Towards Robust Image Classification with Deep learning and Real-Time DNN Inference on Mobile

A Dissertation Presented
by

Pu Zhao

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Northeastern University
Boston, Massachusetts

Dec. 2020

NORTHEASTERN UNIVERSITY

Graduate School of Engineering

Dissertation Title: Towards Robust Image Classification with Deep learning and Real-Time DNN Inference on Mobile.

Author: Pu Zhao.

Department: Electrical and Computer Engineering.

Approved for Dissertation Requirements of the Doctor of Philosophy Degree

Dissertation Advisor: Principle Adviser Name	Date
Dissertation Reader: First Reader Name	Date
Dissertation Reader: Second Reader Name	Date
Department Chair: Dr. Shelia Hemami	Date

Director of the Graduate School:

Dean: Dr. Sara Wadia-Fascetti	Date

To my family.

Contents

List of Figures

List of Tables

List of Acronyms

DNN Deep Neural Network.

DL Deep Learning.

CNN Convolutional Neural Network.

ADMM Alternating Direction Method of Multipliers.

MLP Multilayer Perceptron.

NGD Natural Gradient Descent.

ZO Zeroth Order.

ASR Attack Success Rate.

CONV Convolutional.

Acknowledgments

I would like to take this opportunity to thank people who have offered invaluable assistance during the preparation of the dissertation.

My deepest gratitude goes first and foremost to Professor Xue, my supervisor, who has walked me through all the stages of the writing of this thesis. Her critical comments, constant encouragement and guidance have greatly enlightened me not only on the academic pursuit but also on the morals of being a man.

Secondly, I would like to express my heartfelt thanks to all the professors and teachers whose insightful lectures have well prepared me for the completion of the thesis during my pursuit of the PhD at NEU. I also greatly appreciate the assistance offered by the co-authors and scholars in the bibliography.

Last, I am deeply indebted to my beloved parents and friends, who always supported me, willingly discussed with me, and offered valuable insights. Their help and support have accompanied me through the thesis and moments of my life.

Abstract of the Dissertation

Towards Robust Image Classification with Deep learning and Real-Time DNN Inference on Mobile

by

Pu Zhao

Doctor of Philosophy in Electrical and Computer Engineering

Northeastern University, Dec. 2020

Principle Adviser Name, Adviser

As the rapidly increasing popularity of deep learning, deep neural networks (DNN) have become the fundamental and essential building blocks in various applications such as image classification and object detection. However, there are two main issues which potentially limit the wide application of DNNs: 1) the robustness of DNN models raises security concerns, and 2) the large computation and storage requirements of DNN models lead to difficulties for its wide deployment on popular yet resource-constrained devices such as mobile phones. To investigate the DNN robustness, we explore the DNN attack, robustness evaluation and defense. More specifically, for DNN attack, we achieve various attack goals (e.g. adversarial examples and fault sneaking attacks) with different algorithms (e.g. alternating direction method of multipliers (ADMM) and natural gradient descent (NGD) attacks) under various conditions (white-box and black-box attacks). For robustness evaluation, we propose a fast evaluation method to obtain the model perturbation bound such that any model perturbation within the bound does not alter the model classification outputs or incur model mis-behaviors. For the DNN defense, we investigate the defense performance with model connection techniques and successfully mitigate the fault sneaking and backdoor attacks. With a deeper understanding of the DNN robustness, we further explore the deployment problem of DNN models on edge devices with limited resources. To satisfy the storage and computation limitation on edge devices, we adopt model pruning to remove the redundancy in models, thus reducing the storage and computation during inference. Besides, as some applications have real-time requirements with high inference speed sensitivities such as object detection on autonomous cars, we further try to implement real-time DNN inference for various DNN applications on mobile devices with pruning and compiler optimization. To summary, we mainly investigate the DNN robustness and implement real-time DNN inference on the mobile.

Chapter 1

Introduction

Modern technologies based on pattern recognition, machine learning, and specifically deep learning, have achieved significant breakthroughs [86] in a variety of application domains, such as computer vision and natural language processing. Deep neural network (DNN) has become a fundamental element and a core enabler in the ubiquitous artificial intelligence techniques.

Although DNNs have achieved great success in many applications, there are two major concerns limiting wide deployment of DNNs: (1) DNN robustness and (2) DNN efficiency.

1.1 DNN Robustness

Despite the impressive performance of DNNs, many recent studies demonstrate that state-of-the-art DNNs are vulnerable to adversarial attacks [57, 148], which add carefully designed imperceptible distortions to legitimate inputs aiming to mislead the DNNs at test time. Some subsequent works show that other application domains including objection detection [170], speech recognition [4] and deep reinforcement learning [92] also suffer from adversarial examples. This raises concerns of the DNN robustness in many applications with high reliability and dependability requirements such as face recognition, autonomous driving, and malware detection [47, 140].

After the exploration of adversarial attacks in image classification and objection detection from 2014, the vulnerability and robustness of DNNs have attracted ever-increasing attentions and efforts in the research field known as *adversarial machine learning*. Since then, a large amount of efforts have been devoted to: 1) design of adversarial attacks against machine learning tasks [24, 27, 182], both at training time (poisoning attack) [12, 168] and at test time (evasion attack) [84];

Proposed method	White-box	Black-box	Perturb input	Perturb model
ADMM attack [183]	✓		✓	
ZO-ADMM attack [181]		✓	✓	
ZO-NGD attack [179]		✓	✓	
Fault sneaking attack [185]	✓			✓

Table 1.1: Proposed Attack Methods.

2) security evaluation methodologies to systematically estimate the DNN robustness [11, 174]; and 3) defense mechanisms under the attacks [22, 35, 106].

We investigate the design of adversarial attacks, robustness evaluation and defense against attacks. For the attack generation, we try to design adversarial perturbations which is added to the inputs or the DNN model parameters to cause certain misclassifications from the given model. Based on whether the attacker has complete access or control of the model, we can further generate adversarial examples in the white-box or black-box setting with different attack objectives and generation difficulties. We proposed various attack methods in different scenarios and summarize the proposed methods in Table 1.1.

For the robustness evaluation, we propose a certified bound on model weights [163] so that any model perturbations within that bound do not cause the change of the model's outputs. We demonstrate the effectiveness of the proposed method by comparing the certified bound with perturbations from fault sneaking attack [185] which tries to fool the DNN model by perturbing the model weights.

For the defense, we investigate the defense performance of model connection against different attacks methods including poisoning attack, fault injection attack and evasion attack [178]. We summarize the attacks and our defense performance for these attacks in Table 1.2.

1.2 DNN Efficiency

Despite the great achievements of DNN, it typically has large amount of parameters and computations, leading to high overhead for the memory and computing hardware resources. As the edge devices such as mobile phones become the main stream computing devices with ever-increasing popularity, it is even harder to run DNN models with large hardware resource requirements on the resource limited edge devices.

	perturb inputs		perturb model	defense performance [178]
	training time	test time		
Poisoning attack	✓			The attack can be mitigated and the model can be restored.
Fault injection attack			✓	The attack can be mitigated and the model can be restored.
Evasion attack		✓		The attack is still successful due to transferiability of adversarial examples

Table 1.2: Defense performance against different attack Methods.

Besides, many applications have real-time inference requirements such as object detection including 2D and 3D detection on edge devices of autonomous vehicles. However, the large computations of DNNs usually lead to long inference latency on edge devices, violating the real-time requirements. For example, the YOLOv4 [15] takes about 280ms to perform 2D object detection for one single image on mobile GPUs, which can hardly achieve real-time inference and lead to difficulties for real-time deployment on autonomous vehicles.

To improve the DNN efficiency with less hardware resource requirements and faster inference speed, we propose a compiler-aware neural pruning search [187], automatically determining the pruning scheme and rate (including bypass) for each individual layer. We can achieve (close-to) real-time, 55ms and 97ms inference times for YOLOv4 based 2D detection and PointPillars based 3D detection, respectively, on a mobile phone with minor (or no) accuracy loss. Our method on 2D detection outperforms other acceleration frameworks such as TVM [29] and MNN [1],while we are the first to support 3D detection on mobile.

Chapter 2

DNN Attack

2.1 Introduction

In recent years, deep neural networks (DNNs) have achieved significant breakthroughs [86] in many machine learning (ML) tasks. However, despite these success stories, there have been many recent studies showing that even state-of-the-art DNNs might still be vulnerable to adversarial misclassification attacks [57, 148, 171]. The adversarial attacks find and add visually imperceptible noises to an originally correctly classified input and essentially cause it to be misclassified by the DNNs. This raises security concerns about the robustness of DNNs in extreme situations with high reliability and dependability requirement such as face recognition, autonomous driving car and malware detection [47, 68, 140]. Investigating adversarial examples has become an increasingly prevailing topic to develop potential defensive measures in trustworthy ML [90, 158]. It essentially lays the groundwork for building a new generation of highly robust and reliable ML models acting as the core engine of future AI technology.

Based on whether attackers have complete access and knowledge of the target DNNs, adversarial attacks can be categorized into white-box attack and black-box attack. Most of preliminary studies on this topic are restricted to the white-box setting where the adversary has complete access and knowledge of the target system (e.g., DNNs) [24, 27, 57, 81]. White-box adversarial examples are constructed by adding negligible distortions onto original legal inputs, and usually the distortions are crafted by formulating and solving an optimization problem such as L-BFGS [148], C&W [24] and EAD [27]. The objectives of the optimization problem have two aspects: (1) misleading the DNN classifier to label the adversarial example as a target class, which is different from the original correct class, and (2) minimizing the ℓ_p norm of the added distortion to keep the noise imperceptible. To

achieve the above optimization objectives, we propose white-box ADMM attack to design adversarial examples with minimal perturbations.

Despite the theoretical interest, white-box attack methods are not adapted to practical black-box threat models. It is often the case that internal states/configurations and operating mechanism of public ML systems are not revealed to the practitioners (e.g., Google Cloud Vision API). Accordingly, the only mode of interaction with the system is via submitting inputs and receiving the corresponding predicted outputs.

To boost the practicality of such approaches in black-box adversarial attack, a few recent works have introduced a new class of threat models that exploit either a surrogate of the target model [126] or a gradient-free attack method [9, 28]. However, adversarial attacks that exploit a surrogate of the target model tend to yield low success rate if the surrogate is inaccurate. On the other hand, while attacks that use zeroth-order gradient estimation [28] are often more effective, they require a large number of queries to obtain an accurate estimate. Thus they are usually not economically efficient, especially in query-limited settings due to budget constraints. To improve the query efficiency, we propose ZO-ADMM method and ZO-NGD method to design adversarial examples in the black-box setting.

The adversarial attacks mainly add perturbations on the input images, i.e., perturbing the input space. There are some other attacks (such as fault injection attack) perturbing the parameter space, i.e., adding perturbations on the DNN model parameters. Fault injection attack perturbs the DNN parameter space. As DNNs are usually implemented and deployed on various hardware platforms including CPUs/GPUs and dedicated accelerators, it is possible to perturb the DNN parameters stored in memory enabled by the development of memory fault injection techniques such as laser beam [138] and row hammer [76].

It is a more challenging task to perturb the parameters (as fault injection attack) than to perturb the input images (as adversarial attack) due to the following two reasons: 1) global effect: perturbing one input would not influence the classifications of other unperturbed inputs while perturbing the parameters has a global effect for all inputs; 2) numerous parameters: the DNNs usually have a much greater number of parameters than the pixel number of an input image. The fault injection attack should be stealthy in that misclassifications are only for certain images while maintaining high model accuracy for the other images, and therefore cannot be easily detected. And it should also be efficient in that the parameter modifications should be as small as possible, and therefore can be implemented easily in the hardware. To tackles these challenges, we propose fault sneaking attack based on ADMM (alternating direction method of multipliers).

We summarize the proposed methods in Table 1.1.

2.2 Related Works

The vulnerability of DNNs was first studied in the seminal works [10, 148], which were followed by a series of white-box threat models [24, 57, 111, 182, 185] that assume full access of the target model's internal parameters/configurations. However, such internal knowledge of the target model is often not revealed and the adversary can only interact with it via submitting input queries and receiving feedback on potential outputs. Therefore, in the remaining of this section, we will summarize recent advances on black-box adversarial attacks and discuss their limitations in comparison to our proposed framework.

2.2.1 White-box Adversarial Attack

White-box threat models [57, 182, 185] assume full access of the target model's internal parameters/configurations. We introduce the representative white-box attacks.

L-BFGS Attack [148] is the first optimization-based attack and is an L_2 attack that uses L_2 norm to measure the distortion in the optimization objective function.

JSMA Attack [123] is an L_0 attack and uses a greedy algorithm that picks the most influential pixels by calculating Jacobian-based Saliency Map and modifies the pixels iteratively. The computational complexity is prohibitive even for applying to ImageNet dataset.

FGSM [56] and IFGSM [81] Attacks are L_∞ attacks and utilize the gradient of the loss function to determine the direction to modify the pixels. They are designed to be fast, rather than optimal. They can be used for adversarial training by directly changing the loss function instead of explicitly injecting adversarial examples into the training data. The fast gradient method (FGM) and the iterative fast gradient method (IFGM) are improvements of FGSM and IFGSM, respectively, that can be fitted as L_1, L_2, and L_∞ attacks.

C&W Attacks [24] are a series of L_0, L_2, and L_∞ attacks that achieve 100% attack success rate with much lower distortions comparing with the above-mentioned attacks. In particular, the C&W L_2 attack is superior to L-BFGS attack (which is also an L_2 attack) because it uses a better objective function.

EAD Attack [27] formulates the process of crafting adversarial examples as an elastic-net regularized optimization problem. Elastic-net regularization is a linear mixture of L_1 and L_2 norms

used in the penalty function. EAD attack is able to craft L_1-oriented adversarial examples and includes the C&W L_2 attack as a special case.

2.2.2 Black-box Adversarial Attack

The internal knowledge of the target model is often not revealed and the adversary can only interact with it via submitting input queries and receiving feedback on potential outputs. Therefore, we will summarize recent advances on black-box adversarial attacks and discuss their limitations.

2.2.2.1 Black-box Attack with Surrogate Model

A black-box attack using surrogate model is essentially a transfer attack [126] in which the adversary trains a DNN with data labeled by the target model. The resulting DNN is then exploited as a surrogate of the target model for which we can apply any state-of-the-art white-box attacks without requiring full access to internal states and operating mechanisms of the target model. Such attacks however depend heavily on the quality of training a surrogate model that closely resembles the true target model [101]. As a result, transfer attack tends to yield low success rate in data-intensive domains (e.g., ImageNet) for which it is hard to find a qualifiled surrogate.

2.2.2.2 Black-box Attacks with Gradient Estimation

Another approach to explore black-box attacks is to use gradient estimation via zeroth-order optimization (ZOO) [28]. They make queries to the model and estimate the output gradients with respect to the corresponding inputs, and then apply the state-of-the-art C&W attack method [24] to generate adversarial examples. However, this method is very computationally intensive as it requires a large number of queries per iteration to generate an accurate gradient estimation. Alternatively, the work [113] aims to estimate output gradient via greedy local search. At each iteration, the proposed technique perturbs only a small subset of input component. Such local search technique is very computationally efficient but it does not explicitly minimize the distortion between the original input and its perturbed version, the crafted noises often appear more visible. The work [70] investigates the more realistic threat models by defining the query-limited setting, the partial information setting, and the label-only setting. Three attacks methods are proposed based on the Natural Evolutionary Strategies and Monte Carlo approximation. But it only puts limits on the ℓ_∞ norm instead of minimizing a certain ℓ_p norm. Based on [70], the work [71] further investigates to utilize the prior information including the time-dependent priors (i.e., successive gradients are in fact heavily

correlated) and the data-dependent priors (i.e., images tend to exhibit a spatially local similarity) for higher query efficiency.

2.2.2.3 Other Black-box Attacks

In addition to the aforementioned works, there are also other black-box attacks [20, 32, 66, 153] under different practical settings, which are explored very recently. Among those, the notable boundary method [20] implements a decision-based attack, which starts from a very large adversarial perturbation (thus causing an immediate misclassification) and tries to reduce the perturbation (i.e., minimize the distortion) through a random walk while remaining adversarial via staying on the boundary between the misclassified class and the true class. However, it suffers from high computational complexity due to a huge number of queries needed to decrease the distortion and it also has no guarantee on the convergence. Different from [20], the work [32] formulates the hard-label black-box attack as a real-valued optimization problem which is usually continuous and can be solved by the zeroth-order optimization algorithm. Similarly, [66] addresses the problem of finding a universal (image-agnostic) perturbation in the hard-label black-box setting.

2.2.3 Perturbing the Parameter Space

Evasion attacks generate adversarial examples to fool DNNs by perturbing the legitimate inputs. Basically, an adversarial example is produced by adding human-imperceptible distortions onto a legitimate image, such that the adversarial example will be classified by the DNN as a target (wrong) label. The norm-ball constrained evasion attacks have been well studied, including the FGM [56] and IFGSM [82] attacks with ℓ_∞ norm restriction, the L-BFGS [148] and C&W [24] attacks minimizing the ℓ_2 distortion, and the JSMA [123] and ADMM [177] attacks trying to perturb the minimum number of pixels, namely, minimizing the ℓ_0 distortion.

Different from adversarial attacks to perturb the input images, there are some other methods to perturb the model parameter to achieve misclassifications. Poisoning attacks, which train DNNs by adding poisoned images into the training data sets, and fault injection attacks, which modify the DNN parameters directly, are attacks that perturb the DNN parameters. Poisoning attack [168] is computation-intensive as it requires iterative retraining and is not the focus of our paper. Fault injection attack [100] was first proposed by Liu et al, which uses a heuristic approach to profile the sink class for single bias attack scheme, and compresses the modification by iteratively enforcing the smallest element as zero and feasibility check for gradient descent attack scheme. Different

from [100], the fault sneaking attack uses a systematic optimization-based approach, achieving flexible designations of target labels and portion of DNN parameters to modify, and enabling both the ℓ_2 and ℓ_0 (non-differential) norms in the objective function.

2.2.3.1 Practical Fault Injection Techniques

The common techniques flipping the logic values in memory include laser beam and row hammer. Laser beam [7] can precisely change any single bit in SRAM by carefully tuning the laser beam such as diameter and energy level [138]. Row hammer [76] can inject faults into DRAM by rapidly and repeatedly accessing a given physical memory location to flip corresponding bits [169]. Some works demonstrate the feasibility of using row hammer on mobile platforms [154] and launching the row hammer to trigger the processor lockdown [73]. However, fine-tuning the laser beam or locating the bits in memory can be time consuming [154]. Therefore, it is essential to minimize the number of modified parameters by our fault sneaking attack. Recently, [19] implements the DNN fault injection attack [100] physically on embedded systems using laser beam. In particular, [19] injects faults into the widely used activation functions in DNNs and demonstrates the possibility to achieve misclassifications by injecting faults into the DNN hidden layer.

2.3 ADMM Attack

2.3.1 Problem Definition

We would like to investigate the problem of designing the perturbation, so that 1) the classification of the image after adding the perturbation can be changed to any target label and 2) the perturbation can be as small as possible under certain distance measure. The optimization problem can be formulated as below,

$$\begin{array}{ll} \underset{\boldsymbol{\delta}}{\text{minimize}} & D(\boldsymbol{\delta}) + g(\mathbf{x}+\boldsymbol{\delta}) \\ \text{subject to} & (\mathbf{x}+\boldsymbol{\delta}) \in [0,1]^n, \end{array} \tag{2.1}$$

where $D(\boldsymbol{\delta})$ denotes the distance measure of perturbation $\boldsymbol{\delta}$, and $g(\boldsymbol{x})$ has the following form,

$$g(\mathbf{x}) = \begin{cases} 0 & \text{if } \max_{i \neq t}\{Z(\mathbf{x})_i\} - Z(\mathbf{x})_t \leq 0 \\ \infty & \text{otherwise.} \end{cases} \tag{2.2}$$

$Z(\boldsymbol{x})_i$ represents the i-th element of the logits $Z(\boldsymbol{x})$, which is the output of all layers except the final softmax layer in the DNN model. D function is usually in the form of ℓ_p norm as the following,

$$D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_p = \left(\sum_{i=1}^{n} |\boldsymbol{\delta}_i|^p\right)^{\frac{1}{p}} \tag{2.3}$$

ℓ_0 norm measures the number of nonzero elements in $\boldsymbol{\delta}$. L_2 norm denotes the standard Euclidean distance of $\boldsymbol{\delta}$. ℓ_∞ norm represents the maximum absolute value of δ. In the paper, we investigate the problem where D function takes the form of ℓ_0, ℓ_1, ℓ_2 and ℓ_∞.

It is known from (2.2) that $g(\mathbf{x}) = 0$ iff

$$Z(\mathbf{x})_t \geq Z(\mathbf{x})_i, \quad \text{for all } i, \tag{2.4}$$

otherwise it would be ∞. So the solution of problem (2.1) exists only if $g(\mathbf{x}) = 0$, that is, the classification of $\boldsymbol{x} + \boldsymbol{\delta}$ is changed to the target label t. Meanwhile, the perturbation is minimized as the D function is optimized in the problem.

Since the function $g(\boldsymbol{x})$ is non-differential, we use the following form inspired by [24]:

$$g(\boldsymbol{x} + \boldsymbol{\delta}) = c \cdot \max\left(\left(\max_{i \neq t} \left(Z(\boldsymbol{x} + \boldsymbol{\delta})\right) - Z(\boldsymbol{x} + \boldsymbol{\delta})_t\right), 0\right) \tag{2.5}$$

The original problem can be transformed into as follows,

$$\begin{aligned} \underset{\boldsymbol{\delta}}{\text{minimize}} \quad & D(\boldsymbol{\delta}) + c \cdot \max\left(\left(\max_{i \neq t} \left(Z(\boldsymbol{x} + \boldsymbol{\delta})\right) - Z(\boldsymbol{x} + \boldsymbol{\delta})_t\right), 0\right) \\ \text{subject to} \quad & (\mathbf{x} + \boldsymbol{\delta}) \in [0, 1]^n, \end{aligned} \tag{2.6}$$

where $c \geq 0$ is a regularization parameter. Note that when $c = 0$, the solution attempts to minimize $D(\boldsymbol{\delta})$ only. In the other extreme case of $c \to \infty$, the solution moves towards the direction that $\boldsymbol{x} + \boldsymbol{\delta}$ must be an adversarial example.

2.3.2 ADMM Attack

2.3.2.1 ADMM Reformulation

To apply ADMM, we introduce *two* auxiliary variables $\mathbf{z}$ and $\mathbf{w}$, so that problem (2.6) can be rewritten as,

$$\begin{aligned} \underset{\boldsymbol{\delta}, \mathbf{z}, \mathbf{w}}{\text{minimize}} \quad & D(\boldsymbol{\delta}) + g(\mathbf{x} + \mathbf{z}) + h(\mathbf{w}) \\ \text{subject to} \quad & \mathbf{z} = \boldsymbol{\delta} \\ & \mathbf{w} = \mathbf{x} + \mathbf{z}, \end{aligned} \tag{2.7}$$

where $h(\mathbf{w})$ is the indicator function,

$$h(\mathbf{w}) = \begin{cases} 0 & \mathbf{w} \in [0,1]^n \\ \infty & \text{otherwise.} \end{cases} \tag{2.8}$$

The augmented Lagrangian of problem (2.7) is given by

$$\begin{aligned} L(\boldsymbol{\delta}, \mathbf{z}, \mathbf{w}, \mathbf{u}, \mathbf{v}) =& D(\boldsymbol{\delta}) + g(\mathbf{x}+\mathbf{z}) + h(\mathbf{w}) \\ &+ \mathbf{u}^T(\boldsymbol{\delta} - \mathbf{z}) + \mathbf{v}^T(\mathbf{w} - \mathbf{z} - \mathbf{x}) \\ &+ \frac{\rho}{2}\|\boldsymbol{\delta} - \mathbf{z}\|_2^2 + \frac{\rho}{2}\|\mathbf{w} - \mathbf{z} - \mathbf{x}\|_2^2, \end{aligned} \tag{2.9}$$

where $\mathbf{u}$ and $\mathbf{v}$ are Lagrangian multipliers.

ADMM yields the following alternating steps

$$\{\boldsymbol{\delta}^{k+1}, \mathbf{w}^{k+1}\} = \arg\min L(\boldsymbol{\delta}, \mathbf{z}^k, \mathbf{w}, \mathbf{u}^k, \mathbf{v}^k) \tag{2.10}$$

$$\mathbf{z}^{k+1} = \arg\min L(\boldsymbol{\delta}^{k+1}, \mathbf{z}, \mathbf{w}^{k+1}, \mathbf{u}^k, \mathbf{v}^k) \tag{2.11}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \rho(\boldsymbol{\delta}^{k+1} - \mathbf{z}^{k+1}) \tag{2.12}$$

$$\mathbf{v}^{k+1} = \mathbf{v}^k + \rho(\mathbf{w}^{k+1} - \mathbf{x}^{k+1} - \mathbf{z}^{k+1}). \tag{2.13}$$

In problem (2.10), the optimal $\boldsymbol{\delta}^{k+1}$ and $\boldsymbol{w}^{k+1}$ are obtained by minimizing the L function with fixed $\boldsymbol{z}^k$, $\boldsymbol{u}^k$ and $\boldsymbol{v}^k$. Similarly, the optimal $\boldsymbol{z}^{k+1}$ is achieved by solving problem (2.11) with fixed $\boldsymbol{\delta}^{k+1}$, $\boldsymbol{w}^{k+1}$, $\boldsymbol{u}^k$ and $\boldsymbol{v}^k$. In Eq. (2.12) and (2.13), we update $\boldsymbol{u}^k$ and $\boldsymbol{v}^k$ with $\boldsymbol{u}^{k+1}$ and $\boldsymbol{v}^{k+1}$, respectively.

The major advantage of the formulation (2.7) lies in the step (2.10). We note that the minimization over $\boldsymbol{\delta}$ and $\mathbf{w}$ can be split into two problems, each of which has the *closed-form* solution. Specifically, problem (2.10) can be equivalently transformed into problem (2.14) and (2.15),

$$\underset{\boldsymbol{\delta}}{\text{minimize}} \;\; D(\boldsymbol{\delta}) + \frac{\rho}{2}\|\boldsymbol{\delta} - \mathbf{z}^k + (1/\rho)\mathbf{u}^k\|_2^2 \tag{2.14}$$

and

$$\underset{\mathbf{w}}{\text{minimize}} \;\; h(\mathbf{w}) + \frac{\rho}{2}\|\mathbf{w} - \mathbf{z}^k - \mathbf{x} + (1/\rho)\mathbf{v}^k\|_2^2. \tag{2.15}$$

In problem (2.11), we need to solve the following problem,

$$\begin{aligned} \underset{\mathbf{z}}{\text{minimize}} \quad & g(\mathbf{x}+\mathbf{z}) + \tfrac{\rho}{2}\|\boldsymbol{\delta}^{k+1} - \mathbf{z} + (1/\rho)\mathbf{u}^k\|_2^2 \\ & + \tfrac{\rho}{2}\|\mathbf{w}^{k+1} - \mathbf{z} - \mathbf{x} + (1/\rho)\mathbf{v}^k\|_2^2, \end{aligned} \tag{2.16}$$

As specified above, the original problem is split into three subproblems, (2.14), (2.15) and (2.16), through ADMM. Note that in each subproblem, we only need to deal with one constraint, which is much easier to solve compared with the original problem dealing with multiple constraints at the same time.

2.3.2.2 δ-minimization: a universal solution framework

In what follows, we investigate how to obtain the solution to problem (2.14) through proximal operator. We first introduce the proximal operator [127] defined as,

$$\mathbf{prox}_{\lambda D}(\boldsymbol{s}) = \arg\min_{\boldsymbol{\delta}} \left(\lambda D(\boldsymbol{\delta}) + \frac{1}{2} \|\boldsymbol{\delta} - \boldsymbol{s}\|_2^2 \right) \tag{2.17}$$

where $\mathbf{prox}_{\lambda D}(\mathbf{s})$ means the optimal solution $\boldsymbol{\delta}^*$ which can achieve the minimal value of $\lambda D(\boldsymbol{\delta}) + \frac{1}{2} \|\boldsymbol{\delta} - \boldsymbol{s}\|_2^2$ given $\boldsymbol{s}$.

In problem (2.14), multiple choices of $D(\boldsymbol{\delta})$ are taken into consideration: 1) $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_0$, 2) $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_1$, 3) $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_2$, and 4) $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_\infty$. Each choice of $D(\boldsymbol{\delta})$ yields an analytic solution of problem (2.14) by evaluating the corresponding proximal operator with $\boldsymbol{s} = \mathbf{z}^k - 1/\rho \mathbf{u}^k$ and $\lambda = 1/\rho$ as specified in the following.

ℓ_0 attack For the case of ℓ_0 norm ($D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_0$), the proximal operator is,

$$\mathbf{prox}_{\lambda 0}(\mathbf{s}) = \arg\min_{\delta} \left(\lambda \|\boldsymbol{\delta}\|_0 + \frac{1}{2} \|\boldsymbol{\delta} - \boldsymbol{s}\|_2^2 \right) \tag{2.18}$$

The solution can be obtained elementwise,

$$(\mathbf{prox}_{\lambda 0}(\mathbf{s}))_i = \begin{cases} 0 & |s_i| < \sqrt{2\lambda} \\ 0 \ or \ s_i & |s_i| = \sqrt{2\lambda} \\ s_i & |s_i| > \sqrt{2\lambda} \end{cases} \tag{2.19}$$

where $\boldsymbol{s} = \mathbf{z}^k - 1/\rho \mathbf{u}^k$ and $\lambda = 1/\rho$.

ℓ_1 attack For the case of L_1 norm, that is, $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_1$, the proximal operator is,

$$\mathbf{prox}_{\lambda 1}(\mathbf{s}) = \arg\min_{\boldsymbol{\delta}} \left(\lambda \|\boldsymbol{\delta}\|_1 + \frac{1}{2} \|\boldsymbol{\delta} - \boldsymbol{s}\|_2^2 \right) \tag{2.20}$$

By performing the (elementwise) soft thresholding operator, we can get the solution,

$$(\mathbf{prox}_{\lambda 1}(\mathbf{s}))_i = \begin{cases} s_i - \lambda & s_i \geq \lambda \\ 0 & |s_i| < \lambda \\ s_i + \lambda & s_i \leq -\lambda \end{cases} \tag{2.21}$$

where $\boldsymbol{s} = \mathbf{z}^k - 1/\rho \mathbf{u}^k$ and $\lambda = 1/\rho$.

ℓ_2 attack For the case of L_2 norm, that is, $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_2$, the proximal operator is,

$$\mathbf{prox}_{\lambda 2}(\mathbf{s}) = \arg\min_{\boldsymbol{\delta}} \left(\lambda \|\boldsymbol{\delta}\|_2 + \frac{1}{2} \|\boldsymbol{\delta} - \boldsymbol{s}\|_2^2 \right) \tag{2.22}$$

By the 'block soft thresholding' operator [127], we can get the solution,

$$\mathbf{prox}_{\lambda 2}(\mathbf{s}) = \begin{cases} (1 - \lambda/\|\mathbf{s}\|_2)\mathbf{s} & \|\mathbf{s}\|_2 \geq \lambda \\ 0 & \|\mathbf{s}\|_2 < \lambda \end{cases} \tag{2.23}$$

where $\boldsymbol{s} = \mathbf{z}^k - 1/\rho \mathbf{u}^k$ and $\lambda = 1/\rho$.

ℓ_∞ attack For the ℓ_∞ norm, problem (2.14) becomes

$$\underset{\boldsymbol{\delta}}{\text{minimize}} \ \|\boldsymbol{\delta}\|_\infty + \frac{\rho}{2} \|\boldsymbol{\delta} - \mathbf{s}\|_2^2, \tag{2.24}$$

where $\mathbf{s} = \mathbf{z}^k - (1/\rho)\mathbf{u}^k$.

By introducing epigraph variable r, then problem (2.24) becomes

$$\begin{array}{ll} \underset{\boldsymbol{\delta}, r}{\text{minimize}} & r + \frac{\rho}{2} \|\boldsymbol{\delta} - \mathbf{s}\|_2^2 \\ \text{subject to} & \delta_i \leq r, \ i = 1, 2, \ldots, n, \end{array} \tag{2.25}$$

where the solution is given by the KKT conditions. The Lagrangian is

$$L(\boldsymbol{\delta}, t, \boldsymbol{\mu}) = t + (\rho/2) \|\boldsymbol{\delta} - \boldsymbol{s}\|_2^2 + \boldsymbol{\mu}^T (\boldsymbol{\delta} - t\mathbf{1}) \tag{2.26}$$

where $\boldsymbol{\mu}$ is the dual variable, and the optimality conditions are

$$\begin{aligned} \boldsymbol{\delta}_i^* \leq t^*, \boldsymbol{\mu}_i^* \geq 0, \boldsymbol{\mu}_i^* (\boldsymbol{\delta}_i^* - t^*) = 0, \\ \rho(\boldsymbol{\delta}_i^* - \boldsymbol{s}_i) + \boldsymbol{\mu}_i^* = 0, \mathbf{1}^T \boldsymbol{\mu}^* = 1 \end{aligned} \tag{2.27}$$

If $\boldsymbol{\delta}_i^* < t^*$, then the third condition implies that $\boldsymbol{\mu}_i^* = 0$, and if $\boldsymbol{\delta}_i^* = t^*$, the fourth condition implies that $\boldsymbol{\mu}_i^* = \rho(\boldsymbol{s}_i - t^*)$. Since $\boldsymbol{\mu}_i^* \geq 0$, we have

$$\boldsymbol{\mu}_i^* = \rho(\boldsymbol{s}_i - t^*)_+ \tag{2.28}$$

Substituting for $\boldsymbol{\mu}_i^*$ in the fifth condition gives

$$\sum_{i=1}^{n} \rho(\boldsymbol{s}_i - t^*)_+ = 1 \tag{2.29}$$

This equation can be solved for t^* by bisection using the initial interval $[\min_i s_i - (1/n), \max_i s_i]$. After solving (2.29) and obtaining t^*, we recover the solution to the original problem (2.24) via the following,

$$\boldsymbol{\delta}_i^* = \min\{t^*, \boldsymbol{s}_i\} \tag{2.30}$$

This follows by applying the third and fourth conditions.

2.3.2.3 w-minimization step

We solve problem (2.15) in this section. Based on the definition of the indicator function h, problem (2.15) becomes a projection problem onto a box constraint,

$$\underset{\mathbf{w}}{\text{minimize}} \ \frac{\rho}{2}\|\mathbf{w} - \mathbf{z}^k - \mathbf{x} + (1/\rho)\mathbf{v}^k\|_2^2 \tag{2.31}$$

$$\text{subject to } \mathbf{w} \in [0,1]^n. \tag{2.32}$$

The solution is given by

$$[\mathbf{w}^{k+1}]_i = \begin{cases} 0 & \text{if } [\mathbf{z}^k + \mathbf{x} - (1/\rho)\mathbf{v}^k]_i < 0 \\ 1 & \text{if } [\mathbf{z}^k + \mathbf{x} - (1/\rho)\mathbf{v}^k]_i > 1 \\ [\mathbf{z}^k + \mathbf{x} - (1/\rho)\mathbf{v}^k]_i & \text{otherwise,} \end{cases} \tag{2.33}$$

where $[\mathbf{w}]_i$ denotes the i-th entry of $\mathbf{w}$.

2.3.2.4 z-minimization step

In this section, we solve problem (2.16). We note that the g function in problem (2.16) depends on the DNN model and is usually non-convex, thus it's almost impossible to obtain its closed-form solution. But we can try to find its solution through gradient descent method. Stochastic gradient descent methods have been widely applied in deep learning for non-convex optimization. There are several gradient descent optimizers including standard gradient descent, gradient descent with momentum [133], and Adam [77]. We use Adam optimizer to solve problem (2.16) as Adam converges more quickly.

2.3.2.5 Linearized ADMM

In the ADMM algorithm (2.10)-(2.13), the computation bottleneck lies in the $\mathbf{z}$-minimization step (2.11), given by

$$\underset{\mathbf{z}}{\text{minimize}} \quad g(\mathbf{x} + \mathbf{z}) + \tfrac{\rho}{2}\|\mathbf{z} - \mathbf{a}\|_2^2 + \tfrac{\rho}{2}\|\mathbf{z} - \mathbf{b}\|_2^2, \tag{2.34}$$

where

$$\mathbf{a} := \boldsymbol{\delta}^{k+1} + (1/\rho)\mathbf{u}^k, \quad \mathbf{b} := \mathbf{w}^{k+1} - \mathbf{x} + (1/\rho)\mathbf{v}^k, \tag{2.35}$$

We assume that the function g (or its alternative) is smooth, and its gradient at the point $\mathbf{z}^k$ is given by $\nabla g(\mathbf{z}^k + \mathbf{x})$. Motivated by the linearized ADMM, we replace the function g with its first-order Taylor expansion plus a regularization term (known as Bregman divergence), $(\nabla g(\mathbf{z}^k + \mathbf{x}))^T(\mathbf{z} - \mathbf{z}^k) + \frac{1}{2}\|\mathbf{z} - \mathbf{z}^k\|^2_{\mathbf{G}}$, where $\mathbf{G}$ is a pre-defined positive definite matrix , and $\|\mathbf{x}\|^2_{\mathbf{G}} = \mathbf{x}^T\mathbf{G}\mathbf{x}$. Using the linearization technique, problem (2.34) simplifies to

$$\begin{aligned} \underset{\mathbf{z}}{\text{minimize}} \quad & (\nabla g(\mathbf{z}^k + \mathbf{x}))^T(\mathbf{z} - \mathbf{z}^k) + \tfrac{1}{2}\|\mathbf{z} - \mathbf{z}^k\|^2_{\mathbf{G}} \\ & + \tfrac{\rho}{2}\|\mathbf{z} - \mathbf{a}\|_2^2 + \tfrac{\rho}{2}\|\mathbf{z} - \mathbf{b}\|_2^2. \end{aligned} \tag{2.36}$$

We remark that the Bregman divergence term is used to stabilize the convergence of $\mathbf{z}$.

If we choose $\mathbf{G} = \alpha\mathbf{I}$ (similar to []) for a constant $\alpha > 0$, then the solution of problem (2.36) is given by

$$\mathbf{z}^{k+1} = \frac{1}{\alpha + 2\rho}(\alpha\mathbf{z}^k + \rho\mathbf{a} + \rho\mathbf{b} - \nabla g(\mathbf{z}^k + \mathbf{x})). \tag{2.37}$$

It is clear that we completely avoid the computation of the EXACT solution at the ADMM step (2.11) through linearization. The convergence of the linearized ADMM for convex problems has been widely studied. Recently, its convergence for nonconvex problems was shown in [8].

2.3.3 Experimental Results

We demonstrate the experimental results of the proposed ADMM attacks compared with state-of-the-art attacks, including C&W attacks [24], EAD attack [27], FGM and IFGM attacks [81], on three image classification datasets, MNIST [85], CIFAR-10 [79] and ImageNet [36].

2.3.3.1 Experiment Setup and Parameter Setting

Based on C&W attack setup[1], we train two networks for MNIST and CIFAR-10 datasets, respectively, and utilize a pre-trained network, Inception-v3 [147], for ImageNet.

For targeted attacks, we show the results of different methods to choose the target labels: 1) the average case randomly selects the target label from all the labels except the correct label, 2) the best case performs attacks using all incorrect labels, and report the target label that is the easiest

[1] https://github.com/carlini/nn_robust_attacks

Table 2.1: Adversarial attack success rate (ASR) and distortion of different L_2 attacks for different datasets

Data Set	Attack Method	Best Case				Average Case				Worst Case			
		ASR	$\boldsymbol{L_2}$	L_1	L_∞	ASR	$\boldsymbol{L_2}$	L_1	L_∞	ASR	$\boldsymbol{L_2}$	L_1	L_∞
MNIST	FGM(L_2)	99.3	2.158	23.7	0.562	43.2	3.18	37.6	0.761	0	N.A.	N.A.	N.A.
	IFGM(L_2)	100	1.61	18.2	0.393	99.7	2.43	31.8	0.574	99.3	3.856	54.1	0.742
	C&W(L_2)	100	1.356	13.32	0.394	100	1.9	21.11	0.533	99.6	2.52	30.44	0.673
	ADMM(L_2)	100	1.268	15.93	0.398	100	1.779	25.06	0.444	99.9	2.269	34.7	0.561
CIFAR-10	FGM(L_2)	99.7	0.418	13.85	0.05	40.6	1.09	37.4	0.62	1.2	4.17	119.3	0.43
	IFGM(L_2)	100	0.185	6.26	0.021	100	0.419	14.9	0.043	100	0.685	22.8	0.0674
	C&W(L_2)	100	0.170	5.721	0.0189	100	0.322	11.28	0.0347	100	0.445	15.79	0.0495
	ADMM(L_2)	100	0.163	5.66	0.0192	100	0.315	10.97	0.0354	100	0.427	15.05	0.0502
ImageNet	FGM(L_2)	15	2.37	815	0.129	3	7.51	2104	0.25	0	N.A.	N.A.	N.A.
	IFGM(L_2)	100	0.984	328	0.031	100	2.38	795	0.079	97.6	4.59	1354	0.177
	C&W(L_2)	100	0.449	126.8	0.0159	100	0.621	198	0.0218	100	0.81	272.3	0.031
	ADMM(L_2)	100	0.412	112.5	0.017	100	0.555	166.7	0.021	100	0.704	225.6	0.0356

to attack, and 3) the worst case performs attacks using all incorrect labels, and report the label that is the most difficult to attack.

The network architecture for MNIST and CIFAR-10 is the same with four convolutional layers, two max pooling layers, two fully connected layers and a softmax layer. It is able to achieve 99.5% accuracy on MNIST and 80% accuracy on CIFAR-10. On ImageNet, the Google Inception model can achieve 96% top-5 accuracy with input images of size $299 \times 299 \times 3$. We conduct all experiments on machines with NVIDIA GTX 1080 TI GPUs.

The implementations of FGM and IFGM are based on the CleverHans package [120]. The key distortion parameter ϵ is determined through a fine-grained grid search. For IFGM, there are 10 FGM iterations and the distortion parameter in each FGM iteration is set to $\epsilon' = \epsilon/10$, as demonstrated in [150] for its effectiveness.

The implementations of C&W attacks and EAD attack[2] are based on the GitHub code released by the authors. In the EAD attack, we use the ℓ_1 distortion measurement (ℓ_1) to select the final adversarial examples since it can usually obtain lower ℓ_1 distortion than the least elastic-net (EN) measurement. The key parameter β is set to 0.001.

Table 2.2: Adversarial attack success rate and distortion of ADMM and C&W L_0 attacks for MNIST and CIFAR-10

Dataset	Attack method	Best case		Average case		Worst case	
		ASR	L_0	ASR	L_0	ASR	L_0
MNIST	C&W(L_0)	100	7.88	100	16.58	100	29.84
	ADMM(L_0)	100	6.94	100	13.35	100	23.66
CIFAR	C&W(L_0)	100	8.16	100	20.82	100	35.07
	ADMM(L_0)	100	7.64	100	18.78	100	32.81

2.3.3.2 Attack Success Rate and Distortion for ADMM ℓ_2 attack

The attack success rate (ASR) is the percentage of the adversarial examples which are successfully misclassified to target labels by the DNN model. We report the average distortion of all successful adversarial examples and the distortion for zero ASR is not available (N.A.).

We perform adversarial attacks on MNIST, CIFAR-10 and ImageNet. For MNIST and CIFAR-10, 1000 correctly classified images are randomly selected from the test sets with 9 target labels for each image, so we craft 9000 adversarial examples for MNIST or CIFAR-10 using each attack method. For ImageNet, we randomly select 100 correctly classified images with 9 random target labels for each image.

We compare the ADMM ℓ_2 attack with FGM, IFGM and C&W ℓ_2 attacks. Table 2.1 shows the results on MNIST, CIFAR-10 and ImageNet. As we can see, FGM fails to generate adversarial examples with high success rate as it does not have any success guarantee. Among IFGM, C&W and ADMM ℓ_2 attacks, ADMM achieves the lowest ℓ_2 distortion for the best case, average case and worst case. IFGM has larger ℓ_2 distortions compared with C&W and ADMM attacks on the three datasets, especially on ImageNet. For the worst case, the ADMM attack can reduce the ℓ_2 distortion by about 10% compared with C&W ℓ_2 attack on MNIST and 12.5% on ImageNet.

2.3.3.3 ASR and Distortion for ADMM ℓ_0 attack

We demonstrate the performance of ADMM ℓ_0 attack in terms of attack success rate and ℓ_0 norm distortion in this section. The ADMM ℓ_0 attack is compared with C&W ℓ_0 attack on MNIST and CIFAR-10. 500 images are randomly selected from the test sets of MNIST and CIFAR-10, respectively, each with 9 target labels.

[2]https://github.com/ysharma1126/EAD-Attack

Table 2.3: Adversarial attack success rate (ASR) and distortion of different L_1 attacks for different datasets

Data Set	Methods	Best Case		Average Case		Worst Case	
		ASR	L_1	ASR	L_1	ASR	L_1
MNIST	IFGM(L_1)	100	17.3	100	34.6	100	58.4
	EAD(L_1)	100	7.74	100	14.16	100	21.38
	ADMM(L_1)	100	6.29	100	12.35	100	17.9
CIFAR-10	IFGM(L_1)	100	5.96	100	15.8	100	20.8
	EAD(L_1)	100	1.94	100	4.62	100	7.25
	ADMM(L_1)	100	1.75	100	3.750	100	5.92
ImageNet	IFGM(L_1)	100	298	100	580	100	685
	EAD(L_1)	100	60.98	100	112.7	100	185
	ADMM(L_1)	100	49.17	100	75.2	100	127

As observed from Table 2.2, both C&W and ADMM ℓ_0 attacks can achieve 100% attack success rate and ADMM ℓ_0 attack can achieve lower ℓ_0 distortion than the C&W ℓ_0 attack. For the worst case, the ADMM attack can reduce the ℓ_0 distortion by about 20% on MNIST compared with the C&W ℓ_0 attack.

2.3.3.4 ASR and Distortion for ADMM ℓ_1 attack

We compare the ADMM ℓ_1 attack with IFGM and EAD ℓ_1 [27] attacks. We report The ASR and the average distortion of all successful adversarial examples.

The results of the ℓ_1 attack are shown in Table 2.3. We can observe that the IFGM, EAD and ADMM ℓ_1 attacks can achieve 100% attack success rate. ADMM ℓ_1 attack can achieve the lowest ℓ_1 distortion. As demonstrated in Table 2.3, in the best case, the ADMM ℓ_1 attack can craft adversarial examples with a ℓ_1 norm about 19% smaller than that of the EAD ℓ_1 attack on MNIST and ImageNet. For the worst case, the ℓ_1 norm of ADMM ℓ_1 attack is about 33% lower on CIFAR-10 or ImageNet compared with the EAD ℓ_1 attack.

2.3.3.5 ASR and Distortion for ADMM ℓ_∞ attack

We compare the ADMM ℓ_∞ attack with the IFGM ℓ_∞ attack and report the ASR and the average distortion of all successful adversarial examples.

The results of the ADMM ℓ_∞ attack are demonstrated in Table 2.4. We can observe that both IFGM and ADMM ℓ_∞ attacks can achieve 100% attack success rate. ADMM ℓ_∞ attack can achieve lower ℓ_∞ norm compared with the IFGM ℓ_∞ attack. In the worst case, the improvement of

Table 2.4: Adversarial attack success rate (ASR) and distortion of different L_∞ attacks for different datasets

Data Set	Methods	Best Case		Average Case		Worst Case	
		ASR	$\boldsymbol{L_\infty}$	ASR	$\boldsymbol{L_\infty}$	ASR	$\boldsymbol{L_\infty}$
MNIST	IFGM(L_∞)	100	0.1535	100	0.234	100	0.367
	ADMM(L_∞)	100	0.1439	100	0.191	100	0.234
CIFAR-10	IFGM(L_∞)	100	0.00655	100	0.0149	100	0.0262
	ADMM(L_∞)	100	0.00548	100	0.011	100	0.0161
ImageNet	IFGM(L_∞)	100	0.0035	100	0.01	100	0.0152
	ADMM(L_∞)	100	0.00268	100	0.00466	100	0.0065

the ADMM ℓ_∞ attack over the IFGM ℓ_∞ attack is much more obvious. The ℓ_∞ distortion measure of the ADMM attack is about 40% smaller than the IFGM attack on MNIST or CIFAR-10 dataset for the worst case. On ImageNet, the ℓ_∞ norm of ADMM attack is 58% lower than that of IFGM attack.

2.4 ZO-ADMM Attack

2.4.1 Problem Definition

We focus on adversarial attacks in the application of image classification with DNNs. In what follows, we first provide a general problem formulation for adversarial attack which is amenable to either white-box or black-box settings. Then, we will develop an efficient solution to the more interesting black-box setting where the adversary only has access to certain types of output of the DNN model (its internal structures and configurations are unknown to the adversary). Specifically, given a legitimate image $\mathbf{x}_0 \in \mathbb{R}^d$ with its correct class label t_0, we aim to design an optimal adversarial perturbation $\boldsymbol{\delta} \in \mathbb{R}^d$ so that the perturbed example $(\mathbf{x}_0 + \boldsymbol{\delta})$ is misclassified to target class $t \neq t_0$ by the DNN model trained on legitimate images. The adversarial perturbation $\boldsymbol{\delta}$ can be obtained by solving the problem of the generic form,

$$\begin{array}{ll} \underset{\boldsymbol{\delta}}{\text{minimize}} & f(\mathbf{x}_0 + \boldsymbol{\delta}, t) + \gamma D(\boldsymbol{\delta}) \\ \text{subject to} & (\mathbf{x}_0 + \boldsymbol{\delta}) \in [0,1]^d,\ \|\boldsymbol{\delta}\|_\infty \leq \epsilon, \end{array} \tag{2.38}$$

where $f(\mathbf{x}, t)$ denotes an attack loss incurred by misclassifying $(\mathbf{x}_0 + \boldsymbol{\delta})$ to target class t, $D(\boldsymbol{\delta})$ is a distortion function that controls perceptual similarity between a legitimate image and an adversarial example, and $\|\cdot\|_\infty$ signifies the ℓ_∞ norm. In problem (2.38), the 'hard' constraints ensure that the perturbed noise $\boldsymbol{\delta}$ at each pixel (normalized to $[0, 1]$) is imperceptible up to a predefined ϵ-tolerant threshold, and the non-negative parameter γ places emphasis on the distortion. Furthermore, in the

above problem, we mainly set $D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_2^2$, which is motivated by the superior performance of the outstanding C&W ℓ_2 adversarial attack.

The problem (2.38) is the general form of the problem in [70, 71] which does not consider the $D(\boldsymbol{\delta})$ term. The advantage is that we are able to minimize the ℓ_p distortion after the adversarial perturbation is obtained, thus keeping the perturbation imperceptible. More specifically, if ϵ is too small, we may not be able to obtain a successful adversarial example. Thus, we need to increase ϵ to achieve a successful adversarial attack. But since ϵ only limits the largest element of the perturbation, the whole perturbation over the image might be relatively large and easy to be recognized in case of large ϵ. Thus, the $D(\boldsymbol{\delta})$ term in problem (2.38) helps to minimize the ℓ_p distortion of the whole perturbation, keeping it unnoticeable.

In the remaining of this subsection, we will discuss possible choices for the loss function $f(\mathbf{x}, t)$. Note that, without loss of generality, we only focus on targeted attack with designated target class t to mislead the DNN since the untargeted attack version can be easily implemented similar to the targeted attack [24]. We also emphasize that in the black-box setting, the gradients of $f(\mathbf{x}, t)$ can not be obtained directly as it does in the white-box setting. The form of the loss function $f(\mathbf{x}, t)$ depends on the constrained information in different black-box feedback settings. In particular, the definition of score-based (Section 2.4.1.1) and decision-based (Section 2.4.1.2) attacks as well as their loss functions will be discussed in the following subsections.

2.4.1.1 Score-based Attack

In the score-based attack setting, the adversaries are able to make queries to DNN to obtain the soft labels (i.e., scores or probabilities of an image belonging to different classes), while information on gradients are not available. The loss function of problem (2.38) in the score-based attack is:

$$f(\mathbf{x}_0 + \boldsymbol{\delta}, t) \;=\; \max\{\max_{j \neq t}\{\log P(\mathbf{x}_0 + \boldsymbol{\delta})_j\} - \log P(\mathbf{x}_0 + \boldsymbol{\delta})_t, -\kappa\}, \tag{2.39}$$

which is motivated by [24] and yields the best known performance among white-box attacks. $P(\mathbf{x})_j$ denotes the target model's prediction score or probability of the j-th class, and κ is a confidence parameter which is usually set to zero. Basically, this implies $f(\mathbf{x}_0 + \boldsymbol{\delta}, t) = 0$ if $P(\mathbf{x}_0 + \boldsymbol{\delta})_t$ is the largest among all classes, which means the perturbation $\boldsymbol{\delta}$ has successfully made the target model misclassified $\mathbf{x}_0 + \boldsymbol{\delta}$ to target class t. Otherwise, it will be larger than zero. Note that in Eqn. (2.39) the log probability $\log P(\mathbf{x})$ is used instead of directly using the actual probability $P(\mathbf{x})$. This is based on the observation that the output probability distribution tends to have one dominating class,

making the query on the probability/score less effective. The utilization of the log operator can help to reduce the effect of the dominating class while it preserves the probability order for all classes.

2.4.1.2 Decision-based Attack

Different from the score-based attack, the decision-based attack is more challenging in that the adversaries can only make queries to get the hard-labels instead of the soft-labels. Let $H(\mathbf{x})_i$ denote the hard-label decision. $H(\mathbf{x})_i = 1$ if the decision for $\boldsymbol{x}$ is label i, and 0 otherwise. We also have $\sum_{i=1}^{K} H(\mathbf{x})_i = 1$ for all K classes. Then the loss function of problem (2.38) in the decision-based attack is specified as

$$f(\mathbf{x}_0 + \boldsymbol{\delta}, t) = \max_{j \neq t} H(\mathbf{x}_0 + \boldsymbol{\delta})_j - H(\mathbf{x}_0 + \boldsymbol{\delta})_t, \tag{2.40}$$

Therefore, $f(\mathbf{x}_0 + \boldsymbol{\delta}, t) \in \{-1, 1\}$, and the attacker succeeds if $f(\mathbf{x}_0 + \boldsymbol{\delta}, t) = -1$. The loss function (2.40) is *nonsmooth* with *discrete outputs*. The decision-based attack is therefore more challenging because existing combinatorial optimization methods become almost ineffective or inapplicable.

2.4.2 ZO-ADMM Method

2.4.2.1 ADMM Reformulation

This subsection develops a general black-box adversarial attack framework for both the score-based and decision-based attacks by leveraging ADMM and gradient-free optimization. We will show that the proposed attack framework yields the following benefits: a) an efficient splitting between the black-box loss function and the adversarial distortion function, b) generalization to various ℓ_p norm involved hard/soft constraints, and c) compatibility to different gradient-free operations. By introducing an auxiliary variable $\mathbf{z}$, problem (2.38) can be rewritten in the favor of ADMM-type methods [18, 186],

$$\begin{array}{ll} \underset{\boldsymbol{\delta}, \mathbf{z}}{\text{minimize}} & f(\mathbf{x}_0 + \boldsymbol{\delta}, t) + \gamma D(\mathbf{z}) + \mathcal{I}(\mathbf{z}) \\ \text{subject to} & \mathbf{z} = \boldsymbol{\delta}, \end{array} \tag{2.41}$$

where $\mathcal{I}(\mathbf{z})$ is the indicator function given by,

$$\mathcal{I}(\mathbf{z}) = \begin{cases} 0 & (\mathbf{x}_0 + \mathbf{z}) \in [0,1]^d,\ \|\mathbf{z}\|_\infty \leq \epsilon, \\ \infty & \text{otherwise.} \end{cases} \tag{2.42}$$

The augmented Lagrangian of the reformulated problem (2.41) is given by

$$\mathcal{L}(\mathbf{z}, \boldsymbol{\delta}, \mathbf{u}) = \gamma D(\mathbf{z}) + \mathcal{I}(\mathbf{z}) + f(\mathbf{x}_0 + \boldsymbol{\delta}, t) + \mathbf{u}^T(\mathbf{z} - \boldsymbol{\delta}) + \frac{\rho}{2}\|\mathbf{z} - \boldsymbol{\delta}\|_2^2, \tag{2.43}$$

where $\mathbf{u}$ is Lagrangian multiplier, and $\rho > 0$ is a given penalty parameter. It can be further transformed as below,

$$\mathcal{L}(\mathbf{z}, \boldsymbol{\delta}, \mathbf{u}) = \gamma D(\mathbf{z}) + \mathcal{I}(\mathbf{z}) + f(\mathbf{x}_0 + \boldsymbol{\delta}, t) + \frac{\rho}{2}\left\|\boldsymbol{z} - \boldsymbol{\delta} + \frac{1}{\rho}\boldsymbol{u}\right\|_2^2 - \frac{1}{2\rho}\|\boldsymbol{u}\|_2^2. \tag{2.44}$$

The ADMM algorithm [18] splits optimization variables into *two* blocks and adopts the following iterative scheme,

$$\mathbf{z}^{k+1} = \arg\min_{\mathbf{z}} \mathcal{L}(\mathbf{z}, \boldsymbol{\delta}^k, \mathbf{u}^k), \tag{2.45}$$

$$\boldsymbol{\delta}^{k+1} = \arg\min_{\boldsymbol{\delta}} \mathcal{L}(\mathbf{z}^{k+1}, \boldsymbol{\delta}, \mathbf{u}^k), \tag{2.46}$$

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \rho(\mathbf{z}^{k+1} - \boldsymbol{\delta}^{k+1}), \tag{2.47}$$

where k denotes the iteration index. In problem (2.45), we minimize $\mathcal{L}(\mathbf{z}, \boldsymbol{\delta}, \mathbf{u})$ over $\mathbf{z}$ given parameters $\boldsymbol{\delta}^k$ and $\mathbf{u}^k$. In problem (2.46), we minimize $\mathcal{L}(\mathbf{z}, \boldsymbol{\delta}, \mathbf{u})$ over $\boldsymbol{\delta}$ given $\boldsymbol{z}^{k+1}$ from the previous step and $\mathbf{u}^k$. Then, the Lagrangian multiplier $\boldsymbol{u}$ is updated in Eqn. (2.47). The major advantage of this ADMM-type algorithm is that it allows us to split the original complex problem into sub-problems, each of which can be solved more efficiently or even analytically. In what follows, we solve problems (2.45) and (2.46) respectively.

2.4.2.2 z-step

Problem (2.45) can be rewritten as

$$\begin{array}{ll} \underset{\mathbf{z}}{\text{minimize}} & D(\mathbf{z}) + \frac{\rho}{2\gamma}\|\mathbf{z} - \mathbf{a}\|_2^2 \\ \text{subject to} & (\mathbf{x}_0 + \mathbf{z}) \in [0,1]^d,\ \|\mathbf{z}\|_\infty \leq \epsilon, \end{array} \tag{2.48}$$

where $\mathbf{a} = \boldsymbol{\delta}^k - (1/\rho)\mathbf{u}^k$. We set $D(\boldsymbol{z}) = \|\boldsymbol{z}\|_2^2$ [24]. Problem (2.48) can be decomposed elementwise as below,

$$\begin{array}{ll} \underset{z_i}{\text{minimize}} & \left(z_i - \frac{\rho}{2\gamma+\rho}a_i\right)^2 \\ \text{subject to} & ([\mathbf{x}_0]_i + z_i) \in [0,1],\ |z_i| \leq \epsilon, \end{array} \tag{2.49}$$

where $[\mathbf{x}]_i$ (or x_i) denotes the i-th element of $\mathbf{x}$. The solution to problem (2.49) is then given by

$$[\mathbf{z}^{k+1}]_i = \begin{cases} \min\{1 - [\mathbf{x}_0]_i, \epsilon\} & \frac{\rho}{2\gamma+\rho} a_i > \min\{1 - [\mathbf{x}_0]_i, \epsilon\} \\ \max\{-[\mathbf{x}_0]_i, -\epsilon\} & \frac{\rho}{2\gamma+\rho} a_i < \max\{-[\mathbf{x}_0]_i, -\epsilon\} \\ \frac{\rho}{2\gamma+\rho} a_i & \text{otherwise.} \end{cases} \tag{2.50}$$

Generalization to various ℓ_p norms. In problem (2.48), in addition to the worst-case perturbation constraint $\|\boldsymbol{z}\|_\infty \leq \epsilon$, it is a common practice to set $D(\boldsymbol{z}) = \|\boldsymbol{z}\|_2^2$ to measure the similarity between the legitimate image and the adversarial example. If $D(\boldsymbol{z})$ takes other ℓ_p norms such as $\|\boldsymbol{z}\|_0$, $\|\boldsymbol{z}\|_1$, or even ℓ_p norm combinations like $\|\boldsymbol{z}\|_1 + \frac{\beta}{2}\|\boldsymbol{z}\|_2^2$ for $\beta \geq 0$, we are still able to obtain the solutions with minor modifications in the z-step.

2.4.2.3 δ-step

Problem (2.46) can be written as

$$\underset{\boldsymbol{\delta}}{\text{minimize}} \quad f(\mathbf{x}_0 + \boldsymbol{\delta}, t) + \tfrac{\rho}{2}\|\boldsymbol{\delta} - \mathbf{b}\|_2^2, \tag{2.51}$$

where $\boldsymbol{b} = \boldsymbol{z}^{k+1} + (1/\rho)\boldsymbol{u}^k$. In the white-box setting, since the gradients of $f(\mathbf{x}_0 + \boldsymbol{\delta}, t)$ are directly accessible, gradient descent method like stochastic gradient descent (SGD) or Adam can be applied straight-forwardly. However, in black-box settings, the gradients of $f(\mathbf{x}_0 + \boldsymbol{\delta}, t)$ are unavailable. Thus, to overcome this difficulty, we adopt two derivative-free methods: the random gradient estimation (RGE) method [43] and the Bayesian optimization [16] corresponding to ZO-ADMM and BO-ADMM, respectively.

Random gradient estimation In the black-box setting, the gradient of $f(\mathbf{x}_0 + \boldsymbol{\delta}, t)$ is estimated through random gradient estimation (RGE),

$$\hat{\nabla} f(\boldsymbol{\delta}) = (d/(\nu Q)) \sum_{j=1}^{Q} \Big[f(\boldsymbol{\delta} + \nu \mathbf{u}_j) - f(\boldsymbol{\delta}) \Big] \mathbf{u}_j, \tag{2.52}$$

where d is the number of optimization variables, $\nu > 0$ is a smoothing parameter, $\{\mathbf{u}_j\}$ denote independent and identically distributed (i.i.d.) random direction vectors drawn from a uniform distribution over a unit sphere, and Q is the number of random direction vectors. It has been shown in [97] that a large Q reduces the gradient estimation error and improves the convergence of ZO-ADMM. However, we find that a moderate size of Q is sufficient to provide a good trade-off between estimation error and query complexity, e.g., $Q = 20$ in our experiments. We also highlight that the

RGE in (2.52) only requires $O(Q)$ query complexity instead of $O(dQ)$ caused by coordinate-wise gradient estimation used in [28]. Note that the natural evolutionary strategy (NES) [70] uses a central difference based gradient estimator requiring $2Q$ queries. By contrast, RGE uses a forward difference based random gradient estimator, yielding $Q+1$ query counts, leading to higher query efficiency.

With the aid of RGE, the solution to problem (2.51) can now be obtained via stochastic gradient descent-like methods. However, it suffers from extremely high iteration and function query complexity due to the non-linearity of f as well as the iterative nature of ADMM. To sidestep this computational bottleneck, we propose the use of the linearized ADMM algorithm [146] in ZO-ADMM with RGE, and thus it enjoys dual advantages of gradient-free operation and linearization of the loss function. By linearization, the loss function $f(\mathbf{x}_0+\boldsymbol{\delta},t)$ in problem (2.51) is replaced with its first-order Taylor expansion plus a regularization term (known as Bregman divergence), that is, $\hat{\nabla} f(\boldsymbol{\delta}^k+\mathbf{x}_0,t))^T(\boldsymbol{\delta}-\boldsymbol{\delta}^k)+\frac{1}{2}\|\boldsymbol{\delta}-\boldsymbol{\delta}^k\|_{\mathbf{G}}^2$, where $\mathbf{G}$ is a pre-defined positive definite matrix, and $\|\mathbf{x}\|_{\mathbf{G}}^2=\mathbf{x}^T\mathbf{G}\mathbf{x}$. We choose $\mathbf{G}=\eta_k\mathbf{I}$ where $1/\eta_k>0$ is a decaying parameter, e.g., $\eta_k=\alpha\sqrt{k}$ for a given constant $\alpha>0$. The Bregman divergence term is used to stabilize the convergence of $\boldsymbol{\delta}$.

Combining linearization and RGE, problem (2.51) now takes the following form:

$$\underset{\boldsymbol{\delta}}{\text{minimize}} \quad (\hat{\nabla} f(\boldsymbol{\delta}^k+\mathbf{x}_0,t))^T(\boldsymbol{\delta}-\boldsymbol{\delta}^k)+\frac{\eta_k}{2}\|\boldsymbol{\delta}-\boldsymbol{\delta}^k\|_2^2+\frac{\rho}{2}\|\boldsymbol{\delta}-\mathbf{b}\|_2^2, \tag{2.53}$$

which yields a quadratic programming problem with the following closed-form solution:

$$\boldsymbol{\delta}^{k+1}=(1/(\eta_k+\rho))\left(\eta_k\boldsymbol{\delta}^k+\rho\mathbf{b}-\hat{\nabla} f(\boldsymbol{\delta}^k+\mathbf{x}_0,t)\right). \tag{2.54}$$

Note that Eqn. (2.54) can be calculated with only one step of gradient estimation, which is a significant improvement on query efficiency compared with solving problem (2.51) using gradient descent method with thousands of random estimations. The convergence of the linearized ADMM for non-convex problems is proved in [96].

2.4.3 Customized Score-based and Decision-based Black-box Attacks

For the score-based black-box attack, problem (2.38) with loss function (2.58) can be naturally solved through the general ADMM framework.

In the decision-based black-box attack, the form of the loss function (2.40) is non-smooth with discrete outputs. To overcome the discontinuity in Eqn. (2.40), a smoothing version of (2.40), denoted by f_μ with smoothing parameter $\mu>0$ [52, 114], is taken into consideration,

$$f_\mu(\mathbf{x}_0+\boldsymbol{\delta},t) \;=\; \mathbb{E}_{\mathbf{u}\in U_{\mathrm{b}}}\left[f(\mathbf{x}_0+\boldsymbol{\delta}+\mu\mathbf{u},t)\right], \tag{2.55}$$

Table 2.5: Performance evaluation of adversarial attacks on MNIST and CIFAR-10.

Data set	Attack method		ASR	ℓ_1 distortion	ℓ_2 distortion	ℓ_∞ distortion	Query count on initial success	Reduction ratio on query count
MNIST	white -box	C&W white-box attack [24]	100%	22.14	1.962	0.5194	-	-
		Transfer attack (via C&W) [126]	30.6%	65.2	4.545	0.803	-	-
	score -based	ZOO attack [28]	98.8%	26.78	1.977	0.522	12,161	0.0 %
		score-based ZO-ADMM attack	98.3%	26.23	1.975	0.513	493.6	95.9%
		score-based BO-ADMM attack	87%	93.6	7.7	0.71	52.1	99.6%
	decision -based	boundary attack [20]	99%	32.9	2.21	0.563	25,328[a]	0%
		decision-based ZO-ADMM attack	100 %	30.48	2.166	0.548	7,603 [a]	62%
CIFAR-10	white -box	C&W white-box attack [24]	100 %	11.7	0.332	0.0349	-	-
		Transfer attack (via C&W) [126]	8.5%	103.6	3.845	0.421	-	-
	score -based	ZOO attack [28]	97.6 %	15.2	0.361	0.0405	9982	0.0 %
		score-based ZO-ADMM attack	98.7 %	13.1	0.417	0.0392	421	95.7%
		score-based BO-ADMM attack	84.1%	148	5.29	0.62	46.3	99.6%
	decision -based	boundary attack [20]	100%	19.4	0.421	0.045	16,720 [a]	0%
		decision-based ZO-ADMM attack	100%	17.25	0.415	0.0413	6,213 [a]	63%

[a] As the decision-based attacks start from images in the target class, it achieves initial success immediately. Therefore, the query count on the initial success of the decision-based attack actually means the query number when it achieves the reported ℓ_2 distortion.

where $U_{\rm b}$ is a uniform distribution within the unit Euclidean ball, or $\mathbf{u}$ can follow a standard Gaussian distribution [70]. The rational behind the smoothing technique in (2.55) is that the convolution of two functions, i.e., $\int_{\mathbf{u}} f(\mathbf{x}_0+\boldsymbol{\delta}+\mu\mathbf{u},t)p(\mathbf{u})d\mathbf{u}$, is at least as smooth as the smoothest of the two original functions [42]. Therefore, when p is the density of a random variable with respect to Lebesgue measure, the loss function (2.55) is then smooth. In practice, we consider an empirical Monte Carlo approximation of (2.55)

$$f_\mu(\mathbf{x}_0+\boldsymbol{\delta},t) \;\approx\; \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_0+\boldsymbol{\delta}+\mu\mathbf{u}_i,t), \tag{2.56}$$

where $\{\mathbf{u}_i\}$ are N i.i.d. samples drawn from $U_{\rm b}$. With the smoothed loss function as expressed in Eqn. (2.56), problem (2.38) can be solved by the proposed general ADMM framework. To initialize ADMM, we initialize the perturbation $\boldsymbol{\delta}$ so that the initial perturbed image belongs to the target class, yielding a benefit of reducing query complexity compared to the initialization with an arbitrary image [20].

2.4.4 Experimental Results

In this section, the experimental results of the score-based and decision-based black-box attacks are demonstrated. We compare the proposed ADMM-based framework with various attack

methods on three image classification datasets, MNIST [85], CIFAR-10 [79] and ImageNet [36]. The results of state-of-the-art white-box attack (i.e., C&W attack) are also provided for reference.

We train two networks for MNIST and CIFAR-10 datasets, respectively, which can achieve 99.5% accuracy on MNIST and 80% accuracy on CIFAR-10. The model architecture has four convolutional layers, two max pooling layers, two fully connected layers and a softmax layer. For ImageNet, we utilize a pre-trained Inception v3 network [147] instead of training our own model, which can achieve 96% top-5 accuracy. All experiments are conducted on machines with NVIDIA GTX 1080 TI GPUs.

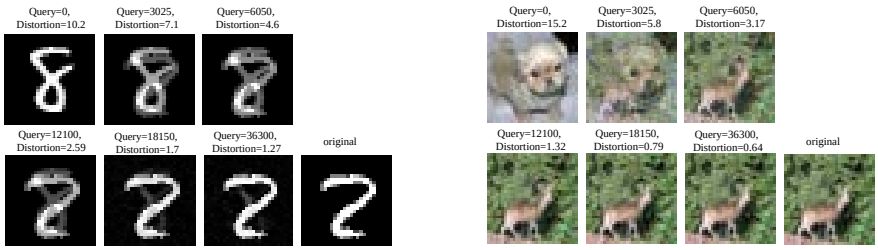
2.4.4.1 Evaluation on MNIST and CIFAR-10

In the evaluation on MNIST and CIFAR-10, 200 correctly classified images are selected from MNIST and CIFAR-10 test datasets, respectively. For each image, the target labels are set to the other 9 classes and a total of 1800 attacks are performed for each attack method.

The implementations of C&W (white-box) attack [24] and ZOO (black-box) attack [28] are based on the GitHub code released by the authors. For ZOO attack, we use ZOO-Adam with default Adam parameters. For the transfer attack [126], we apply C&W attack to the surrogate model with $\kappa = 20$ to improve the attack transferability and 2,000 iterations in each binary search step. In the proposed ZO-ADMM attack, the sampling number in random gradient estimation as defined in Eqn. (2.52), Q, is set to 20 and the sampling number for the decision-based smoothed loss function (2.56), N, is set to 10. We set $\rho = 10$ and $\gamma = 1$ for MNIST, $\rho = 2000$ and $\gamma = 10$ for CIFAR-10, and $\rho = 1000$ and $\gamma = 1$ for ImageNet. ϵ is set to 1 for MNIST and CIFAR-10 [3] and 0.05 for ImageNet. In Eq. (2.52), we set $\nu = 0.5$ for three datasets. The parameter μ in Eq. (2.56) is set to 1 for MNIST, 0.1 for CIFAR-10, and 0.01 for ImageNet.

The experimental results are shown in Table 2.5. Besides the attack success rate (ASR) and the ℓ_p norms, we report the query number required to achieve the first successful attack, which characterizes how fast the generated adversarial perturbation can mislead DNNs. We observe that the transfer attack suffers from low ASR and large ℓ_2 distortion. Both the ZOO attack and the proposed ZO-ADMM attack with RGE can achieve high ASR and competitive ℓ_2 distortion close to the C&W white-box attack. Compared with the ZOO attack, the score-based ZO-ADMM attack requires fewer queries to obtain the first successful adversarial example. The query count in ZO-ADMM attack with RGE is reduced by 95.9% and 95.7% on MNIST and CIFAR-10, respectively. The reduction of query

[3] This setting is intended to make a fair comparison to the pure ℓ_2-norm attack framework ZOO.



(a) An adversarial example evolution for MNIST starting from an image in the target class.

(b) An adversarial example evolution for CIFAR-10 starting from an image in the target class.

Figure 2.1: Adversarial examples generated by the proposed decision-based ZO-ADMM attack on MNIST and CIFAR-10.

number is achieved by Eq. (2.54) in ZO-ADMM, which only requires one step of gradient estimation to solve the approximation problem instead of thousands of steps to solve the original problem. We also observe that the score-based BO-ADMM attack can achieve smaller query number compared with the RGE method, but it causes much larger ℓ_p distortion. The reason is that BO-ADMM does not have very precise control for the perturbation. So it requires larger perturbation to mislead the DNN model. Although BO-ADMM may have its limitations, we find that combining the advantage of BO- and ZO-ADMM can lead to more query-efficient attacks. Please refer to the BO-ZO-ADMM section in the appendix.

We notice that the decision-based ZO-ADMM attack achieves an ℓ_2 distortion slightly larger than the score-based attack with more queries as shown in Table 2.5. This is not surprising, since only the hard label outputs are available in the decision-based attack, which is more difficult to be optimized than the score-based attack. Although the ℓ_2 distortion is a bit larger, the perturbations are still visually indistinguishable. We compare the decision-based ZO-ADMM attack with the boundary attack [20]. As demonstrated in Table 2.5, the queries of the decision-based ZO-ADMM attack is about 60% less than that of the boundary attack to achieve the same level ℓ_2 distortion. We show the evolution of several adversarial examples in the decision-based attack versus the query number in Fig. 2.1. The decision-based attack starts from an image in the target class. Then it tries to decrease the ℓ_2 norm while keeping the classified label unchanged. After about 20,000 queries, the example is close to the original image with a satisfied ℓ_2 distortion.

Table 2.6: Performance evaluation of adversarial attacks on ImageNet.

Attack method		Untargeted attack			Targeted attack		
		ASR	Query count on initial success	Reduction ratio	ASR	Query count on initial success	Reduction ratio
score-based	C&W white-box attack [24]	100%	-	-	99%	-	-
	ZOO attack [28]	90%	15631	0.0%	78%	2.11×10^6	0.0%
	Query-limited attack [70]	100%	4785	69.4%	98%	34128	98.4%
	$\text{Bandits}_{\text{TD}}$ attack [71]	94%	1259	92%	- [a]	-	-
	score-based ZO-ADMM attack	98%	891	94.3%	97%	16058	99.2%
decision-based	Label only [70]	- [b]	-	-	92 %	1.89×10^6 [c]	10.4%
	decision-based ZO-ADMM attack	100%	11742 [c]	24.9%	94%	1.52×10^6 [c]	28%

[a] It mainly explores untargeted attack. [b] The label only attack mainly explores targeted attack. [c] The query count on initial success for the decision-based attack means the query number when it achieves the same ℓ_2 distortion with the ZOO attack on its initial success.

2.4.4.2 Evaluation on ImageNet

We perform targeted and untargeted attacks in the score-based and decision-based settings on ImageNet. 100 correctly classified images are randomly selected. For each image in targeted attack, 9 random labels out of 1000 classes are selected to serve as the targets. We do not perform the transfer attack since it does not scale well to ImageNet due to training of the surrogate model. Instead, we provide the results of new baselines on ImageNet, including the query-limited attack as well as the label-only attack proposed in [70], and the bandit optimization based attack with time and data-dependent priors (named as $\text{Bandits}_{\text{TD}}$) [71]. The query-limited and $\text{Bandits}_{\text{TD}}$ attacks are score-based attacks. The label-only attack is a decision-based attack.

The experimental results are summarized in Table 2.6. For score-based attacks, we can observe that the score-based ZO-ADMM attack can achieve a high ASR with fewer queries than the other attacks. It reduces the query number on initial success by 94.3% and 99.2% for untargeted and targeted attacks, respectively, compared with the ZOO attack. For decision-based attacks, the ZO-ADMM attack can obtain a high ASR with fewer queries compared with the label-only attack or even the ZOO attack using score-based information.

2.4.4.3 Comparison with AutoZoom and Boundary method

For the comparison with AutoZoom [153], we report the averaged number of queries for attacking 500images at the same ℓ_2 distortion level for MNIST, CIFAR-10,and ImageNet in Table 2.7. As we can see, the proposed ZO-ADMM method is more query-efficient, while it is worth noting that AutoZOOM produces adversarial perturbation in low-dimensional latent space,and thus saves

more computation cost.

Table 2.7: Comparison with AutoZOOM in attack success rate (ASR) and query #.

	MNIST		CIFAR-10		ImageNet	
	ASR	# of Query	ASR	# of Query	ASR	# of Query
AutoZoom	100%	1821	99.2%	1639	98.3%	43547
ZO-ADMM	100%	562	99%	492	99%	16390

In Table 2.8, we show the comparison of ZO-ADMM method with the Query-limited [70] and Boundary methods [20] in terms of query number and ℓ_p norms on ImageNet.

Table 2.8: Comparison with Query-limited and Boundary methods on ImageNet.

Settings	Methods	ASR	ℓ_1	ℓ_2	ℓ_∞	Query #
Score-based	Query-limited [18]	98%	1251	4.8	0.049	3.4×10^5
	ZO-ADMM	97%	785	3.5	0.039	1.6×10^5
Decision-based	Boundary [20]	85%	1120	3.99	0.045	2.2×10^6
	ZO-ADMM	93%	962	3.92	0.042	1.5×10^6

2.4.4.4 Convergence of the ZO-ADMM Attack

In Fig. 2.2, we demonstrate the convergence of the proposed ZO-ADMM targeted black-box attack, where the average ℓ_2 distortion of 9 targeted adversarial examples versus the query number is presented. As we can see, since we initialize the adversarial distortion from zeros, the score-based ZO-ADMM attack increases ℓ_2 distortion until a successful adversarial example is found. After that, it tries to decrease the ℓ_2 distortion but keeps the target label unchanged. For the decision-based attack, Fig 2.2 shows that the ℓ_2 distortion is initially large as ZO-ADMM starts from an image in the target class instead of the original image. The resulting ℓ_2 distortion then decreases as the query number increases. We highlight that the ZO-ADMM attack is able to reach the successful attack with hundreds of queries on MNIST or CIFAR-10 and tens of thousands of queries on ImageNet, which significantly outperforms the ZOO attack. Besides Fig. 2.2 demonstrating the ℓ_2 distortion versus query number, we present the ℓ_2 distortion versus ADMM iteration number in the supplementary material and similar results can be drawn.

2.4.4.5 Evaluation for Various ℓ_p Norms

In the previous experiments, we mainly consider the case of $D(\boldsymbol{z}) = \|\boldsymbol{z}\|_2^2$ for a fair comparison with other white-box and black-box algorithms. However, we highlight that the ZO-ADMM method is able to optimize various ℓ_p norms, not only ℓ_2 norm. In Table 2.9, we present

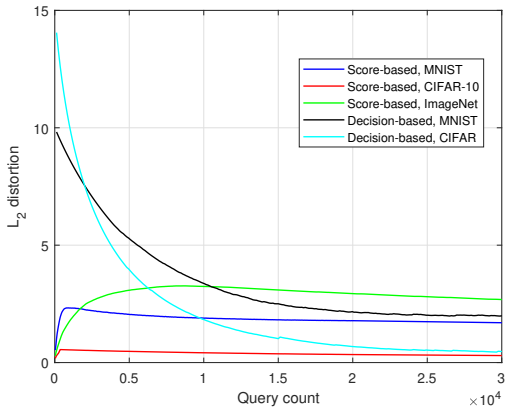


Figure 2.2: Convergence of the ZO-ADMM attack.

Table 2.9: Performance evaluation of the ZO-ADMM attacks on MNIST for different ℓ_p norms.

Attack method	ASR	ℓ_0	ℓ_1	ℓ_2
ZO-ADMM ℓ_0	100%	**18.5**	12.6	9.72
ZO-ADMM ℓ_1	100%	465	**10.5**	2.71
ZO-ADMM ℓ_2	100%	483	22.09	**1.93**

the experimental results for different ℓ_p norms when solving problem (2.48). Here we focus on three score-based black-box attacks with ZO-ADMM by minimizing the ℓ_0, ℓ_1 and ℓ_2 distortion, respectively. As we can see, our proposed method is well adapted to different ℓ_p norms in the design of black-box adversarial examples.

2.5 ZO-NGD Attack

2.5.1 Problem Definition

Threat Model: We mainly investigate black-box adversarial attacks for image classification with DNNs. Different from the white-box setting which has fully access to the DNN model and its internal structures/parameters, the black-box setting constrains the information available to the adversary. The attacker can only query the model by providing an input image and obtain the DNN output score/probability of the input. The black-box setting is more consistent with the scenario of "machine-learning-deployed-as-a-service" like Google Cloud Vision API.

In the following, we first provide a general problem formulation for adversarial attack which can be adopted to either white-box or black-box settings. Then, an efficient solution is proposed for the black-box setting. We highlight that this method can be easily adopted to the white-box setting by using the exact gradients to achieve higher query efficiency.

Attack Model: Given a legitimate image $\mathbf{x} \in \mathbb{R}^d$ with its correct class label t, the objective is to design an optimal adversarial perturbation $\boldsymbol{\delta} \in \mathbb{R}^d$ so that the perturbed example $(\mathbf{x} + \boldsymbol{\delta})$ can lead to a misclassification by the DNN model trained on legitimate images. The DNN model would misclassify the adversarial example to another class $t' \neq t$. $\boldsymbol{\delta}$ can be obtained by solving the following problem,

$$\underset{\boldsymbol{\delta} \in \mathbb{S}}{\text{minimize}}\, f(\mathbf{x} + \boldsymbol{\delta}, t), \tag{2.57}$$

where $\mathbb{S} = \{\boldsymbol{\delta} | (\mathbf{x} + \boldsymbol{\delta}) \in [0,1]^d,\ \|\boldsymbol{\delta}\|_\infty \leq \epsilon\}$ and $f(\mathbf{x} + \boldsymbol{\delta}, t)$ denotes an attack loss incurred by misclassifying $(\mathbf{x} + \boldsymbol{\delta})$ to another class t'. $\|\cdot\|_\infty$ denotes the ℓ_∞ norm. In problem (2.57), the constraints on $\mathbb{S}$ ensure that the perturbed noise $\boldsymbol{\delta}$ at each pixel (normalized to $[0,1]$) is imperceptible up to a predefined ϵ-tolerant threshold.

Motivated by [24], the loss function $f(\boldsymbol{x}, t)$ is expressed as

$$f(\mathbf{x} + \boldsymbol{\delta}, t) \ = \ \max\{\log p(t|\mathbf{x} + \boldsymbol{\delta}) - \max_{i \neq t}\{\log p(i|\mathbf{x} + \boldsymbol{\delta})\}, -\kappa\}, \tag{2.58}$$

where $p(i|\mathbf{x})$ denotes the model's prediction score or probability of the i-th class for the input $\boldsymbol{x}$, and κ is a confidence parameter usually set to zero. Basically, $f(\mathbf{x} + \boldsymbol{\delta}, t)$ achieves its minimum value 0 if $p(t|\mathbf{x} + \boldsymbol{\delta})$ is smaller than $\max_{i \neq t} \log p(i|\mathbf{x} + \boldsymbol{\delta})$, indicating there is a label with higher probability than the correct label t and thus a misclassification is achieved by adding the perturbation $\boldsymbol{\delta}$ to $\boldsymbol{x}$. In this paper, we mainly investigate the untargeted attack which does not specify the target misclassified label. The targeted attack can be easily implemented following nearly the same problem formulation and loss function with slight modifications [24, 71]. We focus on the general formulation here and omit the targeted attack formulation.

Note that in Eq. (2.58), we use the log probability $\log p(i|\mathbf{x})$ instead of $p(i|\mathbf{x})$ because the output probability distribution tends to have one dominating class. The log operator is used to reduce the effect of the dominating class while it still preserves the probability order of all classes.

As most of the white-box attack methods rely on gradient descent methods, the unavailability of the gradients in black-box settings will limit their application. Gradient estimation methods (known as zeroth-order optimization) are applied to perform the normal projected first-order gradient descent process [70, 98] as follows,

$$\boldsymbol{\delta}_{k+1} = \prod_{\mathbb{S}} \left(\boldsymbol{\delta}_k - \lambda \hat{\nabla} f(\boldsymbol{\delta}_k) \right), \tag{2.59}$$

where λ is the learning rate and the $\prod_{\mathbb{S}}(\cdot)$ performs the projection onto the feasible set $\mathbb{S}$.

In the black-box setting, it is often the case that the query number is limited or high query efficiency is required by the adversary. The zeroth-order method tries to extract gradient information of the objective function and the first-order method is applied to minimize the loss due to its wide application in ML. However, the second-order information of the queries is not fully exploited. In this paper, we aim to take advantages of the model's second-order information and propose a novel method named ZO-NGD optimization.

2.5.2 ZO-NGD Method

The proposed method is based on NGD [108] and ZO optimization [41]. In the applications of optimizing probabilistic models, NGD uses the natural gradient by multiplying the gradient with the FIM to update the parameters. NGD seems to be a potentially attractive alternative method as it requires fewer total iterations than gradient descent [59, 109, 118].

Motivated from the perspective of information geometry, NGD defines the steepest descent/direction in the realizable distribution space instead of the parameter space. The distance in the distribution space is measured with a special "Riemannian metric" [5], which is different from the standard Euclidean distance metric in the parameter space. This Riemannian metric does not rely on the parameters like the Euclidean metric, but depends on the distributions themselves. Thus it is invariant to any smooth or invertible reparameterization of the model. More details are discussed in the Geometric Interpretation Section.

Next we will introduce the FIM and the implementation details to perform NGD. Basically, the proposed framework first queries the model to estimate the gradients and Fisher information. Then after the damping and inverting processes, natural gradient is obtained to update the perturbation. Algorithm 1 shows the pseudo code of the ZO-NGD.

2.5.2.1 Fisher Information Matrix and Natural Gradient

We introduce and derive the FIM in this section. In general, finding an adversarial example can be formulated as a training problem. In the idealized setting, input vectors $\boldsymbol{x}$ are drawn independently from a distribution $Q_{\boldsymbol{x}}$ with density function $q(\boldsymbol{x})$, and the corresponding output t is drawn from a conditional target distribution $Q_{t|\mathbf{x}}$ with density function $q(t\,|\boldsymbol{x}\,)$. The target joint distribution is $Q_{t,\mathbf{x}}$ with the density of $q(t, \boldsymbol{x}) = q(t|\boldsymbol{x})q(\boldsymbol{x})$. By finding an adversarial perturbation

Algorithm 1 Framework of ZO-NGD.

Input:

The legitimate image $\boldsymbol{x}$; the correct label t; the model to be queried; the learning rate λ; the sampling step size μ;

Output:

Adversarial perturbation $\boldsymbol{\delta}$;

initialize $\boldsymbol{\delta}_0$ with all zeros;
for $k = 0, ..., K$ **do**
  Query the model with $\boldsymbol{\delta}_k$ and obtain the probability $p(t|\boldsymbol{x}, \boldsymbol{\delta}_k) := p(t|\boldsymbol{x} + \boldsymbol{\delta}_k)$;
  for $j = 1, ..., R$ **do**
    Generate a random direction vector $\mathbf{u}_j$ drawn from a uniform distribution over the surface of a unit sphere;
    Query the model with $\boldsymbol{x} + \boldsymbol{\delta}_k + \mu\mathbf{u}_j$ and obtain $p(t|\boldsymbol{x}, \boldsymbol{\delta}_k + \mu\mathbf{u}_j)$;
  end for
  Estimate the gradients of the loss function $\hat{\nabla} f(\boldsymbol{\delta}_k)$ according to Eq. (2.72);
  Estimate the gradients of the log-likelihood function $\hat{\nabla} \log p(t\,|\mathbf{x}, \delta_{\mathbf{k}})$ according to Eq. (2.73);
  Compute the FIM $\mathbf{F}$ according to Eq. (2.74) and perform the nature gradient update as shown in Eq. (2.75).
end for

$\boldsymbol{\delta}$, we obtain the learned distribution $P_{t,\mathbf{x}}(\boldsymbol{\delta})$, whose density is $p(t, \boldsymbol{x}|\boldsymbol{\delta}) = p(t|\boldsymbol{x} + \boldsymbol{\delta})q(\boldsymbol{x}) := p(t|\boldsymbol{x}, \boldsymbol{\delta})q(\boldsymbol{x})$.

In statistics, the score function [34] indicates how sensitive a likelihood function $p(t, \boldsymbol{x}|\boldsymbol{\delta})$ is to its parameters $\boldsymbol{\delta}$. Explicitly, the score function for $\boldsymbol{\delta}$ is the gradient of the log-likelihood with respect to $\boldsymbol{\delta}$ asbelow,

$$s(\boldsymbol{\delta}) = \nabla \log p(t, \boldsymbol{x}|\boldsymbol{\delta}). \tag{2.60}$$

Lemma 2.5.1. *The expected value of the score function with respect to $\boldsymbol{\delta}$ is zero.*

Proof.

$$\begin{aligned}
\mathop{\mathbb{E}}_{p(t,\boldsymbol{x}|\boldsymbol{\delta})}[s(\boldsymbol{\delta})] &= \mathop{\mathbb{E}}_{p(t,\boldsymbol{x}|\boldsymbol{\delta})}[\nabla \log p(t,\boldsymbol{x}|\boldsymbol{\delta})] \\
&= \int \nabla \log p(t,\boldsymbol{x}|\boldsymbol{\delta}) p(t,\boldsymbol{x}|\boldsymbol{\delta}) dt\, d\boldsymbol{x} \\
&= \int \frac{\nabla p(t,\boldsymbol{x}|\boldsymbol{\delta})}{p(t,\boldsymbol{x}|\boldsymbol{\delta})} p(t,\boldsymbol{x}|\boldsymbol{\delta}) dt\, d\boldsymbol{x} \\
&= \nabla \int p(t,\boldsymbol{x}|\boldsymbol{\delta}) dt\, d\boldsymbol{x} \\
&= 0
\end{aligned}$$

□

We can define an uncertainty measure around the expected value (i.e., the covariance of the score function) as follows,

$$\mathbb{E}\left[(s(\boldsymbol{\delta}) - \mathbb{E}[s(\boldsymbol{\delta})])\,(s(\boldsymbol{\delta}) - \mathbb{E}[s(\boldsymbol{\delta})])^{\mathrm{T}}\right]. \tag{2.61}$$

The covariance of the score function above is the definition of the Fisher information. It is in the form of a matrix and the FIM can be written as

$$\mathbf{F} = \mathbb{E}_{\boldsymbol{x}\in Q_x}\left[\mathbb{E}_{\tilde{t}\sim p(\cdot|\boldsymbol{x},\boldsymbol{\delta})}\left[\nabla \log p\left(\tilde{t},\boldsymbol{x}\,|\boldsymbol{\delta}\right) \nabla \log p\left(\tilde{t},\boldsymbol{x}\,|\boldsymbol{\delta}\right)^{T}\right]\right]. \tag{2.62}$$

Note that this expression involves the losses on all possible values of the classes $\tilde{t}$, not only the actual label for each data sample. As $\boldsymbol{\delta}$ only corresponds to a single input $\boldsymbol{x}$, the training set only contains one data sample. Besides, since $p(\tilde{t},\boldsymbol{x}|\boldsymbol{\delta}) = p(\tilde{t}|\boldsymbol{x}+\boldsymbol{\delta})q(x) = p(\tilde{t}|\boldsymbol{x},\boldsymbol{\delta})q(\boldsymbol{x})$ and $q(\boldsymbol{x})$ does not depend on $\boldsymbol{\delta}$, we have

$$\begin{aligned}
\nabla \log p(\tilde{t},\boldsymbol{x}|\boldsymbol{\delta}) &= \nabla \log p(\tilde{t}|\boldsymbol{x},\boldsymbol{\delta}) + \nabla \log q(\boldsymbol{x}) \\
&= \nabla \log p(\tilde{t}|\boldsymbol{x},\boldsymbol{\delta}).
\end{aligned} \tag{2.63}$$

Then the FIM can be transformed to

$$\mathbf{F} = \mathbb{E}_{\tilde{t}\sim p(\cdot|\boldsymbol{x},\boldsymbol{\delta})}\left[\nabla \log p(\tilde{t}|\boldsymbol{x},\boldsymbol{\delta}) \nabla \log p(\tilde{t}|\boldsymbol{x},\boldsymbol{\delta})^{\mathrm{T}}\right]. \tag{2.64}$$

The exact expectation with T categories is expressed as,

$$\mathbf{F} = \sum_{\tilde{t}=1}^{T} p\left(\tilde{t}\,|\boldsymbol{x},\boldsymbol{\delta}\right) \nabla \log p\left(\tilde{t}\,|\boldsymbol{x},\boldsymbol{\delta}\right) \nabla \log p\left(\tilde{t}\,|\boldsymbol{x},\boldsymbol{\delta}\right)^{T} \tag{2.65}$$

The usual definition of the natural gradient is

$$\tilde{\nabla} f(\boldsymbol{\delta}) = \mathbf{F}^{-1} \nabla f(\boldsymbol{\delta}), \tag{2.66}$$

and the NGD minimizes the loss function through

$$\boldsymbol{\delta}_{k+1} = \boldsymbol{\delta}_k - \lambda \tilde{\nabla} f(\boldsymbol{\delta}_k). \tag{2.67}$$

2.5.2.2 Outer Product and Monte Carlo Approximation

The FIM involves an expectation over all possible classes $\tilde{t} \sim p(\cdot|x, \delta)$ drawn from the probability distribution output. In the case with large number of classes, it is impractical to compute the exact FIM due to the intensive computation. To address the high computation overhead, in general there are two methods to approximate the FIM, the outer product approximation and the Monte Carlo approximation.

Outer Product Approximation The outer product approximation of the FIM [118, 129] only uses the actual label t to avoid the expectation over all possible labels $\tilde{t} \sim p(\cdot|x, \delta)$, as below,

$$\mathbf{F}_{OP} = \nabla \log p(t|\boldsymbol{x}, \boldsymbol{\delta}) \nabla \log p(t|\boldsymbol{x}, \boldsymbol{\delta})^T. \tag{2.68}$$

Thus a rank-one matrix can be obtained directly.

Monte Carlo Approximation Monte Carlo (MC) approximation [118] replaces the expectation over $\tilde{t}$ with n_{MC} samples,

$$\mathbf{F}_{MC} = \frac{1}{n_{MC}} \sum_{i=1}^{n_{MC}} \nabla \log p\left(\tilde{t}_i|\boldsymbol{x}, \boldsymbol{\delta}\right) \nabla \log p\left(\tilde{t}_i|\boldsymbol{x}, \boldsymbol{\delta}\right)^T. \tag{2.69}$$

where each $\tilde{t}_i$ is drawn from the distribution $p(\cdot|x, \delta)$. The MC natural gradient works well in practice with $n_{MC} = 1$.

For higher query efficiency, we adopt the outer product approximation as it does not require additional queries.

2.5.2.3 Gaussian Smoothing and Gradient Estimation

To compute the FIM and perform NGD, we need to obtain the gradients of the loss function $\nabla f(\boldsymbol{\delta})$ and the gradients of the log-likelihood $\nabla p(t|\boldsymbol{x}, \boldsymbol{\delta})$, which are not directly available in the black-box setting.

To address this difficulty, we first introduce the Gaussian approximation of $f(\boldsymbol{x})$ [115],

$$f_\mu(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}} \int_{R^d} f(\boldsymbol{x} + \mu\boldsymbol{u}) \exp\left(-\frac{1}{2}\|\boldsymbol{u}\|^2\right) d\boldsymbol{u}, \tag{2.70}$$

where $\|\cdot\|$ is the Frobenius norm, $\mu > 0$ is a smoothing parameter and $\boldsymbol{u}$ is a random vector distributed uniformly over the surface of a unit sphere, i.e., $\boldsymbol{u} \sim N(0, \mathbf{I}_d)$. Its gradient can be written as

$$\begin{aligned} \nabla f_\mu(\boldsymbol{x}) &= \frac{1}{M} \int_{R^d} \frac{f(\boldsymbol{x} + \mu\boldsymbol{u}) - f(\boldsymbol{x})}{\mu} \boldsymbol{u} \exp\left(-\frac{1}{2}\|\boldsymbol{u}\|^2\right) d\boldsymbol{u} \\ &= E_{\boldsymbol{u}}\left(\frac{f(\boldsymbol{x} + \mu\boldsymbol{u}) - f(\boldsymbol{x})}{\mu}\boldsymbol{u}\right), \end{aligned} \tag{2.71}$$

where $E_{\boldsymbol{u}}$ is the Gaussian smoothing function. Thus, based on Eq. (2.71), we apply the zeroth-order random gradient estimation to estimate the gradients by

$$\hat{\nabla} f(\boldsymbol{\delta}) = \frac{1}{R} \sum_{j=1}^{R} \frac{f(\boldsymbol{\delta} + \mu \mathbf{u}_j, t) - f(\boldsymbol{\delta}, t)}{\mu} \mathbf{u}_j, \tag{2.72}$$

and

$$\hat{\nabla} \log p(t \,|\mathbf{x}, \boldsymbol{\delta}\,) = \frac{1}{R\mu} \sum_{j=1}^{R} [\log p(t \,|\mathbf{x}, \boldsymbol{\delta} + \mu \mathbf{u}_j) - \log p(t \,|\mathbf{x}, \boldsymbol{\delta}\,)] \mathbf{u}_j, \tag{2.73}$$

where R is the number of random direction vectors and $\{\mathbf{u}_j\}$ denote independent and identically distributed (i.i.d.) random direction vectors following Gaussian distribution.

We note that in each gradient estimation step, by querying the model $R + 1$ times, we can *simultaneously* obtain both the $\hat{\nabla} f(\boldsymbol{\delta})$ and $\hat{\nabla} \log p(t \,|\mathbf{x}, \boldsymbol{\delta}\,)$ as demonstrated in Algorithm 1. Different from the zeroth-order gradient descent which only estimates the gradients of the loss function $\hat{\nabla} f(\boldsymbol{\delta})$ (such as [28] and [70]), ZO-NGD obtains $\hat{\nabla} \log p(t \,|\mathbf{x}, \boldsymbol{\delta}\,)$ and computes the FIM from the same query outputs without incurring additional query complexity. This is one major difference between ZO-NGD and other zeroth-order methods. Thus, higher query-efficiency can be achieved by leveraging the FIM and second-order optimization.

2.5.2.4 Damping for Fisher Information Matrix

The inverse of the FIM is required for natural gradient. However, the eigenvalue distribution of the FIM is known to have an extremely long tail [74], where most of the eigenvalues are close to zero. This in turn causes the eigenvalues of the inverse FIM to be extremely large, leading to the unstable training. To mitigate this problem, damping technique is used to add a positive value to the diagonal of the FIM to stabilize the training as shown below,

$$\hat{\mathbf{F}} = \hat{\nabla} \log p\,(t \,|\boldsymbol{x}, \boldsymbol{\delta}\,)\, \hat{\nabla} \log p\,(t \,|\boldsymbol{x}, \boldsymbol{\delta}\,)^T + \gamma \mathbf{I}, \tag{2.74}$$

where γ is a constant. As the damping limits the maximum eigenvalue of the inverse FIM, we can restrict the norm of the gradients. This prevents ZO-NGD from moving too far in flat directions.

With the obtained FIM, the perturbation update is

$$\boldsymbol{\delta}_{k+1} = \prod_{\mathbb{S}} \left(\boldsymbol{\delta}_k - \lambda \hat{\mathbf{F}}^{-1} \hat{\nabla} f(\boldsymbol{\delta}_k) \right). \tag{2.75}$$

ZO-NGD tries to extract the Fisher information to perform second-order optimization for faster convergence rate and better query efficiency.

2.5.2.5 Scalability to High Dimensional Datasets

Note that the FIM has a dimension of d^2 where d is the dimension of the input image. On ImageNet dataset which typically contains images with about $270,000$ pixels ($\boldsymbol{x} \in \mathbb{R}^{299\times299\times3}$), the FIM would have billions of elements and thus it is quite difficult to compute or store the FIM, not to mention its inverse. In the application of training DNN models, the Kronecker Factored Approximate Curvature (K-FAC) method [109, 119] is adopted to deal with the difficulty of high dimensions of the DNN model. However, K-FAC methods may not be suitable in the application of finding adversarial examples as the assumption of uncorrelated channels is not valid and thus we can not apply the block diagonalization method for the FIM. Instead, we propose another method to compute $\hat{\mathbf{F}}^{-1}$ of high dimensions as follows. First we have

$$\hat{\nabla} \log p\left(t \,|\boldsymbol{x}, \boldsymbol{\delta}\right) = c\frac{\hat{\nabla} \log p\left(t \,|\boldsymbol{x}, \boldsymbol{\delta}\right)}{\|\hat{\nabla} \log p\left(t \,|\boldsymbol{x}, \boldsymbol{\delta}\right)\|} = c\frac{\hat{s}(\boldsymbol{\delta})}{\|\hat{s}(\boldsymbol{\delta})\|}, \tag{2.76}$$

where $c = \|\hat{s}(\boldsymbol{\delta})\|$. The inverse matrix $\hat{\mathbf{F}}^{-1}$ can be represented as,

$$\hat{\mathbf{F}}^{-1} = \frac{\left(\left(c^2+\gamma\right)^{-1} - \gamma^{-1}\right)}{c^2}\hat{s}\left(\boldsymbol{\delta}\right)\hat{s}\left(\boldsymbol{\delta}\right)^T + \gamma^{-1}\mathbf{I}. \tag{2.77}$$

This can be verified simply by checking their multiplication and we omit the proof here. Then the gradient update $\Delta\boldsymbol{\delta} = \lambda\hat{\mathbf{F}}^{-1}\nabla f(\boldsymbol{\delta})$ in Eq. (2.75) is

$$\begin{aligned}\Delta\boldsymbol{\delta} &= \lambda\left[\frac{\left(\left(c^2+\gamma\right)^{-1} - \gamma^{-1}\right)}{c^2}\hat{s}\left(\boldsymbol{\delta}\right)\hat{s}\left(\boldsymbol{\delta}\right)^T + \gamma^{-1}\mathbf{I}\right]\hat{\nabla} f\left(\boldsymbol{\delta}\right) \\ &= \lambda\frac{\left(\left(c^2+\gamma\right)^{-1} - \gamma^{-1}\right)}{c^2}\hat{s}\left(\boldsymbol{\delta}\right)\left[\hat{s}\left(\boldsymbol{\delta}\right)^T\hat{\nabla} f\left(\boldsymbol{\delta}\right)\right] \\ &\quad + \lambda\gamma^{-1}\hat{\nabla} f\left(\boldsymbol{\delta}\right).\end{aligned} \tag{2.78}$$

During the computation of $\Delta\boldsymbol{\delta} = \lambda\hat{\mathbf{F}}^{-1}\nabla f(\boldsymbol{\delta})$, we compute $\hat{s}\left(\boldsymbol{\delta}\right)^T\hat{\nabla} f\left(\boldsymbol{\delta}\right)$ first in Eq. (2.78) and obtain a scalar, then $\Delta\boldsymbol{\delta}$ is simply the sum of two vectors. Although $\hat{\mathbf{F}}$ and its inverse might have billions of elements, we avoid directly computing them and the dimension of the internal computation is at most the same level as the dimension d of the images, rather than its square d^2. Thus, the ZO-NGD method can be applied on datasets with high dimensional images.

2.5.2.6 Geometric Interpretation

We provide a geometric interpretation for the natural gradient here. The negative gradient $-\nabla f(\boldsymbol{\delta})$ can be interpreted as the steepest descent direction in the sense that it yields the most reduction in f per unit of change of $\boldsymbol{\delta}$, where the change is measured by the standard Euclidean norm

$\|\cdot\|$ [108], as shown below,

$$\frac{-\nabla f(\boldsymbol{\delta})}{\|\nabla f(\boldsymbol{\delta})\|} = \lim_{\epsilon \to 0} \frac{1}{\epsilon} \arg\min_{\|\alpha\| \leq \epsilon} f(\boldsymbol{\delta} + \boldsymbol{\alpha}). \tag{2.79}$$

By following the $-\nabla f(\boldsymbol{\delta})$ direction, we can obtain the change of $\boldsymbol{\delta}$ within a certain ϵ-neighbourhood to minimize the loss function.

As the loss function is related to the likelihood, we can explore the steepest direction to minimize the loss in the space of all possible likelihood (i.e., distribution space). First we need to decide a distance metric in the distribution space. KL divergence [80] is a popular choose to measure the distance between two distributions, although it is only approximately symmetric within a local neighbourhood.

For two distributions $p\left(t|\boldsymbol{\delta}\right)$ and $p\left(t|\boldsymbol{\delta}'\right)$, their KL divergence is defined as

$$\mathrm{KL}\left[p\left(t|\boldsymbol{\delta}\right)||p\left(t|\boldsymbol{\delta}'\right)\right] = \int p\left(t|\boldsymbol{\delta}\right) \log \frac{p\left(t|\boldsymbol{\delta}\right)}{p\left(t|\boldsymbol{\delta}'\right)} dt. \tag{2.80}$$

Lemma 2.5.2. *The FIM $\boldsymbol{F}$ is the Hessian of KL divergence between two distributions $p(t|\boldsymbol{\delta})$ and $p(t|\boldsymbol{\delta} + \boldsymbol{\alpha})$, with respect to $\boldsymbol{\alpha}$, evaluated at $\boldsymbol{\alpha} = \mathbf{0}$.*

Proof.

$$\begin{aligned}
\nabla_\alpha \mathrm{KL}\left[p(t|\delta)||p(t|\delta+\alpha)\right] &= \nabla_\alpha \int p(t|\delta) \log \frac{p(t|\delta)}{p(t|\delta+\alpha)} dt \\
&= \nabla_\alpha \int p(t|\delta) \log p(t|\delta) dt \\
&\quad - \nabla_\alpha \int p(t|\delta) \log p(t|\delta+\alpha) dt \\
&= -\nabla_\alpha \int p(t|\delta) \log p(t|\delta+\alpha) dt \\
&= -\int p(t|\delta) \nabla_\alpha \log p(t|\delta+\alpha) dt
\end{aligned} \tag{2.81}$$

$$\nabla_\alpha^2 \mathrm{KL}\left[p(t|\delta)||p(t|\delta+\alpha)\right] = -\int p(t|\delta) \nabla_\alpha^2 \log p(t|\delta+\alpha) dt \tag{2.82}$$

The Hessian of the KL divergence is defined by

$$\begin{aligned}
\mathbf{H}_{\mathrm{KL}[p(t|\delta)||p(t|\delta+\alpha)]} &= -\int p(t|\delta) \nabla_\alpha^2 \log p(t|\delta+\alpha)|_{\alpha=\mathbf{0}} dt \\
&= -\int p(t|\delta) \mathbf{H}_{\log p(t|\delta)} dt \\
&= -\underset{p(t|\delta)}{E}\left[\mathbf{H}_{\log p(t|\delta)}\right] \\
&= \mathbf{F}
\end{aligned} \tag{2.83}$$

□

By Lemma 2.5.2, the FIM can be regarded as the curvature in the distribution space.

Lemma 2.5.3. *The second order Taylor expansion of the KL divergence can be expressed as*

$$\mathrm{KL}\left[p(t|\boldsymbol{\delta})||p(t|\boldsymbol{\delta}+\boldsymbol{\alpha})\right] \approx \frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{F}\boldsymbol{\alpha}. \tag{2.84}$$

Proof.

$$\begin{aligned}
\mathrm{KL}\left[p(t|\delta)||p(t|\delta+\alpha)\right] \approx\ & \mathrm{KL}\left[p(t|\delta)||p(t|\delta)\right] \\
& + \left(\nabla_{\alpha}\mathrm{KL}\left[p(t|\delta)||p(t|\delta+\alpha)\right]|_{\alpha=\mathbf{0}}\right)^{\mathrm{T}}\alpha \\
& + \frac{1}{2}\alpha^{\mathrm{T}}\mathbf{F}\alpha \\
=\ & \mathrm{KL}\left[p(t|\delta)||p(t|\delta)\right] \\
& - \underset{p(t|\delta)}{E}\left[\nabla_{\delta}\log p(t|\delta)\right]^{\mathrm{T}}\alpha \\
& + \frac{1}{2}\alpha^{\mathrm{T}}\mathbf{F}\alpha
\end{aligned} \tag{2.85}$$

Notice that the first term is zero as they are the same distributions. The second term is zero due to Lemma 2.5.1. □

Next we explore the direction to minimize the loss function in the distribution space where the distance is measured by the KL divergence. Although in general, the KL divergence is not symmetric, it is (approximately) symmetric in a local neighborhood. The problem can be formulated as

$$\boldsymbol{\alpha}^{*} = \underset{\mathrm{KL}[p(t|\boldsymbol{\delta})||p(t|\boldsymbol{\delta}+\boldsymbol{\alpha})]=m}{\arg\min} f(\boldsymbol{\delta}+\boldsymbol{\alpha}). \tag{2.86}$$

where m is a certain constant. The purpose of fixing the KL divergence to a constant is to move along the distribution space with a constant speed, regardless of the curvature.

Lemma 2.5.4. *The negative natural gradient is the steepest descent direction in the distribution space.*

Proof. The Lagrangian function of the minimization can be formulated as

$$\begin{aligned}
L(\alpha) &= f(\delta+\alpha) + \phi\left(\mathrm{KL}\left[p(t|\delta)||p(t|\delta+\alpha)\right] - c\right) \\
&\approx f(\delta) + \nabla f(\delta)^{\mathrm{T}}\alpha + \frac{1}{2}\phi\alpha^{\mathrm{T}}\mathbf{F}\alpha - \phi c
\end{aligned} \tag{2.87}$$

To solve this minimization, we set its derivative to zero:

$$\frac{\partial}{\partial \alpha}\left(f(\delta)+\nabla_\delta f(\delta)^{\mathrm{T}}\alpha+\frac{1}{2}\varphi\alpha^{\mathrm{T}}\mathbf{F}\alpha-\varphi c\right)=0 \tag{2.88}$$

$$\nabla_\delta f(\delta)+\varphi\mathbf{F}\alpha=0 \tag{2.89}$$

$$\alpha=-\frac{1}{\varphi}\mathbf{F}^{-1}\nabla_\delta f(\delta) \tag{2.90}$$

We can see that the negative natural gradient defines the steepest direction in distribution space. □

In the parameter space, the negative gradient is the steepest descent direction to minimize the loss function. By contrast, in the distribution space, the steepest descent direction is the negative natural gradient. Thus, the direction in distribution space defined by the natural gradient will be invariant to the choice of parameterization [130], i.e., it will not be affected by how the model is parametrized, but only depends on the distribution induced by the parameters.

2.5.3 Experimental Results

In this section, we present the experimental results of the ZO-NGD method. We compare ZO-NGD with various attack methods on three image classification datasets, MNIST [85], CIFAR-10 [79] and ImageNet [36].

We train two networks for MNIST and CIFAR-10 datasets, respectively. The model for MNIST achieves 99.6% accuracy with four convolutional layers, two max pooling layers, two fully connected layers and a softmax layer. For CIFAR-10, we adopt the same model architecture as MNIST, achieving 80% accuracy. For ImageNet, a pre-trained Inception v3 network [147] is utilized instead of training our own model, attaining 96% top-5 accuracy. All experiments are performed on machines with NVIDIA GTX 1080 TI GPUs.

2.5.3.1 Evaluation of White-box Attack

We first check the white-box setting, where we compare the proposed NGD with PGD from adversarial training. PGD is a typical first-order method while NGD utilizes the second-order FIM. The query here is defined as one forward pass and one subsequent backpropagation as we need to obtain the gradients through backpropagation. We report the average number of queries over 500 images for successful adversaries on each dataset. On MNIST, NGD requires 2.12 queries while PGD needs 4.88 queries with $\epsilon = 0.2$. On CIFAR-10, NGD requires 2.06 queries while PGD needs 4.21 queries with $\epsilon = 0.1$. On ImageNet, NGD requires 2.20 queries while PGD needs 5.62 queries with $\epsilon = 0.05$. We can see that NGD achieves higher query efficiency by incorporating FIM.

Table 2.10: Performance evaluation of black-box adversarial attacks on MNIST and CIFAR-10.

dataset	Attack method	success rate	average queries	reduction rate
MNIST	Transfer attack	82%	-	-
	ZOO attack	100%	8,300	0%
	NES-PGD	98.2%	1,243	85%
	ZO-NGD	98.7%	523	93.7%
CIFAR	Transfer attack	85%	-	-
	ZOO attack	99.8 %	6,500	0%
	NES-PGD	98.9%	417	93.6%
	ZO-NGD	99.2 %	131	98%

2.5.3.2 Evaluation on MNIST and CIFAR-10

In the evaluation on MNIST and CIFAR-10, we select 2000 correctly classified images from MNIST and CIFAR-10 test datasets, respectively, and perform black-box attacks for these images. We compare the ZO-NGD method with the transfer attack [126], ZOO black-box attack [28], and the natural-evolution-strategy-based projected gradient descent method (NES-PGD) [70]. For the transfer attack [126], we apply C&W attack [24] to the surrogate model. For the attack methods, the pixel values of all images are normalized to the range of $[0, 1]$. In the proposed ZO-NGD method, the sampling number R in the random gradient estimation as defined in Eq. (2.72) and (2.73) is set to 40. ϵ is set to 0.4 for MNIST and 0.2 for CIFAR-10 or ImageNet. In Eq. (2.72) and (2.73), we set $\mu = 1$ for three datasets. γ is set to 0.01.

The experimental results are summarized in Table 2.10. We show the success rate and the average queries over successful adversarial examples for the black-box attack methods on MNIST and CIFAR-10 datasets. As shown in Table 2.10, the transfer attack does not achieve high success rate due to the difference between the surrogate model and the original target model. The ZOO attack method can achieve high success rate at the cost of excessive query complexity since it performs gradient estimation for each pixel of the input image. We can observe that the ZO-NGD method requires significantly less queries than the NES-PGD method. NES-PGD uses natural evolutionary strategies for gradient estimation and then perform first-order gradient descent to obtain the adversarial perturbations. Compared with NES-PGD, the proposed ZO-NGD not only

Table 2.11: Performance evaluation of black-box adversarial attacks on ImageNet.

Attack method	success rate	average queries	reduction ratio
ZOO attack	98.9%	16,800	0%
NES-PGD	94.6%	1,325	92.1%
Bandits[TD]	96.1%	791	95.3%
ZO-NGD	97%	582	96.5%

estimates the first-order gradients of the loss function, but also tries to obtain the second-order Fisher information from the queries without incurring additional query complexities, leading to higher query-efficiency. From Table 2.10, we can observe that the ZO-NGD method attains the smallest number of queries to successfully obtain the adversarial examples in the black-box setting. Benchmarking on the ZOO method, the query reduction ratio of ZO-NGD can be as high as 93.7% on MNIST and 98% on CIFAR-10.

2.5.3.3 Evaluation on ImageNet

We perform black-box adversarial attacks on ImageNet where 1000 correctly classified images are randomly selected. On ImageNet, we compare the proposed ZO-NGD with the ZOO attack, NES-PGD method and the bandit attack with time and data-dependent priors (named as Bandits[TD]) [71]. The transfer attack is not performed since it is not easy to train a surrogate model on ImageNet. The Bandits[TD] method makes use of the prior information for the gradients estimation, including the time-dependent priors which explores the heavily correlated successive gradients, and the data-dependent priors which exploits the spatially local similarity exhibited in images. After gradient estimation with the priors or bandits information, first-order gradient descent method is applied.

We present the performance evaluation on ImageNet in Table 2.11. The success rate and the average queries over successful attacks for various black-box attack methods are reported. Table 2.11 shows the ZOO attack method can achieve high success rate with high query complexity due to its element-wise gradient estimation. We can have a similar observation that the ZO-NGD method only requires a much smaller number of queries than the NES-PGD method due to the faster convergence rate of second-order optimization by exploring the Fisher information. We also find that

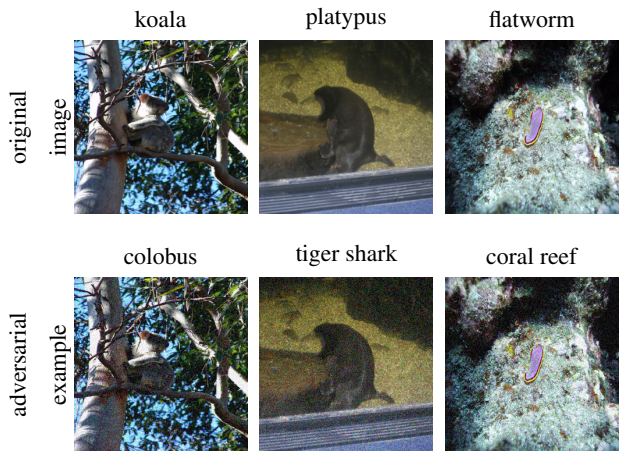


Figure 2.3: The legitimate images and their adversarial examples generated by ZO-NGD.

the ZO-NGD method also outperforms the Bandits[TD] method in terms of query efficiency. The Bandits[TD] method enhances the query efficiency of gradient estimations through the incorporation of priors information for the gradients, but its attack methodology is still based on the first-order gradient descent method. As observed from Table 2.11, the ZO-NGD method achieves the highest query-efficiency for successful adversarial attacks in the black-box setting. It can obtain 96.5% query reduction ratio on ImageNet when compared with the ZOO method. In Figure 2.3, we show some legitimate images on ImageNet and their corresponding adversarial examples obtained by ZO-NGD. We can observe that the adversarial perturbations are imperceptible. More examples on MNIST and CIFAR-10 are shown in the appendix.

2.5.3.4 Ablation study

In this ablation study, we perform sensitivity analysis on the proposed ZO-NGD method based variations in model architectures and different parameter settings. Below we summarize the conclusion and findings from this ablation study and report their details in the appendix. (1) Tested on VGG16 and ResNet and varying the parameters μ and ϵ in ZO-NGD, the results demonstrate the consistent superior performance of ZO-NGD by leveraging the second-order optimization. (2) We inspect the approximation techniques used in ZO-NGD including damping and outer product method. The results show that there is a wide range of proper γ values such that damping can work effectively to reduce the loss, and the outer product is a reasonable approximation based on the empirical evidence. We also note that the ASR of ZOO is higher than ZO-NGD. We provide a

discussion about the ASR v.s. query number in the appendix.

2.5.3.5 Query Number Distribution

Figure 2.4 shows the cumulative distribution (CDF) of the query number for 1000 images on three datasets, validating ZO-NGD's query efficiency.

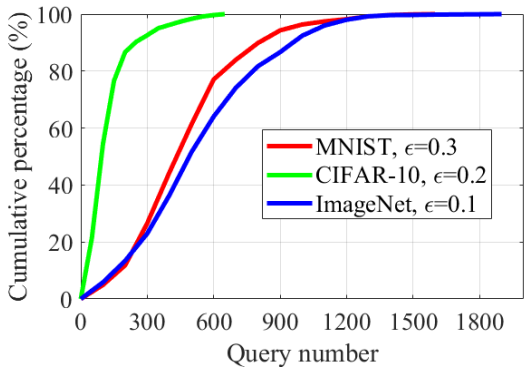


Figure 2.4: CDF of query number on three datasets using ZO-NGD.

2.5.3.6 Transferability

The transferability of adversarial examples is an interesting and valuable metric to measure the performance. To show the transferability, we use 500 targeted adversarial examples generated by ZO-NGD on ImageNet with $\epsilon = 0.1$ on Inception to attack ResNet and VGG16 model. It achieves 94.4% and 95.6% ASR, respectively, demonstrating high transferability of our method. Our transferred ASR is also higher than NES-PGD (92.1% and 92.9% ASR).

2.6 Fault Sneaking Attack

2.6.1 Motivation

2.6.2 Problem Formulation

Threat Model: We consider an adversary tampering with the DNN classification results of certain input images into designated target labels by modifying the DNN model parameters. In this paper, we assume white-box attack, i.e., the adversary has the complete knowledge of the DNN model (including both structures and parameters) and low-level implementation details (how and where

DNN parameters are located in the memory), as the highest and most stringent security standard to assess the robustness of DNN systems under fault sneaking attack. Given existing fault injection techniques can precisely flip any bit of the data in memory, we assume the adversary can modify any parameter in DNN to any value that is in the valid range of the used arithmetic format. Note that, we do not assume the adversary knows the training and testing data sets, which are usually not available to the system users.

The adversary has two constraints when launching the fault sneaking attack: (i) Stealthy, in that the classification results of the other images should be kept as unchanged as possible; (ii) Efficient, in that the modifications of DNN parameters in terms of number of modified parameters or magnitude of parameter modifications should be as small as possible. The first constraint is important because even if the attack is specified for certain input images, it is highly possible to change the classification results of the other images when modifying the DNN parameters, thereby resulting in obviously low DNN model accuracy and easy detection of the attack. The second constraint minimizing the parameter modifications can reduce the influence and difficulty of implementing the attack.

Attack Model: Given R images $\mathcal{X} = \{\boldsymbol{x}_i | i = 1, \cdots, R\}$ with their correct labels $\mathcal{L} = \{l_i | i = 1, \cdots, R\}$, we would like to change the classification results of the first S ($S \leq R$) images to their target labels $\mathcal{T} = \{t_i | i = 1, \cdots, S\}$, while the classifications of the rest $R - S$ images are unchanged, by modifying parameters in the DNN model. Note that the unchanged labels of the other $R - S$ images are to make the attack stealthy and hard to detect.

The original DNN model parameters are denoted as $\boldsymbol{\theta}$, and $\boldsymbol{\delta}$ represents the parameter modifications. So the parameters after the modification are $\boldsymbol{\theta} + \boldsymbol{\delta}$. Note that $\boldsymbol{\theta}$ has the flexibility of specifying either all the DNN parameters or only a portion of the parameters, e.g., weight parameters of the specific layer(s). The fault sneaking attack can be formulated as an optimization problem:

$$\min_{\boldsymbol{\delta}} \quad D(\boldsymbol{\delta}) + G(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L}), \tag{2.91}$$

where $D(\boldsymbol{\delta})$ measures the DNN parameter modifications; and $G(\boldsymbol{\theta} + \boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L})$ represents the misclassification requirements, i.e., with the modified DNN model parameters $\boldsymbol{\theta} + \boldsymbol{\delta}$, the first S images in set $\mathcal{X}$ will be classified as target labels $\mathcal{T}$, while the classifications of the rest $R - S$ images are kept unchanged. The details of the D and G functions are to be explained in the following sections.

2.6.2.1 Measurements of Parameter Modifications

$D(\boldsymbol{\delta})$ represents the measurement of the parameter modifications, which should be minimized for the attack implementation efficiency. In this paper, ℓ_0 and ℓ_2 norms are used as $D(\boldsymbol{\delta})$ as follows,

$$D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_0 \quad \text{or} \quad D(\boldsymbol{\delta}) = \|\boldsymbol{\delta}\|_2. \tag{2.92}$$

The ℓ_0 norm of $\boldsymbol{\delta}$ measures the number of nonzero elements in $\boldsymbol{\delta}$ and therefore measures the number of modified parameters by the attack. Minimizing ℓ_0 norm can make it easier to implement the attack in DNN systems, considering that the difficulty of parameter modifications in real systems relates to the number of modified parameters [76]. The ℓ_2 norm of $\boldsymbol{\delta}$ denotes the standard Euclidean distance between the modified and original parameters, and therefore measures the magnitude of parameter modifications. Minimizing ℓ_2 norm can lead to minimal influence of the attack.

Minimizing the ℓ_0 norm in the objective function is much harder than minimizing the ℓ_2 norm, because the ℓ_0 norm is non-differential. In this paper, the proposed ADMM framework enables both ℓ_0 and ℓ_2 norms in the objective function with only minor differences in the solution methods as specified in Sec. 2.6.3.3.

2.6.2.2 Misclassification Requirements

In Eq. (2.91), $G(\boldsymbol{\theta}+\boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L})$ denotes the misclassification requirements: 1) the first S images $\mathcal{X}_1 = \{\boldsymbol{x}_i | i = 1, \cdots, S\}$ should be classified as the target labels $\mathcal{T}$ instead of their correct labels, and 2) the classifications of the rest $R - S$ images $\mathcal{X}_2 = \{\boldsymbol{x}_i | i = S+1, \cdots, R\}$ should remain unchanged as their correct labels.

In the area of adversarial machine learning, the most effective objective function to specify that an input $\boldsymbol{x}$ should be labeled as t is the following g function [24]:

$$g(\boldsymbol{\theta}+\boldsymbol{\delta}, t) = \max\left(\max_{j \neq t}(Z(\boldsymbol{\theta}+\boldsymbol{\delta}, \boldsymbol{x})_j) - Z(\boldsymbol{\theta}+\boldsymbol{\delta}, \boldsymbol{x})_t, 0\right) \tag{2.93}$$

where $Z(\boldsymbol{\theta}+\boldsymbol{\delta}, \boldsymbol{x})_j$ denotes the j-th element of the logits, i.e., the input to the softmax layer. The softmax layer is the last layer in the DNN model, which takes logits as input and generates the final probability distribution outputs. The final outputs from the softmax layer are not utilized in the above g function, because the final outputs are usually dominated by the most significant class in a well trained model and thus less effective during computation. The DNN chooses the label with the largest logit, that is, $j^* = \arg\max_j \ Z(\boldsymbol{\theta}+\boldsymbol{\delta}, \boldsymbol{x})_j$. To enforce the input $\boldsymbol{x}$ is classified as label t, the

logit of label t, $Z(\boldsymbol{\theta}+\boldsymbol{\delta},\boldsymbol{x})_t$, must be larger than all of the other logits, $\max_{j\neq t}(Z(\boldsymbol{\theta}+\boldsymbol{\delta},\boldsymbol{x})_j)$. Thus, $g(\boldsymbol{\theta}+\boldsymbol{\delta},t)$ will achieve its minimal value if $\boldsymbol{x}$ is classified as label t.

From the above analysis, we propose the detailed form of G as:

$$G(\boldsymbol{\theta}+\boldsymbol{\delta},\mathcal{X},\mathcal{T},\mathcal{L}) = G_1(\boldsymbol{\theta}+\boldsymbol{\delta},\mathcal{X}_1,\mathcal{T}) + G_2(\boldsymbol{\theta}+\boldsymbol{\delta},\mathcal{X}_2,\mathcal{L}), \tag{2.94}$$

where G_1 stands for the targeted misclassifications of $\mathcal{X}_1$ and G_2 denotes keeping classifications of $\mathcal{X}_2$ unchanged. G_1 and G_2 are:

$$\begin{aligned} &G_1(\boldsymbol{\theta}+\boldsymbol{\delta},\mathcal{X}_1,\mathcal{T}) \\ &= \sum_{i=1}^{S} c_i \cdot \max\left(\max_{j\neq t_i}(Z(\boldsymbol{\theta}+\boldsymbol{\delta},\boldsymbol{x}_i)_j) - Z(\boldsymbol{\theta}+\boldsymbol{\delta},\boldsymbol{x}_i)_{t_i}, 0\right), \end{aligned} \tag{2.95}$$

$$\begin{aligned} &G_2(\boldsymbol{\theta}+\boldsymbol{\delta},\mathcal{X}_2,\mathcal{L}) \\ &= \sum_{i=S+1}^{R} c_i \cdot \max\left(\max_{j\neq l_i}(Z(\boldsymbol{\theta}+\boldsymbol{\delta},\boldsymbol{x}_i)_j) - Z(\boldsymbol{\theta}+\boldsymbol{\delta},\boldsymbol{x}_i)_{l_i}, 0\right). \end{aligned} \tag{2.96}$$

The c_i's represent their relative importance to the measurement of modifications $D(\boldsymbol{\delta})$. t_i represents the target label for the i-th image in the S images. G_1 achieves its minimum value, when the labels of the first S images are changed to their target labels $\mathcal{T}$. Similarly, G_2 obtains its minimum value when the classifications of the rest $R-S$ images are kept unchanged.

2.6.3 ADMM Framework

We propose a solution framework based on ADMM to solve problem (2.91) for the fault sneaking attack. The framework is *general* in that it can deal with both ℓ_0 and ℓ_2 norms as $D(\boldsymbol{\delta})$. ADMM was first introduced in the mid-1970s with roots in the 1950s and becomes popular recently for large scale statistics and machine learning problems [18]. ADMM solves the problems in the form of a decomposition-alternating procedure, where the global problem is split into local subproblems first, and then the solutions to small local subproblems are coordinated to find a solution to the large global problem. It has been proved in [67] that ADMM has at least the linear convergence rate, and it empirically converges in a few tens of iterations.

2.6.3.1 ADMM Reformulation

As ADMM requires multiple variables for reducing the objective function in alternating directions, we introduce a new auxiliary variable $\boldsymbol{z}$ and problem (2.91) can now be reformulated as,

$$\begin{aligned} \min_{\boldsymbol{\delta},\boldsymbol{z}} \quad & D(\boldsymbol{z}) + G(\boldsymbol{\theta}+\boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L}), \\ \text{s.t.} \quad & \boldsymbol{z} = \boldsymbol{\delta}. \end{aligned} \tag{2.97}$$

The augmented Lagrangian function of the above problem is:

$$L_\rho(\boldsymbol{\delta}, \boldsymbol{z}, \boldsymbol{u}) = D(\boldsymbol{z}) + G(\boldsymbol{\theta}+\boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L}) + \boldsymbol{u}^T(\boldsymbol{z}-\boldsymbol{\delta}) + \frac{\rho}{2}\left\|\boldsymbol{z}-\boldsymbol{\delta}\right\|_2^2. \tag{2.98}$$

Applying the scaled form of ADMM by defining $\boldsymbol{u} = \rho\boldsymbol{s}$, we obtain

$$L_\rho(\boldsymbol{\delta}, \boldsymbol{z}, \boldsymbol{s}) = D(\boldsymbol{z}) + G(\boldsymbol{\theta}+\boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L}) + \frac{\rho}{2}\left\|\boldsymbol{z}-\boldsymbol{\delta}+\boldsymbol{s}\right\|_2^2 - \frac{\rho}{2}\left\|\boldsymbol{s}\right\|_2^2. \tag{2.99}$$

2.6.3.2 ADMM Iterations

ADMM optimizes problem (2.99) in iterations. Specifically, in the k-th iteration, the following steps are performed:

$$\boldsymbol{z}^{k+1} = \arg\min_{\boldsymbol{z}} L_\rho(\boldsymbol{\delta}^k, \boldsymbol{z}, \boldsymbol{s}^k), \tag{2.100}$$

$$\boldsymbol{\delta}^{k+1} = \arg\min_{\boldsymbol{\delta}} L_\rho(\boldsymbol{\delta}, \boldsymbol{z}^{k+1}, \boldsymbol{s}^k), \tag{2.101}$$

$$\boldsymbol{s}^{k+1} = \boldsymbol{s}^k + \boldsymbol{z}^{k+1} - \boldsymbol{\delta}^{k+1}. \tag{2.102}$$

As demonstrated above, problem (2.99) is split into two subproblems, Eq. (2.100) and Eq. (2.101) through ADMM. In Eq. (2.100), the optimal solution $\boldsymbol{z}^{k+1}$ is obtained by minimizing the augmented Lagrangian function $L_\rho(\boldsymbol{\delta}^k, \boldsymbol{z}, \boldsymbol{s}^k)$ with fixed $\boldsymbol{\delta}^k$ and $\boldsymbol{s}^k$. Similarly, Eq. (2.101) finds the optimal $\boldsymbol{\delta}^{k+1}$ to minimize $L_\rho(\boldsymbol{\delta}, \boldsymbol{z}^{k+1}, \boldsymbol{s}^k)$ with fixed $\boldsymbol{z}^{k+1}$ and $\boldsymbol{s}^k$. In Eq. (2.102), we update $\boldsymbol{s}^{k+1}$ with $\boldsymbol{z}^{k+1}$ and $\boldsymbol{\delta}^{k+1}$. We can observe that ADMM updates the two arguments in an alternating fashion, where comes from the term *alternating direction*.

In the ADMM iterations, Eq. (2.100) and Eq. (2.101) are detailed as:

$$\min_{\boldsymbol{z}} \quad D(\boldsymbol{z}) + \frac{\rho}{2}\left\|\boldsymbol{z}-\boldsymbol{\delta}^k+\boldsymbol{s}^k\right\|_2^2, \tag{2.103}$$

$$\min_{\boldsymbol{\delta}} \quad G(\boldsymbol{\theta}+\boldsymbol{\delta}, \mathcal{X}, \mathcal{T}, \mathcal{L}) + \frac{\rho}{2}\left\|\boldsymbol{z}^{k+1}-\boldsymbol{\delta}+\boldsymbol{s}^k\right\|_2^2. \tag{2.104}$$

The solutions to the two problems are specified as follows.

2.6.3.3 z step

In this step, we mainly solve Eq. (2.103). The specific closed-form solution depends on the D function (ℓ_0 or ℓ_2 norm).

Solution for ℓ_0 norm If the D function takes the ℓ_0 norm, Eq. (2.103) has the following form:

$$\min_{\boldsymbol{z}} \quad \|\boldsymbol{z}\|_0 + \frac{\rho}{2}\left\|\boldsymbol{z} - \boldsymbol{\delta}^k + \boldsymbol{s}^k\right\|_2^2. \tag{2.105}$$

The solution can be obtained elementwise [127] as

$$\boldsymbol{z}_i^{k+1} = \begin{cases} \left(\boldsymbol{\delta}^k - \boldsymbol{s}^k\right)_i, & \text{if } \left(\boldsymbol{\delta}^k - \boldsymbol{s}^k\right)_i^2 > \frac{2}{\rho} \\ 0, & \text{otherwise} \end{cases}. \tag{2.106}$$

Solution for ℓ_2 norm If the D function takes the ℓ_2 norm, Eq. (2.103) has the following form:

$$\min_{\boldsymbol{z}} \quad \|\boldsymbol{z}\|_2 + \frac{\rho}{2}\left\|\boldsymbol{z} - \boldsymbol{\delta}^k + \boldsymbol{s}^k\right\|_2^2. \tag{2.107}$$

By 'block soft thresholding' operator [127], the solution is given by

$$z^{k+1} = \begin{cases} \left(1 - \frac{1}{\rho\left\|\delta^k - s^k\right\|_2}\right)\left(\delta^k - s^k\right) & \text{if } \left\|\delta^k - s^k\right\|_2 \geq \frac{1}{\rho} \\ 0 & \text{if } \left\|\delta^k - s^k\right\|_2 < \frac{1}{\rho} \end{cases}. \tag{2.108}$$

2.6.3.4 δ step

In this step, we mainly solve Eq. (2.104). It can be rewritten as

$$\min_{\boldsymbol{\delta}} \quad \sum_{i=1}^{R} g_i(\boldsymbol{\theta} + \boldsymbol{\delta}, \boldsymbol{x}_i) + \frac{\rho}{2}\left\|\boldsymbol{z}^{k+1} - \boldsymbol{\delta} + \boldsymbol{s}^k\right\|_2^2, \tag{2.109}$$

where

$$g_i(\theta + \delta, x_i) = \begin{cases} c_i \cdot \max\left(\max\limits_{j \neq t_i}\left(Z(\boldsymbol{\theta} + \boldsymbol{\delta}, \boldsymbol{x}_i)_j\right) - Z(\boldsymbol{\theta} + \boldsymbol{\delta}, \boldsymbol{x}_i)_{t_i}, 0\right), \\ \qquad\qquad \text{if } i \in [1, S]; \\ c_i \cdot \max\left(\max\limits_{j \neq l_i}\left(Z(\boldsymbol{\theta} + \boldsymbol{\delta}, \boldsymbol{x}_i)_j\right) - Z(\boldsymbol{\theta} + \boldsymbol{\delta}, \boldsymbol{x}_i)_{l_i}, 0\right), \\ \qquad\qquad \text{if } i \in [S+1, R]. \end{cases} \tag{2.110}$$

The g_i function takes different forms according to the i value. If $i \in [1, S]$, g_i obtains its minimum value when the classification of $\boldsymbol{x}_i$ is changed to the target label t_i. If $i \in [S+1, R]$, g_i achieves its minimum when the classification is kept as the original label l_i.

Table 2.12: ℓ_0 norm of DNN parameter modifications (i.e., the number of modified parameters) in different fully connected layers for MNIST.

	Total Parameters	ℓ_0 norm		
		S=1,R=1	S=4,R=4	S=16,R=16
The first FC layer	205000	14016	40649	120597
The second FC layer	40200	5390	14086	34069
The last FC layer	2010	222	682	1755

Motivated by the linearized ADMM [51, 96, Sec. 2.2], we replace the function g_i with its first-order Taylor expansion plus a regularization term (known as Bregman divergence), $\nabla g_i(\boldsymbol{\theta} + \boldsymbol{\delta}^k, \boldsymbol{x}_i, l_i)(\boldsymbol{\delta} - \boldsymbol{\delta}^k) + \frac{1}{2}\left\|\boldsymbol{\delta} - \boldsymbol{\delta}^k\right\|_H^2$, where $\boldsymbol{H}$ is a pre-defined positive definite matrix , and $\|\boldsymbol{x}\|_{\boldsymbol{H}}^2 = \boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x}$. Eq. (2.104) can then be reformulated as:

$$\begin{aligned}\min_{\boldsymbol{\delta}} \quad & \left(\sum_{i=1}^{R} \nabla g_i(\boldsymbol{\theta} + \boldsymbol{\delta}^k, \boldsymbol{x}_i)\right)(\boldsymbol{\delta} - \boldsymbol{\delta}^k) + \frac{R}{2}\left\|\boldsymbol{\delta} - \boldsymbol{\delta}^k\right\|_H^2 \\ & + \frac{\rho}{2}\left\|\boldsymbol{z}^{k+1} - \boldsymbol{\delta} + \boldsymbol{s}^k\right\|_2^2. \end{aligned} \tag{2.111}$$

Letting $\boldsymbol{H} = \alpha\boldsymbol{I}$, the solution can be obtained through

$$\boldsymbol{\delta}^{k+1} = \frac{1}{\alpha R + \rho}\left(\rho\left(\boldsymbol{z}^{k+1} + \boldsymbol{s}^k\right) + \alpha R\boldsymbol{\delta}^k - \left(\sum_{i=1}^{R} \nabla g_i(\boldsymbol{\theta} + \boldsymbol{\delta}^k, \boldsymbol{x}_i)\right)\right). \tag{2.112}$$

2.6.4 Performance Evaluation

We demonstrate the experimental results of the proposed fault sneaking attack on two image classification datasets, MNIST [85] and CIFAR-10 [79]. We train two networks for MNIST and CIFAR-10 datasets, respectively, sharing the same network architecture with four convolutional layers, two max pooling layers, two fully connected layers and one softmax layer. They achieve 99.5% accuracy on MNIST and 79.5% accuracy on CIFAR-10, respectively, which are comparable to the state-of-the-arts. The experiments are conducted on machines with NVIDIA GTX 1080 TI GPUs.

2.6.4.1 Layer and Type of Parameters to Modify

The DNN model used has three fully connected (FC) layers. We modify the parameters in different FC layers. We show the ℓ_0 norm (i.e., the number of parameter modifications) achieved

Table 2.13: ℓ_0 norm and attack success rate when modifying different types of parameters in the last fully connected layer for MNIST.

	S=1, R=1	S=2, R=2	S=4, R=4	S=8, R=8
ℓ_0 norm for weight params.	236	458	715	1644
Success rate for weight params.	100%	100%	100%	100%
ℓ_0 norm for bias params.	2	4	-*	-*
Success rate for bias params.	100%	100%	0%	0%

* There is no need to show the ℓ_0 norm if it can not succeed.

by the fault sneaking attack when we modify each FC layer in Table 2.12. We observe that more parameters are needed to be modified with increasing S and R. Besides, changing the last FC layer requires fewer parameter modifications compared with the first or second FC layer. The reason is that the last FC layer has more direct influence on the output, leading to smaller number of modifications by the fault sneaking attack. Therefore, in the following experiments, we focus on modifying only the last FC layer parameters.

Next we determine the type of parameters to modify that is more effective to implement the fault sneaking attack. In the FC layer, the output depends on the weights $\boldsymbol{W}$ and the biases $\boldsymbol{b}$, that is, $FC(\boldsymbol{x}') = \boldsymbol{W}\boldsymbol{x}' + \boldsymbol{b}$, where $\boldsymbol{x}'$ is the input of the layer. As we can see, the bias parameters are more directly related to the output than the weight parameters. We show the ℓ_0 norm and the attack success rate if we only modify the weight parameters or the bias parameters in the last FC layer in Table 2.13. As the bias parameters are more directly related to the output, it usually needs to change fewer bias parameters to achieve the same attack objective. However, only changing bias parameters has very limited capability which can only lead to the misclassification of 1 or 2 images. As observed from Table 2.13, changing the classification of 4 or more images would be beyond the capability of modifying bias parameters only. This demonstrates the limitation of the single bias attack (SBA) scheme in [100], which only modifies the bias to misclassify only one image. Also we find that SBA can not be extended to solve the case of multiple images with multiple target labels. Considering the limitation of only modifying bias parameters, we choose to perturb both the weight and bias parameters in the following experiments.

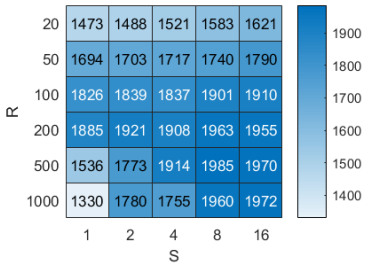


Figure 2.5: ℓ_0 norm of DNN parameter modifications in the last fully connected layer for MNIST.

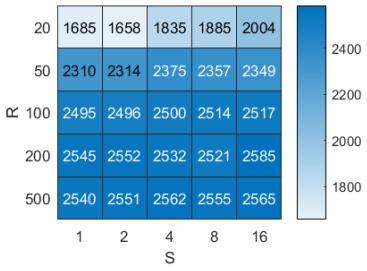


Figure 2.6: ℓ_0 norm of DNN parameter modifications in the last fully connected layer for CIFAR-10.

2.6.4.2 ℓ_0 Norm of Parameter Modifications

We demonstrate the number of parameter modifications, i.e., the ℓ_0 norm, by the fault sneaking attack in this section. As observed from Fig. 2.5 and 2.6, for the same R, the ℓ_0 norm of parameter modifications keeps increasing as S increases since more parameters need to be modified to change the classifications of more images into their target labels. We have an interesting finding that when S is in the range of $\{1, 2, 4\}$, the ℓ_0 norm tends to be smaller as R increases from 200 to 1000 for MNIST. The reason is that larger R means the labels of more images $(R - S)$ need to be kept unchanged, then the modified model should be more similar to the original model and therefore fewer modifications are required.

We also notice that this phenomenon disappears when S is larger than 8 for MNIST or for CIFAR-10. Considering the 99.5% and 79.5% accuracy on MNIST and CIFAR-10, we believe the disappearance is related to the DNN model capability. When S is small on MNIST, the DNN model is able to hide a small number of misclassifications by modifying only a few parameters of the last FC layer. However, when S is relatively large, it is not that easy to hide so many misclassifications and the fault sneaking attack has to perturb almost all parameters in the last FC layer without extra ability to spare. The reason for CIFAR-10 is similar since the capability of the model for CIFAR-10

Table 2.14: ℓ_0 and ℓ_2 norms of DNN parameter modifications in the last fully connected layer for the ℓ_0 and ℓ_2 based attacks for MNIST.

	S=1, R=10		S=5, R=10		S=5, R=20	
	ℓ_0 norm	ℓ_2 norm	ℓ_0 norm	ℓ_2 norm	ℓ_0 norm	ℓ_2 norm
ℓ_0 attack	1026	863	1208	804	1606	498
ℓ_2 attack	1431	393	1432	344	1964	226

is limited, with only 79.5% accuracy.

2.6.4.3 Comparison of ℓ_0 and ℓ_2 based Attacks

In problem (2.100), the ℓ_0 or ℓ_2 norm can be minimized, leading to the corresponding ℓ_0 or ℓ_2 based fault sneaking attacks. Table 2.14 compares the ℓ_0 and ℓ_2 norms of the ℓ_0 and ℓ_2 based attacks for various configurations. As seen from Table 2.14, the ℓ_0 based attack achieves smaller ℓ_0 norm than the ℓ_2 based attack with larger ℓ_2 norm, due to the reason that the ℓ_2 based attack tries to minimize the Euclidean distance between the perturbed and original model without considering the number of parameter modifications.

2.6.4.4 Test Accuracy after Parameter Modification

As the fault sneaking attack perturbs the DNN parameters to satisfy specific attack requirements, it is important to measure the influence of the attack beyond the required objective. In the problem formulation, we try to reduce the influence of fault sneaking attack by enforcing the rest $R - S$ images to have unchanged classifications. In Table 2.15, we show the test accuracy on the whole testing datasets for MNIST and CIFAR-10 after perturbing the model.

The test accuracy of the original model is 99.5% for MNIST and 79.5% for CIFAR. As observed from Table 2.15, with fixed R, the test accuracy on the modified model decreases as S increases. This demonstrates that as a nature outcome, changing parameters to misclassify certain images may downgrade the overall accuracy performance of the model. In the case of $S = 16$ and $R = 50$, the test accuracy drops from 99.5% to 29.7% for MNIST and from 79.5% to 18.3% for CIFAR. However, we observe that as R increases, the test accuracy keeps increasing for fixed S. It demonstrates that keeping the labels of the $R - S$ images unchanged helps to stabilize the model and reduce the influence of changing the labels of the S images. In the case of $S = 16$, if R is increased

Table 2.15: Test accuracy after DNN parameter modifications for MNIST and CIFAR.

Dataset	Test Acc.	S=1	S=2	S=4	S=8	S=16
MNIST	R=50	85.2%	73.1%	64.7%	37.4%	29.7%
	R=100	96.9%	86.6%	81.3%	76.1%	65.2%
	R=200	96.7%	96.1%	95.4%	93.2%	92.6%
	R=500	98.6%	98.5%	97.8%	96.9%	95.9%
	R=1000	98.7%	97.9%	98.1%	96.8%	96.9%
CIFAR	R=50	57.7%	52.9%	44.9%	26.2%	18.3%
	R=100	67.5%	68.7%	55.8%	42.5%	31.5%
	R=200	72.3%	67.6%	69.6%	57.2%	35.4%
	R=500	78.5%	77.4%	76.2%	74.5%	73.2%
	R=1000	78.5%	78.2%	77.5%	77.9%	76.4%

from 50 to 1000, the test accuracy on the 10,000 test images increases from 29.7% to 96.9% for MNIST and from 18.3% to 76.4% for CIFAR. The fault sneaking attack can achieve classification accuracy as high as 98.7% and 78.5% in the case of $S = 1$ and $R = 1000$ for MNIST and CIFAR, which only degrades the accuracy by 0.8 percent and 1.0 percent respectively, from the original models. Note that under the same assumption of misclassifying only one image, [100] degrades the accuracy by 3.86 percent and 2.35 percent, respectively, for MNIST and CIFAR in the best case. Compared with [100], the proposed attack achieves a great improvement to reduce the influence of model perturbation.

2.6.4.5 Tolerance for Sneaking Faults

One objective of fault sneaking attack is to hide faults by perturbing the DNN parameters. In the experiments, we found that in the case of large S, not all of the S images are changed to their target labels successfully. We define the success rate of the S images as the percentage of images successfully changed their labels to the target labels within the S images. We show the success rate of the S images with various S and R configurations in Fig. 2.7. We observe that the success rate keeps almost 100% if S is smaller than 10. When S is larger than 10, the success rate would drop as S increases. Besides, the number of successful injected faults in S is usually around 10 for different configuration of S. This demonstrates a limitation of changing the classifications of certain

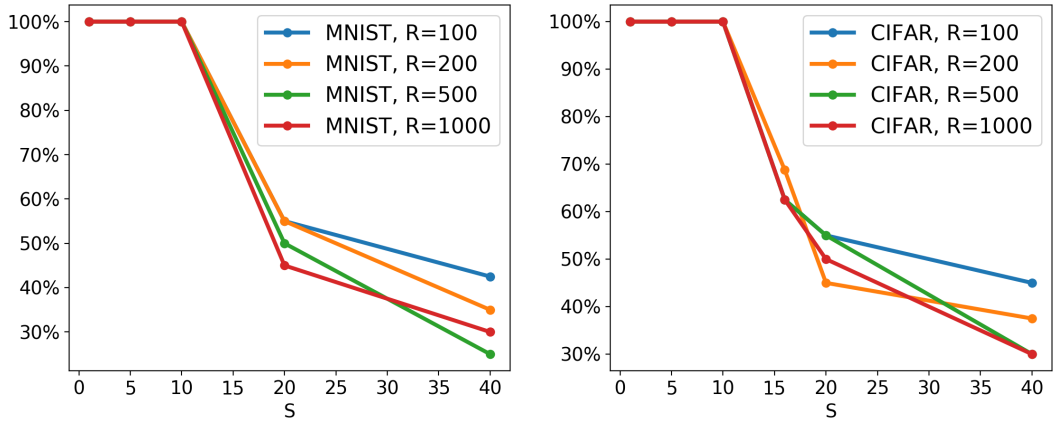


Figure 2.7: Fault sneaking attack success rate of the S images after DNN parameter modifications for MNIST and CIFAR.

images by modifying DNN parameters. The DNN model has a tolerance for the sneaking faults - 10 successful misclassifications by modifying the last FC layer.

Chapter 3

DNN Robustness Evaluation

3.1 Motivation

Although deep neural networks (DNNs) have achieved human-level performance in many learning tasks, Szegedy et al. [149] and subsequent work [27, 57, 111, 116] made the discovery of *adversarial examples* against DNNs. An ever-increasing amount of research effort has been devoted to implementing adversarial attacks in various applications [6, 24, 25, 124, 144, 191] and defense methods ranging from heuristic methods to provable approaches with certain robustness certificate [78, 93, 99, 106, 125]. Different from those work on the well-studied problem of robustness against *input perturbations*, we aim to evaluate the sensitivity of DNNs to *weight perturbations*.

Weight perturbations of DNNs are of realistic significance. First, a new threat model of weight perturbations was proposed by [100, 184], which showed that the so-called *fault sneaking/injection attack* can enforce DNN to misclassify some natural input images into target labels by slightly modifying weights at a single layer, while maintaining the classification of unspecified input images intact. This implies that the outputs of DNNs are also sensitive to weight perturbations. Moreover, there also exist weight perturbations in the non-adversarial setting. For example, *weight quantization* [87, 137, 188], a major DNN model compression technique commonly utilized by industry for DNN acceleration/implementation, induces weight perturbations by replacing full floating-point precision weights with fixed-point lower precision weights. It is even well supported in GPUs and mobile devices, e.g., PyTorch [131] in NVIDIA GPUs and TensorFlow Lite [3] for mobile devices. However, such direct mapping from full precision weights of DNNs into quantized weights could result in significant generalization error [141].

Different from the robustness issue caused by input perturbations, weight perturbations

focus on DNN models for natural (unperturbed) examples rather than adversarial examples. If training and testing samples stem from the same distribution, evaluating the model robustness against weight perturbations in the training dataset (namely, sensitivity of training accuracy to weight perturbations) is able to provide informative guidelines on the generalization ability of the weight-perturbed network (e.g., weight-quantized network). Thus, both *fault injection attack* and *weight quantization* motivate us to study the problem of model robustness against weight perturbations.

We formulate the certificate problem of model robustness against weight perturbations. The solution to this problem provides the *certified weight perturbation region* such that DNNs will maintain the accuracy if weight perturbations are within that region. We find provable and non-trivial lower bounds on the exact certified weight perturbation region in two scenarios: a) single-layer perturbation and b) multi-layer perturbation. We empirically show that the certified lower bound provides a reasonable assessment on the practical robustness of a model against fault injection attack [100, 184].

3.2 Related Works

A line of work relevant to ours is formal verification of neural networks [93], which provides robustness guarantee that any input perturbation within a neighborhood of the natural example cannot fool the classifier's top-1 prediction. In [23, 44, 46, 75], satisfiability modulo theory (SMT) or mixed-integer programming (MIP) based methods provided exact robustness certificate with respect to the input perturbation strength. However, these approaches suffer from the scalability issue due to their high computational complexity. Moreover, the work [45, 135, 161, 162, 164, 165] relaxed the exact verification problem by over-approximating the output space of a network given a neighborhood near the natural input. Such a relaxation leads to fast computation in the verification process but only proves a lower bound of exact robustness guarantees. Our paper is extended from the second line of work on certification of networks but with the main difference on the threat model: weight perturbations rather than input perturbations. Besides rigorously certifying network robustness, the work [31] empirically showed the robustness of convolutional neural networks (CNNs) against weight perturbations.

Different from the existing literature, our work on certified robustness extended from input to weight perturbations is non-trivial, since the latter could be coupled at multiple layers. Most importantly, we provide a use case, design of weight quantization scheme, showing that robustness against weight perturbations matters even in the non-adversarial setting.

It is also worth mentioning that different from the existing weight quantization work [33,69,87,89,128,136,166,188], our work provides a solution through the leans of formal verification of model robustness and has the potential to be integrated with well-developed weight quantization frameworks [2, 131].

3.3 Problem Setup

3.3.1 Notations

In this section, unless specified otherwise we use the following notations. Let $(\mathbf{x}, c)$ denote a pair of example $\mathbf{x}$ and class label c. For a K-layer neural network, let n_k, $\mathbf{W}^{(k)} \in \mathbb{R}^{n_k \times n_{k-1}}$, $\mathbf{b}^{(k)} \in \mathbb{R}^{n_k}$ denote the number of neurons, the weight matrix and the bias vector at layer k, respectively. We use the superscript (k) to indicate the variable associated with layer k. We also define $\mathbf{W} := \{\mathbf{W}^{(k)}\}_{k=1}^{K}$ and $\mathbf{b} := \{\mathbf{b}^{(k)}\}_{k=1}^{K}$ to denote the vector/matrix/set containing all variables indexed by k. And we use $[K]$ to denote the integer set $\{1, 2, \ldots, K\}$. Let $f(\mathbf{x}; \mathbf{W}, \mathbf{b}) \in \mathbb{R}^{n_K}$ be a neural network function with respect to the input $\mathbf{x}$ for n_K output classes. Here we refer f as the logit layer. The softmax layer can be safely discarded in our analysis due to its monotonicity. We use $f_j(\mathbf{x}; \mathbf{W}, \mathbf{b})$ (or simply f_j) to denote the j-th class output of the neural network. Let $\tilde{\Phi}^{(k)}(\mathbf{x}) \in \mathbb{R}^{n_k}$ and $\Phi^{(k)}(\mathbf{x}) \in \mathbb{R}^{n_k}$ denote the pre-activation and post-activation values of the k-th layer, respectively. And let $\sigma(\cdot)$ denote a non-linear element-wise activation function.

3.3.2 Neural Network Model

We focus on the K-layer fully-connected (FC) feedforward neural network with ReLU activation functions but the results can also be generalized to convolutional neural networks and general activations. The input-output relationship of the network is given by

$$\begin{aligned} \Phi^{(k)}(\mathbf{x}) &= \sigma\left(\tilde{\Phi}^{(k)}(\mathbf{x})\right), \\ \tilde{\Phi}^{(k)}(\mathbf{x}) &= \mathbf{W}^{(k)}\Phi^{(k-1)}(\mathbf{x}) + \mathbf{b}^{(k)}, \quad k \in [K], \end{aligned} \tag{3.1}$$

where $\Phi^{(0)}(\mathbf{x}) := \mathbf{x}$, and $f(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \Phi^{(K)}(\mathbf{x})$. In the classification setting, the predicted class c is the class that has the largest output value: $\arg\max_j f_j$.

Weight Perturbations The problem of our interest is to provide a robustness certificate for a neural network when its weight parameters are perturbed. We define ℓ_∞-norm based weight perturbations

as

$$\mathcal{B}(\mathbf{W}, \epsilon) = \left\{ \hat{\mathbf{W}} \mid \hat{\mathbf{W}} = \{\hat{\mathbf{W}}^{(k)}\}, \qquad \|\hat{\mathbf{W}}^{(k)} - \mathbf{W}^{(k)}\|_\infty \leq \epsilon, \forall k \in [K] \right\}, \tag{3.2}$$

where ϵ is the perturbation radius, and $\hat{\mathbf{W}}^{(k)}$ denotes the perturbed weights against the original weights $\mathbf{W}^{(k)}$ at each layer. Given ϵ, verifying the neural network robustness (at input $\mathbf{x}$ with the true label c) against weight perturbation can be cast as the following optimization problem

$$\begin{array}{ll} \underset{\hat{\mathbf{W}}}{\text{minimize}} & f_c(\hat{\mathbf{W}}) - \max_{j \neq c} f_j(\hat{\mathbf{W}}) \\ \text{subject to} & \hat{\mathbf{W}} \in \mathcal{B}(\mathbf{W}, \epsilon), \end{array} \tag{3.3}$$

where we use the simplified notation $f_j(\mathbf{W})$ to highlight the dependency of classification on model weights by omitting $\mathbf{x}$ and $\mathbf{b}$. If the optimal value of problem (3.3) is *positive*, then the robustness of the neural network is certified under ϵ-tolerant weight perturbation at input $\mathbf{x}$.

Goal of robustness certification under weight perturbation: Finding the *largest* ϵ such that problem (3.3) has the *positive* optimal value.

We remark that problem (3.3) maintains the similar formulation of verifying neural network's robustness against *input perturbation*, e.g., recent works [161, 162, 164, 165]. However, none of them investigated the direction of certifying robustness against weight perturbation. Different from the input perturbation that generates an adversarial example, the effect of weight perturbation on network robustness is measured under the original input. At this sense, the certified perturbation region in terms of ϵ in problem (3.3) offers the new perspective on how sensitive the prediction accuracy of a well-trained network could be against weight perturbation. Moreover, since weights can be perturbed at multiple layers, the problem of weight perturbation suffers from a more complicated layer-wise coupling issue than that of certifying input perturbation.

3.4 Certified Bound

In this section, we formally describe the idea of certified lower bound when the weights of neural networks are perturbed. We start from a simple 2-layer MLP example and provide general results in Theorem 3.1. Based on Theorem 3.1, we illustrate how to further generalize our results to multi-layer perturbation setting.

3.4.1 Certified lower bound: Single-layer weight perturbation

When the weight perturbation only occurs at a single layer (e.g. the N-th layer, $N \leq K$), we have the constraint:

$$\|\hat{\mathbf{W}}^{(N)} - \mathbf{W}^{(N)}\|_\infty \leq \epsilon,\ \hat{\mathbf{W}}^{(k)} = \mathbf{W}^{(k)},\ \text{if } k \neq N, k \in [K]. \tag{3.4}$$

Ideally, we would like to solve problem (3.3) *exactly* to get the maximum possible (exact) tolerance ϵ on the weight perturbation such that the top-1 prediction of a neural network classifier will not change. However, it has been shown that there does not exist a polynomial time algorithm to compute the *exact robustness* of neural networks [75]. Hence, our goal is to find a non-trivial ϵ efficiently and this problem can be formulated as follows.

Let $f_c^L(\hat{\mathbf{W}})$ and $f_c^U(\hat{\mathbf{W}})$ be two linear functions of $\hat{\mathbf{W}}$ such that $f_c^L(\hat{\mathbf{W}}) \leq f_c(\hat{\mathbf{W}}) \leq f_c^U(\hat{\mathbf{W}})$ for all $\hat{\mathbf{W}}$, and let

$$\gamma_c^L = \min_{\hat{\mathbf{W}} \in \mathcal{B}(\mathbf{W}, \epsilon)} f_c^L(\hat{\mathbf{W}}),\ \gamma_c^U = \max_{\hat{\mathbf{W}} \in \mathcal{B}(\mathbf{W}, \epsilon)} f_c^U(\hat{\mathbf{W}}). \tag{3.5}$$

We can compute ϵ by solving the following problem:

$$\begin{aligned} \underset{\epsilon}{\text{maximize}} \quad & \epsilon \\ \text{subject to} \quad & \gamma_c^L - \gamma_j^U > 0, \forall j \neq c. \end{aligned} \tag{3.6}$$

Note that the constraint set $\gamma_c^L - \max_{j \neq c} \gamma_j^U > 0$ (namely, $\gamma_c^L - \gamma_j^U > 0, \forall j \neq c$) is more restricted than $f_c(\hat{\mathbf{W}}) - \max_{j \neq c} f_j(\hat{\mathbf{W}}) > 0$. Thus, the solution to problem (3.6) provides a certified lower bound on the maximum ϵ to ensure the positive objective value of problem (3.3). In fact, in the following, we will show that γ_c^L and γ_c^U can be computed analytically, and hence we are able to find the solution of Eq. (3.6) efficiently through bi-section on ϵ. Note that in the work [161], the authors proposed an efficient algorithm *Fast-Lin* to compute a certified lower bound for neural networks with *input perturbation*, whereas we focus on *weight-perturbation* on the neural networks and show that it is also possible to derive certified lower bound for this problem setting.

3.4.1.1 Neural network f is bounded by two linear functions $f_c^L(\hat{\mathbf{W}})$ and $f_c^U(\hat{\mathbf{W}})$.

The core idea to deriving the linear bounds $f_c^L(\hat{\mathbf{W}})$, $f_c^U(\hat{\mathbf{W}})$ of a K-layer feed-forward neural network f is to apply linear upper and lower bound on each neuron's activation and consider the signs of associated weights. We start with a 2-layer network ($K = 2$) and then extend it to the general case. Suppose that the first layer weights are perturbed and we have Eq. (3.4) with $K = 1$. The j-th output of the network (with respect to $\hat{\mathbf{W}}^{(1)}$) is then given by

$$f_j(\hat{\mathbf{W}}^{(1)}) = \sum_{r \in [n_1]} W_{j,r}^{(2)} \sigma\left(\hat{\mathbf{W}}_{r,:}^{(1)} \mathbf{x} + b_r^{(1)}\right) + b_j^{(2)}, \tag{3.7}$$

where $W_{j,r}$ denotes the (j,r)-th entry of $\mathbf{W}$, and $\hat{\mathbf{W}}_{r,:}$ denotes the r-th row of $\hat{\mathbf{W}}$. Since $\mathbf{W}^{(1)}_{r,:} - \epsilon \leq \hat{\mathbf{W}}^{(1)}_{r,:} \leq \mathbf{W}^{(1)}_{r,:} + \epsilon$, the pre-activation $y^{(1)}_r := \hat{\mathbf{W}}^{(1)}_{r,:}\mathbf{x} + b^{(1)}_r$ at the 1-st layer is bounded by some constants $l^{(1)}_r$, and $u^{(1)}_r$, which are determined by the signs of $\mathbf{x}$ and the bounds of $\hat{\mathbf{W}}^{(1)}_{r,:}$.

Given $y^{(1)}_r \in [l^{(1)}_r, u^{(1)}_r]$, the non-linear activation function $\sigma(y^{(1)}_r)$ has explicit linear bounds [175] with slope and bias parameters $\{\alpha^{(1)}_{L,r}, \alpha^{(1)}_{U,r}\}$ and $\{\beta^{(1)}_{L,r}, \beta^{(1)}_{U,r}\}$ as follows:

$$\alpha^{(1)}_{L,r}(y^{(1)}_r + \beta^{(1)}_{L,r}) \leq \sigma(y^{(1)}_r) \leq \alpha^{(1)}_{U,r}(y^{(1)}_r + \beta^{(1)}_{U,r}). \tag{3.8}$$

If $l^{(1)}_r < 0 < u^{(1)}_r$, then $\alpha^{(1)}_{L,r} = \alpha^{(1)}_{U,r} = \frac{u^{(1)}_r}{u^{(1)}_r - l^{(1)}_r}$, $\beta^{(1)}_{L,r} = 0$, and $\beta^{(1)}_{U,r} = -l^{(1)}_r$; if $l^{(1)}_r \leq u^{(1)}_r \leq 0$, then all parameters are zeros; if $0 \leq l^{(1)}_r \leq u^{(1)}_r$, then $\alpha^{(1)}_{L,r} = \alpha^{(1)}_{U,r} = 1$ and $\beta^{(1)}_{L,r} = \beta^{(1)}_{U,r} = 0$. The equations (3.7) and (3.8) of the 2-layer network example imply two general rules:

1. The pre-activation bounds at the N-th layer are known *a priori* (since no weight prior to the N-th layer is perturbed), and thus we only need to perform bound propagation for $k > N$ layers;

2. The final layer bounds $f^L_c(\hat{\mathbf{W}})$ and $f^U_c(\hat{\mathbf{W}})$ can be computed via bound propagation. The idea is to compute the pre-activation bounds layer by layer (which is so-called bound propagation) analytically via Theorem 3.4.1. In Theorem 3.4.1, we show the analytic output bounds of neural networks when there exists single-layer ℓ_p-norm weight perturbation with $p \geq 1$.

Theorem 3.4.1. *Suppose that the N-th layer weights are perturbed in a K-layer neural network. Let $f : \mathbb{R}^{n_N \times n_{N-1}} \to \mathbb{R}^{n_K}$ denote the mapping from perturbed weights $\hat{\mathbf{W}}^{(N)}$ at the single layer N to predicted outputs at the final layer K. Then there exist two explicit functions $f^L_j : \mathbb{R}^{n_N \times n_{N-1}} \to \mathbb{R}$ and $f^U_j : \mathbb{R}^{n_N \times n_{N-1}} \to \mathbb{R}$ for class $\forall j \in [n_K]$, such that the following inequality holds*

$$f^L_j(\hat{\mathbf{W}}^{(N)}) \leq f_j(\hat{\mathbf{W}}^{(N)}) \leq f^U_j(\hat{\mathbf{W}}^{(N)}), \tag{3.9}$$

where $\|\hat{\mathbf{W}}^{(N)}_{s,:} - \mathbf{W}^{(N)}_{s,:}\|_p \leq \epsilon$ and $\hat{\mathbf{W}}^{(k)} = \mathbf{W}^{(k)}$ for $\forall k \neq N$, and $p \geq 1$. The closed forms of lower and upper bounds in (3.9) *are given by*

$$f^U_j(\hat{\mathbf{W}}^{(N)}) = \mathbf{\Lambda}^{(N-1)}_{j,:}\hat{\mathbf{W}}^{(N)}\Phi^{(N-1)}(\mathbf{x}) + \sum_{k=N}^{K} \mathbf{\Lambda}^{(k)}_{j,:}(\mathbf{b}^{(k)} + \mathbf{\Delta}^{(k)}_{:,j}), \tag{3.10}$$

$$f^L_j(\hat{\mathbf{W}}^{(N)}) = \mathbf{\Omega}^{(N-1)}_{j,:}\hat{\mathbf{W}}\Phi^{(N-1)}(\mathbf{x}) + \sum_{k=N}^{K} \mathbf{\Omega}^{(k)}_{j,:}(\mathbf{b}^{(k)} + \mathbf{\Theta}^{(k)}_{:,j}), \tag{3.11}$$

where

$$\mathbf{\Lambda}_{j,:}^{(k-1)} = \begin{cases} \mathbf{e}_j^\top & \text{if } k = K+1, \\ \mathbf{\Lambda}_{j,:}^{(k)} \odot \lambda_{j,:}^{(k-1)} & \text{if } k = N, \\ (\mathbf{\Lambda}_{j,:}^{(k)} \mathbf{W}^{(k)}) \odot \lambda_{j,:}^{(k-1)} & \text{otherwise.} \end{cases}$$

$$\mathbf{\Omega}_{j,:}^{(k-1)} = \begin{cases} \mathbf{e}_j^\top & \text{if } k = K+1, \\ \mathbf{\Omega}_{j,:}^{(k)} \odot \omega_{j,:}^{(k-1)} & \text{if } k = N, \\ (\mathbf{\Omega}_{j,:}^{(k)} \mathbf{W}^{(k)}) \odot \omega_{j,:}^{(k-1)} & \text{otherwise.} \end{cases}$$

Here the matrices $\lambda^{(k)}, \omega^{(k)}, \mathbf{\Delta}^{(k)}, \mathbf{\Theta}^{(k)}$ are functions of the linear bounding parameters $\{\alpha_{L,i}^{(k)}, \alpha_{U,i}^{(k)}\}$ and $\{\beta_{L,i}^{(k)}, \beta_{U,i}^{(k)}\}$ on each neuron i at layer k:

$$\lambda_{j,i}^{(k)} = \begin{cases} \alpha_{U,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Lambda}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0, \\ \alpha_{L,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Lambda}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0, \\ 1 & \text{if } k = N-1. \end{cases}$$

$$\omega_{j,i}^{(k)} = \begin{cases} \alpha_{L,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0, \\ \alpha_{U,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0, \\ 1 & \text{if } k = N-1. \end{cases}$$

$$\mathbf{\Delta}_{i,j}^{(k)} = \begin{cases} \beta_{U,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Lambda}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0, \\ \beta_{L,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Lambda}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0, \\ 0 & \text{if } k = N-1. \end{cases}$$

$$\mathbf{\Theta}_{i,j}^{(k)} = \begin{cases} \beta_{L,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} \geq 0, \\ \beta_{U,i}^{(k)} & \text{if } k \neq N-1,\ \mathbf{\Omega}_{j,:}^{(k+1)} \mathbf{W}_{:,i}^{(k+1)} < 0, \\ 0 & \text{if } k = N-1. \end{cases}$$

where $\odot$ is the Hadamard product and $\mathbf{e}_j \in \mathbb{R}^{n_K}$ is a j-th basis vector.

Proof. The proof on the input perturbation [17,161,175] can be adapted to weight perturbation in our case, where we have $f_j(\hat{\mathbf{W}}^{(N)}) \leq f_j^{U,K-1}(\hat{\mathbf{W}}^{(N)}) \leq f_j^{U,K-2}(\hat{\mathbf{W}}^{(N)}) \leq \ldots \leq f_j^{U,N+1}(\hat{\mathbf{W}}^{(N)}) = \tilde{\mathbf{W}}_{j,:}^{(N+1)} \sigma(\hat{\mathbf{W}}^{(N)} \Phi^{(N-1)} + \mathbf{b}^{(N)})$. The derivation of $\mathbf{\Lambda}_{j,:}^{(k-1)}, \mathbf{\Omega}_{j,:}^{(k-1)}$ from layers $N+1$ to $K-1$ is the same except for the N-th layer. For N-th layer, since the perturbation is now on $\hat{\mathbf{W}}^{(N)}$ rather than $\mathbf{x}$, we let $\mathbf{\Lambda}_{j,:}^{(N-1)} = \mathbf{\Lambda}_{j,:}^{(N)} \odot \lambda_{j,:}^{(N-1)}$. By using the same trick to decompose $\sigma(y)$ by the inequality (3.8) with associated sign of the equivalent matrix $\tilde{\mathbf{W}}_{j,:}^{(N+1)}$, we get the final upper bound (3.10). The lower bound $f_j^L(\hat{\mathbf{W}}^{(N)})$ can be derived similarly. □ □

3.4.1.2 Global output bounds γ_j^U and γ_j^L.

Based on Eq. (3.10) and (3.11), we can further derive the global output bounds γ_j^U and γ_j^L, which are constants, determined by Eq. (3.5). Since f_j^L and f_j^U are two linear functions and $\hat{\mathbf{W}} \in \mathcal{B}(\mathbf{W}, \epsilon)$ is a convex norm constraint, the optimal value of problem (3.5) can be obtained by Holder's inequalities:

$$\begin{aligned}\gamma_j^U &= \epsilon \|\mathbf{\Lambda}_{j,:}^{(N-1)}\|_1 \cdot \|\Phi^{(N-1)}(\mathbf{x})\|_q \\ &\quad + \mathbf{\Lambda}_{j,:}^{(N-1)} \mathbf{W}^{(N)} \Phi^{(N-1)}(\mathbf{x}) + \sum_{k=N}^{K} \mathbf{\Lambda}_{j,:}^{(k)} (\mathbf{b}^{(k)} + \mathbf{\Delta}_{:,j}^{(k)}), \qquad (3.12) \\ \gamma_j^L &= -\epsilon \|\mathbf{\Omega}_{j,:}^{(N-1)}\|_1 \cdot \|\Phi^{(N-1)}(\mathbf{x})\|_q \\ &\quad + \mathbf{\Omega}_{j,:}^{(N-1)} \mathbf{W}^{(N)} \Phi^{(N-1)}(\mathbf{x}) + \sum_{k=N}^{K} \mathbf{\Omega}_{j,:}^{(k)} (\mathbf{b}^{(k)} + \mathbf{\Theta}_{:,j}^{(k)}), \qquad (3.13)\end{aligned}$$

where $1/q = 1 - 1/p$. As γ_j^U and γ_c^L are monotonically increasing and decreasing with respect to ϵ, respectively, we can solve problem (3.6) by the bisection search over ϵ, which renders the certified lower bound on network's robustness against weight perturbation.

As a final remark, our work is different from adversarial robustness against input perturbation in that f is a function of perturbed weight matrix $\hat{\mathbf{W}}^{(N)}$ bounded by two linear functions $f_j^U(\hat{\mathbf{W}}^{(N)}), f_j^L(\hat{\mathbf{W}}^{(N)})$ as opposed to a function of perturbed input $\mathbf{x}$. Interestingly, once the neuron's activation are linearly bounded, we can directly apply the similar idea in [17, 161, 175] to compute the bounds layer-by-layer with Theorem 3.4.1. Our results can also be directly extended to convolutional layers following [17].

3.4.2 Certified lower bound: Multi-layer perturbation

When there exists multi-layer weights perturbations, deriving the certified lower bound becomes much more involved as the weight perturbations will be coupled across layers. However, computing layer-wise bound for each neuron is still possible – we can integrate previous single-layer results with interval bound propagation as demonstrated below. Without loss of generality, assume that the weight matrices at the S-th layer and the N-th layer are both perturbed with $S < N < K$. Eq. (3.12) and (3.13) can be directly used to compute the perturbation bounds at each neuron of layer S by viewing the $N-1$-th layer as the output layer. For $N-1 \leq i \leq K$, we have $\hat{l}_r^{(i)} \leq \Phi_r^{(i)} \leq \hat{u}_r^{(i)}$, where $\hat{u}_r^{(i)} = \sigma(u_r^{(i)})$, $\hat{l}_r^{(i)} = \sigma(l_r^{(i)})$. Let $k = N-1$, $p = \infty$ and $q = 1$, we then have:

- if $i = k$,

$$u_j^{(i+1)} = |\mathbf{W}_{j,:}^{(i+1)}|\frac{\hat{u}^{(i)} - \hat{l}^{(i)}}{2} + \mathbf{W}_{j,:}^{(i+1)}\frac{\hat{u}^{(i)} + \hat{l}^{(i)}}{2} + \mathbf{b}_j^{(k+1)} + \epsilon\|\frac{\hat{u}^{(i)} - \hat{l}^{(i)}}{2}\|_q + \epsilon\|\frac{\hat{u}^{(i)} + \hat{l}^{(i)}}{2}\|_q$$

$$l_j^{(i+1)} = -|\mathbf{W}_{j,:}^{(i+1)}|\frac{\hat{u}^{(i)} - \hat{l}^{(i)}}{2} + \mathbf{W}_{j,:}^{(i+1)}\frac{\hat{u}^{(i)} + \hat{l}^{(i)}}{2} + \mathbf{b}_j^{(k+1)} - \epsilon\|\frac{\hat{u}^{(i)} - \hat{l}^{(i)}}{2}\|_q + \epsilon\|\frac{\hat{u}^{(i)} + \hat{l}^{(i)}}{2}\|_q$$

- if $i > k$,

$$u_j^{(i+1)} = |\mathbf{W}_{j,:}^{(i+1)}|\frac{\hat{u}^{(i)} - \hat{l}^{(i)}}{2} + \mathbf{W}_{j,:}^{(i+1)}\frac{\hat{u}^{(i)} + \hat{l}^{(i)}}{2} + \mathbf{b}_j^{(i+1)}$$
$$l_j^{(i+1)} = -|\mathbf{W}_{j,:}^{(i+1)}|\frac{\hat{u}^{(i)} - \hat{l}^{(i)}}{2} + \mathbf{W}_{j,:}^{(i+1)}\frac{\hat{u}^{(i)} + \hat{l}^{(i)}}{2} + \mathbf{b}_j^{(i+1)}$$

and the network output $f(\hat{\mathbf{W}}^{(S)}, \hat{\mathbf{W}}^{(N)})$ is bounded by $l_j^{(K)} \leq f_j(\hat{\mathbf{W}}^{(S)}, \hat{\mathbf{W}}^{(N)}) \leq u_j^{(K)}$. For simplicity, we present the above analysis with same ϵ and without bias perturbation, but our analysis can be extended to the case where ϵ_i associated to the i-th layer and when biases $\mathbf{b}$ are perturbed.

3.5 Experimental Results

We demonstrate the effectiveness of our approach under against fault sneaking attack [184]. To align with our theoretical results, we perform experiments under multilayer perceptron (MLP) models of various numbers of layers. The performance is evaluated under 4 datasets, MNIST, MNIST-fashion, SVHN, and CIFAR-10.

3.5.1 Model robustness against fault sneaking attack [184].

It was shown in [184] that slightly perturbing model weights at a single layer is capable of misclassfying a specific set of natural images toward target labels but keeping classification of unspecified input images intact. The corresponding threat model is called *fault sneaking attack (FSA)*. It is worth mentioning that such an attack is commonly performed at deep layers of a network due to its stealthyness requirement. We refer readers to Appendix for the detailed setting of attack generation and our experiment.

Table 3.1: Certified perturbation bounds and ℓ_∞-norm of weight perturbations caused by FSA. Here FAS perturbs the last 4 layers of a 10-layer MLP under 4 datasets.

dataset	layer index	7	8	9	10
MNIST	original Acc (%)	97.8	97.8	97.8	97.8
	Acc after attack (%)	94	93.9	93.6	94.3
	certified bound	2×10^{-6}	2.6×10^{-5}	0.0003	0.0083
	attack perturbation	0.042	0.048	0.052	0.073
MNIST-Fashion	original Acc (%)	88.6	88.6	88.6	88.6
	Acc after attack (%)	84.6	84.3	84.0	84.3
	certified bound	2×10^{-6}	2.4×10^{-5}	0.00033	0.0117
	attack perturbation	0.025	0.0306	0.0351	0.11
SVHN	original Acc (%)	82.6	82.6	82.6	82.6
	Acc after attack (%)	80.5	80.1	80.3	80.6
	certified bound	4×10^{-6}	5.6×10^{-5}	0.0008	0.035
	attack perturbation	0.023	0.027	0.036	0.102
CIFAR-10	original Acc (%)	56.7	56.7	56.7	56.7
	Acc after attack (%)	50.2	51.1	51.5	51.2
	certified bound	4×10^{-6}	5×10^{-5}	0.0007	0.027
	attack perturbation	0.036	0.04	0.056	0.15

Table 3.1 shows the original accuracy (Acc), Acc after FSA, certified lower bound on weight perturbation, and the ℓ_∞ norm of weight perturbations caused by FSA. We see that the certified lower bound decreases as the layer index decreases since the linear bound approximation becomes looser when it needs to propagate over more layers prior to the output layer. Moreover, the certified lower bound shares the same pattern of FSA against the layer index. The indicates the intrinsic robustness of networks against FSA: A shallower layer is more vulnerable to FSA, which is also verified by Figure 3.1.

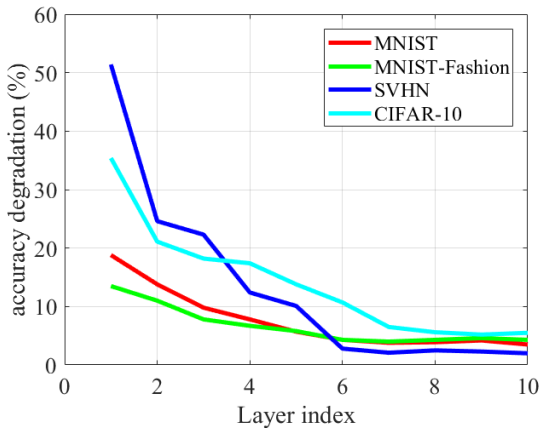


Figure 3.1: Test accuracy degradation after perturbing each layer of the model using fault injection attack.

Chapter 4

DNN Defense

4.1 Motivation

Recent studies on mode connectivity show that two independently trained deep neural network (DNN) models with the same architecture and loss function can be connected on their loss landscape using a high-accuracy/low-loss path characterized by a simple curve [40, 53, 58]. This insight on the loss landscape geometry provides us with easy access to a large number of similar-performing models on the low-loss path between two given models, and [53] use this to devise a new model ensembling method. Another line of recent research reveals interesting geometric properties relating to adversarial robustness of DNNs [49, 50, 159, 172]. An adversarial data or model is defined to be one that is close to a bonafide data or model in some space, but exhibits unwanted or malicious behavior. Motivated by these geometric perspectives, in this study, we propose to employ mode connectivity to study and improve adversarial robustness of DNNs against different types of threats.

A DNN can be possibly tampered by an adversary during different phases in its life cycle. For example, during the training phase, the training data can be corrupted with a designated trigger pattern associated with a target label to implant a backdoor for trojan attack on DNNs [60, 102]. During the inference phase when a trained model is deployed for task-solving, prediction-evasive attacks are plausible [14, 56, 182], even when the model internal details are unknown to an attacker [28, 70, 181]. In this research, we will demonstrate that by using mode connectivity in loss landscapes, we can repair backdoored or error-injected DNNs. We also show that mode connectivity analysis reveals the existence of a robustness loss barrier on the path connecting regular and adversarially-trained models.

We motivate the novelty and benefit of using mode connectivity for mitigating training-phase adversarial threats through the following practical scenario: as training DNNs is both time- and resource-consuming, it has become a common trend for users to leverage pre-trained models released in the public domain[1]. Users may then perform model fine-tuning or transfer learning with a small set of bonafide data that they have. However, publicly available pre-trained models may carry an unknown but significant risk of tampering by an adversary. It can also be challenging to detect this tampering, as in the case of a backdoor attack[2], since a backdoored model will behave like a regular model in the absence of the embedded trigger. Therefore, it is practically helpful to provide tools to users who wish to utilize pre-trained models while mitigating such adversarial threats. We show that our proposed method using mode connectivity with limited amount of bonafide data can repair backdoored or error-injected DNNs, while greatly countering their adversarial effects.

Our main contributions are summarized as follows:

- For backdoor and error-injection attacks, we show that the path trained using limited bonafide data connecting two tampered models can be used to repair and redeem the attacked models, thereby resulting in high-accuracy and low-risk models. The performance of mode connectivity is significantly better than several baselines including fine-tuning, training from scratch, pruning, and random weight perturbations. We also provide technical explanations for the effectiveness of our path connection method based on model weight space exploration and similarity analysis of input gradients for clean and tampered data.

- For evasion attacks, we use mode connectivity to study standard and adversarial-robustness loss landscapes. We find that between a regular and an adversarially-trained model, training a path with standard loss reveals no barrier, whereas the robustness loss on the same path reveals a barrier. This insight provides a geometric interpretation of the "no free lunch" hypothesis in adversarial robustness [21, 37, 152]. We also provide technical explanations for the high correlation observed between the robustness loss and the largest eigenvalue of the input Hessian matrix on the path.

- Our experimental results on different DNN architectures (ResNet and VGG) and datasets (CIFAR-10 and SVHN) corroborate the effectiveness of using mode connectivity in loss landscapes to understand and improve adversarial robustness. We also show that our path connection is resilient

[1] For example, the Model Zoo project: `https://modelzoo.co`

[2] See the recent call for proposals on Trojans in AI announced by IARPA: `https://www.iarpa.gov/index.php/research-programs/trojai/trojai-baa`

to the considered adaptive attacks that are aware of our defense. To the best of our knowledge, this is the first work that proposes using mode connectivity approaches for adversarial robustness.

4.2 Background on Model Connection and DNN Attacks

4.2.1 Mode Connectivity in Loss Landscapes

Let w_1 and w_2 be two sets of model weights corresponding to two neural networks independently trained by minimizing any user-specified loss $l(w)$, such as the cross-entropy loss. Moreover, let $\phi_\theta(t)$ with $t \in [0,1]$ be a continuous piece-wise smooth parametric curve, with parameters θ, such that its two ends are $\phi_\theta(0) = w_1$ and $\phi_\theta(1) = w_2$.

To find a high-accuracy path between w_1 and w_2, it is proposed to find the parameters θ that minimize the expectation over a uniform distribution on the curve [53],

$$L(\theta) = E_{t\sim q_\theta(t)}\left[l(\phi_\theta(t))\right] \tag{4.1}$$

where $q_\theta(t)$ is the distribution for sampling the models on the path indexed by t.

Since $q_\theta(t)$ depends on θ, in order to render the training of high-accuracy path connection more computationally tractable, [53, 58] proposed to instead use the following loss term,

$$L(\theta) = E_{t\sim U(0,1)}\left[l(\phi_\theta(t))\right] \tag{4.2}$$

where $U(0,1)$ is the uniform distribution on $[0,1]$.

The following functions are commonly used for characterizing the parametric curve function $\phi_\theta(t)$.

Polygonal chain [55]. The two trained networks w_1 and w_2 serve as the endpoints of the chain and the bends of the chain are parameterized by θ. For instance, the case of a chain with one bend is

$$\phi_\theta(t) = \begin{cases} 2\left(t\theta + (0.5-t)\,\omega_1\right), & 0 \le t \le 0.5 \\ 2\left((t-0.5)\,\omega_2 + (1-t)\,\theta\right), & 0.5 \le t \le 1. \end{cases} \tag{4.3}$$

Bezier curve [48]. A Bezier curve provides a convenient parametrization of smoothness on the paths connecting endpoints. For instance, a quadratic Bezier curve with endpoints w_1 and w_2 is given by

$$\phi_\theta(t) = \ (1-t)^2\omega_1 + 2t\,(1-t)\,\theta + t^2\omega_2, \quad 0 \le t \le 1. \tag{4.4}$$

It is worth noting that, while current research on mode connectivity mainly focuses on generalization analysis [40, 53, 58, 156] and has found remarkable applications such as fast model

ensembling [53], our results show that its implication on adversarial robustness through the lens of loss landscape analysis is a promising, yet largely unexplored, research direction. [172] scratched the surface but focused on interpreting decision surface of input space and only considered evasion attacks.

4.2.2 Backdoor, Evasion, and Error-Injection Adversarial Attacks

Backdoor attack. Backdoor attack on DNNs is often accomplished by designing a designated trigger pattern with a target label implanted to a subset of training data, which is a specific form of data poisoning [13, 72, 139]. A backdoored model trained on the corrupted data will output the target label for any data input with the trigger; and it will behave as a normal model when the trigger is absent. For mitigating backdoor attacks, majority of research focuses on backdoor detection or filtering anomalous data samples from training data for re-training [26, 151, 155], while our aim is to repair backdoored models for models using mode connectivity and limited amount of bonafide data.

Evasion attack. Evasion attack is a type of inference-phase adversarial threat that generates adversarial examples by mounting slight modification on a benign data sample to manipulate model prediction [14, 158]. For image classification models, evasion attack can be accomplished by adding imperceptible noises to natural images and resulting in misclassification [24, 56, 157, 171]. Different from training-phase attacks, evasion attack does not assume access to training data. Moreover, it can be executed even when the model details are unknown to an adversary, via black-box or transfer attacks [28, 122, 180].

Error-injection attack. Different from attacks modifying data inputs, error-injection attack injects errors to model weights at the inference phase and aims to cause misclassification of certain input samples [100, 185]. At the hardware level of a deployed machine learning system, it can be made plausible via laser beam [7] and row hammer [154] to change or flip the logic values of the corresponding bits and thus modifying the model parameters saved in memory.

4.3 Experiment Setup

Here we report the experimental results, provide technical explanations, and elucidate the effectiveness of using mode connectivity for studying and enhancing adversarial robustness in three representative themes: (i) backdoor attack; (ii) error-injection attack; and (iii) evasion attack. Our

Table 4.1: Test accuracy of untampered models for different datasets and model architectures.

	architecture	VGG	ResNet
CIFAR-10	model $(t = 0)$	88%	87%
	model $(t = 1)$	86%	91%
SVHN	model $(t = 0)$	96%	97%
	model $(t = 1)$	98%	99%

experiments were conducted on different network architectures (VGG and ResNet) and datasets (CIFAR-10 and SVHN).

When connecting models, we use the cross entropy loss and the quadratic Bezier curve as described in Eq. (4.4). In what follows, we begin by illustrating the problem setups bridging mode connectivity and adversarial robustness, summarizing the results of high-accuracy (low-loss) pathways between untampered models for reference, and then delving into detailed discussions. Depending on the context, we will use the terms error rate and accuracy on clean/adversarial samples interchangeably. The error rate of adversarial samples is equivalent to their attack failure rate as well as 100%- attack accuracy.

4.3.1 Network Architecture and Training

We mainly use two model architectures, VGG and ResNet. The VGG model [143] has 13 convolutional layers and 3 fully connected layers. The ResNet model is based on the Preactivation-ResNet implementation [63] with 26 layers. The clean test accuracy of untampered models of different architectures on CIFAR-10 and SVHN are given in Table 4.1. All of the experiments are performed on 6 GTX 1080Ti GPUs. The experiments are implemented with Python and Pytorch.

4.3.2 Illustration and Implementation Details of Backdoor and Error-Injection Attacks

Backdoor attack The backdoor attack is implemented by poisoning the training dataset and then training a backdoored model with this training set. To poison the training set, we randomly pick 10% images from the training dataset and add a trigger to each image. The shape and location of the

trigger is shown in Figure 4.1. Meanwhile, we set the labels of the triggered images to the target label(s) as described in Section 4.4.

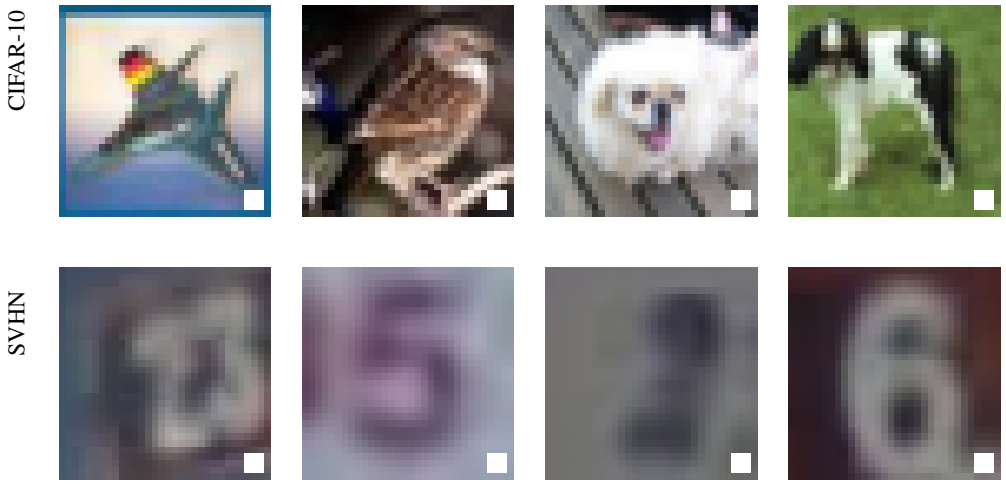


Figure 4.1: Examples of backdoored images on CIFAR-10 and SVHN. The triggers are white blocks located at the right-bottom area of each image.

Error-injection attack We select 1000 images from the test set and pick 4 images as targeted samples with randomly selected target labels for inducing attack. The target labels of the 4 selected images are different from their original correct labels. The goal of the attacker is to change the classification of the 4 images to the target labels while keeping the classification of the remaining 996 images unchanged through modifying the model parameters. To obtain the models with injected errors on CIFAR-10, we first train two models with a clean accuracy of 88% and 86%, respectively. Keeping the classification of a number of images unchanged can help to mitigate the accuracy degradation incurred by the model weight perturbation. After perturbing the model weights, the 4 errors can be injected into the model successfully with 100% accuracy for their target labels. The accuracy for other clean images become 78% and 75%, respectively.

4.3.3 Problem Setup and Mode Connection between Untampered Models

Problem setup for backdoor and error-injection attacks. We consider the practical scenario where a user has two potentially tampered models and a limited number of bonafide data at hand. The tampered models behave normally as untampered ones on non-triggered/non-targeted inputs so the user aims to fully exploit the model power while alleviating adversarial effects. The

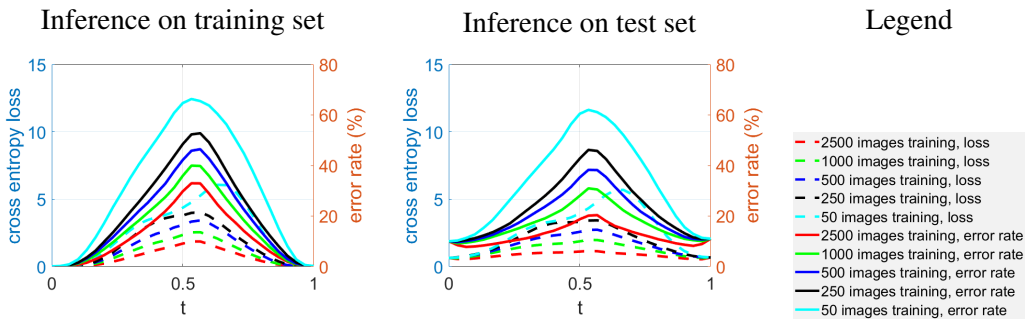


Figure 4.2: Loss and error rate on the path connecting two untampered VGG models trained on CIFAR-10. The path connection is trained using different settings as indicated by the curve colors. The inference results on test set are evaluated using 5000 samples, which are separate from what are used for path connection.

problem setup applies to the case of one tampered model, where we use the bonafide data to train a fine-tuned model and then connect the given and the fine-tuned models.

Problem setup for evasion attack. For gaining deeper understanding on evasion attacks, we consider the scenario where a user has access to the entire training dataset and aims to study the behavior of the models on the path connecting two independently trained models in terms of standard and robust loss landscapes, including model pairs selected from regular and adversarially-trained models.

Regular path connection between untampered models. Figure 4.2 and Figure 4.3 show the cross entropy loss and training/test error rate of models on the path connecting untampered models. The untampered models are independently trained using the entire training data. While prior results have demonstrated high-accuracy path connection using the entire training data [40, 53, 58], our path connection is trained using different portion of the original test data corresponding to the scenario of limited amount of bonafide data. Notably, when connecting two DNNs, a small number of clean data is capable of finding models with good performance. For example, path connection using merely 1000/2500 CIFAR-10 samples only reduces the test accuracy (on other 5000 samples) of VGG16 models by at most 10%/5% when compared to the well-trained models (those at $t = 0$ and $t = 1$), respectively. In addition, regardless of the data size used for path connection, the model having the worst performance is usually located around the point $t = 0.5$, as it is geometrically the farthest model from the two end models on the path.

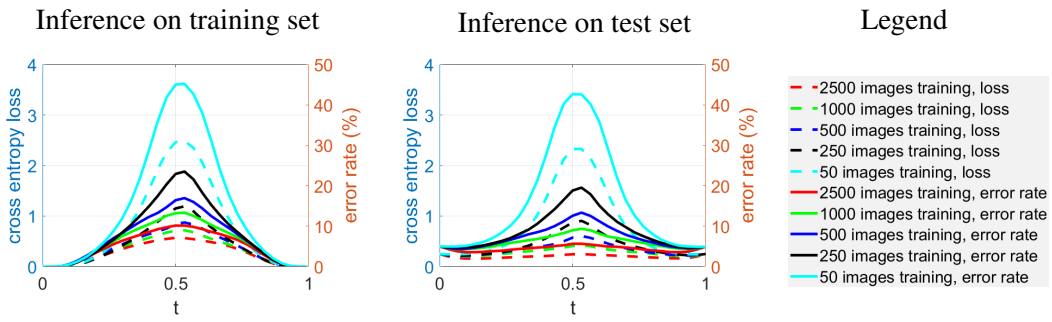


Figure 4.3: Loss and error rate on the path connecting two untampered ResNet models trained on SVHN. The path connection is trained using different settings as indicated by the curve colors. The inference results on test set are evaluated using 5000 samples, which are separate from what are used for path connection.

Table 4.2: Error rate of backdoored models. The error rate of clean/backdoored samples means standard-test-error/attack-failure-rate, respectively. The results are evaluated on 5000 non-overlapping clean/triggered images selected from the test set. For reference, the test errors of clean images on untampered models are 12% for CIFAR-10 (VGG), and 4% for SVHN (ResNet), respectively.

	Backdoor attacks	Single-target attack		All-targets attack	
	Dataset	CIFAR-10 (VGG)	SVHN (ResNet)	CIFAR-10 (VGG)	SVHN (ResNet)
Model ($t = 0$)	Clean images	15%	5.4%	14.2%	6.1%
	Triggered images	0.07%	0.22%	12.9%	8.3%
Model ($t = 1$)	Clean images	13%	7.7%	19%	7.5%
	Triggered images	2%	0.17%	13.6%	9.2%

4.4 Defense Performance against Backdoor Attack

Attack implementation. We follow the procedures in [60] to implement backdoor attacks and obtain two backdoored models trained on the same poisoned training data. The trigger pattern is placed at the right-bottom of the poisoned images as shown in Figure 4.1. Specifically, 10% of the training data are poisoned by inserting the trigger and changing the original correct labels to the target label(s). Here we investigate two kinds of backdoor attacks: (a) single-target attack which sets the target label T to a specific label (we choose $T =$ class 1); and (b) all-targets attack where the target label T is set to the original label i plus 1 and then modulo 9, i.e., $T = i + 1 (\text{mod } 9)$. Their performance on clean (untriggered) and triggered data samples are given in Table 4.2.

We show the prediction error of the triggered data relative to the true labels on all datasets

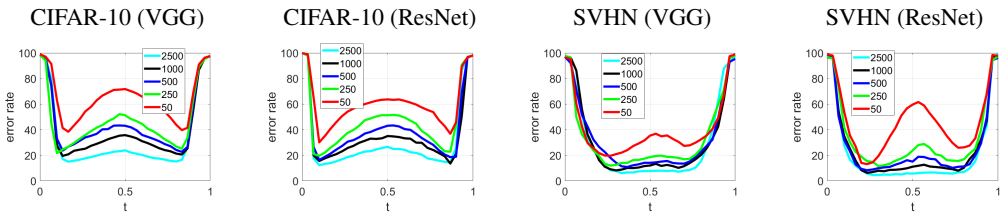


Figure 4.4: Prediction error rate against backdoor attacks on the connection path.

and networks in Figure 4.4. The error rate means the fraction of triggered images having top-1 predictions different from the original true labels. The prediction error rate of triggered data is high at path ends ($t = 0, 1$) since the two end models are tampered. It has similar trend as standard test error for models not too close to the path ends, suggesting path connection can find models having good classification accuracy on triggered data.

The backdoored models have similar performance on clean data as untampered models but will indeed misclassify majority of triggered samples. Comparing to single-target attack, all-targets attack is more difficult and has a higher attack failure rate, since the target labels vary with the original labels.

Evaluation and analysis. We train a path connecting the two backdoored models with limited amount of bonafide data. As shown in Figure 4.5, at both path endpoints ($t = \{0, 1\}$) the two tampered models attain low error rates on clean data but are also extremely vulnerable to backdoor attacks (low error rate on backdoored samples means high attack success rate). Nonetheless, we find that path connection with limited bonafide data can effectively mitigate backdoor attacks and redeem model power. For instance, the models at $t = 0.1$ or $t = 0.9$ can simultaneously attain similar performance on clean data as the tampered models while greatly reducing the backdoor attack success rate from close to 100% to nearly 0%. Moreover, most models on the path (e.g. when $t \in [0.1, 0.9]$) exhibit high resilience to backdoor attacks, suggesting mode connection with limited amount of bonafide data can be an effective countermeasure. While having resilient models to backdoor attacks on the path, we also observe that the amount of bonafide data used for training path has a larger impact on the performance of clean data. Path connection using fewer data samples will yield models with higher error rates on clean data, which is similar to the results of path connection between untampered models discussed in Section 4.3.3. The advantages of redeeming model power using mode connectivity are consistent when evaluated on different network architectures (VGG and ResNet) and datasets (CIFAR-10 and SVHN) as shown in Figure 4.6 and Figure 4.7.

Comparison with baselines. We compare the performance of mode connectivity against

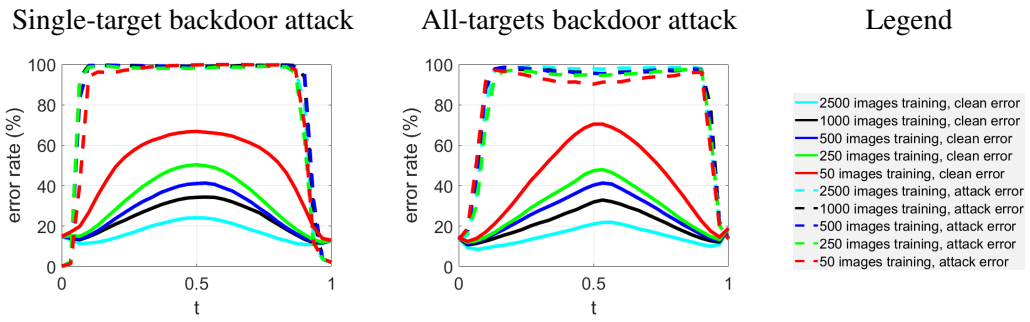


Figure 4.5: Error rate against backdoor attacks on the connection path for CIFAR-10 (VGG). The error rate of clean/backdoored samples means the standard-test-error/attack-failure-rate, respectively.

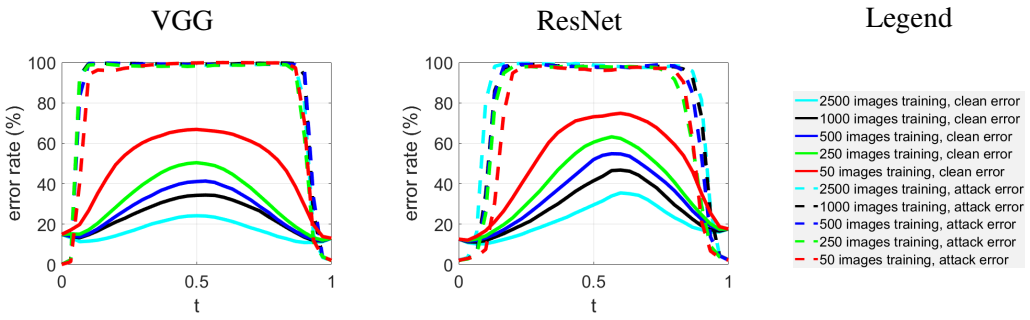


Figure 4.6: Error rate of single-target backdoor attack on the connection path for CIFAR-10. The error rate of clean/backdoored samples means standard-test-error/attack-failure-rate, respectively.

backdoor attacks with the following baseline methods: (i) fine-tuning backdoored models with bonafide data; (ii) training a new model of the same architecture from scratch with bonafide data; (iii) model weight pruning and then fine-tuning with bonafide data using [88]; and (iv) random Gaussian perturbation to the model weights leading to a noisy model. The results are summarized in Table 4.3 and Table 4.4. For our proposed path connection method, we train the connection using different number of images as given in Table 4.3 for 100 epochs and then report the performance of the model associated with a selected index t on the path. For the fine-tuning and training-from-scratch methods, we report the model performance after training for 100 epochs. For the random Gaussian perturbation to model weights, we evaluate the model performance under Gaussian noise perturbations on the model parameters. There are two given models which are the models at $t = 0$ and $t = 1$. The Gaussian noise has zero mean with a standard deviation of the absolute value of the difference between the two given models. Then we add the Gaussian noise to the two given models respectively and test their accuracy for clean and triggered images. For Gaussian noise, the

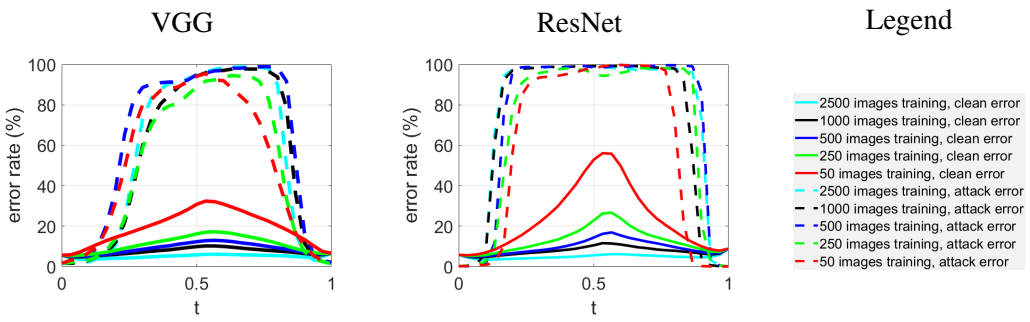


Figure 4.7: Error rate of single-target backdoor attack on the connection path for SVHN. The error rate of clean/backdoored samples means standard-test-error/attack-failure-rate, respectively.

experiment is performed multiple times (50 times) and we report the average accuracy. We can see that adding Gaussian noise perturbations to the model does not necessarily change the model status from robust to non-robust or from non-robust to robust. The path connection or evolution from the model at $t = 0$ to that $t = 1$ follows a specific path achieving robustness against backdoor attack rather than random exploration. For pruning, we use the filter pruning method [88] to prune filters from CNNs that are identified as having a small effect on the output accuracy. By removing the whole filters in the network together with their connecting feature maps, the computation costs are reduced significantly. We first prune about 60% of its parameters for VGG or 20% parameters for ResNet. Then we retrain the network with different number of images as given in Table 4.3 for 100 epochs. The clean accuracy and backdoor accuracy are as reported.

Evaluated on different network architectures and datasets, the path connection method consistently maintains superior accuracy on clean data while simultaneously attaining low attack accuracy over the baseline methods, which can be explained by the ability of finding high-accuracy paths between two models using mode connectivity. For CIFAR-10 (VGG), even using as few as 50 bonafide samples for path connection, the subsequent model in Table 4.3 still remains 63% clean accuracy while constraining backdoor accuracy to merely 2.5%. The best baseline method is fine-tuning, which has similar backdoor accuracy as path connection but attains lower clean accuracy (e.g. 17% worse than path connection when using 50 bonafide samples). For SVHN (ResNet), the clean accuracy of fine-tuning can be on par with path connection, but its backdoor accuracy is significantly higher than path connection. For example, when trained with 250 samples, they have the same clean accuracy but the backdoor accuracy of fine-tuning is 58.7% higher than path connection. Training from scratch does not yield competitive results given limited amount of training data. Noisy models perturbed by adding zero-mean Gaussian noises to the two models are not effective against

Table 4.3: Performance against single-target backdoor attack. The clean/backdoor accuracy means standard-test-accuracy/attack-success-rate, respectively.

		Method / Bonafide data size	2500	1000	500	250	50
CIFAR-10 (VGG)	Clean Accuracy	Path connection ($t = 0.1$)	88%	83%	80%	77%	63%
		Fine-tune	84%	82%	78%	74%	46%
		Train from scratch	50%	39%	31%	30%	20%
		Noisy model ($t = 0$)	21%	21%	21%	21%	21%
		Noisy model ($t = 1$)	24%	24%	24%	24%	24%
		Prune	88%	85%	83%	82%	81%
	Backdoor Accuracy	Path connection ($t = 0.1$)	1.1%	0.8%	1.5%	3.3%	2.5%
		Fine-tune	1.5%	0.9%	0.5%	1.9%	2.8%
		Train from scratch	0.4%	0.7%	0.3%	3.2%	2.1%
		Noisy model ($t = 0$)	97%	97%	97%	97%	97%
		Noisy model ($t = 1$)	91%	91%	91%	91%	91%
		Prune	43%	49%	81%	79%	82%
SVHN (ResNet)	Clean Accuracy	Path connection ($t = 0.2$)	96%	94%	93%	89%	82%
		Fine-tune	96%	94%	91%	89%	76%
		Train from scratch	87%	75%	61%	34%	12%
		Noisy model ($t = 0$)	13%	13%	13%	13%	13%
		Noisy model ($t = 1$)	11%	11%	11%	11%	11%
		Prune	96%	95%	93%	91%	89%
	Backdoor Accuracy	Path connection ($t = 0.2$)	2.5%	3%	3.6%	4.3%	16%
		Fine-tune	14%	7%	29%	63%	60%
		Train from scratch	3%	3.6%	5%	2.2%	3.9%
		Noisy model ($t = 0$)	51%	51%	51%	51%	51%
		Noisy model ($t = 1$)	42%	42%	42%	42%	42%
		Prune	80%	90%	88%	92%	94%

backdoor attacks and may suffer from low clean accuracy. Pruning gives high clean accuracy but has very little effect on mitigating backdoor accuracy.

Technical Explanations. To provide technical explanations for the effectiveness of our proposed path connection method in repairing backdoored models, we run two sets of analysis: (i) model weight space exploration and (ii) data similarity comparison. For (i), we generate 1000 noisy versions of a backdoored model via random Gaussian weight perturbations. We find that they suffer from low clean accuracy and high attack success rate, which suggests that a good model with high-clean-accuracy and low-attack-accuracy is unlikely to be found by chance. We also report the distinct difference between noisy models and models on the path in the weight space to validate the necessity of using our path connection for attack mitigation and model repairing.

For (ii), we run similarity analysis of the input gradients between the end (backdoored) models and models on the connection path for both clean data and triggered data. We find that the

Table 4.4: Performance comparison of path connection and baselines against single-target backdoor attack. The clean/backdoor accuracy means standard-test-accuracy/attack-success-rate, respectively.

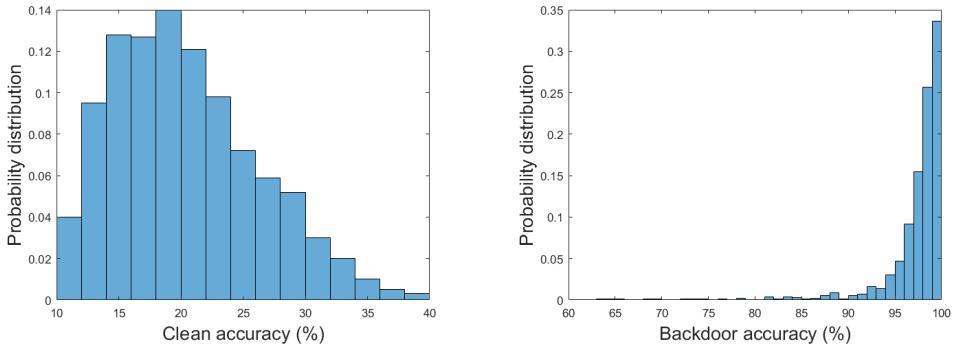
		Methods / bonafide data size	2500	1000	500	250	50
CIFAR-10 (ResNet)	Clean Accuracy	Path connection ($t = 0.2$)	87%	83%	80%	77%	67%
		Fine-tune	85%	84%	83%	83%	72%
		Train from scratch	42%	36%	33%	29%	22%
		Noisy model ($t = 0$)	12%	12%	12%	12%	12%
		Noisy model ($t = 1$)	10%	10%	10%	10%	10%
		Prune	85%	82%	80%	79%	77%
	Backdoor Accuracy	Path connection ($t = 0.2$)	0.6%	1.2%	1.3%	2.3%	5.3%
		Fine-tune	0.7%	8.7%	19%	20%	18%
		Train from scratch	3.7%	4.3%	3.5%	8.2%	6.5%
		Noisy model ($t = 0$)	95%	95%	95%	95%	95%
		Noisy model ($t = 1$)	97%	97%	97%	97%	97%
		Prune	37%	52%	78%	86%	88%
SVHN (VGG)	Clean Accuracy	Path connection ($t = 0.7$)	95%	93%	91%	87%	75%
		Fine-tune	93%	92%	90%	89%	77%
		Train from scratch	88%	82%	76%	54%	25%
		Noisy model ($t = 0$)	22%	22%	22%	22%	22%
		Noisy model ($t = 1$)	16%	16%	16%	16%	16%
		Prune	94%	92%	91%	89%	86%
	Backdoor Accuracy	Path connection ($t = 0.7$)	2%	6%	2%	9%	22%
		Fine-tune	11%	10%	13%	30%	61%
		Train from scratch	2.6%	2.3%	3.3%	4.7%	13%
		Noisy model ($t = 0$)	79%	79%	79%	79%	79%
		Noisy model ($t = 1$)	62%	62%	62%	62%	62%
		Prune	75%	69%	78%	86%	91%

similarity of triggered data is much lower than that of clean data when the model is further away in the path from the end models, suggesting that our path connection method can neutralize the backdoor effect.

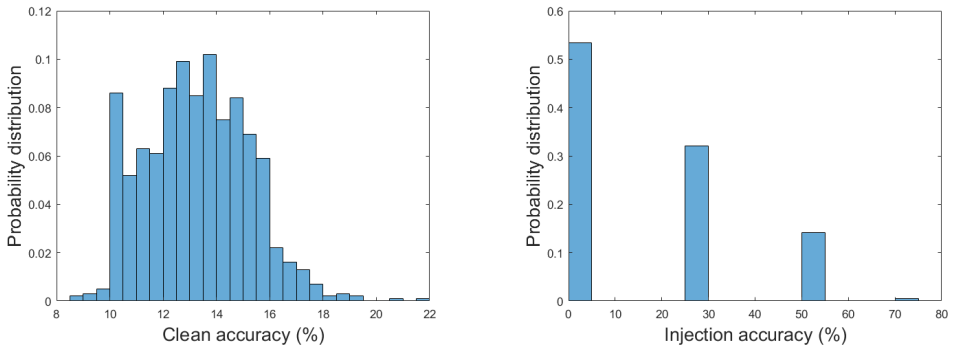
To explore the model weight space, we add Gaussian noise to the weights of a backdoored model to generate 1000 noisy models of the backdoored model at $t = 0$. The standard normal Gaussian noise has zero mean and a standard deviation of the absolute difference between the two end models on the path. The distribution of clean accuracy and backdoor accuracy of these noisy models are reported in Figure 4.8 (a). The results show that the noisy models are not ideal for attack mitigation and model repairing since they suffer from low clean accuracy and high attack success rate. In other words, it is unlikely, if not impossible, to find good models by chance. We highlight that finding a path robust to backdoor attack between two backdoored models is highly non-intuitive, considering the high failure rate of adding noise. We can observe similar phenomenon

for the injection attack in Figure 4.8 (b).

The advantage of our path connection method over fine-tuning demonstrates the importance of using the knowledge of mode connectivity for model repairing.



(a) backdoor attack



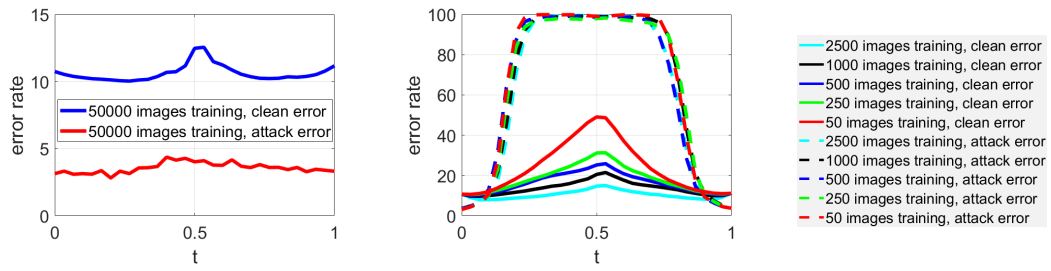
(b) error-injection attack

Figure 4.8: Clean and attack accuracy distribution for 1000 noisy models.

Adaptive Attack. To justify the robustness of our proposed path connection approach to adaptive attacks, we consider the advanced attack setting where the attacker knows path connection is used for defense but cannot compromise the bonafide data that are private to an user. Furthermore, we allow the attacker to use the *same* path training loss function as the defender. To attempt breaking path connection, the attacker trains a compromised path such that every model on this path is a backdoored model and then releases the path-aware tampered models. We show that our approach is still resilient to this adaptive attack.

To attempt breaking path connection, the attacker first separately trains two backdoored models with one poisoned dataset. Then the attacker uses the same poisoned dataset to connect the two models and hence compromises the models on the path. Note that when training this tampered

path, in addition to learning the path parameter θ, the start and end models are not fixed and they are fine-tuned by the poisoned dataset. Next the adversary releases the start and end models $(t = 0, 1)$ from this tampered path. Finally, the defender trains a path from these two models with bonafide data. We conduct the advanced (path-aware) single-target backdoor attack experiments on CIFAR-10 (VGG) by poisoning 10% of images in the training set with a trigger. Figure 4.9 (a) shows the entire path has been successfully compromised due to the attacker's poisoned path training data, yielding less than 5% attack error rate on 10000 test samples. Figure 4.9 (b) shows the defense performance with different number of clean data to connect the two specific models released by the attacker. We find that path connection is still resilient to this advanced attack and most models on the path (e.g. $t \in [0.25, 0.75]$) can be repaired. Although this advanced attack indeed decreases the portion of robust models on the path, it still does not break our defense. In Table 4.5, we also compare the generalization and defense performance and demonstrated that path connection outperforms other approaches.



(a) Path with poisoned data (b) Path with clean data (c) Legend

Figure 4.9: Error rate against path-aware single-target backdoor attacks for CIFAR-10 (VGG).

Table 4.5: Performance against path-aware single-target backdoor attack on CIFAR-10 (VGG).

	Clean accuracy					Backdoor accuracy				
Method / Bonafide data size	2500	1000	500	250	50	2500	1000	500	250	50
Path connection $(t = 0.27)$	90%	87%	83%	81%	75%	3.8%	2.9%	3.6%	4.2%	5.6%
Fine-tune	88%	84%	82%	80%	69%	4.1%	4.6%	4.4%	3.9%	5.9%
Train from scratch	58%	48%	36%	28%	20%	0.6%	0.5%	0.9%	1.7%	1.9%
Noisy model $(t = 0)$	38%	38%	38%	38%	38%	91%	91%	91%	91%	91%
Noisy model $(t = 1)$	35%	35%	35%	35%	35%	86%	86%	86%	86%	86%
Prune	87%	85%	83%	81%	79%	29%	48%	69%	77%	81%

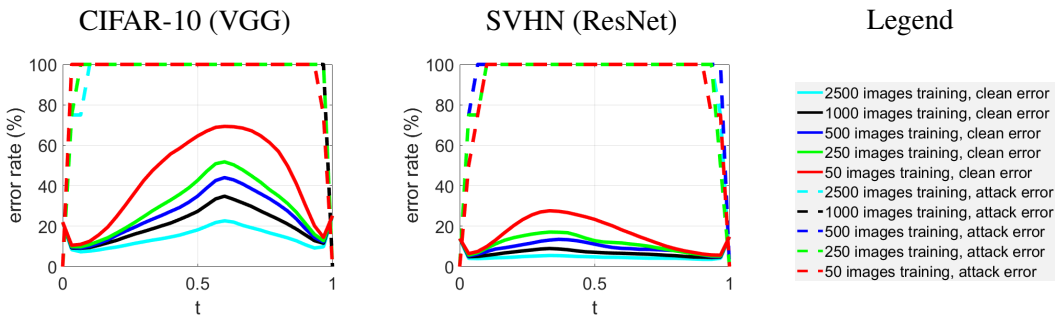


Figure 4.10: Error rate against error-injection attack on the connection path for CIFAR-10 (VGG). The error rate of clean/targeted samples means standard-test-error/attack-failure-rate, respectively.

4.5 Defense Performance against Fault Injection Attack

Attack implementation. We adopt the fault sneaking attack [185] for injecting errors to model weights. Given two untampered and independently trained models, the errors are injected with selected samples as targets such that the tampered models will cause misclassification on targeted inputs and otherwise will behave as untampered models. More details are given in Section 4.3.2.

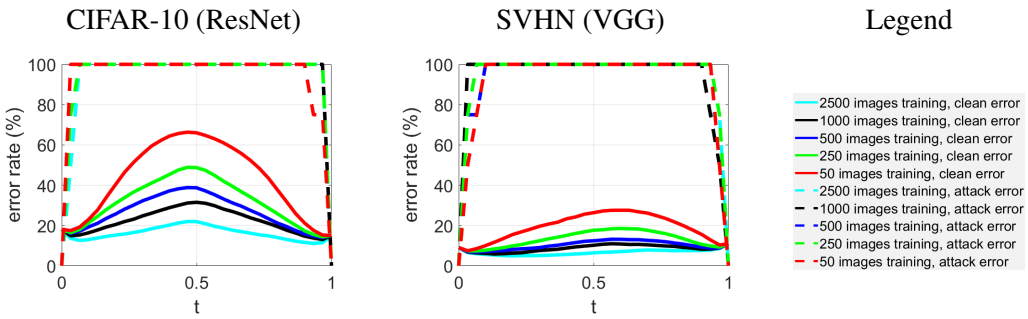


Figure 4.11: Error rate against error-injection attack on the connection path for CIFAR-10 (VGG). The error rate of clean/targeted samples means standard-test-error/attack-failure-rate, respectively.

Evaluation and analysis. Similar to the setup in Section 4.4, Figure 4.10 and Figure 4.11 show the clean accuracy and attack accuracy of the models on the path connecting two error-injected models using limited amount of bonafide data. For the error-injected models ($t = \{0, 1\}$), the attack accuracy is nearly 100%, which corresponds to 0% attack failure rate on targeted samples. However, using path connection and limited amount of bonafide data, the injected errors can be removed almost completely. Varying the size of path training data consistently sanitizes the error-injected models and mainly affects the standard test error. Most of the models on the path can attain nearly 100% fault

Table 4.6: Performance against error-injection attack. The clean/injection accuracy means standard-test-accuracy/attack-success-rate, respectively. Path connection has the best clean accuracy and can completely remove injected errors (i.e. 0% attack accuracy).

		Clean Accuracy					Injection Accuracy				
Dataset	Method / Bonafide data size	2500	1000	500	250	50	2500	1000	500	250	50
CIFAR-10 (VGG)	Path connection ($t = 0.1$)	92%	90%	90%	90%	88%	0%	0%	0%	0%	0%
	Fine-tune	88%	87%	86%	84%	82%	0%	0%	0%	0%	0%
	Train from scratch	45%	37%	27%	25%	10%	0%	0%	0%	0%	0%
	Noisy model ($t = 0$)	14%	14%	14%	14%	14%	36%	36%	36%	36%	36%
	Noisy model ($t = 1$)	12%	12%	12%	12%	12%	19%	19%	19%	19%	19%
	Prune	91%	89%	88%	88%	88%	0%	0%	25%	25%	25%
SVHN (ResNet)	Path connection ($t = 0.1$)	96%	94%	92%	91%	90%	0%	0%	0%	0%	0%
	Fine-tune	94%	93%	91%	89%	88%	0%	25%	0%	25%	25%
	Train from scratch	90%	83%	75%	61%	21%	0%	0%	0%	0%	0%
	Noisy model ($t = 0$)	11%	11%	11%	11%	11%	28%	28%	28%	28%	28%
	Noisy model ($t = 1$)	11%	11%	11%	11%	11%	18%	18%	18%	18%	18%
	Prune	95%	93%	92%	90%	89%	0%	0%	25%	0%	25%

tolerance (i.e. 100% attack failure rate) to the injected errors. The models on the path near $t = 0$ or $t = 1$ have comparable performance on clean data and exhibit strong fault tolerance to injected errors.

Comparison with baselines and extensions. In Table 4.6 and Table 4.7, we adopt the same baselines as in Section 4.4 to compare with path connection. For our proposed path connection method, we train the connection using different number of images as given in Table 4.6 and Table 4.7 for 100 epochs and then report the performance of the model associated with a selected index on the path. The start model and end model have been injected with the same 4 errors (misclassifying 4 given images), starting from two different unperturbed models obtained with different training hyper-parameters.

For the fine-tuning and training-from-scratch methods, we report the model performance after training for 100 epochs. For the random Gaussian perturbation to model weights, we evaluate the model performance under Gaussian noise perturbations on the model parameters. There are two given models which are the models at $t = 0$ and $t = 1$. The Gaussian noise has zero mean with a standard deviation of the absolute value of the difference between the two given models. Then we add the Gaussian noise to the two given models respectively and test their accuracy for clean and triggered images. The experiment is performed multiple times (50 times) and we report the average

Table 4.7: Performance evaluation against error-injection attack. The clean/injection accuracy means standard-test-accuracy/attack-success-rate, respectively. Path connection attains the highest clean accuracy and lowest (0%) attack accuracy among all methods.

		Methods / bonafide data size	2500	1000	500	250	50
CIFAR-10 (ResNet)	Clean Accuracy	Path connection ($t = 0.1$)	87%	83%	81%	80%	77%
		Fine-tune	80%	79%	78%	75%	72%
		Train from scratch	46%	39%	32%	28%	18%
		Noisy model ($t = 0$)	11%	11%	11%	11%	11%
		Noisy model ($t = 1$)	10%	10%	10%	10%	10%
		Prune	84%	82%	81%	80%	78%
	Injection Accuracy	Path connection ($t = 0.1$)	0%	0%	0%	0%	0%
		Fine-tune	0%	0%	0%	25%	25%
		Train from scratch	0%	0%	0%	0%	0%
		Noisy model ($t = 0$)	33%	33%	33%	33%	33%
		Noisy model ($t = 1$)	26%	26%	26%	26%	26%
		Prune	0 %	0%	0%	25%	25%
SVHN (VGG)	Clean Accuracy	Path connection ($t = 0.1$)	96%	94%	93%	91%	90%
		Fine-tune	94%	93%	92%	90%	81%
		Train from scratch	88%	82%	76%	68%	28%
		Noisy model ($t = 0$)	33%	33%	33%	33%	33%
		Noisy model ($t = 1$)	37%	37%	37%	37%	37%
		Prune	94%	93%	92%	90%	89%
	Injection Accuracy	Path connection ($t = 0.1$)	0%	0%	0%	0%	0%
		Fine-tune	0%	0%	25%	0%	25%
		Train from scratch	0%	0%	0%	0%	0%
		Noisy model ($t = 0$)	24%	24%	24%	24%	24%
		Noisy model ($t = 1$)	29%	29%	29%	29%	29%
		Prune	0%	25%	0%	25%	25%

accuracy. We can see that adding Gaussian noise perturbations to the model does not necessarily change the model status from robust to non-robust or from non-robust to robust.

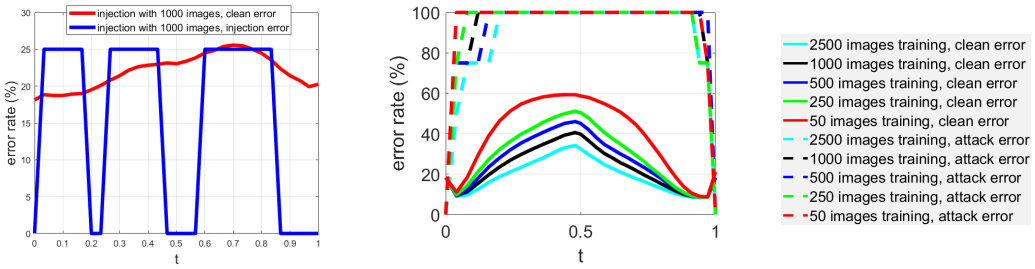
The path connection or evolution from the model at $t = 0$ to that $t = 1$ follows a specific path achieving robustness against backdoor attack rather than random exploration. The training-from-scratch baseline method usually obtains the lowest clean test accuracy, especially in the case of training with 50 images, where training with so little images does not improve the accuracy.

We find that only path connection and training-from-scratch can successfully sanitize the error-injected models and attain 0% attack accuracy, and other baselines are less effective. Table 4.6 also shows the clean accuracy of path connection is substantially better than the effective baselines,

suggesting novel applications of mode connectivity for finding accurate and adversarially robust models.

Technical explanations and adaptive attack. Consistent with the results in backdoor attacks, we explore the model weight space to demonstrate the significant difference between the models on our connection path and random noisy models. In addition, our path connection is resilient to the advanced path-aware error-injection attack.

For the advanced attack in the error-injection setting, the attacker first trains two models with injected errors and then connects the two models with clean data. Then, the attacker tries to inject errors to the three models w_1, w_2 and θ. Note that based on Eq. (4.3) or (4.4), all the models on the path can be expressed as a combination of these three models. Thus, the whole path is expected to be tampered. Next, the attacker releases the start and end models from the "bad" path. Finally, the defender trains a path from these two models with bonafide data. We conduct the advanced (path-aware) error-injection attack experiments on CIFAR-10 (VGG) by injecting 4 errors. Figure 4.12 (a) shows that the entire path has been successfully compromised by the advanced path-aware injection. While the clean error rate is stable across the path, at least 3 out of 4 injected errors (corresponding to 25% attack error rate) yield successful attacks. After training a path to connect the two released models with the bonafide data, the clean error and attack error are shown in Figure 4.12 (b). We can observe that path connection can effectively eliminate the injected errors on the path (i.e., high attack error rate).



(a) Path with injection attack (b) Path without injection attack (c) Legend

Figure 4.12: Error rate against path-aware error-injection attacks for CIFAR-10 (VGG).

4.6 Defense Performance against Adversarial Attack

To gain insights on mode connectivity against evasion attacks, here we investigate the standard and adversarial-robustness loss landscapes on the same path connecting two untampered

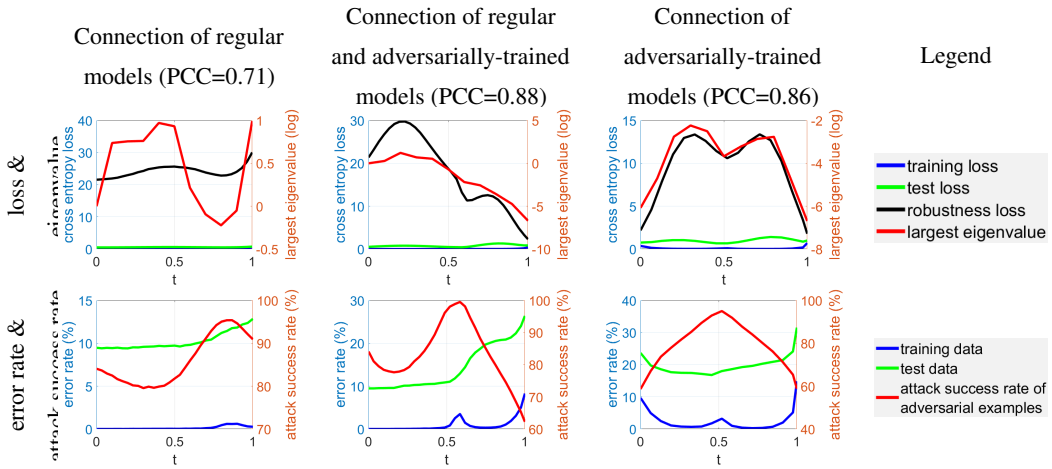


Figure 4.13: Loss, error rate, attack success rate and largest eigenvalue of input Hessian on the path connecting different model pairs on CIFAR-10 (VGG) using standard loss. The error rate of training/test data means standard training/test error, respectively. In all cases, there is no standard loss barrier but a robustness loss barrier. There is also a high correlation between the robustness loss and the largest eigenvalue of input Hessian, and their Pearson correlation coefficient (PCC) is reported in the title.

and independently trained models. The path is trained using the entire training dataset for minimizing (4.2) with standard cross entropy loss. The robustness loss refers to the cross entropy of class predictions on adversarial examples generated by evasion attacks and their original class labels. Higher robustness loss suggests the model is more vulnerable to evasion attacks. In addition, we will investigate the robustness loss landscape connecting regular (non-robust) and adversarially-trained (robust) models, where the path is also trained with standard cross entropy loss. We will also study the behavior of the largest eigenvalue of the Hessian matrix associated with the cross entropy loss and the data input, which we call the input Hessian. As adversarial examples are often generated by using the input gradients, we believe the largest eigenvalue of the input Hessian can offer new insights on robustness loss landscape, similar to the role of model-weight Hessian on quantifying generalization performance [156, 167].

Attack Implementation. We uniformly select 9 models ($t = \{0.1, 0.2, \ldots, 0.9\}$) on the path and run evasion attacks on each of them using the ℓ_∞-norm-ball based projected gradient descent (PGD) method proposed in [106]. The robustness loss is evaluated using the non-targeted adversarial examples crafted from the entire test set, and the attack perturbation strength is set to $\epsilon = 8/255$ with 10 iterations. We also use the PGD method for adversarial training to obtain adversarially-trained models that are robust to evasion attacks but pay the price of reduced accuracy on clean data [106].

As the adversarial examples perform small perturbations around the clean images to increase the robustness loss function, thus incurring a misclassification, it relates to the Hessian matrix of the loss function with reference to the input images. A Hessian matrix (or Hessian) is the second-order partial derivatives of a scalar-valued function, describing the local curvature of a function with many variables. In general, large Hessian spectrum means the function reaches a sharp minima, thus leading to a more vulnerable model as the input can leave this minima with small distortions. By contrast, in the case of flat minima with small Hessian spectrum, it takes more efforts for the input to leave the minima.

As we find the robustness loss barrier on the path, we are interested in the evolution of the input Hessian for the models on the path. We uniformly pick the models on the connection and compute the largest eigenvalue of the Hessian w.r.t. to the inputs. To deal with the high dimension difficulties for the Hessian calculation, the power iteration method [110] is adopted to compute the largest eigenvalue of the input Hessian with the first-order derivatives obtained by back-propagating. Unless otherwise specified, we continue the power iterations until reaching a relative error of 1E-4.

For ease of visual comparison, in Figure 4.13, we plot the log value of the largest eigenvalue of the input Hessian together with the error rate and loss for clean images and adversarial examples. We note that the Pearson correlation coefficient (PCC) is indeed computed using the original largest eigenvalue of input Hessian and robustness loss. As demonstrated in Figure 4.13, the evolution of the input Hessian is very similar to the change of the loss of adversarial examples. As we can see, the largest eigenvalue on the path does not necessarily have a high correlation with the error rate of the clean images in the training set and test set or the adversarial examples. Instead, it seems to be highly correlated with the loss of adversarial examples as they share very similar shapes. This inspires us to explore the relationship between the largest eigenvalue of the input Hessian and the robustness loss.

Evaluation and Analysis. To study the standard and robustness loss landscapes, we scrutinize the models on the path connecting the following pairs of models: (i) independently trained regular (non-robust) models; (ii) regular to adversarially-trained models; and (iii) independently adversarially-trained models. These results are shown in Figure 4.13. For ease of visual comparison, in Figure 4.13, we plot the log value of the largest eigenvalue of the input Hessian together with the error rate and loss for clean images and adversarial examples. We note that the Pearson correlation coefficient (PCC) is indeed computed using the original largest eigenvalue of input Hessian and robustness loss. As demonstrated in Figure 4.13, the evolution of the input Hessian is very similar to the change of the loss of adversarial examples. As we can see, the largest eigenvalue on the path does not necessarily have a high correlation with the error rate of the clean images in the training set

and test set or the adversarial examples. Instead, it seems to be highly correlated with the loss of adversarial examples as they share very similar shapes. This inspires us to explore the relationship between the largest eigenvalue of the input Hessian and the robustness loss.

We summarize the major findings as follows.

- No standard loss barrier in all cases: Regardless of the model pairs, all models on the paths have similar standard loss metrics in terms of training and test losses, which are consistent with the previous results on the "flat" standard loss landscape for mode connectivity [40, 53, 58]. The curve of standard loss in case (ii) is skewed toward one end due to the artifact that the adversarially-trained model ($t = 1$) has a higher training/test error than the regular model ($t = 0$).

- Robustness loss barrier: Unlike standard loss, we find that the robustness loss on the connection path has a very distinct characteristic. In all cases, there is a robustness loss barrier (a hill) between pairs of regular and adversarially-trained models. The gap (height) of the robustness loss barrier is more apparent in cases (ii) and (iii). For (ii), the existence of a barrier suggests the modes of regular and adversarially-trained models are not connected by the path in terms of robustness loss, which also provides a geometrical evidence of the "no free lunch" hypothesis that adversarially robust models cannot be obtained without additional costs [21, 37, 152]. For (iii), robustness loss barriers also exist. The models on the path are less robust than the two adversarially-trained models at the path end, despite they have similar standard losses. The results suggest that there are essentially no better adversarially robust models on the path connected by regular training using standard loss.

- High correlation between the largest eigenvalue of input Hessian and robustness loss: Inspecting the largest eigenvalue of input Hessian $H_t(x)$ of a data input x on the path, denoted by $\lambda_{\max}(t)$, we observe a strong accordance between $\lambda_{\max}(t)$ and robustness loss on the path, verified by the high empirical Pearson correlation coefficient (PCC) averaged over the entire test set as reported in Figure 4.13. As evasion attacks often use input gradients to craft adversarial perturbations to x, the eigenspectrum of input Hessian indicates its local loss curvature and relates to adversarial robustness [172]. Below we provide technical explanations for the empirically observed high correlation between $\lambda_{\max}(t)$ and the oracle robustness loss on the path, defined as $\max_{\|\delta\| \leq \epsilon} l(w(t), x + \delta)$.

To investigate the high correlation between the largest eigenvalue of input Hessian and robustness loss, we drop the the notation dependency on the input data sample x. It also suffices to

consider two models $w := w(t)$ and $w + \Delta w := w(t + \Delta t)$ for some small Δt on the path for our analysis. We begin by proving the following lemma.

Lemma 4.6.1. *Given assumption (a), for any vector norm $\|\cdot\|$, for any data sample x,*

$$\|\nabla_x l\,(w + \Delta w, x)\| - \|\nabla_x l\,(w, x)\| = 0 \tag{4.5}$$

Proof. With assumption (a), we have $l\,(w + \Delta w, x) = l\,(w, x) = \text{const}$. Differentiating at both sides with respect to x and then applying the vector norm gives the lemma. □

Proposition 4.6.1. *Let $f_w(\cdot)$ be a neural network classifier with its model weights denoted by w and let $l(w, x)$ denote the classification loss (e.g. cross entropy of $f_w(x)$ and the true label y of a data sample x). Consider the oracle robustness loss $\max_{\|\delta\|\le\epsilon} \ell(w(t), x + \delta)$ of the model t on the path, where δ denotes a perturbation to x confined by an ϵ-ball induced by a vector norm $\|\cdot\|$. Assume*
(a) the standard loss $l(w(t), x)$ on the path is a constant for all $t \in [0, 1]$.
(b) $l(w(t), x + \delta) \approx l(w(t), x) + \nabla_x l(w(t), x)^T \delta + \frac{1}{2}\delta^T H_t(x)\delta$ for small δ, where $\nabla_x l(w(t), x)$ is the input gradient and $H_t(x)$ is the input Hessian of $l(w(t), x)$ at x.
Let c denote the normalized inner product in absolute value for the largest eigenvector v of $H_t(x)$ and $\nabla_x l(w(t), x)$, $\frac{|\nabla_x l(w(t),x)^T v|}{\|\nabla_x l(w(t),x)\|} = c$. Then we have $\max_{\|\delta\|\le\epsilon} l(w(t), x + \delta) \sim \lambda_{\max}(t)$ as $c \to 1$.

Proof. As the adversarial perturbation is generated using the PGD method [106] with an ϵ-ℓ_∞ ball constraint, the optimal first-order solution of the perturbation is

$$\delta^* = \epsilon \cdot \frac{\nabla f\,(x)}{\|\nabla f\,(x)\|} \tag{4.6}$$

Assume the PGD method can well approximate the robustness loss, the difference of the robustness losses of the two models w and $w + \Delta w$ on the path can be represented as

$$\begin{aligned}
&\max_{\|\delta\|\le\epsilon} l\,(w + \Delta w, x + \delta) \;-\; \max_{\|\delta\|\le\epsilon} l\,(w, x + \delta) \\
&= l\left(w + \Delta w, x + \delta^*_{w+\Delta w}\right) - l\,(w, x + \delta^*_w) && (4.7)\\
&= \left[l\,(w + \Delta w, x) + \epsilon \frac{\nabla_x l\,(w + \Delta w, x)^T \nabla_x l\,(w + \Delta w, x)}{\|\nabla_x l\,(w + \Delta w, x)\|} + \frac{\epsilon^2}{2} \frac{\nabla_x l\,(w + \Delta w, x)^T}{\|\nabla_x l\,(w + \Delta w, x)\|} H_{w+\Delta w} \frac{\nabla_x l\,(w + \Delta w, x)}{\|\nabla_x l\,(w + \Delta w, x)\|}\right] \\
&- \left[l\,(w, x) + \epsilon \frac{\nabla_x l\,(w, x)^T \nabla_x l\,(w, x)}{\|\nabla_x l\,(w, x)\|} + \frac{\epsilon^2}{2} \frac{\nabla_x l\,(w, x)^T}{\|\nabla_x l\,(w, x)\|} H_w \frac{\nabla_x l\,(w, x)}{\|\nabla_x l\,(w, x)\|}\right] + O(\epsilon^3) && (4.8)\\
&\approx l\,(w + \Delta w, x) - l\,(w, x) + \epsilon\left[\|\nabla_x l\,(w + \Delta w, x)\| - \|\nabla_x l\,(w, x)\|\right] \\
&+ \frac{\epsilon^2}{2}\left[\frac{\nabla_x l\,(w + \Delta w, x)^T}{\|\nabla_x l\,(w + \Delta w, x)\|} H_{w+\Delta w} \frac{\nabla_x l\,(w + \Delta w, x)}{\|\nabla_x l\,(w + \Delta w, x)\|} - \frac{\nabla_x l\,(w, x)^T}{\|\nabla_x l\,(w, x)\|} H_w \frac{\nabla_x l\,(w, x)}{\|\nabla_x l\,(w, x)\|}\right] && (4.9)\\
&= \frac{\epsilon^2}{2}\left[\frac{\nabla_x l\,(w + \Delta w, x)^T}{\|\nabla_x l\,(w + \Delta w, x)\|} H_{w+\Delta w} \frac{\nabla_x l\,(w + \Delta w, x)}{\|\nabla_x l\,(w + \Delta w, x)\|} - \frac{\nabla_x l\,(w, x)^T}{\|\nabla_x l\,(w, x)\|} H_w \frac{\nabla_x l\,(w, x)}{\|\nabla_x l\,(w, x)\|}\right] && (4.10)
\end{aligned}$$

The first equality uses the definition of optimal first-order solution of the robustness loss. Using (4.6) and Taylor series expansion on $\ell(\cdot,\cdot)$ with respect to its second argument gives (4.8). Rearranging (4.8) and using Assumption (b) (i.e., ignoring the $O(\epsilon^3)$ term in (4.8)) gives (4.9). Finally, Based on (4.9), the term $l\left(w+\Delta w,x\right)-l\left(w,x\right)=0$ due to Assumption (a). Then, applying Lemma 4.6.1 to (4.9) gives (4.10).

Without loss of generality, the following analysis assumes $\|\cdot\|$ to be the Euclidean norm and consider the case when $\frac{\nabla_x l(w,x)^T v}{\|\nabla_x l(w,x)\|}=c$. The results can generalize to other norms since the correlation is invariant to scaling factors. With assumption (c), for any input Hessian H and its largest eigenvector v, we have

$$\begin{aligned}
&\frac{\nabla_x l\left(w,x\right)^T}{\|\nabla_x l\left(w,x\right)\|}H\frac{\nabla_x l\left(w,x\right)}{\|\nabla_x l\left(w,x\right)\|}\\
&=\left(\frac{\nabla_x l\left(w,x\right)}{\|\nabla_x l\left(w,x\right)\|}-v+v\right)^T H\left(\frac{\nabla_x l\left(w,x\right)}{\|\nabla_x l\left(w,x\right)\|}-v+v\right)\\
&=\left(\frac{\nabla_x l\left(w,x\right)}{\|\nabla_x l\left(w,x\right)\|}-v\right)^T H\left(\frac{\nabla_x l\left(w,x\right)}{\|\nabla_x l\left(w,x\right)\|}-v\right)+2\frac{\nabla_x l\left(w,x\right)^T}{\|\nabla_x l\left(w,x\right)\|}Hv-\lambda_{\max}\\
&=(2c-1)\lambda_{\max}+h_x,
\end{aligned}\tag{4.11}$$

where $h_x:=\left(\frac{\nabla_x l(w,x)}{\|\nabla_x l(w,x)\|}-v\right)^T H\left(\frac{\nabla_x l(w,x)}{\|\nabla_x l(w,x)\|}-v\right)$ and the last equality uses the fact that $Hv=\lambda_{\max}\cdot v$ and assumption (c). Note that the lower bound is tight when v and $\nabla_x l\left(w,x\right)$ are collinear, i.e., $v=\frac{\nabla_x l(w,x)^T}{\|\nabla_x l(w,x)\|}$, as in this case $h_x=0$ and $c=1$. On the other hand, using the definition of largest eigenvalue gives an upper bound

$$\frac{\nabla_x l\left(w+\Delta w,x\right)^T}{\|\nabla_x l\left(w+\Delta w,x\right)\|}H\frac{\nabla_x l\left(w+\Delta w,x\right)}{\|\nabla_x l\left(w+\Delta w,x\right)\|}\leq\lambda_{\max}.\tag{4.12}$$

Finally, applying (4.12) and (4.11) to (4.10), we have

$$\max_{\|\delta\|\leq\epsilon} l\left(w+\Delta w,x+\delta\right)\;-\;\max_{\|\delta\|\leq\epsilon} l\left(w,x+\delta\right)\;\leq\frac{\epsilon^2}{2}\left[\lambda_{\max}(t+\Delta t)-(2c-1)\lambda_{\max}(t)-h_x(t)\right]\tag{4.13}$$

and

$$\max_{\|\delta\|\leq\epsilon} l\left(w+\Delta w,x+\delta\right)\;-\;\max_{\|\delta\|\leq\epsilon} l\left(w,x+\delta\right)\;\geq\frac{\epsilon^2}{2}\left[(2c-1)\lambda_{\max}(t+\Delta t)-\lambda_{\max}(t)+h_x(t+\Delta t)\right]\tag{4.14}$$

As v becomes more aligned with $\nabla_x l\,(w, x)$, c increases toward 1 and h_x decreases toward 0, implying $2c - 1 \approx 1$. Therefore, in this case we have $\max_{\|\delta\| \leq \epsilon} l\,(w + \Delta w, x + \delta) - \max_{\|\delta\| \leq \epsilon} l\,(w, x + \delta)$ as a linear function of $\lambda_{\max}(t + \Delta t) - \lambda_{\max}(t)$ and $\max_{\|\delta\| \leq \epsilon} l\,(w(t), x + \delta) \sim \lambda_{\max}(t)$.

If the higher order term $O(\epsilon^3)$ in (4.8) is non-negligible, which means Assumption (b) does not hold, then the following analysis will have an extra offset term of the order $O(\epsilon)$ as c increases toward 1, i.e., $\max_{\|\delta\| \leq \epsilon} l\,(w(t), x + \delta) \sim \lambda_{\max}(t) + O(\epsilon)$. □

Assumption (a) follows by the existence of high-accuracy path of standard loss landscape from mode connectivity analysis. Assumption (b) assumes the local landscape with respect to the input x can be well captured by its second-order curvature based on Taylor expansion. The value of c is usually quite large, which has been empirically verified in both regular and adversarially-trained models [112].

Extensions. Although we find that there is a robustness loss barrier on the path connected by regular training, we conduct additional experiments to show that it is possible to find an robust path connecting two adversarially-trained or regularly-trained model pairs using adversarial training [106], which we call the "robust connection" method. However, model ensembling using either the regular connection or robust connection has little gain against evasion attacks, as the adversarial examples are known to transfer between similar models [121, 145].

To train a robust path, we minimize the expected maximum loss on the path like the following,

$$\min_{\theta} E_{t \sim U(0,1)} \left[\max_{\delta \in S} L(\phi_\theta(t), \mathbf{x} + \delta) \right] \tag{4.15}$$

where

$$S = \left\{ \delta \,\middle|\, \mathbf{x} + \delta \in [0, 1]^d, \|\delta\|_\infty \leq \epsilon \right\} \tag{4.16}$$

To solve the problem, we first sample $\tilde{t}$ from the uniform distribution $U(0, 1)$ and we can obtain the model $\phi_\theta(\tilde{t})$. Based on the model $\phi_\theta(\tilde{t})$, we find the perturbation maximizing the loss within the range S,

$$\max_{\delta \in S} L(\phi_\theta(\tilde{t}), \mathbf{x} + \delta) \tag{4.17}$$

We can use projected gradient descent method for the maximization.

$$\delta_{k+1} = \prod_{S} \left(\delta_k + \eta \cdot \operatorname{sgn}\left(\nabla_\delta L(\phi_\theta(\tilde{t}), \mathbf{x} + \delta) \right) \right), \tag{4.18}$$

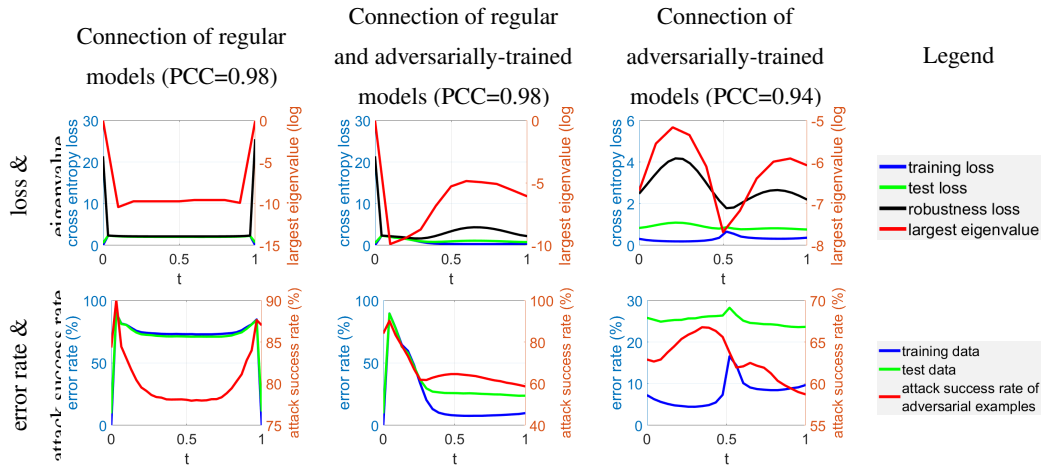


Figure 4.14: Loss, error rate, attack success rate and largest eigenvalue of input Hessian on the path connecting different model pairs on CIFAR-10 (VGG) using robust connection. The path is obtained with robust training method. The error rate of training/test data means standard training/test error, respectively. There is still a robustness loss barrier between non-robust and robust models, but not there is no robustness loss barrier between robust model pair and non-robust model pair (verified by small loss variance and flat attack success rate on the path). There is also a high correlation between the robustness loss and the largest eigenvalue of input Hessian, and their Pearson correlation coefficient (PCC) is given in the title of each plot.

where $\prod_S$ denotes the projection to the feasible perturbation space S, and $\mathrm{sgn}(\cdot)$ denotes element-wise sign function taking the value of either 1 or -1.

After finding the perturbation $\hat{\delta}$ maximizing the loss, we would minimize the expectation. At each iteration, we make a gradient step for $\boldsymbol{\theta}$ as follows,

$$\theta = \theta - \eta \cdot \nabla_\theta L(\phi_\theta(\tilde{t}), \mathbf{x} + \hat{\delta}) \tag{4.19}$$

We show the robust connection for a pair of non-robust and non-robust models, a pair of non-robust and robust models, and a pair of robust models in Figure 4.14. We can observe that with the robust training method, there is still a robustness loss barrier between the non-robust and robust models. However, for the robust connection of robust model pair and non-robust (regular) model pairs, there is no robustness loss barrier, verified by small loss variance and flat attack success rate on the path. Our results also suggest that there is always a loss barrier between non-robust and robust models, no matter using standard or robust loss functions for path connection, as intuitively the two models are indeed not connected in their respective loss landscapes. Moreover, the attack success rate of adversarial examples are relatively small on the whole path compared with the robust connection of non-robust and robust models.

Here we test the performance of (naive) model ensembling against evasion attacks. Given two untampered and independently trained CIFAR-10 (VGG) models, we first build a regular connection of them. Then we randomly choose models on the path (randomly choose the value of t) and take the average output of these models as the final output if given an input image for classification. The adversarial examples are generated based on the start model $(t = 0)$ or end model $(t = 1)$ and we assume the attacker does not have any knowledge about the connection path nor the models on the path. We use these adversarial examples to test the performance of the model ensembling with the models on the connection. The attack success rate of adversarial examples can decrease from 85% to 79%. The defense improvement of this naive model ensembling strategy is not very significant, possibly due to the well-known transferrability of adversarial examples [121, 145]. Similar findings can be concluded for the robust connection method.

Chapter 5

DNN Deployment

5.1 Motivation

As the rapid development of the autonomous vehicles, which attempts to navigate roadways without human intervention, the environment sensing (known as perception) serves as a fundamental building block. Among various subtasks of perception, the object detection including 2D and 3D detection is one of the most important prerequisites to autonomous navigation. 2D and 3D detection makes use of the 2D image and 3D point clouds information from camera and LiDAR sensors, respectively, to detect objects in the environments, thus providing the autonomous navigation with the desirable knowledge about its environment and enabling high-level navigation computations and optimizations.

It is essential to implement real-time object detection for both 2D and 3D detection on autonomous vehicles as they need to interact with the environment instantaneously to avoid potential accidents. However, as 2D and 3D object detections are implemented with deep neural networks (DNNs) such as YOLO [15] and PointPillars [83], respectively, with tremendous memory and computation requirements, it is challenging to achieve real-time on autonomous vehicles with limited memory and computation resources. The more powerful high-end GPUs are usually too costly and power-hungry to be deployed on vehicles. Thus, it is desirable to satisfy the real-time requirement within the limited memory and computation constraints for detection models.

DNN weight pruning [61, 160] has been proved as an effective model compression technique to remove redundant weights, thereby reducing computations and accelerating inference. Existing work mainly focus on *unstructured pruning* [61, 62, 103] where arbitrary weight can be removed, and (coarse-grained) *structured pruning* [65, 95, 103, 104, 160, 173] to eliminate whole

filters. The former results in high accuracy but limited hardware parallelism (and acceleration), while the latter is the opposite.

Recent work [38, 105, 117] proposed the novel concept of *fine-grained structured pruning* scheme: *Pattern-based pruning* [105, 117] assigns potentially different patterns to convolutional (CONV) kernels, while *block-based pruning* [38] divides the weight matrix into equal-sized blocks and performs independent row/column pruning in each block. High accuracy can be achieved as a result of intra-kernel (or intra-block) flexibility, while high hardware parallelism (and mobile inference acceleration) can be achieved with the assist of compiler-level code generation techniques [117]. Despite the promising results, pattern-based pruning [105, 117] is only applied to 3×3 CONV layers, while block-based pruning [38] is only suitable for fully-connected (FC) layers, which limit their applicability.

As the **first contribution**, we generalize the concept of fine-grained structured pruning to cover all types of CONV and FC layers, which is necessary for both 2D and 3D real-time object detection of autodriving. In this way, we can fully leverage the new opportunity to achieve high accuracy and high acceleration simultaneously *with the aid of advanced compiler optimizations*. Block-based pruning scheme can be naturally extended to 1×1 CONV layers. We extend pattern-based pruning to larger kernel sizes beyond 3×3, overcoming the limitation of increased computation overhead with an increasing number of pattern types. We develop a comprehensive, compiler-based automatic code generation framework *supporting different (proposed and existing) pruning schemes in a unified manner*, and *different rates for different layers*.

While our compiler optimizations provide notable mobile acceleration and support of various sparsity schemes, it introduces *a larger model optimization space* beyond the scope of prior work: Different sparsity schemes result in different accuracy performances, and different accelerations under compiler optimizations. Motivated by the recent idea of Neural Architecture Search (NAS) [94, 190], we propose to perform a novel *compiler-aware neural pruning search*, automatically determining the pruning scheme and rate (including bypass) for each individual layer. The *objective* is to maximize accuracy satisfying an inference latency constraint on the target mobile device. The DNN latency will be actually measured on target device, thanks to the fast auto-tuning capability of our compiler for efficient inference on different mobile devices.

Our framework employs pre-trained DNN models as starting point, which are already optimized for high detection accuracy in autodriving tasks to accelerate the pruning search. The overall latency constraint is satisfied through the synergic efforts of (i) incorporating overall DNN latency constraint into automatic search process, and (ii) effective pruning algorithm to perform

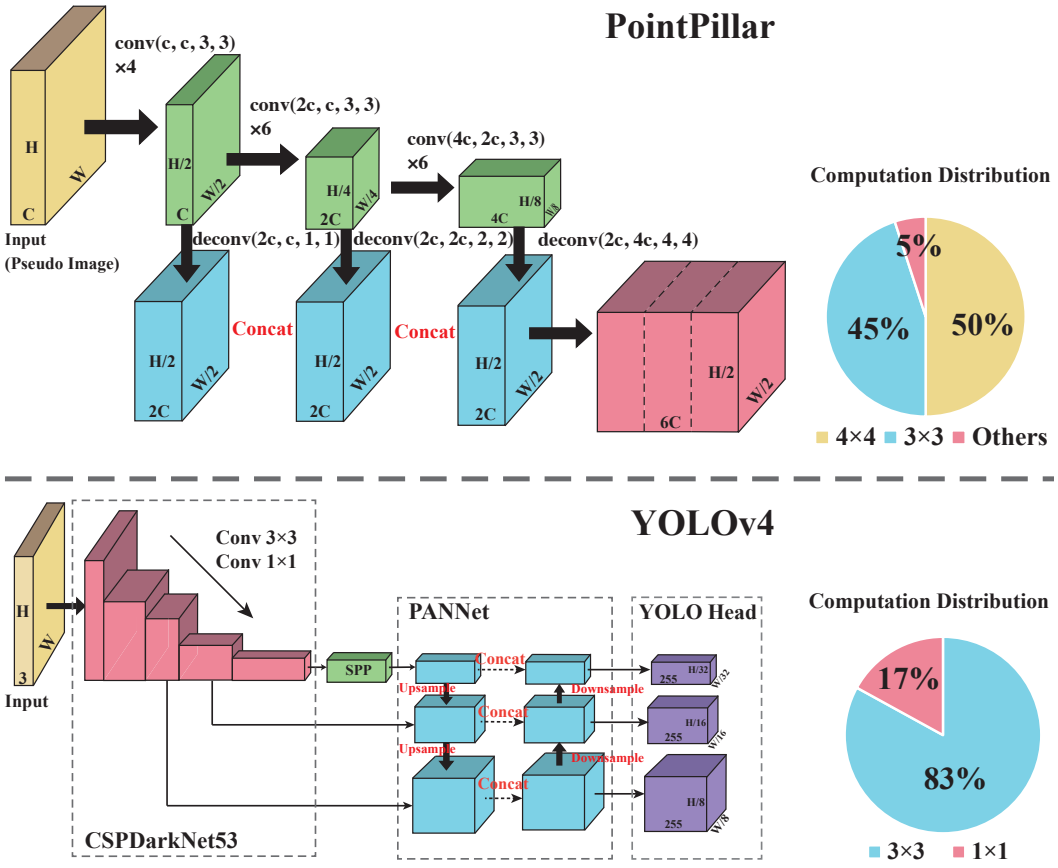


Figure 5.1: The PointPillars data flow for 3D detection and YOLOv4 data flow for 2D object detection with computation distributions.

weight training/pruning accordingly. To perform efficient search under a large search space, we propose a meta-modeling procedure with Bayesian optimization (BO). We can achieve (close-to) real-time, 55ms and 97ms inference times for YOLOv4 based 2D detection and PointPillars based 3D detection, respectively, on a mobile phone with minor (or no) accuracy loss. Our method on 2D detection outperforms other acceleration frameworks such as TVM [29] and MNN [1], while we are the first to support 3D detection on mobile.

5.2 Background on Various Pruning Schemes

2D object detection detects various objects including pedestrians, cyclists and cars from 2D camera images. Similarly, 3D object detection detects objects in the environment from 3D LiDAR images with point clouds. They are crucial for autodriving perception to gain basic knowledge about

its environment and almost all of the navigation decisions are built on the detection information. YOLOv4 [15] and PointPillars [83] are two popular DNN models for 2D and 3D object detection, respectively. YOLOv4 has a backbone and a head to detect and regress 2D boxes. Similarly, a PointPillar model consists of three main stages: (1) A feature encoder network to convert a point cloud to a sparse pseudo-image; (2) a 2D CONV backbone to process the pseudo-image into high-level representation; and (3) a detection head to regress 3D boxes. We show the data flow and computation distribution for YOLOv4 and PointPillars in Fig. 5.1. We observe that (1) there are various kinds of CONV operations in the model such as 1×1, 3×3 and 4×4 (The operations in DECONV layers are also CONV), and (2) the CONV operations take up most of the computations in models.

5.2.1 Weight Pruning: Schemes and Algorithms

Existing weight pruning research can be categorized according to pruning schemes and pruning algorithms.

Pruning Scheme: Previous weight pruning work can be categorized according to the pruning scheme: *unstructured pruning* [61, 62, 107], *coarse-grained structured pruning* [65, 103, 104, 160], and fine-grained structured pruning schemes including *pattern-based* [105, 117] and *block-based* [38] pruning.

Unstructured pruning (Fig. 5.2 (a) and (b)) removes weights at arbitrary position. Despite the large compression rate, the irregular sparse weight matrix with indices damages the parallel implementations, leading to limited acceleration on hardware.

To overcome this limitation, many work [65,104,160,173] studied coarse-grained structured pruning at the level of filters and channels as shown in Fig. 5.2 (c) and (d). With the elimination of filters/channels, the pruned model still maintains the network structure with high regularity which can be parallelized on hardware. The downside is the obvious accuracy degradation, which limits model compression rate.

Fig. 5.2 (e) and (f) show pattern-based pruning [105, 117] and block-based pruning [38], respectively, as representative fine-grained structured pruning schemes. The former is applied to 3×3 CONV layers, and the latter to FC layers, in the original papers. Pattern-based pruning assigns a pattern (from a predefined library) to each CONV kernel, maintaining a fixed number of weights in each kernel. As shown in the figure, each kernel reserves 4 non-zero weights (on a pattern) out of the original 3×3 kernels. Besides being assigned a pattern, a kernel can be completely removed

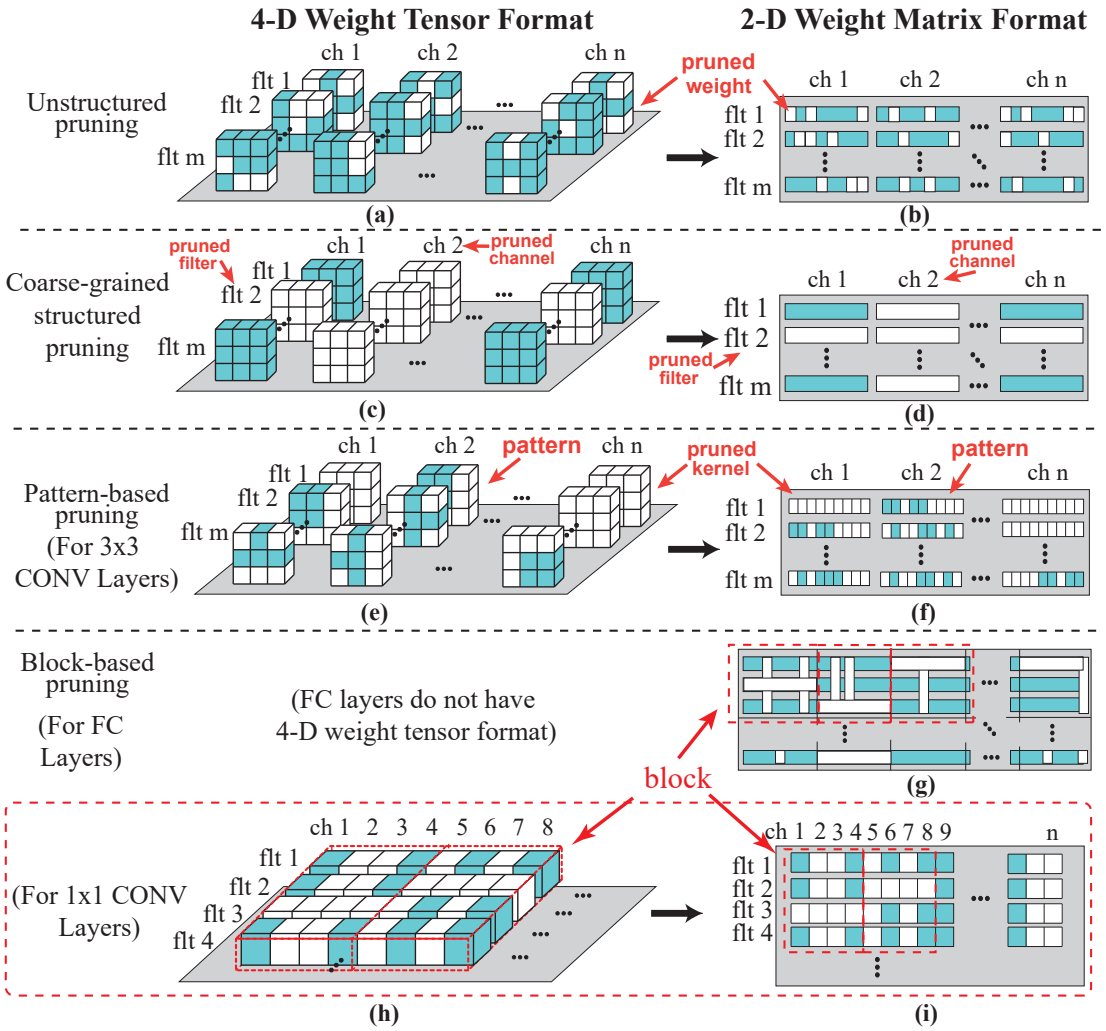


Figure 5.2: Different weight pruning schemes for CONV layers using 4D tensor and 2D matrix representation.

to achieve higher compression rate. On the other hand, block-based pruning divides the FC weight matrix into equal-sized blocks and performs independent row/column pruning in each block.

For these fine-grained structured pruning schemes, high accuracy can be achieved as a result of intra-kernel (or intra-block) flexibility, while high hardware parallelism (and mobile acceleration) can be achieved with the assist of compiler-based code generation. For pattern-based pruning [117], compiler can perform filter reorder to group kernels with the same pattern, and allocate a thread to each group, thereby achieving high hardware parallelism. For block-based pruning [38], thanks to the large weight matrices, the remaining computation in each block still achieves high parallelism on mobile CPU/GPU with computation overhead eliminated by compiler.

Our goal is to overcome the limited application of pattern-based and block-based pruning,

and generalize to all types of CONV and FC layers. 1×1 CONV layers do not exhibit intra-kernel flexibility and thus are not suitable for pattern-based pruning, but we found that block-based pruning can be naturally extended to 1×1 layers because such CONV layers are also computed as matrix multiplication [2, 132]. On the other hand, there is a **key difficulty** in generalizing pattern-based pruning to larger kernel sizes beyond 3×3: It results in a large number of pattern types, which incurs notable computation overhead in compiler-generated executable codes.

Pruning Algorithm: Two main categories exist: *heuristic pruning algorithm* [61, 62, 104, 173] and *regularization-based pruning algorithm* [65, 103, 160, 176]. Heuristic pruning was firstly performed in an iterative, magnitude-based manner on unstructured pruning [62], and gets improved in later work [61] and incorporated into coarse-grained structured pruning [39, 104, 173]. Regularization-based algorithm uses mathematics-oriented method to deal with pruning problem. Early work [65, 160] incorporates ℓ_1 or ℓ_2 regularization in loss function to solve filter/channel pruning problems. In [176], an advanced optimization solution framework ADMM (Alternating Direction Methods of Multipliers) is utilized to achieve dynamic regularization penalty which notably reduces accuracy loss.

5.3 Compiler Optimization

We develop a comprehensive, compiler-based automatic code generation framework supporting the pattern-based pruning schemes for various CONV kernel sizes in a unified manner. It also supports other pruning schemes such as unstructured, coarse-grained structured, block-based pruning. Fast *auto-tuning* capability is incorporated for efficient end-to-end inference on different mobile CPU/GPU. We achieve superior inference acceleration on both dense (before pruning) and sparse DNN models, as to be shown in the experimental results.

5.4 Neural Pruning Search

5.4.1 Overview

As shown in Fig. 5.3, the framework consists of two basic components: a *controller* and an *evaluator*. The controller first generates various *pruning proposals* from the search space. Then the evaluator evaluates their detection accuracy and speed performance. Based on the performance, the evaluator provides guidance for controller about what a satisfying pruning proposal looks like. Next

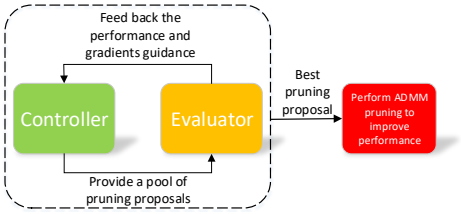


Figure 5.3: Automatic network pruning search framework

Table 5.1: Search space for each DNN layer

Pruning scheme	{Filter [189], Pattern-based, Block-based [38]}
Pruning rate	{ 1×, 2×, 2.5×, 3×, 5×, 7×, 10×, skip }

the controller generates new pruning proposals with the guidance. After iterations, the controller outputs the best pruning proposal with desirable detection performance while satisfying the real-time requirement.

With the derived best pruning proposal, it achieves real-time inference with compiler optimization. Besides the satisfying speed performance, to further improve detection performance, we choose to adopt ADMM pruning [176] to perform an enhanced pruning following the best proposal, after comparing multiple pruning algorithms. It supports different sparsity schemes with the help of group-Lasso regularization [160].

5.4.2 Controller

The controller generates *pruning proposals* from the search space. Each pruning proposal is a directed graph consisting of the pruning scheme and pruning rate for each layer of the model. For example, it has 10 nodes for a 5-layer DNN model.

5.4.2.1 Search Space

Each pruning proposal contains layer-wise pruning scheme and pruning rate, as shown in Tab. 5.1.

Per-layer pruning schemes: The controller can choose from filter (channel) pruning [189], pattern-based pruning (including our extension to larger kernel sizes) and block-based pruning [38] for each layer. As different layers may have different best-suited pruning schemes, we allow the controller the flexibility to choose different pruning schemes for different layers, also supported by our compiler code generation.

Per-layer pruning rate: We can choose from the list $\{1\times, 2\times, 2.5\times, 3\times, 5\times, 7\times, 10\times, \text{skip}\}$, where $1\times$ means the layer is not pruned, and "skip" means bypassing this layer.

5.4.2.2 Pruning Proposal Updating

The evaluator provides the *gradients guidance* specified in Sec. 5.4.3.2 to guide the controller for generating new pruning proposals. The gradients guidance contains a currently best pruning proposal and its corresponding replacement probability obtained from its gradients. The controller determines whether to replace each node in the proposal according to the replacement probability. Next if replaced, the controller chooses randomly from two nodes with the lowest probabilities as its replacement.

5.4.3 Evaluator

The evaluator needs to evaluate pruning proposal performance. We define the performance measurement (reward) as:

$$r = V - \alpha \cdot \max(0, t - T), \tag{5.1}$$

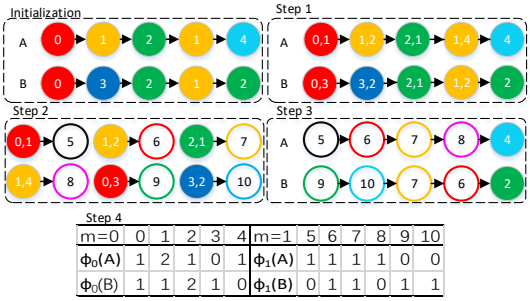
where V is the validation mean average precision (mAP) of the model, t is the model inference speed or latency, which is actually measured on a mobile device[1]. T is the threshold for the latency requirement. Generally, r is high when the model satisfies real-time requirement ($t < T$) with high mAP. Otherwise r is small, especially when the real-time latency requirement is violated.

5.4.3.1 Evaluation with BO

As evaluating each proposal needs to prune and retrain the model with multiple epochs, incurring large time cost, we use Bayesian optimization (BO) [30] to accelerate evaluation. The controller generates a *proposal pool* with K pruning proposals. We first use BO to select B ($B < K$) proposals with potentially better performance. Then the selected proposals are evaluated to derive the accurate detection and speed performance while the rest $(K - B)$ potentially weak proposals are not evaluated. Thus, we reduce the number of actual evaluated proposals.

To deal with the non-continuous and graph-like pruning proposals, we build a Gaussian process (GP) for BO with a Weisfeiler-Lehman (WL) graph kernel [142]. We select the propos-

[1]The compiler code generation is much faster than DNN training thanks to the auto-tuning capability, and can be performed in parallel with the pruning as measuring inference speed does not need accurate weight values.



m=0	0	1	2	3	4	m=1	5	6	7	8	9	10
Φ_0(A)	1	2	1	0	1	Φ_1(A)	1	1	1	1	0	0
Φ_0(B)	1	1	2	1	0	Φ_1(B)	0	1	1	0	1	1

Figure 5.4: WL kernel illustration. At initialization, there are two pruning proposals with features at $m = 0$. At step 1, WL kernel collects the next label of each node. At step 2, it re-labels the new nodes with neighbour information. At step 3, it obtains a new graph with features at $m = 1$. At step 4, WL kernel compares the histogram on both $m = 0$ and $m = 1$ features. The iteration repeats until $m = M$.

als according to their acquisition function values (we employ *Expected Improvement* [134] here). Algorithm 2 provides a summary.

We illustrate the WL graph kernel in Fig. 5.4. More specifically, the WL kernel compares two directed graphs in iterations. In the m-th WL iteration, it first obtains the graph feature vectors $\phi_m(g)$ and $\phi_m(g')$ for two graphs. Then it compares the two graphs with $k_{\text{base}}\big(\phi_m(g), \phi_m(g')\big)$ where we employ dot product as k_{base} here. The iterative procedure stops until $m = M$ and the resultant WL kernel is

$$k_{\text{WL}}^{M}(g, g') = \sum_{m=0}^{M} w_m k_{\text{base}}\big(\phi_m(g), \phi_m(g')\big). \tag{5.2}$$

where w_m contains the weights for each WL iteration m, which is set to equal for all m following [142].

After selecting B pruning proposals from the pool, we evaluate their performance using magnitude based framework [62] following their pruning proposals for each layer.

5.4.3.2 Gradients Guidance

To guide the proposal updating, we employ the derivatives of the GP predictive mean. Formally, the derivative with respect to the j-th element of $\phi^t = \phi(G_t)$ (the test graph feature vector) is also Gaussian with an expected value:

$$\mathbb{E}_{p(r|G_t, \mathcal{D}_{t-1})}\left[\frac{\partial r}{\partial [\phi^t]_j}\right] = \frac{\partial \mu}{\partial [\phi^t]_j} = \frac{\partial \langle \phi^t, \Phi^{1:t-1} \rangle}{\partial [\phi^t]_j} L_{1:t-1}^{-1} \mathbf{r}_{1:t-1} \tag{5.3}$$

where $\Phi_{1:t-1}$, $L_{1:t-1}$ and $\mathbf{r}_{1:t-1}$ are the stacked feature matrix, kernel values and rewards from previous observations.

Algorithm 2 Evaluation with BO

Input: Observation data $\mathcal{D}$, BO batch size B, BO acquisition function $\alpha(\cdot)$

Output: The best pruning proposal g

for steps **do**

Generate a pool of candidate pruning proposals $\mathcal{G}_c$;

Select $\{\hat{g}^i\}_{i=1}^B = \arg\max_{g \in \mathcal{G}_c} \alpha(g|\mathcal{D})$;

Evaluate the proposal and obtain reward $\{r^i\}_{i=1}^B$ of $\{\hat{g}^i\}_{i=1}^B$;

Obtain the gradients guidance information;

$\mathcal{D} \leftarrow \mathcal{D} \cup (\{\hat{g}^i\}_{i=1}^B, \{r^i\}_{i=1}^B)$;

Update GP of BO with $\mathcal{D}$;

end for

Following Eq. (3), we can obtain $\mathbb{E}\left[\frac{\partial r}{\partial [\phi_0^t]_j}\right]$ where ϕ_0^t is the list of the number of each node in the proposal. Positive gradients show that the node is beneficial to improve the reward and we should keep the node, while negative gradients mean that the node decreases the performance and it should be replaced. To make the gradients more illustrative, we transform the gradients into a probability distribution (replacement probability) using a sigmoid transformation on the negative of the gradients and then normalize them. Thus, negative gradients lead to high replacement probabilities.

To summarize, the evaluator provides the gradient guidance including the best evaluated pruning proposal and its corresponding replacement probability obtained from its gradients.

5.5 Performance Evaluation

5.5.1 Experiment Setup

For 2D object detection, we use a YOLOv4 [15] model as starting point and test on COCO dataset [91]. For 3D detection, we employ the PointPillars as starting point [83] and test on KITTI dataset [54]. We use 50 GPUs for parallel training and pruning search and it takes about 12 days to find the best pruning proposal in each experiment. In Eq. (1), we set α to 0.01 and the mobile inference time is measured in milliseconds. We set $w_m = 1$ in Eq. (2). The pool size K is set to 50 and the Bayesian batch size B is set to 10. All the acceleration results are tested on the mobile GPU (Qualcomm Adreno 640) of a Samsung Galaxy S20 smartphone.

Table 5.2: Comparison of various pruning methods on YOLOv4

Pruning Methods	Parameter #	Computation # (MACs)	mAP	Mobile GPU Speed (ms)
Original	64.36M	17.9G	56.5	285.7
Pattern [105]	16.09M	5.58G	47.6	114.9
Filter [64]	4.33M	1.87G	25.2	47.6
Unstructured [176]	4.33M	1.87G	49.9	98.6
Ours	4.33M	1.87G	49.3	55.2

5.5.2 Performance on 2D Object Detection

As shown in Table 5.2, we applied our proposed method to YOLOv4 and compared it with other pruning methods in terms of accuracy (mAP) and end-to-end inference latency on mobile GPU, with our compiler framework. Our method reduces the original computation from 17.9G MACs to 1.87G MACs and achieves 5.18× inference acceleration (285.7ms vs. 55.2ms). We also adopt the same pruning rate to YOLOv4 using other pruning schemes except pattern-based pruning, because there are 17% weights that come from 1×1 CONV layer cannot be pruned using pattern-based pruning and hence a lower overall pruning rate. Under the same pruning rate, our method is 16% slower than structured pruning on mobile GPU but achieves much higher accuracy (49.3 vs. 25.2 in mAP). With slightly lower accuracy, our method is 1.79× faster than unstructured pruning. And our method outperforms the pattern-based pruning in both accuracy and inference speed.

Fig. 5.5 shows the accuracy vs. latency comparison results of our method with other DNN inference frameworks and using other object detectors (on our compiler framework). The rectangular shapes show the comparison results of dense (unpruned) YOLOv4, as general sparsity is not supported in other frameworks. By leveraging our compiler optimizations, our method outperforms other representative DNN inference frameworks, including TensorFlow-Lite, TVM, and MNN. All the red star shapes represent our pruned models with different latency constraints and using YOLOv4 as the starting point. And our method achieves near Pareto-optimality considering both accuracy and latency. The results clearly show the effectiveness of our proposed method.

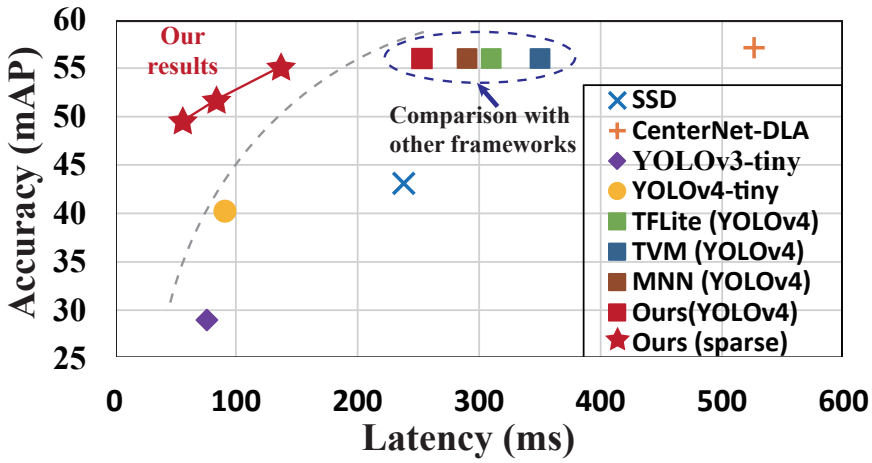


Figure 5.5: mAP vs. latency for various object detection approaches with different acceleration frameworks.

5.5.3 Performance on 3D Object Detection

We show the performance of the original unpruned PointPillars model and the model derived by our method with different grid sizes (0.16m and 0.24m, where large grid size means small input size for the model) in Tab. 5.3. The real-time requirements are set to 200ms for 0.16m gird size and 100ms for 0.24m. We can observe that increasing grid size can reduce the pseudo-image input size, leading to smaller parameter and computation numbers, and faster inference speed on mobile GPUs. For the same grid size, our method can significantly reduce the parameter count and computation, thus satisfying the real-time requirement, while achieving state-of-the-art detection performance.

For a grid size of 0.24m, the pruning ratio of other pruning schemes are set to the same with the overall pruning ratio of our pruned model (86%). As observed, the proposed method can achieve the best detection performance compared with other methods where all the layers share the same pruning scheme, demonstrating the advantages of using flexible pruning scheme for each layer. Besides, with compiler optimization, filter pruning is the fastest but suffers from obvious detection performance degradation.

The proposed method can process one LiDAR image within 97ms with the highest precision, demonstrating the superior performance of the proposed method to achieve (close-to) real-time inference on mobile with state-of-the-art detection performance. Although other DNN inference frameworks exist such as Tensorflow-Lite [2], TVM [29] and MNN [1], we are the **first** to support 3D object detection on mobile device.

Table 5.3: Comparison of various pruning methods for PointPillars

Methods (grid size)	Para. #	Comp. # (MACs)	Speed (ms)	Car 3D detection		
				Easy	Moderate	Hard
PointPillars (0.16)	5.8M	60G	542	84.99	74.11	69.53
Ours (0.16)	1.1M	10.7G	189	**85.50**	**76.58**	**70.23**
PointPillars (0.24)	5.8M	28G	257	84.05	74.99	68.30
Filter [64] (0.24)	0.8M	4.0G	81	81.54	68.10	65.90
Pattern [105] (0.24)	0.8M	3.9G	111	80.97	73.77	68.05
Ours (0.24)	0.8M	3.9G	97	**85.20**	**75.57**	**68.37**

Chapter 6

Conclusion

There are two main issues which potentially limit the wide application of DNNs: 1) the robustness, and 2) the large computation & storage requirements. To investigate the DNN robustness, we explore the DNN attack, robustness evaluation and defense. More specifically, for DNN attack, we investigate ADMM attack, ZO-ADMM attack, ZO-NGD attack and fault sneaking attack to achieve various attack goals with different algorithms under various conditions. For robustness evaluation, we propose a fast evaluation method to obtain the model perturbation bound such that any model perturbation within the bound does not alter the model classification outputs or incur model mis-behaviors. For the DNN defense, we investigate the defense performance with model connection techniques and successfully mitigate the fault sneaking and backdoor attacks.

With a deeper understanding of the DNN robustness, we further explore the deployment problem of DNN models on edge devices with limited resources. To satisfy the storage and computation limitation on edge devices, we adopt model pruning to remove the redundancy in models, thus reducing the storage and computation during inference. Specifically, we propose a compiler-aware neural pruning search framework to achieve high-speed inference. The framework automatically searches the pruning scheme and rate for each layer to find a best-suited pruning for optimizing detection accuracy and speed performance under compiler optimization. Our experiments demonstrate that for the first time, the proposed method achieves (close-to) real-time, 55ms and 97ms inference times for YOLOv4 based 2D object detection and PointPillars based 3D detection, respectively, on an off-the-shelf mobile phone with minor (or no) accuracy loss.

To summary, we mainly investigate the DNN robustness and implement real-time DNN inference on the mobile.

Bibliography

[1] https://github.com/alibaba/MNN.

[2] https://www.tensorflow.org/mobile/tflite/.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, and et. al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2016. Software available from tensorflow.org.

[4] Moustafa Alzantot, Bharathan Balaji, and Mani B. Srivastava. Did you hear that? adversarial examples against automatic speech recognition. *CoRR*, abs/1801.00554, 2018.

[5] Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*, volume 191. American Mathematical Soc., 2007.

[6] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[7] A. Barenghi, L. Breveglieri, and et. al. Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures. *Proceedings of the IEEE*, 2012.

[8] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[9] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Exploring the space of black-box attacks on deep neural networks. *CoRR*, abs/1712.09491, 2017.

[10] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402, 2013.

[11] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. *IEEE TKDE*, 26(4):984–996, 2014.

[12] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the ICLM 2012*, pages 1467–1474, 2012.

[13] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the International Conference on International Conference on Machine Learning*, pages 1467–1474, 2012.

[14] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[15] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv:2004.10934*, 2020.

[16] Ilija Bogunovic, Jonathan Scarlett, Andreas Krause, and Volkan Cevher. Truncated variance reduction: A unified approach to bayesian optimization and level-set estimation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1507–1515. Curran Associates, Inc., 2016.

[17] Akhilan Boopathy, Tsui-Wei Weng, Pin-Yu Chen, Sijia Liu, and Luca Daniel. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *AAAI*, Jan 2019.

[18] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[19] Jakub Breier, Xiaolu Hou, and et. al. Practical fault attack on deep neural networks. In *Proceedings of the ACM SIGSAC*, CCS '18, pages 2204–2206. ACM, 2018.

[20] W. Brendel, J. Rauber, and M. Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[21] Sébastien Bubeck, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. *International Coference on International Conference on Machine Learning*, 2019.

[22] S. Rota Bulò, B. Biggio, I. Pillai, M. Pelillo, and F. Roli. Randomized prediction games for adversarial machine learning. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2466–2478, Nov 2017.

[23] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *NIPS*, pages 4790–4799, 2018.

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

[25] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7, 2018.

[26] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[27] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. *arXiv preprint arXiv:1709.04114*, 2017.

[28] 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.

[29] Tianqi Chen, Thierry Moreau, et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *USENIX*, pages 578–594, 2018.

[30] Yutian Chen, Aja Huang, et al. Bayesian optimization in alphago. *arXiv:1812.06855*, 2018.

[31] Nicholas Cheney, Martin Schrimpf, and Gabriel Kreiman. On the robustness of convolutional neural networks to internal architecture and weight perturbations. *arXiv preprint arXiv:1703.08245*, 2017.

[32] 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.

[33] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[34] David Roxbee Cox and David Victor Hinkley. *Theoretical statistics*. Chapman and Hall/CRC, 1979.

[35] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2018.

[36] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[37] Elvis Dohmatob. Limitations of adversarial robustness: strong no free lunch theorem. *International Conference on International Conference on Machine Learning*, 2018.

[38] Peiyan Dong, Siyue Wang, et al. Rtmobile: Beyond real-time mobile acceleration of rnns for speech recognition. *arXiv:2002.11474*, 2020.

[39] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *NeurIPS*, pages 759–770, 2019.

[40] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially no barriers in neural network energy landscape. In *International Conference on Machine Learning*, volume 80, pages 1309–1318, Jul 2018.

[41] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, May 2015.

[42] John C Duchi, Peter L Bartlett, and Martin J Wainwright. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2):674–701, 2012.

[43] John C Duchi, Michael I Jordan, Martin J Wainwright, and Andre Wibisono. Optimal rates for zero-order convex optimization: The power of two function evaluations. *IEEE Transactions on Information Theory*, 61(5):2788–2806, 2015.

[44] Souradeep Dutta, Susmit Jha, Sriram Sanakaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130*, 2017.

[45] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, et al. A dual approach to scalable verification of deep networks. *UAI*, 2018.

[46] Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 269–286. Springer, 2017.

[47] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on machine learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[48] Rida T. Farouki. The bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379 – 419, 2012.

[49] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. The robustness of deep networks: A geometrical perspective. *IEEE Signal Processing Magazine*, 34(6):50–62, 2017.

[50] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, Pascal Frossard, and Stefano Soatto. Empirical study of the topology and geometry of deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3762–3770, 2018.

[51] X. Gao and S.-Z. Zhang. First-order algorithms for convex optimization with nonseparable objective and coupled constraints. *Journal of the ORSC*, 2017.

[52] Xiang Gao, Bo Jiang, and Shuzhong Zhang. On the information-adaptive variants of the admm: an iteration complexity perspective. *Journal of Scientific Computing*, 76(1):327–363, 2018.

[53] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of DNNs. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.

[54] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *CVPR*, 2012.

[55] Jonas Gomes, Luiz Velho, and Mario Costa Sousa. *Computer Graphics: Theory and Practice*. A. K. Peters, Ltd., Natick, MA, USA, 1st edition, 2012.

[56] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[57] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *stat*, 1050:20, 2015.

[58] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. Using mode connectivity for loss landscape analysis. *arXiv preprint arXiv:1806.06977*, 2018.

[59] Roger Grosse and Ruslan Salakhudinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2304–2313, Lille, France, 07–09 Jul 2015. PMLR.

[60] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. BadNets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[61] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NeurIPS*, pages 1379–1387, 2016.

[62] Song Han, Jeff Pool, et al. Learning both weights and connections for efficient neural network. In *NeurIPS*, pages 1135–1143, 2015.

[63] Kaiminaripov He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[64] Yang He, Ping Liu, et al. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *CVPR*, 2019.

[65] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, pages 1389–1397, 2017.

[66] Thomas A Hogan and Bhavya Kailkhura. Universal hard-label black-box perturbations: Breaking security-through-obscurity defenses. *arXiv preprint arXiv:1811.03733*, 2018.

[67] Mingyi Hong and Zhi-Quan Luo. On the linear convergence of the alternating direction method of multipliers. *Mathematical Programming*, 162(1):165–199, Mar 2017.

[68] Weiwei Hu and Ying Tan. Black-box attacks against RNN based malware detection algorithms. *CoRR*, abs/1705.08131, 2017.

[69] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NeurIPS*, 2016.

[70] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, July 2018.

[71] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. *ICLR*, 2019.

[72] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 19–35, 2018.

[73] Yeongjin Jang, Jaehyuk Lee, and et. al. Sgx-bomb: Locking down the processor via rowhammer attack. In *Proceedings of the 2nd Workshop on SysTEX*, page 5. ACM, 2017.

[74] Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal statistics of fisher information in deep neural networks: Mean field approach. *arXiv preprint arXiv:1806.01316*, 2018.

[75] Guy Katz, Clark Barrett, David L Dill, et al. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[76] Y. Kim, R. Daly, and et. al. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *ACM/IEEE ISCA 2014*, 2014.

[77] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *2015 ICLR*, arXiv preprint arXiv:1412.6980, 2015.

[78] J Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *2018 ICML*, arXiv preprint arXiv:1711.00851, 2018.

[79] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[80] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[81] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[82] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *2017 ICLR*, arXiv preprint arXiv:1611.01236, 2017.

[83] Alex H Lang, Sourabh Vora, et al. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, pages 12697–12705, 2019.

[84] Pavel Laskov et al. Practical evasion of a learning-based classifier: A case study. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 197–211. IEEE, 2014.

[85] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[86] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[87] Cong Leng, Zesheng Dou, Hao Li, Shenghuo Zhu, and Rong Jin. Extremely low bit neural network: Squeeze the last bit out with admm. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[88] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *International Conference on Learning Representations*, 2017.

[89] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[90] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *ICLR*, 2019.

[91] Tsung-Yi Lin et al. Microsoft coco: Common objects in context. In *ECCV*.

[92] Yen-Chen Lin, Zhang-Wei Hong, and et. al. Tactics of adversarial attack on deep reinforcement learning agents. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3756–3762. AAAI Press, 2017.

[93] Changliu Liu, Tomer Arnon, Christopher Lazarus, Clark Barrett, and Mykel J Kochenderfer. Algorithms for verifying deep neural networks. *arXiv preprint arXiv:1903.06758*, 2019.

[94] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[95] Ning Liu, Xiaolong Ma, et al. Autocompress: An automatic dnn structured pruning framework for ultra-high compression rates. In *AAAI*, 2020.

[96] Q. Liu, X. Shen, and Y. Gu. Linearized admm for non-convex non-smooth optimization with convergence analysis. *arXiv preprint arXiv:1705.02502*, 2017.

[97] S. Liu, J. Chen, P.-Y. Chen, and A. O. Hero. Zeroth-order online admm: Convergence analysis and applications. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84, pages 288–297, April 2018.

[98] Sijia Liu, Pin-Yu Chen, Xiangyi Chen, and Mingyi Hong. SignSGD via zeroth-order oracle. *International Conference on Learning Representations*, 2019.

[99] Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. Towards robust neural networks via random self-ensemble. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 369–385, 2018.

[100] Y. Liu, L. Wei, B. Luo, and Q. Xu. Fault injection attack on deep neural network. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 131–138, Nov 2017.

[101] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *CoRR*, abs/1611.02770, 2016.

[102] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[103] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[104] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, pages 5058–5066, 2017.

[105] Xiaolong Ma et al. Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices. In *AAAI*, 2020.

[106] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *International Conference on Learning Representations*, 2018.

[107] Huizi Mao, Song Han, et al. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv:1705.08922*, 2017.

[108] James Martens. New perspectives on the natural gradient method. *CoRR*, abs/1412.1193, 2014.

[109] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pages 2408–2417, 2015.

[110] James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade*, pages 479–535. Springer, 2012.

[111] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[112] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Jonathan Uesato, and Pascal Frossard. Robustness via curvature regularization, and vice versa. *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[113] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. *CoRR*, abs/1612.06299, 2016.

[114] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 2(17):527–566, 2015.

[115] Yurii Nesterov and Vladimir Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

[116] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[117] Wei Niu et al. Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning. *arXiv:2001.00138*, 2020.

[118] Yann Ollivier. Riemannian metrics for neural networks i: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153, 2015.

[119] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *CVPR 2019*, 2018.

[120] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016.

[121] 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.

[122] 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.

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

[124] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. Crafting adversarial input sequences for recurrent neural networks. In *IEEE Military Communications Conference (MILCOM)*, pages 49–54, 2016.

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

[126] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/1605.07277, 2016.

[127] Neal Parikh, Stephen Boyd, et al. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[128] Eunhyeok Park, Junwhan Ahn, and Sungjoo Yoo. Weighted-entropy-based quantization for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5456–5464, 2017.

[129] Razvan Pascanu and Yoshua Bengio. Natural gradient revisited. *CoRR*, abs/1301.3584, 2013.

[130] Razvan Pascanu and Yoshua Bengio. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013.

[131] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch, 2017.

[132] Adam Paszke, Sam Gross, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.

[133] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999.

[134] Chao Qin, Diego Klabjan, and Daniel Russo. Improving the expected improvement algorithm. In *NeurIPS*, pages 5381–5391, 2017.

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

[136] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, pages 525–542. Springer, 2016.

[137] Ao Ren, Tianyun Zhang, Shaokai Ye, et al. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *ASPLOS*, pages 925–938. ACM, 2019.

[138] Bodo Selmke, Stefan Brummer, and et. al. Precise laser fault injections into 90 nm and 45 nm sram-cells. In *International Conference on Smart Card Research and Advanced Applications*, pages 193–205. Springer, 2015.

[139] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*, pages 6103–6113, 2018.

[140] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Adversarial generative nets: Neural network attacks on state-of-the-art face recognition. *CoRR*, abs/1801.00349, 2018.

[141] Tao Sheng, Chen Feng, Shaojie Zhuo, et al. A quantization-friendly separable convolution for mobilenets. In *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications*, pages 14–18. IEEE, 2018.

[142] Nino Shervashidze, Pascal Schweitzer, et al. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77), 2011.

[143] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[144] Dawn Song, Kevin Eykholt, Ivan Evtimov, et al. Physical adversarial examples for object detectors. In *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[145] 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 *European Conference on Computer Vision*, pages 631–648, 2018.

[146] T. Suzuki. Dual averaging and proximal gradient descent for online alternating direction multiplier method. In *International Conference on Machine Learning*, pages 392–400, 2013.

[147] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[148] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *ICLR*, 2013.

[149] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, et al. Intriguing properties of neural networks. *2014 ICLR*, 2014.

[150] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. Ensemble adversarial training: Attacks and defenses. *2018 ICLR*, arXiv preprint arXiv:1705.07204, 2018.

[151] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*, pages 8000–8010, 2018.

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

[153] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. *arXiv preprint arXiv:1805.11770*, 2018.

[154] Victor Van Der Veen, Yanick Fratantonio, and et. al. Drammer: Deterministic rowhammer attacks on mobile platforms. In *ACM SIGSAC conference on computer and communications security*, pages 1675–1689. ACM, 2016.

[155] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *IEEE Symposium on Security and Privacy*, 2019.

[156] Huan Wang, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. Identifying generalization properties in neural networks. *arXiv preprint arXiv:1809.07402*, 2018.

[157] S. Wang, X. Wang, S. Ye, P. Zhao, and X. Lin. Defending dnn adversarial attacks with pruning and logits augmentation. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1144–1148, 2018.

[158] S. Wang, X. Wang, P. Zhao, W. Wen, D. Kaeli, P. Chin, and X. Lin. Defensive Dropout for Hardening Deep Neural Networks under Adversarial Attacks. *ArXiv e-prints*, September 2018.

[159] Timothy E Wang, Jack Gu, Dhagash Mehta, Xiaojun Zhao, and Edgar A Bernal. Towards robust deep neural networks. *arXiv preprint arXiv:1810.11726*, 2018.

[160] Wei Wen, Chunpeng Wu, et al. Learning structured sparsity in deep neural networks. In *NeurIPS*, pages 2074–2082, 2016.

[161] Tsui-Wei Weng, Huan Zhang, Hongge Chen, et al. Towards fast computation of certified robustness for relu networks. *ICML*, 2018.

[162] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, et al. Evaluating the robustness of neural networks: An extreme value theory approach. *ICLR*, 2018.

[163] Tsui-Wei Weng, Pu Zhao, Sijia Liu, Pin-Yu Chen, Xue Lin, and Luca Daniel. Towards certificated model robustness against weight perturbations. In *AAAI 2020*, 2020.

[164] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *arXiv preprint arXiv:1711.00851*, 2017.

[165] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. *arXiv preprint arXiv:1805.12514*, 2018.

[166] Jiaxiang Wu, Cong Leng, Yuhang Wang, et al. Quantized convolutional neural networks for mobile devices. In *CVPR*, pages 4820–4828, 2016.

[167] Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. In *International Conference on Machine Learning*, 2017.

[168] Huang Xiao, Battista Biggio, and et.al. Is feature selection secure against training data poisoning? In *ICML*, pages 1689–1698, 2015.

[169] Yuan Xiao, Xiaokuan Zhang, and et. al. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *USENIX Security Symposium*, pages 19–35, 2016.

[170] C. Xie, J. Wang, Z. Zhang, Y. Zhou, L. Xie, and A. Yuille. Adversarial examples for semantic segmentation and object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1378–1387, Oct. 2018.

[171] 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. In *International Conference on Learning Representations*, 2019.

[172] Fuxun Yu, Chenchen Liu, Yanzhi Wang, and Xiang Chen. Interpreting adversarial robustness: A view from decision surface in input space. *arXiv preprint arXiv:1810.00144*, 2018.

[173] Ruichi Yu, Ang Li, et al. Nisp: Pruning networks using neuron importance score propagation. In *CVPR*, pages 9194–9203, 2018.

[174] H. Zhang, T.-W. Weng, and et. al. Efficient Neural Network Robustness Certification with General Activation Functions. In *NIPS 2018*.

[175] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *NIPS*, dec 2018.

[176] Tianyun Zhang, Shaokai Ye, et al. Systematic weight pruning of dnns using alternating direction method of multipliers. *ECCV*, 2018.

[177] P. Zhao, K. Xu, T. Zhang, M. Fardad, Y. Wang, and X. Lin. Reinforced adversarial attacks on deep neural networks using admm. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1169–1173, Nov 2018.

[178] Pu Zhao, Pin-Yu Chen, Payel Das, Karthikeyan Natesan Ramamurthy, and Xue Lin. Bridging mode connectivity in loss landscapes and adversarial robustness. *ICLR*, 2020.

[179] Pu Zhao, Pin-Yu Chen, Siyue Wang, and Xue Lin. Towards query-efficient black-box adversary with zeroth-order natural gradient descent. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6909–6916, 2020.

[180] Pu Zhao, Pin-Yu Chen, Siyue Wang, and Xue Lin. Towards query-efficient black-box adversary with zeroth-order natural gradient descent. In *AAAI 2020*, 2020.

[181] Pu Zhao, Sijia Liu, Pin-Yu Chen, Nghia Hoang, Kaidi Xu, Bhavya Kailkhura, and Xue Lin. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 121–130, 2019.

[182] Pu Zhao, Sijia Liu, Yanzhi Wang, and Xue Lin. An admm-based universal framework for adversarial attacks on deep neural networks. *arXiv preprint arXiv:1804.03193*, 2018.

[183] Pu Zhao, Sijia Liu, Yanzhi Wang, and Xue Lin. An admm-based universal framework for adversarial attacks on deep neural networks. *CoRR*, abs/1804.03193, 2018.

[184] Pu Zhao, Siyue Wang, Cheng Gongye, et al. Fault sneaking attack: a stealthy framework for misleading deep neural networks. In *Proceedings of the 56th Annual Design Automation Conference*, 2019.

[185] Pu Zhao, Siyue Wang, Cheng Gongye, Yanzhi Wang, Yunsi Fei, and Xue Lin. Fault sneaking attack: a stealthy framework for misleadingdeep neural networks. *DAC*, 2019.

[186] Pu Zhao, Kaidi Xu, Sijia Liu, Yanzhi Wang, and Xue Lin. Admm attack: An enhanced adversarial attack for deep neural networks with undetectable distortions. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ASPDAC '19, pages 499–505, New York, NY, USA, 2019. ACM.

[187] Pu Zhao, Geng Yuan, Yuxuan Cai, Wei Niu, Qi Liu, Wujie Wen, Bin Ren, Yanzhi Wang, and Xue Lin. Neural pruning search for real-time objectdetection of autonomous vehicles. In *DAC 2021*, 2021.

[188] Aojun Zhou, Anbang Yao, Yiwen Guo, et al. Incremental network quantization: Towards lossless cnns with low-precision weights. *2017 ICLR*, 2017.

[189] Zhuangwei Zhuang, Mingkui Tan, et al. Discrimination-aware channel pruning for deep neural networks. In *NeurIPS*, pages 875–886, 2018.

[190] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

[191] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *SIGKDD*, pages 2847–2856, 2018.