy that is very close to 0 even in the absence of adversarial perturbations; hence, there is no room for an increase in adversarial attack success rates. On the other hand, for f_6 with the L_2-FGSM attack, the line is nearly horizontal even though there is some room for an increase in the adversarial attack success rate. This implies that f_6 can provide robustness to neural networks. However, since f_6 gives an unacceptably low test accuracy, this loss function is not recommendable in practice.

3. It is surprising that f_5, which is the best-performing loss function in WRM (Sec-

tion 5.3.1), is completely useless when used in FWDRO. It is unknown what causes the poor performance of f_5 in FWDRO.

5.3.3 PGD-trained neural networks

In this section, I test the performance of $f_1, \ldots, f_7$ on distributional PGD. The activation function is again ReLU, for Section 5.2.3 indicates that ReLU gives more robustness than ELU.

The results are displayed in Figure 5.11 and 5.12.

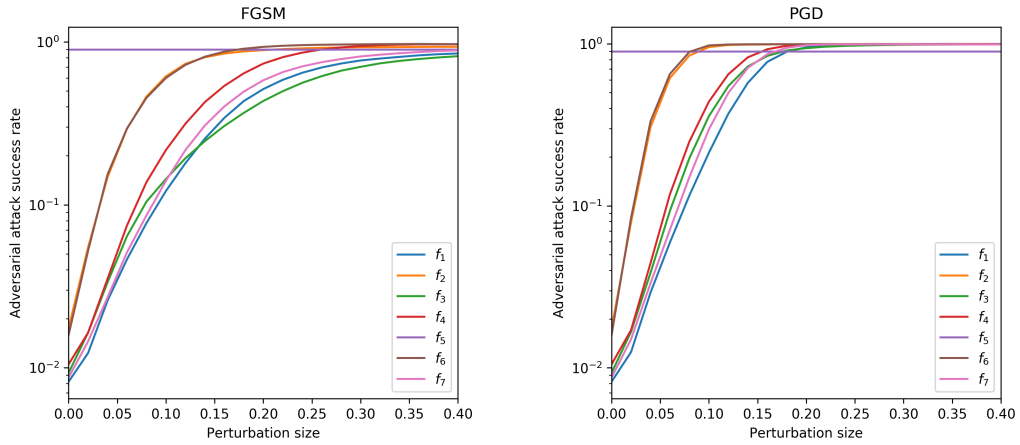

Figure 5.11: Robustness of PGD-trained neural networks with various loss functions. The L_∞ norm is used in adversarial attacks.

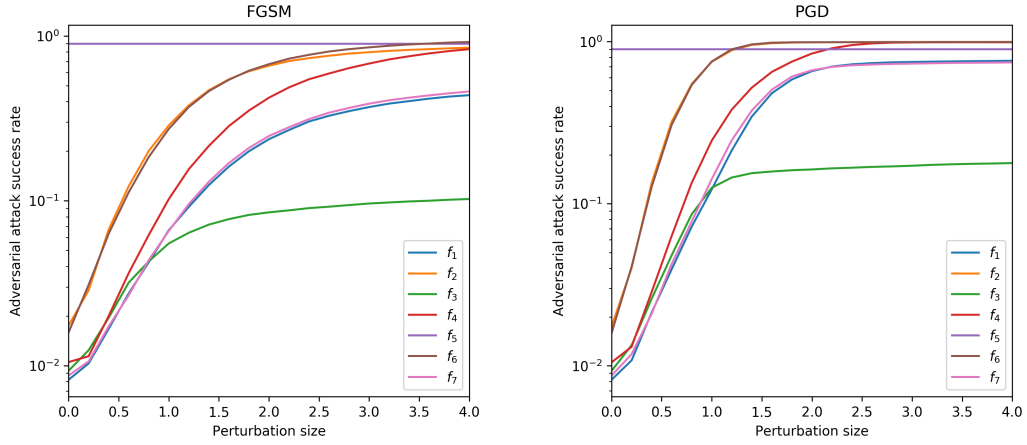

Figure 5.12: Robustness of PGD-trained neural networks with various loss functions. The L_2 norm is used in adversarial attacks.

We can make the following observations:

1. When the L_2 norm is used in adversarial attacks, f_3 is markedly the best-performing loss function. However, if L_∞ norm is used, although f_3 is one of the three best-performing loss functions, it is not uniquely the best. Other contenders are f_1 and f_7. Speaking of f_1, when the L_2 norm is used in adversarial attacks, although f_1 clearly does not give as much robustness as f_3, f_1 is nonetheless the second best-performing loss function.

2. As in the experiment on FWDRO (Section 5.3.2), f_5 exhibits an unacceptably low test accuracy and a nearly horizontal line of adversarial attack success rates. This is true for both of FGSM and PGD and both of the L_∞ and L_2 norms.

Chapter 6

Conclusion

6.1 Conclusion

In this work, I have identified two issues with the current state of research on DRO (Section 3.3). I have also proved that WRM, which is a distributionally robust optimization (DRO) algorithm by Sinha et al. [2018], can be viewed as an algorithm for adversarial training where adversarial examples are devised using the formulation of adversarial attacks in [Carlini and Wagner, 2017] (Section 4.3).

Furthermore, I have empirically studied the relationship between loss functions and the robustness of neural networks in the same setting of DRO as [Staib and Jegelka, 2017; Sinha et al., 2018] (Chapter 5). In these empirical studies, I have tested three DRO algorithms (i.e distributional PGD, FWDRO, and WRM), evaluating the performance of the seven loss functions used in [Carlini and Wagner, 2017] and identifying an optimal loss function for each DRO algorithm. In addition, I have conducted experiments in order to reproduce part of the experiment results reported by Staib and Jegelka [2017] and Sinha et al. [2018], confirming that their results generally agree with my experiment results.

Regarding the relationship between loss functions and the robustness of neural networks trained by DRO, the key takeaways are:

1. Optimal loss functions are not universal; rather, they are specific to settings and hyper-parameters. Concretely, through the experiment results of this project, we know that optimal loss functions depend on DRO algorithms and that an optimal loss function for one DRO algorithm may perform poorly on another DRO algorithm.

2. As far as the three DRO algorithms examined in this project and the specific neural network architecture used in this project are concerned, the cross entropy loss function, which is widely used, generally produces acceptable performance with respect to the robustness of neural networks.

6.2 Future work

Potential avenues for future research are discussed below.

Dependency of optimal loss functions on neural network architecture The fact that [Carlini and Wagner, 2017] and this project yield different optimal loss functions suggests that optimal loss functions in one setting (whether it is for adversarial attacks or adversarial training) do not necessarily carry over to other settings. The differences between the setting of [Carlini and Wagner, 2017] and this project's settings are listed in Section 5.3.1. This motivates further research on dependencies of optimal loss functions in adversarial attacks/training; e.g. architecture of neural networks.

Development of distributional adversarial attacks As explained in Section D.1, if we use distributional PGD to launch distributional adversarial attacks, the algorithm has a considerably lower attack success rate than conventional PGD. One direction for future research is to investigate why distributional PGD is less effective than (pointwise) PGD. This research direction is worthwhile because as of now, DRO algorithms are tested against pointwise adversarial attacks, which DRO is not meant to deal with (See Section 3.3). To evaluate DRO algorithms in a fair manner, it is required to establish methodologies for "distributionally" adversarially perturbing probability distributions.

Investigation into loss functions with low accuracy In the experiment results of Sections 5.3.2 and 5.3.3, we observe that some loss functions produce nearly horizontal lines of high adversarial attack success rates across the whole range of ϵ (including $\epsilon = 0$). What is worth noting is that f_5, which performs well in WRM, yields a significantly low test accuracy at $\epsilon = 0$ if we use FWDRO. It is therefore interesting to examine why f_5, for instance, does not perform well in FWDRO.

Appendix A

Fundamentals of optimization

A.1 Distances, Lipschitz continuity, and norms

This section introduces formal definitions of three notions that come up often in the literature of deep neural networks and adversarial perturbations: distances, Lipschitz continuity, and norms.

A.1.1 Distances and Lipschitz continuity

Definition A.1.1 (Metric space)**.** *Given a set A and a function $d_A : A \times A \to \mathbb{R}$, the pair (A, d_A) is a metric space if and only if the following properties are satisfied:*

$$\begin{aligned}
&\forall x, y. d_A(x, y) \geq 0 \\
&\forall x, y. d_A(x, y) = 0 \iff x = y \\
&\forall x, y. d_A(x, y) = d_A(y, x) \\
&\forall x, y, z. d_A(x, z) \leq d_A(x, y) + d_A(y, z).
\end{aligned}$$

The function d_A is called a metric/distance.

Definition A.1.2 (Lipschitz continuity)**.** *Consider a function $f : X \to Y$, where (X, d_X) and (Y, d_Y) are metric spaces. The function f is said to be Lipschitz continuous (or K-Lipschitz continuous) if and only if there exists a real constant $K \geq 0$ such that*

$$d_Y(f(x_1, x_2)) \leq K d_X(x_1, x_2)$$

for all $x_1, x_2 \in X$. K is called a Lipschitz constant, and the smallest such K, denoted by K_{best}, is called the best Lipschitz constant.

Intuitively, Lipschitz continuity means that the function's output cannot change at an arbitrarily large rate with respect to the change in the function's input. If a function is differentiable at all points in the domain, then its Lipschitz constant is an upper bound of the function's gradient. For instance, $f : \mathbb{R}^{>0} \to \mathbb{R}$ defined as $f := \frac{1}{x}$ is not Lipschitz continuous, since its derivative on the interval $(0, \infty)$ cannot be bounded by any constant.

A.1.2 Norms

Vectors are ubiquitous in this field of artificial neural networks, serving as tools to express weights and states of neural networks. The definition of metrics allows many distinct distances to be defined, whether we deal with matrices or other mathematical entities. If we focus on vector spaces, their metrics are usually defined using the notion of norms, which formalizes the intuitive concept of "length" of a vector.

As we mostly work with real numbers as opposed to complex numbers in artificial neural networks, it suffices to consider vector spaces over the real field. Henceforth, I only consider vector spaces over the real field.

The following definitions for bilinear forms, positive definiteness, and inner products are taken from the lecture notes on linear algebra at the University of Oxford in 2014.

Definition A.1.3 (Bilinear form)**.** *Given a vector space V over the real field, a function $\theta : V \times V \to \mathbb{R}$ is a bilienar form if and only if θ satisfies*

$$\theta(\lambda u + \mu v, w) = \lambda\theta(u, w) + \mu\theta(v, w)$$
$$\theta(u, \lambda v + \mu w) = \lambda\theta(u, v) + \mu\theta(u, w),$$

where $u, v, w \in V$ and $\lambda, \mu \in \mathbb{R}$. θ is said to be symmetric if and only if

$$\theta(u, v) = \theta(v, u)$$

for all $u, v \in V$.

Definition A.1.4 (Positive definiteness)**.** *A symmetric bilinear form θ on a vector space V is positive definite if and only if for all $u \in V$ we have*

$$\theta(u, u) \geq 0$$

and

$$\theta(u, u) = 0 \iff u = 0$$

Definition A.1.5 (Inner product)**.** *Given a vector space V over the real field, an inner product on V is defined as a positive definite symmetric bilinear form. In other words, an inner product on V is $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ such that the following are satisfied:*

1. *Positive definiteness: for all $v \in V$, we have $\langle u, u \rangle \geq 0$ and $\langle u, u \rangle \iff u = 0$.*
2. *Symmetry: for all $u, v \in V$, we have $\langle u, v \rangle = \langle v, u \rangle$.*
3. *Linearity: for all $u, v, w \in V$ and $\lambda, \mu \in \mathbb{R}$, we have $\langle \lambda u + \mu v, w \rangle = \lambda\langle u, w \rangle + \mu\langle v, w \rangle$.*

The standard inner product of column vectors is

$$\langle x, y \rangle = x^\intercal y = \sum_{i=1}^{D} x_i y_i,$$

where x and y have the dimension of D. The same definition applies to row vectors, with slight modification (i.e. we should have $xy^\intercal$ instead of $x^\intercal y$).

Definition A.1.6 (Norm)**.** *Given a vector space V over the real field, a function $f : V \to \mathbb{R}$ is a norm on V if and only if it satisfies*

1. *Positive definiteness: $f(u) \geq 0$ for all $u \in V$ and $f(u) = 0 \iff u = 0$.*
2. *Homogeneity: $f(cu) = |c| f(u)$ for all $c \in \mathbb{R}$ and $u \in V$.*
3. *Triangle inequality: $f(u + v) \leq f(u) + f(v)$ for all $u, v \in V$.*

We can obtain norms from inner products as stated in the next proposition.

Proposition A.1.1. *Given an inner product $\langle \cdot, \cdot \rangle$ on a vector space V, $f(u) = \sqrt{\langle u, u \rangle}$ is a norm.*

Some commonly used norms of column/row vectors belong to the class of the L_p norms, where $p \geq 1$ (and it is possible to have $p = \infty$). If $u = (u_1, \ldots, u_D)$ or $u = (u_1, \ldots, u_D)^\intercal$, then its L_p norm is given by

$$\|u\|_p = \left(\sum_{i=1}^{D} |u_i^p| \right)^{\frac{1}{p}}.$$

The L_1 norm is also known as the Manhattan distance, and the L_2 norm is the Euclidean distance. Further, we can define the L_∞ norm by taking the limit of the L_p norm as p approaches infinity. This yields

$$\|u\|_\infty = \max_{1 \leq i \leq D} |u_i|.$$

In a vector space of column/row vectors, the most predominant metric is the L_p norm of the difference between two input vectors. Formally, this metric is defined as

$$d(x, y) = \|x - y\|_p.$$

A.2 Primer on convex optimization

This section reviews some widely deployed gradient descent algorithms for (constrained) convex optimization. I will begin by introducing key definitions and concepts in convex optimization to set the stage. The presentation style of this section is attributed to Bubeck [2015].

A.2.1 Introduction

A convex optimization problem takes the shape of

$$\begin{aligned} \text{minimise} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{X}, \end{aligned}$$

where the objective function $f(x)$ is convex and $\mathcal{X}$ is a convex set. Without loss of generality, the aim of convex optimization is assumed to be minimizing an objective function instead of its maximization.

In numerical optimization, one common paradigm is the black-box model, where we can only retrieve information about individual points of an objective function to be optimized. If we only need to access the values of an objective function, this optimization method is said to be zeroth order. If we additionally need to access gradients of the objective function (i.e. $\nabla f(x)$), the optimization method is first order. As for the constraints specifying feasible regions, in the black-box model, we can assume that they are known. Alternatively, we can use a more restrictive model in which we only have access to a separation oracle. A separation oracle tells whether a given point is in the feasible region or not (and if not, the oracle returns a hyperplane separating the point and the feasible region).

Another paradigm in numerical optimization is called the structured model, in which we can access information about the global structure of an objective function. For instance, in the LASSO problem, the loss function, which is usually the least squared error, is smooth, whilst the penalty term is not smooth (smoothness will be formally defined later). By exploiting this structure, we can develop an efficient algorithm to the LASSO problem.

Convex optimization problems are an important class of optimization problems to investigate for numerous reasons, a couple of which are

1. Convexity gives rise to some nice properties that are helpful in optimization. One particularly noteworthy property is that local optima are also global optima. To check whether a given point is locally optimal, we only need local information. On the other hand, it is much more difficult to check whether a given point is globally optimal (under no additional assumptions on the objective function or the feasible region) if we use the black-box model. By contrast, in convex optimization problems, finding global optima amounts to finding local optima.

2. Many real-world problems can be formulated (or reformulated) as convex optimization problems.

On the first point, to prove that local optima are global optima in convex optimization, we use not only the assumption that f is convex but also the assumption that $\mathcal{X}$ is convex. Thus, in the definition of convex optimization problems, it is essential to impose the convexity requirement on both objective functions and feasible regions. On a related note, convex optimization problems are usually assumed to be constrained. Unconstrained convex optimization can be viewed as constrained convex optimization problems with the feasible region of $\mathbb{R}^n$, which is indeed convex.

The second point by no means suggests that nonconvex optimization is less prevalent than convex optimization. The real world also abounds with nonconvex optimization problems, including deep neural networks. It is possible to borrow algorithms from convex optimization and apply them to nonconvex optimization to obtain meaningful results.

A.2.2 Projected subgradient descent

In gradient descent-based algorithms for convex optimization, we use gradients to determine the direction in which an objective function decreases. One issue with using gradients is that they are not well-defined if the objective function is not differentiable. To sidestep this issue, we define the notion of subgradients, which play the same role as gradients and can be well-defined even if the objective function is not differentiable.

Definition A.2.1 (Subgradients)**.** *Given $\mathcal{X} \subseteq \mathbb{R}^n$ and $f : \mathcal{X} \to \mathbb{R}$, $g \in \mathbb{R}^n$ is a subgradient of f at $x \in \mathcal{X}$ if and only if for any $y \in \mathcal{X}$, we have*

$$f(x) - f(y) \leq g^\intercal (x - y).$$

The set of subgradients of f at x is denoted by $\partial f(x)$.

The following proposition relates convexity of functions and existence of subgradients:

Proposition A.2.1 (Proposition 1.1 in [Bubeck, 2015])**.** *Consider a convex set $\mathcal{X} \subseteq \mathbb{R}^n$ and a function $f : \mathcal{X} \to \mathbb{R}$. If $\partial f(x) \neq \emptyset$ for all $x \in \mathcal{X}$, f is a convex function. Conversely, if f is convex, then for any x in the interior of $\mathcal{X}$, $\partial f(x) \neq \emptyset$. Furthermore, if f is convex and differentiable at x, then $\nabla f(x) \in \partial f(x)$.*

Projected subgradient descent iteratively makes a step towards a minimizer of the objective function using the following equation:

$$\begin{aligned} y_{t+1} &:= x_t - \eta g_t \\ x_{t+1} &:= \Pi_{\mathcal{X}}(y_{t+1}), \end{aligned}$$

where $g_t \in \partial f(x_t)$, $\Pi_{\mathcal{X}}$ is a projection operator on $\mathcal{X}$, and η is a fixed step size. Projected gradient descent (PGD) is a variant of projected subgradient descent in which the objective function is differentiable and hence the gradient $\nabla f(x_t)$ is used in place of a subgradient.

The projection operator $\Pi_{\mathcal{X}}$ is formally defined by

$$\Pi_{\mathcal{X}}(x) := \arg\min_{y \in \mathcal{X}} \|x - y\|_2 .$$

It is worth observing that we use the Euclidean distance in the definition of project operators for projected subgradient descent. The reason for using the Euclidean distance will be revealed later in the convergence proof of projected subgradient descent.

The following property of projection operators will also prove to be useful in the convergence proof:

Lemma A.2.1 (Lemma 3.1 in [Bubeck, 2015])**.** *Given $x \in \mathcal{X}$ and $y \in \mathbb{R}^n$, we have*

$$(\Pi_{\mathcal{X}}(y) - x)^\intercal (\Pi_{\mathcal{X}}(y) - y) \leq 0.$$

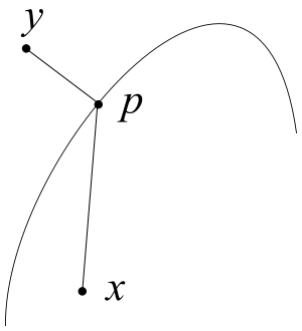

Figure A.1: Projection on a convex set

This lemma is illustrated in Figure A.1, where point p is the projection of y on a convex set. Due to convexity of the set, three points y, p, and x form an angle larger than or equal to $\frac{\pi}{2}$. Hence, $(\Pi_{\mathcal{X}}(y) - x)^\intercal(\Pi_{\mathcal{X}}(y) - y)$, which is the dot product of two vectors $\overrightarrow{xp}$ and $\overrightarrow{yp}$, is smaller than or equal to 0. In addition, for the same reason, we have $\|\Pi_{\mathcal{X}}(y) - x\|_2^2 + \|\Pi_{\mathcal{X}}(y) - y\|_2^2 \leq \|y - x\|_2^2$.

The following theorem provides us with a suitable step size and a convergence rate of projected subgradient descent.

Theorem A.2.1 (Theorem 3.2 in [Bubeck, 2015])**.** *Projected subgradient descent with the step size of* $\eta = \frac{R}{L\sqrt{t}}$ *satisfies*

$$f\left(\frac{1}{t}\sum_{s=1}^{t} x_s\right) - f(x^*) \leq \frac{RL}{\sqrt{t}},$$

where L is a Lipschitz constant of f, t is the number of iterations, and R is the radius (in the Euclidean distance) of a ball that completely contains $\mathcal{X}$. Here, $x^* \in \mathcal{X}$ denotes an optimal value.

Theorem A.2.1 suggests that $f\left(\frac{1}{t}\sum_{s=1}^{t} x_s\right)$ is a good approximation of $f(x^*)$. Alternatively, the final output of projected subgradient descent can be the smallest objective value that we have computed, namely $\min\{f(x_s) \mid 1 \leq s \leq t\}$. Similarly to Theorem A.2.1, $\min\{f(x_s) \mid 1 \leq s \leq t\}$ satisfies

$$\min\{f(x_s) \mid 1 \leq s \leq t\} - f(x^*) \leq \frac{RL}{\sqrt{t}}.$$

A.2.3 Important properties of objective functions

Here, I introduce two properties of objective functions that can speed up convergence of projected subgradient descent (and more generally gradient descent-based iterative algorithms for convex optimization problems).

The first property, β-smoothness, is defined below.

Definition A.2.2. *A continuously differentiable function f is said to be β-smooth if and only if*

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq \beta \|x - y\|_2 .$$

In words, the gradient of f is β-Lipschitz continuous.

Because different notions of smoothness are utilized in the literature of numerical optimization, we should be mindful of which definition of smoothness is in use in the current context. For example, in [Nocedal and Wright, 1999], by smoothness, the authors mean that the second derivatives exist and are continuous.

In projected subgradient descent, if the objective function is assumed to be β-smooth as well as convex, the convergence rate improves from $O\left(\frac{1}{\sqrt{t}}\right)$ to $O\left(\frac{1}{t}\right)$.

The second key property, strong convexity, is defined as follows.

Definition A.2.3. *A (differentiable) convex function $f : \mathcal{X} \to \mathbb{R}$ is α-strongly convex if and only if for any $x, y \in \mathcal{X}$*

$$f(x) - f(y) \leq \nabla f(x)^\intercal (x - y) - \frac{\alpha}{2} \|x - y\|_2^2 .$$

We can define α-strong convexity even when f is convex but not differentiable. This is achieved by replacing the gradient $\nabla f(x)$ in the above definition with a subgradient $g \in \partial f(x)$.

The following lemma offers an alternative characterization of α-strong convexity:

Lemma A.2.2 (Bubeck [2015])**.** *A convex function f is α-strongly convex if and only if $x \mapsto f(x) - \frac{\alpha}{2}\|x\|_2^2$ is convex.*

The above lemma suggests that α is (a lower bound of) a measure of the curvature of f. To illustrate this, for simplicity, suppose x is a one-dimensional variable, that is, a scalar. If the second derivative of f is small (i.e. the curvature of f is low), $x \mapsto f(x) - \frac{\alpha}{2}x^2$ remains convex only for small α. Conversely, if f's second derivative is large (that is, f's curvature is high), then α can be large before it makes $x \mapsto f(x) - \frac{\alpha}{2}x^2$ nonconvex.

Strong convexity alone improves the convergence rate of projected subgradient descent from $O\left(\frac{1}{\sqrt{t}}\right)$ to $O\left(\frac{1}{t}\right)$. Moreover, strong convexity combined with smoothness can further improve the convergence rate [Bubeck, 2015].

A.2.4 Frank-Wolfe algorithm

The Frank-Wolfe algorithm (also known as conditional gradient descent) was first published by Frank and Wolfe [1956]. Like projected subgradient descent, the Frank-Wolfe method uses gradient descent to iteratively make a step towards a minimizer of an objective function. However, one major difference between them is that the Frank-Wolfe algorithm does not use projection, for we always stay inside the feasible region $\mathcal{X}$ designated by constraints.

We assume that f is β-smooth for the Frank-Wolfe method. The iterative update in the Frank-Wolfe method works by

$$y_t \in \arg\min_{y \in \mathcal{X}} \nabla f(x_t)^\intercal y$$
$$x_{t+1} := (1 - \gamma_t)x_t + \gamma_t y_t,$$

where $(\gamma_s)_{s \geq 1}$ is a fixed sequence. It is clear that

$$y_t = x_t + \arg\min_{\alpha + x_t \in \mathcal{X}} \nabla f(x_t)^\intercal \alpha.$$

The optimal value for α has the longest component along the descent direction (that is, the opposite of the directional derivative) of f at x_t. Essentially, this step can be regarded as finding an optimal point (within $\mathcal{X}$) of the linear approximation to f. Computing y_t amounts to solving a linear optimization problem (note that only the objective function is linear; the constraint is not necessarily linear), which in some cases is computationally more tractable than computing projection.

The Frank-Wolfe method then selects a point x_{t+1} located somewhere between x_t and y_t. In order for the algorithm to work properly, x_t in the first iteration (i.e. x_0 or x_1, depending on the starting index) must be drawn from $\mathcal{X}$. The first x_t is usually randomly drawn from $\mathcal{X}$.

With regard to the sequence $(\gamma_s)_{s \geq 1}$, there are various rules for determining x_{t+1} from x_t and y_t. One natural choice is to compute $\gamma_s = \arg\min_{\gamma \in [0,1]} f(x_s + \gamma(y_s - x_s))$, and this scheme is known as line search. Another choice is to use $\gamma_s = \frac{2}{s+1}$ for $1 \leq s \leq t$. In this case, the convergence rate is given by the following theorem.

Theorem A.2.2 (Theorem 3.8 in [Bubeck, 2015]). *Fix a norm $\|\cdot\|$. Let f be convex and β-smooth with respect to the norm $\|\cdot\|$. If $R = \sup_{x,y \in \mathcal{X}} \|x - y\|$ and $\gamma_s = \frac{2}{s+1}$ for $1 \leq s \leq t$, then for any $t \geq 2$, we have*

$$f(x_t) - f(x^*) \leq \frac{2\beta R^2}{t+1}.$$

In this theorem, the β-smoothness with respect to an arbitrary norm $\|\cdot\|$ is defined by

$$\|\nabla f(x) - \nabla f(y)\|_* \leq \beta \|x - y\|,$$

where $\|\cdot\|_*$ is the dual norm; i.e. $\|g\|_* := \sup_{x \in \mathbb{R}^n : \|x\| \leq 1} g^\intercal x$. For instance, the dual norm of the L_p norm is the L_q norm, where $\frac{1}{p} + \frac{1}{q} = 1$ (if one of p and q is 1, the other takes the value of ∞).

In comparison with projected subgradient descent, if f is assumed to be convex and β-smooth, both optimization methods have the convergence rate of $O\left(\frac{1}{t}\right)$. One advantage of the Frank-Wolfe method over projected subgradient descent is that the former does not require projection and instead uses a linear minimization oracle over $\mathcal{X}$ (i.e. an algorithm to solve a linear optimization problem with the feasible region being $\mathcal{X}$). In many applications in machine learning, it is easier to solve linear optimization problems

than computing projection because the constraints that specify feasible regions are well-structured. This is why the Frank-Wolfe method has recently gained more popularity in the machine learning community [Lacoste-Julien, 2016; Reddi et al., 2016].

A.2.5 Stochastic gradient descent and coordinate descent

In this section, I briefly introduce two techniques to reduce the amount of computation in each iteration of gradient-based optimization methods (e.g. projected subgradient descent and the Frank-Wolfe method). Since they specify how to reduce the amount of computation in each iteration rather than in what direction to make a step or how large the step size should be, these techniques are orthogonal to convex optimization methods. Hence, we can combine these techniques with any iterative convex optimization method (although the convergence analysis may need to be carried out for each individual case).

The first technique works by computing an approximation of a subgradient whose expected value (when seen as a random variable) is the true subgradient. If we use this technique in projected subgradient descent, the resulting optimization algorithm is known as stochastic gradient descent (SGD). In [Bubeck, 2015], SGD is studied in the context of constrained convex optimization problems, but in other sources, SGD is presented in the context of unconstrained convex optimization problems.

In many applications in machine learning, an objective function to be optimized is in the form of $\sum_{i=1}^{n} f(x_i)$, where f is some function (e.g. composition of a neural network and a loss function) and x_i are training examples. When n is large, it is time-consuming to compute the gradient of the objective function by summing all the gradients of $f(x_i)$ for $1 \leq i \leq n$. This is where projected gradient descent comes in handy, because it computes an approximation to the gradient by using only a small set of randomly chosen training examples.

The second technique to reduce the amount of computation in each iteration operates by updating only a small number of coordinates. An iterative convex optimization method using this technique is called coordinate descent (CD), and there are many variants of CD [Wright, 2015]. If more than one coordinate is updated in each iteration and this fact is to be emphasized, this method is called block coordinate descent (BCD). If the coordinates to be updated are picked out randomly, then the method is called random/stochastic coordinate descent (RCD/SCD). With respect to the type of optimization problems, many sources on coordinate descent seem to focus on the unconstrained setting [Bubeck, 2015; Wright, 2015].

A.3 General constrained optimization

In this section, I give an overview of general constrained optimization, where we do not assume convexity of objective functions and feasible regions. This section follows the same presentation style as the one in [Ruszczyński, 2006].

A.3.1 Lagrangian duality and relaxation

A general constrained optimization problem can be formulated as

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0 \qquad i = 1, \ldots, p \\ & h_j(x) = 0 \qquad j = 1, \ldots, q. \end{aligned} \tag{A.3.1}$$

The Lagrangian of (A.3.1) is given by

$$L(x, \lambda, \mu) := f(x) + \sum_{i=1}^{p} \lambda_i g_i(x) + \sum_{i=1}^{q} \mu_i h_i(x).$$

The domain of this function is $\mathbb{R} \times \Lambda_0$, where $\Lambda_0 = \mathbb{R}^p_{\geq 0} \times \mathbb{R}^q$. The primal function associated with (A.3.1) is defined as

$$L_P(x) := \sup_{(\lambda,\mu)\in\Lambda_0} L(x, \lambda, \mu),$$

and the dual function is

$$L_D(\lambda, \mu) := \inf_{x\in\mathbb{R}} L(x, \lambda, \mu).$$

Additionally, the primal problem is

$$\min_{x\in\mathbb{R}} L_P(x),$$

and the dual problem is

$$\max_{(\lambda,\mu)\in\Lambda_0} L_D(\lambda, \mu).$$

If x is feasible in the original optimization problem (A.3.1), then we have $L_p(x) = f(x)$. In this case, because $g_i(x) \leq 0$ for all i and $h_i(x) = 0$ for all i, the largest value for $L_p(x)$ is obtained by substituting $\lambda_i = 0$ for all i and any value for μ_j. Consequently, we obtain $L_p(x) = f(x)$ when x is feasible. Conversely, if x is infeasible in (A.3.1), then $L_p(x) = \infty$. To sum up, we have

$$L_P(x) = \begin{cases} f(x) & \text{if } x \text{ is feasible in (A.3.1)} \\ \infty & \text{otherwise.} \end{cases}$$

Here, we adopt the convention that the minimum of the empty set is ∞; hence, (A.3.1) takes ∞ if there are no feasible points.

As for the dual function, it can be interpreted as a lower bound of the optimal objective value of (A.3.1). To illustrate this, we can first see that if x is feasible

$$\forall(\lambda, \mu) \in \Lambda_0 . f(x) \geq L(x, \lambda, \mu).$$

This yields

$$\forall(\lambda, \mu) \in \Lambda_0 . f(x) \geq \min_{y\in\mathbb{R}} L(y, \lambda, \mu) = L_D(\lambda, \mu),$$

where x is assumed to be feasible. Here, because a feasible point exists (namely x), $\min_{y\in\mathbb{R}} L(y,\lambda,\mu)$ is smaller than or equal to $L(x,\lambda,\mu)$, which is no larger than $f(x)$. Therefore, for any feasible point x,

$$f(x) \geq \max_{(\lambda,\mu)\in\Lambda_0} L_D(\lambda,\mu).$$

This means $p^* \geq d^*$, where p^* is the optimal objective value of (A.3.1) and d^* is the optimal objective value of the dual problem (that is, $\max_{(\lambda,\mu)\in\Lambda_0} L_D(\lambda,\mu)$). In fact, $p^* \geq d^*$ holds regardless of whether (A.3.1) has feasible points. If (A.3.1) has a feasible point, by the above reasoning, $p^* \geq d^*$ holds. Conversely, if we have no feasible points, $p^* = \infty$ and hence $p^* \geq d^*$ again holds. Thus, the optimal objective value of the dual problem is a lower bound on the optimal objective value of the primal problem, and this result is known as the weak duality.

The quantity $p^* - d^*$ is called a duality gap, and this is not always 0. However, under some conditions on objective functions' convexity and constraints, the duality gap is 0, and this result is called strong duality.

If strong duality holds, we can solve the dual problem to obtain an optimal objective value of the primal problem. More specifically, we first evaluate $L_D(\lambda,\mu)$ for different values of (λ,μ) and return the largest optimal objective value. Oftentimes, we only evaluate $L_D(\lambda,\mu)$ once with a fixed (λ,μ). This approach is known as Lagrangian relaxation.

A.3.2 Penalty method

One general approach to general constrained optimization is to convert a given problem to a (sequence of) unconstrained optimization problem(s). This can be achieved by embedding constraints in objective functions in the form of penalty, which is the idea behind the penalty method. The presentation style of this section is attributed to Ruszczyński [2006].

Suppose we are given

$$\min_{x\in\mathcal{X}} f(x), \tag{A.3.2}$$

where $\mathcal{X}$ is a feasible region encoded by constraints. To embed the constraints in the objective function, we create a penalty function $P : \mathbb{R}^n \to \mathbb{R}$ such that

$$\begin{aligned} P(x) &= 0 \qquad \text{if } x \in \mathcal{X} \\ P(x) &> 0 \qquad \text{otherwise.} \end{aligned}$$

The objective function is then replaced with $f(x) + \lambda P(x)$, where $\lambda > 0$ is called a penalty parameter. We consequently obtain the following unconstrained optimization problem:

$$\min_{x\in\mathbb{R}} f(x) + \lambda P(x). \tag{A.3.3}$$

The following lemma relates the solutions to (A.3.2) and (A.3.3):

Lemma A.3.1 (Lemma 6.5 in [Ruszczyński, 2006]). *If* (A.3.3) *has an optimal solution* x^* *for some* $\lambda > 0$ *such that* $x \in \mathcal{X}$*, then* x^* *is also an optimal solution to* (A.3.2)*.*

We cannot know in advance which value of λ will yield a solution that is feasible in the original constrained optimization problem. Hence, in the penalty method, we construct a sequence of values for λ and solve an unconstrained problem instance for each λ in the sequence. One concrete strategy is to keep solving penalty problems with different values for λ until an optimal solution that is also feasible in the original problem is found. Termination of this strategy is not guaranteed; i.e. it can be the case that for any λ, the associated penalty problem does not yield an optimal solution x^* that satisfies $x^* \in \mathcal{X}$. However, as stated in the following theorem, a sequence of penalty problems may yield a subsequence converging to an optimal solution of the original problem.

Theorem A.3.1 (Theorem 6.6 in [Ruszczyński, 2006]). *Suppose that* (A.3.2) *has an optimal solution. Given a sequence* $(\lambda_k)_{k\geq 1}$ *such that* $\lim_{k\to\infty} \lambda_k = \infty$*, suppose* x_k *is an optimal solution to the unconstrained optimization problem with* λ_k*. Then every accumulation point of* $(x_k)_{k\geq 1}$ *is an optimal solution to* (A.3.2)*.*

Here, an accumulation point of a sequence is defined as a limit of some subsequence. Note that this theorem does not guarantee the existence of an accumulation point in $(x_k)_{k\geq 1}$—the theorem only asserts that accumulation points, if they exist, are also optimal solutions to the original optimization problem. To ensure the existence of an accumulating point, we need to make an additional assumption. For more detail and the proof of the theorem, the reader is referred to [Ruszczyński, 2006].

One common penalty function is a quadratic function. Given an inequality constraint $g(x) \leq 0$, the associated quadratic penalty function is $(\max\{0, g(x)\})^2$. An equality constraint $h(x) = 0$ is associated with the penalty $h(x)^2$. One advantage of using a quadratic function for penalty is that $(\max\{0, g(x)\})^2$ is (continuously) differentiable (provided that g is (continuously) differentiable), unlike $\max\{0, g(x)\}$.

Finally, it is worth noting that although the penalty method resembles Lagrangian relaxation, they are distinct. In the penalty method, the terms we add to the original objective functions represent penalty for violating constraints, whereas in Lagrangian relaxation, for inequality constraints, the additional terms can represent "rewards" rather than penalty. For instance, given an inequality constraint $g(x) < 0$, Lagrangian relaxation always adds $g(x)$ to the Lagrangian, whilst in the penalty method, if $g(x) < 0$, then $P(x) = 0$ (by the definition of penalty functions); consequently, 0 is added to the objective function instead of $g(x)$.

Appendix B

Adversarial attacks and training

B.1 Adversarial attacks

B.1.1 L-BFGS

This is a targeted adversarial attack proposed by Szegedy et al. [2013].

Let F denote a classifier. Given an instance $x \in \mathcal{X}$, a concept c to be learned, and the target label $t \neq c(x)$, Szegedy et al. [2013] formulate the problem of finding (optimal) adversarial examples as follows:

$$\begin{aligned} \text{minimize} \quad & \|\delta\| \\ \text{subject to} \quad & F(x+\delta) = t \\ & x + \delta \in \mathcal{X}. \end{aligned}$$

The norm operator $\|\cdot\|$ measures the size of the perturbation δ. Common norm operators for adversarial examples in computer vision include the L_2 norm and L_∞ norm.

Note that this is by no means the only way to formulate adversarial examples. As F is usually non-convex, we generally cannot precisely solve the above optimization problems to compute (optimal) adversarial examples. Instead, the problem is "approximately" solved by a numerical optimization method called L-BFGS.

B.1.2 Fast gradient sign method (FGSM)

This is untargeted adversarial attack proposed by Goodfellow et al. [2015].

Given an instance $x \in \mathcal{X}$, a loss function ℓ, and a user-specified adversarial budget ϵ, an adversarial perturbation is given by

$$\delta = \epsilon \cdot \operatorname{sign}(\nabla_x \ell(\theta, x, y)). \tag{B.1.1}$$

In those applications where each component of a vector $x \in \mathcal{X}$ is required to fall into a range $[a, b]$ (known as a box constraint), we need to apply clipping to $x + \delta$. Clipping

is defined by $\text{clip}_{[a,b]} : x \mapsto y$ such that

$$y_i := \begin{cases} a & \text{if } x_i < a \\ x & \text{if } a \leq x \leq b \\ b & \text{if } x_i \geq b. \end{cases}$$

The rationale behind the design of FGSM is based on the observation by Goodfellow et al. [2015]: neural networks tend to be locally linear. Goodfellow et al. [2015] maintain that the local linearity is attributed to many loss functions being locally linear. For instance, ReLU is piecewise linear, and the tanh and sigmoid functions are nearly linear in the region close to the origin.

There exists a variant of FGSM for the L_2 norm, where the sign operation in (B.1.1) is replaced by projection on a unit L_2-ball around x. This variant is oftentimes called fast gradient method (FGM) because we use Euclidean projection instead of the sign function. Note that in FGSM, the sign function is not identical to the (Euclidean or L_∞) projection operator on a unit L_∞-norm ball. This is because δ does not necessarily have the same direction as $\nabla_x \ell(\theta, x, y)$; in other words, δ is not necessarily a scalar product of $\nabla_x \ell(\theta, x, y)$ when we use FGSM.

B.1.3 Projected gradient descent (PGD)

Another adversarial attack employs projected gradient descent (PGD), which is a numerical optimization technique (See Appendix A.2.2 for details of PGD in convex optimization). This attack works by iteratively making a step along the gradient of the loss and then applying projection.

More formally, assume that x is an instance from a training dataset, c is a target concept to be learned, and ℓ is a loss function. PGD is guided by

$$x_{t+1} := \Pi_\epsilon \left(x_t + \alpha \cdot \frac{\nabla_{x_t} \ell(\theta, x_t, c(x))}{\|\nabla_{x_t} \ell(\theta, x_t, c(x))\|} \right),$$

where $\|\cdot\|$ is a norm operator and Π_ϵ is a projection operator on the ball of radius ϵ (measured by $\|\cdot\|$) around x. Also, ϵ is the overall adversarial budget, and α is the size of a step made in each iteration. Both ϵ and α are parameters chosen by users. As in FGSM, if necessary, we apply clipping to x_{t+1} after the projection by Π_ϵ.

If we use the L_∞ norm in PGD and we use the sign function in place of projection, the resulting adversarial attack is an iterative variant of FGSM. This is known as iterative FGSM (IFGSM) and is first introduced in [Kurakin et al., 2017].

Further details on PGD as an optimization method will be given in Section A.2.2.

B.2 Solving adversarial training in practice

In Section 2.4.2, adversarial training is formulated in terms of robust optimization. In that section, it was explained that we usually run adversarial training by repeatedly

alternate between (i) computing adversarial examples and (ii) update the weights of a neural network accordingly. Although this workflow of adversarial training seems to be reasonable at a first glance, there is a technical issue that needs to be resolved.

Adversarial training generally solves the outer minimization in (2.4.1) using gradient descent whilst keeping δ in (2.4.2) fixed. That is, adversarial training implicitly assumes that the maximizer δ of the inner problem is independent of θ. However, because δ can be dependent on θ, solving the outer minimization problem whilst fixing the maximizer (i.e. δ) may not yield a good approximation to the robust optimization problem's solution. Assuming that we use gradient descent to solve the outer optimization problem, Madry et al. [2017] resolve this issue by resorting to Danskin's theorem. This states that 'gradients at maximizers of the inner problem correspond to descent directions for the saddle-point problem':

Theorem B.2.1 (Danskin's theorem from [Madry et al., 2017]). *Let $\mathcal{S}$ be a nonempty compact topological space. Given $g : \mathbb{R}^n \times \mathcal{S} \to \mathbb{R}$, assume that $g(\cdot, \delta)$ is differentiable for every $\delta \in \mathcal{S}$ and $\nabla_\theta g(\theta, \delta)$ is continuous on $\mathbb{R} \times \mathcal{S}$. Additionally, suppose $\delta^*(\theta) = \{\delta \in \arg\max_{\delta \in \mathcal{S}} g(\theta, \delta)\}$. Then the corresponding max-function*

$$\phi(\theta) := \max_{\delta \in \mathcal{S}} g(\theta, \delta)$$

is locally Lipschitz continuous and directionally differentiable. Further, its directional derivative along vector h is

$$\phi'(\theta, h) = \sup_{\delta \in \delta^*(\theta)} h^\intercal \nabla_\theta g(\theta, \delta).$$

In particular, if for some $\theta \in \mathbb{R}^n$ the set $\delta^*(\theta) = \{\delta_\theta^*\}$ is singleton, then the max-function is differentiable at θ and

$$\nabla \phi(\theta) = \nabla_\theta g(\theta, \delta_\theta^*). \tag{B.2.1}$$

A directional derivative of a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$ refers to the function's gradient along a given vector $h \in \mathbb{R}^n$. Note that on the right hand side of (B.2.1), the value of δ_θ^* is fixed when the gradient of $g(\theta, \delta_\theta^*)$ with respect to θ is evaluated. Therefore, (B.2.1) equates the gradient of $\phi(\theta)$, in which the maximizer δ can be dependent on θ, with the gradient of $g(\theta, \delta_\theta^*)$, in which the maximizer δ_θ^* is fixed. Employing Danskin's theorem, we obtain the following corollary.

Corollary B.2.1 (Madry et al. [2017]). *Let $\bar{\delta} \in \mathcal{S}$ be a maximizer for $\max_\delta \ell(\theta, x + \delta, y)$. Then, $-\nabla_\theta \ell(\theta, x + \bar{\delta}, y)$ is a descent direction for $\phi(\theta) = \max_{\delta \in \mathcal{S}} \ell(\theta, x + \delta, y)$, provided that $\nabla_\theta \ell(\theta, x + \bar{\delta}, y)$ is nonzero.*

Proof. This proof is attributed to Madry et al. [2017]. To apply Theorem B.2.1, set

$$g(\theta, \delta) := \ell(\theta, x + \delta, y).$$

Theorem B.2.1 yields that the directional derivative of $\phi(\theta) := \max_{\delta \in \mathcal{S}} \ell(\theta, x+\delta, y)$ in the direction of $h = \nabla_\theta \ell(\theta, x+\bar{x}, y)$ is

$$\phi'(\theta, h) = \sup_{\delta \in \delta^*(\theta)} h^\intercal \nabla_\theta \ell(\theta, x+\delta, y) \tag{B.2.2}$$

$$\geq h^\intercal h. \tag{B.2.3}$$

Thus, if $h = \nabla_\theta \ell(\theta, x+\bar{\delta}, y)$ is nonzero, (B.2.3) is positive. Hence, on the left hand side of (B.2.2), $\phi'(\theta, h)$ should also be positive. It follows that $-h$ is indeed a descent direction. □

Appendix C

Supplementary analysis

C.1 Convergence the Wasserstein distance

In this section, I will analyze the difference between MIX and OPT. This analysis is crucial because a bound on the convergence rate is necessary if we would like to know how many training examples are needed to obtain a desired distance between MIX and OPT.

To this end, because we have already derived a bound on $|\text{MIX} - \text{OPT}_{\text{empirical}}|$ in Section 3.2.1, it remains to derive a bound on $|\text{OPT} - \text{OPT}_{\text{empirical}}|$. The difference between OPT and $\text{OPT}_{\text{empirical}}$ lies in the sets of probability distributions over which the maximum of $\mathbb{E}[\ell(\theta, x, y)]$ is evaluated. In OPT, the set of probability distributions under consideration is $\{\tilde{\mathcal{D}} \mid W_p(\tilde{\mathcal{D}}, \mathcal{D}) \leq \epsilon\}$, whereas in $\text{OPT}_{\text{empirical}}$, $\{\tilde{\mathcal{D}} \mid W_p(\tilde{\mathcal{D}}, \mathcal{D}_n) \leq \epsilon\}$ is used, where $\mathcal{D}_n$ is an empirical instance distribution. I will first examine the convergence of $W_p(\mathcal{D}, \mathcal{D}_n)$ to 0 and then investigate how $W_p(\mathcal{D}, \mathcal{D}_n)$ affects $|\text{OPT} - \text{OPT}_{\text{empirical}}|$.

Research results on the convergence of $W_p(\mathcal{D}, \mathcal{D}_n)$ are already available in the literature [Fournier and Guillin, 2015; Lei, 2018]. For instance, Fournier and Guillin [2015] derive moment estimates and concentration estimates, which respectively look like

$$\mathbb{E}[W_p(\mathcal{D}, \mathcal{D}_n)] \leq \alpha(n) \qquad \text{for all } n \geq 1,$$

and

$$\mathbb{P}(W_p(\mathcal{D}, \mathcal{D}_n) \geq x) \leq \alpha(n, x) \qquad \text{for all } n \geq 1, x > 0.$$

As the exact details are complicated, I will not present them here.

Now that we have obtained a convergence rate of the Wasserstein distance, the next step is to study the relationship between $W_p(\mathcal{D}, \mathcal{D}_n)$ and $|\text{OPT} - \text{OPT}_{\text{empirical}}|$. Fix $\mathcal{D}$ and $\mathcal{D}_n$, and suppose $W_p(\mathcal{D}, \mathcal{D}_n) = \delta$. Figure C.1 depicts two uncertainty regions: $\{\tilde{\mathcal{D}} \mid W_p(\tilde{\mathcal{D}}, \mathcal{D}) \leq \epsilon\}$ and $\{\tilde{\mathcal{D}} \mid W_p(\tilde{\mathcal{D}}, \mathcal{D}_n) \leq \epsilon\}$. In this diagram, the uncertainty regions are depicted as circles whose centers are $\mathcal{D}$ and $\mathcal{D}_n$. Let A, B, and C denote probability distributions that achieve the highest values of $\mathbb{E}[\ell(\theta, x, y)]$ in respective

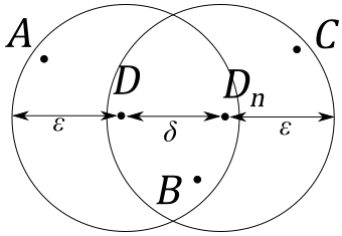

Figure C.1: Uncertainty regions around $\mathcal{D}$ and $\mathcal{D}_n$

regions. Consequently, we have

$$\begin{aligned}\text{OPT} &= \max\left\{\mathop{\mathbb{E}}_{A}[\ell(\theta,x,y)], \mathop{\mathbb{E}}_{B}[\ell(\theta,x,y)]\right\}\\ \text{OPT}_{\text{empirical}} &= \max\left\{\mathop{\mathbb{E}}_{B}[\ell(\theta,x,y)], \mathop{\mathbb{E}}_{C}[\ell(\theta,x,y)]\right\}.\end{aligned}$$

Let B' denote a probability distribution from $\{\tilde{\mathcal{D}} \mid W_p(\tilde{\mathcal{D}},\mathcal{D}) \leq \epsilon, W_p(\tilde{\mathcal{D}},\mathcal{D}_n) \leq \epsilon\}$ that is the closest to A in terms of the W_p distance. Since $W_p(\mathcal{D},\mathcal{D}_n) = \delta$, we have $W_p(A,B') \leq \delta$. This can be formally proved as follows.

Proposition C.1.1. *Let X be a Polish space and we consider probably distributions on X. Assume that $W_p(\mathcal{D},\mathcal{D}_n) = \delta$. Further, given a probability distribution A, assume that $W_p(A,\mathcal{D}) \leq \epsilon < W_p(A,\mathcal{D}_n)$. If B' is the closest probably to A (with respect to the W_p distance) such that $W_p(B',\mathcal{D}_n) \leq \epsilon$, then $W_p(A,B') \leq \delta$.*

Proof. Since the Wasserstein distance is a distance metric, the triangle inequality holds,. This gives

$$\begin{aligned}W_p(A,\mathcal{D}_n) &\leq W_p(A,\mathcal{D}) + W_p(\mathcal{D},\mathcal{D}_n)\\ &\leq \epsilon + \delta.\end{aligned}$$

I will first focus on the case of $p = 1$. According to Section 7.4 of [Villani, 2003], for any cost function c (in Definition 3.1.2), any probability measures μ_1,μ_2,ν_1,ν_2 and any $\alpha \in [0,1]$, we have

$$W_1(\alpha\mu_1 + (1-\alpha)\mu_2, \alpha\nu_1 + (1-\alpha)\nu_2) \leq \alpha W_1(\mu_1,\nu_1) + (1-\alpha)W_1(\mu_2,\nu_2),$$

where these four probably measures are defined on a Polish space. Applying this inequality to our setting, we obtain

$$\begin{aligned}W_1(\alpha A + (1-\alpha)\mathcal{D}_n, \alpha\mathcal{D}_n + (1-\alpha)\mathcal{D}_n) &\leq \alpha W_1(A,\mathcal{D}_n) + (1-\alpha)W_1(\mathcal{D}_n,\mathcal{D}_n)\\ \therefore W_1(\alpha A + (1-\alpha)\mathcal{D}_n, \mathcal{D}_n) &\leq \alpha W_1(A,\mathcal{D}_n).\end{aligned}$$

If we set $B'' = \alpha A + (1-\alpha)\mathcal{D}_n$ and $\alpha = \frac{\epsilon}{W_1(A,\mathcal{D}_n)}$, it gives

$$W_1(B'',\mathcal{D}_n) \leq \epsilon.$$

By a similar argument, we can prove

$$W_1(B'', A) \leq \delta.$$

Therefore, by definition, we also have $W_1(B', A) \leq \delta$.

Finally, to prove the general case of W_p, as explained in Section 3.1, we observe $W_{p,d}(X, Y) = W_{1,d^p}(X, Y)^{\frac{1}{p}}$, where the second subscript of W denotes the underlying distance metric. Hence, we can translate, for example, $W_{p,d}(A, \mathcal{D}) \leq \epsilon$ into $W_{1,d^p}(A, \mathcal{D}) \leq \epsilon^p$. As a consequence, we can lift the above result of $W_1(B', A) \leq \delta$ into

$$W_{1,d^p}(B', A) \leq \delta^p$$
$$\therefore W_{p,d}(B', A) \leq \delta.$$

This concludes the proof. □

As $W_p(A, B') \leq \delta$, we have

$$\int d(x, x')^p \, \mathrm{d}\gamma(x, x') \leq \delta^p \tag{C.1.1}$$

for some coupling $\gamma \in \Pi(A, B')$ such that γ does not change labels. Fix this coupling γ. Here, $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{\geq 0}$ is a metric on $\mathcal{X}$. Since $x \mapsto x^p$ for any $p \in [1, \infty)$ is a convex function on the domain $[0, \infty)$ and $d(\cdot, \cdot)$ always returns non-negative values, it follows from Jensen's inequality that

$$\left(\int d(x, x') \, \mathrm{d}\gamma(x, x') \right)^p \leq \int d(x, x')^p \, \mathrm{d}\gamma(x, x'). \tag{C.1.2}$$

Combining (C.1.1) and (C.1.2) yields

$$\int d(x, x') \, \mathrm{d}\gamma(x, x') \leq \delta. \tag{C.1.3}$$

The same result can be obtained for the case of $p = \infty$.

Following the convention used in [Staib and Jegelka, 2017], I assume the L_q norm is used for the underlying metric d. For any $q \in [1, \infty]$ and any $x, x' \in \mathbb{R}^m$, we have

$$\frac{1}{\sqrt{m}} \left\| x - x' \right\|_2 \leq d(x, x') \leq \sqrt{m} \left\| x - x' \right\|_2. \tag{C.1.4}$$

This can be easily seen from Figure C.2, which illustrates the relationship between the L_1, L_2, and L_∞ norms in a two-dimensional space. As p increases, the unit circle of L_p approaches a square of radius one. By comparing the unit circles of the three L_p norms, it is straightforward to see that $\frac{\|x-x'\|_1}{\|x-x'\|_2} \leq \sqrt{2}$ and $\frac{\|x-x'\|_2}{\|x-x'\|_\infty} \leq \sqrt{2}$ in $\mathbb{R}^2$. These two upper bounds are generalized to $\sqrt{m}$ if the space we work with has the dimension m. Consequently, (C.1.4) is yielded.

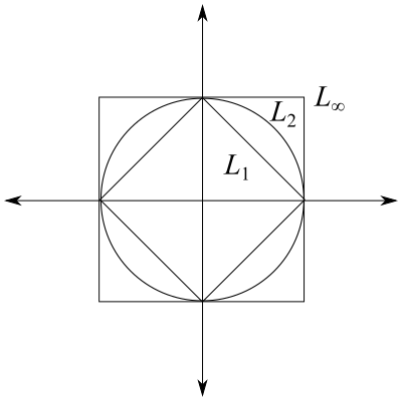

Figure C.2: Unit circles with respect to the L_1, L_2, and L_∞ norms

By (C.1.4), we obtain

$$\frac{1}{\sqrt{m}} \int \left\| x - x' \right\|_2 \mathrm{d}\gamma(x, x') \leq \int d(x, x') \, \mathrm{d}\gamma(x, x').$$

Combining this with (C.1.3) gives

$$\int \left\| x - x' \right\|_2 \mathrm{d}\gamma(x, x') \leq \delta\sqrt{m}. \tag{C.1.5}$$

Because a Lipschitz constant of $\ell(\theta, \cdot, y)$ is L for any y, $|\ell(\theta, x, y) - \ell(\theta, x', y)| \leq L \left\| x - x' \right\|_2$ holds, provided that we use the L_2 norm in the definition of Lipschitz continuity. It follows from (C.1.5) that

$$\left| \mathop{\mathbb{E}}_{A}[\ell(\theta, x, y)] - \mathop{\mathbb{E}}_{B'}[\ell(\theta, x, y)] \right| \leq L\delta\sqrt{m}.$$

By the definition of B, we have $\mathbb{E}_{B'}[\ell(\theta, x, y)] \leq \mathbb{E}_B[\ell(\theta, x, y)]$. Hence, $\mathbb{E}_B[\ell(\theta, x, y)] \leq \mathbb{E}_A[\ell(\theta, x, y)] + L\delta\sqrt{m}$ must hold. By the same reasoning, we have $\mathbb{E}_B[\ell(\theta, x, y)] \leq \mathbb{E}_C[\ell(\theta, x, y)] + L\delta\sqrt{m}$. Therefore, by case analysis, we derive

$$|\mathrm{OPT} - \mathrm{OPT}_{\mathrm{empirical}}| \leq L\delta\sqrt{m}.$$

C.2 Relationship between ϵ and γ

In a number of adversarial training procedures and adversarial attacks, ϵ is commonly used as a robustness parameter (specifically, adversarial budget). On the other hand, in WRM by Sinha et al. [2018], the relaxation parameter for the Lagrangian relaxation, denoted by γ, is used to regulate robustness. In this section, I discuss the relationship between these two parameters.

Recall that DRO with the Wasserstein distance is formulated as

$$\min_{\theta} \max_{\tilde{\mathcal{D}}: W_1(\mathcal{D}, \tilde{\mathcal{D}}) \leq \epsilon} \mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}} [\ell(\theta, x, y)], \tag{C.2.1}$$

where ϵ is a robustness parameter specified by the user and $\mathcal{D}$ is a distribution over $\mathcal{X} \times \mathcal{Y}$. Sinha et al. [2018] applies the Lagrangian relaxation to (C.2.1), transforming the inner maximization problem to

$$\sup_{\tilde{\mathcal{D}}} \left[\mathop{\mathbb{E}}_{(x,y)\in\tilde{\mathcal{D}}} [\ell(\theta, x, y)] - \lambda W_1(\tilde{\mathcal{D}}, \mathcal{D}) \right], \tag{C.2.2}$$

where γ is a penalty parameter. Due to the duality result (See Theorem 3.2.2), we have (3.2.5).

Thus, one approach to (numerically) solving the inner problem of (C.2.1) is to solve (C.2.2) multiple times, each with a different value for γ, and pick the one with the best objective value. In the specific context of adversarial training, instead of selecting optimal γ in terms of $\lambda\epsilon + \mathbb{E}_{(x,y)\sim\mathcal{D}}[\phi_\lambda(\theta, x, y)]$ appearing in (3.2.5), it may be more sensible to select γ that results in the highest robustness to adversarial attacks. Staib and Jegelka [2017] adopt this approach in their experiments for comparing DRO methods (e.g. the Frank-Wolfe method-based DRO) that use ϵ and the Lagrangian relaxation-based method, which uses γ instead of ϵ. They tested γ drawn from

$$\{0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1\},$$

where the ratios between two consecutive values are approximately the same. They report that $\gamma = 0.001$ produced the best performance against an IFGSM adversary.

However, this (supposedly optimal) value of γ is much lower than a typical value used in the experiments by Sinha et al. [2018]. For instance, $\gamma = 2$ is used for a synthetic dataset that has a circular decision boundary, and approximately $\gamma = 0.37$ is used for MNIST. In [Sinha et al., 2018], (C.2.2) is solved using stochastic gradient descent (SGD), and they derive the convergence bound under several assumptions on the loss function ℓ, the cost function c, and the value of γ. Specifically, Sinha et al. [2018] assume that γ is sufficiently large, which corresponds to a sufficiently large value of ϵ. The inverse relationship between ϵ and γ in effect follows from the duality result in (3.2.5).

In order to illustrate this inverse relationship more explicitly, we substitute (3.2.6) into (3.2.5), yielding

$$\sup_{\tilde{\mathcal{D}}: W_1(\tilde{\mathcal{D}}, \mathcal{D}) \leq \epsilon} \mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}} [\ell(\theta, x, y)] = \inf_{\lambda \geq 0} \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\sup_{x' \in \mathcal{X}} \left[\ell(\theta, x', y) - \lambda(c(x, x') - \epsilon) \right] \right], \tag{C.2.3}$$

Here, for simplicity, the cost function is $c : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^{\geq 0}$ instead of $c : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}^{\geq 0} \cup \{\infty\}$. In other words, I adopt the standard convention that $c((x, y), (x', y')) = \infty$ whenever $y \neq y'$.

If ϵ increases, the left hand side will likely increase as well (or at least remains the same). Thus, the right hand side must not decrease if ϵ increases. On the right

hand side, $c(x, x') - \epsilon$ decreases as ϵ increases. Hence, γ needs to decrease such that $\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\sup_{(x',y')}\left[\ell(\theta, x', y') - \lambda(c(x, x') - \epsilon)\right]\right]$ does not decrease. Therefore, ϵ and γ are inversely related.

Additionally, the theory underlying WRM suggests that if γ is sufficiently large, the resulting optimisation problem is strongly concave and hence can be approximately solved with good accuracy by gradient descent. This is in accord with the experiment results of [Sinha et al., 2018] (see Sections A.3 and A.4 of [Sinha et al., 2018]). As γ decreases, the performance of the Lagrangian relaxation-based DRO becomes less effective in the sense that it becomes more similar to the performance of other adversarial training procedures. Hence, the Lagrangian relaxation-based DRO is meaningful only when γ is large (or equivalently, ϵ is small).

In [Staib and Jegelka, 2017], the optimal γ is quite small A neural network trained by WRM with a small value of γ is unlikely to faithfully exhibit the true nature of WRM. Thus, it is worthwhile to investigate whether the same optimal value of γ can be obtained.

Sinha et al. [2018], on the other hand, relate ϵ and γ in the following way:

$$\epsilon^2 := \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_n}[c(T_\gamma(\theta_{\mathrm{WRM}}, x, y), x)], \tag{C.2.4}$$

where $T_\gamma(\theta, x, y)$ returns the maximizer of (3.2.6); that is,

$$T_\gamma(\theta, x, y) := \arg\max_{x'\in\mathcal{X}}\{\ell(\theta, x', y) - \gamma c(x, x')\}. \tag{C.2.5}$$

Also, θ_{WRM} refers to the parameters of a neural network trained using the Lagrangian relaxation-based DRO and $\mathcal{D}_n$ is the empirical distribution. The justification of (C.2.4) follows from the next proposition:

Proposition C.2.1 (Sinha et al. [2018])**.** *Suppose*

$$P(\theta) := \frac{1}{n}\sum_{i=1}^{n}\delta_{T_\gamma(\theta, x_i, y_i)} \tag{C.2.6}$$

$$= \arg\max_{\tilde{\mathcal{D}}}\left\{\mathop{\mathbb{E}}_{(x,y)\sim\tilde{\mathcal{D}}}[\ell(\theta, x, y)] - \gamma W_1(\tilde{\mathcal{D}}, \mathcal{D}_n)\right\}, \tag{C.2.7}$$

where $\{(x_i, y_i)\}_{i=1}^n$ is a training sample and δ_z denotes a probability distribution with a unit mass as point z. Strictly speaking, the second line is ill-defined because it does not return a unique probability distribution. However, since this abuse of notation does not cause confusion, I will adopt this notational convention. Then for any parameters θ, we have

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}_n}[c(T_\gamma(\theta, x, y), x)] = W_1(P(\theta), \mathcal{D}_n). \tag{C.2.8}$$

Proof. By Theorem 3.2.2, for any probability distributions A and B, we have

$$\sup_A\left\{\mathop{\mathbb{E}}_{(x,y)\sim A}[\ell(\theta, x, y)] - \gamma W_1(A, B)\right\} = \mathop{\mathbb{E}}_{(x,y)\sim B}[\phi_\gamma(\theta, x, y)],$$

where ϕ_γ is given in (3.2.6). By (C.2.7), we obtain

$$\mathop{\mathbb{E}}_{(x,y)\sim P(\theta)}[\ell(\theta,x,y)] - \gamma W_1(P(\theta),\mathcal{D}_n) = \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\phi_\gamma(\theta,x,y)] \tag{C.2.9}$$

$$\therefore \gamma W_1(P(\theta),\mathcal{D}_n) = \mathop{\mathbb{E}}_{(x,y)\sim P(\theta)}[\ell(\theta,x,y)] - \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\phi_\gamma(\theta,x,y)]. \tag{C.2.10}$$

Therefore, we have

$$\begin{aligned}
&\mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[c(T_\gamma(\theta,x,y),x)] = W_1(P(\theta),\mathcal{D}_n) \\
\iff &\gamma \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[c(T_\gamma(\theta,x,y),x)] = \gamma W_1(P(\theta),\mathcal{D}_n) \\
\iff &\mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\gamma c(T_\gamma(\theta,x,y),x)] = \mathop{\mathbb{E}}_{(x,y)\sim P(\theta)}[\ell(\theta,x,y)] - \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\phi_\gamma(\theta,x,y)] \quad \text{by (C.2.10)} \\
\iff &\mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\gamma c(T_\gamma(\theta,x,y),x)] + \mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\phi_\gamma(\theta,x,y)] = \mathop{\mathbb{E}}_{(x,y)\sim P(\theta)}[\ell(\theta,x,y)] \\
\iff &\mathop{\mathbb{E}}_{(x,y)\sim \mathcal{D}_n}[\ell(\theta,T_\gamma(\theta,x,y),y)] = \mathop{\mathbb{E}}_{(x,y)\sim P(\theta)}[\ell(\theta,x,y)],
\end{aligned}$$

where the last line follows from the definitions of ϕ_γ and T_γ given in (3.2.6) and (C.2.5). Finally, it follows from the definition of $P(\theta)$ in (C.2.6) that the last equality holds. Therefore, the claim (C.2.8) is true. □

On the right hand side of (C.2.8), $P(\theta)$ is an adversarial probability distribution that achieves the highest objective value, where the objective function is derived from (C.2.1) by Lagrangian relaxation. To put it differently, $P(\theta)$ is an "relaxed/empirical" estimate of $\arg\max_{\tilde{\mathcal{D}}:W_1(\mathcal{D}_n,\tilde{\mathcal{D}})\leq\epsilon} \mathbb{E}_{(x,y)\sim\tilde{\mathcal{D}}}[\ell(\theta,x,y)]$. Therefore, it is reasonable to take the distance between $\mathcal{D}_n$ and $P(\theta)$ as a measure of the robustness achieved by a trained neural neural network with the parameters θ.

C.3 Comparison of FWDRO and WRM

In this section, I compare FWDRO, a DRO algorithm developed by Staib and Jegelka [2017] and based on the Frank-Wolfe method, and WRM, a DRO algorithm developed by Sinha et al. [2018] and based on Lagrangian relaxation.

C.3.1 Theoretical formulations

One similarity between FWDRO and WRM is that both of them solve the inner maximization problem (i.e. finding adversarial examples) and the outer minimization problem (i.e. optimizing model parameters) alternately. As for the outer optimization problem, both of the two approaches use stochastic (projected) gradient descent.

Thus, they differ in the way the inner optimization problem is approximately solved. In FWDRO, the empirical approximation to DRO is directly solved by means of the Frank-Wolfe method. Hence, this approach retains the constraint of the inner problem. By contrast, in WRM, Lagrangian relaxation is applied in order to remove the constraint

of the inner optimization problem As a result, the inner optimization problem becomes unconstrained. The empirical formulation of DRO that this approach induces is equivalent to the Lagrangian relaxation of RO. Hence, in a sense, WRM loses some flavour of distributional robustness.

C.3.2 Empirical performance

Staib and Jegelka [2017] conduct experiments to compare the performance of different approaches to adversarial training. Specifically, they empirically study the following methods:

- Empirical risk minimization (ERM)
- Fast DRO (FDRO) and Frank-Wolfe DRO (FWDRO): these are the empirical approximations to DRO developed by Staib and Jegelka [2017]. FDRO only makes one step in the Frank-Wolfe method (see line 5 of Algorithm 1), whereas FWDRO makes multiples steps as in the standard Frank-Wolfe method.
- WRM
- Fast gradient sign method (FGSM) and iterative FGSM (IFGSM): they first generate adversarial examples using FGSM and IFGSM (See Section 2.4.1), respectively, and then train neural networks using the newly generated adversarial examples in addition to original training data. For convenience, when there is no risk of ambiguity, I will refer to the corresponding adversarial training procedures as FGSM and IFGSM, although they technically refer to adversarial attacks rather than defenses.

Staib and Jegelka [2017] compare these methods in terms of their defenses against an IFGSM adversary on MNIST, and the above methods are ranked as follows in the descending order of defense success rates:

1. IFGSM
2. FGSM and FWDRO (roughly tied)
3. WRM and FDRO. They are tied in one experiment, but in a different experiment that is more favorable to WRM, WRM performs better than FDRO.
4. ERM.

Staib and Jegelka [2017] remark that IFGSM performs better than FWDRO possibly because FWDRO does not reach optimality or because DRO "tolerates more pointwise loss to counteract the stronger adversary".

Sinha et al. [2018] also conduct experiments to assess the performance of different adversarial training procedures. Since they use $c((x,y),(x',y')) = \|x - x'\|_2^2 + \infty \cdot \mathbb{1}_{y \neq y'}$, their formulation of empirical DRO is equivalent to the formulation by Staib and Jegelka

[2017] using the W_2 and L_2 distances. In fact, in this case, since projection on an $L_{2,2}$-ball can be computed efficiently, we can directly solve (3.2.2) rather than using FWDRO or WRM. Although Sinha et al. [2018] use the W_2 with L_2 norms for the defense, they consider both the L_2 and L_∞ norms when launching adversarial attacks.

Their experiment results confirm that WRM only works well for large λ, which corresponds to the robustness guarantee with small ϵ. WRM empirically outperforms other approaches when they only defend against adversarial examples that are nearly imperceptible to the human eye. However, when ϵ is sufficiently large (and hence the corresponding λ becomes small), the superiority of WRM diminishes.

Comparing WRM with other methods, Sinha et al. [2018] report that WRM performs better than FGSM-based and IFGSM-based adversarial training procedures, provided that the adversary's budged is small. This contradicts the experiment result by Staib and Jegelka [2017], who report that IFGSM performs better than WRM under the adversary of IFGSM. It could be the case that Staib and Jegelka [2017] and Sinha et al. [2018] use different robustness levels for other adversarial training methods when comparing them with WRM. In addition, in [Sinha et al., 2018], in many test cases (including the cases where IFGSM is an adversary), FGSM performs better than IFGSM, which contradicts the report of Staib and Jegelka [2017].

Appendix D

Supplementary experiment results

This chapter contains supplementary experiment results for Section 5.2.

D.1 Distributional PGD attacks

Overview

In Section 5.2.2, we use FGSM and PGD attacks to evaluate ten values of γ for WRM. However, although it is a common practice in the literature to evaluate the robustness of a neural networks using pointwise attacks (e.g. FGSM and PGD), DRO algorithms, including WRM, are meant to defend against "distributional" adversarial perturbations instead of pointwise attacks. To launch distributional adversarial attacks, we are required to solve (3.2.1). It can be approximated by (3.2.2), which can in turn be approximately solved by PGD. This is precisely what is done in distributional PGD to solve the inner maximization problem of DRO in (3.1.1).

Regarding hyper-parameters, the distributional distance is the 2-Wasserstein distance, and the pointwise distance function is L_2. Forty-one evenly separated values of ϵ are sampled from the range $[0, 10.0]$. The size of each batch used to create distributional perturbations is 512, and 40 iterations are executed for each batch. As indicated in Section 5.1, 15 iterations are used when training neural networks by means of distributional PGD; however, when distributional PGD is used for attacks, the number of iterations is increased from 15 to 40.

Results

The experiment results are presented in Figure D.1.

The adversarial attack success rate sometimes slightly drops even when the adversarial budget ϵ increases. This is probably due to the stochastic nature of distributional PGD: the dataset is randomly divided into batches.

It is worth observing that the adversarial attack success rates are noticeably low. For instance, when $\epsilon = 4.0$, the attack success rate is less than 0.05 whether we use ReLU or ELU. On the other hand, if we use PGD instead of distributional PGD to attack neural

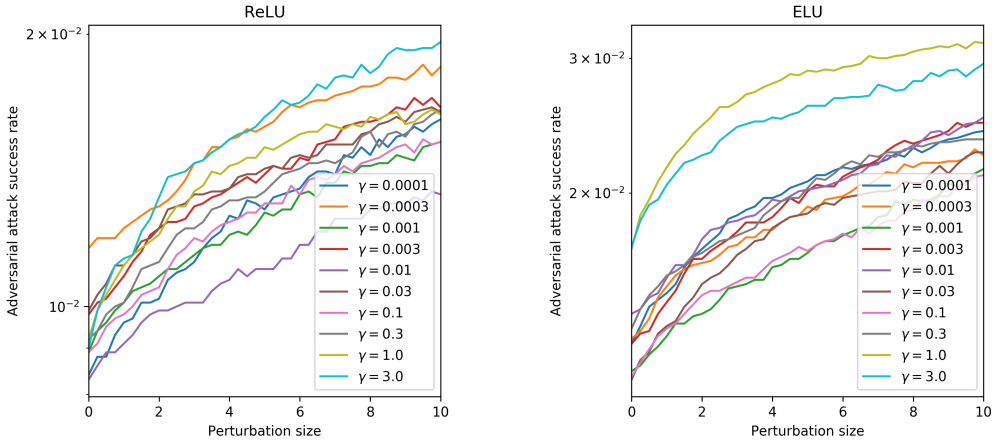

Figure D.1: Robustness of WRM-trained models to distributional PGD attacks

networks, as shown by Figure D.3, the attack success rate at $\epsilon = 4.0$ is more than 0.1. Hence, distributional PGD is deficient as a (distributional) adversarial attack algorithm.

The poor performance of distributional PGD as an adversarial attack ties in with its performance as a training algorithm compared to FWDRO and WRM (See Section 5.2.3).

D.2 Comparison of DRO models using PGD attacks

The robustness results of WRM-trained neural networks with various values of γ to PGD attacks are presented in Figure D.2 and Figure D.3.

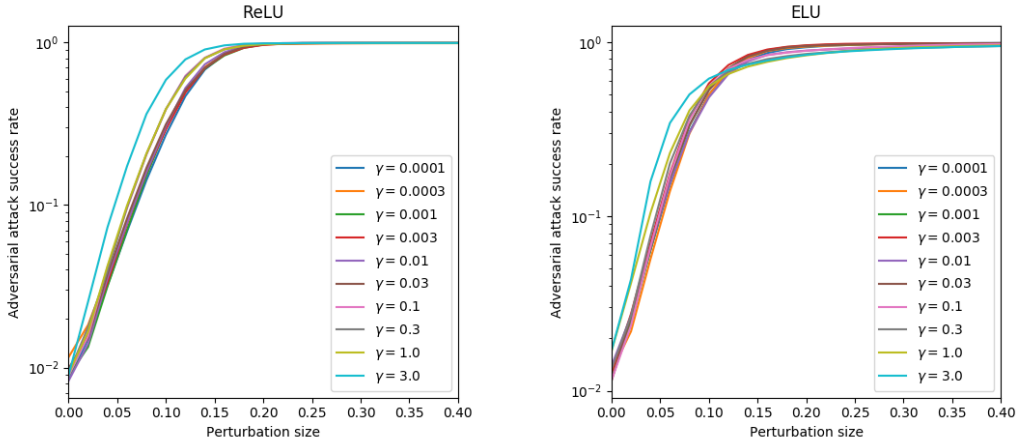

Figure D.2: Robustness of WRM-trained models to PGD with the L_∞ norm

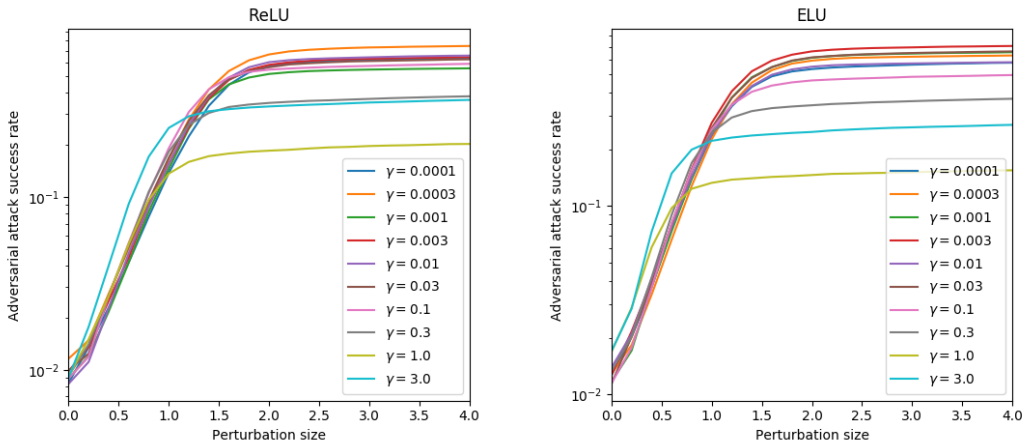

Figure D.3: Robustness of WRM-trained models to PGD with the L_2 norm

Bibliography

Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018. ISSN 2169-3536. doi: 10.1109/access.2018.2807385. URL http://dx.doi.org/10.1109/ACCESS.2018.2807385. 13

Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton University Press, 2009. 9

Aharon Ben-Tal, Dick Den Hertog, Anja De Waegenaere, Bertrand Melenberg, and Gijs Rennen. Robust solutions of optimization problems affected by uncertain probabilities. *Management Science*, 59(2):341–357, 2013. 7

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. *Lecture Notes in Computer Science*, page 387–402, 2013. ISSN 1611-3349. doi: 10.1007/978-3-642-40994-3_25. URL http://dx.doi.org/10.1007/978-3-642-40994-3_25. 6

Jose Blanchet and Karthyek RA Murthy. Quantifying distributional model risk via optimal transport. *arXiv preprint arXiv:1604.01446*, 2016. 21

Jose Blanchet, Yang Kang, and Karthyek Murthy. Robust Wasserstein profile inference and applications to machine learning. *arXiv preprint arXiv:1610.05627*, 2016. 7, 17

Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015. 48, 50, 51, 52, 53, 54

Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints, 2018. URL https://arxiv.org/abs/1805.10204. 6

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017*, pages 39–57, 2017. 5, 6, 7, 8, 13, 24, 25, 26, 27, 28, 37, 38, 39, 40, 44, 45

J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848. 5

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 272–279, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390191. URL http://doi.acm.org/10.1145/1390156.1390191. 19

John Duchi, Peter Glynn, and Hongseok Namkoong. Statistics of robust optimization: A generalized empirical likelihood approach. *arXiv preprint arXiv:1610.03425*, 2016. 7

Nicolas Fournier and Arnaud Guillin. On the rate of convergence in Wasserstein distance of the empirical measure. *Probability Theory and Related Fields*, 162(3):707–738, Aug 2015. ISSN 1432-2064. doi: 10.1007/s00440-014-0583-7. 62

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956. doi: 10.1002/nav.3800030109. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030109. 52

Rui Gao and Anton J Kleywegt. Distributionally robust stochastic optimization with Wasserstein distance. *arXiv preprint arXiv:1604.02199*, 2016. 7, 17, 18

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572. 4, 5, 6, 7, 58, 59

Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015. 14

Jörn-Henrik Jacobsen, Jens Behrmann, Richard Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=BkfbpsAcF7. 7

Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *Schedae Informaticae*, 1/2016, 2017. ISSN 2083-8476. doi: 10.4467/20838476si.16.004.6185. URL http://dx.doi.org/10.4467/20838476SI.16.004.6185. 7

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. 13

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257. 4

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *ICLR Workshop*, 2017. URL https://arxiv.org/abs/1607.02533. 5, 59

Simon Lacoste-Julien. Convergence rate of frank-wolfe for non-convex objectives. *arXiv preprint arXiv:1607.00345*, 2016. 54

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–444, 2015. URL https://www.nature.com/articles/nature14539. 11, 12

Jing Lei. Convergence and concentration of empirical measures under Wasserstein distance in unbounded functional spaces. *arXiv preprint arXiv:1804.10556*, 2018. 62

Chunchuan Lyu, Kaizhu Huang, and Hai-Ning Liang. A unified gradient regularization family for adversarial examples. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 301–309. IEEE, 2015. 14

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *CoRR*, 2017. 7, 14, 24, 60

Peyman Mohajerin Esfahani and Daniel Kuhn. Data-driven distributionally robust optimization using the Wasserstein metric: performance guarantees and tractable reformulations. *Mathematical Programming*, 171(1):115–166, Sep 2018. ISSN 1436-4646. doi: 10.1007/s10107-017-1172-1. URL https://doi.org/10.1007/s10107-017-1172-1. 7, 17

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012. ISBN 978-0-262-01825-8. URL http://mitpress.mit.edu/books/foundations-machine-learning-0. 10

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. 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, 2016. 7

Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v0.8.0. *CoRR*, 1807.01069, 2018. URL https://arxiv.org/pdf/1807.01069. 30

Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer New York, 1999. ISBN 9780387987934. 52

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016*, pages 372–387, 2016a. 7

Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy, SP 2016*, pages 582–597, 2016b. 7, 24

Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic frank-wolfe methods for nonconvex optimization. *arXiv preprint arXiv:1607.08254*, 2016. 54

Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Comput.*, 16(5):1063–1076, May 2004. ISSN 0899-7667. doi: 10.1162/089976604773135104. URL http://dx.doi.org/10.1162/089976604773135104. 7

Andrzej Ruszczyński. *Nonlinear Optimization*. Nonlinear optimization. Princeton University Press, 2006. ISBN 9780691119151. 54, 56, 57

Sean Saito and Sujoy Roy. Effects of loss functions and target representations on adversarial robustness, 2018. 7

Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015. 14

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. URL https://www.nature.com/articles/nature16961. 4

Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. In *ICLR*, 2018. 5, 6, 7, 8, 15, 16, 21, 22, 23, 27, 29, 30, 32, 33, 34, 35, 36, 38, 44, 65, 66, 67, 68, 69, 70

Suvrit Sra. Fast projections onto mixed-norm balls with applications. *Data Mining and Knowledge Discovery*, 25(2):358–377, 2012. 19

Matthew Staib and Stefanie Jegelka. Distributionally robust deep learning as a generalization of adversarial training. In *NIPS*, 2017. 5, 6, 7, 8, 15, 16, 17, 18, 19, 20, 23, 26, 29, 30, 31, 32, 33, 34, 35, 36, 38, 44, 64, 66, 67, 68, 69, 70

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, 2013. URL http://arxiv.org/abs/1312.6199. 4, 5, 6, 7, 58

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. 13

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SyxAb30cY7. 35

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, November 1984. ISSN 0001-0782. doi: 10.1145/1968.1972. URL http://doi.acm.org/10.1145/1968.1972. 10

Vladimir Naumovich Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999. 11

Cédric Villani. *Topics in Optimal Transportation*. Graduate studies in mathematics. American Mathematical Society, 2003. ISBN 9780821833124. 15, 63

Cédric Villani. *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg, 2008. ISBN 9783540710493. 16

Stephen J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151 (1):3–34, Jun 2015. ISSN 1436-4646. doi: 10.1007/s10107-015-0892-3. URL https://doi.org/10.1007/s10107-015-0892-3. 54

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016. 4