Robust Machine Learning Models and Their Applications

by

Hongge Chen

B.E., Tsinghua University (2015)
S.M., Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

Author .....
Department of Electrical Engineering and Computer Science
November 25, 2020

Certified by .....
Duane S. Boning
Clarence J. LeBel Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Robust Machine Learning Models and Their Applications

by

Hongge Chen

Submitted to the Department of Electrical Engineering and Computer Science
on November 25, 2020, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Recent studies have demonstrated that machine learning models are vulnerable to adversarial perturbations – a small and human-imperceptible input perturbation can easily change the model output completely. This has created serious security threats to many real applications, so it becomes important to formally verify the robustness of machine learning models. This thesis studies the robustness of deep neural networks as well as tree-based models, and considers the applications of robust machine learning models in deep reinforcement learning.

We first develop a novel algorithm to learn robust trees. Our method aims to optimize the performance under the worst case perturbation of input features, which leads to a max-min saddle point problem when splitting nodes in trees. We propose efficient tree building algorithms by approximating the inner minimizer in this saddle point problem, and present efficient implementations for classical information gain based trees as well as state-of-the-art tree boosting models such as XGBoost. Experiments show that our method improve the model robustness significantly.

We also propose an efficient method to verify the robustness of tree ensembles. We cast the tree ensembles verification problem as a max-clique problem on a multi-partite graph. We develop an efficient multi-level verification algorithm that can give tight lower bounds on robustness of decision tree ensembles, while allowing iterative improvement and termination at any-time. On random forest or gradient boosted decision trees models trained on various datasets, our algorithm is up to hundreds of times faster than the previous approach that requires solving a mixed integer linear programming, and is able to give tight robustness verification bounds on large ensembles with hundreds of deep trees.

For neural networks, we contribute a number of empirical studies on the practicality and the hardness of adversarial training. We show that even with adversarial defense, a model's robustness on a test example has a strong correlation with the distance between that example and the manifold of training data embedded by the network. Test examples that are relatively far away from this manifold are more likely to be vulnerable to adversarial attacks. Consequentially, we demonstrate that an adversarial training based defense is vulnerable to a new class of attacks, the "blind-spot attack,"

where the input examples reside in low density regions ("blind-spots") of the empirical distribution of training data but are still on the valid ground-truth data manifold.

Finally, we apply neural network robust training methods to deep reinforcement learning (DRL) to train agents that are robust against perturbations on state observations. We propose the state-adversarial Markov decision process (SA-MDP) to study the fundamental properties of this problem, and propose a theoretically principled regularization which can be applied to different DRL algorithms, including deep Q networks (DQN) and proximal policy optimization (PPO). We significantly improve the robustness of agents under strong white box adversarial attacks, including new attacks of our own.

Thesis Supervisor: Duane S. Boning
Title: Clarence J. LeBel Professor of Electrical Engineering and Computer Science

Acknowledgments

My PhD studies and this thesis would have been impossible without the support of many people. Through my five-year "PhD grind," I have come to realize that each person I worked with, talked with, or met with has changed the trajectory of my life.

First, I am very fortunate to be advised by Professor Duane Boning. I am extremely grateful for this opportunity he provided to work with him. My advisor searching process was not very smooth before I came to MIT. Once at MIT, it was the freedom, support, and consistent encouragement he provided that paved the road for my PhD studies. In the first two years, I was not sure what to do for my PhD, and he connected me with different people and let me try various topics. He always encouraged me to take classes in each semester to learn new things. Beyond the detailed suggestions he has given to my research, he has also given me much valuable advice, both for my professional career and for my outlook on life. I still remember our chats in his office and his effort to help me polish my papers in the late nights before deadlines. His passion for research and perfection is something I will remember for all my life.

I would also like to thank my committee members Professor Luca Daniel and Professor Devavrah Shah for their advice and comments on my research and this thesis.

I am also grateful to all my collaborators, especially Huan and Cho, who provided me with tremendous help and inspired me a lot. I would like to thank my collaborators: Ciprian Chelba, Pin-Yu Chen, Cho-Jui Hsieh, Sanjiv Kumar, Yang Li, Sijia Liu, Si Si, Zhao Song, Dong Su, Yihan Wang, Lily Weng, Chaowei Xiao, Kaidi Xu, Jinfeng Yi, and Huan Zhang.

During several summers, I interned at different places where I was fortunate to connect with many people who mentored me or helped me in my career. I really appreciate your mentorship and help: Ciprian Chelba, Jayant Kalagnanam, Guowang Li, Yang Li, Jack Lu, Lam M. Nguyen, Deval Patel, Si Si, and Kyongmin Yeo.

A big thank you to all current and past members of the Boning group, as well as my friends. I have greatly benefited from my conversations with Changchen Chen,

Jingkai Chen, Robert Chen, Siyuan Dong, Sally I Elhenawy, Dylan Grullon, Jinchi Han, Chris Lang, John Lee, Chengtao Li, Ruizhi Liao, Ge Liu, Jiaming Luo, Hongzi Mao, Jie Mei, Daniel Moon, Guannan Qu, Fan-Keng Sun, Jun Wan, Tian Xie, Zhi Xu, Guo Zhang, Zhengxing Zhang, and Xijia Zheng. My special thanks to the support and encouragement from Jie, Changchen, Siyuan and my alumni mentor Robert during my hard times. I would also like to thank MIT-CHIEF. I learned a lot about entrepreneurship during my three years in this community.

Additionally, I am grateful to Professor Shan X. Wang from Stanford, as well as Professor Xinjie Yu and Professor Jinliang He from Tsinghua. Without their support, I would not be able to join MIT at all.

Finally and most importantly, I would like to thank my parents and family, who always provide me with their strong support from China. I really appreciate your love and effort to bring me up to be a better individual. This thesis is dedicated to you.

Bibliographic Notes

The main content in this thesis has been published in peer-reviewed conferences. The list of papers is as follows:

- **Robust Decision Trees Against Adversarial Examples**, International Conference on Machine Learning (ICML), 2019
- **Robustness Verification of Tree-based Models**, Conference on Neural Information Processing Systems (NeurIPS), 2019
- **The Limitations of Adversarial Training and the Blind-spot Attack**, International Conference on Learning Representations (ICLR), 2019
- **Robust Deep Reinforcement Learning against Adversarial Perturbations on Observations**, Conference on Neural Information Processing Systems (NeurIPS), 2020

In some cases, these reflect joint first-authorship. The collaborations leading to these papers and contributions are gratefully acknowledged. The code for the works presented in this thesis is publicly available at https://github.com/chenhongge.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Adversarial Robustness Problems

In the past ten years, we have seen significant breakthroughs of machine learning in many applications [60, 64, 80, 106]. For most machine learning models, their core structure is a function approximator f that maps data from an input in space $\mathcal{X}$ to an output in space $\mathcal{Y}$. Both spaces can be either continuous or discrete. Machine learning algorithms are proposed to obtain f from a large amount of training data from $\mathcal{X} \times \mathcal{Y}$ in the training phase. For parametric models, such as deep neural networks (DNNs), f can be learned by iteratively changing the value of its parameters until a specific loss function is minimized. For non-parametric models, such as decision trees and k-nearest neighbors, f has structures which can also be optimized given training data. Once f is obtained, the models are evaluated in the testing phase, where they are expected to provide output to test examples that are not present in training data.

The effectiveness of machine learning algorithms relies strongly on an assumption that the training data and test data are sampled from the same distribution, or at least similar distributions. However, this is not true in many cases, which may lead to serious consequences in the deployment of the models. The distributions of training data and test data may be inherently different from each other, for example, when we collect training data from historical stock prices and try to make predictions on future prices. Importantly, both training data and test data can be manipulated by

an adversary; this is a serious concern especially when the models are deployed to security-sensitive tasks such as autonomous driving, face or voice recognition, and anomaly detection. Malicious manipulation of data can create mismatch between training and test data distribution, mislead the models, and significantly damage their performance. We define this kind of malicious manipulation of data as *adversarial attacks*. More specifically, adversarial attacks on training data are called *data poison attacks,* and adversarial attacks on test data are termed *evasion attacks*. In this thesis, we mainly focus on evasion attacks, and unless otherwise indicated, whenever we refer to adversarial attacks we mean evasion attacks. In evasion attacks, the models are assumed to be well-trained and fixed. The adversarial attack is conducted by creating test examples on which a state-of-the-art machine learning model makes incorrect predictions. Moreover, these crafted test examples are required to be indistinguishable from natural examples to humans. The examples satisfying these two requirements are called *adversarial examples* [146]. Usually, adversarial examples are created by perturbing natural examples. This perturbation is not purely random perturbation, but rather perturbation by adding noise that is generated with computation. Hence adversarial noise are usually model-specific and example-specific.

For simplicity, we first consider a multi-class classification model $f : \mathbb{R}^d \rightarrow \{1, \ldots, C\}$ where d is the input dimension and C is the number of classes. For an input test example x_0 with ground truth label y_0, assuming that $y_0 = f(x_0)$, the **minimal distance adversarial example** is defined by

$$x^* = \underset{x}{\operatorname{argmin}}\, d(x, x_0) \quad \text{s.t.} \quad f(x) \neq y_0, \tag{1.1}$$

where $d(\cdot, \cdot)$ is a distance metric. In this thesis, we focus on the ℓ_p norm distance metric, $d(x, x_0) = \|x_0 - x\|_p$, $p \geq 0$, which is widely used in recent studies [95, 162, 18]. In this case, the **minimal distance adversarial distance** r^* has the following definition:

$$r^* = \min_{\delta} \|\delta\|_p \quad \text{s.t.} \quad f(x_0 + \delta) \neq y_0. \tag{1.2}$$

The robustness of a machine learning model can be evaluated by the mean of r^*

among all test examples. However, exactly solving (1.1) or (1.2) is usually intractable. For example, if $f(\cdot)$ is a neural network with ReLU activations, (1.1) is non-convex and [76] showed that solving (1.1) is an NP-complete problem in terms of number of neurons in the network. Therefore, researchers solve upper or lower bounds of r^* in practice.

Upper bound: adversarial attacks – Solving an upper bound of (1.1) can be achieved by conducting adversarial attacks. Basically, adversarial attacks search δ such that

$$f(x_0 + \delta) \neq f(x_0) \text{ and } \|\delta\|_p \leq \epsilon \tag{1.3}$$

for a given a perturbation budget $\epsilon > 0$. The condition $\|\delta\|_p \leq \epsilon$ is designed such that $x_0 + \delta$ is indistinguishable from x_0 by humans. An attack is successful if a δ satisfying (1.3) is found. If the attack is targeted to a specific class t (i.e., the goal is to cause the classification of the perturbed input to be class t), we use $f(x_0 + \delta) = t$ to replace $f(x_0 + \delta) \neq f(x_0)$ as a constraint. Adversarial attacks can be roughly divided into two categories: *white-box attacks* and *black-box attacks*. White-box attacks assume that the model is fully exposed to the attacker, including parameters and structures, while in black-box attacks, the attackers can query the model but have no direct access to any internal information inside the model.

Lower bound: robustness verification – Solving a lower bound is typically achieved by or termed *robustness verification*. A lower bound $\underline{r}$ is found such that the model output is guaranteed to remain the same as long as the perturbation ℓ_p norm is smaller than $\underline{r}$:

$$f(x_0 + \delta) = f(x_0) \text{ for all } \|\delta\|_p \leq \underline{r}\,. \tag{1.4}$$

1.2 Background on Adversarial Attacks and Model Robustness in Deep Neural Networks

The phenomenon of adversarial examples was first discovered in DNN models [146], and so we first introduce background on adversarial attack and model robustness of

DNNs in this section. For other machine learning models including decision trees, we will expand the discussion of adversarial attack and model robustness in Chapters 2 and 3.

1.2.1 Adversarial Attacks

Adversarial attacks are algorithms developed for finding a feasible solution δ of (1.3), where $\|\delta\|_p$ is an *upper bound* of r^* if the attack is successful. Many algorithms have been proposed for attacking DNN models [53, 81, 20, 95, 29, 30, 70, 9, 33, 111, 91, 170]. Most practical attacks cannot guarantee to reach the minimal adversarial perturbation r^* due to the non-convexity of the DNN models, and therefore, attacking algorithms cannot provide formal guarantees on model robustness [2, 155]. Here we first introduce several representative white-box adversarial attacks for neural networks.

- **Fast Gradient Sign Attack (FGSM)** – FGSM [53] is a pioneering attacking algorithm, and is well-known for its efficiency and easy implementation. However, it suffers from low attack success rate. This method only needs to compute the gradient once to generate an adversarial example x from the original example x_0:

$$x \leftarrow \text{clip}[x_0 + \epsilon\ \mathbf{sgn}(\nabla J(f(x_0), y_0))],$$

  where $\mathbf{sgn}(\nabla J(x_0, y_0))$ is the sign of the gradient of the training loss J with respect to x_0, and $\text{clip}(x)$ ensures that x stays within the range of valid values, such as pixel range in image data. It is efficient for generating adversarial examples, as it is just a one-step attack.

- **I-FGSM and PGD Attack** – To improve FGSM's attack success rate, authors in [81] propose an iterative version which greatly improves performance. Iterative FGSM, or I-FGSM, applies FGSM multiple times within a neighborhood of the original example, and is able to fool a DNN model with very high success rate. When one runs I-FGSM for K iterations, one sets the per-iteration perturbation to $\frac{\epsilon}{K}\ \mathbf{sgn}(\nabla J(x_0, t))$. I-FGSM can be viewed as a projected gradient descent (PGD) method inside an ℓ_∞ ball [35, 95].

- **C&W attack** – Carlini and Wagner [20] formulate the problem of generating adversarial examples x as the following optimization problem:

$$\begin{aligned} \min_{x} \quad & \lambda f(x,t) + \|x - x_0\|_2^2 \\ \text{s.t.} \quad & x \text{ is in a valid range,} \end{aligned}$$

  where $f(x,t)$ is a loss function to measure the distance between the prediction of x and the target label t. Usually we choose

$$f(x,t) = \max\{\max_{i \neq t}[(\mathbf{Logit}(x))_i - (\mathbf{Logit}(x))_t], -\kappa\}$$

  as it was shown to be effective by [20]. $\mathbf{Logit}(x)$ denotes the vector representation of x at the logit layer, κ is a confidence level, and a larger κ generally improves transferability of adversarial examples. This attack is so strong that it can achieve almost 100% attack success rate and has bypassed ten different adversary detection methods as reported in [20].

- **EAD-L1 attack** – EAD-L1 attack [29] refers to the **E**lastic-Net **A**ttacks to **D**NNs, which is a more general formulation than C&W attack. It proposes to use elastic-net regularization, a linear combination of ℓ_1 and ℓ_2 norms, to penalize large distortion between the original and adversarial examples. Specifically, it learns the adversarial example x via

$$\begin{aligned} \min_{x} \quad & \lambda f(x,t) + \|x - x_0\|_2^2 + \beta \|x - x_0\|_1 \\ \text{s.t.} \quad & x \text{ is in a valid range,} \end{aligned}$$

  where $f(x,t)$ is the same as used in the C&W attack. Ref. [29] showed that EAD-L1 attack is highly transferable and can bypass many defenses.

According to [19], PGD attack, C&W attack, and EAD-L1 attacks are by far the strongest of state-of-the-art attacks that find adversarial examples with the smallest ℓ_1, ℓ_2, and ℓ_∞ perturbations, respectively. To bypass some defenses with obfuscated

gradients, the Backward Pass Differentiable Approximation (BPDA) attack introduced in [2], is shown to successfully circumvent many defenses.

The aforementioned attacks all rely on explicit gradient information from the model, and thus are all white-box attacks. The white-box setting is often argued as being unrealistic in the literature. In contrast, several recent works have studied ways to fool the model given only model output scores or probabilities. Methods in [30] and [70] are able to craft adversarial examples by making queries to obtain the corresponding probability outputs of the model. A stronger and more general attack has been developed recently by [33], that does not rely on the gradient or the smoothness of model output. This enables attackers to successfully attack models that only output hard labels.

1.2.2 Robustness Verification

On the other hand, **robustness verification algorithms** are designed to find the exact value or a *lower bound* of r^*. An exact verifier needs to solve (1.1) to the global optimum, and so we typically resort to relaxed verifiers that give lower bounds. After a verification algorithm finds a lower bound $\underline{r}$, it guarantees that no adversarial example exists within a radius $\underline{r}$ ball around x. This is important for deploying machine learning algorithms to safety-critical applications such as autonomous vehicles or aircraft control systems [76, 73]. For verification, instead of solving (1.1) we can solve the following **decision problem of robustness verification** for a given budget ϵ:

$$\textit{Does there exist an } x' \in \mathrm{Ball}(x, \epsilon) \quad \textit{such that } f(x') \neq y_0? \tag{1.5}$$

In our setting $\mathrm{Ball}(x_0, \epsilon) := \{x : \|x - x_0\|_p \leq \epsilon\}$. If we can answer this decision ("yes"/"no") problem, a binary search can give us the value of r^*, so the complexity of (1.5) is in the same order of (1.1). Furthermore, solving (1.1) using an approximation algorithm (with answer "unknown" allowed) can lead to a lower bound of r^*, which is useful for verification. The decision version is also widely used in the verification community since "verified accuracy under ϵ perturbation" is an important metric,

which is defined as the portion of test samples that the answers to (1.5) are "no". Verification methods for neural networks have been studied extensively in the past few years [162, 163, 161, 182, 139, 51, 140].

1.2.3 Adversarial Defense

It is difficult to defend against adversarial examples, especially under strong and adaptive attacks. Some early methods, including feature squeezing [173] and defensive distillation [112] have been proven ineffective against stronger attacks like C&W. Many recently proposed defense methods are based on obfuscated gradients [57, 142, 17, 93, 126] and are already overcome by BPDA attack [3].

Adversarial training, first introduced in [82], is effective on DNNs against various attacks. In adversarial training, adversarial examples are generated during the training process and are used as training data to increase model robustness. This technique has been formally posed as a min-max robust optimization problem in [95] and has achieved very good performance under adversarial attacks. Several recent work have tried to improve over the original adversarial training formulation [89, 90, 177]. There are some other methods in the literature seeking to give provable guarantees on the robustness performance, such as distributional robust optimization [141], convex relaxations [162, 163, 158] and semidefinite relaxations [120]. Some of these methods can be deployed in medium-sized networks and achieve satisfactory robustness.

1.3 Thesis Organization

In this thesis, we study the robustness of machine learning models. Studies of this topic are generally from three different perspectives, adversarial attack, robustness verification, and adversarial defense. In this thesis, we focus on robustness verification and adversarial defense. The overall structure of this thesis is shown in Figure 1-1. Specifically, we present some pioneering research on the verification and defense methods for decision tree-based models in Chapters 2 and 3. For DNNs, we demonstrate empirical studies on the robustness generalization problems in Chapter 4. Finally, we

use robust machine learning models as building blocks in reinforcement learning and show how we improve the robustness of reinforcement learning agents in Chapter 5.

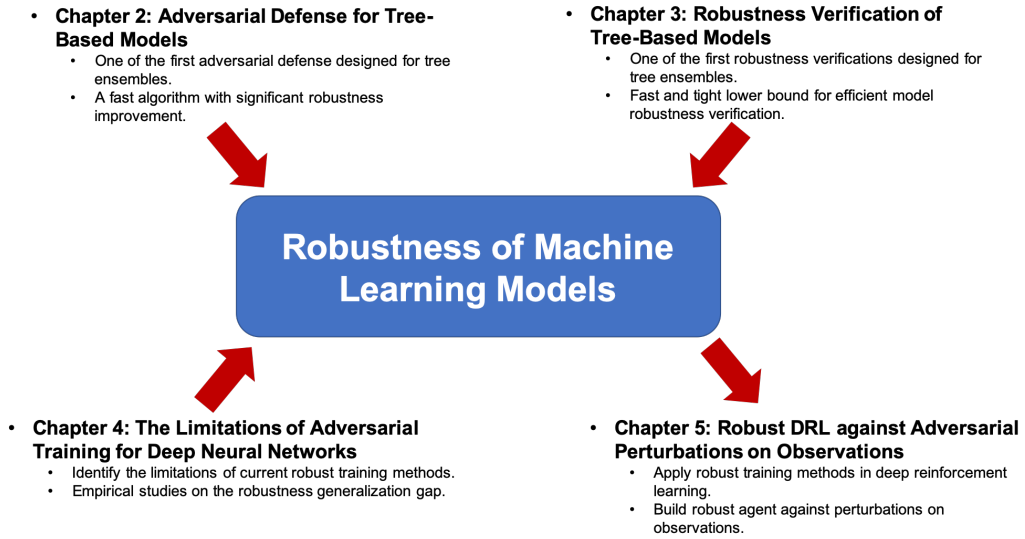


Figure 1-1: An overview of thesis organization.

Though adversarial examples and model robustness have been extensively studied in the context of deep neural networks, research on this issue in tree-based models is still relatively limited. In Chapter 2, we show that tree based models are also vulnerable to adversarial examples and introduce a pioneering method to improve the robustness of tree-based models. Due to the discrete nature of trees, learning robust tree models are fundamentally different than learning robust neural networks. At its core, this method aims to optimize the performance under the worst case perturbation of input data, which leads to a max-min saddle point problem when splitting tree nodes. Incorporating this saddle point objective into the decision tree building procedure is non-trivial: a naive approach to find the best split according to will take exponential time. To make our approach tractable and scalable, we propose efficient tree building algorithms by approximating the inner minimizer in this saddle point problem, and present efficient implementations for classical information gain based trees as well as state-of-the-art large scale tree boosting models such as XGBoost [31]. Experimental results on real world datasets demonstrate that the proposed algorithms can efficiently and substantially improve the robustness of tree-based models against adversarial examples.

Then in Chapter 3 we introduce an efficient robustness verification method for tree-based models, including individual decision trees and tree ensembles, such as random forests (RFs) and gradient boosted decision trees (GBDTs). Exactly verifying robustness of tree-based models involves finding the exact minimal adversarial perturbation or a guaranteed lower bound of it. We show that verifying the robustness of a single decision has complexity linear to the number of leaves in the model. For tree-ensembles, however, existing approaches find the minimal adversarial perturbation by a mixed integer linear programming (MILP) problem [75], which takes exponential time in theory and is computationally expensive in practice when models are large. We cast the tree ensembles verification problem as a max-clique problem on a multi-partite graph. By exploiting the boxicity of the graph, we develop an efficient multi-level verification algorithm that can give tight lower bounds on robustness of decision tree ensembles, while allowing iterative improvement and termination at any-time. On random forest or gradient boosted decision trees models trained on various datasets, our algorithm is up to hundreds of times faster than a the previous approach that requires solving MILPs, and is able to give tight robustness verification bounds on large GBDTs with hundreds of deep trees.

In Chapter 4, we continue on learning robust models but shift the focus and study the adversarial defense problem for neural networks. Recently, there are many works on improving robustness of deep neural networks. In this chapter, we demonstrate some empirical studies on the practicality and the hardness of adversarial training. We show that the robustness of a model with adversarial defense on a test example has a strong correlation with the distance between that example and the manifold of training data embedded by the network. Specifically, test examples that are relatively far away from this manifold are more likely to be vulnerable to adversarial attacks. Consequentially, we demonstrate that an adversarial training based defense is vulnerable to a new class of attacks, the "blind-spot attack", where the input images reside in low density regions ("blind-spots") of the empirical distribution of training data but is still on the valid ground-truth data manifold. For image datasets such as MNIST and Fashion-MNIST, we found that these blind-spots can be easily

found by simply scaling and shifting image pixel values. We conjecture that for large image datasets with high dimensional and complex data manifolds such as CIFAR-10 and ImageNet, due to the curse of dimensionality and the scarcity of training data, the existence of blind-spots in adversarial training makes defending on any valid test examples difficult. We demonstrate that blind-spots exist in both certified and uncertified adversarial defenses and models' robustness has a robustness generalization gap between the training set and the test set.

In Chapter 5 we apply neural network adversarial defense methods to reinforcement learning to train agents that are robust against perturbations on state observations. A deep reinforcement learning (DRL) agent takes action based on observation of its current state, which may contain natural measurement noises or adversarial perturbations. As shown in prior works, the inconsistency between observations and true states can mislead the agent into making sub-optimal actions. Existing works on improving the robustness of deep reinforcement learning with perturbation on observation have limited success and do not have theoretical foundations. In this chapter, we empirically show that naively applying existing method on improving robustness for supervised classification tasks, such as adversarial training, is ineffective for many reinforcement learning problems. Thus, we propose the state-adversarial Markov decision process (SA-MDP) to study the fundamental properties of this problem, and develop a theoretically principled regularization which can be applied to different deep reinforcement learning algorithms, including deep Q networks (DQN) and proximal policy optimization (PPO) for both discrete and continuous action control problems. We significantly improve the robustness of agents under strong white box adversarial attacks, including new attacks of our own.

Finally, Chapter 6 concludes this thesis and suggests future directions. We summarize the key contributions of the thesis stemming from adversarial verification and defense for tree-based machine learning models, studies on adversarial defense for deep neural networks, to applications of robust models to deep reinforcement learning. This section also includes unsolved challenges in adversarial robustness and points out future directions based on the insights from adversarial robustness.

Chapter 2

Adversarial Defense for Tree-Based Models

2.1 Introduction

The discovery of adversarial examples in various deep learning models [146, 78, 34, 27, 21, 68] has led to extensive studies of deep neural network (DNN) robustness under such maliciously crafted subtle perturbations. Although deep learning-based model robustness has been well-studied in the recent literature from both attack and defense perspectives, studies on the robustness of tree-based models are quite limited [110].

In this chapter, we shed light on the adversarial robustness of an important class of machine learning models — decision trees. Among machine learning models used in practice, tree-based methods stand out in many applications, with state-of-the-art performance. Tree-based methods have achieved widespread success due to their simplicity, efficiency, interpretability, and scalability on large datasets. They have been suggested as an advantageous alternative to deep learning in some cases [185]. In this chapter, we study the robustness of tree-based models under adversarial attacks, and more importantly, we propose a novel robust training framework for tree-based models. The materials presented in this chapter are based on [26]. Below we highlight the major contributions of this chapter:

- We study the robustness of decision tree-based machine learning algorithms through

the lens of adversarial examples. We study both classical decision trees and state-of-the-art ensemble boosting methods such as XGBoost. We show that, similar to neural networks, tree-based models are also vulnerable to adversarial examples.

- We propose a novel robust decision tree training framework to improve robustness against adversarial examples. This method seeks to optimize the worst case condition by solving a max-min problem. This framework is quite general and can be applied to tree-based models with any score function used to choose splitting thresholds. To the best of our knowledge, this is the first work contributing a general robust decision tree training framework against adversarial examples.
- We implement our framework in both classical information gain based classification trees and state-of-the-art large-scale tree boosting systems. To scale up our framework, we make necessary and efficient approximations to handle complex models and real world data sets. Our experimental results show consistent and substantial improvements on adversarial robustness.

2.2 Decision Tree and Gradient Boosted Decision Tree

Decision tree learning methods are widely used in machine learning and data mining. As considered here, the goal is to create a tree structure with each interior node corresponding to one of the input features. Each interior node has two children, and edges to child nodes represent the split condition for that feature. Each leaf provides a prediction value of the model, given that the input features satisfy the conditions represented by the path from the root to that leaf. In practice, decision tree learning algorithms are based on greedy search, which builds a tree starting from its root by making locally optimal decisions at each node. Classical decision tree training recursively chooses features, sets thresholds and splits the examples on a node by maximizing a pre-defined score, such as information gain or Gini impurity.

Decision trees are often used within ensemble methods. A well-known gradient tree

boosting method has been developed by [47, 48] and [49] to allow optimization of an arbitrary differentiable loss function. Later scalable tree boosting systems have been built to handle large datasets. For example, pGBRT [152] parallelizes the training procedure by data partitioning for faster and distributed training. XGBoost [31] is a prominent tree boosting software framework; in data mining contests, 17 out of 29 published winning solutions at Kaggle's blog in 2015 used XGBoost in their models. LightGBM [77, 181] is another highly efficient boosting framework that utilizes histograms on data features to significantly speed up training. mGBDT [45] learns hierarchical representations by stacking multiple layers of gradient boosted decision trees (GBDTs). Other variants such as extreme multi-label GBDT [137] and cost efficient tree boosting approaches [114, 174] have also been proposed recently.

2.3 Adversarial Examples of Decision Tree Based Models

Recent developments in machine learning have resulted in the deployment of large-scale tree boosting systems in critical applications such as fraud and malware detection. Unlike deep neural networks (DNNs), tree based models are non-smooth, non-differentiable and sometimes interpretable, which might lead to the belief that they are more robust than DNNs. However, the experiments in this chapter show that similar to DNNs, tree-based models can also be easily compromised by adversarial examples. In this chapter, we focus on untargeted attacks, which are considered to be successful as long as the model misclassifies the adversarial examples.

Unlike DNNs, algorithms for crafting adversarial examples for tree-based models are poorly studied. The main reason is that tree-based models are discrete and non-differentiable, thus we cannot use common gradient descent based methods for white-box attack. An early attack algorithm designed for single decision trees has been proposed by [110], based on greedy search. To find an adversarial example, this method searches the neighborhood of the leaf which produces the original prediction,

and finds another leaf labeled as a different class by considering the path from the original leaf to the target leaf, and changing the feature values accordingly to result in misclassification.

A white-box attack against binary classification tree ensembles has been proposed by [75]. This method finds the exact smallest distortion (measured by some ℓ_p norm) necessary to mislead the model. However, the algorithm relies on Mixed Integer Linear Programming (MILP) and thus can be very time-consuming when attacking large scale tree models as arise in XGBoost. In this chapter, we use the ℓ_∞ version of Kantchelian's attack as one of our methods to evaluate small and mid-size binary classification model robustness. Ref. [75] also introduce a faster approximation to generate adversarial examples using symbolic prediction with ℓ_0 norm minimization and combine this method into an adversarial training approach. Unfortunately, the demonstrated adversarial training is not very effective; despite increasing model robustness for ℓ_0 norm perturbations, robustness for ℓ_1, ℓ_2 and ℓ_∞ norm perturbations are noticeably reduced compared to the naturally (non-robustly) trained model.

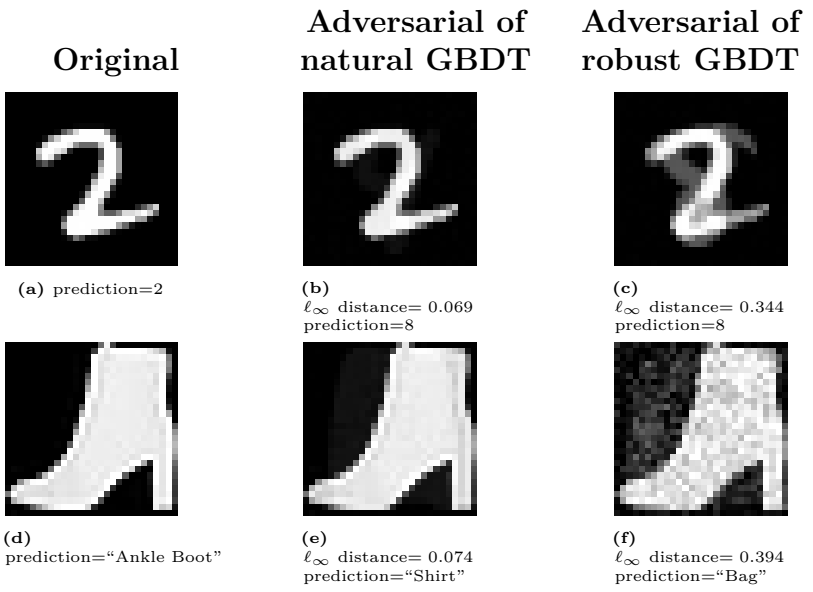


Figure 2-1: MNIST and Fashion-MNIST examples and their adversarial examples found using the untargeted attack proposed by [33] on 200-tree gradient boosted decision tree (GBDT) models trained using XGBoost with depth=8. Natural GBDT models (nat.) are fooled by small ℓ_∞ perturbations (b, e), while our robust (rob.) GBDT models require much larger perturbations (c, f) for successful attacks. For both MNIST and Fashion-MNIST robust models, we use $\epsilon = 0.3$ (a robust training hyper-parameter which will be introduced in Section 2.4). More examples are shown in the appendix.

In this chapter, in addition to Kantchelian attacks we also use a general attack method proposed in [33] which does not rely on the gradient nor the smoothness of output of a machine learning model. Cheng's attack method has been used to efficiently evaluate the robustness of complex models on large datasets, even under black-box settings. To deal with non-smoothness of model output, this method focuses on the distance between the benign example and the decision boundary, and reformulates the adversarial attack as a minimization problem of this distance. Despite the non-smoothness of model prediction, the distance to decision boundary is usually smooth within a local region, and can be found by binary search on vector length given a direction vector. To minimize this distance without gradient, [33] used a zeroth order optimization algorithm with a randomized gradient-free method. In this chapter, we use the ℓ_∞ version of Cheng's attack. Some adversarial examples obtained by this method are shown in Figure 2-1, where we display results on both MNIST and Fashion-MNIST datasets. The models we test are natural GBDT models trained using XGBoost and our robust GBDT models, each with 200 trees and a tree depth of 8. Cheng's attack is able to craft adversarial examples with very small distortions on natural models; for human eyes, the adversarial distortion added to the natural model's adversarial examples appear as imperceptible noise. We also conduct white-box attacks using the MILP formulation [75], which takes much longer time to solve but the ℓ_∞ distortion found by MILP is comparable to Cheng's method; see Section 5.4 for more details. In contrast, for our robust GBDT model, the required adversarial example distortions are so large that we can even vaguely see a number 8 in subfigure (c). The substantial increase in the ℓ_∞ distortion required to misclassify as well as the increased visual impact of such distortions shows the effectiveness of our robust decision tree training, which we will introduce in detail next. In the main text, we use the ℓ_∞ version of Kantchelian's attack; we present results of ℓ_1 and ℓ_2 Kantchelian attacks in the appendix.

2.4 Robust Decision Trees

2.4.1 Intuition

As shown in Section 2.3, tree-based models are vulnerable to adversarial examples. Thus it is necessary to augment the classical natural tree training procedure in order to obtain reliable models robust against adversarial attacks. Our method formulates the process of optimally finding best split threshold in decision tree training as a robust optimization problem. As a conceptual illustration, Figure 2-2 presents a special case where the traditional greedy optimal splitting may yield non-robust models. A horizontal split achieving high accuracy or score on original points may be easily compromised by adversarial perturbations. On the other hand, we are able to select a better vertical split considering possible perturbations in ℓ_∞ balls. At a high level, the robust splitting feature and threshold take the distances between data points into account (which is often ignored in most decision tree learning algorithms) and tries to optimize the *worst case* performance under adversarial perturbations. Some recent works in DNNs [71, 151] divided features into two categories, robust features and non-robust features. In tree-based models, the effect of this dichotomy on the robustness is straight forward, as seen in the two different splits in Figure 2-2 using $x^{(1)}$ (a robust feature) and $x^{(2)}$ (a non-robust feature).

2.4.2 General Robust Decision Tree Framework

In this section we formally introduce our robust decision tree training framework. For a training set with N examples and d real valued features $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$ ($1 \leq i \leq N$, $y_i \in \mathbb{R}$, $\mathbf{x}_i = [x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(j)}, \ldots, x_i^{(d)}] \in \mathbb{R}^d$), we first normalize the feature values to $[0,\ 1]$ such that $\mathbf{x}_i \in [0, 1]^d$ (the best feature value for split will also be scaled accordingly, but it is irrelevant to model performance). For a general decision tree based learning model, at a given node, we denote $\mathcal{I} \subseteq \mathcal{D}$ as the set of points at that node. For a split on the j-th feature with a threshold η, the sets that will be mentioned in Sections 2.4.2, 2.4.3 and 2.4.4 are summarized in Table 2.1.

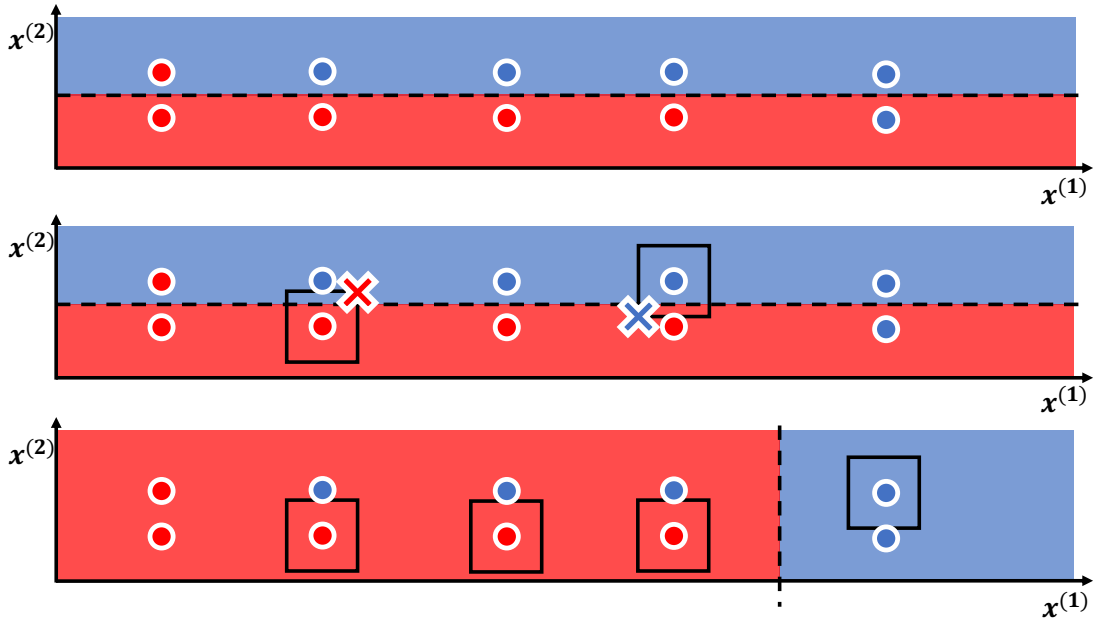


Figure 2-2: (Best viewed in color) A simple example illustrating how robust splitting works. Upper: A set of 10 points that can be easily separated with a horizontal split on feature $x^{(2)}$. The accuracy of this split is 0.8. Middle: The high accuracy horizontal split cannot separate the ℓ_∞ balls around the data points and thus an adversary can perturb any example $\mathbf{x}_i$ within the indicated ℓ_∞ ball to mislead the model. The worst case accuracy under adversarial perturbations is 0 if all points are perturbed within the square boxes (ℓ_∞ norm bounded noise). Lower: a more robust split would be a split on feature $x^{(1)}$. The accuracy of this split is 0.7 under all possible perturbations within the same size ℓ_∞ norm bounded noise (square boxes).

In classical tree based learning algorithms (which we refer to as "natural" trees in this chapter), the quality of a split on a node can be gauged by a *score function* $S(\cdot)$: a function of the splits on left and right child nodes ($\mathcal{I}_L$ and $\mathcal{I}_R$), or equivalently on the chosen feature j to split and a corresponding threshold value η. Since $\mathcal{I}_L$ and $\mathcal{I}_R$ are determined by j, η and $\mathcal{I}$, we abuse the notation and define $S(j,\ \eta,\ \mathcal{I}) := S(\mathcal{I}_L,\ \mathcal{I}_R)$.

Traditionally, people consider different scores for choosing the "best" split, such as information gain used by ID3 [119] and C4.5 [119], or Gini impurity in CART [8]. Modern software packages [31, 77, 39] typically find the best split that minimize a loss function directly, allowing decision trees to be used in a large class of problems (i.e., mean square error loss for regression, logistic loss for classification, and ranking loss for ranking problems). A regular ("natural") decision tree training process will either exactly or approximately evaluate the score function, for all possible features and split thresholds on the leaf to be split, and select the best $j,\ \eta$ pair:

$$j^*, \eta^* = \underset{j,\ \eta}{\operatorname{argmax}}\ S(\mathcal{I}_L,\ \mathcal{I}_R) = \underset{j,\ \eta}{\operatorname{argmax}}\ S(j,\ \eta,\ \mathcal{I}). \tag{2.1}$$

Notation	Definition
$\mathcal{I}$	set of examples on the current node
$\mathcal{I}_0$	$\mathcal{I} \cap \{(\mathbf{x}_i, y_i) \vert y_i = 0\}$ (for classification)
$\mathcal{I}_1$	$\mathcal{I} \cap \{(\mathbf{x}_i, y_i) \vert y_i = 1\}$ (for classification)
$\mathcal{I}_L$	$\mathcal{I} \cap \{(\mathbf{x}_i, y_i) \vert x^{(j)} < \eta\}$
$\mathcal{I}_R$	$\mathcal{I} \cap \{(\mathbf{x}_i, y_i) \vert x^{(j)} \geq \eta\}$
$\Delta\mathcal{I}$	$\mathcal{I} \cap \{(\mathbf{x}_i, y_i) \vert \eta - \epsilon \leq x^{(j)} \leq \eta + \epsilon\}$
$\Delta\mathcal{I}_L$	$\Delta\mathcal{I} \cap \mathcal{I}_L$
$\Delta\mathcal{I}_R$	$\Delta\mathcal{I} \cap \mathcal{I}_R$
$\mathcal{I}_L^o$	$\mathcal{I}_L \setminus \Delta\mathcal{I}$
$\mathcal{I}_R^o$	$\mathcal{I}_R \setminus \Delta\mathcal{I}$

Table 2.1: Notations of different sets in Section 2.4. We assume a split is made on the j-th feature with a threshold η, and this feature can be perturbed by $\pm\epsilon$.

In our setting, we consider the case where features of examples in $\mathcal{I}_L$ and $\mathcal{I}_R$ can be perturbed by an adversary. Since a typical decision tree can only split on a single feature at one time, it is natural to consider adversarial perturbations within an ℓ_∞ ball of radius ϵ around each example $\mathbf{x}_i$:

$$B_\epsilon^\infty(\mathbf{x}_i) := [x_i^{(1)} - \epsilon,\ x_i^{(1)} + \epsilon] \times \cdots \times [x_i^{(d)} - \epsilon,\ x_i^{(d)} + \epsilon].$$

Such perturbations enable the adversary to minimize the score obtained by our split. So instead of finding a split with highest score, an intuitive approach for robust training is to maximize the minimum score value obtained by all possible perturbations in an ℓ_∞ ball with radius ϵ,

$$j^*,\ \eta^* = \underset{j,\ \eta}{\operatorname{argmax}}\ RS(j,\ \eta,\ \mathcal{I}), \tag{2.2}$$

where $RS(\cdot)$ is a *robust score function* defined as

$$\begin{aligned} &RS(j,\ \eta,\ \mathcal{I}) := \min_{\mathcal{I}' = \{(\mathbf{x}_i',\ y_i)\}} S(j,\ \eta,\ \mathcal{I}') \\ &\text{s.t. } \mathbf{x}_i' \in B_\epsilon^\infty(\mathbf{x}_i), \text{ for all } \mathbf{x}_i' \in \mathcal{I}'. \end{aligned} \tag{2.3}$$

In other words, each $\mathbf{x}_i \in \mathcal{I}$ can be perturbed individually under an ℓ_∞ norm bounded perturbation to form a new set of training examples $\mathcal{I}'$. We consider the worst case perturbation, such that the set $\mathcal{I}'$ triggers the worst case score after split with feature j and threshold η. The training objective (2.2) becomes a max-min optimization

problem. Note that there is an intrinsic consistency between boundaries of the ℓ_∞ balls and the decision boundary of a decision tree. For the split on the j-th feature, perturbations along features other than j do not affect the split. So we only need to consider perturbations within $\pm\epsilon$ along the j-th feature. We define $\Delta\mathcal{I}$ as the *ambiguity set*, containing examples with feature j inside the $[\eta-\epsilon, \eta+\epsilon]$ region (see Table 2.1). Only examples in $\Delta\mathcal{I}$ may be perturbed from $\mathcal{I}_L$ to $\mathcal{I}_R$ or from $\mathcal{I}_R$ to $\mathcal{I}_L$ to reduce the score. Perturbing points in $\mathcal{I} \setminus \Delta\mathcal{I}$ will not change the score or the leaves they are assigned to. We denote $\mathcal{I}_L^o$ and $\mathcal{I}_R^o$ as the set of examples that are certainly on the left and right child leaves under perturbations (see Table 2.1 for definitions). Then we introduce 0-1 variables $s_i = \{0, 1\}$ denoting an example in the ambiguity set $\Delta\mathcal{I}$ to be assigned to $\mathcal{I}_L$ and $\mathcal{I}_R$, respectively. Then the RS can be formulated as a 0-1 integer optimization problem with $|\Delta\mathcal{I}|$ variables, which is NP-hard in general. Additionally, we need to scan through all d features of all examples and solve $O(|\mathcal{I}|d)$ minimization problems for a single split at a single node. This large number of problems to solve makes this computation intractable. Therefore, we need to find an approximation for the $RS(j,\ \eta,\ \mathcal{I})$. In Sections 2.4.3 and 2.4.4, we present two different approximations and corresponding implementations of our robust decision tree framework, first for classical decision trees with information gain score, and then for modern tree boosting systems which can minimize any loss function. It is worth mentioning that we normalize features to $[0,\ 1]^d$ for the sake of simplicity in this chapter. One can also define $\epsilon_1,\ \epsilon_2,\ \ldots,\ \epsilon_d$ for each feature and then the adversary is allowed to perturb $\mathbf{x}_i$ within $[x_i^{(1)} - \epsilon_1,\ x_i^{(1)} + \epsilon_1] \times \cdots \times [x_i^{(d)} - \epsilon_d,\ x_i^{(d)} + \epsilon_d]$. In this case, we would not need to normalize the features. Also, ϵ is a hyper-parameter in our robust model. Models trained with larger ϵ are expected to be more robust and when $\epsilon = 0$, the robust model is the same as a natural model.

2.4.3 Robust Splitting for Decision Trees with Information Gain Score

Here we consider a decision tree for binary classification, $y_i \in \{0, 1\}$, with information gain as the metric for node splitting. The information gain score is

$$S(j,\ \eta,\ \mathcal{I}) := IG(j,\ \eta) = H(y) - H(y|x^{(j)} < \eta),$$

where $H(\cdot)$ and $H(\cdot|\cdot)$ are entropy and conditional entropy on the empirical distribution. For simplicity, we denote $N_0 := |\mathcal{I}_0|$, $N_1 := |\mathcal{I}_1|$, $n_0 := |\mathcal{I}_L \cap \mathcal{I}_0|$ and $n_1 := |\mathcal{I}_L \cap \mathcal{I}_1|$. The following theorem shows adversary's perturbation direction to minimize the information gain.

Theorem 1. *If $\frac{n_0}{N_0} < \frac{n_1}{N_1}$ and $\frac{n_0+1}{N_0} \leq \frac{n_1}{N_1}$, perturbing one example in $\Delta\mathcal{I}_R$ with label 0 to $\mathcal{I}_L$ will decrease the information gain.*

Similarly, if $\frac{n_1}{N_1} < \frac{n_0}{N_0}$ and $\frac{n_1+1}{N_1} \leq \frac{n_0}{N_0}$, perturbing one example in $\Delta\mathcal{I}_R$ with label 1 to $\mathcal{I}_L$ will decrease the information gain. The proof of this theorem will be presented in Section A.1 in the appendix. Note that we also have a similar conclusion for Gini impurity score, which will be shown in Section A.2 in the appendix. Therefore, to decrease the information gain score, the adversary needs to perturb examples in $\Delta\mathcal{I}$ such that $\frac{n_0}{N_0}$ and $\frac{n_1}{N_1}$ are close to each other (the ideal case $\frac{n_0}{N_0} = \frac{n_1}{N_1}$ may not be achieved because n_0, n_1, N_0 and N_1 are integers). The robust split finding algorithm is shown in Algorithm 1. In this algorithm we find a perturbation that minimizes $\left|\frac{n_0}{N_0} - \frac{n_1}{N_1}\right|$ as an approximation and upper bound to the optimal solution. Algorithm 5 in Section A.1 in the appendix shows an $O(|\mathcal{I}|)$ procedure to find such perturbation to approximately minimize the information gain. Since the algorithm scans through $\{x_1^{(j)}, \ldots, x_d^{(j)}\}$ in the sorted order, the sets $\Delta\mathcal{I}, \mathcal{I}_L^o, \mathcal{I}_R^o$ can be maintained in amortized $O(1)$ time in the inner loop. Therefore, the computational complexity of the robust training algorithm is $O(d|\mathcal{I}|^2)$ per split.

Although it is possible to extend our conclusion to other traditional scores of classification trees, we will focus on the modern scenario where we use a regression

Algorithm 1 Robust Split with Information Gain

Input: Training set $\{(x_i,\ y_i)\}|_{i=1}^N$, $x_i \in [0,1]^d$, $y_i \in \{0,1\}$.
Input: The instance set of the current node I.
Input: ϵ, the radius of the ℓ_∞ ball.
Output: Optimal split of the current node.
$\mathcal{I}_0 \leftarrow \{(x_i,\ y_i)|y_i = 0\}, \mathcal{I}_1 \leftarrow \{(x_i,\ y_i)|y_i = 1\}$;
$N_0 \leftarrow |\mathcal{I} \cap \mathcal{I}_0|,\ N_1 \leftarrow |\mathcal{I} \cap \mathcal{I}_1|$;
for $j \leftarrow 1$ **to** d **do**
  for m in sorted($\mathcal{I}$, ascending order by x_m^j) **do**
    $\eta \leftarrow \frac{1}{2}(x_m^j + x_{m+1}^j)$, $\Delta\mathcal{I} \leftarrow \mathcal{I} \cap \{(x_i,\ y_i)|\eta - \epsilon \le x^{(j)} \le \eta + \epsilon\}$;
    $\mathcal{I}_L^o \leftarrow \{(x_i,\ y_i)|x^{(j)} < \eta - \epsilon\}, \mathcal{I}_R^o \leftarrow \{(x_i,\ y_i)|x^{(j)} > \eta + \epsilon\}$;
    $n_0^o \leftarrow |\mathcal{I}_L^o \cap \mathcal{I}_0|$, $n_1^o \leftarrow |\mathcal{I}_L^o \cap \mathcal{I}_1|$;
    Find Δn_0^*, Δn_1^* to minimize $|\frac{\Delta n_0^* + n_0^o}{N_0} - \frac{\Delta n_1^* + n_1^o}{N_1}|$ using Algorithm 5 in Section A.1 in the appendix;
    From $\Delta\mathcal{I}$, add Δn_0^* points with $y = 0$ and Δn_1^* points with $y = 1$ to $\mathcal{I}_L^o$ and obtain $\mathcal{I}_L$;
    Add remaining points in $\Delta\mathcal{I}$ to $\mathcal{I}_R^o$ and obtain $\mathcal{I}_R$;
    $RS(j,\ \eta) \leftarrow IG(\mathcal{I}_L, \mathcal{I}_R)$;
  end for
end for
$j^*, \eta^* \leftarrow \text{argmax}_{j,\ \eta}\, RS(j,\ \eta)$;
Split on feature j^* with a threshold η^*;

tree to fit any loss function in Section 2.4.4.

2.4.4 Robust Splitting for GBDT models

We now introduce the regression tree training process used in many modern tree boosting packages including XGBoost [31], LightGBM [77] and CatBoost [39]. Specifically, we focus on the formulation of gradient boosted decision tree (GBDT), which is one of the most successful ensemble models and has been widely used in industry. GBDT is an additive tree ensemble model $\phi(\cdot)$ combining outputs of K trees

$$\hat{y}_i = \phi_K(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$$

where each f_k is a decision tree and $\hat{y}_i$ is the final output for $\mathbf{x}_i$. Here we only focus on regression trees where $\hat{y}_i \in \mathbb{R}$. Note that even for a classification problem, the modern treatment in GBDT is to consider the data with logistic loss, and use a regression tree

Algorithm 2 Robust Split for Boosted Tree

Input: training set $\{(x_i,\ y_i)\}|_{i=1}^N$, $x_i \in [0,1]^d$, $y_i \in \mathbb{R}$.
Input: The instance set of the current node I.
Input: ϵ, the radius of the ℓ_∞ ball.
Output: Optimal split of the current node.
for $j \leftarrow 1$ **to** d **do**
  for m in sorted($\mathcal{I}$, ascending order by x_m^j) **do**
    $\eta \leftarrow \frac{1}{2}(x_m^j + x_{m+1}^j)$;
    $\mathcal{I}_L^o \leftarrow \{(x_i,\ y_i)|x^{(j)} < \eta - \epsilon\}, \Delta\mathcal{I}_L \leftarrow \mathcal{I} \cap \{(x_i,\ y_i)|\eta - \epsilon \le x^{(j)} < \eta\}$;
    $\mathcal{I}_R^o \leftarrow \{(x_i,\ y_i)|x^{(j)} > \eta + \epsilon\}$, $\Delta\mathcal{I}_R \leftarrow \mathcal{I} \cap \{(x_i,\ y_i)|\eta \le x^{(j)} \le \eta + \epsilon\}$;
    $S_1 = S(\mathcal{I}_L, \mathcal{I}_R)$, $S_2 = S(\mathcal{I}_L^o,\ \mathcal{I}_R^o \cup \Delta\mathcal{I})$, $S_3 = S(\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o)$, $S_4 = S(\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L)$;
    $RS(j,\ \eta) \leftarrow \min\{S_1, S_2, S_3, S_4\}$;
  end for
end for
$j^*, \eta^* \leftarrow \text{argmax}_{j,\ \eta}\, RS(j,\ \eta)$;
Split on feature j^* with a threshold η^*;

to minimize this loss. During GBDT training, the trees f_k are generated in an additive manner: when we consider the tree f_K, all previous trees $f_k, k \in \{1, \cdots, K-1\}$ are kept unchanged. For a general convex loss function l (such as MSE or logistic loss), we desire to minimize the following objective

$$\begin{aligned}\mathcal{L}(\phi, \mathcal{D}) &= \sum_{i=1}^N l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k) \\ &= \sum_{i=1}^N l\left(y_i, \phi_{K-1}(\mathbf{x}_i) + f_K(\mathbf{x}_i)\right) + \sum_{k=1}^{K-1} \Omega(f_k) + \Omega(f_K)\end{aligned}$$

where $\Omega(f)$ is a regularization term to penalize complex trees; for example, in XGBoost, $\Omega(f) = \gamma T + \frac{1}{2}\lambda\|\omega\|^2$, where T is the number of leaves, ω is a vector of all leaf predictions and $\lambda,\ \gamma \ge 0$ are regularization constants. Importantly, when we consider f_K, ϕ_{K-1} is a constant. The impact of $f_K(\mathbf{x}_i)$ on $l(y_i, \hat{y}_i)$ can be approximated using a second order Taylor expansion:

$$\begin{aligned}&l(y_i, \phi_K(\mathbf{x}_i)) \approx \hat{l}(y_i, \phi_K(\mathbf{x}_i)) \\ &:= l(y_i, \phi_{K-1}(\mathbf{x}_i)) + g_i f_K(\mathbf{x}_i) + \frac{1}{2} h_i (f_K(\mathbf{x}_i))^2\end{aligned}$$

where $g_i = \frac{\partial l(y_i, \phi_K(\mathbf{x}_i))}{\partial f_K(\mathbf{x}_i)}$ and $h_i = \frac{\partial^2 l(y_i, \phi_K(\mathbf{x}_i))}{\partial f_K^2(\mathbf{x}_i)}$ are the first and second order derivatives

on the loss function with respect to the prediction of decision tree f_K on point $\mathbf{x}_i$. Conceptually, ignoring the regularization terms, the score function can be given as:

$$S(\mathcal{I}_L, \mathcal{I}_R) = \sum_{i \in \mathcal{I}_L} \hat{l}(y_i, \phi_K(\mathbf{x}_i))|_{\phi_K(\mathbf{x}_i)=\omega_L} + \sum_{i \in \mathcal{I}_R} \hat{l}(y_i, \phi_K(\mathbf{x}_i))|_{\phi_K(\mathbf{x}_i)=\omega_R} - \sum_{i \in \mathcal{I}} \hat{l}(y_i, \phi_K(\mathbf{x}_i))|_{\phi_K(\mathbf{x}_i)=\omega_P}$$

where ω_L, ω_R and ω_P are the prediction values of the left, right and parent nodes. The score represents the improvements on reducing the loss function $\mathcal{L}$ for all data examples in $\mathcal{I}$. The exact form of score used in XGBoost with regularization terms is given in [31]:

$$S(j,\ \eta,\ \mathcal{I}) = S(\mathcal{I}_L,\ \mathcal{I}_R) := \frac{1}{2}\left[\frac{(\sum_{i \in \mathcal{I}_L} g_i)^2}{\sum_{i \in \mathcal{I}_L} h_i + \lambda} + \frac{(\sum_{i \in \mathcal{I}_R} g_i)^2}{\sum_{i \in \mathcal{I}_R} h_i + \lambda} - \frac{(\sum_{i \in \mathcal{I}} g_i)^2}{\sum_{i \in \mathcal{I}} h_i + \lambda}\right] - \gamma, \quad (2.4)$$

where γ is a regularization constant. Again, to minimize the score by perturbing points in $\Delta\mathcal{I}$, the adversary needs to solve an intractable 0-1 integer optimization at each possible splitting position. Since GBDT is often deployed in large scale data mining tasks with a large amount of training data to scan through at each node, and we need to solve RS $O(|\mathcal{I}|d)$ times, we cannot afford any expensive computation. For efficiency, our robust splitting procedure for boosted decision trees, as detailed in Algorithm 2, approximates the minimization by considering only four representative cases: (1) no perturbations: $S_1 = S(\mathcal{I}_L,\ \mathcal{I}_R)$; (2) perturb all points in $\Delta\mathcal{I}$ to the right: $S_2 = S(\mathcal{I}_L^o, \mathcal{I}_R^o \cup \Delta\mathcal{I})$; (3) perturb all points in $\Delta\mathcal{I}$ to the left: $S_3 = S(\mathcal{I}_L^o \cup \Delta\mathcal{I}, \mathcal{I}_R^o)$; (4) swap the points in $\Delta\mathcal{I}$: $S_4 = S(\mathcal{I}_L^o \cup \Delta\mathcal{I}_R, \mathcal{I}_R^o \cup \Delta\mathcal{I}_L)$. We take the minimum among the four representative cases as an approximation of the RS:

$$RS(j,\ \eta,\ \mathcal{I}) \approx \min\{S_1,\ S_2,\ S_3,\ S_4\}. \quad (2.5)$$

Though this method only takes $O(1)$ time to give a rough approximation of the RS at each possible split position, it is effective empirically as demonstrated next in Section 2.5.

2.5 Experiments

2.5.1 Robust Information Gain Decision Trees

We present results on three small datasets with robust information gain based decision trees using Algorithm 1.[1] We focus on untargeted adversarial attacks. For each dataset we test on 100 examples (or the whole test set), and we only attack correctly classified images. Attacks proceed until the attack success rate is 100%; the differences in robustness are reflected in the distortion of the adversarial examples required to achieve a successful attack. In Table 2.2, we present the average ℓ_∞ distortion of the adversarial examples of both classical natural decision trees and our robust decision trees trained on different datasets. We use Papernot's attack as well as ℓ_∞ versions of Cheng's and Kantchelian's attacks. The ℓ_1 and ℓ_2 distortion found by Kantchelian's ℓ_1 and ℓ_2 attacks are presented in Table A.1 in Appendix A. The adversarial examples found by Cheng's, Papernot's and Kantchelian's attacks have much larger ℓ_∞ norm for our robust trees compared to those for the natural trees, demonstrating that our robust training algorithm improves the decision tree robustness substantially. In some cases our robust decision trees also have higher test accuracy than the natural trees. This may be due to the fact that the robust score tends to encourage the tree to split at thresholds where fewer examples are in the ambiguity set, and thus the split is also robust against random noise in the training set. Another possible reason is the implicit regularization in the robust splitting. The robust score is always lower than the regular score and thus our splitting is more conservative. Also, from results in Table 2.2 we see that most of the adversarial examples found by Papernot's attack have larger ℓ_∞ norm than those found by Cheng's ℓ_∞ attack. This suggests that the straight-forward greedy search attack is not as good as a sophisticated general attack for attacking decision trees. Cheng's attack is able to achieve similar ℓ_∞ distortion as Kantchelian's attack, without solving expensive MILPs. While not scalable to large datasets, Kantchelian's attack can find the *minimum* adversarial examples, reflecting the true robustness of a tree-based model.

[1] Our code is at https://github.com/chenhongge/RobustTrees.

Dataset	training set size	test set size	# of features	# of classes	robust ϵ	depth		test acc.		avg. ℓ_∞ dist. by Cheng's ℓ_∞ attack		avg. ℓ_∞ dist. by Papernot's attack		avg. ℓ_∞ dist. by Kantchelian's ℓ_∞ attack	
						robust	natural	robust	natural	robust	natural	robust	natural	robust	natural
breast-cancer	546	137	10	2	0.3	5	5	.948	.942	**.531**	.189	**.501**	.368	**.463**	.173
diabetes	614	154	8	2	0.2	5	5	.688	.747	**.206**	.065	**.397**	.206	**.203**	.060
ionosphere	281	70	34	2	0.2	4	4	.986	.929	**.388**	.109	**.408**	.113	**.358**	.096

Table 2.2: Test accuracy and robustness of information gain based single decision tree model. The robustness is evaluated by the average ℓ_∞ distortion of adversarial examples found by Cheng's, Papernot's and Kantchelian's attacks. Average ℓ_∞ distortion of robust decision tree models found by three attack methods are consistently larger than that of the naturally trained ones.

2.5.2 Robust GBDT Models

In this subsection, we evaluate our algorithm in the tree boosting setting, where multiple robust decision trees are created in an ensemble to improve model accuracy. We implement Algorithm 2 by slightly modifying the node splitting procedure in XGBoost. Our modification is only relevant to computing the scores for selecting the best split, and is compatible with other existing features of XGBoost. We also use XGBoost to train natural (undefended) GBDT models. Again, we focus on untargeted adversarial attacks. We consider nine real world large or medium sized datasets and two small datasets [25], spanning a variety of data types (including both tabular and image data). For small datasets we use 100 examples and for large or medium sized datasets, we use 5000 examples for robustness evaluation, except for MNIST 2 vs. 6, where we use 100 examples. MNIST 2 vs. 6 is a subset of MNIST to only distinguish between 2 and 6. This is the dataset tested in [75]. We use the same number of trees, depth and step size shrinkage as in [75] to train our robust and natural models. Same as in [75], we only test 100 examples for MNIST 2 vs. 6 since the model is relatively large. In Table 2.3, we present the average ℓ_∞ distortion of adversarial examples found by Cheng's ℓ_∞ attack for both natural GBDT and robust GBDT models trained on those datasets. For small and medium binary classification models, we also present results of Kantchelian's ℓ_∞ attack, which finds the *minimum* adversarial example in ℓ_∞ norm. The ℓ_1 and ℓ_2 distortion found by Kantchelian's ℓ_1 and ℓ_2 attacks are presented in Table A.2 in Appendix A. Kantchelian's attack can only handle binary classification problems and small scale models due to its time-consuming MILP formulation. Papernot's attack is inapplicable here because it is for attacking a single tree only. The natural and robust models have the same

number of trees for comparison. We only attack correctly classified images and all examples are successfully attacked. We see that our robust GBDT models consistently outperform the natural GBDT models in terms of ℓ_∞ robustness.

For some datasets, we need to increase tree depth in robust GBDT models in order to obtain accuracy comparable to the natural GBDT models. The requirement of larger model capacity is common in the adversarial training literature: in the state-of-the-art defense for DNNs, [95] argues that increasing the model capacity is essential for adversarial training to obtain good accuracy.

Dataset	training set size	test set size	# of features	# of classes	# of trees	robust ϵ	depth		test acc.		**avg. ℓ_∞ dist. by Cheng's ℓ_∞ attack**		dist. improv.	**avg. ℓ_∞ dist. by Kantchelian's ℓ_∞ attack**		dist. improv.
							robust	natural	robust	natural	robust	natural		robust	natural	
breast-cancer	546	137	10	2	4	0.3	8	6	.978	.964	**.411**	.215	**1.91X**	**.406**	.201	**2.02X**
covtype	400,000	181,000	54	7	80	0.2	8	8	.847	.877	**.081**	.061	**1.31X**	not binary	not binary	—
cod-rna	59,535	271,617	8	2	80	0.2	5	4	.880	.965	**.062**	.053	**1.16X**	**.054**	.034	**1.59X**
diabetes	614	154	8	2	20	0.2	5	5	.786	.773	**.139**	.060	**2.32X**	**.114**	.047	**2.42X**
Fashion-MNIST	60,000	10,000	784	10	200	0.1	8	8	.903	.903	**.156**	.049	**3.18X**	not binary	not binary	—
HIGGS	10,500,000	500,000	28	2	300	0.05	8	8	.709	.760	**.022**	.014	**1.57X**	time out	time out	—
ijcnn1	49,990	91,701	22	2	60	0.1	8	8	.959	.980	**.054**	.047	**1.15X**	**.037**	.031	**1.19X**
MNIST	60,000	10,000	784	10	200	0.3	8	8	.980	.980	**.373**	.072	**5.18X**	not binary	not binary	—
Sensorless	48,509	10,000	48	11	30	0.05	6	6	.987	.997	**.035**	.023	**1.52X**	not binary	not binary	—
webspam	300,000	50,000	254	2	100	0.05	8	8	.983	.992	**.049**	.024	**2.04X**	time out	time out	—
MNIST 2 vs. 6	11,876	1,990	784	2	1000	0.3	6	4	.997	.998	**.406**	.168	**2.42X**	**.315**	.064	**4.92X**

Table 2.3: The test accuracy and robustness of GBDT models. Average ℓ_∞ distortion of our robust GBDT models are consistently larger than those of the naturally trained models. The robustness is evaluated by the average ℓ_∞ distortion of adversarial examples found by Cheng's and Kantchelian's attacks. Only small or medium sized binary classification models can be evaluated by Kantchelian's attack, but it finds the minimum adversarial example with smallest possible distortion.

Figure 2-3 and Figure A-2 in Appendix A show the distortion and accuracy of MNIST and Fashion-MNIST models with different number of trees. The adversarial examples are found by Cheng's ℓ_∞ attack. Models with k trees are the first k trees during a single boosting run of K ($K \geq k$) trees. The ℓ_∞ distortion of robust models are consistently much larger than those of the natural models. For MNIST dataset, our robust GBDT model loses accuracy slightly when the model has only 20 trees. This loss is gradually compensated as more trees are added to the model; regardless of the number of trees in the model, the robustness improvement is consistently observed, as our robust training is embedded in each tree's building process and we create robust trees beginning from the very first step of boosting. Adversarial training in [75], in contrast, adds adversarial examples with respect to the current model at each boosting round so adversarial examples produced in the later stages of boosting are only learned by part of the model. The non-robust trees in the first

few rounds of boosting still exist in the final model and they may be the weakness of the ensemble. Similar problems are not present in DNN adversarial training since the whole model is exposed to new adversarial examples throughout the training process. This may explain why adversarial training in [75] failed to improve ℓ_1, ℓ_2, or ℓ_∞ robustness on the MNIST 2 vs. 6 model, while our method achieves significant robustness improvement with the same training parameters and evaluation metrics, as shown in Tables 2.3 and A.2. Additionally, we also evaluate the robustness of natural and robust models with different number of trees on a variety of datasets using Cheng's ℓ_∞ attack, presented in Table A.4 in Appendix A. We also test our framework on random forest models and the results are shown in Appendix A.7.

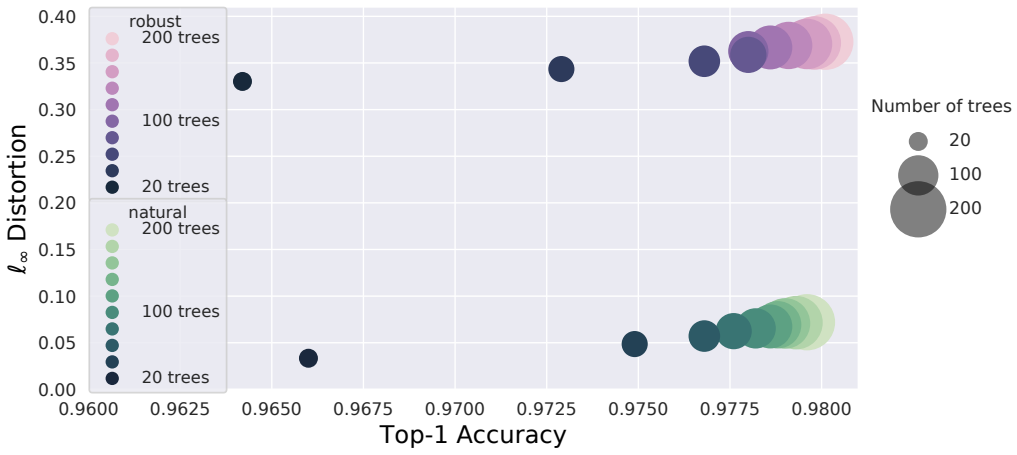


Figure 2-3: (Best viewed in color) ℓ_∞ distortion vs. classification accuracy of GBDT models on MNIST dataset with different numbers of trees (circle size). The adversarial examples are found by Cheng's ℓ_∞ attack. The robust training parameter $\epsilon = 0.3$ for MNIST. With robust training (purple) the distortion needed to fool a model increases dramatically with less than 1% accuracy loss.

2.6 Discussion and Remarks

In this chapter, we study the robustness of tree-based machine learning models under adversarial attacks. Our experiments show that just as in DNNs, tree-based models are also vulnerable to adversarial attacks. To address this issue, we propose a novel robust decision tree training framework. We make necessary approximations to ensure scalability and implement our framework in both classical decision tree and tree boosting settings. Extensive experiments on a variety of datasets show that our

method substantially improves model robustness. Our framework can be extended to other tree-based models such as Gini impurity based classification trees, random forest, and CART.

Chapter 3

Robustness Verification of Tree Based Models

3.1 Introduction

In this chapter, We study the robustness verification problem of tree-based models, including a single decision tree and tree ensembles such as random forests (RFs) and gradient boosted decision trees (GBDTs). These models have been widely used in practice [31, 77, 181] and recent studies have demonstrated that both RFs and GBDTs are vulnerable to adversarial perturbations [75, 33, 26]. It is thus important to develop a formal robustness verification algorithm for tree-based models. Robustness verification requires computing the minimal adversarial perturbation. Ref. [75] showed that computing minimal adversarial perturbation for tree ensemble is NP-complete in general, and they proposed a Mixed-Integer Linear Programming (MILP) based approach to compute the minimal adversarial perturbation. Although exact verification is NP-hard, in order to have an efficient verification algorithm for real applications we seek to answer the following questions:

- Do we have polynomial time algorithms for exact verification under some special circumstances?
- For general tree ensemble models with a large number of trees, can we efficiently compute meaningful lower bounds on robustness while scaling to large tree ensem-

bles?

In this chapter, we answer the above-mentioned questions affirmatively by formulating the verification problem of tree ensembles as a graph problem. First, we show that for a single decision tree, robustness verification can be done exactly in linear time. Then we show that for an ensemble of K trees, the verification problem is equivalent to finding the maximum cliques in a K-partite graph, and the graph is in a special form with boxicity equal to the input feature dimension. Therefore, for low-dimensional problems, verification can be done in polynomial time with maximum clique searching algorithms. Finally, for large-scale tree ensembles, we propose a multiscale verification algorithm by exploiting the boxicity of the graph, which can give tight lower bounds on robustness. Furthermore, it supports any-time termination: we can stop the algorithm at any time to obtain a reasonable lower bound given a computation time constraint. Our proposed algorithm is efficient and is scalable to large tree ensemble models. For instance, on a large multi-class GBDT with 200 trees robustly trained (using [26]) on the MNIST dataset, we obtained 78% verified robustness accuracy on test set with maximum ℓ_∞ perturbation $\epsilon = 0.2$ and the time used for verifying each test example is 12.6 seconds, whereas the MILP method uses around 10 min for each test example. The material presented in this chapter are based on [28].

3.2 Related Works

Unlike neural networks, decision-tree based models are non-continuous step functions, and thus existing neural network verification techniques cannot be directly applied. In [5], a single decision tree was verified to evaluate the robustness of reinforcement learning policies. For tree ensembles, [75] showed that solving (1.1) for general tree ensemble models is NP-complete, so no polynomial time algorithm can compute r^* for arbitrary trees unless P=NP. A Mixed Integer Linear Programming (MILP) algorithm was thus proposed in [75] to compute (1.1) in exponential time. Recently, [41] and [127] verified the robustness of tree ensembles using an SMT solver, which is also

NP-complete in its natural formulation. Additionally, an approximate bound for tree ensembles was proposed recently in [149] by directly combining the bounds of each tree together, which can be seen as a special case of our proposed method.

On the other hand, robustness can be empirically evaluated through adversarial attacks [110]. Some hard-label attacking algorithms for neural networks, including the boundary attack [9] and OPT-attack [33], can be applied to tree based models since they only require function evaluation of the non-smooth (hard-label) decision function $f(\cdot)$. These attacks computes an upper bound of r^*. In contrast, our work focuses on efficiently computing a tight lower bound of r^* for ensemble trees.

3.3 Proposed Algorithm

In this chapter, we focus on ℓ_∞ norm verification. The exact verification problem of tree ensemble is NP-complete by its nature, and here we propose a series of efficient verification algorithms for real applications. First, we will introduce a linear time algorithm for exactly computing the minimal adversarial distortion r^* for verifying a single decision tree. For an ensemble of trees, we cast the verification problem into a max-clique searching problem in K-partite graphs. For large-scale tree ensembles, we then propose an efficient multi-level algorithm for verifying an ensemble of decision trees.

3.3.1 Exactly Verifying a Single Tree in Linear Time

Although computing r^* for a tree ensemble is NP-complete [75], we show that a **linear time** algorithm exists for finding the minimum adversarial perturbation and computing r^* for a single decision tree. We assume the decision tree has n nodes and the root node is indexed as 0. For a given example $x = [x_1, \ldots, x_d]$ with d features, starting from the root, x traverses the decision tree model until reaching a leaf node. Each internal node, say node i, has two children and a univariate feature-threshold pair (t_i, η_i) to determine the traversal direction—x will be passed to the left child if $x_{t_i} \leq \eta_i$ and to the right child otherwise. Each leaf node has a value v_i corresponding

to the predicted class label for a classification tree, or a real value for a regression tree.

Conceptually, the main idea of our single tree verification algorithm is to compute a d-dimensional box for each leaf node such that any example in this box will fall into this leaf. Mathematically, the node i's box is defined as the Cartesian product $B^i = (l_1^i, r_1^i] \times \cdots \times (l_d^i, r_d^i]$ of d intervals on the real line. By definition, the root node has box $[-\infty, \infty] \times \cdots \times [-\infty, \infty]$ and given the box of an internal node i, its children's boxes can be obtained by changing only one interval of the box based on the split condition (t_i, η_i). More specifically, if p, q are node i's left and right child node respectively, then we set their boxes $B^p = (l_1^p, r_1^p] \times \cdots \times (l_d^p, r_d^p]$ and $B^q = (l_1^q, r_1^q] \times \cdots \times (l_d^q, r_d^q]$ by setting

$$(l_t^p, r_t^p] = \begin{cases} (l_t^i, r_t^i] & \text{if } t \neq t_i \\ (l_t^i, \min\{r_t^i, \eta_i\}] & \text{if } t = t_i \end{cases}, \quad (l_t^q, r_t^q] = \begin{cases} (l_t^i, r_t^i] & \text{if } t \neq t_i \\ (\max\{l_t^i, \eta_i\}, r_t^i] & \text{if } t = t_i. \end{cases} \tag{3.1}$$

After computing the boxes for internal nodes, we can also obtain the boxes for leaf nodes using (3.1). Therefore computing the boxes for all the leaf nodes of a decision tree can be done by a depth-first search traversal of the tree with time complexity $O(nd)$.

With the boxes computed for each leaf node, the minimum perturbation required to change x to go to a leaf node i can be written as a vector $\boldsymbol{\epsilon}(x, B^i) \in \mathbb{R}^d$ defined as

$$\boldsymbol{\epsilon}(x, B^i)_t := \begin{cases} 0 & \text{if } x_t \in (l_t^i, r_t^i] \\ x_t - r_t^i & \text{if } x_t > r_t^i \\ l_t^i - x_t & \text{if } x_t \leq l_t^i. \end{cases} \tag{3.2}$$

Then the minimal distortion can be computed as $r^* = \min_{i:v_i \neq y_0} \|\boldsymbol{\epsilon}(x, B^i)\|_\infty$, where y_0 is the original label of x, and v_i is the label for leaf node i. To find r^*, we check B^i for all leaves and choose the smallest perturbation. This is a linear-time algorithm for exactly verifying the robustness of a single decision tree. In fact, this $O(nd)$ time algorithm is used to illustrate the concept of "boxes" that will be used later on for

the tree ensemble case. If our final goal is to verify a single tree, we can have a more efficient algorithm by combining the distance computation (4.2) in the tree traversal procedure, and the resulting algorithm will take only $O(n)$ time. This algorithm is presented as Algorithm 6 in Appendix B.

3.3.2 Verifying Tree Ensembles by Max-clique Enumeration

Now we discuss the robustness verification for tree ensembles. Assuming the tree ensemble has K decision trees, we use $S^{(k)}$ to denote the set of leaf nodes of tree k and $m^{(k)}(x)$ to denote the function that maps the input example x to the leaf node of tree k according to its traversal rule. Given an input example x, the tree ensemble will pass x to each of these K trees independently and x reaches K leaf nodes $i^{(k)} = m^{(k)}(x)$ for all $k = 1, \ldots, K$. Each leaf node will assign a prediction value $v_{i^{(k)}}$. For simplicity we start with the binary classification case, with x's original label being $y_0 = -1$ and we want to turn it into $+1$. For binary classification the prediction of the tree ensemble is computed by $\text{sign}(\sum_k v_{i^{(k)}})$, which covers both GBDTs and random forests, two widely used tree ensemble models. Assume x has a label $y_0 = -1$, which means $\text{sign}(\sum_k v_{i^{(k)}}) < 0$ for x, and our task is to verify if the sign of the summation can be flipped within $\text{Ball}(x, \epsilon)$.

We consider the decision problem of robustness verification (1.5). A naive analysis will need to check all the points in $\text{Ball}(x, \epsilon)$ which is uncountably infinite. To reduce the search space to finite, we start by defining some notation: let $\mathbb{C} = \{(i^{(1)}, \ldots, i^{(K)}) \mid i^{(k)} \in S^{(k)}, \ \forall k = 1, \ldots, L\}$ to be all the possible tuples of leaf nodes and let $\mathcal{C}(x) = [m^{(1)}(x), \ldots, m^{(K)}(x)]$ be the function that maps x to the corresponding leaf nodes. Therefore, a tuple $C \in \mathbb{C}$ directly determines the model prediction $\sum v_C := \sum_k v_{i^{(k)}}$. Now we define a valid tuple for robustness verification:

Definition 1. *A tuple $C = (i^{(1)}, \ldots, i^{(K)})$ is valid if and only if there exists an $x' \in Ball(x, \epsilon)$ such that $C = \mathcal{C}(x')$.*

The decision problem of robustness verification (1.5) can then be written as:

$$\textit{Does there exist a valid tuple } C \textit{ such that } \sum v_C > 0?$$

Next, we show how to model the set of valid tuples. We have two observations. First, if a tuple contains any node i with $\inf_{x' \in B^i}\{\|x - x'\|_\infty\} > \epsilon$, then it will be invalid. Second, there exists an x such that $C = \mathcal{C}(x)$ if and only if $B^{i^{(1)}} \cap \cdots \cap B^{i^{(K)}} \neq \emptyset$, or equivalently:

$$(l_t^{i^{(1)}}, r_t^{i^{(1)}}] \cap \cdots \cap (l_t^{i^{(K)}}, r_t^{i^{(K)}}] \neq \emptyset, \quad \forall t = 1, \ldots, d.$$

We show that the set of valid tuples can be represented as cliques in a graph $G = (V, E)$, where $V := \{i | B^i \cap \text{Ball}(x, \epsilon) \neq \emptyset\}$ and $E := \{(i, j) | B^i \cap B^j \neq \emptyset\}$. In this graph, nodes are the leaves of all trees and we remove every leaf that has empty intersection with $\text{Ball}(x, \epsilon)$. There is an edge (i, j) between node i and j if and only if their boxes intersect. The graph will then be a K-partite graph since there cannot be any edge between nodes from the same tree, and thus maximum cliques in this graph will have K nodes. We define each part of the K-partite graph as V_k. Here a "part" means a disjoint and independent set in the K-partite graph. The following lemma shows that intersections of boxes have very nice properties:

Lemma 1. *For boxes $B^1, \ldots, B^K$, if $B^i \cap B^j \neq \emptyset$ for all $i, j \in [K]$, let $\bar{B} = B^1 \cap B^2 \cap \cdots \cap B^K$ be their intersection. Then $\bar{B}$ will also be a box and $\bar{B} \neq \emptyset$.*

The proof can be found in the Appendix B. Based on the above lemma, each K-clique (fully connected subgraph with K nodes) in G can be viewed as a set of leaf nodes that has nonempty intersection with each other and also has nonempty intersection with $\text{Ball}(x, \epsilon)$, so the intersection of those K boxes and $\text{Ball}(x, \epsilon)$ will be a nonempty box, which implies each K-clique corresponds to a valid tuple of leaf nodes:

Lemma 2. *A tuple $C = (i^{(1)}, \ldots, i^{(K)})$ is valid if and only if nodes $i^{(1)}, \ldots, i^{(K)}$ form a K-clique (maximum clique) in graph G constructed above.*

Therefore the robustness verification problem can be formulated as

$$\textit{Is there a maximum clique } C \textit{ in } G \textit{ such that } \sum v_C > 0 \textit{?} \tag{3.3}$$

This reformulation indicates that the tree ensemble verification problem can be solved by an efficient maximum clique enumeration algorithm. Some standard maximum clique searching algorithms can be applied here to perform verification:

- **Finding K-cliques in K-partite graphs:** Any algorithm for finding all the maximum cliques in G can be used. The classic B-K backtracking algorithm [11] takes $O(3^{\frac{m}{3}})$ time to find all the maximum cliques where m is the number of nodes in G. Furthermore, since our graph is a K-partite graph, we can apply some specialized algorithms designed for finding all the K-cliques in K-partite graphs [102, 115, 130].
- **Polynomial time algorithms exist for low-dimensional problems:** Another important property for graph G is that each node in G is a d-dimensional box and each edge indicates the intersection of two boxes. This implies our graph G is with "boxicity d" (see [24] for detail). Ref. [24] proved that the number of maximum cliques will only be $O((2m)^d)$ and it is able to find the maximum weight clique in $O((2m)^d)$ time. Therefore, for problems with a very small d, the time complexity for verification is actually polynomial.

Therefore we can exactly solve the tree ensemble verification problem using algorithms for maximum cliques searching in K-partite graph, and its time complexity is found to be as follows:

Theorem 2. *Exactly verifying the robustness of a K-tree ensemble with at most n leaves per tree and d dimensional features takes* $\min\{O(n^K), O((2Kn)^d)\}$ *time.*

This is a direct consequence of the fact that the number of K-cliques in a K-partite graph with n vertices per part is bounded by $O(n^K)$, and number of maximum cliques in a graph with a total of m nodes with boxicity d is $O((2m)^d)$.

Algorithm 3 Enumerating all K-cliques on a K-partite graph

input : $V_1,\ V_2,\ ,\ldots,\ V_K$ are the K independent sets ("parts") of a K-partite graph

for $k \leftarrow 1,\ 2,\ 3,\ \ldots,\ K$ **do**

$U_k \leftarrow \{(A_i,\ B^{i^{(k)}})|i^{(k)} \in V_k,\ A_i = \{i^{(k)}\}\}$

/* U is a set of tuples (A, B), which stores a set of cliques and their corresponding boxes. A is the set of nodes in one clique and B is the corresponding box of this clique. Initially, each node in V_k forms a 1-clique itself. */

end

output : CliqueEnumerate($U_1,\ U_2,\ ,\ldots,\ U_K$)

Function CliqueEnumerate($U_1,\ U_2,\ ,\ldots,\ U_K$)

$\hat{U}_{\text{old}} \leftarrow U_1$

for $k \leftarrow 2,\ 3,\ \ldots,\ K$ **do**

$\hat{U}_{\text{new}} \leftarrow \emptyset$

for $(\hat{A},\ \hat{B}) \in \hat{U}_{old}$ **do**

for $(A,\ B) \in U_k$ **do**

if $B \cap \hat{B} \neq \emptyset$ **then**

/* A k-clique is found; add it as a pseudo node with the intersection of two boxes. */

$\hat{U}_{\text{new}} \leftarrow \hat{U}_{\text{new}} \cup \{(A \cup \hat{A},\ B \cap \hat{B})\}$

end

end

$\hat{U}_{\text{old}} \leftarrow \hat{U}_{\text{new}}$

end

return $\hat{U}_{\text{new}}$

end

For a general graph, since K and d can be in $O(n)$ and $O(m)$ [123], it can still be exponential. But the theorem gives a more precise characterization for the complexity of the verification problem for tree ensembles. Based on the nice properties of

maximum cliques searching problem, we propose a simple and elegant algorithm that enumerates all K-cliques on a K-partite graph with a known boxicity d in Algorithm 3, and we can use this algorithm for tree ensemble verification when the number of trees or the dimension of features is small. For a K-partite graph G, we define the set $\tilde{V} := \{V_1, V_2, \cdots, V_K\}$ which is a set of independent sets ("parts") in G.

The algorithm first looks at any first two parts V_1 and V_2 of the graph and enumerates all 2-cliques in $O(|V_1||V_2|)$ time. Then, each 2-clique found is converted into a "pseudo node" (this is possible due to Lemma 1), and all 2-cliques form a new part V_2' of the graph. Then we replace V_1 and V_2 with V_2', and continue to enumerate all 2-cliques between V_2' and V_3 to form V_3'. A 2-clique between V_2' and V_3 represents a 3-clique in V_1, V_2 and V_3 due to boxicity. Note that enumerating all 3-cliques in a general 3-partite graph takes $O(|V_1||V_2||V_3|)$ time; thanks to boxicity, our algorithm takes $O(|V_2'||V_3|)$ time which equals to $O(|V_1||V_2||V_3|)$ only when V_1 and V_2 form a complete bipartite graph, which is unlikely in common cases. This process continues recursively until we process all K parts and have only V_K' left, where each vertex in V_K' represents a K-clique in the original graph. After obtaining all K-cliques, we can verify their prediction values to compute a verification bound.

3.3.3 An Efficient Multi-level Algorithm for Verifying the Robustness of a Tree Ensemble

Practical tree ensembles usually have tens or hundreds of trees with large feature dimensions, so Algorithm 3 will take exponential time and will be too slow. We thus develop an efficient multi-level algorithm for computing verification bounds by further exploiting the boxicity of the graph.

Figure 3-1 illustrates the graph and how our multilevel algorithm runs. There are four trees and each tree has four leaf nodes. A node is colored if it has nonempty intersection with Ball(x, ϵ); uncolored nodes are discarded. To answer question (3.3), we need to compute the maximum $\sum v_C$ among all K-cliques, denoted by v^*. As mentioned before, for robustness verification we only need to compute an upper bound

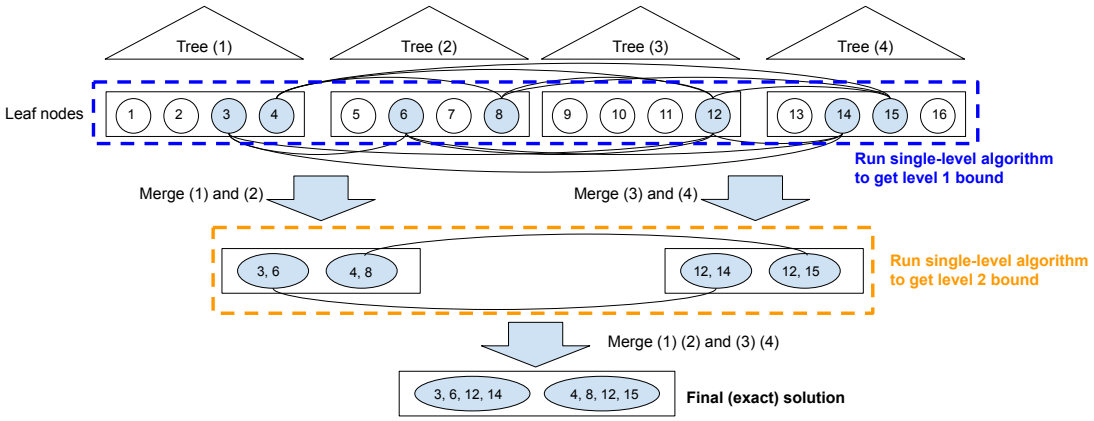


Figure 3-1: The proposed multi-level verification algorithm. Lines between leaf node i on tree t_1 and leaf node j on t_2 indicate that their ℓ_∞ feature boxes intersect (i.e., there exists an input such that tree 1 predicts v_i and tree 2 predicts v_j).

of v^* in order to get a lower bound of minimal adversarial perturbation. In the following, we will first discuss algorithms for computing an upper bound at the top level, and then show how our multi-scale algorithm iteratively refines this bound until reaching the exact solution v^*.

Bounds for a single level – To compute an upper bound of v^*, a naive approach is to assume that the graph is fully connected between independent sets (fully connected K-partite graph) and in this case the maximum sum of node values is the sum of the maximum value of each independent set:

$$\sum_{k=1}^{|\tilde{V}|} \max_{i \in V_k} v_i \geq v^*. \tag{3.4}$$

Here we abuse the notation v_i by assuming that each node i in V_k has been assigned a "pseudo prediction value", which will be used in the multi-level setting. In the simplest case, each independent set represents a single tree, $V_k = S^{(k)}$ and v_i is the prediction of a leaf. One can easily show this is an upper bound of v^* since any K-clique in the graph is still considered when we add more edges to the graph, and eventually it becomes a fully connected K-partite graph.

Another slightly better approach is to exploit the edge information but only between tree t and $t+1$. If we search over all the length-K paths $[i^{(1)}, \ldots, i^{(K)}]$ from the first to the last part and define the value of a path to be $\sum_k v_{i^{(k)}}$, then the maximum valued path will be a upper bound of v^*. This can be computed in linear

time using dynamic programming. We scan nodes from tree 1 to tree K, and for each node we store a value d_i which is the maximum value of paths from tree 1 to this node. At tree k and node i, the d_i value can be computed by

$$d_i = v_i + \max_{j:j\in V_{k-1} \text{ and } (j,i)\in E} d_j. \tag{3.5}$$

Then we take the max d value in the last tree. It produces an upper bound of v^*, since the maximum valued path found by dynamic programming is not necessarily a K-clique. Again $V_{k-1} = S^{(k-1)}$ in the first level but it will be generalized below.

Merging T independent sets – To refine the relatively loose single-level bound, we partition the graph into K/T subgraphs, each with T independent sets. Within each subgraph, we find all the T-cliques and use a new "pseudo node" to represent each T-clique. T-cliques in a subgraph can be enumerated efficiently if we choose T to be a relatively small number (e.g., two or three in the experiments).

Now we exploit the boxicity property to form a new graph among these T-cliques (illustrated as the second level nodes in Figure 3-1). By Lemma 1, we know that the intersection of T boxes will still be a box, so each T-clique is still a box and can be represented as a pseudo node in the level-2 graph. Also because each pseudo node is still a box, we can easily form edges between pseudo nodes to indicate the nonempty overlapping between them and this will be a (K/T)-partite boxicity graph since no edge can be formed for the cliques within the same subgraph. Thus we get the level-2 graph. With the level-2 graph, we can again run the single level algorithm to compute a upper bound on v^* to get a lower bound of r^* in (1.1), but different from the level-1 graph, now we already considered all the within-subgraph edges so the bounds we get will be tighter.

The overall multi-level framework – We can run the algorithm level by level until merging all the subgraphs into one, and in the final level the pseudo nodes will correspond to the K-cliques in the original graph, and the maximum value will be exactly v^*. Therefore, our algorithm can be viewed as an anytime algorithm that refines the upper bound level-by-level until reaching the maximum value. Although

getting to the final level still requires exponential time, in practice we can stop at any level (denoted as L) and get a reasonable bound. In experiments, we will show that by merging few trees we already get a bound very close to the final solution. Algorithm 4 gives the complete procedure.

Handling multi-class tree ensembles – For a multiclass classification problem, say a C-class classification problem, C groups of tree ensembles (each with K trees) are built for the classification task; for the k-th tree in group c, prediction outcome is denoted as $i^{(k,c)} = m^{(k,c)}(x)$ where $m^{(k,c)}(x)$ is the function that maps the input example x to a leaf node of tree k in group c. The final prediction is given by $\text{argmax}_c \sum_k v_{i^{(k,c)}}$. Given an input example x with ground-truth class c and an attack target class c', we extract $2K$ trees for class c and class c', and flip the sign of all prediction values for trees in group c', such that initially $\sum_t v_{i^{(t,c)}} + \sum_t v_{i^{(t,c')}} < 0$ for a correctly classified example. Then, we are back to the binary classification case with $2K$ trees, and we can still apply our multi-level framework to obtain a lower bound $\underline{r}_{(c,c')}$ of $r^*_{(c,c')}$ for this target attack pair (c, c'). Robustness of an untargeted attack can be evaluated by taking $\underline{r} = \min_{c' \neq c} \underline{r}_{(c,c')}$.

3.3.4 Verification Problems Beyond Ordinary Robustness

The above discussions focus on the decision problem of ℓ_∞ robustness verification (1.5). In fact, our approach works for a more general verification problem for *any* d-dimensional box B:

$$\textit{Is there any } x' \in B \textit{ such that } f(x') \neq y_0 \textit{?} \tag{3.6}$$

In typical robustness verification settings, B is defined to be $\text{Ball}(x, \epsilon)$ but in fact we can allow any boxes in our algorithm. For a general B, Lemma 1 still holds so all of our algorithms and analysis can go through. The only change is to compute the intersection between B and each box of leaf node at the first level in Figure 3-1 and eliminate nodes that have an empty intersection with B. So robustness verification is just a special case where we remove all the nodes with empty intersection with

Ball(x, ϵ). For example, we can identify a set of unimportant variables, where any individual feature change in this set cannot alter the prediction for a given sample x. For each feature i, we can choose B as $B_i = [-\infty, \infty]$ (or the the entire input domain, like $[0, 1]$ for image data) and $B_{j \neq i} = \{x_j\}$ otherwise. If the model is robust to such a single-feature perturbation, then this feature is added to the unimportant set. Similarly, we can get a set of anchor features (similar to [122]) such that once a set of features are fixed, any perturbation outside the set cannot change the prediction.

Algorithm 4 Multi-level verification framework

input : The set of leaf nodes of each tree, $S^{(1)},\ S^{(2)},\ ,\ldots,\ S^{(K)}$; maximum number of independent sets in a subgraph (denoted as T); maximum number of levels (denoted as L), $L \leq \lceil \log_T(K) \rceil$

for $k \leftarrow 1,\ 2,\ \ldots,\ K$ **do**

  /* U is defined the same as in Algorithm 3. At level 0, each V_k forms a 1-clique by itself. */

  $U_k^{(0)} \leftarrow \{(A_i,\ B^{i^{(k)}}) | i^{(k)} \in S^{(k)},\ A_i = \{i^{(k)}\}\}$

end

for $l \leftarrow 1,\ 2,\ \ldots,\ L$ **do**

  /* Enumerate all cliques in each subgraph at this level. Total $\lceil K/T^l \rceil$ subgraphs. */

  for $k \leftarrow 1,\ 2,\ \ldots,\ \lceil K/T^l \rceil$ **do**

    $U_k^{(l)} \leftarrow$ `CliqueEnumerate`$(U_{(k-1)T+1}^{(l-1)},\ U_{(k-1)T+2}^{(l-1)},\ \ldots,\ U_{kT}^{(l-1)})$

  end

end

for $k \leftarrow 1,\ 2,\ \ldots,\ \lceil K/T^L \rceil$ **do**

  /* Define an independent set V'_k for each $U_k^{(L)}$. In each V'_k, we create "pseudo nodes" which combines multiple nodes from lower levels, and assign "pseudo prediction values" to them. */

  $V'_k \leftarrow \{A \big| (A, B) \in U_k^{(L)}\}$; /* V'_k is a set of sets; each element in V'_k represents a clique. */

  /* Construct the "pseudo prediction value" for each element in V'_k by summing up all prediction values in the corresponding clique. */

  For all $A \in V'_k$: $v_A \leftarrow \sum_{i \in A} v_i$

end

$\bar{v} \leftarrow$ an upper bound of v^* using (3.4) or (3.5), given $\tilde{V} = \{V'_1, \cdots, V'_{\lceil K/T^L \rceil}\}$

/* If $\lceil K/T^L \rceil = 1$, only one independent set left and each pseudo node represents a K-clique; (3.4) or (3.5) will have a trivial solution where v^* is the maximum v_A in $U_1^{(L)}$. */

3.4 Experiments

We evaluate our proposed method for robustness verification of tree ensembles on two tasks: binary and multiclass classification on 9 public datasets including both small and large scale datasets.[1] We run our experiments on Intel Xeon Platinum 8160 CPUs. The datasets other than MNIST and Fashion-MNIST are from LIBSVM [25]. The statistics of the data sets are shown in Appendix B.1. As we defined in Section 3.2, r^* is the radius of minimum adversarial perturbation that reflects true model robustness, but is hard to obtain; our method finds $\underline{r}$ that is a lower bound of r^*, which guarantees that *no adversarial example* exists within radius $\underline{r}$. A high quality lower bound $\underline{r}$ should be close to r^*. We include the following algorithms in our comparisons:

- Cheng's attack [33] provides results on adversarial attacks on these models, which gives an upper bound of the model robustness r^*. We denote it as $\overline{r}$ and $\overline{r} \geq r^*$.
- MILP: an MILP (Mixed Integer Linear Programming) based method [75] gives the exact r^*. It can be very slow when the number of trees or dimension of the features increases.
- LP relaxation: a Linear Programming (LP) relaxed MILP formulation by directly changing all binary variables to continuous ones. Since the binary constraints are removed, solving the minimization of MILP gives a lower bound of robustness, $\underline{r}_{LP}$, serving as a baseline method.
- Our proposed multi-level verification framework in Section 3.3.3 (with pseudo code as Algorithm 4 in the appendix). We are targeting to compute robustness interval $\underline{r}_{our}$ for tree ensemble verification.

In Tables 3.1 and 3.2 we show empirical comparisons on 9 datasets. We consider ℓ_∞ robustness, and normalize our datasets to $[0,\ 1]$ such that perturbations on different datasets are comparable. We use (3.4) to obtain single layer bounds. Results using dynamic programming in (3.5) are provided in Appendix B.2. We include both

[1] Our code (XGBoost compatible) is available at https://github.com/chenhongge/treeVerification

standard (naturally trained) GBDT models (Table 3.1) and robust GBDT models [26] (Table 3.2). The robust GBDTs were trained by considering model performance under the worst-case perturbation, which leads to a max-min saddle point problem when finding the optimal split at each node [26]. All GBDTs are trained using the XGBoost framework [31]. The number of trees in GBDTs and parameters used in training GBDTs for different datasets are shown in Table B.1 in the appendix. Because we solve the decision problem of robustness verification, we use a 10-step binary search to find the largest $\underline{r}$ in all experiments, and the reported time is the total time including all binary search trials. We present the average of $\underline{r}$ or r^* over 500 examples. The MILP based method from [75] is an accurate but very slow method; the results marked with a star ($\star$) in the table have very long running time and thus we only evaluate 50 examples instead of 500.

Dataset	Cheng's attack [33]		MILP [75]		LP relaxation		Ours (without DP)				Ours vs. MILP	
	avg. $\bar{r}$	avg. time	avg. r^*	avg. time	avg. $\underline{r}_{LP}$	avg. time	T	L	avg. $\underline{r}_{our}$	avg. time	$\underline{r}_{our}/r^*$	speedup
breast-cancer	.221	2.18s	.210	.012s	.064	.009s	2	1	.208	.001s	.99	12X
covtype	.058	4.76s	.028*	355*s	.005*	154*s	2	3	.022	3.39s	.79	105X
diabetes	.064	1.70s	.049	.061s	.015	.026s	3	2	.042	.018s	.86	3.4X
Fashion-MNIST	.048	12.2s	.014*	1150*s	.003*	898*s	2	1	.012	11.8s	.86	97X
HIGGS	.015	3.80s	.0028*	68*min	.00035*	50*min	4	1	.0022	1.29s	.79	3163X
ijcnn1	.047	2.72s	.030	4.64s	.008	2.67s	2	2	.026	.101s	.87	4.6X
MNIST	.070	11.1s	.011*	367*s	.003*	332*s	2	2	.011	5.14s	1.00	71X
webspam	.027	5.83s	.00076	47.2s	.0002	39.7s	2	1	.0005	.404s	.66	117X
MNIST 2 vs. 6	.152	12.0s	.057	23.0s	.016	11.6s	4	1	.046	.585s	.81	39X

Table 3.1: Average ℓ_∞ distortion over 500 examples and average verification time per example for three verification methods. Here we evaluate the bounds for **standard (natural) GBDT models**. Results marked with a start ("$\star$") are the averages of 50 examples due to long running time. T is the number of independent sets and L is the number of levels in searching cliques used in our algorithm. A bound ratio close to 1 indicates better lower bound quality. Dynamic programming in (3.5) is not applied. Results using dynamic programming are provided in Appendix B.2.

Dataset	Cheng's attack [33]		MILP [75]		LP relaxation		Ours (without DP)				Ours vs. MILP	
	avg. $\bar{r}$	avg. time	avg. r^*	avg. time	avg. $\underline{r}_{LP}$	avg. time	T	L	avg. $\underline{r}_{our}$	avg. time	$\underline{r}_{our}/r^*$	speedup
breast-cancer	.404	1.96s	.400	.009s	.078	.008s	2	1	.399	.001s	1.00	9X
covtype	.079	.481s	.046*	305*s	.0053*	159*s	2	3	.032	4.84s	.70	63X
diabetes	.137	1.52s	.112	.034s	.035	.013s	3	2	.109	.006s	.97	5.7X
Fashion-MNIST	.153	13.9s	.091*	41*min	.009*	34*min	2	1	.071	18.0s	.78	137X
HIGGS	.023	3.58s	.0084*	59*min	.00031*	54*min	4	1	.0063	1.41s	.75	2511X
ijcnn1	.054	2.63s	.036	2.52s	.009	1.26s	2	2	.032	0.58s	.89	4.3X
MNIST	.367	1.41s	.264*	615*s	.019*	515*s	2	2	.253	12.6s	.96	49X
webspam	.048	4.97s	.015	83.7s	.0024	60.4s	2	1	.011	.345s	.73	243X
MNIST 2 vs. 6	.397	17.2s	.313	91.5s	.039	40.0s	4	1	.308	3.68s	.98	25X

Table 3.2: Verification bounds and running time for **robustly trained GBDT models** introduced in [26]. The settings for each method are similar to the settings in Table 3.1.

From Tables 3.1 and 3.2 we can see that our method gives a tight lower bound $\underline{r}$

compared to r^* from MILP, while achieving up to $\sim$ 3000X speedup on large models. The running time of the baseline LP relaxation, however, is on the same order of magnitude as the MILP method, but the results are much worse, with $\underline{r}_{LP} \ll r^*$. Figure 3-2 shows how the tightness of our robustness verification lower bounds changes with different size of clique per level (T) and different number of levels (L). We test on a 20-tree standard GBDT model on the diabetes dataset. We also show the exact bound r^* by the MILP method. Our verification bound converges to the MILP bound as more levels of clique enumerations are used. Also, when we use larger cliques in each level, the bound becomes tighter.

To show the scalability of our method, we vary the number of trees in GBDTs and compare per example running time with the MILP method on ijcnn1 dataset in Figure 3-3. We see that our multi-level method spends much less time on each example compared to the MILP method and our running time grows slower than MILP when the number of trees increases.

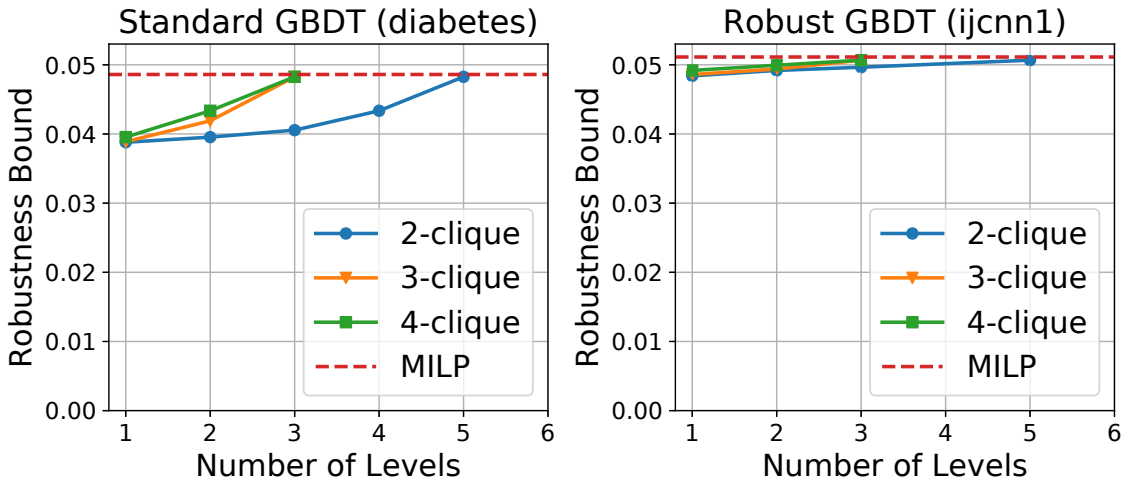


Figure 3-2: Robustness bounds obtained with different parameters ($T = \{2, 3, 4\}$, $L = \{1, \cdots, 6\}$) on a 20-tree standard GBDT model trained on diabetes dataset (left) and a 20-tree robust GBDT model trained on ijcnn1 dataset (right). The bounds obtained with our method converge to r^* as L increases.

In Section 3.3.4, we showed that our algorithm works for more general verification problems such as identifying unimportant features, where any changes on one of those features alone cannot alter the prediction. We use MNIST to demonstrate pixel importance, where we perturb each pixel individually by $\pm\epsilon$ while keeping other pixels unchanged, and obtain the largest ϵ such that prediction is unchanged. In

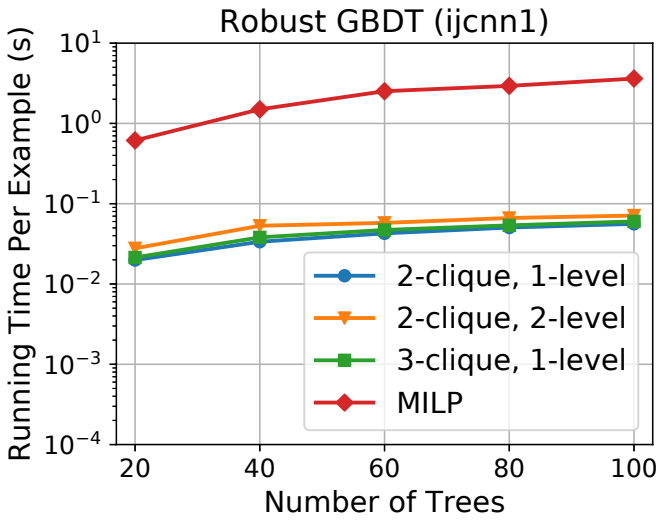


Figure 3-3: Running time of MILP and our method on robust GBDTs with different number of trees (ijcnn1 dataset).

Figure 3-4, yellow pixels cannot change prediction for any perturbation and a darker pixel represents a smaller lower bound $\underline{r}$ of perturbation to change the model output using that pixel. The standard naturally trained model has some very dark pixels compared to the robust model. Discussion on the connection between this score and other feature importance scores is in Section B.3 in the appendix.

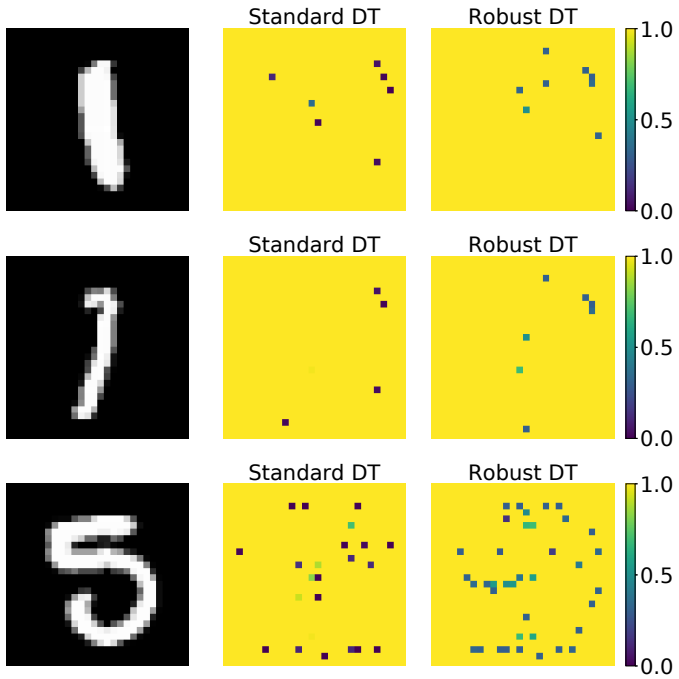


Figure 3-4: MNIST pixel importance. For each 3-image group, left: digit image; middle: results on standard DT model; right: results on robust DT model. Changing one of any yellow pixels to any valid values between 0 and 1 cannot alter model prediction; pixels in darker colors tend to affect model prediction more than pixels in lighter colors.

Chapter 4

The Limitations of Adversarial Training for Deep Neural Networks

4.1 Introduction

Since the discovery of adversarial examples in deep neural networks (DNNs) [146], adversarial training under the robustness optimization framework [95, 141] has become one of the most effective methods to defend against adversarial examples. A recent study by [2] showed that adversarial training does not rely on obfuscated gradients and delivers promising results for defending adversarial examples on small datasets. Adversarial training approximately solves the following min-max optimization problem:

$$\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\in\mathcal{X}} \left[\max_{\delta\in S} L(x+\delta; y; \theta) \right], \tag{4.1}$$

where $\mathcal{X}$ is the set of training data, L is the loss function, θ is the parameter of the network, and S is usually a norm constrained ℓ_p ball centered at 0. [95] propose to use projected gradient descent (PGD) to approximately solve the maximization problem within $S = \{\delta \mid \|\delta\|_\infty \leq \epsilon\}$, where $\epsilon = 0.3$ for MNIST dataset on a 0-1 pixel scale, and $\epsilon = 8$ for CIFAR-10 dataset on a 0-255 pixel scale. This approach achieves impressive defending results on the MNIST test set: so far the best available white-box attacks by

[184] can only decrease the test accuracy from approximately 98% to 88%[1]. However, on CIFAR-10 dataset, a simple 20-step PGD can decrease the test accuracy from 87% to less than 50%[2].

The effectiveness of adversarial training is measured by the robustness on the test set. However, the adversarial training process itself is done on the training set. Suppose we can optimize (4.1) perfectly, then certified robustness may be obtained on those training data points. However, if the empirical distribution of training dataset differs from the true data distribution, a test point drawn from the true data distribution might lie in a low probability region in the empirical distribution of training dataset and is not "covered" by the adversarial training procedure. For datasets that are relatively simple and have low intrinsic dimensions (MNIST, Fashion MNIST, etc), we can obtain enough training examples to make sure adversarial training covers most part of the data distribution. For high dimensional datasets (CIFAR, ImageNet), adversarial training have been shown difficult [81, 150] and only limited success was obtained.

A recent attack proposed by [143] shows that adversarial training can be defeated when the input image is produced by a generative model (for example, a generative adversarial network) rather than selected directly from the test examples. The generated images are well recognized by humans and thus valid images in the ground-truth data distribution. In our interpretation, this attack effective finds the "blind-spots" in the input space that the training data do not well cover.

For higher dimensional datasets, we hypothesize that many test images already fall into these blind-spots of training data and thus adversarial training only obtains a moderate level of robustness. It is interesting to see that for those test images that adversarial training fails to defend, if their distances (in some metrics) to the training dataset are indeed larger. In this chapter, we try to explain the success of robust optimization based adversarial training and show the limitations of this approach when the test points are slightly off the empirical distribution of training data. The materials in this chapter are based on [178]. Our main contributions are:

[1] https://github.com/MadryLab/mnist_challenge [2] https://github.com/MadryLab/cifar10_challenge

- We show that on the original set of test images, the effectiveness of adversarial training is highly correlated with the distance (in some distance metrics) from the test image to the manifold of training images. For MNIST and Fashion MNIST datasets [166], most test images are close to the training data and very good robustness is observed on these points. For CIFAR, there is a clear trend that the adversarially trained network gradually loses its robustness property when the test images are further away from training data.
- We identify a new class of attacks, "*blind-spot attacks*", where the input image resides in a "blind-spot" of the empirical distribution of training data (far enough from any training examples in some embedding space) but is still in the ground-truth data distribution (well recognized by humans and *correctly classified* by the model). Adversarial training cannot provide good robustness on these blind-spots and their adversarial examples have small distortions.
- We show that blind-spots can be easily found on a few strong defense models including [95], [162] and [141]. We propose a few simple transformations (slightly changing contrast and background), that do not noticeably affect the accuracy of adversarially trained MNIST and Fashion MNIST models, but these models become vulnerable to adversarial attacks on these sets of transformed input images. These transformations effectively move the test images slightly out of the manifold of training images, which does not affect generalization but poses a challenge for robust learning.
- Our results imply that current adversarial training procedures cannot scale to datasets with a large (intrinsic) dimension, where any practical amount of training data cannot cover all the blind-spots. This explains the limited success for applying adversarial training on ImageNet dataset, where many test images can be sufficiently far away from the empirical distribution of training dataset.

4.2 Related Works

4.2.1 Defending Against Adversarial Examples

Adversarial examples in DNNs have brought great threats to the deep learning-based AI applications such as autonomous driving and face recognition. Therefore, defending against adversarial examples is an urgent task before we can safely deploy deep learning models to a wider range of applications. Following the emergence of adversarial examples, various defense methods have been proposed, such as defensive distillation by [112] and feature squeezing by [173]. Some of these defense methods have been proven vulnerable or ineffective under strong attack methods such as C&W in [20]. Another category of recent defense methods is based on gradient masking or obfuscated gradient [17, 93, 57, 142, 126], but these methods are also successfully evaded by the stronger BPDA attack [2]. Randomization in DNNs [38, 168, 88] is also used to reduce the success rate of adversarial attacks, however, it usually incurs additional computational costs and still cannot fully defend against an adaptive attacker [2, 3].

An effective defense method is adversarial training, which trains the model with adversarial examples freshly generated during the entire training process. First introduced by [53], adversarial training demonstrates the state-of-the-art defending performance. [95] formulated the adversarial training procedure into a min-max robust optimization problem and has achieved state-of-the-art defending performance on MNIST and CIFAR datasets. Several attacks have been proposed to attack the model release by [95]. On the MNIST testset, so far the best attack by Ref. [184] can only reduce the test accuracy from 98% to 88%. Analysis by [2] shows that this adversarial training framework does not rely on obfuscated gradient and truly increases model robustness; gradient based attacks with random starts can only achieve less than 10% success rate with given distortion constraints and are unable to penetrate this defense. On the other hand, attacking adversarial training using generative models have also been investigated; both [164] and [143] propose to use GANs to produce adversarial examples in black-box and white-box settings, respectively.

Finally, a few certified defense methods [120, 141, 162] were proposed, which are able to provably increase model robustness. Besides adversarial training, in this chapter we also consider several certified defenses which can achieve relatively good performance (i.e., test accuracy on natural images does not drop significantly and training is computationally feasible), and can be applied to medium-sized networks with multiple layers. Notably, [141] analyzes adversarial training using distributional robust optimization techniques. [162] and Refs. [163] proposed a robustness certificate based on the dual of a convex relaxation for ReLU networks, and used it for training to provably increase robustness. During training, certified defense methods can provably guarantee that the model is robust on training examples; however, on unseen test examples a non-vacuous robustness generalization guarantee is hard to obtain.

4.2.2 Analyzing Adversarial Examples

Along with the attack-defense arms race, some insightful findings have been discovered to understand the natural of adversarial examples, both theoretically and experimentally. Ref. [129] shows that even for a simple data distribution of two class-conditional Gaussians, robust generalization requires significantly larger number of samples than standard generalization. Ref. [36] extends the well-known PAC learning theory to the case with adversaries, and derives the adversarial VC-dimension which can be either larger or smaller than the standard VC-dimension. Ref. [16] conjectures that a robust classifier can be computationally intractable to find, and gives a proof for the computation hardness under statistical query (SQ) model. Recently, [15] proves a computational hardness result under a standard cryptographic assumption. Additionally, finding the safe area approximately is computationally hard according to [76] and [161]. Ref. [96] explains the prevalence of adversarial examples by making a connection to the "concentration of measure" phenomenon in metric measure spaces. Ref. [144] conducted large scale experiments on ImageNet and finds a negative correlation between robustness and accuracy. Ref. [71] discovered that data examples consist of robust and non-robust features, and adversarial training tends to find robust features that have strong correlations with the labels.

Both adversarial training and certified defenses significantly improve robustness on training data, but it is still unknown if the trained model has good *robust generalization* property. Typically, we evaluate the robustness of a model by computing an upper bound of error on the test set; specifically, given a norm bounded distortion ϵ, we verify if each image in test set has a robustness certificate [182, 40, 139]. There might exist test images that are still within the capability of standard generalization (i.e., correctly classified by DNNs with high confidence, and well recognized by humans), but behaves badly in robust generalization (i.e., adversarial examples can be easily found with small distortions). This chapter complements those existing findings by showing the strong correlation between the effectiveness of adversarial defenses (both adversarial training and some certified defenses) and the distance between training data and test points. Additionally, we show that a tiny shift in input distribution (which may or may not be detectable in embedding space) can easily destroy the robustness property of an robust model.

4.3 Methodology

4.3.1 Measuring the distance between training dataset and a test data point

To verify the correlation between the effectiveness of adversarial training and how close a test point is to the manifold of training dataset, we need to propose a reasonable distance metric between a test example and a set of training examples. However, defining a meaningful distance metric for high dimensional image data is a challenging problem. Naively using an Euclidean distance metric in the input space of images works poorly as it does not reflect the true distance between the images on their ground-truth manifold. One strategy is to use (kernel-)PCA, t-SNE [94], or UMAP [101] to reduce the dimension of training data to a low dimensional space, and then define distance in that space. These methods are sufficient for small and simple datasets like MNIST, but for more general and complicated dataset like CIFAR, extracting a meaningful

low-dimensional manifold directly on the input space can be really challenging.

On the other hand, using a DNN to extract features of input images and measuring the distance in the deep feature embedding space has demonstrated better performance in many applications [66, 67], since DNN models can capture the manifold of image data much better than simple methods such as PCA or t-SNE. Although we can form an empirical distribution using kernel density estimation (KDE) on the deep feature embedding space and then obtain probability densities for test points, our experience showed that KDE work poorly in this case because the features extracted by DNNs are still high dimensional (hundreds or thousands dimensions).

Taking the above considerations into account, we propose a simple and intuitive distance metric using deep feature embeddings and k-nearest neighbour. Given a feature extraction neural network $h(x)$, a set of n training data points $\mathcal{X}_{\text{train}} = \{x^1_{\text{train}}, x^2_{\text{train}}, \cdots, x^n_{\text{train}}\}$, and a set of m test data points $\mathcal{X}_{\text{test}} = \{x^1_{\text{test}}, x^2_{\text{test}}, \cdots, x^m_{\text{test}}\}$ from the true data distribution, for each $j \in [m]$, we define the following distance between x^j_{test} and $\mathcal{X}_{\text{train}}$:

$$D(x^j_{\text{test}}, \mathcal{X}_{\text{train}}) := \frac{1}{k} \sum_{i=1}^{k} \|h(x^j_{\text{test}}) - h(x^{\pi_j(i)}_{\text{train}})\|_p \tag{4.2}$$

where $\pi_j : [n] \to [n]$ is a permutation that $\{\pi_j(1), \pi_j(2), \cdots, \pi_j(n)\}$ is an ascending ordering of training data based on the ℓ_p distance between x^j_{test} and x^i_{train} in the deep embedding space, i.e.,

$$\forall i < i', \|h(x^j_{\text{test}}) - h(x^{\pi_j(i)}_{\text{train}})\|_p \leq \|h(x^j_{\text{test}}) - h(x^{\pi_j(i')}_{\text{train}})\|_p$$

In other words, we average the embedding space distance of k nearest neighbors of x_j in the training dataset. This simple metric is non-parametric and we found that the results are not sensitive to the selection of k; also, for naturally trained and adversarially trained feature extractors, the distance metrics obtained by different feature extractors reveal very similar correlations with the effectiveness of adversarial training.

4.3.2 Measuring the distance between training and test datasets

We are also interested to investigate the "distance" between the training dataset and the test dataset to gain some insights on how adversarial training performs on the entire test set. Unlike the setting in Section 4.3.1, this requires to compute a divergence between two empirical data distributions.

Given n training data points $\mathcal{X}_{\text{train}} = \{x^1_{\text{train}}, x^2_{\text{train}}, \cdots, x^n_{\text{train}}\}$ and m test data points $\mathcal{X}_{\text{test}} = \{x^1_{\text{test}}, x^2_{\text{test}}, \cdots, x^m_{\text{test}}\}$, we first apply a neural feature extractor h to them, which is the same as in Section 4.3.1. Then, we apply a non-linear projection (in our case, we use t-SNE) to project both $h(x^i_{\text{train}})$ and $h(x^j_{\text{test}})$ to a low dimensional space, and obtain $\bar{x}^i_{\text{train}} = \text{proj}(h(x^i_{\text{train}}))$ and $\bar{x}^j_{\text{test}} = \text{proj}(h(x^j_{\text{test}}))$. The dataset after feature extraction and projection is denoted as $\bar{\mathcal{X}}_{\text{train}}$ and $\bar{\mathcal{X}}_{\text{test}}$. Because $\bar{x}^i_{\text{train}}$ and $\bar{x}^j_{\text{test}}$ are low dimensional, we can use kernel density estimation (KDE) to form empirical distributions $\bar{p}_{\text{train}}$ and $\bar{p}_{\text{test}}$ for them.

We use p_{train} and p_{test} to denote the true distributions. Then, we approximate the KL divergence between p_{train} and p_{test} via a numerical integration of Eq.(4.3):

$$D_{\text{KL}}(p_{\text{train}}||p_{\text{test}}) \approx \int_V \bar{p}_{\text{train}}(x) \log \frac{\bar{p}_{\text{train}}(x)}{\bar{p}_{\text{test}}(x)} \mathrm{d}x, \tag{4.3}$$

where $\bar{p}_{\text{train}}(x) = \frac{1}{n}\sum_{i=1}^n K(x - \bar{x}^i_{\text{train}}; H)$ and $\bar{p}_{\text{test}}(x) = \frac{1}{m}\sum_{j=1}^m K(x - \bar{x}^j_{\text{test}}; H)$ are the KDE density functions. K is the kernel function (specifically, we use the Gaussian kernel) and H is the bandwidth parameter automatically selected by Scott's rule [132]. V is chosen as a box bounding all training and test data points.

For a multi-class dataset, we compute the aforementioned KDE and KL divergence for each class separately. We should emphasize that this method only gives us a rough characterization which might help us understand the limitations of adversarial training. The true divergence between general training and test distributions in high dimensional space is not accessible in our setting.

4.3.3 The Blind-Spot Attack: a new class of adversarial attacks

Inspired by our findings of the negative correlation between the effectiveness of adversarial training and the distance between a test image and training dataset, we identify a new class of adversarial attacks called "blind-spot attacks", where we find input images that are "far enough" from any existing training examples such that:

- They are still drawn from the ground-truth data distribution (i.e. well recognized by humans) and *classified correctly* by the model (within the generalization capability of the model);
- Adversarial training cannot provide good robustness properties on these images, and we can easily find their adversarial examples with small distortions using a simple gradient based attack.

Importantly, blind-spot images are *not* adversarial images themselves. However, after performing adversarial attacks, we can find their adversarial examples with small distortions, despite adversarial training. In other words, we exploit the weakness in a model's robust generalization capability.

We find that these blind-spots are prevalent and can be easily found without resorting to complex generative models like in [143]. For the MNIST dataset which [95], [162] and [141] demonstrate the strongest defense results so far, we propose a simple transformation to find the blind-spots in these models. We simply scale and shift each pixel value. Suppose the input image $x \in [-0.5, 0.5]^d$, we scale and shift each test data example x element-wise to form a new example x':

$$x' = \alpha x + \beta, \text{ s.t. } x' \in [-0.5, 0.5]^d$$

where α is a constant close to 1 and β is a constant close to 0. We make sure that the selection of α and β will result in a x' that is still in the valid input range $[-0.5, 0.5]^d$. This transformation effectively adjusts the contrast of the image, and/or adds a gray background to the image. We then perform Carlini & Wagner's attacks on these transformed images x' to find their adversarial examples x'_{adv}. It is important that

the blind-spot images x' are still undoubtedly valid images; for example, a digit that is slightly darker than the one in test set is still considered as a valid digit and can be well recognized by humans. Also, we found that with appropriate α and β the accuracy of MNIST and Fashion-MNIST models barely decreases; the model has enough generalization capability for this set of slightly transformed images, yet their adversarial examples can be easily found. Although the blind-spot attack is beyond the threat model considered in adversarial training (e.g. ℓ_∞ norm constrained perturbations), our argument is that adversarial training (and some other defense methods with certifications only on training examples such as [162]) are unlikely to scale well to datasets that lie in a high dimensional manifold, as the limited training data only guarantees robustness near these training examples. The blind-spots are almost inevitable in high dimensional case. For example, in CIFAR-10, about 50% of test images are already in blind-spots and their adversarial examples with small distortions can be trivially found despite adversarial training. Using data augmentation may eliminate some blind-spots, however for high dimensional data it is impossible to enumerate all possible inputs due to the curse of dimensionality.

4.4 Experiments

In this section we present our experimental results on adversarially trained models by [95]. Results on certified defense models by [162, 163] and [141] are very similar and are demonstrated in Section C.4 in the Appendix.

4.4.1 Setup

We conduct experiments on adversarially trained models by [95] on four datasets: MNIST, Fashion MNIST, and CIFAR-10. For MNIST, we use the "secret" model release for the MNIST attack challenge.[3] For CIFAR-10, we use the public "adversarially trained" model.[4] For Fashion MNIST, we train our own model with the

[3] https://github.com/MadryLab/mnist_challenge [4] https://github.com/MadryLab/cifar10_challenge

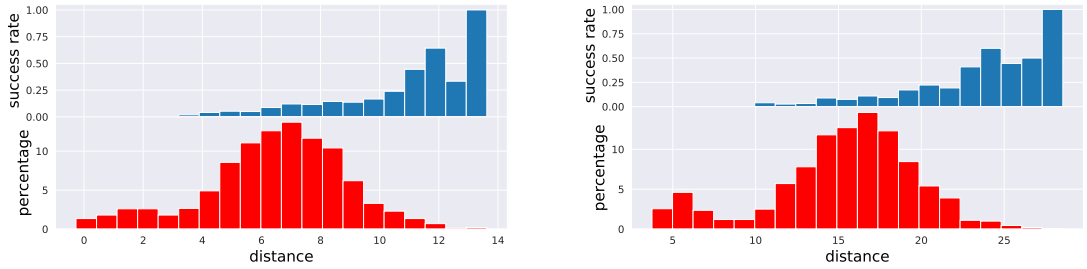
same model structure and parameters as the robust MNIST model, except that the iterative adversary is allowed to perturb each pixel by at most $\epsilon = 0.1$ as a larger ϵ will significantly reduce model accuracy.

We use our presented simple blind-spot attack in Section 4.3.3 to find blind-spot images, and use Carlini & Wagner's (C&W's) ℓ_∞ attack [20] to find their adversarial examples. We found that C&W's attacks generally find adversarial examples with smaller perturbations than projected gradient descent (PGD). To avoid gradient masking, we initial our attacks using two schemes: (1) from the original image plus a random Gaussian noise with a standard deviation of 0.2; (2) from a blank gray image where all pixels are initialized as 0. A successful attack is defined as finding an perturbed example that changes the model's classification and the ℓ_∞ distortion is less than a given ϵ used for robust training. For MNIST, $\epsilon = 0.3$; for Fashion-MNIST, $\epsilon = 0.1$; and for CIFAR, $\epsilon = 8/255$. All input images are normalized to $[-0.5, 0.5]$.

4.4.2 Effectiveness of adversarial training and the distance to training set

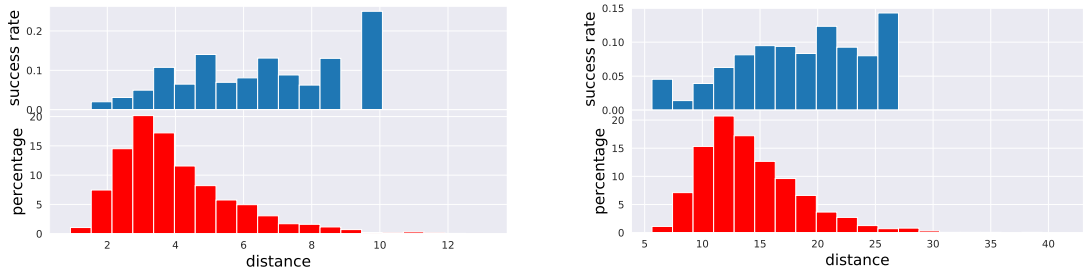
In this set of experiments, we build a connection between attack success rate on adversarially trained models and the distance between a test example and the whole training set. We use the metric defined in Section 4.3.1 to measure this distance. For MNIST and Fashion-MNIST, the outputs of the first fully connected layer (after all convolutional layers) are used as the neural feature extractor $h(x)$; for CIFAR, we use the outputs of the last average pooling layer. We consider both naturally and adversarially trained networks as the neural feature extractor, with $p = 2$ and $k = 5$. The results are shown in Figures 4-1, 4-2 and 4-3. For each test set, after obtaining the distance of each test point, we bin the test data points based on their distances to the training set and show them in the histogram at the bottom half of each figure (red). The top half of each figure (blue) represents the attack success rates for the test images in the corresponding bins. Some bars on the right are missing because there are too few points in the corresponding bins. We only attack correctly classified

images and only calculate success rate on those images. Note that we should not compare the distances shown between the left and right columns of Figures 4-1, 4-2 and 4-3 because they are obtained using different embeddings, however the overall trends are very similar.



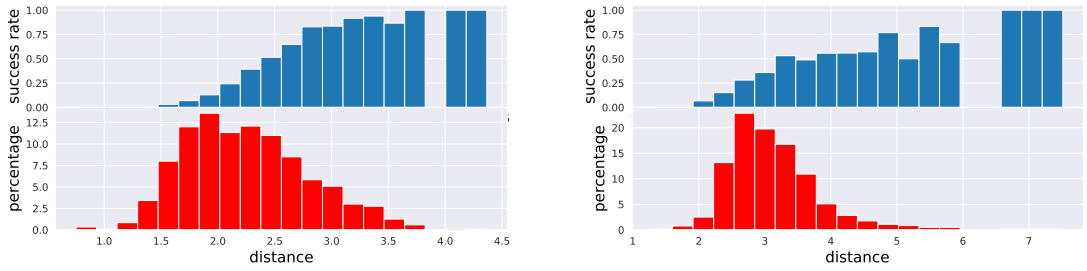
(a) Using adversarially trained model as $h(x)$ **(b)** Using naturally trained model as $h(x)$

Figure 4-1: Attack success rate and distance distribution of MNIST model in [95]. Upper: C&W ℓ_∞ attack success rates, $\epsilon = 0.3$. Lower: The distribution of the average ℓ_2 (embedding space) distance between the images in test set and the top-5 nearest images in training set.



(a) Using adversarially trained model as $h(x)$ **(b)** Using naturally trained model as $h(x)$

Figure 4-2: Attack success rate and distance distribution of Fashion MNIST model trained using [95]. Upper: C&W ℓ_∞ attack success rate, $\epsilon = 0.1$. Lower: The distribution of the average ℓ_2 (embedding space) distance between the images in test set and the top-5 nearest images in training set.



(a) Using adversarially trained model as $h(x)$ **(b)** Using naturally trained model as $h(x)$

Figure 4-3: Attack success rate and distance distribution of CIFAR model in [95]. Upper: C&W ℓ_∞ attack success rate, $\epsilon = 8/255$. Lower: The distribution of the average ℓ_2 (embedding space) distance between the images in test set and the top-5 nearest images in training set.

As we can observe in all three figures, most successful attacks in test sets for adversarially trained networks concentrate on the right hand side of the distance distribution, and the success rates tend to grow when the distance is increasing. The

trend is independent of the feature extractor being used (naturally or adversarially trained). The strong correlation between attack success rates and the distance from a test point to the training dataset supports our hypothesis that adversarial training tends to fail on test points that are far enough from the training data distribution.

4.4.3 KL Divergence between training and test sets vs attack success rate

To quantify the overall distance between the training and the test set, we approximately calculate the KL divergence between the KDE distributions of training set and test set for each class according to Eq. (4.3). Then, for each dataset we take the average KL divergence across all classes, as shown in Table 4.1. We use both adversarially trained and naturally trained networks as feature extractors $h(x)$. Additionally, we also calculate the average normalized distance by calculating the ℓ_2 distance between each test point and the training set as in Section 4.4.2, and taking the average over all test points. To compare between different datasets, we normalize each element of the feature representation $h(x)$ to mean 0 and variance 1. We average this distance among all test points and divide it by $\sqrt{d_t}$ to normalize the dimension, where d_t is the dimension of the feature representation $h(\cdot)$.

Clearly, Fashion-MNIST is the dataset with the strongest defense as measured by the attack success rates on test set, and its KL divergence is also the smallest. For CIFAR, the divergence between training and test sets is significantly larger, and adversarial training only has limited success. The hardness of training a robust model for MNIST is in between Fashion-MNIST and CIFAR. Another important observation is that the effectiveness of adversarial training *does not depend on the accuracy*; for Fashion-MNIST, classification is harder as the data is more complicated than MNIST, but training a robust Fashion-MNIST model is easier as the data distribution is more concentrated and adversarial training has less "blind-spots".

Dataset	Avg KL div. (adv. trained)	Avg KL div. (nat. trained)	Avg. normalized ℓ_2 Distance	Attack Success Rates (Test Set)	Test Accuracy
Fashion-MNIST	0.046	0.058	0.4233	6.4%	86.1%
MNIST	0.119	0.095	0.3993	9.7%	98.2%
CIFAR	0.571	0.143	0.6715	37.9%	87.0%

Table 4.1: Average KL divergence and normalized ℓ_2 distance between training and test sets across all classes. We use both adversarially trained networks (adv.) and naturally trained networks (nat.) as our feature extractors when computing KL divergence. Note that we only attack images that are correctly classified and report success rate on those images.

4.4.4 Blind-Spot Attack on MNIST and Fashion MNIST

In this section we focus on applying the proposed blind-spot attack to MNIST and Fashion MNIST. As mentioned in Section 4.3.3, for an image x from the test set, the blind-spot image $x' = \alpha x + \beta$ obtained by scaling and shifting is considered as a new natural image, and we use the C&W ℓ_∞ attack to craft an adversarial image x'_{adv} for x'. The attack distortion is calculated as the ℓ_∞ distance between x' and x'_{adv}. For MNIST, $\epsilon = 0.3$ so we set the scaling factor to $\alpha = \{1.0, 0.9, 0.8, 0.7\}$. For Fashion-MNIST, $\epsilon = 0.1$ so we set the scaling factor to $\alpha = \{1.0, 0.95, 0.9\}$. We set β to either 0 or a small constant. The case $\alpha = 1.0, \beta = 0.0$ represents the original test set images. We report the model's accuracy and attack success rates for each choice of α and β in Table 4.2 and Table 4.3. Because we scale the image by a factor of α, we also set a stricter criterion of success – the ℓ_∞ perturbation must be less than $\alpha\epsilon$ to be counted as a successful attack. For MNIST, $\epsilon = 0.3$ and for Fashion-MNIST, $\epsilon = 0.1$. We report both success criterion, ϵ and $\alpha\epsilon$ in Tables 4.2 and 4.3.

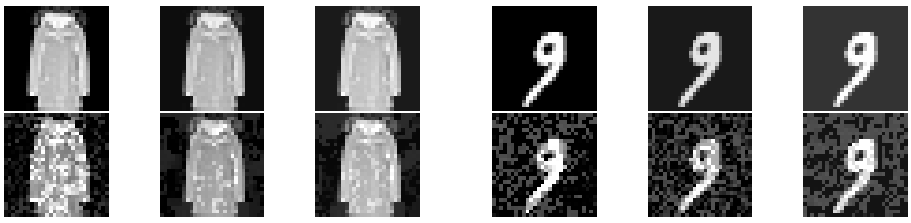
We first observe that for all pairs of α and β the transformation does not affect the models' test accuracy at all. The adversarially trained model classifies these slightly scaled and shifted images very well, with test accuracy equivalent to the original test set. Visual comparisons in Figure 4-4 show that when α is close to 1 and β is close

α, β	$\alpha = 1.0$	$\alpha = 0.9$				$\alpha = 0.8$				$\alpha = 0.7$			
	$\beta = 0$	$\beta = 0$		$\beta = 0.05$		$\beta = 0$		$\beta = 0.1$		$\beta = 0$		$\beta = 0.15$	
acc	98.2%	98.3%		98.5%		98.4%		98.5%		98.4%		98.1%	
th.	0.3	0.3	0.27	0.3	0.27	0.3	0.24	0.3	0.24	0.3	0.21	0.3	0.21
suc. rate	9.70%	75.20%	15.20%	93.65%	82.50%	94.85%	52.30%	99.55%	95.45%	98.60%	82.45%	99.95%	99.95%

Table 4.2: Attack success rate (suc. rate) and test accuracy (acc) of scaled and shifted MNIST. An attack is considered successful if its ℓ_∞ distortion is less than thresholds (th.) 0.3 or 0.3α.

α, β	$\alpha = 1.0$	$\alpha = 0.95$				$\alpha = 0.9$			
	$\beta = 0$	$\beta = 0$		$\beta = 0.025$		$\beta = 0$		$\beta = 0.05$	
acc	86.1%	86.1%		86.4%		86.1%		86.2%	
th.	0.1	0.1	0.095	0.1	0.095	0.1	0.09	0.1	0.09
suc. rate	6.40%	11.25%	9.05%	22.55%	18.55%	25.70%	19.15%	62.60%	55.95%

Table 4.3: Attack success rate (suc. rate) and test accuracy (acc) of scaled and shifted Fashion-MNIST. An attack is considered as successful if its ℓ_∞ distortion is less than threshold (th.) 0.1 or 0.1α.

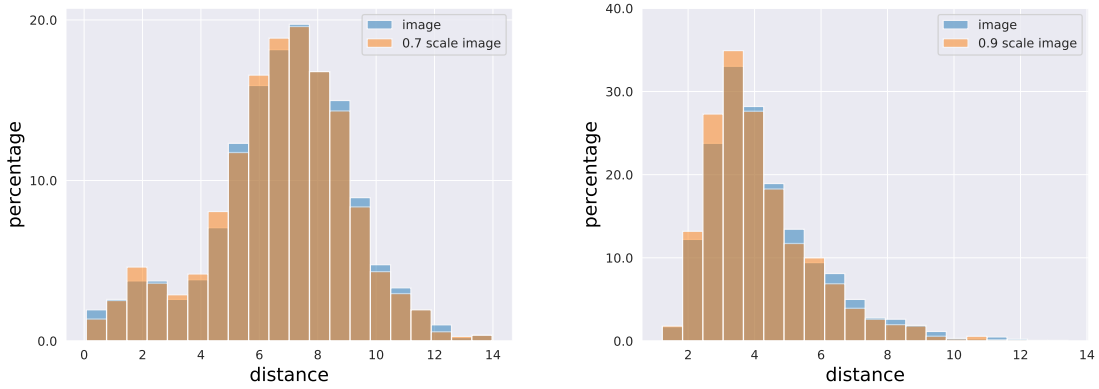


(a) $\alpha = 1.0$ $\beta = 0.0$ dist= 0.218
(b) $\alpha = 0.9$ $\beta = 0.0$ dist= 0.099
(c) $\alpha = 0.9$ $\beta = 0.05$ dist= 0.070
(d) $\alpha = 1.0$ $\beta = 0.0$ dist= 0.338
(e) $\alpha = 0.8$ $\beta = 0.0$ dist= 0.229
(f) $\alpha = 0.8$ $\beta = 0.1$ dist= 0.129

Figure 4-4: Blind-spot attacks on Fashion-MNIST and MNIST data with scaling and shifting on adversarially trained models [95]. First row contains input images after scaling and shifting and the second row contains the found adversarial examples. "dist" represents the ℓ_∞ distortion of adversarial perturbations. The first rows of figures (a) and (d) represent the original test set images ($\alpha = 1.0, \beta = 0.0$); first rows of figures (b), (c), (e), and (f) illustrate the images after transformation. Adversarial examples for these transformed images have small distortions.

to 0, it is hard to distinguish the transformed images from the original images. On the other hand, according to Tables 4.2 and 4.3, the attack success rates for those transformed test images are significantly higher than the original test images, for both the original criterion ϵ and the stricter criterion $\alpha\epsilon$. In Figure 4-4, we can see that the ℓ_∞ adversarial perturbation required is much smaller than the original image after the transformation. Thus, the proposed scale and shift transformations indeed move test images into blind-spots. More figures are in the Appendix.

One might think that we can generally detect blind-spot attacks by observing



(a) MNIST **(b)** Fashion-MNIST

Figure 4-5: The distribution of the ℓ_2 distance between the original and scaled images in test set and the top-5 nearest images ($k = 5$) in training set using the distance metric defined in Eq. (4.2).

their distances to the training dataset, using a metric similar to Eq. (4.2). Thus, we plot histograms for the distances between tests points and training dataset, for both original test images and those slightly transformed ones in Figure 4-5. We set $\alpha = 0.7, \beta = 0$ for MNIST and $\alpha = 0.9, \beta = 0$ for Fashion-MNIST. Unfortunately, the differences in distance histograms for these blind-spot images are so tiny that we cannot reliably detect the change, yet the robustness property drastically changes on these transformed images.

4.5 Discussion and Remarks

In this chapter, we observe that the effectiveness of adversarial training is highly correlated with the characteristics of the dataset, and data points that are far enough from the distribution of training data are prone to adversarial attacks despite adversarial training. Following this observation, we defined a new class of attacks called "blind-spot attack" and proposed a simple scale-and-shift scheme for conducting blind-spot attacks on adversarially trained MNIST and Fashion MNIST datasets with high success rates. Our findings suggest that adversarial training can be challenging due to the prevalence of blind-spots in high dimensional datasets.

Chapter 5

Robust Deep Reinforcement Learning against Adversarial Perturbations on Observations

5.1 Introduction

With deep neural networks (DNNs) as powerful function approximators, deep reinforcement learning (DRL) has achieved great success on many complex tasks [106, 85, 83, 138, 55] and even on some safety-critical applications (e.g., autonomous driving [157, 124, 109]). Despite achieving super-human level performance on many tasks, the existence of adversarial examples [146] in DNNs and many successful attacks to DRL [68, 6, 86, 113, 165] motivates us to study robust DRL algorithms.

When an RL agent obtains its current state via observations, the observations may contain uncertainty that naturally originates from unavoidable sensor errors or equipment inaccuracy. A policy not robust to such uncertainty can lead to catastrophic failures (Figure 5-1). To ensure safety under the *worst case* uncertainty, in this chapter we consider the adversarial setting where the observation is adversarially perturbed from s to $\nu(s)$, yet the underlying true environment state s where the agent locates is unchanged. This setting is aligned with many adversarial attacks

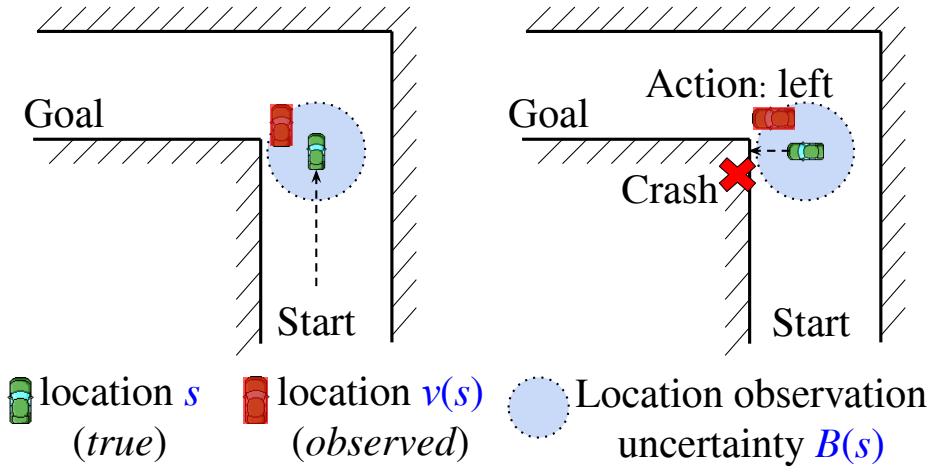


Figure 5-1: A car observes its location through sensors (e.g., GPS) and plans its route to the goal. Without considering the uncertainty in observed location (e.g., error of GPS coordinates), an unsafe policy may crash into the wall because the observed location and true location differ.

on state observations (e.g., [68, 86]). To improve robustness under this setting, a natural approach is to extend existing adversarial defenses for supervised learning, e.g., adversarial training [81, 95, 177], to DRL. Specifically, we can attack the agent and generate trajectories adversarially during training time, and apply any existing DRL algorithm to hopefully obtain a robust policy. Unfortunately, we show that for most environments, naive adversarial training can make training unstable and deteriorate agent performance (a similar observation is made in [7, 46]), or does not significantly improve robustness under strong attacks. Since RL and supervised learning are quite different problems, naively applying techniques from supervised learning to RL without a proper theoretical justification can be unsuccessful.

Additionally, DRL agents can be brittle even without any adversarial attacks – an agent may fail occasionally but catastrophically during regular (non-adversarial) rollouts, and debugging these failure cases can be quite challenging [154]. In practical applications, such small noise can be naturally prevalent and thus prohibits the use of DRL in safety critical domains like autonomous driving. As we will show in Section 5.4, the agents trained using our proposed robust policy optimization objective (SA-MDP) can obtain significantly better worst case reward with much less variance.

This chapter studies the theory and practice of robust RL against perturbations on state observations. The material presented in this chapter are based on [180]. We

summarize our contributions as follows:

- We formulate the perturbation on state observations as a modified Markov decision process (MDP), which we call state-adversarial MDP (SA-MDP), and study its fundamental properties. We show that under an optimal adversary, a stationary and Markovian optimal policy may not exist for SA-MDP.
- Based on our theory of SA-MDP, we propose a theoretically principled robust policy regularizer that is related to the total variation distance or KL-divergence on perturbed policies. It can be practically and efficiently applied to a wide range of RL algorithms, including proximal policy optimization (PPO) and deep Q networks (DQN).
- We conduct experiments on 11 environments ranging from Atari games with discrete actions to complex robotic control tasks in continuous action space. Our proposed method significantly improves robustness under strong white-box attacks on state observations, including two new attacks we design, the maximal action difference attack (MAD attack).

5.2 Related Works

Robust Reinforcement Learning – Since each element of reinforcement learning (observations, actions, transition dynamics and rewards) can contain uncertainty, robust reinforcement learning has been studied from different perspectives. Robust Markov decision process (RMDP) [72, 107] considers the worst case perturbation with respect to transition probabilities, and has been extended to distributional settings [169] and partially observed MDPs [108]. The agent observes the original true state from the environment and acts accordingly, but the environment can choose from a set of transition probabilities that minimizes rewards. Compared to our SA-MDP where the adversary changes only observations, in RMDP the ground-truth states are changed, and thus RMDP is more suitable for modeling *environment parameter changes* (e.g., changes in physical parameters like mass and length, etc.). RMDP theory has inspired robust deep Q-learning [136] and policy gradient algorithms [100, 37, 99]

that are robust against small environment changes.

Additionally, several works [116, 84] consider the adversarial setting of multi-agent reinforcement learning [147, 13]. In the simplest two-player setting (referred to as minimax games in [87]), each agent chooses an action at each step, and the environment transits based on both actions. The regular Q function $Q(s, a)$ can be extended to $Q(S, a, o)$ where o is the opponent's action and Q-learning is still convergent. This setting can be extended to deep Q learning and policy gradient algorithms [84, 116]. Authors in [116] show that learning an opponent agent simultaneously can improve the agent's performance as well as its robustness against environment turbulence and test conditions (e.g., change in mass or friction). The authors in [56] carried out real-world experiments on the two-player adversarial learning game. Additionally, [148] considered adversarial perturbations on the action space. In contrast, [50] investigated how to learn a robust reward. All of these settings are different from ours: in our state-adversarial setting, we manipulate only the observations but do not change the underlying environment (the true states) or actions directly. Our SA-MDP is more suitable to model worst case errors in the observation process (e.g., sensor errors and equipment inaccuracy).

Adversarial Attacks on State Observations in DRL – Ref. [68] evaluated the robustness of deep reinforcement learning policies through an FGSM based attack on Atari games with discrete actions. Authors in [79] proposed to use the value function to guide adversarial perturbation search. Ref. [86] considered a more complicated case where the adversary is allowed to only attack at a subset of time steps, and used a generative model to generate attack plans luring the agent to a designated target state. Ref. [6] studied black-box attacks on DQN with discrete action space via transferability of adversarial examples. Then [113] further enhanced adversarial attacks to DRL with multi-step gradient descent and better engineered loss functions, but these require a critic or Q function to perform attacks. Typically, the critic learned during agent training is used. We find that using this critic can be sub-optimal or impractical in many cases, and propose our two *critic-independent* attacks in Section 5.3.4. We refer the reader to recent surveys [165, 69] for a taxonomy and a comprehensive list of

adversarial attacks in DRL.

Improving Robustness for State Observations in DRL – For discrete action RL tasks, [79] first presented preliminary results of adversarial training on Pong (one of the simplest Atari environments) using weak FGSM attacks on pixel space. Ref. [7] applied adversarial training to several Atari games with DQN, and found it challenging for the agent to adapt to the attacks during training time – one must attack only a portion of frames, and even then the agent performance still suffers under test time attacks. For Pong, adversarial training can improve reward under attack from -21 (lowest) to -5, yet is still far away from the optimal reward ($+21$). For continuous action RL tasks (e.g., MuJoCo environments), [97] used a weak FGSM based attack with policy gradient to adversarially train a few simple RL tasks. The authors in [113] used stronger multi-step gradient based attacks; however, their evaluation focused on robustness against environment changes rather than state perturbations. Unlike our work which first develops principles and then applies those to different DRL algorithms, these works mostly extend adversarial training in supervised learning to the DRL setting. We show that *adversarial training does not reliably improve test time performance* under strong attacks in Section 5.4.

To obtain better performance than adversarial training, [103, 46] treat the *discrete action* outputs of DQN as labels, and apply existing certified defense methods for classification [104] to robustly predict actions using imitation learning. This approach outperforms [7], but it is unclear how to apply it to environments with continuous action spaces. Compared to their approach, our SA-DQN does not use imitation learning and achieves better performance on most environments. Other related works include [59], which proposed a meta online learning procedure with a master agent detecting the presence of the adversary and switching between a few sub-policies, but did not discuss how to train a single agent robustly. Ref. [32] applied adversarial training specifically for RL-based path-finding algorithms. Ref. [92] considered the worst-case scenario during rollouts for an existing DQN agent to ensure safety, but it relies on an existing policy and does not include a training procedure. Robust DRL for perturbations on state observations, especially for continuous action space

tasks, still has many open challenges, and existing approaches lack proper theoretical justifications.

5.3 Methodology

5.3.1 State-Adversarial Markov Decision Process (SA-MDP)

Notation – A Markov decision process (MDP) is defined as a 5-tuple, $(\mathcal{S}, \mathcal{A}, R, p, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $p : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ represents the transition probability of environment, where $\mathcal{P}(\mathcal{S})$ defines the set of all possible probability measures on $\mathcal{S}$. The transition probability is $p(s'|s, a) = \Pr(s_{t+1} = s'|s_t = s, a_t = a)$, where t is the time step. We denote a stationary policy as $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, the set of all stochastic and Markovian policies as Π_{MR}, the set of all deterministic and Markovian policies as Π_{MD}, and the discount factor as $0 < \gamma < 1$.

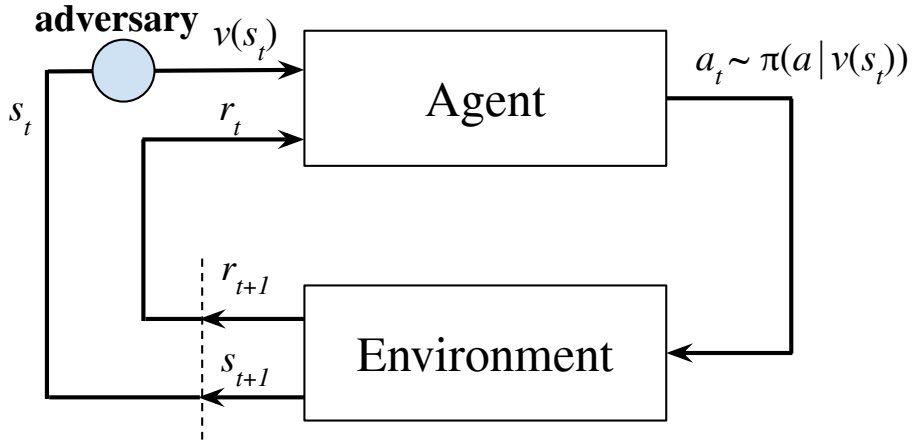


Figure 5-2: Reinforcement learning with perturbed state observations. The agent observes a perturbed state $\nu(s_t)$ rather than the true environment state s_t.

In state-adversarial MDP (SA-MDP), we introduce an adversary $\nu(s) : \mathcal{S} \to \mathcal{S}$. The adversary perturbs only the state observations of the agent, such that the action is taken as $\pi(a|\nu(s))$; the environment still transits from the true state s rather than $\nu(s)$ to the next state. Since $\nu(s)$ can be different from s, the agent's action from $\pi(a|\nu(s))$ may be sub-optimal, and thus the adversary is able to reduce the reward.

In real world RL problems, the adversary can be reflected as the worst case noise in measurement or state estimation uncertainty. Note that this scenario is different from the two-player Markov game [87] where both players see unperturbed true environment states and interact with the environment directly; the opponent's action can change the true state of the game.

To allow a formal analysis, we first make the assumption for the adversary ν:

Assumption 1 (Stationary, Deterministic and Markovian Adversary)**.** *$\nu(s)$ is a deterministic function $\nu : \mathcal{S} \to \mathcal{S}$ which only depends on the current state s, and ν does not change over time.*

This assumption holds for many adversarial attacks [68, 86, 79, 113]. These attacks only depend on the current state input and the policy or Q network so they are Markovian; the network parameters are frozen at test time, so given the same s the adversary will generate the same (stationary) perturbation. We leave the formal analysis of non-Markovian, non-stationary adversaries as future work.

If the adversary can perturb a state s arbitrarily without bounds, the problem can become trivial. To fit our analysis to the most realistic settings, we need to restrict the power of an adversary. We define perturbation set $B(s)$, to restrict the adversary to perturb a state s only to a predefined set of states:

Definition 2 (Adversary Perturbation Set)**.** *We define a set $B(s)$ which contains all allowed perturbations of the adversary. Formally, $\nu(s) \in B(s)$ where $B(s)$ is a set of states and $s \in B(s)$.*

$B(s)$ is usually a set of task-specific "neighboring" states of s (e.g., bounded sensor measurement errors), which makes the observation still meaningful (yet not accurate) even with perturbations. After defining B, an SA-MDP can be represented as a 6-tuple $(\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$.

Analysis of SA-MDP – We first derive Bellman Equations and a basic policy evaluation procedure, then we discuss the possibility of obtaining an optimal policy for SA-MDP. The adversarial value and action-value functions under ν in an SA-MDP

are similar to those of a regular MDP:

$$\tilde{V}_{\pi\circ\nu}(s) = \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right], \quad \tilde{Q}_{\pi\circ\nu}(s,a) = \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right],$$

where the reward at step-t is defined as r_t and $\pi\circ\nu$ denotes the policy under observation perturbations: $\pi(a|\nu(s))$. Based on these two definitions, we first consider the simplest case with *fixed* π and ν:

Theorem 3 (Bellman equations for fixed π and ν)**.** *Given $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and $\nu : \mathcal{S} \to \mathcal{S}$, we have*

$$\tilde{V}_{\pi\circ\nu}(s) = \sum_{a\in\mathcal{A}}\pi(a|\nu(s))\sum_{s'\in\mathcal{S}}p(s'|s,a)\left[R(s,a,s') + \gamma\tilde{V}_{\pi\circ\nu}(s')\right]$$
$$\tilde{Q}_{\pi\circ\nu}(s,a) = \sum_{s'\in\mathcal{S}}p(s'|s,a)\left[R(s,a,s') + \gamma\sum_{a'\in\mathcal{A}}\pi(a'|\nu(s'))\tilde{Q}_{\pi\circ\nu}(s',a')\right].$$

The proof of Theorem 3 is simple, as when π, ν are fixed, they can be "merged" as a single policy, and existing results from MDP can be directly applied. Now we consider a more complicated case, where we want to find the value functions under *optimal adversary* $\nu^*(\pi)$, minimizing the total expected reward for a *fixed* π. The optimal adversarial value and action-value functions are defined as:

$$\tilde{V}_{\pi\circ\nu^*}(s) = \min_{\nu}\tilde{V}_{\pi\circ\nu}(s), \quad \tilde{Q}_{\pi\circ\nu^*}(s,a) = \min_{\nu}\tilde{Q}_{\pi\circ\nu}(s,a).$$

Theorem 4 (Bellman contraction for optimal adversary)**.** *Define Bellman operator $\mathscr{L} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$,*

$$(\mathscr{L}\tilde{V})(s) = \min_{s_\nu\in B(s)}\sum_{a\in\mathcal{A}}\pi(a|s_\nu)\sum_{s'\in\mathcal{S}}p(s'|s,a)\left[R(s,a,s') + \gamma\tilde{V}(s')\right]. \tag{5.1}$$

The Bellman equation for optimal adversary ν^* can then be written as: $\tilde{V}_{\pi\circ\nu^*} = \mathscr{L}\tilde{V}_{\pi\circ\nu^*}$. Additionally, $\mathscr{L}$ is a contraction that converges to $\tilde{V}_{\pi\circ\nu^*}$.

Theorem 4 says that given a *fixed* policy π, we can evaluate its performance (value

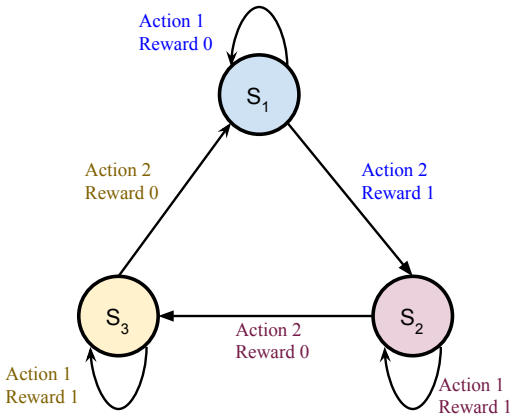


Figure 5-3: A toy environment for SA-MDP.

functions) under the optimal (strongest) adversary, through a Bellman contraction. It is functionally similar to the "policy evaluation" procedure in regular MDP. The proof of Theorem 4 is in the same spirit as the proof of Bellman optimality equations for solving the optimal policy for an MDP; the important difference here is that we solve the optimal adversary, for a *fixed* policy π. Given π, value functions for MDP and SA-MDP can be vastly different. Here we show a 3-state toy environment in Figure 5-3; an optimal MDP policy is to take action 2 in S_1, action 1 in S_2 and S_3. Under the presence of an adversary $\nu(S_1) = S_2$, $\nu(S_2) = S_1$, $\nu(S_3) = S_1$, this policy receives zero total reward as the adversary can make the action $\pi(a|\nu(s))$ totally wrong regardless of the states. On the other hand, a policy taking random actions on all three states (which is a non-optimal policy for MDP) is unaffected by the adversary and obtains non-zero rewards in SA-MDP. Details are given in Appendix D.1.

Finally, we discuss our ultimate quest of finding an *optimal* policy π^* under the strongest adversary $\nu^*(\pi)$ in the SA-MDP setting (we use the notation $\nu^*(\pi)$ to explicit indicate that ν^* is the optimal adversary for a given π). An optimal policy should be the best among all policies on every state:

$$\tilde{V}_{\pi^* \circ \nu^*(\pi^*)}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s) \quad \text{for } \forall s \in \mathcal{S} \text{ and } \forall \pi, \tag{5.2}$$

where both π and ν are not fixed. The first question is, what policy classes we need to consider for π^*. In MDPs, deterministic policies are sufficient. We show that this does not hold anymore in SA-MDP:

Theorem 5. *There exists an SA-MDP and some stochastic policy $\pi \in \Pi_{MR}$ such that we cannot find a better deterministic policy $\pi' \in \Pi_{MD}$ satisfying $\tilde{V}_{\pi' \circ \nu^*(\pi')}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s)$ for all $s \in \mathcal{S}$.*

The proof is done by constructing a counterexample where some stochastic policies are better than *any* other deterministic policies in SA-MDP (see Appendix D.1). Contrarily, in MDP, for any stochastic policy we can find a deterministic policy that is at least as good as the stochastic one. Unfortunately, even looking for both deterministic and stochastic policies still cannot always find an optimal one:

Theorem 6. *Under the optimal ν^*, an optimal policy $\pi^* \in \Pi_{MR}$ does not always exist for SA-MDP.*

The proof follows the same counterexample as in Theorem 5. The optimal policy π^* requires to have $\tilde{V}_{\pi^* \circ \nu^*(\pi^*)}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s)$ for all s and any π. In an SA-MDP, sometimes we have to make a trade-off between the value of states and no policy can maximize the values of all states.

Despite the difficulty of finding an optimal policy under the optimal adversary, we show that under certain assumptions, the loss in performance due to an optimal adversary can be bounded:

Theorem 7. *Given a policy π for a non-adversarial MDP and its value function is $V_\pi(s)$. Under the optimal adversary ν in SA-MDP, for all $s \in \mathcal{S}$ we have*

$$\max_{s \in \mathcal{S}} \left\{ V_\pi(s) - \tilde{V}_{\pi \circ \nu^*(\pi)}(s) \right\} \leq \alpha \max_{s \in \mathcal{S}} \max_{\hat{s} \in B(s)} \mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s})) \tag{5.3}$$

where $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$ is the total variation distance between $\pi(\cdot|s)$ and $\pi(\cdot|\hat{s})$, and $\alpha := 2[1 + \frac{\gamma}{(1-\gamma)^2}] \max_{(s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}} |R(s,a,s')|$ is a constant that does not depend on π.

Theorem 7 says that as long as differences between the action distributions under state perturbations (the term $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$) are not too large, the performance gap between $\tilde{V}_{\pi \circ \nu^*}(s)$ (state value of SA-MDP) and $V_\pi(s)$ (state value of regular

MDP) can be bounded. An important consequence is the motivation of regularizing $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$ during training to obtain a policy robust to strong adversaries. The proof is based on tools developed in constrained policy optimization [1], which gives an upper bound on value functions given two policies with bounded divergence. In our case, we desire that a bounded state perturbation $\hat{s}$ produces bounded divergence between $\pi(\cdot|s)$ and $\pi(\cdot|\hat{s})$.

We now study a few practical DRL algorithms, including both deep Q-learning (DQN) for discreteactions and actor-critic based policy gradient methods (PPO) for continuous actions.

5.3.2 State-Adversarial DRL for Stochastic Policies: A Case Study on PPO

We start with the most general case where the policy $\pi(a|s)$ is stochastic (e.g., in PPO [131]). The total variation distance is not easy to compute for most distributions, so we upper bound it again by KL divergence: $\mathrm{D}_{\mathrm{TV}}(\pi(a|s), \pi(a|\hat{s})) \leq \sqrt{\frac{1}{2}\mathrm{D}_{\mathrm{KL}}(\pi(a|s)\|\pi(a|\hat{s}))}$. When Gaussian policies are used, we denote $\pi(a|s) \sim \mathcal{N}(\mu_s, \Sigma_s)$ and $\pi(a|\hat{s}) \sim \mathcal{N}(\mu_{\hat{s}}, \Sigma_{\hat{s}})$. Their KL-divergence can be given as:

$$\mathrm{D}_{\mathrm{KL}}(\pi(a|s)\|\pi(a|\hat{s})) = \frac{1}{2}\left(\log|\Sigma_{\hat{s}}\Sigma_s^{-1}| + \mathrm{tr}(\Sigma_{\hat{s}}^{-1}\Sigma_s) + (\mu_{\hat{s}} - \mu_s)^\top \Sigma_{\hat{s}}^{-1}(\mu_{\hat{s}} - \mu_s) - |\mathcal{A}|\right). \tag{5.4}$$

Regularizing KL distance (5.4) for all $\hat{s} \in B(s)$ will lead to a smaller upper bound in (D.12), which is directly related to agent performance under optimal adversary. In PPO, the mean terms μ_s, $\mu_{\hat{s}}$ are produced by neural networks: $\mu_{\theta_\mu}(s)$ and $\mu_{\theta_\mu}(\hat{s})$, and Σ is a diagonal matrix independent of state s (i.e., $\Sigma_{\hat{s}} = \Sigma_s = \Sigma$), so regularizing the above KL-divergence over all s from sampled trajectories and all $\hat{s} \in B(s)$ leads to the following robust policy regularizer for PPO, ignoring constant terms:

$$\mathcal{R}_{\mathrm{PPO}}(\theta_\mu) = \frac{1}{2}\sum_s \max_{\hat{s}\in B(s)} \left(\mu_{\theta_\mu}(\hat{s}) - \mu_{\theta_\mu}(s)\right)^\top \Sigma^{-1} \left(\mu_{\theta_\mu}(\hat{s}) - \mu_{\theta_\mu}(s)\right) := \frac{1}{2}\sum_s \max_{\hat{s}\in B(s)} \mathcal{R}_s(\hat{s}, \theta_\mu). \tag{5.5}$$

We replace $\max_{s\in\mathcal{S}}$ term in (D.12) with a more practical and optimizer-friendly summation over all states in sampled trajectory. A similar treatment was used in TRPO [83] which was also derived as a KL-based regularizer, albeit on θ_μ space rather than on state space. However, minimizing (5.5) is challenging as it is a minimax objective, and we also have $\nabla_{\hat{s}}\mathcal{R}(\hat{s},\theta_\mu)|_{\hat{s}=s} = 0$ so using gradient descent directly cannot solve the inner maximization problem to a local maximum. Instead of using the more expensive second order methods, we propose the following two first order approaches to solve (5.5). Here we focus on discussing convex relaxation based method, and we defer Stochastic Gradient Langevin Dynamics (SGLD) based solver to Section D.3.2.

Convex relaxation of non-linear units in neural networks enables an efficient analysis of the outer bounds for a neural network [162, 182, 139, 40, 161, 159, 125, 140]. Several works have used it for certified adversarial defenses [163, 104, 158, 54, 179], but here we leverage it as a generic optimization tool for solving minimax functions involving neural networks. Using this technique, we can obtain an upper bound for $\mathcal{R}_s(\hat{s},\theta_\mu)$: $\overline{\mathcal{R}}_s(\theta_\mu) \geq \mathcal{R}_s(\hat{s},\theta_\mu)$ for all $\hat{s}\in B(s)$. $\overline{\mathcal{R}}_s(\theta_\mu)$ is also a function of θ_μ, and computing $\overline{\mathcal{R}}_s(\theta_\mu)$ is only a constant factor slower than computing $\mathcal{R}_s(s,\theta_\mu)$ (for a fixed s) when an efficient relaxation [104, 54, 179] is used. We can then solve the following minimization problem:

$$\min_{\theta_\mu}\frac{1}{2}\sum_s \overline{\mathcal{R}}_s(\theta_\mu) \geq \min_{\theta_\mu}\frac{1}{2}\sum_s \max_{\hat{s}\in B(s)}\mathcal{R}_s(\hat{s},\theta_\mu) = \min_{\theta_\mu}\mathcal{R}_{\text{PPO}}(\theta_\mu).$$

Since we minimize an *upper bound* of the inner max, the original objective (5.5) is guaranteed to be minimized. Using convex relaxations can also provide certain *robustness certificates* for DRL as a bonus (e.g., we can guarantee an action has bounded changes under bounded perturbations), discussed in Appendix D.5. We use `auto_LiRPA`, a recently developed tool [171], to give $\overline{\mathcal{R}}_s(\theta_\mu)$ efficiently and automatically. Once the inner maximization problem is solved, we can add $\mathcal{R}_{\text{PPO}}$ as part of the policy optimization objective, and solve PPO using SGD as usual.

Note that although Eq (5.5) looks similar to smoothness based regularizers in (semi-

)supervised learning settings to avoid overfitting [105] and improve robustness [177], our regularizer is based on the foundations of SA-MDP. Our theory justifies the use of such a regularizer in reinforcement learning setting, while [105, 177] are developed for quite different settings not related to reinforcement learning.

5.3.3 State-Adversarial DRL for Q Learning: A Case Study on DQN

The action space for DQN is finite, and the deterministic action is determined by the max Q value: $\pi(a|s) = 1$ when $a = \text{argmax}_{a'} Q(s, a')$ and 0 otherwise. The total variation distance in this case is

$$\mathrm{D}_{TV}(\pi(\cdot|s), \pi(\cdot|\hat{s})) = \begin{cases} 0 & \text{argmax}_a \, \pi(a|s) = \text{argmax}_a \, \pi(a|\hat{s}) \\ 1 & \text{otherwise.} \end{cases}$$

Thus, we want to make the top-1 action stay unchanged after perturbation, and we can use a hinge-like robust policy regularizer, where $a^*(s) = \text{argmax}_a \, Q_\theta(s, a)$ and c is a small positive constant:

$$\mathcal{R}_{\text{DQN}}(\theta) := \sum_s \max\{\max_{\hat{s} \in B(s)} \max_{a \neq a^*} Q_\theta(\hat{s}, a) - Q_\theta(\hat{s}, a^*(s)), -c\}. \tag{5.6}$$

The sum is over all s in a sampled batch. Unlike PPO, it is more similar to the robustness of classification tasks, if we treat $a^*(s)$ as the "correct" label. The maximization can be solved using projected gradient descent (PGD) or convex relaxation of neural networks. Due to its similarity to classification, we defer the details on solving $\mathcal{R}_{\text{DQN}}(\theta)$ and full SA-DQN algorithm to Appendix D.6.

5.3.4 Robustness Evaluation via Adversarial Attacks under Assumption 1

In this section and Appendix D.4 we discuss a few strong adversarial attacks under Assumption 1.

Attacks for PPO

For policy gradient and actor-critic based RL algorithms, [113] and many follow-on works use the gradient of $Q(s,a)$ to provide the direction to update states adversarially in K steps:

$$s^{k+1} = s^k - \eta \cdot \text{proj}\left[\nabla_{s^k} Q(s^0, \pi(s^k))\right], \quad k = 0, \ldots, K-1, \text{and define } \hat{s} := s^{K-1}. \tag{5.7}$$

Here proj$[\cdot]$ is a projection to $B(s)$, η is the learning rate, and s^0 is the state under attack. It attempts to find a state $\hat{s}$ triggering an action $\pi(\hat{s})$ minimizing the action-value at state s^0. The formulation in [113] has a glitch that the gradient is evaluated as $\nabla_{s^k} Q(s^k, \pi(s^k))$ rather than $\nabla_{s^k} Q(s^0, \pi(s^k))$. We found that the corrected form (5.7) is more successful. If Q is a perfect action-value function, $\hat{s}$ leads to the worst action that minimizes the value at s^0. However, this attack has a few drawbacks:

- Attack strength strongly depends on critic quality; if Q is poorly learned, is not robust against small perturbations or has obfuscated gradients, the attack fails as no correct update direction is given.
- It relies on the Q function which is specific to the training process, but not used during roll-out.
- Not applicable to many actor-critic methods (e.g., TRPO and PPO) using a learned value function $V(s)$ instead of $Q(s,a)$. Finding $\hat{s} \in B(s)$ minimizing $V(s)$ does not correctly reflect the setting of perturbing observations, as $V(\hat{s})$ represents the value of $\hat{s}$ rather than the value of taking $\pi(\hat{s})$ at s^0.

When we evaluate the robustness of a policy, we desire it to be independent of a specific critic network to avoid these problems. We thus propose a simple yet very effective attack for PPO, which does not depend on a critic, the **Maximal Action Difference (MAD) attack**. Following our Theorem 7, we can find an adversarial state $\hat{s}$ by *maximizing* $D_{\text{KL}}\left(\pi(\cdot|s) \| \pi(\cdot|\hat{s})\right)$. For actions parameterized by Gaussian mean $\pi_{\theta_\pi}(s)$ and covariance matrix Σ (independent of s), we minimize

$L_{\text{MAD}}(\hat{s}) := -D_{\text{KL}}\left(\pi(\cdot|s)\|\pi(\cdot|\hat{s})\right)$ to find $\hat{s}$:

$$\begin{aligned}\operatorname*{argmin}_{\hat{s}\in B(s)} L_{\text{MAD}}(\hat{s}) &= \operatorname*{argmax}_{\hat{s}\in B(s)} D_{\text{KL}}(\pi(\cdot|s)\|\pi(\cdot|\hat{s})) \\ &= \operatorname*{argmax}_{\hat{s}\in B(s)} \left(\pi_{\theta_\pi}(s) - \pi_{\theta_\pi}(\hat{s})\right)^\top \Sigma^{-1} \left(\pi_{\theta_\pi}(s) - \pi_{\theta_\pi}(\hat{s})\right)\end{aligned} \tag{5.8}$$

Attack for DQN

For DQN, we use the regular untargeted Projected Gradient Decent (PGD) attack in the literature [86, 113, 165]. The untargeted PGD attack with K iterations updates the state K times as follows:

$$\begin{aligned} s^{k+1} &= s^k + \eta \cdot \text{proj}[\nabla_{s^k} \mathcal{H}(Q_\theta(s^k, \cdot), a^*)], \\ s^0 &= s, \quad k = 0, \ldots, K-1 \end{aligned} \tag{5.9}$$

where $\mathcal{H}(Q_\theta(s^k, \cdot), a^*)$ is the cross-entropy loss between the output logits of $Q_\theta(s^k, \cdot)$ and the onehot-encoded distribution of $a^* := \text{argmax}_a\, Q_\theta(s, a)$. proj$[\cdot]$ is a projection operator depending on the norm constraint of $B(s)$ and η is the learning rate. A successful untargeted PGD attack will then perturb the state to lead the Q network to output an action other than the optimal action a^* chosen at the original state s. To guarantee that the final state obtained by the attack is within an ℓ_∞ ball around s ($B_\epsilon(s) = \{\hat{s} : s - \epsilon \le \hat{s} \le s + \epsilon\}$), the projection proj$[\cdot]$ is a sign operator and η is typically set to $\eta = \frac{\epsilon}{K}$.

5.4 Experiments

In our experiments, the set of adversarial states $B(s)$ is defined as an ℓ_∞ norm ball around s with a radius ϵ: $B(s) := \{\hat{s} : \|s - \hat{s}\|_\infty \le \epsilon\}$. Here ϵ is also referred to as the perturbation budget. In some environments, the ℓ_∞ norm is applied on normalized state representations. The code for SA-PPO and SA-DQN is available at https://github.com/chenhongge/StateAdvDRL.

5.4.1 Evaluation of SA-PPO

We use the PPO implementation from [43], which conducted hyperparameter search and published the optimal hyperparameters for PPO on three Mujoco environments in OpenAI Gym [10]. We use their optimal hyperparameters for PPO, and the same set of hyperparameters for SA-PPO without further tuning. We run Walker2d and Hopper 2×10^6 steps and Humanoid 1×10^7 steps to ensure convergence. Our PPO baselines achieve similar or better performance than reported in the literature [43, 62, 58]. Detailed hyperparameters are in Appendix D.7.

SA-PPO has one additional regularization parameter, κ_{PPO}, for the regularizer $\mathcal{R}_{\text{PPO}}$, which is chosen in $\{0.01, 0.03, 0.1, 0.3, 1.0\}$. The perturbation ϵ is added into the normalized state space as ℓ_∞ noise. We include three baselines: vanilla PPO, and adversarially trained PPO [97, 113] with 50% and 100% training steps under critic attack [113]. The attack has to be conducted on $V(s)$ instead of $Q(s, a)$, as PPO does not learn a Q function during learning. We report SA-PPO objective solved using both SGLD and convex relaxation methods. We use three attacks detailed in Sec. 5.3.4 and Appendix D.4.

In Table 5.1, we observe that *adversarial training deteriorates performance* and does not reliably improve robustness in all three environments. But we can see that our SA-PPO achieves higher rewards under attacks.

Our MAD attack is very effective in most environments and achieves lower rewards than critic and random attacks; this shows the importance of evaluation using strong attacks. SA-PPO, solved either by SGLD or the convex relaxation objective, *significantly improves robustness* against strong attacks. Additionally, SA-PPO achieves natural performance (without attacks) similar to that of vanilla PPO in Walker2d and Hopper, and *significantly improves the reward in Humanoid environment.* Humanoid has high state-space dimension (376) and is usually hard to train [58], and our results suggest that a robust objective can be helpful even in a non-adversarial setting. We include the convergence experiments of PPO and SA-PPO in Appendix D.7.

Table 5.1: Average rewards ± standard deviation over 50 episodes on three baselines and SA-PPO. We report natural rewards (no attacks) and rewards under thre adversarial attacks. In each row we bold the best (lowest) attack reward over all three attacks. The gray rows are the most robust agents.

Env.	ϵ	Method	Natural Reward	Attack Reward			Best Attack
				Critic	Random	MAD	
Hopper	0.075	PPO (vanilla)	3167.6± 541.6	1799.0± 935.2	2915.2±677.7	**1505.2± 382.0**	1505.2
		PPO (adv. 50%)	174± 146	69 ±83	141± 128	**42± 46**	42
		PPO (adv. 100%)	6.1± 2.6	4.4 ±1.8	6.1± 3.2	**5.8± 2.7**	5.8
		SA-PPO (SGLD)	3523.1±329.0	3665.5± 8.2	3080.2± 745.4	**2996.6± 786.4**	2996.6
		SA-PPO (Convex)	3704.1± 2.2	3698.4± 4.4	3708.7± 23.8	**3443.1± 466.672**	3443.1
Walker2d	0.05	PPO (vanilla)	4619.5± 38.2	4589.3± 12.4	4480.0±465.3	**4469.1±715.6**	4469.1
		PPO (adv. 50%)	-11 ± 0.9	-10.6 ± 0.86	**-10.99 ± 0.95**	-10.78 ± 0.89	-10.99
		PPO (adv. 100%)	-113 ± 4.14	-111.9 ± 4.13	-111 ± 4.27	**-112 ± 4.08**	-112
		SA-PPO (SGLD)	4911.8± 188.9	5019.0± 65.2	4894.8± 139.9	**4755.7± 413.1**	4755.7
		SA-PPO (Convex)	4486.6± 60.7	4572.0± 52.3	4475.0± 48.7	**4343.4± 329.4**	4343.4
Humanoid	0.075	PPO (vanilla)	5270.6±1074.3	5494.7± 118.7	5648.3± 86.8	**1140.3± 534.8**	1140.3
		PPO (adv. 50%)	234± 28	198 ± 58	240 ± 19.4	**148 ± 73**	148
		PPO (adv. 100%)	141.4 ± 20.6	140.25 ± 16.6	142.13 ± 16	**140.23 ± 34.5**	140.23
		SA-PPO (SGLD)	6624.0± 25.5	6587.0± 23.1	6614.1± 21.4	**6586.4± 23.5**	6586.4
		SA-PPO (Convex)	6400.6± 156.8	6397.9 ±35.6	**6207.9± 783.3**	6379.5± 30.5	6207.9

5.4.2 Evaluation of SA-DQN

We implement Double DQN [156] and Prioritized Experience Replay [128] on four Atari games. We train Atari agents for $6,000,000$ frames for both vanilla DQN and SA-DQN. Detailed parameters and training procedures are in Appendix D.6. For Atari games, we normalize the pixel values to $[0, 1]$ and we add ℓ_∞ adversarial noise with norm $\epsilon = 1/255$. We include vanilla DQNs and adversarially trained DQNs with 50% of frames under attack [7] during training time as baselines, and we report results of the robust imitation learning based approach as in [46]. We evaluate all environments under 10-step untargeted PGD attacks, except that results from [46] are evaluated using a weaker four-step PGD attack. For the most robust Atari models (SA-DQN convex), we additionally attack them using 50-step PGD attacks, and find that the rewards do not further reduce compared to the 10-step attacks.

In Table 5.2, we can see that our SA-DQN achieves much higher rewards under attacks in most environments, and naive adversarial training is mostly ineffective under strong attacks. We obtain better rewards than [46] in most environments, as we learn the agent directly rather than using two-step imitation learning.

5.4.3 Robustness certificates

When our robust policy regularizer is trained using convex relaxations, we can obtain certain robustness certification under observation perturbations. For a simple environ-

Table 5.2: Average rewards ± std and action certification rate over 50 episodes on three baselines and SA-DQN. We report natural rewards (no attacks) and PGD attack rewards (under 10-step PGD). For the most robust Atari models (SA-DQN convex), we additionally attack them using 50-step PGD attacks. Action Cert. Rate is the proportion of the actions during rollout that are guaranteed unchanged by any attacks within the given ϵ. **Bold** numbers indicate the most robust model; *italic* numbers indicate models with poor robustness. Training time is reported in Section D.6.

Environment		Pong	Freeway	BankHeist	RoadRunner
ℓ_∞ norm perturbation budget ϵ		1/255			
DQN (vanilla)	Natural Reward	21.0 ± 0.0	34.0 ± 0.2	1308.4 ± 24.1	45534.0 ± 7066.0
	PGD Attack Reward (10 steps)	*-21.0±0.0*	*0.0±0.0*	*56.4±21.2*	*0.0±0.0*
	Action Cert. Rate	0.0	0.0	0.0	0.0
DQN Adv. Training (attack 50% frames) Behzadan et al.[7]	Natural Reward	10.1 ± 6.6	25.4±0.8	1126.0±70.9	22944.0±6532.5
	PGD Attack Reward (10 steps)	*-21.0 ± 0.0*	*0.0±0.0*	*9.4±13.6*	*14.0±34.7*
	Action Cert. Rate	0.0	0.0	0.0	0.0
Imitation learning Fisher et al.[46]	Natural Reward	19.73	32.93	238.66	12106.67
	PGD Attack Reward (4 steps)	18.13	**32.53**	*190.67*	5753.33
SA-DQN (PGD)	Natural Reward	21.0±0.0	33.9 ± 0.4	1245.2±14.5	34032.0±3845.0
	PGD Attack Reward (10 steps)	21.0±0.0	23.7 ± 2.3	1006.0±226.4	20402.0±7551.1
	Action Cert. Rate	0.0	0.0	0.0	0.0
SA-DQN (convex)	Natural Reward	21.0 ± 0.0	30.0±0.0	1235.4±9.8	44638.0±7367.0
	PGD Attack Reward (10 steps)	**21.0 ± 0.0**	30.0±0.0	**1232.4±16.2**	**44732.0±8059.5**
	PGD Attack Reward (50 steps)	**21.0 ± 0.0**	30.0±0.0	**1234.6±16.6**	**44678.0±6954.0**
	Action Cert. Rate	1.000	1.000	0.984	0.475

ment like Pong, we can guarantee that actions do not change for all frames during rollouts, and thus can guarantee the accumulative rewards under perturbation. Unfortunately, for most RL tasks, due to the complexity of environments, it is impossible to obtain a "certified reward" as the certified test error in supervised learning settings. We leave further discussion to Section D.3.1 in the Appendix.

5.5 Discussion and Remarks

In this chapter, we develop State-Adversarial Markov Decision Process (SA-MDP) as a model for studying reinforcement learning problems under adversarial perturbations on state observations. Then, we apply our theoretically principled robustness regularizer to a few practical deep reinforcement learning algorithms, including PPO and DQN. Reinforcement learning has the great potential to be applied into many mission-critical tasks such as autonomous driving [134, 124, 175], if its robustness can be established. The robustness considered in this chapter is important for many realistic settings such as sensor noise, measurement errors, and man-in-the-middle (MITM) attacks for a DRL system. This chapter presents the first work that studies this problem in a fundamental and systematic manner, and derives principles that are widely applicable to existing RL algorithms.

Chapter 6

Conclusion and Future Works

This chapter discusses the contributions of this thesis, and possibilities for future work in this research area.

6.1 Thesis Contributions

In this thesis, we study the problem of machine learning model robustness against adversarial examples, which is vital to deploying machine learning models to security sensitive applications. Specifically, this thesis proposes pioneering methods for robustness verification and defense methods for decision tree-based machine learning models, as these topics for decision trees have been little studied in the literature. This thesis also empirically studies deep neural network robustness when test examples lie in low density regions of the training data empirical distribution. Finally, we apply robust deep neural networks in reinforcement learning and develop training method for robust agents against perturbation on observations. The major technical contributions of this thesis are summarized as below:

- A novel robustness verification method for tree ensemble classification model is developed. This method gives a bound such that the model classification output is guaranteed unchanged as long as the added perturbation has an ℓ_∞ norm less than this bound. This method is developed by casting the problem as a max-clique problem on a multi-partite graph. It is an efficient algorithm

that iteratively improves the bound and can be terminated at anytime during running. On large scale random forest or gradient boosted decision tree models, this method's output is shown to be very tight compared to the exact bound and is up to hundreds of times faster than a previous approach solving the exact bound.

- A novel training algorithm to improve the robustness of decision tree-based classification models is proposed. The perturbation needed to change the output label of robust models are much larger than those for natural models trained with regular tree learning algorithms. This new method takes possible perturbations on feature values into consideration and optimizes the model performance under the worst case perturbation of input data. Max-min saddle point optimization problems are solved during node splitting when building trees, and necessary approximations are made to drastically reduce the training time while the robustness improvement is preserved. This method is implemented both in simple information gain-based decision trees and in large scale gradient boosting models. Experimental results show that it can significantly enhance the robustness of models with hundreds of deep trees. Tree-based machine learning models are widely used in security sensitive industries such as finance and manufacturing. We believe our verification and robust training methods are directly applicable in global efforts to improve the efficiency and security of these industries.

- We empirically observe that the effectiveness of robust training methods in the literature for neural networks is highly correlated with the characteristics of the test data distribution. Even with the state-of-the-art robust training, test data examples that are far enough from the distribution of training data tend to be more vulnerable to adversarial attacks. Following this observation, a new class of attack called the "blind-spot attack" is proposed. A simple pixel value scale-and-shift scheme for conducting blind-spot attacks is able to drastically reduce the effectiveness of adversarial training on MNIST and Fashion MNIST

datasets. This phenomenon supports the existence of a robustness generalization gap, and argues that robust training can be challenging due to the prevalence of blind-spots in high dimensional datasets.

- Based on robust training methods developed for deep neural network classification models, we develop a novel algorithm to train robust reinforcement learning agents that can achieve high performance even when adversarial noise is present in the observations. We formulate these kinds of reinforcement learning problems under adversarial perturbations on state observations as State-Adversarial Markov Decision Processes (SA-MDP). Then we leverage robust training methods for deep neural networks to learn policies that are not sensitive to input perturbations. Specifically, a robustness regularizer is added to the training loss; the regularizer can be solved with either a gradient-based approach or convex relaxation on neural networks. Experimental results show that our method can train robust agents which are able to achieve high reward even under strong adversarial perturbations on observations. Applications of robust reinforcement learning methodologies are effectively endless in a variety of areas such as autonomous driving, predictive maintenance, and robot control, where measurement noise and security are major concerns.

6.2 Future Works

Future works in this area could involve further fundamental studies on why training robust machine learning models is difficult, and on how robust machine learning model can help us understand what is inside the black-box of machine learning models. In fact, adversarial examples being an annoying but intriguing property of current machine learning models is a valuable reminder that our understanding of this "alchemy" is still limited, and blindly using a powerful tool which we do not completely understand can be risky. This phenomenon also suggests that the way deep neural networks (or other function approximators) "see" the world may be fundamentally different than our eyes.

It is unclear how far we are from the next level of intelligence. It is possible that the mind-blowing progress of deep learning we have seen in the past several years is largely due to a dramatic growth of computation power and that we are still developing a very low level of "intelligence," by remembering a large amount of training data. Perhaps when researchers decades from now look back to the development of artificial intelligence in the early twenty first century, they will categorize all our current models — tree ensembles, neural networks or transformers, etc. — as "brute force." It is also possible that we are actually on a right track, but our datasets are still too small, and our computation power is still too limited. The recent GPT-3 model [12] with 175 billion parameters sheds some light on how powerful a gigantic machine learning model can be, even if its basic building blocks are still transformers. Finally and most likely, we may be at a preliminary stage of both methodology and computation power development, which means that we still have a very long way to go.

6.2.1 Challenges of Learning Robust Models

Theoretical Foundations – One of the most important questions in this area is that, is it really possible to learn large scale robust machine learning models? Existing robust training methods face two main drawbacks that prevent them from being successfully deployed in real-world applications. The first problem is that robust training methods require more training time, and the second is that when applied to large datasets, model performances drop significantly. Though many recent works try to make robust training more efficient and scalable [133, 176], robust model accuracy and training time are still far from that of regular training. Chapter 4 and some recent theoretical works suggest that learning robust models requires more computation power or more training data [16, 129].

Evaluation Metrics and Ground Truth – Another challenge in the research community is how to define a realistic metric for model robustness evaluation. Currently, ℓ_p norm distance is the most popular metric for image data due to its simplicity. The original definition of adversarial examples requires the noise to be "imperceptible" to human eyes [146], but in practice, it is very difficult and time-consuming to manually

examine a large dataset, and ℓ_p norm (usually ℓ_1, ℓ_2, and ℓ_∞) has become popular for automatic evaluation. However, many researchers have expressed their concerns on the suitability of ℓ_p norm, and even a small ℓ_p distance may also change the semantic meaning of an image; while even under large ℓ_p norm noise, some image meanings are still preserved [135, 42]. For other data types such as language data, ℓ_p norm may not be applicable. Also, we have many other data types that are inherently not human-interpretable, such as graph, time series, or unstructured data. The distance norms to formally define "imperceptible" for these datasets are not well-studied, either.

To identify the errors made by machine learning models, we need a ground truth. For interpretable data such as images and languages, this can be collected from human answers. But for uninterpretable data, the ground truth may require conducting simulations or real experiments, which can be expensive, time-consuming or even impossible. There are recent works using unlabeled data combined with a few labeled examples to train robust models [22, 153], which use a similar intuition as training robust reinforcement learning agents in Chapter 5. However, it is still unclear how supervision or ground truth play roles in model robustness. In fact, some recent works suggest that a discrepancy exists between the notion of model robustness from human's perspective and based on the geometry of the data [71].

6.2.2 Valuable Insights and Future Directions

Though machine learning model robustness is a largely unsolved open question with the number of research papers growing exponentially in the past seven years, we still see numerous valuable insights from this unique point of view. These new insights pave the way to many future research directions.

Connections between Robustness and Interpretability – Many recent works in this area, especially on deep learning models for image data, have identified an interesting connection between robustness and interpretability. For example, researchers in [151] discovered that robust model loss gradients with respect to input pixels have an intriguing alignment with the original image. This connection between the saliency map interpretability and adversarial robustness has been widely studied

in many follow-up works [44, 98, 23]. The reasons why this happens in robust models instead of in naturally trained models are still not fully understood from a theoretical point of view. It is well-known that deep neural network training loss is nonconvex and have numerous local minimum, but the reason why robust training can converge to such interesting local minima is also unclear.

Adversarial Example and Robust Training May Improve Model Performance – Recent studies also indicate that adversarial example and robust training may also improve the model performance on natural, unperturbed examples [167]. We also observe similar phenomena for the Humanoid experiment in Figure D-5 in Appendix D.7. Many robust training techniques have some connection to data augmentation and regularization, and such connections may create yet to be explored "side" effects on model performance.

Bibliography

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

[3] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[4] Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations*, 2019.

[5] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

[6] Vahid Behzadan and Arslan Munir. Vulnerability of deep reinforcement learning to policy induction attacks. In *International Conference on Machine Learning and Data Mining in Pattern Recognition*, pages 262–275. Springer, 2017.

[7] Vahid Behzadan and Arslan Munir. Whatever does not kill deep reinforcement learning, makes it stronger. *arXiv preprint arXiv:1712.09344*, 2017.

[8] Leo Breiman. *Classification and Regression Trees*. Routledge, 1984.

[9] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR*, 2018.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[11] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[12] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[13] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[14] Sébastien Bubeck, Ronen Eldan, and Joseph Lehec. Finite-time analysis of projected Langevin Monte Carlo. In *Advances in Neural Information Processing Systems*, pages 1243–1251, 2015.

[15] Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya Razenshteyn. Adversarial examples from cryptographic pseudo-random generators. *arXiv preprint arXiv:1811.06418*, 2018.

[16] Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. In *International Conference on Machine Learning*, pages 831–840, 2019.

[17] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. *International Conference on Learning Representations*, 2018.

[18] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pages 4790–4799, 2018.

[19] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv preprint arXiv:1902.06705*, 2019.

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

[21] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *arXiv preprint arXiv:1801.01944*, 2018.

[22] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems*, pages 11192–11203, 2019.

[23] Alvin Chan, Yi Tay, Yew Soon Ong, and Jie Fu. Jacobian adversarially regularized networks for robustness. *arXiv preprint arXiv:1912.10185*, 2019.

[24] L Sunil Chandran, Mathew C Francis, and Naveen Sivadasan. Geometric representation of graphs in low dimension using axis parallel boxes. *Algorithmica*, 56(2):129, 2010.

[25] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[26] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *International Conference on Machine Learning*, pages 1122–1131, 2019.

[27] Hongge Chen, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, and Cho-Jui Hsieh. Attacking visual language grounding with adversarial examples: A case study on neural image captioning. In *Annual Meeting of the Association for Computational Linguistics*, 2018.

[28] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. In *Advances in Neural Information Processing Systems*, pages 12317–12328, 2019.

[29] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: Elastic-net attacks to deep neural networks via adversarial examples. In *AAAI*, 2018.

[30] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[31] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[32] Tong Chen, Wenjia Niu, Yingxiao Xiang, Xiaoxuan Bai, Jiqiang Liu, Zhen Han, and Gang Li. Gradient band-based adversarial training for generalized attack immunity of A3C path finding. *arXiv preprint arXiv:1807.06752*, 2018.

[33] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, JinFeng Yi, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *International Conference on Learning Representations*, 2019.

[34] Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *arXiv preprint arXiv:1803.01128*, 2018.

[35] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pages 854–863, 2017.

[36] Daniel Cullina, Arjun Nitin Bhagoji, and Prateek Mittal. Pac-learning in the presence of evasion adversaries. *arXiv preprint arXiv:1806.01471*, 2018.

[37] Esther Derman, Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Soft-robust actor-critic policy-gradient. *arXiv preprint arXiv:1803.04848*, 2018.

[38] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

[39] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

[40] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *UAI*, 2018.

[41] Gil Einziger, Maayan Goldstein, Yaniv Sa'ar, and Itai Segall. Verifying robustness of gradient boosted models. In *AAAI*, 2019.

[42] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pages 3910–3920, 2018.

[43] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *arXiv preprint arXiv:2005.12729*, 2020.

[44] Christian Etmann, Sebastian Lunz, Peter Maass, and Carola Schoenlieb. On the connection between adversarial robustness and saliency map interpretability. In *International Conference on Machine Learning*, pages 1823–1832, 2019.

[45] Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. In *Neural Information Processing Systems*, 2018.

[46] Marc Fischer, Matthew Mirman, and Martin Vechev. Online robustness training for deep reinforcement learning. *arXiv preprint arXiv:1911.00887*, 2019.

[47] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2):337–407, 2000.

[48] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[49] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[50] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

[51] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[52] Saul B Gelfand and Sanjoy K Mitter. Recursive stochastic algorithms for global optimization in $\mathbb{R}^d$. *SIAM Journal on Control and Optimization*, 29(5):999–1018, 1991.

[53] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[54] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

[55] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[56] Zhaoyuan Gu, Zhenzhong Jia, and Howie Choset. Adversary A3C for robust reinforcement learning. *arXiv preprint arXiv:1912.00330*, 2019.

[57] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[58] Perttu Hämäläinen, Amin Babadi, Xiaoxiao Ma, and Jaakko Lehtinen. PPO-CMA: Proximal policy optimization with covariance matrix adaptation. *arXiv preprint arXiv:1810.02541*, 2018.

[59] Aaron Havens, Zhanhong Jiang, and Soumik Sarkar. Online robust policy learning in the presence of unknown adversaries. In *Advances in Neural Information Processing Systems*, pages 9916–9926, 2018.

[60] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[61] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2266–2276, 2017.

[62] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[63] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[64] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[65] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

[66] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Discriminative deep metric learning for face verification in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1875–1882, 2014.

[67] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Deep transfer metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 325–333, 2015.

[68] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[69] Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala Al-Fuqaha, Dinh Thai Hoang, and Dusit Niyato. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *arXiv preprint arXiv:2001.09684*, 2020.

[70] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning*, pages 2142–2151, 2018.

[71] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

[72] Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

[73] Kyle D Julian, Shivam Sharma, Jean-Baptiste Jeannin, and Mykel J Kochenderfer. Verifying aircraft collision avoidance neural networks through linear approximations of safe regions. *arXiv preprint arXiv:1903.00762*, 2019.

[74] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.

[75] Alex Kantchelian, JD Tygar, and Anthony Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pages 2387–2396, 2016.

[76] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[77] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Neural Information Processing Systems*, pages 3146–3154, 2017.

[78] Jernej Kos, Ian Fischer, and Dawn Song. Adversarial examples for generative models. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 36–42. IEEE, 2018.

[79] Jernej Kos and Dawn Song. Delving into adversarial attacks on deep policies. *arXiv preprint arXiv:1705.06452*, 2017.

[80] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[81] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[82] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *International Conference on Learning Representations*, 2017.

[83] Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[84] Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, and Stuart Russell. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4213–4220, 2019.

[85] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[86] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. *arXiv preprint arXiv:1703.06748*, 2017.

[87] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*, pages 157–163. Elsevier, 1994.

[88] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. *ECCV*, 2018.

[89] Xuanqing Liu and Cho-Jui Hsieh. Rob-gan: Generator, discriminator, and adversarial attacker. In *IEEE conference on Computer Vision and Pattern Recognition*, 2019.

[90] Xuanqing Liu, Yao Li, Chongruo Wu, and Cho-Jui Hsieh. Adv-bnn: Improved adversarial defense through robust bayesian neural network. In *International Conference on Learning Representations*, 2019.

[91] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[92] Björn Lütjens, Michael Everett, and Jonathan P How. Certified adversarial robustness for deep reinforcement learning. *arXiv preprint arXiv:1910.12908*, 2019.

[93] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Michael E Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[94] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[95] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

[96] Saeed Mahloujifar, Dimitrios I Diochnos, and Mohammad Mahmoody. The curse of concentration in robust learning: Evasion and poisoning attacks from concentration of measure. *arXiv preprint arXiv:1809.03063*, 2018.

[97] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3932–3939. IEEE, 2017.

[98] Puneet Mangla, Vedant Singh, and Vineeth N Balasubramanian. On saliency maps and adversarial robustness. *arXiv preprint arXiv:2006.07828*, 2020.

[99] Daniel J Mankowitz, Nir Levine, Rae Jeong, Abbas Abdolmaleki, Jost Tobias Springenberg, Timothy Mann, Todd Hester, and Martin Riedmiller. Robust reinforcement learning for continuous control with model misspecification. *arXiv preprint arXiv:1906.07516*, 2019.

[100] Daniel J Mankowitz, Timothy A Mann, Pierre-Luc Bacon, Doina Precup, and Shie Mannor. Learning robust options. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[101] Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[102] Mohammad Mirghorbani and P Krokhmal. On finding k-cliques in k-partite graphs. *Optimization Letters*, 7(6):1155–1165, 2013.

[103] Matthew Mirman, Marc Fischer, and Martin Vechev. Distilled agent DQN for provable adversarial robustness, 2018.

[104] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.

[105] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015.

[106] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[107] Arnab Nilim and Laurent El Ghaoui. Robustness in Markov decision problems with uncertain transition matrices. In *Advances in Neural Information Processing Systems*, pages 839–846, 2004.

[108] Takayuki Osogami. Robust partially observable Markov decision process. In *International Conference on Machine Learning*, pages 106–115, 2015.

[109] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[110] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[111] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519. ACM, 2017.

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

[113] Anay Pattanaik, Zhenyi Tang, Shuijing Liu, Gautham Bommannan, and Girish Chowdhary. Robust deep reinforcement learning with adversarial attacks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2040–2042. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[114] Sven Peter, Ferran Diego, Fred A Hamprecht, and Boaz Nadler. Cost efficient gradient boosting. In *Neural Information Processing Systems*, 2017.

[115] Charles A Phillips, Kai Wang, Erich J Baker, Jason A Bubier, Elissa J Chesler, and Michael A Langston. On finding and enumerating maximal and maximum k-partite cliques in k-partite graphs. *Algorithms*, 12(1):23, 2019.

[116] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

[117] Matteo Pirotta, Marcello Restelli, Alessio Pecorino, and Daniele Calandriello. Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315, 2013.

[118] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[119] J. Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[120] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *ICLR*, 2018.

[121] Maxim Raginsky, Alexander Rakhlin, and Matus Telgarsky. Non-convex learning via stochastic gradient Langevin dynamics: a nonasymptotic analysis. *arXiv preprint arXiv:1702.03849*, 2017.

[122] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[123] Fred S Roberts. On the boxicity and cubicity of a graph. *Recent Progresses in Combinatorics*, pages 301–310, 1969.

[124] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[125] Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Advances in Neural Information Processing Systems 32*, pages 9832–9842. Curran Associates, Inc., 2019.

[126] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

[127] Naoto Sato, Hironobu Kuruma, Yuichiroh Nakagawa, and Hideto Ogawa. Formal verification of decision-tree ensemble model and detection of its violating-input-value ranges. *arXiv preprint arXiv:1904.11753*, 2019.

[128] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[129] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *NIPS*, pages 5019–5031, 2018.

[130] Markus Schneider and Burkhard Wulfhorst. Cliques in k-partite graphs and their application in textile engineering. 2002.

[131] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[132] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[133] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3358–3369, 2019.

[134] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[135] Mahmood Sharif, Lujo Bauer, and Michael K Reiter. On the suitability of lp-norms for creating and preventing adversarial examples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1605–1613, 2018.

[136] Shirli Di-Castro Shashua and Shie Mannor. Deep robust Kalman filter. *arXiv preprint arXiv:1703.02310*, 2017.

[137] Si Si, Huan Zhang, S Sathiya Keerthi, Dhruv Mahajan, Inderjit S Dhillon, and Cho-Jui Hsieh. Gradient boosted decision trees for high dimensional sparse output. In *International Conference on Machine Learning*, 2017.

[138] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[139] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10802–10813, 2018.

[140] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.

[141] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. In *International Conference on Learning Representations*, 2018.

[142] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. *arXiv preprint arXiv:1710.10766*, 2017.

[143] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Generative adversarial examples. *arXiv preprint arXiv:1805.07894*, 2018.

[144] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy?–a comprehensive study on the robustness of 18 deep image classification models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648, 2018.

[145] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[146] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[147] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

[148] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. *arXiv preprint arXiv:1901.09184*, 2019.

[149] John Törnblom and Simin Nadjm-Tehrani. Formal verification of input-output mappings of tree ensembles. *arXiv preprint arXiv:1905.04194*, 2019.

[150] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations (ICLR)*, 2018. https://arxiv.org/abs/1705.07204.

[151] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *International Conference on Learning Representations*, 2019.

[152] Stephen Tyree, Kilian Q Weinberger, Kunal Agrawal, and Jennifer Paykin. Parallel boosted regression trees for web search ranking. In *Proceedings of the 20th International Conference on World Wide Web*, pages 387–396. ACM, 2011.

[153] Jonathan Uesato, Jean-Baptiste Alayrac, Po-Sen Huang, Robert Stanforth, Alhussein Fawzi, and Pushmeet Kohli. Are labels required for improving adversarial robustness? In *Advances in Neural Information Processing Systems*, 2019.

[154] Jonathan Uesato, Ananya Kumar, Csaba Szepesvari, Tom Erez, Avraham Ruderman, Keith Anderson, Nicolas Heess, Pushmeet Kohli, et al. Rigorous agent evaluation: An adversarial approach to uncover catastrophic failures. *arXiv preprint arXiv:1812.01647*, 2018.

[155] Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666*, 2018.

[156] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[157] Voyage. Introducing voyage deepdrive -unlocking the potential of deep reinforcement learning. https://news.voyage.auto/introducing-voyage-deepdrive-69b3cf0f0be6, 2019.

[158] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[159] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6367–6377, 2018.

[160] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003, 2016.

[161] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for ReLU networks. In *International Conference on Machine Learning*, pages 5273–5282, 2018.

[162] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292, 2018.

[163] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, pages 8400–8409, 2018.

[164] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610*, 2018.

[165] Chaowei Xiao, Xinlei Pan, Warren He, Jian Peng, Mingjie Sun, Jinfeng Yi, Bo Li, and Dawn Song. Characterizing attacks on deep reinforcement learning. *arXiv preprint arXiv:1907.09470*, 2019.

[166] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[167] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L Yuille, and Quoc V Le. Adversarial examples improve image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 819–828, 2020.

[168] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[169] Huan Xu and Shie Mannor. Distributionally robust markov decision processes. In *Advances in Neural Information Processing Systems*, pages 2505–2513, 2010.

[170] Kaidi Xu, Sijia Liu, Pu Zhao, Pin-Yu Chen, Huan Zhang, Quanfu Fan, Deniz Erdogmus, Yanzhi Wang, and Xue Lin. Structured adversarial attack: Towards general implementation and better interpretability. *ICLR*, 2019.

[171] Kaidi Xu, Zhouxing Shi, Huan Zhang, Minlie Huang, Kai-Wei Chang, Bhavya Kailkhura, Xue Lin, and Cho-Jui Hsieh. Automatic perturbation analysis on general computational graphs. *arXiv preprint arXiv:2002.12920*, 2020.

[172] Pan Xu, Jinghui Chen, Difan Zou, and Quanquan Gu. Global convergence of langevin dynamics based algorithms for nonconvex optimization. In *Advances in Neural Information Processing Systems*, pages 3122–3133, 2018.

[173] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[174] Zhixiang Eddie Xu, Matt J Kusner, Kilian Q Weinberger, and Alice X Zheng. Gradient regularized budgeted boosting. *arXiv preprint arXiv:1901.04065*, 2019.

[175] Changxi You, Jianbo Lu, Dimitar Filev, and Panagiotis Tsiotras. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. *Robotics and Autonomous Systems*, 114:1–18, 2019.

[176] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. In *Advances in Neural Information Processing Systems*, pages 227–238, 2019.

[177] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[178] Huan Zhang, Hongge Chen, Zhao Song, Duane Boning, Inderjit S Dhillon, and Cho-Jui Hsieh. The limitations of adversarial training and the blind-spot attack. In *International Conference on Learning Representations*, 2018.

[179] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Duane Boning, and Cho-Jui Hsieh. Towards stable and efficient training of verifiably robust neural networks. *ICLR*, 2020.

[180] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on observations. In *Advances in Neural Information Processing Systems*, 2020.

[181] Huan Zhang, Si Si, and Cho-Jui Hsieh. GPU-acceleration for large-scale tree boosting. *SysML Conference*, 2018.

[182] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pages 4939–4948, 2018.

[183] Yuchen Zhang, Percy Liang, and Moses Charikar. A hitting time analysis of stochastic gradient Langevin dynamics. *arXiv preprint arXiv:1702.05575*, 2017.

[184] Tianhang Zheng, Changyou Chen, and Kui Ren. Distributionally adversarial attack. *arXiv preprint arXiv:1808.05537*, 2018.

[185] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. In *International Joint Conferences on Artificial Intelligence*, 2017.

Appendix A

Appendix of Chapter 2

A.1 Proof of Theorem 1

Here we prove Theorem 1 for information gain score.

Proof. $H(y)$ and $H(y|x^{(j)} < \eta)$ are defined as

$$H(y) = -\frac{|\mathcal{I}_0|}{|\mathcal{I}|}\log(\frac{|\mathcal{I}_0|}{|\mathcal{I}|}) - \frac{|\mathcal{I}_1|}{|\mathcal{I}|}\log(\frac{|\mathcal{I}_1|}{|\mathcal{I}|}),$$

and

$$\begin{aligned} &H(y|x^{(j)} < \eta) = \\ &-\frac{|\mathcal{I}_L|}{|\mathcal{I}|}\left[\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_L|}\log(\frac{|\mathcal{I}_L \cap \mathcal{I}_0|}{|\mathcal{I}_L|}) + \frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_L|}\log(\frac{|\mathcal{I}_L \cap \mathcal{I}_1|}{|\mathcal{I}_L|})\right] \\ &-\frac{|\mathcal{I}_R|}{|\mathcal{I}|}\left[\frac{|\mathcal{I}_R \cap \mathcal{I}_0|}{|\mathcal{I}_R|}\log(\frac{|\mathcal{I}_R \cap \mathcal{I}_0|}{|\mathcal{I}_R|}) + \frac{|\mathcal{I}_R \cap \mathcal{I}_1|}{|\mathcal{I}_R|}\log(\frac{|\mathcal{I}_R \cap \mathcal{I}_1|}{|\mathcal{I}_R|})\right]. \end{aligned}$$

For simplicity, we denote $N_0 := |\mathcal{I}_0|$, $N_1 := |\mathcal{I}_1|$, $n_0 := |\mathcal{I}_L \cap \mathcal{I}_0|$ and $n_1 := |\mathcal{I}_L \cap \mathcal{I}_1|$. The information gain of this split can be written as a function of n_0 and n_1:

$$\begin{aligned} IG = C_1[&n_0\log(\frac{n_0}{N_0(n_1+n_0)}) + n_1\log(\frac{n_1}{N_1(n_1+n_0)}) \\ &+ (N_0-n_0)\log(\frac{N_0-n_0}{N_0(N_1+N_0-n_1-n_0)}) \\ &+ (N_1-n_1)\log(\frac{N_1-n_1}{N_1(N_1+N_0-n_1-n_0)})] + C_2, \end{aligned} \tag{A.1}$$

Algorithm 5 Finding Δn_0^* and Δn_1^* to Minimize Information Gain or Gini Impurity

Input: N_0 and N_1, number of instances with label 0 and 1. n_0^o and n_1^o, number of instances with label 0 and 1 that are certainly on the left.
Input: $|\Delta\mathcal{I} \cap \mathcal{I}_0|$ and $|\Delta\mathcal{I} \cap \mathcal{I}_1|$, number of instances with label 0 and 1 that can be perturbed.
Output: Δn_0^*, Δn_1^*, optimal number of points with label 0 and 1 in $\Delta\mathcal{I}$ to be place on the left.
$\Delta n_0^* \leftarrow 0,\ \Delta n_1^* \leftarrow 0,\ \text{min\_diff} \leftarrow |\frac{n_0^o}{N_0} - \frac{n_1^o}{N_1}|$;
for $\Delta n_0 \leftarrow 0$ **to** $|\Delta\mathcal{I} \cap \mathcal{I}_0|$ **do**
  ceil $\leftarrow \lceil \frac{N_1(n_0^o+\Delta n_0)}{N_0} \rceil - n_1^o$;
  floor $\leftarrow \lfloor \frac{N_1(n_0^o+\Delta n_0)}{N_0} \rfloor - n_1^o$;
  for $\Delta n_1'$ **in** {ceil, floor} **do**
    $\Delta n_1 \leftarrow \max\{\min\{\Delta n_1',\ |\Delta\mathcal{I} \cap \mathcal{I}_1|\},\ 0\}$;
    if min_diff $> |\frac{\Delta n_0+n_0^0}{N_0} - \frac{\Delta n_1+n_1^0}{N_1}|$ **then**
      $\Delta n_0^* \leftarrow \Delta n_0,\ \Delta n_1^* \leftarrow \Delta n_1,\ \text{min\_diff} \leftarrow |\frac{\Delta n_0+n_0^0}{N_0} - \frac{\Delta n_1+n_1^0}{N_1}|$;
    end if
  end for
end for
Return Δn_0^* and Δn_1^*;

where $C_1 > 0$ and C_2 are constants with respect to n_0. Taking n_0 as a continuous variable, we have

$$\frac{\partial IG}{\partial n_0} = C_1 \cdot \log(1 + \frac{n_0 N_1 - N_0 n_1}{(N_0 - n_0)(n_1 + n_0)}) \tag{A.2}$$

When $\frac{\partial IG}{\partial n_0} < 0$, perturbing one example in $\Delta\mathcal{I}_R$ with label 0 to $\mathcal{I}_L$ will increase n_0 and decrease the information gain. It is easy to see that $\frac{\partial IG}{\partial n_0} < 0$ if and only if $\frac{n_0}{N_0} < \frac{n_1}{N_1}$. This indicates that when $\frac{n_0}{N_0} < \frac{n_1}{N_1}$ and $\frac{n_0+1}{N_0} \leq \frac{n_1}{N_1}$, perturbing one example with label 0 to $\mathcal{I}_L$ will always decrease the information gain. □

Similarly, if $\frac{n_1}{N_1} < \frac{n_0}{N_0}$ and $\frac{n_1+1}{N_1} \leq \frac{n_0}{N_0}$, perturbing one example in $\Delta\mathcal{I}_R$ with label 1 to $\mathcal{I}_L$ will decrease the information gain. As mentioned in the main text, to decrease the information gain score in Algorithm 1, the adversary needs to perturb examples in $\Delta\mathcal{I}$ such that $\frac{n_0}{N_0}$ and $\frac{n_1}{N_1}$ are close to each other. Algorithm 5 gives an $O(|\Delta\mathcal{I}|)$ method to find Δn_0^* and Δn_1^*, the optimal number of points in $\Delta\mathcal{I}$ with label 0 and 1 to be added to the left.

A.2 Gini Impurity Score

We also have a theorem for Gini impurity score similar to Theorem 1.

Theorem 8. *If $\frac{n_0}{N_0} < \frac{n_1}{N_1}$ and $\frac{n_0+1}{N_0} \leq \frac{n_1}{N_1}$, perturbing one example in $\Delta\mathcal{I}_R$ with label 0 to $\mathcal{I}_L$ will decrease the Gini impurity.*

Proof. The Gini impurity score of a split with threshold η on feature j is

$$\begin{aligned} Gini = &(1 - \frac{|\mathcal{I}_0|^2}{|\mathcal{I}|^2} - \frac{|\mathcal{I}_1|^2}{|\mathcal{I}|^2}) \\ &- \frac{|\mathcal{I}_L|}{|\mathcal{I}|}(1 - \frac{|\mathcal{I}_0 \cap \mathcal{I}_L|^2}{|\mathcal{I}_L|^2} - \frac{|\mathcal{I}_1 \cap \mathcal{I}_L|^2}{|\mathcal{I}_L|^2}) \\ &- \frac{|\mathcal{I}_R|}{|\mathcal{I}|}(1 - \frac{|\mathcal{I}_0 \cap \mathcal{I}_R|^2}{|\mathcal{I}_R|^2} - \frac{|\mathcal{I}_1 \cap \mathcal{I}_R|^2}{|\mathcal{I}_R|^2}) \\ &= C_3[\frac{n_0^2 + n_1^2}{n_1 + n_0} + \frac{(N_0 - n_0)^2 + (N_1 - n_1)^2}{(N_0 + N_1 - n_0 - n_1)}] + C_4, \end{aligned} \tag{A.3}$$

where we use the same notation as in (A.1). $C_3 > 0$ and C_4 are constants with respect to n_0. Taking n_0 as a continuous variable, we have

$$\frac{\partial\ Gini}{\partial n_0} = 2C_3 \frac{m_1 m_0 (n_0 m_1 + n_1 m_0 + 2n_1 m_1)}{(n_0 + n_1)^2 (m_0 + m_1)^2} (\frac{n_0}{m_0} - \frac{n_1}{m_1}), \tag{A.4}$$

where $m_0 := N_0 - n_0$ and $m_1 := N_1 - n_1$. Then $\frac{\partial\ Gini}{\partial n_0} < 0$ holds if $\frac{n_0}{m_0} < \frac{n_1}{m_1}$, which is equivalent to $\frac{n_0}{N_0} < \frac{n_1}{N_1}$. □

Since the conditions of Theorem 1 and Theorem 8 are the same, Algorithm 1 and Algorithm 5 also work for tree-based models using Gini impurity score.

A.3 Decision Boundaries of Robust and Natural Models

Figure A-1 shows the decision boundaries and test accuracy of natural trees as well as robust trees with different ϵ values on two dimensional synthetic datasets. All trees have depth 5 and we plot training examples in the figure. The results show

that the decision boundaries of our robust decision trees are simpler than the decision boundaries in natural decision trees, agreeing with the regularization argument in the main text.

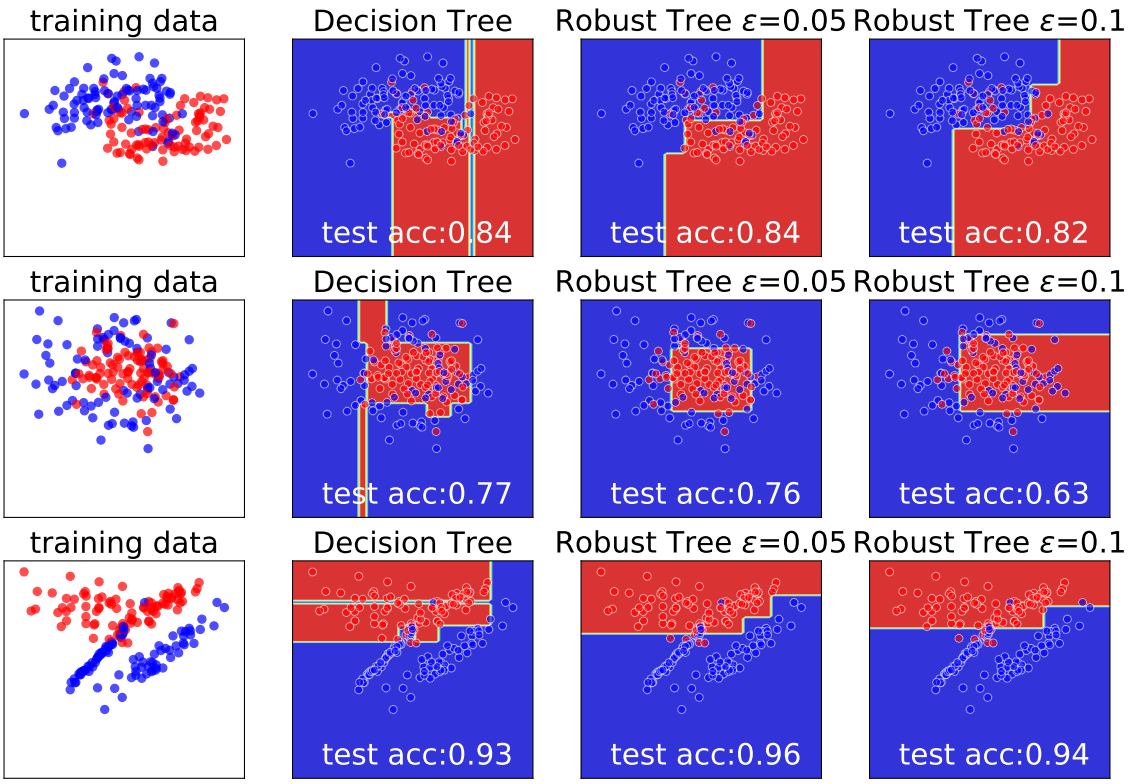


Figure A-1: (Best viewed in color) The decision boundaries and test accuracy of natural decision trees and robust decision trees with depth 5 on synthetic datasets with two features.

A.4 Omitted Results on ℓ_1 and ℓ_2 distortion

In Tables A.1 and A.2 we present the ℓ_1 and ℓ_2 distortions of vanilla (information gain based) decision trees and GBDT models obtained by Kantchelian's ℓ_1 and ℓ_2 attacks. Again, only small or medium sized binary classification models can be evaluated by Kantchelian's attack. From the results we can see that although our robust decision tree training algorithm is designed for ℓ_∞ perturbations, it can also improve models ℓ_1 and ℓ_2 robustness significantly.

A.5 Omitted Results on Models with Different Number of Trees

Figure A-2 shows the ℓ_∞ distortion and accuracy of Fashion-MNIST GBDT models with different number of trees. In Table A.4 we present the test accuracy and ℓ_∞ distortion of models with different number of trees obtained by Cheng's ℓ_∞ attack. For each dataset, models are generated during a single boosting run. We can see that the robustness of robustly trained models consistently outperforms that of natural models with the same number of trees. Another interesting finding is that for MNIST and Fashion-MNIST datasets in Figures 2-3 (in the main text) and A-2, models with more trees are generally more robust. This may not be true in other datasets; for example, results from Table A.4 in the Appendix shows that on some other datasets, the natural GBDT models lose robustness when more trees are added.

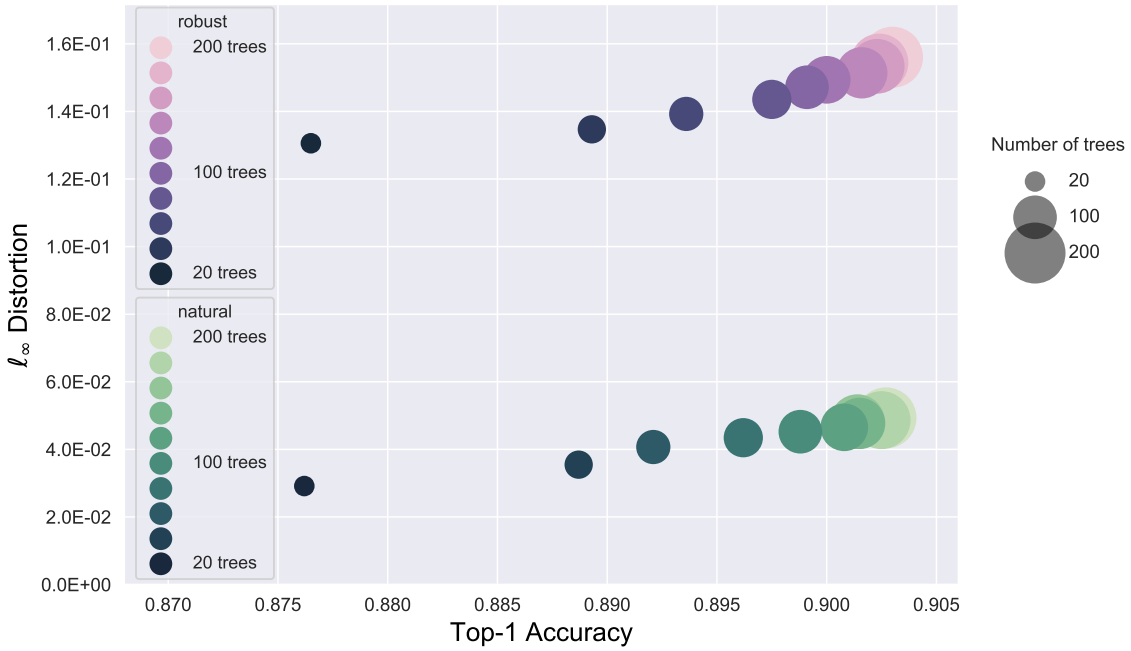


Figure A-2: (Best viewed in color) ℓ_∞ distortion vs. classification accuracy of GBDT models on Fashion-MNIST datasets with different numbers of trees (circle size). The adversarial examples are found by Cheng's ℓ_∞ attack. The robust training parameter $\epsilon = 0.1$ for Fashion-MNIST. With robust training (purple) the distortion needed to fool a model increases dramatically with less than 1% accuracy loss.

A.6 Reducing Depth Does Not Improve Robustness

One might hope that one can simply reduce the depth of trees to improve robustness since shallower trees provide stronger regularization effects. Unfortunately, this is not true. As demonstrated in Figure A-3, the robustness of naturally trained GBDT models are much worse when compared to robust models, no matter how shallow they are or how many trees are in the ensemble. Also, when the number of trees in the ensemble model is limited, reducing tree depth will significantly lower the model accuracy.

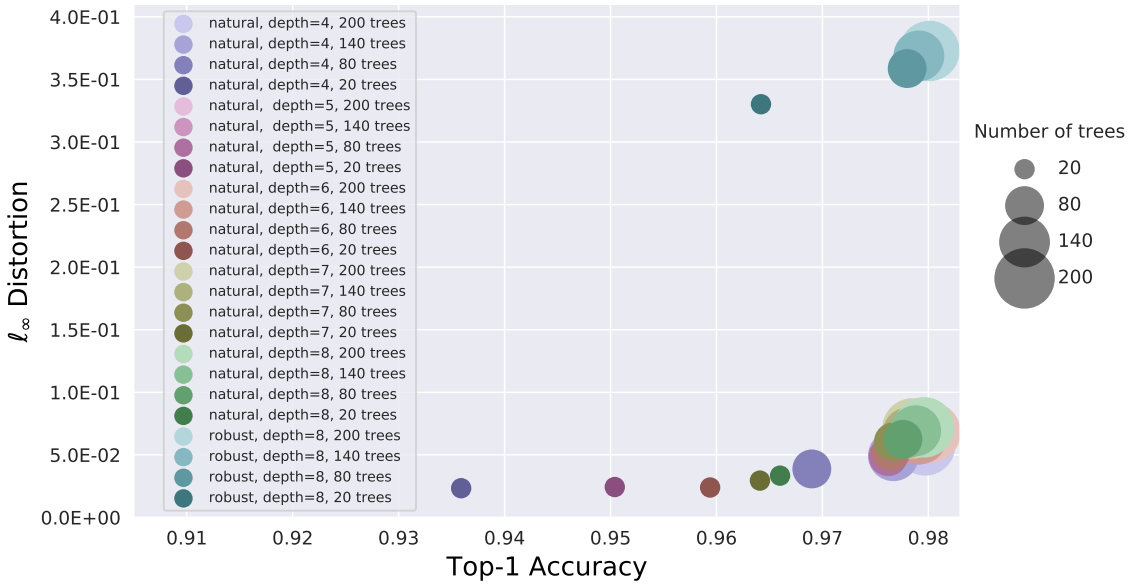


Figure A-3: (Best viewed in color) Robustness vs. classification accuracy plot of GBDT models on MNIST dataset with different depth and different numbers of trees. The adversarial examples are found by Cheng's ℓ_∞ attack. The robust training parameter $\epsilon = 0.3$. Reducing the model depth cannot improve robustness effectively compared to our proposed robust training procedure.

A.7 Random Forest Model Results

We test our robust training framework on random forest (RF) models and our results are in Table A.3. In these experiments we build random forest models with 0.5 data sampling rate and 0.5 feature sampling rate. We test the robust and natural random forest model on three datasets and in each dataset, we tested 100 points using Cheng's

and Kantchelian's ℓ_∞ attacks. From the results we can see that our robust decision tree training framework can also significantly improve random forest model robustness.

A.8 More MNIST and Fashion-MNIST Adversarial Examples

In Figure A-4 we present more adversarial examples for MNIST and Fashion-MNIST datasets using GBDT models.

Dataset	training set size	test set size	# of features	# of classes	robust ϵ	depth		test acc.		**avg. ℓ_1 dist. by Kantchelian's ℓ_1 attack**		**avg. ℓ_2 dist. by Kantchelian's ℓ_2 attack**	
						robust	natural	robust	natural	robust	natural	robust	natural
breast-cancer	546	137	10	2	0.3	5	5	.948	.942	**.534**	.270	**.504**	.209
diabetes	614	154	8	2	0.2	5	5	.688	.747	**.204**	.075	**.204**	.065
ionosphere	281	70	34	2	0.2	4	4	.986	.929	**.358**	.127	**.358**	.106

Table A.1: The test accuracy and robustness of information gain based single decision tree models. The robustness is evaluated by the average ℓ_1 and ℓ_2 distortions of adversarial examples found by Kantchelian's ℓ_1 and ℓ_2 attacks. Average ℓ_∞ distortions of robust decision tree models found by the two attack methods are consistently larger than those of the naturally trained ones.

Dataset	training set size	test set size	# of features	# of classes	# of trees	robust ϵ	depth		test acc.		**avg. ℓ_1 dist. by Kantchelian's ℓ_1 attack**		dist. improv.	**avg. ℓ_2 dist. by Kantchelian's ℓ_2 attack**		dist. improv.
							robust	natural	robust	natural	robust	natural		robust	natural	
breast-cancer	546	137	10	2	4	0.3	8	6	.978	.964	**.488**	.328	**1.49X**	**.431**	.251	**1.72X**
cod-rna	59,535	271,617	8	2	80	0.2	5	4	.880	.965	**.065**	.059	**1.10X**	**.062**	.047	**1.32X**
diabetes	614	154	8	2	20	0.2	5	5	.786	.773	**.150**	.081	**1.85X**	**.135**	.059	**2.29X**
ijcnn1	49,990	91,701	22	2	60	0.1	8	8	.959	.980	**.057**	.051	**1.12X**	**.048**	.042	**1.14X**
MNIST 2 vs. 6	11,876	1,990	784	2	1000	0.3	6	4	.997	.998	**1.843**	.721	**2.56X**	**.781**	.182	**4.29X**

Table A.2: The test accuracy and robustness of GBDT models. Average ℓ_1 and ℓ_2 distortions of robust GBDT models are consistently larger than those of the naturally trained models. The robustness is evaluated by the average ℓ_1 and ℓ_2 distortions of adversarial examples found by Kantchelian's ℓ_1 and ℓ_2 attacks.

Dataset	training set size	test set size	# of features	# of classes	# of trees	robust ϵ	depth		test acc.		**avg. ℓ_∞ distance by Cheng's ℓ_∞ attack**		dist. improv.	**avg. ℓ_∞ distance by Kantchelian's ℓ_∞ attack**		dist. improv.
							robust	natural	robust	natural	robust	natural		robust	natural	
breast-cancer	546	137	10	2	60	0.3	8	6	.993	.993	**.406**	.297	**1.37X**	**.396**	.244	**1.62X**
diabetes	614	154	8	2	60	0.2	5	5	.753	.760	**.185**	.093	**1.99X**	**.154**	.072	**2.14X**
MNIST 2 vs. 6	11,876	1,990	784	2	1000	0.3	6	4	.986	.983	**.445**	.180	**2.47X**	**.341**	.121	**2.82X**

Table A.3: The test accuracy and robustness of random forest models. Average ℓ_∞ distortion of our robust random forest models are consistently larger than those of the naturally trained models. The robustness is evaluated by the average ℓ_∞ distortion of adversarial examples found by Cheng's and Kantchelian's attacks.

breast-cancer (2)	train	test	feat.	# of trees	1		2		3		4		5		6		7		8		9		10	
$\epsilon = 0.3$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = 8,\ \mathbf{depth}_n = 6$	546	137	10	tst. acc.	.985	.942	.971	.964	.978	.956	.978	.964	.985	.964	.985	.964	.985	.971	.993	.971	.993	.971	1.00	.971
				ℓ_∞ dist.	**.383**	.215	**.396**	.229	**.411**	.216	**.411**	.215	**.406**	.226	**.407**	.229	**.406**	.248	**.439**	.234	**.439**	.238	**.437**	.241
covtype (7)	train	test	feat.	# of trees	20		40		60		80		100		120		140		160		180		200	
$\epsilon = 0.2$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 8$	400,000	181,000	54	tst. acc.	.775	.828	.809	.850	.832	.865	.847	.877	.858	.891	.867	.902	.875	.912	.882	.921	.889	.926	.894	.930
				ℓ_∞ dist.	**.125**	.066	**.103**	.064	**.087**	.062	**.081**	.061	**.079**	.060	**.077**	.059	**.077**	.058	**.075**	.056	**.075**	.056	**.073**	.055
cod-rna (2)	train	test	feat.	# of trees	20		40		60		80		100		120		140		160		180		200	
$\epsilon = 0.2$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = 5,\ \mathbf{depth}_n = 4$	59,535	271,617	8	tst. acc.	.810	.947	.861	.959	.874	.963	.880	.965	.892	.966	.900	.967	.903	.967	.915	.967	.922	.967	.925	.968
				ℓ_∞ dist.	**.077**	.057	**.066**	.055	**.063**	.054	**.062**	.053	**.059**	.053	**.057**	.052	**.056**	.052	**.056**	.052	**.056**	.052	**.058**	.052
diabetes (2)	train	test	feat.	# of trees	2		4		6		8		10		12		14		16		18		20	
$\epsilon = 0.2$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 5$	614	154	8	tst. acc.	.760	.753	.760	.753	.766	.753	.773	.753	.773	.734	.779	.727	.779	.747	.779	.760	.779	.773	.786	.773
				ℓ_∞ dist.	**.163**	.066	**.163**	.065	**.154**	.071	**.151**	.071	**.152**	.073	**.148**	.072	**.146**	.067	**.144**	.062	**.138**	.062	**.139**	.060
Fashion-MNIST (10)	train	test	feat.	# of trees	20		40		60		80		100		120		140		160		180		200	
$\epsilon = 0.1$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 8$	60,000	10,000	784	tst. acc.	.877	.876	.889	.889	.894	.892	.898	.896	.899	.899	.900	.901	.902	.902	.902	.901	.902	.903	.903	.903
				ℓ_∞ dist.	**.131**	.029	**.135**	.035	**.139**	.041	**.144**	.043	**.147**	.045	**.149**	.047	**.151**	.048	**.153**	.048	**.154**	.049	**.156**	.049
HIGGS (2)	train	test	feat.	# of trees	50		100		150		200		250		300		350		400		450		500	
$\epsilon = 0.05$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 8$	10,500,000	500,000	28	tst. acc.	.676	.747	.688	.753	.700	.755	.702	.758	.705	.759	.709	.760	.711	.762	.712	.764	.716	.763	.718	.764
				ℓ_∞ dist.	**.023**	.013	**.023**	.014	**.022**	.014	**.022**	.014	**.022**	.014	**.022**	.014	**.021**	.015	**.021**	.015	**.021**	.015	**.021**	.015
ijcnn1 (2)	train	test	feat.	# of trees	10		20		30		40		50		60		70		80		90		100	
$\epsilon = 0.1$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 8$	49,990	91,701	22	tst. acc.	.933	.973	.942	.977	.947	.977	.952	.979	.958	.980	.959	.980	.962	.980	.964	.980	.967	.980	.968	.980
				ℓ_∞ dist.	**.065**	.048	**.061**	.047	**.058**	.048	**.057**	.047	**.054**	.048	**.054**	.047	**.054**	.047	**.053**	.047	**.052**	.047	**.052**	.047
MNIST (10)	train	test	feat.	# of trees	20		40		60		80		100		120		140		160		180		200	
$\epsilon = 0.3$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 8$	60,000	10,000	784	tst. acc.	.964	.966	.973	.975	.977	.977	.978	.978	.978	.978	.979	.979	.979	.979	.980	.979	.980	.979	.980	.980
				ℓ_∞ dist.	**.330**	.033	**.343**	.049	**.352**	.057	**.359**	.062	**.363**	.065	**.367**	.067	**.369**	.069	**.370**	.071	**.371**	.072	**.373**	.072
Sensorless (11)	train	test	feat.	# of trees	3		6		9		12		15		18		21		24		27		30	
$\epsilon = 0.05$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 6$	48,509	10,000	48	tst. acc.	.834	.977	.867	.983	.902	.987	.923	.991	.945	.992	.958	.994	.966	.996	.971	.996	.974	.997	.978	.997
				ℓ_∞ dist.	**.037**	.022	**.036**	.022	**.035**	.023	**.035**	.023	**.035**	.023	**.035**	.023	**.035**	.023	**.035**	.023	**.035**	.023	**.035**	.023
webspam (2)	train	test	feat.	# of trees	10		20		30		40		50		60		70		80		90		100	
$\epsilon = 0.05$				model	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.	rob.	nat.
$\mathbf{depth}_r = \mathbf{depth}_n = 8$	300,000	50,000	254	tst. acc.	.950	.976	.964	.983	.970	.986	.973	.989	.976	.990	.978	.990	.980	.991	.981	.991	.982	.992	.983	.992
				ℓ_∞ dist.	**.049**	.010	**.048**	.015	**.049**	.019	**.049**	.021	**.049**	.023	**.049**	.024	**.049**	.024	**.049**	.024	**.048**	.024	**.049**	.024

Table A.4: The test accuracy and robustness of GBDT models. Here depth_n is the depth of natural trees and depth_r is the depth of robust trees. Robustness is evaluated by the average ℓ_∞ distortion of adversarial examples found by Cheng's attack [33]. The number in the parentheses after each dataset name is the number of classes. Models are generated during a single boosting run. We can see that the robustness of our robust models consistently outperforms that of natural models with the same number of trees.

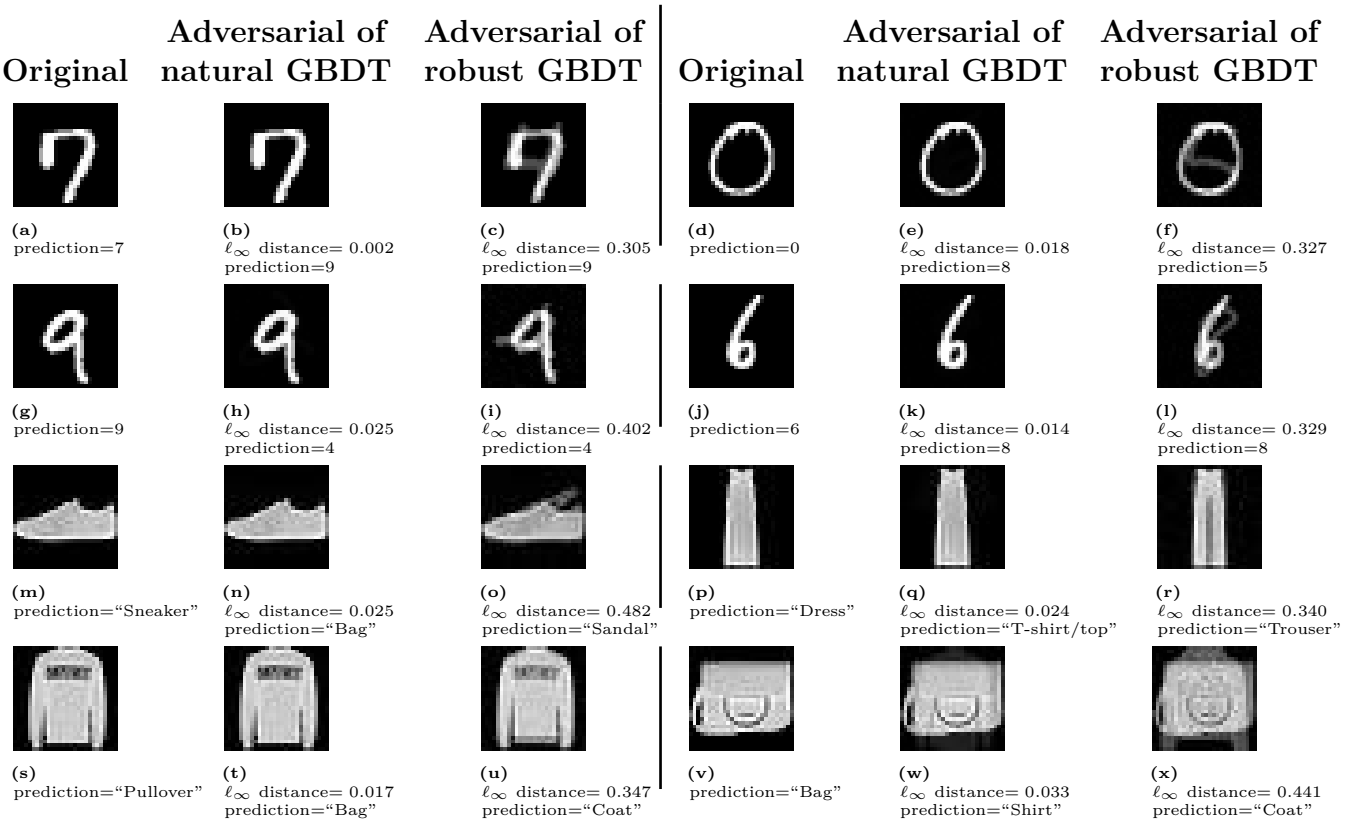


Figure A-4: MNIST and Fashion-MNIST examples and their adversarial examples found using the untargeted Cheng's ℓ_∞ attack [33] on 200-tree gradient boosted decision tree (GBDT) models trained using XGBoost with depth=8. For both MNIST and Fashion-MNIST robust models, we use $\epsilon = 0.3$.

Appendix B

Appendix of Chapter 3

B.1 Data Statistics and Model Parameters in Tables 3.1 and 3.2

Table B.1 presents data statistics and parameters for the models in Tables 3.1 and 3.2 in the main text. The standard test accuracy is the model accuracy on natural, unmodified test sets.

Dataset	training set size	test set size	# of features	# of classes	# of trees	robust ϵ	depth		standard test acc.	
							robust	natural	robust	natural
breast-cancer	546	137	10	2	4	0.3	8	6	.978	.964
covtype	400,000	181,000	54	7	80	0.2	8	8	.847	.877
diabetes	614	154	8	2	20	0.2	5	5	.786	.773
Fashion-MNIST	60,000	10,000	784	10	200	0.1	8	8	.903	.903
HIGGS	10,500,000	500,000	28	2	300	0.05	8	8	.709	.760
ijcnn1	49,990	91,701	22	2	60	0.1	8	8	.959	.980
MNIST	60,000	10,000	784	10	200	0.3	8	8	.980	.980
webspam	300,000	50,000	254	2	100	0.05	8	8	.983	.992
MNIST 2 vs. 6	11,876	1,990	784	2	1000	0.3	6	4	.997	.998

Table B.1: The data statistics and parameters for the models presented in Tables 3.1 and 3.2.

B.2 Results for Solving Single Layer Bounds with Dynamic Programming

In this section we provide results of our algorithm by using Eq. (3.5) for solving the last single layer bounds. Since using dynamic programming to find the maximum valued path in a graph can take significantly longer time than using (3.4), we found that the solving time increases noticeably if using the same T and L values. For some models, we reduce the values of T or L in order to speed up our method with dynamic programming. But even with smaller T or L values, the lower bounds $\underline{r}$ can also be improved with dynamic programming.

Dataset	MILP [75]		Ours (with DP)				Ours vs. MILP	
	avg. r^*	avg. time	T	L	avg. $\underline{r}_{our}$	avg. time	$\underline{r}_{our}/r^*$	speedup
breast-cancer	.210	.012s	2	1	.209	.001s	1.00	12X
covtype	.028*	355*s	2	3	.024	5.70s	.86	62X
diabetes	.049	.061s	2	2	.044	.013s	.90	4.7X
Fashion-MNIST	.014*	1150*s	2	1	.012	22.8s	.86	50X
HIGGS	.0028*	68*min	4	1	.0023	22.1s	.82	185X
ijcnn1	.030	4.64s	2	1	.027	.053s	.90	88X
MNIST	.011*	367*s	2	1	.011	5.10s	1.00	72X
webspam	.00076	47.2s	2	1	.00051	3.29s	.67	14X
MNIST 2 vs. 6	.057	23.0s	4	1	.050	2.41s	.88	9.5X

Table B.2: Average ℓ_∞ distortion over 500 examples and average verification time per example for three verification methods. Here we evaluate the bounds for **standard (natural) GBDT models**. Results marked with a star ("$\star$") are the averages of 50 examples due to long running time. T is the number of independent sets and L is the number of levels in searching cliques used in our algorithm. A ratio $\underline{r}_{our}/r^*$ close to 1 indicates better lower bound quality.

Dataset	MILP [75]		Ours (with DP)				Ours vs. MILP	
	avg. r^*	avg. time	T	L	avg. $\underline{r}_{our}$	avg. time	$\underline{r}_{our}/r^*$	speedup
breast-cancer	.400	.009s	2	1	.399	.001s	1.00	9.0X
covtype	.046*	305*s	2	2	.035	3.69s	.76	83X
diabetes	.112	.034s	2	2	.111	.005s	.98	7.1X
Fashion-MNIST	.091*	41*min	2	1	.071	19.9s	.78	124X
HIGGS	.0084*	59*min	4	1	.0069	4.25s	.82	783X
ijcnn1	.036	2.52s	2	2	.035	.655s	.97	3.8X
MNIST	.264*	615*s	2	1	.264	7.74s	1.00	63X
webspam	.015	83.7s	2	1	.011	1.26s	.73	66X
MNIST 2 vs. 6	.313	91.5s	2	1	.309	5.91s	.99	15.5X

Table B.3: Verification bounds and running time for **robustly trained GBDT models** introduced in [26]. The settings for each method are similar to the settings in Table B.2.

B.3 Connection between the Score in Figure 3-4 and Other Feature Importance Scores

We note that our perturbation-sensitivity notion of feature importance is complementary to the conventional tree/forest feature importance, with several critical differences. In Figure B-1 below we show the feature importance map of the same standard and robust models used in Figure 3-4 in the main text. A feature's importance is measured by the average gain across all the splits it is used in. Pixels with darker color have larger importance and yellow pixels have zero importance. Our single-feature robustness bounds shown in Figure 4 are different from importance scores (Figure B-1) in the following ways:

- The conventional feature importance score only depends on the model itself, and is test data independent. Conversely, our single-feature robustness bound depends on both the model and the test data point; for different data points, the model may be sensitive to different features.
- The conventional feature importance is a heuristic score. Our robustness bound can give a formal guarantee that the model output would not change if this single feature is perturbed within a given range.
- The conventional feature importance score assigns non-zero importance to more pixels than our method does in general.

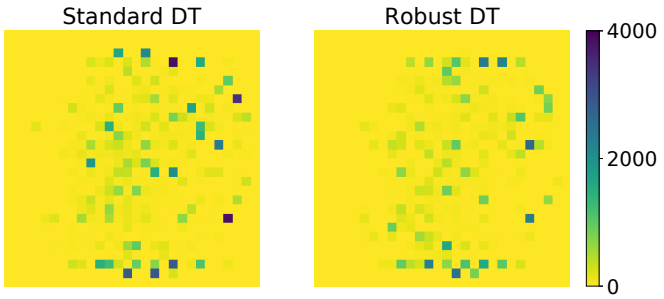


Figure B-1: Feature importance of the same models as in Figure 4 in the main text. Left: standard DT model; Right: robust DT model. Yellow pixels have zero feature importance while darker pixels have larger importance. A feature's importance is measured by the average gain across all the splits it is used in.

B.4 An $O(n)$ time algorithm for verifying a decision tree

The robustness of a single tree can be easily verified by the following $O(n)$ algorithm, which traverse the whole tree and computes the bounding boxes for each node in a depth-first search fashion.

Algorithm 6 A linear time ℓ_∞ untargeted attack for single decision trees.

Initial $p^* = 0, \ell_t = -\infty, r_t = \infty, \quad \forall t = 1, \dots d$

output : `ComputeRecursive`$(0, 0)$

Function `ComputeRecursive`(i, p)

- **if** *i is leaf node* **then**
  - **if** $v_i \neq y_0$ **then**
    - $p^* \leftarrow \min(p^*, p)$
- **else**
  - `/* Checking conditions for the left child */`
  - $s \leftarrow r_{t_i}$, $r_{t_i} \leftarrow \min(r_{t_i}, l_{t_i})$
  - **if** $l_{t_i} \leq r_{t_i}$ **then**
    - **if** $r_{t_i} < x_{t_i}$ **then**
      - `ComputeRecursive`$(i.$*left_child,* $\max(p, |x_{t_i} - r_{t_i}|))$
    - **else**
      - `ComputeRecursive`$(i.$*left_child,* $p)$
    - $r_{ti} \leftarrow s$
  - `/* Checking conditions for the right child */`
  - $s \leftarrow l_{t_i}$, $l_{t_i} \leftarrow \max(l_{t_i}, l_{t_i})$
  - **if** $l_{t_i} \leq r_{t_i}$ **then**
    - **if** $l_{t_i} > x_{t_i}$ **then**
      - `ComputeRecursive`$(i.$*right_child,* $\max(p, |x_{t_i} - l_{t_i}|))$
    - **else**
      - `ComputeRecursive`$(i.$*right_child,* $p)$

end

B.5 Proof of Lemma 1

Lemma 1. For boxes $B^1, \ldots, B^K$, if $B^i \cap B^j \neq \emptyset$ for all $i, j \in [K]$, let $\bar{B} = B^1 \cap B^2 \cap \cdots \cap B^K$ be their intersection. Then $\bar{B}$ will also be a box and $\bar{B} \neq \emptyset$.

Proof. If we have K one dimensional intervals $I_1 = (l_1, r_1], I_2 = (l_2, r_2], \ldots, I_T = (l_K, r_K]$, we want to prove if every pair of them have nonempty overlap $I_1 \cap \cdots \cap I_K \neq \emptyset$. This can be proved by the following. Without loss of generality we assume $l_1 \leq l_2 \leq \cdots \leq l_K$. For each $k < K$, $I_k \cap I_K \neq \emptyset$ implies $l_K < r_k$. Therefore, $(l_T, \min(r_1, r_2, \ldots, r_K)]$ will be a nonempty set that is contained in $I_1, I_2, \ldots, I_K$. Therefore $I_1 \cap I_2 \cap \cdots \cap I_K \neq \emptyset$ and it is another interval.

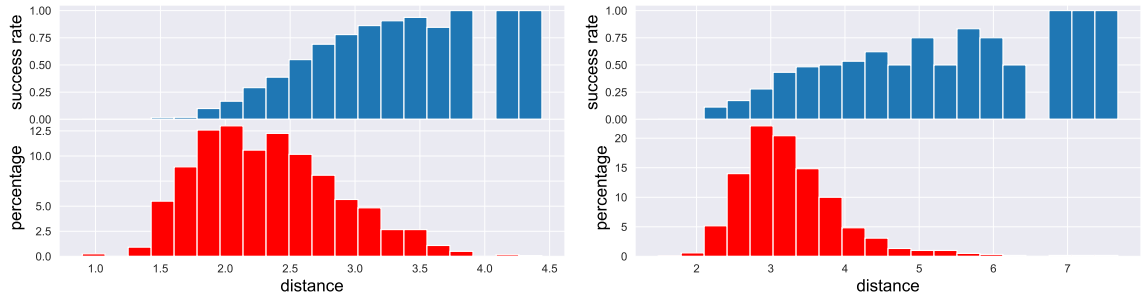
This can be generalized to d-dimensional boxes. Assume we have boxes $B_1, \ldots, B_K$ such that $B_i \cap B_j \neq \emptyset$ for any i and j. Then for each dimension we can apply the above proof, which implies that $B_1 \cap B_2 \cap \cdots \cap B_K \neq \emptyset$ and the intersection will be another box. □

Appendix C

Appendix of Chapter 4

C.1 Distance distributions under different nearest neighbour parameters k

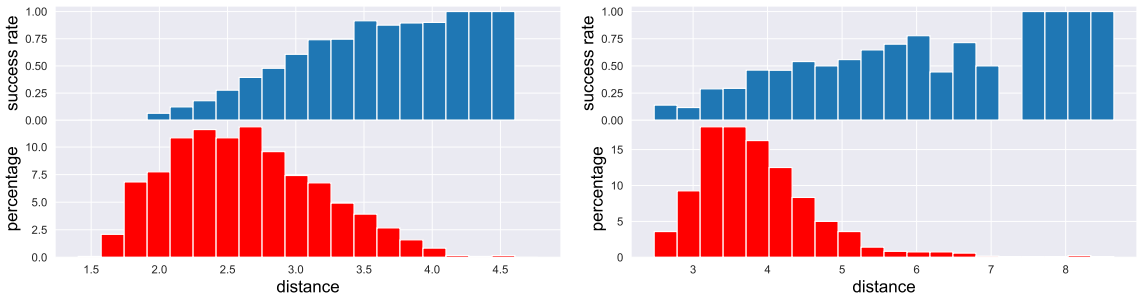
As discussed in Section 4.3.1, we use k-nearest neighbour in embedding space to measure the distance between a test example and the training set. In Section 4.4.2 we use $k = 5$. In this section we show that the choice of k does not have much influence on our results. We use the adversarially trained model on the CIFAR dataset as an example. In Figures C-1, C-2 and C-3 we choose $k = 10, 100, 1000$, respectively. The results are similar to those we have shown in Figure 4-3: a strong correlation between attack success rates and the distance from a test point to the training dataset.



(a) Using adversarially trained model as $h(x)$

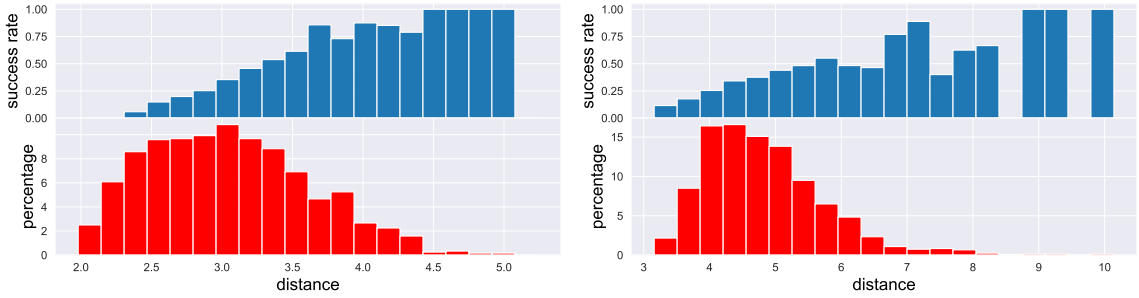
(b) Using naturally trained model as $h(x)$

Figure C-1: Attack success rates and distance distribution of the adversarially trained CIFAR model by [95]. Upper: C&W ℓ_∞ attack success rate, $\epsilon = 8/255$. Lower: distribution of the average ℓ_2 (embedding space) distance between the images in test set and the **top-10** ($k = 10$) nearest images in training set.



(a) Using adversarially trained model as $h(x)$ **(b)** Using naturally trained model as $h(x)$

Figure C-2: Attack success rates and distance distribution of the adversarially trained CIFAR model by [95]. Upper: C&W ℓ_∞ attack success rate, $\epsilon = 8/255$. Lower: distribution of the average ℓ_2 (embedding space) distance between the images in test set and the **top-100** ($k = 100$) nearest images in training set.



(a) Using adversarially trained model as $h(x)$ **(b)** Using naturally trained model as $h(x)$

Figure C-3: Attack success rates and distance distribution of the adversarially trained CIFAR model by [95]. Upper: C&W ℓ_∞ attack success rate, $\epsilon = 8/255$. Lower: distribution of the average ℓ_2 (embedding space) distance between the images in test set and the **top-1000** ($k = 1000$) nearest images in training set.

C.2 More visualization results

We demonstrate more MNIST and Fashion-MNIST visualizations in Figure C-4.

C.3 German traffic sign (GTS) dataset

We also studied the German Traffic Sign (GTS) [65] dataset. For GTS, we train our own model with the same model structure and parameters as the adversarially trained CIFAR model [95]. We set $\epsilon = 8/255$ for adversarial training with PGD, and also use the same ϵ as the threshold of success. The results are shown in Figure C-5. The GTS model behaves similarly to the CIFAR model: attack success rates are much higher when the distances between the test example and the training dataset are larger.

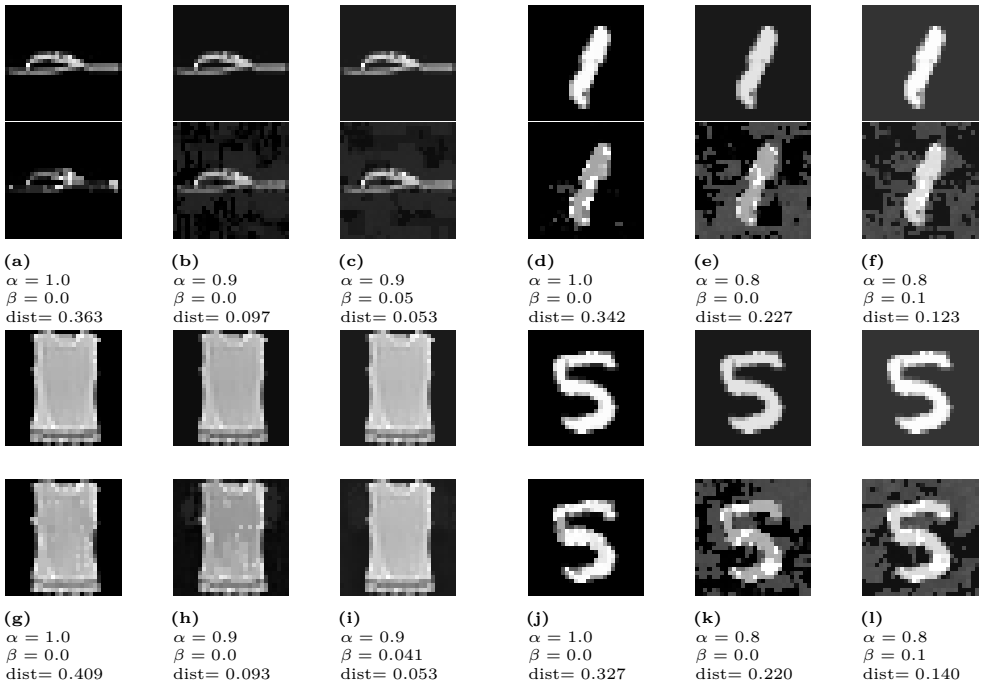
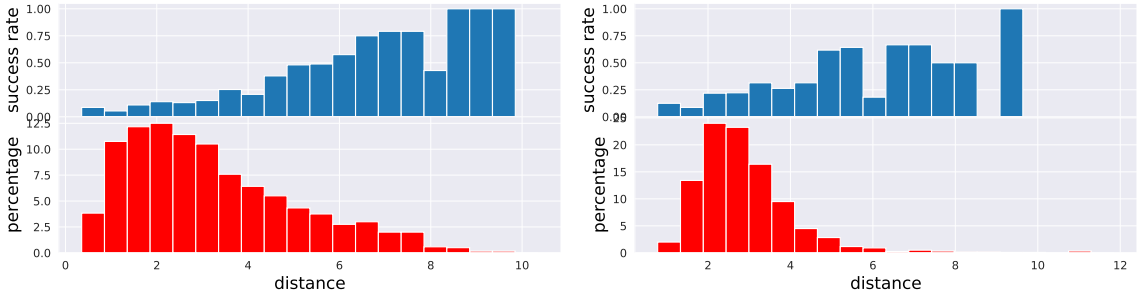


Figure C-4: Blind-spot attacks on Fashion-MNIST and MNIST data with scaling and shifting in [95]. First row contains input images after scaling and shifting and the second row contains the found adversarial examples. "dist" represents the ℓ_∞ distortion of adversarial perturbations. The first rows of figures (a), (d), (g) and (j) represent the original test set images ($\alpha = 1.0, \beta = 0.0$); first rows of figures (b), (c), (e), (f), (h), (i), (k) and (l) illustrate the images after transformation. Adversarial examples for these transformed images can be found with small distortions.



(a) Using adversarially trained model as $h(x)$

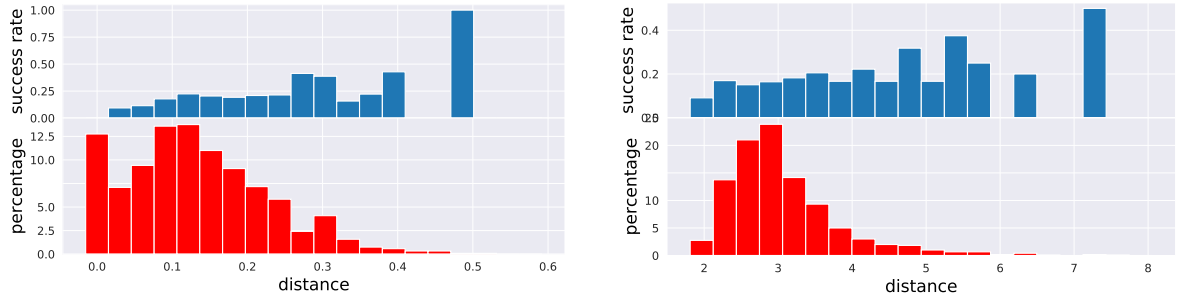
(b) Using naturally trained model as $h(x)$

Figure C-5: Attack success rate and distance distribution of GTS in [95]. Upper: C&W ℓ_∞ attack success rate, $\epsilon = 8/255$. Lower: distribution of the average ℓ_2 (embedding space) distance between the images in test set and the top-5 nearest images in training set.

C.4 Results on other robust training methods

In this section we demonstrate our experimental results on two other state-of-the-art *certified* defense methods, including convex adversarial polytope by [163] and [162], and distributional robust optimization based adversarial training by [141]. Different from the adversarial training by [95], these two methods can provide a formal certification on the robustness of the model and provably improve robustness on the training dataset.

However, they cannot practically guarantee non-trivial robustness on test data. We did not include other certified defenses like [120] and [61] because they are not applicable to multi-layer networks. For all defenses, we use their official implementations and pretrained models (if available). Figure C-6 shows the results on CIFAR using the small CIFAR model in [163]. Tables C.1 and C.2 show the blind-spot attack results on MNIST and Fashion-MNIST for robust models in [162] and [141], respectively. Figure C-7 shows the blind-spot attack examples on [95], [162] and [141].



(a) Using adversarially trained model as $h(x)$

(b) Using naturally trained model as $h(x)$

Figure C-6: Attack success rates and distance distribution of the small CIFAR-10 model in [163]. Lower: the histogram of the average ℓ_2 (in embedding space) distance between the images in test set and the top-5 nearest images in training set. Upper: the C&W ℓ_∞ attack success rate with success criterion $\epsilon = 8/255$.

MNIST	α, β	$\alpha = 1.0$	$\alpha = 0.95$				$\alpha = 0.9$			
		$\beta = 0$	$\beta = 0$		$\beta = 0.025$		$\beta = 0$		$\beta = 0.05$	
	Accuracy	97.5%	97.5%		97.5%		97.5%		97.4%	
	Success criterion (ℓ_∞ norm)	0.1	0.1	0.095	0.1	0.095	0.1	0.09	0.1	0.09
	Success rates	2.15%	5.55%	4.35%	28.5%	17.55%	30.1%	15.4%	**86.35%**	80.7%
Fashion-MNIST	α, β	$\alpha = 1.0$	$\alpha = 0.95$				$\alpha = 0.9$			
		$\beta = 0$	$\beta = 0$		$\beta = 0.025$		$\beta = 0$		$\beta = 0.05$	
	Accuracy	79.1%	79.1%		79.4%		79.2%		79.2%	
	Success criterion (ℓ_∞ norm)	0.1	0.1	0.095	0.1	0.095	0.1	0.09	0.1	0.09
	Success rates	6.85%	15.45%	9.3%	39.75%	29.35%	34.25%	24.65%	**69.95%**	65.2%

Table C.1: Blind-spot attack on MNIST and Fashion-MNIST for robust models by [162]

MNIST	α, β	$\alpha = 1.0$	$\alpha = 0.95$				$\alpha = 0.9$			
		$\beta = 0$	$\beta = 0$		$\beta = 0.025$		$\beta = 0$		$\beta = 0.05$	
	Accuracy	98.7%	98.5%		98.6%		98.7%		98.4%	
	Success criterion (ℓ_2 norm)	2	2	1.9	2	1.9	2	1.8	2	1.8
	Success rates	12.2%	27.05%	22.95%	36.15%	30.9%	45.25%	31.55%	**58.9%**	45.6%
Fashion-MNIST	α, β	$\alpha = 1.0$	$\alpha = 0.95$				$\alpha = 0.9$			
		$\beta = 0$	$\beta = 0$		$\beta = 0.025$		$\beta = 0$		$\beta = 0.05$	
	Accuracy	88.5%	88.3%		88.2%		88.1%		87.8%	
	Success criterion (ℓ_2 norm)	2	2	1.9	2	1.9	2	1.8	2	1.8
	Success rates	31.4%	46.3%	41.1%	58 %	53.3%	61.2%	51.8%	**69.1%**	62.85%

Table C.2: Blind-spot attack on MNIST and Fashion-MNIST for robust models by [141]. Note that we use ℓ_2 distortion for this model as it is the threat model under study in their work.

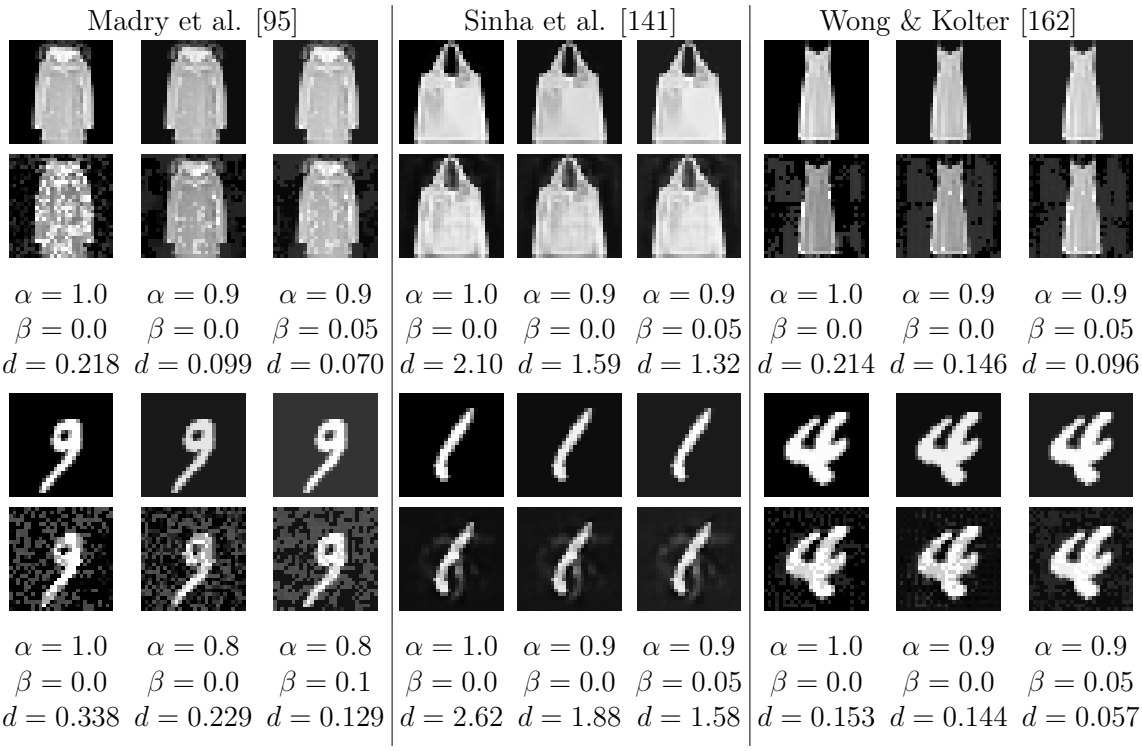


Figure C-7: Blind-spot attacks on Fashion-MNIST and MNIST datasets with scaling and shifting. For each group, the first row contains input images transformed with different scaling and shifting parameter α, β ($\alpha = 1.0, \beta = 0.0$ is the original image) and the second row contains the found adversarial examples. d represents the distortion of adversarial perturbations. For models from [95] and [162] we use ℓ_∞ norm and for models from [141] we use ℓ_2 norm. Adversarial examples for these transformed images can be found with small distortions d.

Appendix D

Appendix of Chapter 5

D.1 An example of SA-MDP

We first show a simple environment and solve it under different settings of MDP and SA-MDP. The environment have three states $\mathcal{S} = \{S_1, S_2, S_3\}$ and 2 actions $\mathcal{A} = \{A_1, A_2\}$. The transition probabilities and rewards are defined as (unmentioned probabilities and rewards are 0):

$$\Pr(s' = S_1 | s = S_1, a = A_1) = 1.0$$
$$\Pr(s' = S_2 | s = S_1, a = A_2) = 1.0$$
$$\Pr(s' = S_2 | s = S_2, a = A_2) = 1.0$$
$$\Pr(s' = S_3 | s = S_2, a = A_1) = 1.0$$
$$\Pr(s' = S_1 | s = S_3, a = A_2) = 1.0$$
$$\Pr(s' = S_2 | s = S_3, a = A_1) = 1.0$$
$$R(s = S_1, a = A_2, s' = S_2) = 1.0$$
$$R(s = S_2, a = A_1, s' = S_2) = 1.0$$
$$R(s = S_3, a = A_1, s' = S_3) = 1.0$$

The environment is illustrated in Figure D-1. For the power of adversary, we allow ν

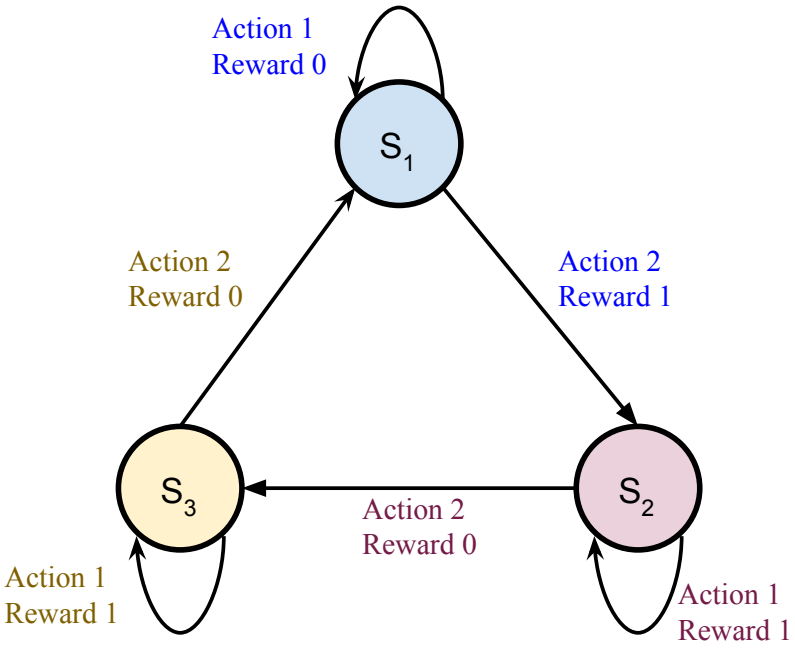


Figure D-1: A simple 3-state environment.

to perturb one state to any other two neighbouring states:

$$B_\nu(S_1) = B_\nu(S_2) = B_\nu(S_3) = \{S_1, S_2, S_3\}$$

Now we evaluate various policies for MDP and SA-MDP for this environment. We use $\gamma = 0.99$ as the discount factor. A stationary and Markovian policy in this environment can be described by 3 parameters p_{11}, p_{21}, p_{31} where $p_{ij} \in [0, 1]$ denotes the probability $\Pr(a = A_j | s = S_i)$. We denote the value function as V for MDP and $\tilde{V}$ for SA-MDP.

- **Optimal Policy for MDP.** For a regular MDP, the optimal solution is $p_{11} = 0$, $p_{21} = 1$, $p_{31} = 1$. We take A_2 to receive reward and leave S_1, and then keep doing A_1 in S_2 and S_3. The values for each state are $V(S_1) = V(S_2) = V(S_3) = \frac{1}{1-\gamma} = 100$, which is optimal. However, this policy obtains $\tilde{V}(S_1) = \tilde{V}(S_2) = \tilde{V}(S_3) = 0$ for SA-MDP, because we can set $\nu(S_1) = S_2$, $\nu(S_2) = S_1$, $\nu(S_3) = S_1$ and consequentially we always take the wrong action receiving 0 reward.

- **A Stochastic Policy for MDP and SA-MDP.** We consider a stochastic policy where $p_{11} = p_{21} = p_{31} = 0.5$. Under this policy, we randomly stay or

move in each state, and has a 50% probability of receiving a reward. The adversary ν has no power because π is the same for all states. In this situation, $V(S_1) = \tilde{V}(S_1) = V(S_2) = \tilde{V}(S_2) = V(S_3) = \tilde{V}(S_3) = \frac{0.5}{1-0.99} = 50$ for both MDP and SA-MDP.

- **Deterministic Policies for SA-MDP.** Now we consider all $2^3 = 8$ possible deterministic policies for SA-MDP. Note that if for any state S_i we have $p_{i1} = 0$ and another state S_j we have $p_{j1} = 1$, we always have $\tilde{V}(S_1) = \tilde{V}(S_2) = \tilde{V}(S_3) = 0$. This is because we can set $\nu(S_1) = S_j$, $\nu(S_2) = S_i$ and $\nu(S_3) = S_i$ and always receive a 0 reward. Thus the only two possible other policies are $p_{11} = p_{21} = p_{31} = 0$ and $p_{11} = p_{21} = p_{31} = 1$, respectively. For $p_{11} = p_{21} = p_{31} = 1$ we have $\tilde{V}(S_1) = 0, \tilde{V}(S_2) = \tilde{V}(S_3) = 100$ as we always take A_1 and never transit to other states; for $p_{11} = p_{21} = p_{31} = 0$, we circulate through all three states and only receive a reward when we leave A_1. We have $\tilde{V}(S_1) = \frac{1}{1-\gamma^3} \approx 33.67$, $\tilde{V}(S_2) = \frac{\gamma^2}{1-\gamma^3} \approx 33.00$ and $\tilde{V}(S_3) = \frac{\gamma}{1-\gamma^3} \approx 33.33$.

Figures D-2, D-3, D-4 give the graph of $\tilde{V}(S_1)$, $\tilde{V}(S_2)$ and $\tilde{V}(S_3)$ under three different settings of p_{11}. The figures are generated using Algorithm 7.

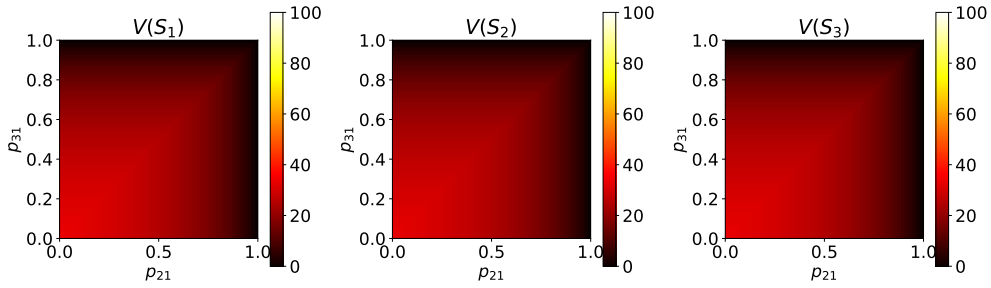


Figure D-2: Value functions for SA-MDP when $p_{11} = 0$, with $p_{21} \in [0, 1]$, $p_{31} \in [0, 1]$

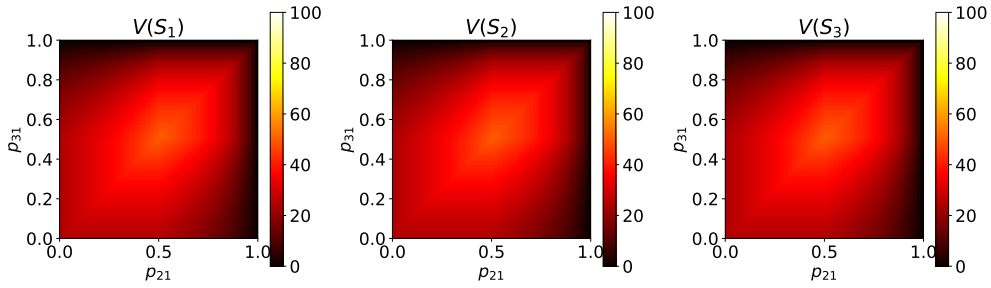


Figure D-3: Value functions for SA-MDP when $p_{11} = 0.5$, with different $p_{21} \in [0, 1]$, $p_{31} \in [0, 1]$

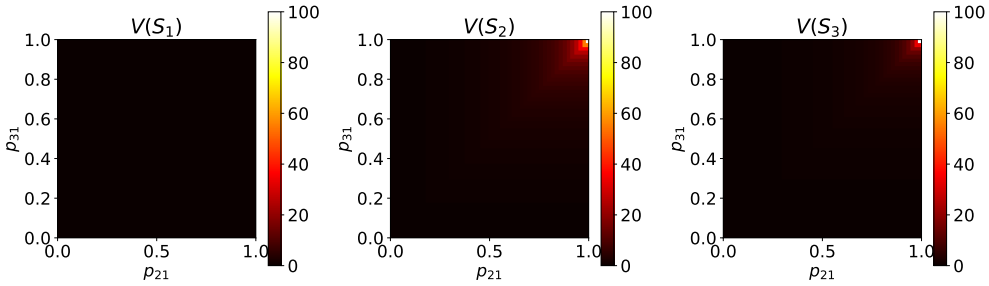


Figure D-4: Value functions for SA-MDP when $p_{11} = 1.0$, with different $p_{21} \in [0, 1]$, $p_{31} \in [0, 1]$

D.2 Proofs for State-Adversarial Markov Decision Process

Theorem 3 (Bellman equations for fixed π and ν)**.** *Given $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ and $\nu : \mathcal{S} \to \mathcal{S}$, we have*

$$\tilde{V}_{\pi\circ\nu}(s) = \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\pi\circ\nu}(s')\right]$$
$$\tilde{Q}_{\pi\circ\nu}(s,a) = \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|\nu(s')) \tilde{Q}_{\pi\circ\nu}(s',a')\right].$$

Proof. Based on the definition of $\tilde{V}_{\pi\circ\nu}(s)$:

$$\begin{aligned}
\tilde{V}_{\pi\circ\nu}(s) &= \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right] \\
&= \mathbb{E}_{\pi\circ\nu}\left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[r_{t+1} + \gamma \mathbb{E}_{\pi\circ\nu}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s'\right]\right] \\
&= \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \sum_{s'\in\mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\pi\circ\nu}(s')\right]
\end{aligned} \tag{D.1}$$

The recursion for $\tilde{Q}_{\pi\circ\nu}(s,a)$ can be derived similarly. Additionally, we note the following useful relationship between $\tilde{V}_{\pi\circ\nu}(s)$ and $\tilde{Q}_{\pi\circ\nu}(s,a)$:

$$\tilde{V}_{\pi\circ\nu}(s) = \sum_{a\in\mathcal{A}} \pi(a|\nu(s)) \tilde{Q}_{\pi\circ\nu}(s,a) \tag{D.2}$$

□

Before starting to prove Theorem 4, first we show that finding the optimal adversary ν^* given a fixed π for a SA-MDP can be cast into the problem of finding an optimal policy in a regular MDP.

Lemma 3 (Equivalence of finding optimal adversary in SA-MDP and finding optimal policy in MDP)**.** *Given an SA-MDP $M = (\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$ and a fixed policy π, there exists a MDP $\hat{M} = (\mathcal{S}, \hat{\mathcal{A}}, \hat{R}, \hat{p}, \gamma)$ such that the optimal policy of $\hat{M}$ is the optimal adversary ν for SA-MDP given the fixed π.*

Proof. For an SA-MDP $M = (\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$ and a fixed policy π, we define a regular MDP $\hat{M} = (\mathcal{S}, \hat{\mathcal{A}}, \hat{R}, \hat{p}, \gamma)$ such that $\hat{\mathcal{A}} = \mathcal{S}$, and ν is the policy for $\hat{M}$. At each state s, our policy ν gives a probability distribution $\nu(\cdot|s) \in \mathcal{P}(\hat{\mathcal{A}}) = \mathcal{P}(\mathcal{S})$ indicating that we perturb a state s to $\hat{s}$ with probability $\nu(\hat{s}|s)$ in the SA-MDP M.

For $\hat{M}$, the reward function is defined as:

$$\hat{R}(s, \hat{a}, s') = \begin{cases} -\frac{\sum_{a \in \mathcal{A}} \pi(a|\hat{a}) p(s'|s,a) R(s,a,s')}{\sum_{a \in \mathcal{A}} \pi(a|\hat{a}) p(s'|s,a)} & \text{for } s, s' \in \mathcal{S} \text{ and } \hat{a} \in B(s) \subset \hat{\mathcal{A}} = \mathcal{S}, \\ C & \text{for } s, s' \in \mathcal{S} \text{ and } \hat{a} \notin B(s). \end{cases} \tag{D.3}$$

The transition probability $\hat{p}$ is defined as

$$\hat{p}(s'|s, \hat{a}) = \sum_{a \in \mathcal{A}} \pi(a|\hat{a}) p(s'|s, a) \quad \text{for } s, s' \in \mathcal{S} \text{ and } \hat{a} \in \hat{\mathcal{A}} = \mathcal{S}.$$

For the case of $\hat{a} \in B(s)$, the above reward function definition is based on the intuition that when the agent receives a reward r at a time step given s, a, s', the adversary's reward is $\hat{r} = -r$. Note that we consider r as a random variable given s, a, s'. To give the distribution of rewards for adversary $p(\hat{r}|s, \hat{a}, s')$, we follow the conditional

probability which marginalizes π:

$$\begin{aligned} p(\hat{r}|s,\hat{a},s') &= \frac{p(\hat{r},s'|s,\hat{a})}{p(s'|s,\hat{a})} \\ &= \frac{\sum_a p(\hat{r},s'|a,s,\hat{a})\pi(a|s,\hat{a})}{\sum_a p(s'|a,s,\hat{a})\pi(a|s,\hat{a})} \\ &= \frac{\sum_a p(\hat{r},s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\ &= \frac{\sum_a p(\hat{r}|s',a,s)p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \end{aligned} \tag{D.4}$$

Considering that $R(s,a,s') := \mathbb{E}[r|s',a,s] = -\mathbb{E}[\hat{r}|s',a,s]$, and taking an expectation in Eq. (D.4) over $\hat{r}$ yield the first case in (D.3):

$$\begin{aligned} \hat{R}(s,\hat{a},s') &\coloneqq \mathbb{E}[\hat{r}|s,\hat{a},s'] \\ &= \sum_{\hat{r}} \hat{r}\frac{\sum_a p(\hat{r}|s',a,s)p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\ &= \frac{\sum_a \left[\sum_{\hat{r}} \hat{r}p(\hat{r}|s',a,s)\right] p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\ &= \frac{\sum_a \mathbb{E}[\hat{r}|s',a,s]p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \\ &= -\frac{\sum_a R(s,a,s')p(s'|a,s)\pi(a|\hat{a})}{\sum_a p(s'|a,s)\pi(a|\hat{a})} \end{aligned} \tag{D.5}$$

The reward for adversary's actions outside $B(s)$ is a constant C such that

$$C < \min\Big\{ -\overline{M}, \quad \frac{\gamma}{(1-\gamma)}\underline{M} - \frac{1}{(1-\gamma)}\overline{M}\Big\},$$

where $\underline{M} := \min_{s,a,s'} R(s,a,s')$ and $\overline{M} := \max_{s,a,s'} R(s,a,s')$. We have for $\forall(s,\hat{a},s')$,

$$C < \hat{R}(s,\hat{a},s') \leq -\underline{M},$$

and for $\forall \hat{a} \in B(s)$, according to Eq. (D.5),

$$-\overline{M} \leq \hat{R}(s,\hat{a},s') \leq -\underline{M}.$$

According basic properties of MDP [118, 145], we know that the $\hat{M}$ has an optimal policy ν^*, which satisfies $\hat{V}_{\pi\circ\nu^*}(s) \geq \hat{V}_{\pi\circ\nu}(s)$ for $\forall s$, $\forall \nu$. We also know that this ν^* is deterministic and assigns a unit mass probability for the optimal action for each s.

We define $\mathfrak{N} := \{\nu : \ \forall s, \ \exists \hat{a} \in B(s), \ \nu(\hat{a}|s) = 1\}$ which restricts the adversary from taking an action not in $B(s)$, and claim that $\nu^* \in \mathfrak{N}$. If this is not true for a state s^0, we have

$$\begin{aligned}
\hat{V}_{\pi\circ\nu^*}(s^0) &= \mathbb{E}_{\hat{p},\nu^*}\Big[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s^0\Big] \\
&= C + \mathbb{E}_{\hat{p},\nu^*}\left[\sum_{k=1}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s^0\right] \\
&\leq C - \frac{\gamma}{1-\gamma}\underline{M} \\
&< -\frac{1}{1-\gamma}\overline{M} \\
&\leq \mathbb{E}_{\hat{p},\nu'}\Big[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s^0\Big] = \hat{V}_{\pi\circ\nu'}(s^0),
\end{aligned}$$

where the second equality holds because ν^* is deterministic, and the last inequality holds for any $\nu' \in \mathfrak{N}$. This contradicts the assumption that ν^* is optimal. So from now on in this proof we only study policies in $\mathfrak{N}$.

For any policy $\nu \in \mathfrak{N}$:

$$\begin{aligned}
\hat{V}_{\pi\circ\nu}(s) &= \mathbb{E}_{\hat{p},\nu}\Big[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+1}|s_t = s\Big] = \mathbb{E}_{\hat{p},\nu}\left[\hat{r}_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+2}|s_t = s\right] \\
&= \sum_{\hat{a}\in\mathcal{S}}\nu(\hat{a}|s)\sum_{s'\in\mathcal{S}}\hat{p}(s'|s,\hat{a})\left[\hat{R}(s,\hat{a},s') + \gamma\mathbb{E}_{\hat{p},\nu}\left[\sum_{k=0}^{\infty}\gamma^k\hat{r}_{t+k+2}|s_{t+1} = s'\right]\right] \\
&= \sum_{\hat{a}\in\mathcal{S}}\nu(\hat{a}|s)\sum_{s'\in\mathcal{S}}\hat{p}(s'|s,\hat{a})\left[\hat{R}(s,\hat{a},s') + \gamma\hat{V}_{\pi\circ\nu}(s')\right] \qquad \text{(D.6)}
\end{aligned}$$

Note that all policies in $\mathfrak{N}$ are deterministic and this class of policies consists ν^*. Also, $\mathfrak{N}$ is consistent with the class of policies studied in Theorem 3. We denote the

deterministic action $\hat{a}$ chosen by a $\nu \in \mathfrak{N}$ at s as $\nu(s)$. Then for $\forall \nu \in \mathfrak{N}$, we have

$$\begin{aligned}
\hat{V}_{\pi \circ \nu}(s) &= \sum_{s' \in \mathcal{S}} \hat{p}(s'|s, \nu(s)) \left[\hat{R}(s, \hat{a}, s') + \gamma \hat{V}_{\pi \circ \nu}(s') \right] \\
&= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \pi(a|\hat{a}) p(s'|s, a) \left[- \frac{\sum_{a \in \mathcal{A}} \pi(a|\hat{a}) p(s'|s, a) R(s, a, s')}{\sum_{a \in \mathcal{A}} \pi(a|\hat{a}) p(s'|s, a)} + \gamma \hat{V}_{\pi \circ \nu}(s') \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a|\nu(s)) \sum_{s' \in \mathcal{S}} p(s'|s, a) \left[-R(s, a, s') + \gamma \hat{V}_{\pi \circ \nu}(s') \right], \qquad \text{(D.7)}
\end{aligned}$$

or

$$-\hat{V}_{\pi \circ \nu}(s) = \sum_{a \in \mathcal{A}} \pi(a|\nu(s)) \sum_{s' \in \mathcal{S}} p(s'|s, a) \left[R(s, a, s') + \gamma(-\hat{V}_{\pi \circ \nu}(s')) \right]. \qquad \text{(D.8)}$$

Comparing (D.8) and (D.1), we know that $-\hat{V}_{\pi \circ \nu} = \tilde{V}_{\pi \circ \nu}$ for any $\nu \in \mathfrak{N}$. The optimal value function $\hat{V}_{\pi \circ \nu^*}$ satisfies:

$$\begin{aligned}
\hat{V}_{\pi \circ \nu^*}(s) &= \max_{\hat{a} \in B(s)} \sum_{s' \in \mathcal{S}} \hat{p}(s'|s, \hat{a}) \left[\hat{R}(s, \hat{a}, s') + \gamma \hat{V}_{\pi \circ \nu}(s') \right] \\
&= \max_{s_\nu \in B(s)} \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s, a) \left[-R(s, a, s') + \gamma \hat{V}_{\pi \circ \nu^*}(s') \right], \qquad \text{(D.9)}
\end{aligned}$$

where we denote the action $\hat{a}$ taken at s as s_ν. So for ν^*, since $-\hat{V}_{\pi \circ \nu^*} = \tilde{V}_{\pi \circ \nu^*}$, we have

$$\tilde{V}_{\pi \circ \nu^*}(s) = \min_{\hat{a} \in B(s)} \sum_{a \in \mathcal{A}} \pi(a|\hat{a}) \sum_{s' \in \mathcal{S}} p(s'|s, a) \left[R(s, a, s') + \gamma \tilde{V}_{\pi \circ \nu^*}(s') \right], \qquad \text{(D.10)}$$

and $\tilde{V}_{\pi \circ \nu^*}(s) \leq \tilde{V}_{\pi \circ \nu}(s)$ for $\forall s$, $\forall \nu \in \mathfrak{N}$. Hence ν^* is also the optimal ν for $\tilde{V}_{\pi \circ \nu}$. $\square$

Lemma 3 gives many good properties for the optimal adversary. First, an optimal adversary always exists under the regularity conditions where an optimal policy exists for a MDP. Second, we do not need to consider stochastic adversaries as there always exists an optimal deterministic adversary. Additionally, showing Bellman contraction for finding the optimal adversary can be done similarly as in obtaining the optimal policy in a regular MDP, as shown in the proof of Theorem 4.

Theorem 4 (Bellman contraction for optimal adversary)**.** *Define Bellman operator* $\mathscr{L} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$,

$$(\mathscr{L}\tilde{V})(s) = \min_{s_\nu \in B(s)} \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}(s') \right]. \tag{D.11}$$

The Bellman equation for optimal adversary ν^* *can then be written as:* $\tilde{V}_{\pi \circ \nu^*} = \mathscr{L}\tilde{V}_{\pi \circ \nu^*}$. *Additionally,* $\mathscr{L}$ *is a contraction that converges to* $\tilde{V}_{\pi \circ \nu^*}$.

Proof. Based on Lemma 3, this proof is technically similar to the proof of "optimal Bellman equation" in regular MDPs, where max over π is replaced by min over ν. By the definition of $\tilde{V}_{\pi \circ \nu^*}(s)$,

$$\begin{aligned}
\tilde{V}_{\pi \circ \nu^*}(s) &= \min_\nu \tilde{V}_{\pi \circ \nu}(s) = \min_\nu \mathbb{E}_{\pi \circ \nu} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] \\
&= \min_\nu \mathbb{E}_{\pi \circ \nu} \left[r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_t = s \right] \\
&= \min_\nu \sum_{a \in \mathcal{A}} \pi(a|\nu(s)) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r_{t+1} + \gamma \mathbb{E}_{\pi \circ \nu} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right] \right] \\
&= \min_{s_\nu \in B_\nu(s)} \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r_{t+1} + \gamma \min_\nu \mathbb{E}_{\pi \circ \nu} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+2} | s_{t+1} = s' \right] \right] \\
&= \min_{s_\nu \in B_\nu(s)} \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r_{t+1} + \gamma \tilde{V}_{\pi \circ \nu^*}(s') \right]
\end{aligned}$$

This is the Bellman equation for the optimal adversary ν^*; ν^* is a fixed point of the Bellman operator $\mathscr{L}$. Now we show the Bellman operator is a contraction. We have, if $\mathscr{L}\tilde{V}_{\pi \circ \nu_1}(s) \geq \mathscr{L}\tilde{V}_{\pi \circ \nu_2}(s)$,

$$\begin{aligned}
\mathscr{L}\tilde{V}_{\pi \circ \nu_1}(s) - \mathscr{L}\tilde{V}_{\pi \circ \nu_2}(s) &\leq \max_{s_\nu \in B_\nu(s)} \Bigg\{ \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\pi \circ \nu_1}(s') \right] \\
&\quad - \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[R(s,a,s') + \gamma \tilde{V}_{\pi \circ \nu_2}(s') \right] \Bigg\} \\
&= \gamma \max_{s_\nu \in B_\nu(s)} \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) [\tilde{V}_{\pi \circ \nu_1}(s') - \tilde{V}_{\pi \circ \nu_2}(s')] \\
&\leq \gamma \max_{s_\nu \in B_\nu(s)} \sum_{a \in \mathcal{A}} \pi(a|s_\nu) \sum_{s' \in \mathcal{S}} p(s'|s,a) \|\tilde{V}_{\pi \circ \nu_1} - \tilde{V}_{\pi \circ \nu_2}\|_\infty \\
&= \gamma \|\tilde{V}_{\pi \circ \nu_1} - \tilde{V}_{\pi \circ \nu_2}\|_\infty
\end{aligned}$$

The first inequality comes from the fact that

$$\min_{x_1} f(x_1) - \min_{x_2} g(x_2) \leq f(x_2^*) - g(x_2^*) \leq \max_x (f(x) - g(x)),$$

where $x_2^* = \text{argmin}_{x_2} g(x_2)$. Similarly, we can prove $\mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) \leq \|\tilde{V}_{\pi\circ\nu_1} - \tilde{V}_{\pi\circ\nu_2}\|_\infty$ if $\mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s) > \mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s)$. Hence

$$\|\mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s)\|_\infty = \max_s |\mathscr{L}\tilde{V}_{\pi\circ\nu_1}(s) - \mathscr{L}\tilde{V}_{\pi\circ\nu_2}(s)| \leq \gamma \|\tilde{V}_{\pi\circ\nu_1} - \tilde{V}_{\pi\circ\nu_2}\|_\infty.$$

Then according to the Banach fixed-point theorem, since $0 < \gamma < 1$, $\tilde{V}_{\pi\circ\nu}$ converges to a unique fixed point, and this fixed point is $\tilde{V}_{\pi\circ\nu^*}$.

□

Algorithm 7 Policy Evaluation for an SA-MDP $(\mathcal{S}, \mathcal{A}, B, R, p, \gamma)$

Input: Policy π, convergence threshold ε
Output: Values for policy π, detnoted as $\tilde{V}_{\pi\circ\nu^*}(s)$

```
Initialize array V(s) ← 0 for all s ∈ S
repeat
  Δ ← 0
  for all s ∈ S do
    v ← ∞, v_0 ← V(s)
    for all s_ν ∈ B(s) do
      v' ← Σ_{a∈A} π(a|s_ν) Σ_{s'∈S} p(s'|s,a) · [R(s,a,s') + γV(s')]
      v ← min(v, v')
    end for
    V(s) ← v
    Δ ← max(Δ, |v_0 − V(s)|)
  end for
until Δ < ε
Ṽ_{π∘ν*}(s) ← V(s)
```

A direct consequence of Theorem 4 is the policy evaluation algorithm (Algorithm 7) for SA-MDP, which obtains the values for each state under *optimal* adversary for a fixed policy π. For both Lemma 3 and Theorem 4, we only consider a fixed policy π, and in this setting finding an optimal adversary is not difficult. However, finding an optimal π under the optimal adversary is more challenging, as we can see in

Section D.1, given the white-box attack setting where the adversary knows π and can choose optimal perturbations accordingly, an optimal policy for MDP can only receive zero rewards under optimal adversary. We now show two intriguing properties for optimal policies in SA-MDP:

Theorem 5. *There exists an SA-MDP and some stochastic policy $\pi \in \Pi_{MR}$ such that we cannot find a better deterministic policy $\pi' \in \Pi_{MD}$ satisfying $\tilde{V}_{\pi' \circ \nu^*(\pi')}(s) \geq \tilde{V}_{\pi \circ \nu^*(\pi)}(s)$ for all $s \in \mathcal{S}$.*

Proof. Proof by giving a counter example that no deterministic policy can be better than a random policy. The SA-MDP example in section D.1 provided such a counter example: all 8 possible deterministic policies are no better than the stochastic policy $p_{11} = p_{21} = p_{31} = 0.5$. □

Theorem 6. *Under the optimal ν^*, an optimal policy $\pi^* \in \Pi_{MR}$ does not always exist for SA-MDP.*

Proof. We will show that the SA-MDP example in section D.1 does not have an optimal policy. First, for π_1 where $p_{11} = p_{21} = p_{31} = 1$ we have $\tilde{V}_{\pi_1 \circ \nu^*(\pi_1)}(S_1) = 0, \tilde{V}_{\pi_1 \circ \nu^*(\pi_1)}(S_2) = \tilde{V}_{\pi_1 \circ \nu^*(\pi_1)}(S_3) = 100$. This policy is not an optimal policy since we have π_2 where $p_{11} = p_{21} = p_{31} = 0.5$ that can achieve $\tilde{V}_{\pi_2 \circ \nu^*(\pi_2)}(S_1) = \tilde{V}_{\pi_2 \circ \nu^*(\pi_2)}(S_2) = \tilde{V}_{\pi_2 \circ \nu^*(\pi_2)}(S_3) = 50$ and $\tilde{V}_{\pi_2 \circ \nu^*(\pi_2)}(S_1) > \tilde{V}_{\pi_1 \circ \nu^*(\pi_1)}(S_1)$.

An optimal policy π, if exists, must be better than π_1 and have $\tilde{V}_{\pi \circ \nu^*(\pi)}(S_1) > 0, V_{\pi \circ \nu^*(\pi)}(S_2) = V_{\pi \circ \nu^*(\pi)}(S_3) = 100$. In order to achieve $V_{\pi \circ \nu^*(\pi)}(S_2) = V_{\pi \circ \nu^*(\pi)}(S_3) = 100$, we must set $p_{21} = p_{31} = 1$ since it is the only possible way to start from S_2 and S_3 and receive $+1$ reward for every step. We can still change p_{11} to probabilities other than 1, however if $p_{11} < 1$ the adversary can set $\nu(S_2) = \nu(S_3) = S_1$ and reduce $V_{\pi \circ \nu^*(\pi)}(S_2)$ and $V_{\pi \circ \nu^*(\pi)}(S_3)$. Thus, no policy better than π_1 exists, and since π_1 is not an optimal policy, no optimal policy exists. □

Theorem 5 and Theorem 6 show that the classic definition of optimality is probably not suitable for SA-MDP. Further works can study how to obtain optimal policies for SA-MDP under some alternative definition of optimality, or using a more complex policy class (e.g., history dependent policies).

Theorem 7. *Given a policy π for a non-adversarial MDP and its value function is $V_\pi(s)$. Under the optimal adversary ν in SA-MDP, for all $s \in \mathcal{S}$ we have*

$$\max_{s\in\mathcal{S}} \left\{ V_\pi(s) - \tilde{V}_{\pi\circ\nu^*(\pi)}(s) \right\} \leq \alpha \max_{s\in\mathcal{S}} \max_{\hat{s}\in B(s)} \mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s})) \tag{D.12}$$

where $\mathrm{D}_{\mathrm{TV}}(\pi(\cdot|s), \pi(\cdot|\hat{s}))$ is the total variation distance between $\pi(\cdot|s)$ and $\pi(\cdot|\hat{s})$.

On the RHS,

$$\alpha := 2[1 + \frac{\gamma}{(1-\gamma)^2}] \max_{(s,a,s')\in\mathcal{S}\times\mathcal{A}\times\mathcal{S}} |R(s,a,s')|$$

is a constant that does not depend on π.

Proof. Our proof of Theorem 7 is based on Theorem 1 proposed in [1]. In fact, many existing works in the literature have proved similar results under different assumptions [74, 117].

For an arbitrary starting state s_0 and two arbitrary policies π and π', Theorem 1 in [1] gives an upper bound of $V_\pi(s_0) - V_{\pi'}(s_0)$. The bound is given by

$$\begin{aligned} V_\pi(s_0) - V_{\pi'}(s_0) \leq & -\mathbb{E}_{\substack{s\sim d^\pi_{s_0} \\ a\sim\pi(\cdot|s) \\ s'\sim p(\cdot|a,s)}} \Big[\Big(\frac{\pi'(a|s)}{\pi(a|s)} - 1 \Big) R(s,a,s') \Big] \\ & + \frac{2\gamma}{(1-\gamma)^2} \max_s \Big\{ \mathbb{E}_{\substack{a\sim\pi'(\cdot|s) \\ s'\sim p(\cdot|a,s)}} \Big[R(s,a,s') \Big] \Big\} \mathbb{E}_{s\sim d^\pi_{s_0}} \Big[\mathrm{D}_{TV}(\pi(\cdot|s), \pi'(\cdot|s)) \Big], \end{aligned} \tag{D.13}$$

where $d^\pi_{s_0}$ is the discounted future state distribution from s_0, defined as

$$d^\pi_{s_0}(s) := (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \mathrm{Pr}(s_t = s|\pi, s_0). \tag{D.14}$$

Note that in Theorem 1 of [1], the author proved a general form with an arbitrary function f and we assume $f \equiv 0$ in our proof. We also assume the starting state is deterministic, so J^π in [1] is replaced by $V^\pi(s_0)$.

Then we simply need to bound both terms on the right hand side of (D.13). For

the first term we know that

$$
\begin{aligned}
-\mathbb{E}_{\substack{s\sim d^{\pi}_{s_0}\\ a\sim\pi(\cdot|s)\\ s'\sim p(\cdot|a,s)}}\Big[\Big(\frac{\pi'(a|s)}{\pi(a|s)}-1\Big)R(s,a,s')\Big] &= \sum_s d^{\pi}_{s_0}(s)\sum_a\Big[\pi(a|s)-\pi'(a|s)\Big]\sum_{s'}p(s'|s,a)R(s,a,s') \\
&\le \sum_s d^{\pi}_{s_0}(s)\sum_a\Big|\pi(a|s)-\pi'(a|s)\Big|\Big|\sum_{s'}p(s'|s,a)R(s,a,s')\Big| \\
&\le \max_{s,a,s'}|R(s,a,s')|\max_s\Big\{\sum_a\Big|\pi(a|s)-\pi'(a|s)\Big|\Big\} \\
&= 2\max_{s,a,s'}|R(s,a,s')|\max_s \mathrm{D}_{TV}(\pi(\cdot|s),\pi'(\cdot|s))
\end{aligned}
\tag{D.15}
$$

The second term is bounded by

$$
\begin{aligned}
&\frac{2\gamma}{(1-\gamma)^2}\max_s\Big\{\mathbb{E}_{\substack{a\sim\pi'(\cdot|s)\\ s'\sim p(\cdot|a,s)}}\Big[R(s,a,s')\Big]\Big\}\mathbb{E}_{s\sim d^{\pi}_{s_0}}\Big[\mathrm{D}_{TV}(\pi(\cdot|s),\pi'(\cdot|s))\Big] \\
&\qquad\le \frac{2\gamma}{(1-\gamma)^2}\max_{s,a,s'}|R(s,a,s')|\max_s \mathrm{D}_{TV}(\pi(\cdot|s),\pi'(\cdot|s))
\end{aligned}
\tag{D.16}
$$

Therefore, the RHS of (D.13) is bounded by $\alpha\max_s \mathrm{D}_{TV}(\pi(\cdot|s),\pi'(\cdot|s))$, where

$$
\alpha = 2[1+\frac{\gamma}{(1-\gamma)^2}]\max_{s,a,s'}|R(s,a,s')| \tag{D.17}
$$

Finally, we simply let $\pi'(\cdot|s) := \pi(\cdot|\nu^*(s))$ and the proof is complete. □

D.3 Optimization Techniques

D.3.1 More Backgrounds for Convex Relaxation of Neural Networks

In this work, we frequently need to solve a minimax problem:

$$
\min_{\theta}\max_{\phi\in\mathcal{S}} g(\theta,\phi) \tag{D.18}
$$

One approach we discussed above is to first solve the inner maximization problem (approximately) using an optimizer like SGLD. However, due to the non-convexity of

π_θ, we cannot solve the inner maximization to global maxima, and the gap between local maxima and global maxima can be large. Using convex relaxations of neural networks, we can instead find an upper bound of $\max_{\phi \in \mathcal{S}} g(\theta, \phi)$:

$$\overline{g}(\theta) \geq \max_{\phi \in \mathcal{S}} g(\theta, \phi)$$

Thus we can minimize an upper bound instead, which can guarantee the original objective (D.18) is minimized.

As an illustration on how to find $\overline{g}(\theta)$ using convex relaxations, following [125] we consider a simple L-layer MLP network $f(\theta, x)$ with parameters $\theta = \{(W^{(i)}, b^{(i)}), i \in \{1, \cdots, L\}\}$ and activation function σ. We denote $x^{(0)} = x$ as the input, $x^{(i)}$ as the post-activation value for layer i, $z^{(i)}$ as the pre-activateion value for layer i. $i \in \{1, \cdots, L\}$. The output of the network $f(\theta, x)$ is $z^{(L)}$. Then, we consider the following optimization problem:

$$\max_{x \in \mathcal{S}} f(\theta, x), \quad \text{where } \mathcal{S} \text{ is the set of perturbations}$$

which is equivalent to the following optimization problem:

$$\begin{aligned} \max \quad & z^{(L)} \\ \text{s.t.} \quad & z^{(l)} = W^{(l)} x^{(l-1)} + b^{(l)}, l \in [L], \\ & x^{(l)} = \sigma(z^{(l)}), l \in [L-1], \\ & x^{(0)} \in \mathcal{S} \end{aligned} \tag{D.19}$$

In this constrained optimization problem (D.19), assuming $\mathcal{S}$ is a convex set, the constraint on $z^{(l)}$ is convex (linear) and the only non-convex constraints are those for $x^{(l)}$, where a non-linear activation function is involved. Note that activation function $\sigma(z)$ itself can be a convex function, but when used as an equality constraint, the feasible solution is constrained to the *graph* of $\sigma(z)$, which is non-convex.

Previous works [162, 182, 125] propose to use convex relaxations of non-linear units to relax the non-convex constraint $x^{(l)} = \sigma(z^{(l)})$ with a convex one, $x^{(l)} =$

convex($\sigma(z^{(l)})$), such that (D.19) can be solved efficiently. We can then obtain an *upper bound* of $f(\theta, x)$ since the constraints are relaxed. Several concrete examples (e.g., ReLU, tanh, sigmoid) on how these relaxations are formed were given in [182]. In the special case where linear relaxations are used, (D.19) can be solved efficiently and automatically (without manual derivation and implementation) for general computational graphs [171]. Generally, using the framework from [171] we can access an oracle function ConvexRelaxUB defined as below:

Definition 3. *Given a neural network function $f(\mathbf{X})$ where $\mathbf{X}$ is any input for this function, and $\mathbf{X} \in \mathbb{S}$ where $\mathbb{S}$ is the set of perturbations, the oracle function* ConvexRelaxUB *provided by an automatic neural network convex relaxation tool returns an upper bound $\overline{f}$, which satisfies:*

$$\overline{f} \geq \max_{\mathbf{X} \in \mathbb{S}} f(\mathbf{X})$$

Note that in the above definition, $\mathbf{X}$ can by *any* input for this computation (e.g., $\mathbf{X}$ can be s, a, or θ for a $Q_\theta(s, a)$ function). In the special case of this work, for simplicity we define the notation ConvexRelaxUB($f, \theta, s \in B(s)$) which returns an upper bound function $\overline{f}(\theta)$ for $\max_{s \in B(s)} f(\theta, s)$. Many kinds of convex relaxation based methods exist [125], where the expensive ones (which gives a tighter upper bound) can be a few magnitudes slower than regular training. The cheapest method is interval bound propagation (IBP), which only incurs twice more costs as forward propagation; however, IBP base training has been reported unstable and hard to reproduce as its bounds are very loose [179, 4]. To avoid potential issues with IBP, in all our environments, we use the IBP+Backward relaxation scheme following [179, 171], which produces considerably tighter bounds, while being only a few times slower than forward propagation (e.g., 3 times slower than forward propagation when loss fusion [171] is implemented). In fact, [171] can use the same relaxation for training downscaled ImageNet dataset on very large vision models. For DRL the policy neural networks are typically small and can be handled well. In this work, we use convex relaxation as a blackbox tool (provided by the `auto_LiRPA` library [171]), and any new

development for improving its efficiency can benefit us.

D.3.2 Solving the Robust Policy Regularizer using SGLD

Stochastic gradient Langevin dynamics (SGLD) [52] can escape saddle points and shallow local optima in non-convex optimization problems [121, 183, 14, 172], and can be used to solve the inner maximization with zero gradient at $\hat{s} = s$. SGLD uses the following update rule to find $\hat{s}^K$ to maximize $\mathcal{R}_s(\hat{s}, \theta_\mu)$:

$$\hat{s}^{k+1} \leftarrow \text{proj}\left(\hat{s}^k - \eta_k \nabla_{\hat{s}^k} \mathcal{R}_s(\hat{s}^k, \theta_\mu) + \sqrt{2\eta_k/\beta_k}\xi\right), \quad \hat{s}^1 = s, \quad k = 1, \cdots, K$$

where η_k is step size, ξ is an i.i.d. standard Gaussian random variable in $\mathbb{R}^{|\mathcal{S}|}$, β_k is an inverse temperature hyperparameter, and $\text{proj}(\cdot)$ projects the update back into $B(s)$. We find that SGLD is sufficient to escape the stationary point at $\hat{s} = s$. However, due to the non-convexity of $\mu_{\theta_\mu}(\hat{s}, \theta_\mu)$, this approach only provides a lower bound $\mathcal{R}_s(\hat{s}^K, \theta_\mu)$ of $\max_{\hat{s} \in B(s)} \mathcal{R}_s(\hat{s}, \theta_\mu)$. Unlike the convex relaxation based approach, minimizing this lower bound does not guarantee to minimize (5.5), as the gap between $\max_{\hat{s} \in B(s)} \mathcal{R}_s(\hat{s}, \theta_\mu)$ and $\mathcal{R}_s(\hat{s}^K, \theta_\mu)$ can be large. In SGLD, we first need to solve the inner maximization problem (such as Eq. (5.5)). The additional time cost depends on the number of SGLD steps. In our experiments for PPO, we find that using 10 steps are sufficient. However, the total training cost does not grow by 10 times, as in many environments the majority of time was spent on environment simulation steps, rather than optimizing a small policy network.

D.4 Additional details for adversarial attacks on state observations

D.4.1 More details on the Critic based attack

In Section 5.3.4 we discuss critic based attack [113] as a baseline. This attack requires a Q function $Q(s, a)$ to find the best perturbed state. In Algorithm 8 we present our

"corrected" critic based attack based on [113]:

Algorithm 8 Critic based attack [113]

Input: A policy function π under attack, a corresponding $Q(s,a)$ network, and a initial state s_0, T is the number of attack steps, η is the step size, $\underline{s}$ and $\overline{s}$ are valid lower and upper range of s.

for $t = 1$ to T **do**

$g_t = \nabla Q_{s_{t-1}}(s_0, \pi(s_{t-1})) = \frac{\partial Q}{\partial \pi}\frac{\partial \pi}{\partial s_{t-1}}$

$g_t \leftarrow \text{proj}(g_t)$ ▷project g_t according to norm constraint of s; for ℓ_∞ norm simply take the sign

$s_t \leftarrow s_{t-1} - \eta g_t$

$s_t \leftarrow \min(\max(s_t, \underline{s}), \overline{s})$

end for

Output: An adversarial state $\hat{s} := s_T$

Note that in Algorithm 4 of [113], they use the gradient $\nabla Q_s(s, \pi(s)) = \frac{\partial Q}{\partial s} + \frac{\partial Q}{\partial \pi}\frac{\partial \pi}{\partial s}$ which essentially attempts to minimize $Q(\hat{s}, \pi(\hat{s}))$, but they then sample randomly along this gradient direction to find the best $\hat{s}$ that minimizes $Q(s_0, \pi(\hat{s}))$. Our corrected formulation directly minimizes $Q(s_0, \pi(\hat{s}))$ using this gradient instead $\nabla Q_s(s_0, \pi(s)) = \frac{\partial Q}{\partial \pi}\frac{\partial \pi}{\partial s}$.

For PPO, since there is no $Q(s,a)$ available during training, we extend [113] to perform attack relying on $V(s)$: we find a state $\hat{s}$ that minimizes $V(\hat{s})$. Unfortunately, it does not match our setting of perturbing state observations; it looks for a state $\hat{s}$ that has the worst value (i.e., taking action $\pi(\hat{s})$ in state $\hat{s}$ is bad), but taking the action $\pi(\hat{s})$ at state s_0 does not necessarily trigger a low reward action, because $V(\hat{s}) = \max_a Q(\hat{s}, a) \neq \max_a Q(s_0, a)$. Thus, in Table 5.1 we can observe that critic based attack typically does not work very well for PPO agents.

D.4.2 More details on the Maximal Action Difference (MAD) attack

We present the full algorithm of MAD in Algorithm 9. It is a relatively simple attack by directly maximizing a KL-divergence using SGLD, yet it usually outperforms random attack and critic attack on some environments (e.g., see Table 5.1).

Algorithm 9 Maximal Action Difference (MAD) Attack (a critic-independent attack)

Input: A policy function π under attack, and a initial state s_0, T is the number of attack steps, η is the step size, β is the (inverse) temperature parameter for SGLD, $\underline{s}$ and $\overline{s}$ are valid lower and upper range of s.
Define loss function $L_{\text{MAD}}(s) = -D_{\text{KL}}(\pi(\cdot|s_0)\|\pi(\cdot|s))$
for $t = 1$ to T **do**
Sample $\xi \sim \mathcal{N}(0, 1)$
$g_t = \nabla L_{\text{MAD}}(s_{t-1}) + \sqrt{\frac{2}{\beta\eta}}\xi$
$g_t \leftarrow \text{proj}(g_t)$ ▷project g_t according to norm constraint of s; for ℓ_∞ norm simply take the sign
$s_t \leftarrow s_{t-1} - \eta g_t$
$s_t \leftarrow \min(\max(s_t, \underline{s}), \overline{s})$
end for
Output: An adversarial state $\hat{s} := s_T$

D.5 Robustness Certificates for Deep Reinforcement Learning

If we use the convex relaxation in Section D.3.1 to train our networks, it can produce robustness certificates for our task. However in some RL tasks the certificates have interpretations different from classification tasks, as discussed in detail below.

Robustness Certificates for DQN. In DQN, the action space is finite, so we have a robustness certificate on the actions taken at each state. More specifically, at each state s, policy π's action is certified if its corresponding Q function satisfies

$$\underset{a}{\text{argmax}}\, Q_\theta(s, a) = \underset{a}{\text{argmax}}\, Q_\theta(\hat{s}, a) = a^*, \text{for all } \hat{s} \in B(s). \tag{D.20}$$

As mentioned in Section 5.3.3 if $u_{Q_\theta, a^*, a} \leq 0$ holds for all $\hat{s} \in B(s)$, we have

$$Q_\theta^-(\hat{s}, a, a^*) := Q_\theta(\hat{s}, a) - Q_\theta(\hat{s}, a^*) \leq 0 \tag{D.21}$$

is guaranteed for all $a \in \mathcal{A}$, which means that the agent's action will not change when the state observation is in $B(s)$. When the agent's action is not changed under adversarial perturbation, its reward and transition at current step will not change in the DQN setting, either.

In some settings, we find that 100% of the actions are guaranteed to be unchanged (e.g., the Pong environment in Table 5.2). In that case, we can in fact also certify the accumulated reward is not changed given the specific initial conditions for testing. However, it can still be challenging to certify that the agent is robust under *any* starting condition. Similarly, in classification problems many existing certified defenses [163, 104, 54, 179] can only practically guarantee robustness on a specific test set (by computing a "verified test error"), rather than on *any* input image.

Robustness Certificates for PPO. In PPO, the action space is continuous, hence it is not possible to certify that actions do not change under adversary. We instead seek for a different type of guarantee, where we can upper bound the change in action given a norm bounded input perturbation:

$$U_s \geq \max_{\hat{s} \in B(s)} \|\pi_{\theta_\pi}(\hat{s}) - \pi_{\theta_\pi}(s)\| \tag{D.22}$$

Given a state s, we can use convex relaxations to compute an upper bound U_s. Generally speaking, if $B(s)$ is small, a robust policy desires to have a small U_s, otherwise it can be possible to find an adversarial state perturbation that greatly changes $\pi_{\theta_\pi}(\hat{s})$ and causes the agent to misbehave. However, giving certificates on accumulative rewards is still challenging, as it requires to bound reward $r(s, a)$ given a fixed state s, and a perturbed and bounded action a (bounded via (D.22)). Since the environment dynamics can be quite complex in practice (except for the simplest environment like InvertedPendulum), it is hard to bound reward changes given a bounded action. We leave this part as a future direction for exploration and we believe the robustness certificates (D.22) can be useful for future works.

D.6 Additional Details for SA-DQN

Algorithm We present the SA-DQN training algorithm in Algorithm 10. The main difference between SA-DQN and DQN is the additional state-adversarial regularizer $\mathcal{R}_{\text{DQN}}(\theta)$, which encourages the network not to change its output under perturbations

on the state observation. We highlighted these changes in Algorithm 10. Note that the use of hinge loss is not required; other loss functions (e.g., cross-entropy loss) may also be used.

Algorithm 10 State-Adversarial Deep Q-Learning (SA-DQN)

Initialize current Q network $Q(s,a)$ with parameters θ.
Initialize target Q network $Q'(s,a)$ with parameters $\theta' \leftarrow \theta$.
Initial replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
  With probability ϵ_t select a random action at a_t, otherwise select $a_t = \text{argmax}_a\, Q_\theta(s_t, a; \theta)$
  Execute action a_t in environment and observe reward r_t and state s_{t+1}
  Store transition $\{s_t, a_t, r_t, s_{t+1}\}$ in $\mathcal{B}$.
  Randomly sample a minibatch of N samples $\{s_i, a_i, r_i, s_i'\}$ from $\mathcal{B}$.
  For all s_i, compute $a_i^* = \text{argmax}_a\, Q_\theta(s_i, a; \theta)$.
  Set $y_i = r_i + \gamma \max_{a'} Q'_{\theta'}(s_i', a'; \theta)$ for non-terminal s_i, and $y_i = r_i$ for terminal s_i.
  Compute TD-loss for each transition: TD-$L(s_i, a_i, s_i'; \theta)$ = Huber$(y_i - Q_\theta(s_i, a_i; \theta))$
  Define $\mathcal{R}_{\text{DQN}}(\theta) := \sum_i \max\Big\{ \max_{\hat{s}_i \in B(s)} \max_{a \neq a_i^*} Q_\theta(\hat{s}_i, a; \theta) - Q_\theta(\hat{s}_i, a_i^*; \theta), -c \Big\}$.
  Option 1: Use projected gradient descent (PGD) to solve $\mathcal{R}_{\text{DQN}}(\theta)$.
    Run PGD to solve: $\hat{s}_i = \text{argmax}_{\hat{s}_i \in B(s_i)} \max_{a \neq a_i^*} Q_\theta(\hat{s}_i, a; \theta) - Q_\theta(\hat{s}_i, a_i^*; \theta)$.
    Compute the sum of hinge loss of each s_i:
    $\overline{\mathcal{R}}_{\text{DQN}}(\theta) = \sum_i \max\{\max_{a \neq a_i^*} Q_\theta(\hat{s}_i, a; \theta) - Q_\theta(\hat{s}_i, a_i^*), -c\}$.
  Option 2: Use convex relaxations of neural networks to solve a surrogate loss of $\mathcal{R}_{\text{DQN}}(\theta)$.
    For all s_i and all $a \neq a_i^*$, obtain upper bounds on $Q_\theta(s, a; \theta) - Q_\theta(s, a_i^*; \theta)$:
    $u_{a_i^*, a}(s_i; \theta) = \text{ConvexRelaxUB}(Q_\theta(s, a; \theta) - Q_\theta(s, a_i^*; \theta), \theta, s \in B(s_i))$
    Compute a surrogate loss for the hinge loss:
    $\overline{\mathcal{R}}_{\text{DQN}}(\theta) = \sum_i \max\Big\{ \max_{a \neq a_i^*} \{u_{a_i^*, a}(s_i)\}, -c \Big\}$
  Perform a gradient descent step to minimize $\frac{1}{N}[\sum_i \text{TD-}L(s_i, a_i, s_i'; \theta) + \kappa_{\text{DQN}} \overline{\mathcal{R}}_{\text{DQN}}(\theta)]$.
  Update Target Network every M steps: $\theta' \leftarrow \theta$.
end for

Hyperparameters for Vanilla DQN training. For Atari games, the deep Q networks have 3 CNN layers followed by 2 fully connected layers (following [160]). The first CNN layer has 32 channels, a kernel size of 8, and stride 4. The second CNN layer has 64 channels, a kernel size of 4, and stride 2. The third CNN layer has 64

channels, a kernel size of 3, and stride 1. The fully connected layers have 512 hidden neurons for both value and advantage heads. We run each environment for 6×10^6 steps without framestack. We set learning rate as 6.25×10^{-5} (following [63]) for Pong, Freeway and RoadRunner; for BankHeist our implementation cannot reliably converge within 6 million steps, so we reduce learning rate to 1×10^{-5}. For all Atari environments, we clip reward to $-1, +1$ (following [106]) and use a replay buffer with a capacity of 2×10^5.

We set discount factor set to 0.99. Prioritized replay buffer sampling is used with $\alpha = 0.5$ and β increased from 0.4 to 1 linearly through the end of training. A batch size of 32 is used in training. Same as in [106], we choose Huber loss as the TD-loss. We update the target network every 2k steps for all environments.

Hyperparameters for SA-DQN training. SA-DQN uses the same network structure and hyperparameters as in DQN training. The total number of SA-DQN training steps in all environments are the same as those in DQN (6 million). We update the target network every 2k steps for all environments except that the target network is updated every 32k steps for RoadRunner's SA-DQN, which improves convergence for our short training schedule of 6 million frames. For the additional state-adversarial regularization parameter κ for robustness, we choose $\kappa \in \{0.005, 0.01, 0.02\}$. For all 4 Atari environments, we train the Q network without regularization for the first 1.5×10^6 steps, then increase ϵ from 0 to the target value in 4×10^6 steps, and then keep training at the target ϵ for the rest 5×10^5 steps.

Training Time As Atari training is expensive, we train DQN and SA-DQN only 6 million frames; the rewards reported in most DQN paper (e.g., [106, 160, 63]) are obtained by training 20 million frames. Thus, the rewards (without attacks) reported maybe lower than some baselines. The training time for vanilla DQN, SA-DQN (SGLD) and SA-DQN (convex) are roughly 15 hours, 40 hours and 50 hours on a single 1080 Ti GPU, respectively. The training time of each environment varies but is very close.

Note that the training time for convex relaxation based method can be further reduced when using an more efficient relaxation. The fastest relaxation is interval bound propagation (IBP), however it is too inaccurate and can make training unstable and hard to tune [179]. We use the tighter IBP+Backward relaxation, and its complexity can be further improved to the same level as IBP with the recently developed loss fusion technique [171], while providing a much better relaxation than IBP. Our work simply uses convex relaxations as a blackbox tool and we leave further improvements on convex relaxation based methods as a future work.

D.7 Additional details for SA-PPO

Algorithm We present the full SA-PPO algorithm in Algorithm 11. Compared to vanilla PPO, we add a robust state-adversarial regularizer which constrains the KL divergence on state perturbations. We highlighted these changes in Algorithm 11. The regularizer $\mathcal{R}_{\text{PPO}}(\theta_\pi)$ can be solved using SGLD or convex relaxations of neural networks. We define the perturbation set $B(s)$ to be an ℓ_p norm ball around state s with radius ϵ: $B_p(s,\epsilon) := \{s' | \|s' - s\|_p \leq \epsilon\}$. We use a ϵ-schedule during training, where the perturbation budget is slowly increasing dduring each epoch t as ϵ_t until reaching ϵ.

Hyperparameters for Regular PPO Training We use the optimal hyperparameters in [43] which were found using a grid search for vanilla PPO. However, we found that their parameters are not optimal for Humanoid and achieves a cumulative reward of only about 2000 after 1×10^7 steps. Thus we redo hyperparameter search on Humanoid and change learning rate for actor to 5×10^{-5} and critic to 1×10^{-5}. This new set of hyperemeters allows us to obtain Humanoid reward about 5000 for vanilla PPO. Note that even under the original, non-optimal set of hyperemeters by [43], our SA-PPO variants still achieve high rewards similarly to those reported in our paper. Our hyperparameter change only significantly improves the performance of vanilla PPO baseline.

Algorithm 11 State-Adversarial Proximal Policy Optimization (SA-PPO)

Input: Number of iterations T, a ϵ schedule ϵ_t

Initialize actor network $\pi(a|s)$ and critic network $V(s)$ with parameter θ_π and θ_V,

for $t = 1$ to T **do**

Run π_{θ_π} to collect a set of trajectories $\mathcal{D} = \{\tau_k\}$ containing $|\mathcal{D}|$ episodes, each τ_k is a trajectory contain $|\tau_k|$ samples, $\tau_k := \{(s_{k,i}, a_{k,i}, r_{k,i}, s_{k,i+1})\}$, $i \in [|\tau_k|]$

Compute cumulative reward $\hat{R}_{k,i}$ for each step i in every episode k using the trajectories and discount factor γ

Update Value function by minimizing the mean-square error:

$$\theta_V \leftarrow \operatorname*{argmin}_{\theta_V} \frac{1}{\sum_k |\tau_k|} \sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \left(V(s_{k,i}) - \hat{R}_{k,i}\right)^2$$

Estimate advantage $\hat{A}_{k,i}$ for each step i in every episode k using generalized advantage estimation (GAE) and value function $V_{\theta_V}(s)$

Define the state-adversarial policy regularier:

$$\mathcal{R}_{\text{PPO}}(\theta_\pi) := \sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \max_{\bar{s}_{k,i} \in B_p(s_{k,i}, \epsilon_t)} \mathrm{D}_{\text{KL}}\left(\pi(a|s_{k,i}) \| \pi(a|\bar{s}_{k,i})\right)$$

Option 1: Solve $\mathcal{R}_{\text{PPO}}(\theta_\pi)$ using SGLD:

find $\hat{s}_{k,i} = \operatorname{argmax}_{\bar{s}_{k,i} \in B_p(s_{k,i}, \epsilon_t)} \mathrm{D}_{\text{KL}}(\pi(a|s_{k,i}) \| \pi(a|\bar{s}_{k,i}))$ using SGLD optimization for all k, i (the objective can be solved in a batch)

set $\overline{\mathcal{R}}_{\text{PPO}}(\theta_\pi) := \sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \mathrm{D}_{\text{KL}}(\pi(a|s_{k,i}) \| \pi(a|\hat{s}_{k,i}))$

Option 2: Solve $\mathcal{R}_{\text{PPO}}(\theta_\pi)$ using convex relaxations:

$\overline{\mathcal{R}}_{\text{PPO}}(\theta_\pi) := \text{ConvexRelaxUB}(\mathcal{R}_{\text{PPO}}, \theta_\pi, \bar{s}_{k,i} \in B_p(s_{k,i}, \epsilon_t))$

Update the policy by minimizing the SA-PPO objective (the minimization is solved using ADAM):

$$\theta_\pi \leftarrow \operatorname*{argmin}_{\theta'_\pi} \frac{1}{\sum_k |\tau_k|} \left[\sum_{\tau_k \in D} \sum_{i=0}^{|\tau_k|} \min\left(r_{\theta'_\pi}(a_{k,i}|s_{k,i}) \hat{A}_{k,i}, g(r_{\theta'_\pi}(a_{k,i}|s_{k,i})) \hat{A}_{k,i} \right) + \kappa_{\text{PPO}} \overline{\mathcal{R}}_{\text{PPO}}(\theta'_\pi) \right]$$

where $r_{\theta'_\pi}(a_{k,i}|s_{k,i}) := \frac{\pi_{\theta'_\pi}(a_{k,i}|s_{k,i})}{\pi_{\theta_\pi}(a_{k,i}|s_{k,i})}$, $g(r) := \text{clip}(r_{\theta'_\pi}(a_{k,i}|s_{k,i}), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}})$

end for

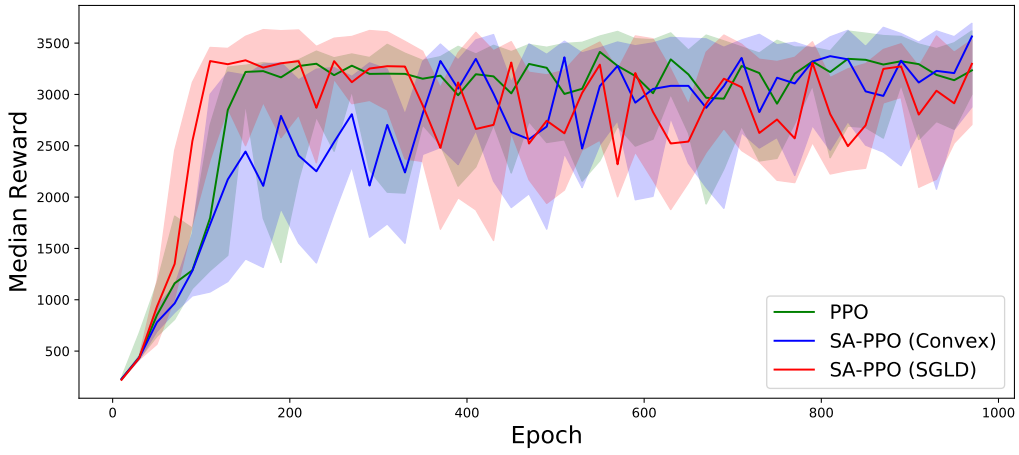
We run 2048 simulation steps per iteration, and run policy optimization of 10 epochs with a minibatch size of 64 using Adam optimizer with learning rate 3×10^{-4}, 4×10^{-4} and 5×10^{-5} for Walker, Hopper and Humanoid, respectively. The value network is also trained in 10 epochs per iteration with a minibatch size of 64, using Adam optimizer with learning rate 0.00025, 3×10^{-4}, and 1×10^{-5} for Walker, Hopper and Humanoid environments, respectively (the same as in [43] without further tuning, except for Humanoid as discussed above). Both networks are 3-layer MLPs with $[64, 64]$ hidden neurons. The clipping value ϵ for PPO is 0.2. We clip rewards to $[-10, 10]$ and states to $[-10, 10]$. The discount factor γ for reward is 0.99 and the discount factor used in generalized advantage estimation (GAE) is 0.95. We found that in [43] the agent rewards are still improving when training finishes, thus in our experiments we run the agents longer for better convergence: we run Walker2d and Hopper 2×10^6 steps (976 iterations) and Humanoid 1×10^7 steps (4882 iterations) to ensure convergence.

Hyperparameter for SA-PPO Training For SA-PPO, we use the same set of hyperparameters as in PPO. Note that the hyperparameters are tuned for PPO but not specifically for SA-PPO. The additional regularization parameter κ_{PPO} for the regularizer $\mathcal{R}_{\text{PPO}}$ is chosen in $\{0.003, 0.01, 0.03, 0.1, 0.3, 1.0\}$. We linearly increase ϵ_t, the norm of ℓ_∞ perturbation on normalized states, from 0 to the target value (ϵ for evaluation, reported in Table 5.1) during the first 3/4 iterations, and keep $\epsilon_t = \epsilon$ for the reset iterations. The same ϵ schedule is used for both SGLD and convex relaxation training. For SGLD, we run 10 iterations with step size $\frac{\epsilon_t}{10}$ and set the temperature parameter $\beta = 1 \times 10^{-5}$. For convex relaxations, we use the efficient IBP+Backward scheme [171], and we use a training schedule similar to [179] by mixing the IBP bounds and backward mode perturbation analysis bounds.

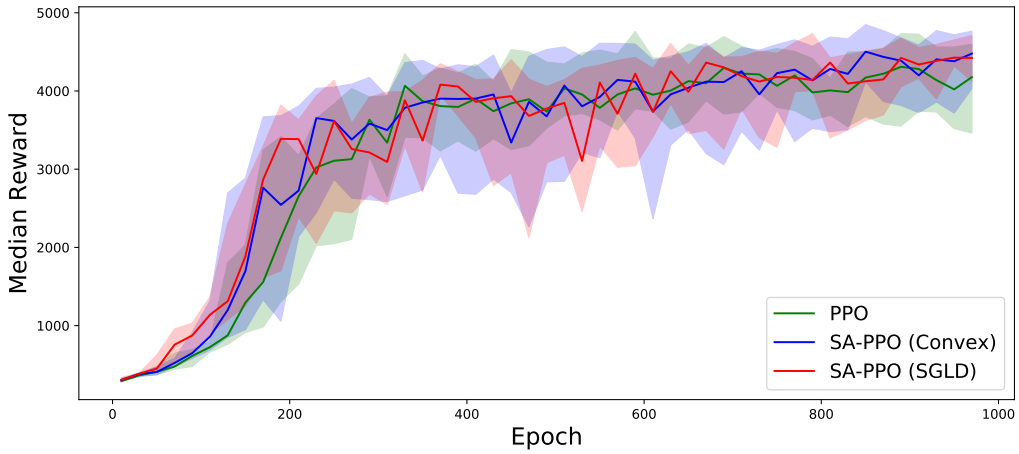
Convergence of PPO and SA-PPO agents Especially, we want to confirm that our significantly better performing Humanoid model is not just by chance. We train each environment using SA-PPO and PPO *at least 30 times*, and collect rewards

during training. We plot the median, 25% and 75% percentile of all these runs in Figure D-5 and here we report the moving average value of 10 consecutive episodes.

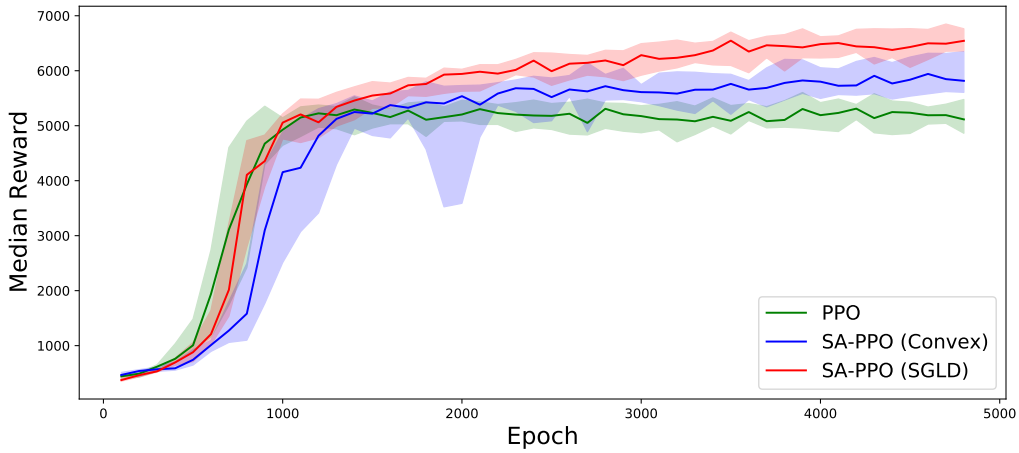
We can see that our SA-PPO models' natural reward during training significantly and consistently outperforms PPO models in Humanoid. Since we also present the 25% and 75% percentile of the rewards among 30 models, we believe this improvement is not because of cherry-picking. For Hopper and Walker environments, SA-PPO achieves slightly lower but still quite competitive rewards; SA-PPO agents obtain significantly more robustness.



(a) Hopper



(b) Walker



(c) Humanoid

Figure D-5: The median, 25% and 75% percentile episode reward of 30 PPO and 30 SA-PPO models during training. We report the moving average value of 10 consecutive episodes. The region of the shaded colors (light blue: SA-PPO solved with SGLD; light green: SA-PPO solved with convex relaxations; light red: vanilla PPO) represent the interval between 25% and 75% percentile rewards over the 30 different training runs, and the solid line is the median rewards over 30 runs.