Mathematical Optimization Algorithms for Model Compression and Adversarial Learning in Deep Neural Networks

Tianyun Zhang
Syracuse University
ABSTRACT

Large-scale deep neural networks (DNNs) have made breakthroughs in a variety of tasks, such as image recognition, speech recognition and self-driving cars. However, their large model size and computational requirements add a significant burden to state-of-the-art computing systems. Weight pruning is an effective approach to reduce the model size and computational requirements of DNNs. However, prior works in this area are mainly heuristic methods. As a result, the performance of a DNN cannot maintain for a high weight pruning ratio. To mitigate this limitation, we propose a systematic weight pruning framework for DNNs based on mathematical optimization. We first formulate the weight pruning for DNNs as a non-convex optimization problem, and then systematically solve it using alternating direction method of multipliers (ADMM). Our work achieves a higher weight pruning ratio on DNNs without accuracy loss and a higher acceleration on the inference of DNNs on CPU and GPU platforms compared with prior works.

Besides the issue of model size, DNNs are also sensitive to adversarial attacks, a small invisible noise on the input data can fully mislead a DNN. Research on the robustness of DNNs follows two directions in general. The first is to enhance the robustness of DNNs, which increases the degree of difficulty for adversarial attacks to fool DNNs. The second is to design adversarial attack methods to test the robustness of DNNs. These two aspects reciprocally benefit each other towards hardening DNNs. In our work, we propose to generate adversarial attacks with low distortion via convex optimization, which achieves 100% attack success rate with lower distortion compared with prior works. We also propose a unified min-max optimization framework for the adversarial attack and defense on DNNs over multiple domains. Our proposed method performs better compared with the prior works, which use average-based strategies to solve the problems over multiple domains.
To my family:

past, present, and future
MATHEMATICAL OPTIMIZATION ALGORITHMS FOR
MODEL COMPRESSION AND ADVERSARIAL LEARNING
IN DEEP NEURAL NETWORKS

By

Tianyun Zhang
B.E., Huazhong University of Science and Technology, 2015

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University
July 2021
ACKNOWLEDGMENTS

First and foremost, I would like to take this opportunity to thank my advisor Prof. Makan Fardad for his invaluable guidance and support throughout my doctoral studies. Makan is always patient and helpful. He continues to give me thoughtful advice on my study and life. In the past five years, I have grown from a beginner of mathematical optimization and deep learning to a researcher with many publications in these fields. I could not have made such progress without his patient and responsible help. I am also very grateful to Prof. Yanzhi Wang. He is a researcher with many intellectual ideas who has provided me with much helpful guidance on my research in the field of deep learning.

I would like to extend my sincere thanks to the faculty, staff, and students at Syracuse University, they have supported me through these years. I could not have experienced the wonderful and enjoyable journey at Syracuse University without their help. I would like to thank my collaborators. I could not have achieved my accomplishments without their support. I would also like to extend my appreciation to my defense committee members Prof. Qinru Qiu, Prof. Reza Zafarani, Prof. Mustafa Cenk Gursoy, Prof. Lixin Shen, and Prof. Yanzhi Wang for their valuable suggestions.

Last but not least, I am truly and deeply grateful to my parents for their unconditional love and support. The duration of my PhD studies was long, and sometimes I felt lonely with feelings of self-doubt. The encouragement and care from my parents always aided me in snapping out of my bad mood. I rarely say thanks to my parents in daily life, here I want to offer my most heartfelt thanks to them.
# Table of Contents

**Acknowledgments**  
[5]

**List of Tables**  
[x]

**List of Figures**  
[xii]

## 1 Introduction  
1.1 Background  .................................................. 1  
1.2 Dissertation organization and chapter summaries  .................. 3  
1.3 Bibliographic note  .............................................. 5

## 2 A Unified DNN Weight Pruning Framework Using ADMM  
2.1 Introduction  .................................................. 7  
2.2 Related work  .................................................. 9  
2.3 Problem statement  .............................................. 11  
2.4 A unified DNN weight pruning framework using ADMM .............. 13  
  2.4.1 Solutions of different types of structured sparsity and discussions 15  
  2.4.2 The masked retraining step  .................................. 17  
  2.4.3 Overall illustration of our proposed framework  ................. 18  
2.5 Methods to improve pruning rate  ................................ 20  
  2.5.1 Progressive DNN weight pruning  ........................... 20  
  2.5.2 Network purification and unused path removal  ............... 22  
2.6 Numerical results  .............................................. 23
3 A Unified DNN Weight Pruning Framework Using Reweighted Optimization Methods

3.1 Introduction

3.2 Related work

3.2.1 Static regularization-based pruning

3.2.2 Dynamic regularization-based pruning

3.3 Problem statement

3.3.1 Non-structured pruning

3.3.2 Structured pruning

3.4 A unified algorithm for non-structured and structured sparsity

3.5 Numerical results

3.5.1 Non-structured pruning on LeNet-5 & AlexNet

3.5.2 Structured pruning on ResNet-18, ResNet-50 and MobileNet-V2-1.0

3.6 Discussion on inference acceleration

3.7 Summary

4 Generation of Low Distortion Adversarial Attacks via Convex Programming

4.1 Introduction

4.2 Related work

4.2.1 Gradient descent based attack methods

4.2.2 Related work on convex programming and mixed integer linear programming

4.2.3 Representative defense method

4.3 Problem statement

4.4 Problem relaxation and proposed algorithms
### 4.5 Numerical results

- **4.5.1 Experiment setup**
- **4.5.2 Attack success rate and distortion for the $\ell_2$ attack**
- **4.5.3 Attack success rate and distortion for the $\ell_\infty$ attack**
- **4.5.4 Attack against adversarial training**

### 4.6 Summary

### 5 A Unified Min-Max Framework for Adversarial Exploration and Robustness

- **5.1 Introduction**
- **5.2 Related work**
- **5.3 Preliminaries**
- **5.4 Min-max power in attack design**
  - **5.4.1 A unified framework for robust adversarial attacks**
  - **5.4.2 Min-max algorithm for adversarial attack generation**
- **5.5 Min-max power in defense**
  - **5.5.1 A unified framework for adversarial training under mixed types of adversarial attacks**
  - **5.5.2 Improved robustness via diversified $\ell_p$ attacks**
  - **5.5.3 Min-max algorithm for generalized adversarial training**
- **5.6 Numerical results**
  - **5.6.1 Min-max adversarial attacks**
  - **5.6.2 Robust training under multiple types of $\ell_p$ attacks**
- **5.7 Summary**

### 6 On the Optimal Interdiction of Transportation Networks

- **6.1 Introduction**
- **6.2 Dynamic model of transportation networks & informal statement of optimal interdiction problem**
**List of Tables**

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Comparison of our method and SSL method on column sparsity <strong>without accuracy loss</strong> on AlexNet/CaffeNet model for ImageNet data set</td>
<td>23</td>
</tr>
<tr>
<td>2.2</td>
<td>Comparison of our method and SSL method on column and row sparsity <strong>with less than 2% accuracy loss</strong> on AlexNet/CaffeNet for ImageNet data set</td>
<td>24</td>
</tr>
<tr>
<td>2.3</td>
<td>Structured pruning results on ResNet-18 model for ImageNet data set</td>
<td>25</td>
</tr>
<tr>
<td>2.4</td>
<td>Structured pruning results on ResNet-50 model for ImageNet data set</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>Comparisons of weight pruning results on AlexNet model for ImageNet data set</td>
<td>27</td>
</tr>
<tr>
<td>2.6</td>
<td>Comparisons of weight pruning results on VGG-16 model for ImageNet data set</td>
<td>27</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparisons of overall weight pruning results on LeNet-5 using MNIST data set</td>
<td>39</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparisons of overall non-structured weight pruning results on AlexNet model for ImageNet dataset</td>
<td>39</td>
</tr>
<tr>
<td>3.3</td>
<td>Structured pruning results on ResNet-18 for ImageNet dataset</td>
<td>40</td>
</tr>
<tr>
<td>3.4</td>
<td>Structured pruning results on ResNet-50 for ImageNet dataset</td>
<td>40</td>
</tr>
<tr>
<td>3.5</td>
<td>Structured pruning results on MobileNet-V2-1.0 for CIFAR-10 dataset</td>
<td>40</td>
</tr>
<tr>
<td>4.1</td>
<td>Comparisons of different $\ell_2$ attacks for MNIST and CIFAR-10 data sets</td>
<td>53</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of different $\ell_\infty$ attacks for MNIST and CIFAR-10 data sets</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparison of average and min-max (APGD) ensemble attack over four models on MNIST. Acc (%) represents the test accuracy of classifiers on adversarial examples</td>
<td>68</td>
</tr>
</tbody>
</table>
5.3 Adversarial training on single attacks ($\ell_\infty$ and $\ell_2$) and multiple attacks (avg. and min max). The perturbation magnitude $\epsilon$ for $\ell_\infty$ and $\ell_2$ attacks are 0.2 and 2.0, respectively. (a) & (b) on MNIST; (c) & (d) on CIFAR-10.

6.1 Comparison of different numerical algorithms for network in Figure 6.1 . . . . . . 83

6.2 Comparison of different numerical algorithms for network in Figure 6.2 . . . . . . 84
LIST OF FIGURES

2.1 Illustration of filter-wise, channel-wise and shape-wise structured sparsity from left to right. ............................................. 15

2.2 Illustration of 2D weight matrix for GEMM (left) and row-wise and column-wise sparsity (right). ............................................. 18

2.3 Weight distribution of every (convolutional or fully connected) layer on LeNet-5. The subfigures in the left column are the weight distributions of the pretrained DNN model (serving as our starting point); the subfigures of the middle column are the weight distributions after the ADMM procedure; the subfigures of the right column are the weight distributions after our final retraining step. Note that the subfigures in the last column include a small number of nonzero weights that are not clearly visible due to the large number of zero weights. .................... 19

2.4 An illustration our proposed progressive DNN weight pruning method on Struct-tADMM framework. ............................................. 22

2.5 Convergence behavior of the ADMM regularization procedure for the convolutional layers 2-5 of AlexNet. .............................. 26

3.1 Comparisons of the distribution of all weights and critical weights (remaining weights after pruning) in the second fully connected layer (FC-2) of a pretrained AlexNet model and $\ell_1$ regularized model. ............................................. 33

3.2 Weight distribution of all layers in AlexNet after our reweighted $\ell_1$ regularization method ............................................. 33
5.1 Ensemble attack against four DNN models on MNIST. (a) & (b): Attack success rate of adversarial examples generated by average (ensemble PGD) or min-max (APGD) attack method. (c): Boxplot of weight $w$ in APGD adversarial loss. Here we adopt the same $\ell_\infty$-attack as Table [5.1].

5.2 Generalized AT of MLP models on MNIST. (a): Violin plot of weight $w$ in APGD versus perturbation magnitude $\epsilon$ of $\ell_2$-attack in AT; (b): Model robustness under different AT schemes; (c): Model robustness and diversity regularization in (5.15) vs. adversarial training epoch.

6.1 Network from [1].

6.2 Network from [1] based on [2, Chap. 19].
CHAPTER 1

INTRODUCTION

1.1 Background

Large-scale deep neural networks or DNNs have made breakthroughs in many fields, such as image recognition [3-5], speech recognition [6,7], game playing [8], and driver-less cars [9]. Despite the huge success, their large model size and computational requirements will add significant burden to state-of-the-art computing systems [3,10,11], especially for embedded and IoT systems. As a result, a number of prior works are dedicated to model compression in order to simultaneously reduce the computation and model storage requirements of DNNs, with minor effect on the overall accuracy. These model compression techniques include weight pruning [11-15], weight clustering [11,16], and low rank approximation [17,18], etc.

Most of the existing methods on weight pruning are heuristic. For example, some of them prune the weights based on the magnitude of their value or gradient. However, the importance of weights can not be determined by their magnitude. Which results in the fact that the heuristic methods cannot achieve high pruning ratio on DNN without accuracy loss. To mitigate these shortcomings, we present a systematic framework of weight pruning based on optimization algorithms. First, we formulate weight pruning as a non-convex optimization problem. By using ADMM, the original non-convex optimization problem is decomposed into two subproblems that are solved iteratively.
In the weight pruning problem, one of these subproblems can be solved using stochastic gradient descent, while the other can be solved analytically. Upon convergence of alternating direction method of multipliers (ADMM), we remove the weights which are (close to) zero and retrain the network. Our extensive numerical experiments indicate that ADMM works very well in practice and is highly suitable for weight pruning. The weight pruning results consistently outperform prior works.

Recent research also demonstrates that DNNs are vulnerable to adversarial attacks, a small invisible change on an image can change how it is classified. Research on the robustness of DNNs follows two directions in general. The first is to enhance the robustness of DNNs, which increases the degree of difficulty for adversarial attacks to fool DNNs [19–22]. The second is to design adversarial attack methods to test the robustness of DNNs [23–27]. These two aspects reciprocally benefit each other towards hardening DNNs.

Most adversarial attack generation problems in the literature are solved by first-order gradient descent method. In contrast to these methods, we first formulate adversarial attack on deep neural networks as a non-convex optimization problem. Then we relax it to a convex problem by proposing a linear approximation on the activation function. We also designed an algorithm which iteratively solve the convex problem and prove that upon convergence the solution is feasible for the original non-convex problem. In experiments, we achieve 100% success rate on neural network models with lower distortion compared with prior works.

Many problem setups in adversarial attacks and defenses need the optimization in multiple domains, such as attacking model ensembles [28,29], devising universal perturbation to input samples [30] and generalized AT over multiple types of threat models [31,32]. However, current methods for solving these tasks often rely on simple heuristics (e.g., uniform averaging). We propose a unified min-max optimization framework on solving these problems, and achieve significant performance improvements compared with current methods.

In transportation networks, mass enters through source cells and, after being routed through the network, are removed at sink cells. The flow of mass is subject to (i) conservation of mass
constraints, and (ii) link capacity constraints. In our work, we study interdiction or attack on transportation networks, which for the sake of concreteness we consider to be highway traffic networks. Our work follows [1] in formulating the optimal interdiction problem as a min-max optimization problem and subsequently employing duality to transform it to a standard bilinear optimization problem. We prove that even without an explicit promotion of sparsity in the formulation, the solution to the optimal interdiction problem is both sparse and binary. Our numerical experiments demonstrate that our approach performs better in comparison with methods reported in earlier work [8].

1.2 Dissertation organization and chapter summaries

The remainder of the dissertation is organized as follows. In Chapter 2, we study a unified DNN weight pruning framework using ADMM. In Chapter 3, a DNN weight pruning method using reweighted optimization methods is discussed. In Chapter 4, we discuss an adversarial attack generation method via convex programming. In Chapter 5, we propose a unified min-max optimization framework for adversarial attack and defense over multiple domains. In Chapter 6, we study an optimal interdiction method of transportation networks. In Chapter 7, we conclude this thesis and give directions for future research.

In Chapter 2, we first formulate weight pruning problem on DNNs as an optimization problem with hard constraints. Then we define indicator functions to integrate the hard constraints into the objective function. Since the indicator functions are non-differentiable, the problem cannot be solved directly by SGD [33] or ADAM [34]. To deal with this issue, we propose to decompose the problem into two subproblems using ADMM. The first subproblem is differentiable and we solve it using stochastic gradient descent. For the second subproblem, we can efficiently find the closed-form solution. We iteratively solve the two subproblems until convergence. In our experiments, we achieve notably higher pruning rate on different DNNs compared with prior works.

In Chapter 3, we propose a DNN weight pruning method based on reweighted optimization
methods. For these soft regularization methods, we can decide the degree of sparsity in each layer based on the distribution of weights after convergence. On one hand, the methods reduce the complexity of setting hyper-parameters compared with the hard constraint method, in which the pruning rate in each layer need to be determined at the time when the problem is formulated. On the other hand, reweighted optimization methods act as the better approximation of $\ell_0$ norm and group $\ell_0$ norm compared with prior soft regularization methods, such as $\ell_1$ norm and group Lasso. And in experiments our method achieve higher weight pruning rate without accuracy loss compared with prior works.

In Chapter 4, we first formulate the adversarial attack generation problem as one with a convex objective function but non-convex constraints. The non-convexity of the problem comes from the non-linear activation function in the DNN. However, for piecewise linear activation function such as Rectified Linear Unit (ReLU), we can first approximate it as a linear function to relax the original problem to a convex problem. Then we iteratively solve the convex problem until a convergence condition. We prove that upon the convergence the solution achieved by our method is a feasible solution of the original non-convex problem, and the corresponding adversarial attacks achieve 100% success rate on the DNN. Our experiment results also demonstrate that our distortion on the adversarial attacks are lower than the prior works.

In Chapter 5, we propose a unified min-max framework for robust adversarial attacks, we show how a general notion of min-max optimization over multiple domains can be adapted to attacking model ensembles, and devising universal perturbation under multiple inputs and data transformations. We also show how min-max optimization can generalize adversarial training to a defensive scheme against diversified attack models. Both the generalized attack and defense problems can be solved via the theoretically-grounded min-max framework.

In Chapter 5.7, we study the optimal interdiction problem of transportation networks. Following the prior work, we first formulate the optimal interdiction problem as a min-max optimization problem. Then we prove that the solution to the optimal interdiction problem is both sparse and binary even without an explicit promotion of sparsity in the formulation. We propose a numerical
method to solve the optimal interdiction problem and it performs better compared with the methods proposed in earlier work.

In Chapter 6.7, we summarize the results of this thesis, and present several future research directions.

1.3 Bibliographic note

Most of the research work appearing in this dissertation has already been published at various venues and has appeared in the publications listed below.

**Publications related to the thesis**

**Journal Paper:**


**Conference Papers:**


**Other related contributions**

**Conference Papers:**


• M. Qin, **T. Zhang**, F. Sun, YK. Chen, M. Fardad, Y. Wang, “Compact Multi-level Sparse Neural Networks with Dynamic Rerouting”, under preparation.
CHAPTER 2
A Unified DNN Weight Pruning Framework Using ADMM

2.1 Introduction

Deep neural networks (DNNs) utilize multiple functional layers cascaded together to extract features at multiple levels of abstraction \[3,6–10\], and are thus both computationally and storage intensive. As a result, many studies on DNN model compression are underway, including weight pruning \[11–15\], low-rank approximation \[17,18,35\], low displacement rank approximation (structured matrices) \[36–38\], etc. Weight pruning can achieve a high model pruning rate without loss of accuracy. An early work \[11,12\] adopts an iterative weight pruning heuristic and results in a sparse neural network structure. It can achieve 9× weight reduction with no accuracy loss on AlexNet \[3\]. This weight pruning method has been extended in \[13,15,39,44\] to either use more sophisticated algorithms to achieve a higher weight pruning rate, or to obtain a fine-grained trade-off between a higher pruning rate and a lower accuracy degradation.

Despite the promising results, these general weight pruning methods often produce non-structured and irregular connectivity in DNNs. This leads to degradation in the degree of parallelism and actual performance in GPU and hardware platforms. Moreover, the weight pruning
rate is mainly achieved through compressing the fully-connected (FC) layers \cite{11, 12, 39}, which are less computationally intensive compared with convolutional (CONV) layers and are becoming less important in state-of-the-art DNNs such as ResNet \cite{5}. To address these limitations, recent works \cite{14, 45} have proposed to learn structured sparsity, including sparsity at the levels of filters, channels, filter shapes, layer depth, etc. These works focus on CONV layers and actual GPU speedup is reported as a result of structured sparsity \cite{14}. However, these structured weight pruning methods are based on fixed regularization techniques and are still quite heuristic \cite{14, 45}. The weight pruning rate and GPU acceleration are both quite limited. For example, the average weight pruning rate on CONV layers of AlexNet is only $1.5 \times$ without any accuracy loss, corresponding to 33.3% sparsity.

In this work, we overcome this limitation by proposing a unified, systematic framework of structured weight pruning for DNNs, named StructADMM, based on the powerful optimization tool Alternating Direction Method of Multipliers (ADMM) \cite{46, 47} shown to perform well on combinatorial constraints. It is a unified framework for different types of structured sparsity such as filter-wise, channel-wise, and shape-wise sparsity, as well as non-structured sparsity. It is a systematic framework of dynamic ADMM regularization and masked mapping and retraining steps, guaranteeing solution feasibility (satisfying all constraints) and providing high solution quality. It achieves a significant improvement in weight pruning rate under the same accuracy, along with fast convergence rate. In the context of deep learning, the StructADMM framework can be understood as a smart and dynamic regularization technique in which the regularization target is analytically updated in each iteration.

Beyond the above single-step, one-shot ADMM framework, we observe the opportunity of performing further weight pruning from the results. This is due to the special property of $\ell_2$-based ADMM regularization process. This observation suggests a progressive, multi-step model compression framework using ADMM. In the progressive framework, the pruning result from the previous step serves as intermediate result and starting point for the subsequent step. It has an additional benefit of reducing the search space for (structured) weight pruning within each step.
Detailed procedure and hyperparameter determination process have been carefully designed towards ultra-high weight pruning rates.

During the post-processing procedure, we find that after model retraining, some weights become less contributing to the network performance. To take advantage of this characteristics, we propose a novel algorithm to detect and remove the redundant weights which slip away from ADMM (structured) weight pruning. Also, we are the first to discover the unused path in a structured pruned DNN model and design an effective optimization framework to further boost compression rate as well as maintain high network accuracy.

2.2 Related work

**General, non-structured weight pruning.** The early work by Han et al. [11][12] achieved 9× reduction in the number of parameters in AlexNet and 13× in VGG-16. However, most reduction is achieved in FC layers, and 2.7× reduction achieved in CONV layers will not lead to an overall acceleration in GPU [14]. Extensions of iterative weight pruning, such as [39] (dynamic network surgery), [13] (NeST) and [48], use more delicate algorithms such as selective weight growing and pruning. But the weight pruning rates on CONV layers are still limited, e.g., 3.1× in [39], 3.23× in [13], and 4.16× in [48] for AlexNet with no accuracy degradation. This level of non-structured weight pruning cannot guarantee GPU acceleration. In fact, our StructADMM framework can achieve 16.1× non-structured weight pruning in CONV layers of AlexNet without accuracy degradation, however, only minor GPU acceleration is actually observed.

**Structured weight pruning.** To overcome the limitation in non-structured, irregular weight pruning, SSL [14] proposes to learn structured sparsity at the levels of filters, channels, filter shapes, layer depth, etc. This work is one of the first with actually measured GPU accelerations. This is because CONV layers after structured pruning will transform to a full matrix multiplication in GPU (with reduced matrix size). However, the weight pruning rate and GPU acceleration are both limited. The average weight pruning rate on CONV layers of AlexNet is only 1.5× without
accuracy loss. The reported GPU acceleration is 49%. Besides, another work [45] achieves 2× channel pruning with 1% accuracy degradation on VGGNet.

**Other types of DNN model compression techniques.** There are many other types of DNN model compression techniques. Weight quantization leverages the inherent redundancy in the number of bits for weight representation. Many of the prior work [49–55] are directed at quantization of weights to binary values, ternary values, or powers of 2 to facilitate hardware implementations, with acceptable accuracy loss. The state-of-the-art techniques [56, 57] adopt an iterative quantization and retraining framework, with some degree of randomness incorporated into the quantization step. This method results in less than 3% accuracy loss on AlexNet for binary weight quantization [57]. Furthermore, knowledge distillation leverages the idea that a smaller student model can absorb knowledge from the larger teacher model [58–61], low-rank approximation using singular-value decomposition (SVD) [17–18, 35], and low-displacement rank approximation using structured matrices such as circulant matrices [36, 62], Toeplitz matrices [37, 38], etc. These techniques result in a regular network structure, but in general a lower pruning rate and larger accuracy degradation compared with parameter pruning. We point out that these compression techniques are compatible with ADMM and will be the topic of future investigations orthogonal to this work.

Besides the works we mention above, there are also several representative recent works in this field. References [63–65] define metrics to measure the importance of the weights, and they prune the weights which are less important according to their metrics. [66, 67] define optimization targets to generate sparse DNNs, and they set the optimization target as a regularization term when they train the DNNs. In these methods, the authors decide the importance of weights on a static model or setting a static optimization target as the regularization term. However, in our method the regularization targets are updated dynamically during the training procedure. This is the major difference between our method and these methods. Note that there are also recent works use dynamically updated approaches in the training to prune DNNs, such as C-SGD [68] and CNN-FCF [69]. C-SGD trains several filters to collapse into a single point in the parameter hyperspace and then remove the identical filters. CNN-FCF defines dynamically updated binary scalars to
constraint the filters and remove the filters corresponding to 0-valued scalars after convergence. Both of the two methods focus only on filter pruning on DNNs, but our framework is unified for different kind of structured pruning, as well as non-structured pruning.

2.3 Problem statement

Consider an \( N \)-layer DNN in which the first \( M \) layers are CONV layers and the rest are FC layers. The weights and biases of the \( i \)-th layer are respectively denoted by \( W_i \) and \( b_i \). Assume that the input to the DNN is \( x \). Every column of \( x \) corresponds to a training image, and the number \( t \) of columns determines the number of training images in the input batch. The input \( x \) will enter the first layer and the output of the first layer is calculated by

\[
h_1 = \sigma(W_1 x + b_1),
\]

where \( h_1 \) and \( b_1 \) have \( t \) columns, and \( b_1 \) is a matrix with identical columns. The non-linear activation function \( \sigma(\cdot) \) acts entrywise on its argument, and is typically chosen to be the ReLU function \([70]\) in state-of-the-art DNNs. Since the output of one layer is the input of the next, the output of the \( i \)-th layer for \( i = 2, \ldots, N - 1 \) is given by

\[
h_i = \sigma(W_i h_{i-1} + b_i).
\]

The output of the DNN corresponding to a batch of images is

\[
s = W_N h_{N-1} + b_N.
\]

In this case \( s \) is a \( k \times t \) matrix, where \( k \) is the number of classes in the classification, and \( t \) is the number of training images in the batch. The element \( s_{ij} \) in matrix \( s \) is the score of the \( j \)-th training
image corresponding to the $i$-th class. The total loss of the DNN is calculated as

$$f(\{W_1, \ldots, W_N\}, \{b_1, \ldots, b_N\}) = -\frac{1}{t} \sum_{j=1}^{t} \log \frac{e^{s_{y_j}j}}{\sum_{i=1}^{N} e^{s_{ij}}} + \lambda \sum_{i=1}^{N} \|W_i\|_F^2,$$

where the first term is cross-entropy loss, $y_j$ is the correct class of the $j$-th image, and the second term is $\ell_2$ weight regularization.

Hereafter, for simplicity of notation we write $\{W_i\}_{i=1}^{N}$, or simply $\{W_i\}$ instead of $\{W_1, \ldots, W_N\}$. The same notational convention applies other variables and parameters. The training of a DNN is a process of minimizing the loss by updating weights and biases. If we use the gradient descent method then the update at every step is

$$W_i = W_i - \alpha \frac{\partial f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N})}{\partial W_i},$$

$$b_i = b_i - \alpha \frac{\partial f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N})}{\partial b_i},$$

computed for $i = 1, \ldots, N$, where $\alpha$ is the learning rate.

In this chapter, our objective is to implement structured pruning on DNNs. In the following discussion we focus on the CONV layers because they have the highest computation requirements. More specifically, we minimize the loss function subject to specific structured sparsity constraints on the weights in the CONV layers, i.e.,

$$\begin{align*}
\text{minimize} \quad & f(\{W_i\}, \{b_i\}), \\
\text{subject to} \quad & W_i \in S_i, \quad i = 1, \ldots, M,
\end{align*}$$

(2.1)

where $S_i$ is the constraint set of weight structure. Next we introduce constraint sets corresponding to different types of structured sparsity. Non-structured, irregular sparsity is also included in the framework. The suitability for GPU acceleration is discussed for different types of sparsity, and we finally introduce the proper combination of structured sparsity to facilitate GPU accelerations. The details of different types of structures will be discussed later in Section 2.4.1.

In problem (2.1) the constraint is non-convex and combinatorial. As a result, this problem
cannot be solved directly by stochastic gradient descent methods (SGD) [33] (or ADAM [34]). However, the property that $W_i$ satisfies certain combinatorial “structures” allows us to integrate the ADMM framework with stochastic gradient descent to effectively solve this problem.

### 2.4 A unified DNN weight pruning framework using ADMM

To apply the ADMM framework, we (i) define indicator functions to incorporate combinatorial constraints into the objective function, and (ii) define auxiliary variables that allow us to decompose the optimization problem into two subproblems that individually can be solved effectively. In what follows, we elaborate on these steps.

Corresponding to every set $S_i$, $i = 1, \ldots, M$ we define the indicator function

$$g_i(W_i) = \begin{cases} 
0 & \text{if } W_i \in S_i, \\
+\infty & \text{otherwise.} 
\end{cases}$$

Furthermore, we incorporate auxiliary variables $Z_i$, $i = 1, \ldots, M$ with the restriction that $Z_i = W_i$. The original problem (2.1) is then equivalent to

$$\begin{aligned}
\min_{\{W_i\}, \{b_i\}} & \quad f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^M g_i(Z_i), \\
\text{subject to} & \quad W_i = Z_i, \quad i = 1, \ldots, M.
\end{aligned}$$

(2.2)

The augmented Lagrangian [46] of problem (2.2) is defined by

$$L_\rho(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N, \{Z_i\}_{i=1}^M, \{\Lambda_i\}_{i=1}^M) = f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^M g_i(Z_i)$$

$$\begin{aligned}
& + \sum_{i=1}^M \text{tr}[\Lambda_i^T(W_i - Z_i)] + \sum_{i=1}^M \frac{\rho_i}{2} ||W_i - Z_i||_F^2,
\end{aligned}$$

where $||\cdot||_F$ denotes the Frobenius norm, $\{\Lambda_i\}_{i=1}^M$ are the dual variables, and the penalty parameters
\( \{\rho_i\}_{i=1}^M \) are positive. With the scaled dual variable \( U_i = (1/\rho_i)\Lambda_i \) for \( i = 1, \ldots, M \), the augmented Lagrangian can be equivalently rewritten as

\[
L_\rho(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N, \{Z_i\}_{i=1}^M, \{\Lambda_i\}_{i=1}^M) = f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^M g_i(Z_i) + \sum_{i=1}^M \frac{\rho_i}{2}\|W_i - Z_i + U_i\|_F^2 - \sum_{i=1}^M \frac{\rho_i}{2}\|U_i\|_F^2.
\]

ADMM consists the following iterations for \( k = 0, 1, \ldots \) [46, 71]:

\[
\{W_i^{k+1}\}_{i=1}^N, \{b_i^{k+1}\}_{i=1}^N := \arg\min_{\{W_i\}, \{b_i\}} L_\rho(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N, \{Z_i^{k}\}_{i=1}^M, \{U_i^k\}_{i=1}^M), \tag{2.3}
\]

\[
\{Z_i^{k+1}\}_{i=1}^M := \arg\min_{\{Z_i\}} L_\rho(\{W_i^{k+1}\}_{i=1}^N, \{b_i^{k+1}\}_{i=1}^N, \{Z_i\}_{i=1}^M, \{U_i^k\}_{i=1}^M), \tag{2.4}
\]

\[
U_i^{k+1} := U_i^k + W_i^{k+1} - Z_i^{k+1} \text{ for } i = 1, \ldots, M, \tag{2.5}
\]

until for \( i = 1, \ldots, M \), both of the following conditions are satisfied

\[
\|W_i^{k+1} - Z_i^{k+1}\|_F^2 \leq \epsilon_i, \quad \|Z_i^{k+1} - Z_i^k\|_F^2 \leq \epsilon_i \tag{2.6}
\]

In order to solve the overall pruning problem, we need to solve subproblems \( (2.3) \) and \( (2.4) \). More specifically, problem \( (2.3) \) can be formulated as

\[
\min_{\{W_i\}, \{b_i\}} f(\{W_i\}_{i=1}^N, \{b_i\}_{i=1}^N) + \sum_{i=1}^M \frac{\rho_i}{2}\|W_i - Z_i^k + U_i^k\|_F^2, \tag{2.7}
\]

where the first term in the objective function of \( (2.7) \) is the differentiable loss function of the DNN, and the second term is a quadratic regularization term of the \( W_i \)'s, which is differentiable and convex. As a result \( (2.7) \) can be solved by stochastic gradient descent. Although we cannot guarantee the global optimality of the solution, it is due to the non-convexity of the DNN loss function rather than the quadratic term enrolled by our method.

On the other hand, problem \( (2.4) \) is given by

\[
\min_{\{Z_i\}} \sum_{i=1}^M g_i(Z_i) + \sum_{i=1}^M \frac{\rho_i}{2}\|W_i^{k+1} - Z_i + U_i^k\|_F^2. \tag{2.8}
\]
Note that \( g_i(\cdot) \) is the indicator function of \( S_i \), thus this subproblem can be solved analytically and optimally \([46]\). For \( i = 1, \ldots, M \), the optimal solution is

\[
Z_{i}^{k+1} = \Pi_{S_i}(W_{i}^{k+1} + U_{i}^{k}),
\]

(2.9)

where \( \Pi_{S_i}(\cdot) \) is Euclidean projection of \( W_{i}^{k+1} + U_{i}^{k} \) onto \( S_i \). The set \( S_i \) is different when we apply different types of structured sparsity. We will discuss how to implement the Euclidean projection to different types of structures in Section 2.4.1.

### 2.4.1 Solutions of different types of structured sparsity and discussions

The collection of weights in the \( i \)-th CONV layer is a four-dimensional tensor, i.e., \( W_i \in \mathbb{R}^{A_i \times B_i \times C_i \times D_i} \), where \( A_i \), \( B_i \), \( C_i \), and \( D_i \) are respectively the number of filters, the number of channels in a filter, the height of the filter, and the width of the filter, in layer \( i \). In what follows, if \( X \) denotes the weight tensor in a specific layer, let \( (X)_{a,:,,:,:} \) denote the \( a \)-th filter in \( X \), \( (X)_{:,b,:,:} \) denote the \( b \)-th channel, and \( (X)_{:,b,c,d} \) denote the collection of weights located at position \((:, b, c, d)\) in every filter of \( X \), as illustrated in Figure 2.1.

![Fig. 2.1: Illustration of filter-wise, channel-wise and shape-wise structured sparsity from left to right.](image-url)
Filter-wise structured sparsity: When we train a DNN with sparsity at the filter level, the constraint on the weights in the $i$-th CONV layer is given by

$$
W_i \in S_i := \{ X \mid \text{the number of nonzero filters in } X \text{ is less than or equal to } \alpha_i \}.
$$

Here, nonzero filter means that the filter contains some nonzero weight. To solve subproblem (2.9) with such constraints, we first calculate

$$
O_a = \| (W_i^{k+1} + U_i^k)_{a,:,:} \|_F^2
$$

for $a = 1, \ldots, A_i$. We then keep $\alpha_i$ elements in $(W_i^{k+1} + U_i^k)_{a,:,:}$ corresponding to the $\alpha_i$ largest values in $\{O_a\}_{a=1}^{A_i}$ and set the rest to zero.

Channel-wise structured sparsity: When we train a DNN with sparsity at the channel level, the constraint on the weights in the $i$-th CONV layer is given by

$$
W_i \in S_i := \{ X \mid \text{the number of nonzero channels in } X \text{ is less than or equal to } \beta_i \}.
$$

Here, we call the $b$-th channel nonzero if $(X)_{:,b,:,:}$ contains some nonzero element. To solve subproblem (2.9) with such constraints, we first calculate

$$
O_b = \| (W_i^{k+1} + U_i^k)_{:,b,:,:} \|_F^2
$$

for $b = 1, \ldots, B_i$. We then keep $\beta_i$ elements in $(W_i^{k+1} + U_i^k)_{:,b,:,:}$ corresponding to the $\beta_i$ largest values in $\{O_b\}_{b=1}^{B_i}$ and set the rest to zero.

Filter shape-wise structured sparsity: When we train a DNN with sparsity at the filter shape level, the constraint on the weights in the $i$-th CONV layer is given by

$$
W_i \in S_i := \{ X \mid \text{the number of nonzero vectors in } \{X_{:,b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i} \text{ is less than or equal to } \theta_i \}.
$$
To solve subproblem (2.9) with such constraints, we first calculate

$$O_{b,c,d} = \| (W_{k+1}^i + U_i^k)_{b,c,d} \|_F^2$$

for $b = 1, \ldots, B_i$, $c = 1, \ldots, C_i$ and $d = 1, \ldots, D_i$. We then keep $\theta_i$ elements in $(W_{k+1}^i + U_i^k)_{b,c,d}$ corresponding to the $\theta_i$ largest values in $\{O_{b,c,d}\}_{b,c,d=1}^{B_i,C_i,D_i}$ and set the rest to zero.

**Non-structured, irregular weight sparsity:** When we train a DNN with non-structured weight sparsity, the constraint on the weights in $i$-th CONV layer is

$$W_i \in S_i := \{X \mid \text{the number of nonzero elements in } X \text{ is less than or equal to } \gamma_i \}.$$ 

To solve subproblem (2.9), we keep $\gamma_i$ elements in $W_{k+1}^i + U_i^k$ with largest magnitudes and set the rest to zero.

**Combination of structured sparsity to facilitate GPU acceleration:** Convolutional computations in DNNs are commonly transformed to matrix multiplications by converting weight tensors and feature map tensors to matrices [72], named *general matrix multiplication* or GEMM. Filter-wise sparsity corresponds to row pruning, whereas channel-wise and filter shape-wise sparsity correspond to column pruning in GEMM. The GEMM matrix maintains a full matrix with number of rows/columns reduced, thereby enabling GPU acceleration. This is illustrated in Figure 2.2 in which $W_{n,m,k}$ means the $k$-th element in the $m$-th channel of the $n$-th filter. In the results we will use *row pruning* to represent the results of filter-wise sparsity in GEMM, and use *column pruning* to represent the results of channel-wise and filter shape-wise sparsity.

2.4.2 The masked retraining step

For very small values of $\epsilon_i$ in (2.6), ADMM needs a large number of iterations to converge. However, in many applications, such as the weight pruning problem considered here, a slight increase in the value of $\epsilon_i$ can result in a significant speedup in convergence. On the other hand, when ADMM stops early the weights to be pruned may not be identically zero, in the sense that there
will be small nonzero elements contained in $W_i$. To deal with this issue, we first perform the Euclidean projection to guarantee that the structured pruning constraints are satisfied. Next, we mask the zero weights and retrain the DNN with non-zero weights using training sets (while keeping the masked weights 0). In this way test accuracy (solution quality) can be partially restored. Note that the convergence is much faster than training the original DNN, since the starting point of the retraining is already close to the point which can achieve the original test/validation accuracy.

2.4.3 Overall illustration of our proposed framework

We take the weight distribution of every (convolutional or fully connected) layer on LeNet-5 as an example to illustrate our systematic weight pruning method. The weight distributions at different stages are shown in Figure 2.3. The subfigures in the left column show the weight distributions of the pretrained model, which serves as our starting point. The subfigures in the middle column show that after the convergence of ADMM for moderate values of $\epsilon_i$, we observe a clear separation between weights whose values are close to zero and the remaining weights. To prune the weights rigorously, we set the values of the close-to-zero weights exactly to zero and retrain the DNN without updating these values. The subfigures in the right column show the weight distributions after our final retraining step. We observe that most of the weights are zero in every layer. This concludes our weight pruning procedure.

As mentioned before, the computation time for the ADMM procedure is similar to the training of the original DNN, and the single retraining step converges much faster than the original training.
Fig. 2.3: Weight distribution of every (convolutional or fully connected) layer on LeNet-5. The subfigures in the left column are the weight distributions of the pretrained DNN model (serving as our starting point); the subfigures of the middle column are the weight distributions after the ADMM procedure; the subfigures of the right column are the weight distributions after our final retraining step. Note that the subfigures in the last column include a small number of nonzero weights that are not clearly visible due to the large number of zero weights.
Algorithm 1 An overall illustration of our proposed StructADMM framework

**Input:** Pretrained model
Initialize \( \{Z_0^i\} \) and \( \{U_0^i\} \)
Set \( j = 0 \) and \( k = 0 \).
Set \( T \) as the number of iterations of ADMM
Set \( \beta \) as the number of epochs in every iteration of ADMM

**for** \( k \leq T \) **do**

**for** \( j \leq \beta \) **do**

Solve problem (2.7) using one epoch of SGD or ADAM

**end for**

Update \( \{Z_{k+1}^i\} \) according to (2.9)

Update \( \{U_{k+1}^i\} \) according to (2.5)

**if** Condition (2.6) is satisfied **then**

Break for loop

**end if**

**end for**

Perform the Euclidean projection according to (2.9) to guarantee that the structured pruning constraints are satisfied. Then mask the zero weights and retrain the DNN with the non-zero weights.

Consequently, the total computation time of our method is less than training the original DNN twice, which is much faster than the iterative pruning and training method in [73]. An overall illustration of our proposed StructADMM framework is shown in Algorithm 1.

### 2.5 Methods to improve pruning rate

#### 2.5.1 Progressive DNN weight pruning

The first motivation of the progressive framework is that during the implementation of the single-step weight pruning framework, we observe that there are a number of unpruned weights with values very close to zero. The reason is the \( \ell_2 \) regularization nature in ADMM regularization step, which tends to generate very small, non-zero weight values even when they are not pruned. As the remaining number of non-zero weights is already significantly reduced during weight pruning, simply mapping these small-value weights to zero will result in accuracy degradation. On the other hand, this motivates us to perform weight pruning in a multi-step, progressive manner. The weights
that have been pruned in the previous step will be masked and only the remaining, non-zero weights will be considered in the subsequent step.

The second motivation of the progressive framework is to reduce the search space for weight pruning within each step. After all, weight pruning problems are essentially combinatorial optimizations. Although recently demonstrated to generate superior results on this type of problems \cite{74,75}, ADMM-based solution still has a superlinear increase of computational complexity as a function of solution space. As a result, the complexity becomes very high with ultra-high compression rates (i.e., very large search space) beyond what can be achieved in prior work. The progressive framework, on the other hand, can mitigate this limitation and reduce the total training time (to $2 \times$ or slightly higher than training time of the original DNN).

Similar approach that masking the zero weights in the model generated from the previous iteration of pruning has been applied in \cite{12,42}, but our motivation is different from these works. Magnitude-based pruning method is used in \cite{12,42}, this method is heuristic and the pruning rate have to be iteratively increased to avoid accuracy loss. One-step of our ADMM-based method can achieve much higher pruning rate than iterative magnitude-based pruning method without accuracy loss. And our major purpose of using progressive method is to reduce the search space in each step to achieve ultra-high pruning rate.

Also, masking the zero weights is not necessary for our method to reduce the search space. In our experiment we find that even if we do not mask the zero weights pruned in each step when we start a new step, the value of the unimportant weights still keep close to zero. This is because the ADMM-based regularization term prevent these unimportant weights to move away from zero. In conclusion, if we start from a model which is already pruned, the search space for our ADMM-based method is being reduced no matter if we mask the zero weights or not. We choose to mask the zero weights to make the training more rigorous.

Figure 2.4 illustrates our proposed progressive DNN weight pruning method on StructADMM framework. The single-step ADMM-based weight pruning is performed multiple times, each as a step in the progressive framework. The pruning results from the previous step serve as intermediate
results and starting point for the subsequent step. Through extensive investigations, we conclude that a two-step progressive procedure will be in general sufficient for weight pruning, in which each step requires approximately the same number of training epochs as original DNN training. Further increase in the number of steps or the number of epochs in each step will result in only marginal improvement in the overall solution quality (e.g., 0.1%-0.2% accuracy improvement).

### 2.5.2 Network purification and unused path removal

After ADMM-based structured weight pruning, we propose the network purification and unused path removal step for further weight reduction without accuracy loss. First, as also noticed by prior work [45], a specific filter in layer $i$ is responsible for generating one channel in layer $i + 1$. As a result, removing the filter in layer $i$ (in fact removing the batch norm results) also results in the removal of the corresponding channel, thereby achieving further weight reduction. Besides this straightforward procedure, there is further margin of weight reduction based on the characteristics of ADMM regularization. As ADMM regularization is essentially a dynamic, $\ell_2$-norm based regularization procedure, there are a large number of non-zero, small weight values after regularization. Due to the non-convex property in ADMM regularization, our observation is that removing these weights can maintain the accuracy or even slightly improve the accuracy occasionally. As a result, we define two thresholds, a column-wise threshold and a filter-wise threshold, for each DNN layer. When the $\ell_2$ norm of a column (or filter) of weights is below the threshold, the column (or filter) will be removed. Also the corresponding channel in layer $i + 1$ can be removed upon filter removal.
in layer $i$. Structures in each DNN layer will be maintained after this purification step.

### 2.6 Numerical results

#### 2.6.1 Experiment results for structure pruning

First, we compare our method with the two configurations of the SSL method [14] on AlexNet/CaffeNet. The first has no accuracy degradation (Top-1 error 42.53%) and average sparsity of 33.3% on conv2-conv5. We note that the 1st CONV layer of AlexNet/CaffeNet is very small with only 35K weights compared with 2.3M in conv2-conv5, and is often not the optimization focus [14, 35]. The second has around 2% accuracy degradation (Top-1 error 44.66%) with total sparsity of 84.4% on conv2-conv5. The first configuration focuses on column sparsity only. The second configuration focuses on a combined row (filter) and column sparsity.

Table 2.1 shows the comparison of our method with the first configuration of SSL. We generate configuration with no accuracy degradation compared with original model (the original model of our work is higher than that in SSL). We can achieve a much higher degree of sparsity of 79.2% on conv2-conv5. This corresponds to $4.8 \times$ weight pruning rate, which is significantly higher compared with $1.5 \times$ pruning in conv2-conv5 in [14].

**Table 2.1:** Comparison of our method and SSL method on column sparsity **without accuracy loss** on AlexNet/CaffeNet model for ImageNet data set

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-1 Acc</th>
<th>Statistics</th>
<th>conv1</th>
<th>conv2</th>
<th>conv3</th>
<th>conv4</th>
<th>conv5</th>
<th>conv2-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL [14]</td>
<td>0%</td>
<td>column sparsity</td>
<td>0.0%</td>
<td>20.9%</td>
<td>39.7%</td>
<td>39.7%</td>
<td>24.6%</td>
<td>33.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pruning rate</td>
<td>1.0×</td>
<td>1.3×</td>
<td>1.7×</td>
<td>1.7×</td>
<td>1.3×</td>
<td>1.5×</td>
</tr>
<tr>
<td><strong>our method</strong></td>
<td>0%</td>
<td>column sparsity</td>
<td>0.0%</td>
<td>70.0%</td>
<td>77.0%</td>
<td>85.0%</td>
<td>81.0%</td>
<td>79.2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pruning rate</td>
<td>1.0×</td>
<td>2.27×</td>
<td>3.35×</td>
<td>3.64×</td>
<td>1.04×</td>
<td>2.58×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GPU1×</td>
<td>1.00×</td>
<td>2.83×</td>
<td>3.92×</td>
<td>4.63×</td>
<td>3.22×</td>
<td>3.65×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GPU2×</td>
<td>1.0×</td>
<td>3.3×</td>
<td>4.3×</td>
<td>6.7×</td>
<td>5.3×</td>
<td>4.8×</td>
</tr>
<tr>
<td>Method</td>
<td>Top-1 Acc</td>
<td>Statistics</td>
<td>conv1</td>
<td>conv2</td>
<td>conv3</td>
<td>conv4</td>
<td>conv5</td>
<td>conv2-5</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>---------</td>
</tr>
<tr>
<td>SSL [14]</td>
<td>2.0%</td>
<td>column sparsity</td>
<td>0.0%</td>
<td>63.2%</td>
<td>76.9%</td>
<td>84.7%</td>
<td>80.7%</td>
<td>84.4%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>row sparsity</td>
<td>9.4%</td>
<td>12.9%</td>
<td>40.6%</td>
<td>46.9%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pruning rate</td>
<td>1.1×</td>
<td>3.2×</td>
<td>7.7×</td>
<td>12.3×</td>
<td>5.2×</td>
<td>6.4×</td>
</tr>
<tr>
<td>our method</td>
<td>0.7%</td>
<td>column sparsity</td>
<td>0.0%</td>
<td>63.9%</td>
<td>78.1%</td>
<td>87.0%</td>
<td>84.9%</td>
<td>86.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>row sparsity</td>
<td>9.4%</td>
<td>12.9%</td>
<td>40.6%</td>
<td>46.9%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pruning rate</td>
<td>1.1×</td>
<td>3.1×</td>
<td>7.3×</td>
<td>14.5×</td>
<td>6.6×</td>
<td>7.3×</td>
</tr>
<tr>
<td>our method</td>
<td>2.0%</td>
<td>column sparsity</td>
<td>0.0%</td>
<td>87.5%</td>
<td>90.0%</td>
<td>90.5%</td>
<td>90.7%</td>
<td>93.7%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>row sparsity</td>
<td>9.4%</td>
<td>12.9%</td>
<td>40.6%</td>
<td>46.9%</td>
<td>0.0%</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pruning rate</td>
<td>1.1×</td>
<td>9.2×</td>
<td>16.8×</td>
<td>19.8×</td>
<td>8.4×</td>
<td>15.0×</td>
</tr>
</tbody>
</table>

§ Total sparsity accounting for both column and row sparsity.

We test the actual GPU accelerations using two GPUs: GPU1 is NVIDIA 1080Ti and GPU2 is NVIDIA TX2. The acceleration rate is computed with respect to the corresponding layer of the original DNN executing on the same GPU and same setup. One can observe that the average acceleration of conv2-conv5 on 1080Ti is 2.58×, while the average acceleration on TX2 is 3.65×. These results clearly outperform the GPU acceleration of 49% reported in SSL [14] without accuracy loss, as well as the more recent work [35]. The acceleration rate on TX2 is higher than 1080Ti because the latter has a high parallelism degree, which will not be fully utilized when the matrix size GEMM of a CONV layer is significantly reduced. Another technique cuDNN can use implicit GEMM for better performance than cuBLAS, but we still outperform it on the inference
Table 2.3: Structured pruning results on ResNet-18 model for ImageNet data set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Prune rate</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1.0×</td>
<td>89.0%</td>
</tr>
<tr>
<td>DCP [66]</td>
<td>1.5×</td>
<td>88.9%</td>
</tr>
<tr>
<td>DCP [66]</td>
<td>2.0×</td>
<td>87.6%</td>
</tr>
<tr>
<td><strong>Our method</strong> (column prune)</td>
<td>2.5×</td>
<td>89.0%</td>
</tr>
<tr>
<td><strong>Our method</strong> (filter + column prune)</td>
<td>3.0×</td>
<td>88.7%</td>
</tr>
</tbody>
</table>

Table 2.4: Structured pruning results on ResNet-50 model for ImageNet data set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Prune rate</th>
<th>Top 5 accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>1.0×</td>
<td>92.7%</td>
</tr>
<tr>
<td>ThiNet-50 [76]</td>
<td>2.0×</td>
<td>90.0%</td>
</tr>
<tr>
<td>ThiNet-30 [76]</td>
<td>3.3×</td>
<td>88.3%</td>
</tr>
<tr>
<td>NISP [63]</td>
<td>1.8×</td>
<td>90.2%</td>
</tr>
<tr>
<td>Efficient ConvNet [64]</td>
<td>1.4×</td>
<td>91.1%</td>
</tr>
<tr>
<td>DCP [66]</td>
<td>2.0×</td>
<td>92.3%</td>
</tr>
<tr>
<td><strong>Our method</strong> (filter prune)</td>
<td>2.7×</td>
<td>91.9%</td>
</tr>
<tr>
<td><strong>Our method</strong> (filter + column prune)</td>
<td>2.7×</td>
<td>92.3%</td>
</tr>
</tbody>
</table>

time without accuracy loss. For example, We compare our result in Table 2.1 to the baseline results with cuDNN, our inference time with cuBLAS is also 2.8x lower than the result with baseline cuDNN on original, uncompressed DNN on TX2 GPU.

Table 2.2 shows the comparison with the second configuration of SSL. With similar (and slightly higher) sparsity in each layer as SSL, we can achieve less accuracy loss. With the same relative accuracy loss (a moderate accuracy loss within 2% compared with original DNN), a higher degree of 93.7% average sparsity of conv2-conv5 is achieved, translating into 15.0× weight pruning. The actual GPU acceleration results are also high: 3.15× on 1080Ti and 8.52× on TX2. One can clearly see that the speedup on 1080Ti saturates because the high parallelism degree cannot be fully exploited. The acceleration on CPU can be higher under this setup, reaching 11.93× on average on conv2-conv5.

Figure 2.5 shows the convergence behavior of ADMM regularization (in StructADMM frame-
Fig. 2.5: Convergence behavior of the ADMM regularization procedure for the convolutional layers 2-5 of AlexNet.

work, single-step ADMM), using the experiment that achieves 4.8× structured pruning rate on AlexNet without accuracy loss. We can observe that ADMM regularization converges in around 12 iterations. The number of ADMM iterations is generally 9 - 12 for most of test cases. In each iteration, we need around 10% to 20% of the number of epochs as original DNN training.

Additionally, in Table 2.3 and Table 2.4, we demonstrate the structured (filter or column) pruning results on ResNet-18 and ResNet-50 models for ImageNet data set. We compare with a list of prior work, and these prior work focus on filter pruning only (we do not find prior work on column pruning on these two models). As shown in the tables, we achieve simultaneously higher pruning rate (weight parameter reduction rate) and higher accuracy compared with prior work, when only applying filter (row) pruning. Also, we can observe that column pruning results in higher pruning rate and/or higher accuracy compared with filter pruning. This is because of the higher flexibility in column pruning by modifying filter shapes. For large-scale GPU/CPU acceleration when GEMM
computation is utilized, column pruning (and potentially effective combination with filter pruning) will be more effective than filter pruning only.

2.6.2 Experiment results for non-structured pruning

We evaluate our method for non-structured pruning on AlexNet and VGG-16 for ImageNet data set. Since we propose a progressive, multi-step weight pruning framework, we achieve a higher pruning rate than the single-step ADMM method in the early version of our work \cite{77} on AlexNet. We also achieve much higher pruning rate compared with other prior work.

Table 2.5: Comparisons of weight pruning results on AlexNet model for ImageNet data set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-5 Acc.</th>
<th>No. Para.</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>80.3%</td>
<td>61.0M</td>
<td>1×</td>
</tr>
<tr>
<td>Network Pruning \cite{73}</td>
<td>80.3%</td>
<td>6.7M</td>
<td>9×</td>
</tr>
<tr>
<td>Optimal Brain Surgeon \cite{40}</td>
<td>80.0%</td>
<td>6.7M</td>
<td>9.1×</td>
</tr>
<tr>
<td>Low Rank and Sparse Decomposition \cite{78}</td>
<td>80.3%</td>
<td>6.1M</td>
<td>10×</td>
</tr>
<tr>
<td>Fine-Grained Pruning \cite{48}</td>
<td>80.4%</td>
<td>5.1M</td>
<td>11.9×</td>
</tr>
<tr>
<td>NeST \cite{13}</td>
<td>80.2%</td>
<td>3.9M</td>
<td>15.7×</td>
</tr>
<tr>
<td>Dynamic Surgery \cite{39}</td>
<td>80.0%</td>
<td>3.4M</td>
<td>17.7×</td>
</tr>
<tr>
<td>Single-step ADMM \cite{77}</td>
<td>80.2%</td>
<td>2.9M</td>
<td>21×</td>
</tr>
<tr>
<td>Hoyer-Square \cite{67}</td>
<td>80.2%</td>
<td>2.86M</td>
<td>21.3×</td>
</tr>
<tr>
<td><strong>Progressive Weight Pruning</strong></td>
<td>80.2%</td>
<td>2.02M</td>
<td>30×</td>
</tr>
<tr>
<td><strong>Progressive Weight Pruning</strong></td>
<td>80.0%</td>
<td>1.97M</td>
<td>31×</td>
</tr>
</tbody>
</table>

Table 2.6: Comparisons of weight pruning results on VGG-16 model for ImageNet data set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-5 Acc.</th>
<th>No. Para.</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>88.7%</td>
<td>138M</td>
<td>1×</td>
</tr>
<tr>
<td>Network Pruning \cite{73}</td>
<td>89.1%</td>
<td>10.6M</td>
<td>13×</td>
</tr>
<tr>
<td>Optimal Brain Surgeon \cite{40}</td>
<td>89.0%</td>
<td>10.3M</td>
<td>13.3×</td>
</tr>
<tr>
<td>Low Rank and Sparse Decomposition \cite{78}</td>
<td>89.1%</td>
<td>9.2M</td>
<td>15×</td>
</tr>
<tr>
<td><strong>Progressive Weight Pruning</strong></td>
<td>88.7%</td>
<td>4.6M</td>
<td>30×</td>
</tr>
<tr>
<td><strong>Progressive Weight Pruning</strong></td>
<td>88.2%</td>
<td>4.1M</td>
<td>34×</td>
</tr>
</tbody>
</table>

Table 2.5 presents the weight pruning comparison results on the AlexNet model between our proposed method and prior work. Our weight pruning results clearly outperform the prior work, in
that we can achieve $31 \times$ weight reduction rate without loss of accuracy. Our progressive weight pruning also outperforms the single-step ADMM weight pruning in [77] that achieves $21 \times$ compression rate.

Table 2.6 presents the comparison results on VGG-16. These weight pruning results we achieved clearly outperform the prior work, consistently achieving the highest sparsity in the benchmark DNN models. On the VGG-16 model, we achieve $30 \times$ weight pruning with comparable accuracy with prior work, while the highest pruning rate in prior work is $19.5 \times$. We also achieve $34 \times$ weight pruning with minor accuracy loss.

In summary, the experimental results demonstrate that our framework applies to a broad set of representative DNN models and consistently outperforms the prior work. It also applies to the DNN models that consist of mainly convolutional layers, which are different with weight pruning for prior methods. These promising results will significantly contribute to the energy-efficient implementation of DNNs in mobile and embedded systems, and on various hardware platforms.

### 2.7 Summary

In this chapter, we proposed a unified, systematic framework of structured weight pruning for DNNs. It is a unified framework for different types of structured sparsity such as filter-wise, channel-wise, shape-wise sparsity as well as non-structured sparsity. By incorporating stochastic gradient descent with ADMM, our framework updates regularization target analytically in each iteration. Based on ADMM, we further propose a progressive weight pruning framework and a network purification and unused path removal procedure, in order to achieve higher pruning rate without accuracy loss. In our experiments, we achieve $2.58 \times$ and $3.65 \times$ measured speedup on two GPUs without accuracy loss. The speedups reach $3.18 \times$ and $8.52 \times$ on GPUs and $10.5 \times$ on CPU when allowing moderate accuracy loss of 2%, and reaches $7.6 \times$ on the Adreno 640 mobile GPU. Our pruning rate and speedup clearly outperform prior work.
CHAPTER 3
A UNIFIED DNN WEIGHT PRUNING FRAMEWORK USING REWEIGHTED OPTIMIZATION METHODS

3.1 Introduction

DNNs have achieved impressive results in many fields including image classification [3], natural language processing [6], autonomous vehicles [79], etc. The state-of-the-art DNNs have large model size and computational requirement, which impedes the critical requirements (e.g., real time, low power) in the inference phase. To address these challenges, prior works have focused on developing DNN model compression techniques, such as weight pruning [12, 14, 39, 77].

The objective of DNN weight pruning is to reduce the number of non-zero elements in the weight matrix while maintaining the prediction accuracy. Early works in weight pruning utilize a static, magnitude-based method [12] or ℓ₁-based regularization [14] to explore sparsity in DNN models. Although these methods can prune the weights without accuracy loss, they are heuristic and can only find a small parts of non-critical weights to prune. To overcome the accuracy degradation while further prune the DNNs, several works [66, 69, 80, 81] propose more well-developed
methods to increase the pruning rate based on the sparsity types proposed by \cite{14}. Recently, reference \cite{77} used the alternating direction method of multiplier (ADMM) \cite{82} to solve the $\ell_0$ constraint problem and achieve good performance on pruning rate without accuracy loss. With the powerful ADMM optimization framework, pruning problems are re-formed into optimization problems with the dynamically updated regularization terms bounded by the designated hard constraint sets which enable arbitrary desired pruning dimensions to fulfill the vast design space. However, ADMM suffers from long convergence time due to the strong non-convexity of the $\ell_0$ constraints. Additionally, it is a highly time-consuming process to set the hyperparameters manually in a hard constraints problem, which intrinsically is a heuristic exploration that mainly relies on the experiences of the designer, and the derived hyperparameters are typically sub-optimal. It is imperative to find a better solution to solve the pruning problem with high efficient and self-adaptive regularization that can automatically determine pruning hyperparameters and maintain accuracy.

In this paper, we propose a unified DNN weight pruning framework with dynamically updated regularization terms bounded by the designated constraint, which can generate both non-structured sparsity and different kinds of structured sparsity. In non-structured pruning, we need to reduce the $\ell_0$ norm of the weight matrix, but it is an intractable problem since $\ell_0$ norm is non-convex and discrete. To deal with this issue, we solve the reweighted $\ell_1$ problem \cite{83} as the proxy of $\ell_0$ problem. Structured pruning requires not only the sparsity in weights but also the position of the zeros elements. To generate different kinds of group sparsity, we propose to use a reweighted method on group lasso regularization. In our proposed framework, we first use a reweighted method to regularize the model, then remove the weights which are close to zero and mask the gradient of these weights to ensure that they no longer update. We retrain the remaining non-zero weights to retrieve the accuracy. We adopt the reweighted regularization method with designated sparsity types, which avoids strongly non-convex $\ell_0$-norm based hard constraints in the state-of-the-art ADMM formulation, therefore accelerates the convergence and reduces the number of hyperparameters. Since the loss function of DNNs is non-convex, when we use the reweighted method we also need to solve
a non-convex problem and cannot achieve the globally optimal solution. This motivates us to use the reweighted method again on the sparse model achieved in our first step. After implementing the reweighted method for several more steps, we achieve higher pruning rate.

For both non-structured and structured pruning, our method achieve higher pruning rate than state-of-the-arts while maintaining accuracy. For non-structured pruning, we achieve $630 \times$ pruning rate on LeNet-5 for MNIST and $45 \times$ pruning rate on AlexNet for ImageNet with minor accuracy loss. For structured pruning, we achieve $4.2 \times$ and $3.2 \times$ pruning rate on the convolutional layers of ResNet-18 and ResNet-50 for ImageNet and $7.2 \times$ on the convolutional layers of MobileNet-V2-1.0 for CIFAR-10 with negligible accuracy loss.

3.2 Related work

3.2.1 Static regularization-based pruning

Early works in weight pruning utilize a static, magnitude-based method or $\ell_1$-based regularization to explore sparsity in DNN models. With specified regularization dimensions on weight vectors, we can perform different types of pruning method including non-structured pruning and structured pruning, but with limited compression rates and non-negligible accuracy degradation due to the intrinsically heuristic and non-optimized approach. Non-structured pruning in \cite{12, 84} iteratively prunes weights at arbitrary location based on their magnitude, resulting in a sparse model to be stored in the compressed sparse column (CSC) format. Although this method achieves $12 \times$ pruning rate on LeNet-5 using MNIST dataset and $9 \times$ pruning rate on AlexNet using ImageNet dataset, it leads to an undermined processing throughput because the indices in the compressed weight representation cause stall or complex workload on highly parallel architectures. To overcome the limitation of the general non-structured weight pruning, recent works \cite{14, 66} proposed to incorporate regularity or “structures” in weight pruning, including filter pruning, channel pruning, and filter shape pruning. Structured pruning targets at generating regular and smaller weight matrices based on $\ell_1$ regularization to eliminate overhead of weight indices and achieve higher
acceleration in CPU/GPU executions. The weight matrix will maintain a full matrix but with reduced dimensions, and indices are no longer needed. As a result, it leads to much higher speedups. However, it suffers from notable accuracy loss due to the poor solution quality.

### 3.2.2 Dynamic regularization-based pruning

Flourished by [77][80][85-87] with the powerful ADMM [82] optimization framework, pruning problems are re-formed into optimization problems with the dynamically updated regularization terms bounded by the designated constraint (i.e., pruning with specific dimensions or with any desired weight matrix shapes) sets that involved into loss function during DNN training. As demonstrated in the previous chapter, the ADMM approach decomposes an original pruning problem into two subproblems, in which one can be solved by standard stochastic gradient decent (SGD) while the other one can be solved by iteratively updating regularization terms with ADMM steps, thus facilitates very high weight reduction and promising accuracy. Same as static regularization-based pruning, the dynamic regularization-based pruning also achieves non-structured sparsity in [77][85] and structured sparsity in [80][87], but with significantly improved accuracy and compression results. In addition, it also achieves special pruning dimension such as hybrid sparsity studied in [86] which can not be obtained by former. ADMM can effectively deal with a subset of combinatorial constraints and yields high quality solutions, which associated constraints in the DNN weight pruning belong to this subset of combinatorial constraints, making ADMM applicable to DNN mode compression and achieves non-structured $246 \times$ compression on LeNet-5 with MNIST, $36 \times$ on AlexNet with ImageNet and structured $4.8 \times$ on AlexNet and $2 \times$ on ResNet-18 with ImageNet.

### 3.3 Problem statement

We observe that many early works use static methods on the weight pruning of DNNs, e.g. the magnitude based methods in [12] and the $\ell_1$ regularization method in [14]. We propose two hypotheses based on these methods. First, some weights with small magnitude are critical to maintain
the accuracy of the model, thus we cannot prune the weights simply based on their magnitude. Second, the $\ell_1$ norm is not a good approximation of the $\ell_0$ norm, and using the $\ell_1$ regularization will penalize some critical weights to values close to zero. These two hypotheses motivate us to find a better approximation for the $\ell_0$ norm in order to generate highly sparse model without accuracy loss.

We prune an AlexNet using the reweighted method to verify our two hypotheses. Fig. 3.1 shows the histogram of the weights in the second fully connected layer (FC-2) of AlexNet. In Fig. 3.1(a), the red area is the histogram of the original weight distribution without pruning (we omit the top part of the distribution due to space limitations), and the blue area is the distribution after removing 97.9% of the weights by the reweighted method without accuracy loss. We can observe that the critical weights (remaining weights after pruning) are approximately uniformly distributed, which means that some weights with small magnitude are also critical. This verifies our first hypothesis. In Fig. 3.1(b), the red area is the histogram of the weights after regularizing the FC-2 layer using $\ell_1$
regularization. Comparing with the red area of Fig. 3.1(a), $\ell_1$ regularization reduces the magnitude of the weights in the entire network. The critical weights, shown in the blue area of Fig. 3.1(a), have a different distribution after $\ell_1$ regularization is applied, as shown in the blue area of Fig. 3.1(b). It is clear that $\ell_1$ regularization penalizes a lot more critical weights towards zero, as is shown in the high peak of the blue area. After pruning, those critical weights will be forced to zero, and thus may negatively impact the model quality. This is because $\ell_1$ regularization casts equal penalty on all weights or weight groups. This violates the original intention of weight pruning, which is to remove the “non-critical” weights instead of “small” weights, thus it is not a good approximation for $\ell_0$ regularization, which verifies our second hypothesis.

Besides static methods used in the early works, a recent work [77] focuses on $\ell_0$ norm based optimization with an ADMM-based hard constraint approach and achieves good performance on DNN pruning without accuracy loss. This method first formulates weight pruning as an optimization problem with a hard constraint on $\ell_0$ norm, and then uses ADMM [82] to solve the problem. In the ADMM-based solution framework, the regularization term is dynamically updated in each iteration, and it achieves better performance compared with the work based on static methods. However, because of the hard constraint on the $\ell_0$ norm, the degree of sparsity in each DNN layer needs to be pre-specified. This fact limits the flexibility of the ADMM-based method. In reality, when the degree of sparsity undefined, it is hard to determine the numbers of weights to prune for each layer. Therefore, ADMM-based method may take an excessive amount of time to tune the parameters to achieve the desired pruning rate without accuracy loss.

In this paper, we propose to use a reweighted method for DNN weight pruning. It is a dynamic regularization-based method, where in each iteration the penalties on different weights are dynamically updated. Different from the ADMM-based method in which the hyperparameters need to be tuned, we only need to set a single penalty parameter in our method, the value of this parameter is easy to set and we will discuss it in Section 3.5. After training with our reweighted regularization method, we can decide the degree of sparsity in each layer based on the distribution of weights. For example, the weight distribution of each layer in AlexNet after our reweighted $\ell_1$ regularization
method is shown in Fig. 3.2. We can observe that most of the weights with large magnitude are distributed in the range of 0.01 to 0.1, and most of the weights with small magnitude are smaller than 0.0001. This means small weights are 100× or more smaller than large weights. Thus, removing the weights with magnitude smaller than 0.0001 has a minor effect on the accuracy of the DNN. Meanwhile, we do not need to specify the pruning rate for each layer as it will be determined dynamically by our proposed reweighted regularization method. Note that our method is essentially different from magnitude-based method. Magnitude-based method directly removes the part of weights with small magnitude. While in our method, the weights are divided into two parts based on the magnitude after the DNN is trained with reweighted regularization method. And one part of the weights is significantly smaller than the other part. We only remove the part of significantly small weights which have minor contribution for the DNN. However, some of the small weights removed by magnitude-based method have critical contribution to the DNN.

### 3.3.1 Non-structured pruning

Consider an $N$-layer DNN, the weights and biases of the $i$-th layer is respectively denoted by $W_i$ and $b_i$, and the collection of weights and biases of all the layers is respectively denoted by $\{W_i\}_{i=1}^{N}$ and $\{b_i\}_{i=1}^{N}$. The loss function associated with the DNN is denoted by $f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N})$. In DNN training, we minimize the loss function to increase the accuracy. For the non-structured weight pruning problem, our aim is to reduce the number of non-zero elements in the weight matrix while maintaining the accuracy. Therefore, we need to minimize the summation of the loss function and the $\ell_0$ regularization term as follows,

$$\text{minimize} \ f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N}) + \lambda \sum_{i=1}^{N} \|W_i\|_{\ell_0},$$

where $\lambda$ is the penalty parameter to adjust the relative importance of accuracy and sparsity.

The problem with the $\ell_0$ norm is intractable, thus we use a reweighted $\ell_1$ method [83] to approximate the $\ell_0$ norm. For the reweighted $\ell_1$ method, we instead solve the problem

$$\text{minimize} \ f(\{W_i\}_{i=1}^{N}, \{b_i\}_{i=1}^{N}) + \lambda \sum_{i=1}^{N} \|P_{\frac{d}{2}} \circ W_i\|_{\ell_1}, \quad (3.1)$$
where \( R(P_i^{(l)}, W_i) = \| P_i^{(l)} \circ W_i \|_{\ell_1} \), the operator \( \circ \) denotes element-wise multiplication, and \( P_i^{(l)} \) is the collection of penalties on different weights, which is updated in every iteration to increase the degree of sparsity beyond the \( \ell_1 \) norm regularization. In each iteration, we denote the solution of \( W_i \) by \( W_i^{(l)} \) and update \( P_i \) by setting \( P_i^{(l+1)} = \frac{1}{|W_i^{(l)}| + \epsilon} \), where \( | \cdot | \) denotes the absolute value and \( \epsilon \) is a small parameter to avoid dividing by zero. In our experiment \( \epsilon = 0.001 \) works well. All operations above are performed element-wise.

### 3.3.2 Structured pruning

Filter-wise pruning and shape-wise pruning are a subset of structured pruning. Different from non-structured pruning, structured pruning requires not only the sparsity in weights but also the position of the zeros elements [14]. To generate different kinds of group sparsity, we propose to use the reweighted method on the group lasso regularization [88]. Problem (3.1) is also applicable to structured pruning. For filter-wise pruning, the regularization term is

\[
R(P_i^{(l)}, W_i) = \sum_{a=1}^{A} \| P_i^{(l)} \circ (W_i)_{a, :, :, :} \|_F^2,
\]

where \((W_i)_{a, :, :, :}\) denotes the \( a \)-th filter of \( W_i \), and \( P_i^{(l+1)} \) is updated by \( P_i^{(l+1)} = \frac{1}{|W_i|_{a, :, :, :}^F + \epsilon} \).

For shape-wise pruning, the regularization term is

\[
R(P_i^{(l)}, W_i) = \sum_{b=1}^{B} \sum_{c=1}^{C} \sum_{d=1}^{D} \| P_i^{(l)} \circ (W_i)_{:, b, c, d} \|_F^2,
\]

where \((W_i)_{:, b, c, d}\) denotes the collection of weights located at position \(( :, b, c, d)\) in every filter and \( P_i \) is updated by \( P_{i, b, c, d}^{(l+1)} = \frac{1}{|W_i|_{b, c, d}^F + \epsilon} \).

### 3.4 A unified algorithm for non-structured and structured sparsity

In [83], the reweighted \( \ell_1 \) method initializes all the penalties on different weights to one. In our problem, since we have pretrained models, we initialize \( P_i \) using the parameters \( W_i \) in the pre-
**Algorithm 2** Unified reweighted method on DNN pruning

**Input:** pretrained model

Initialize $P_i$

Set $l = 1$

Set $T$ as the number of iterations of the reweighted method

for $l ≤ T$ do

Solve the regularization Problem (3.1) using SGD or ADAM

Update $P_{i}^{(l+1)}$ using the solution of $W_{i}^{(l)}$

end for

Remove the weights (or group of weights) which are close to zeros and retrain the DNN using the non-zero weights

Then we mask the gradients of the weights we already set to zeros (these zero weights will no longer change), and use reweighted method to generate further sparsity based on the model found in the first step. We observe that after the first step, we obtain a sparse model with sparsity larger than state-of-the-art works [77, 85]. However, we can further increase the degree of sparsity by using the reweighted method repeatedly.

Since the loss function in the regularization problem is non-convex, we cannot find the globally optimum of this problem. This impacts the performance of the reweighted method to search for a model with high degree of sparsity in a single step. However, if we use the single step repeatedly, we can keep the balance between the degree of sparsity and accuracy of the model in each step. Then we can finally achieve a highly sparse model without accuracy loss. More specifically, we use a moderate $\lambda$ and apply the reweighted method for several steps, and thus obtain a highly sparse trained model. We use SGD or ADAM [34] to solve the regularization problem (3.1). We set the parameters of the pretrained model as the starting point of the first iteration of the reweighted method, and we set the solution obtained after one iteration of the reweighted method as the starting point of the next iteration. The above approaches we used (the way to initialize $P_i$ and set starting point) can reduce the total computational time in the reweighted method. After using the reweighted method, we remove the weights (or group of weights) that are close to zero and retrain the DNN using the remaining non-zero weights. Algorithm 2 summarizes a single step of our proposed method.
model with competitive accuracy.

3.5 Numerical results

In this section, we evaluate our proposed framework for both non-structured pruning and structured pruning on different DNN models. We implement our non-structured pruning method on AlexNet [3] models for ImageNet ILSVRC-2012 dataset. We also implement our structured pruning method on ResNet-18 and ResNet-50 [5] models for ImageNet dataset, and MobileNet-V2-1.0 [89] model for CIFAR-10 dataset.

For the penalty parameter $\lambda$, it is used for adjusting the relative importance of accuracy and sparsity. Excessively small $\lambda$ fails to regularize enough non-critical weights to values close to zero and excessively large $\lambda$ is fails to minimize the loss function and the model accuracy cannot be maintained. In our experiment we find an appropriate way to tune $\lambda$. In the regularization problems (3.1), when we adjust $\lambda$ to set the value of the regularization term in the range of

$$4l \leq \lambda \sum_{i=1}^{N} R(P_i^{(1)}, W_i^{(0)}) \leq 8l,$$

we achieve good pruning rate without accuracy loss. Where $W_i^{(0)}$ is the weights in pretrained model, and $P_i^{(1)}$ is derived by $W_i^{(0)}$, and $l$ is the training loss of the pretrained model.

3.5.1 Non-structured pruning on LeNet-5 & AlexNet

LeNet-5 on MNIST

We first evaluate the performance of our non-structured pruning method on LeNet-5 model using MNIST dataset. The comparisons of our method and previous methods are shown in Table 3.1. We achieve $630 \times$ pruning rate with 99.0% accuracy and $301 \times$ pruning rate with 99.2% accuracy.
Table 3.1: Comparisons of overall weight pruning results on LeNet-5 using MNIST data set.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>Pruning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>99.2%</td>
<td>1×</td>
</tr>
<tr>
<td>Iterative Pruning [12]</td>
<td>99.2%</td>
<td>12.5×</td>
</tr>
<tr>
<td>One-step ADMM [77]</td>
<td>99.2%</td>
<td>71.2×</td>
</tr>
<tr>
<td>Optimal Brain Surgery [39]</td>
<td>98.3%</td>
<td>111×</td>
</tr>
<tr>
<td>Progressive ADMM [85]</td>
<td>99.2%</td>
<td>200×</td>
</tr>
<tr>
<td><strong>Our method</strong></td>
<td>99.2%</td>
<td>301×</td>
</tr>
</tbody>
</table>

Table 3.2: Comparisons of overall non-structured weight pruning results on AlexNet model for ImageNet dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-5 accuracy</th>
<th>Accuracy loss</th>
<th>Pruning rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>80.2%/82.4%</td>
<td>0.0%</td>
<td>1×</td>
</tr>
<tr>
<td>Iterative Pruning [12]</td>
<td>80.3%</td>
<td>−0.1%</td>
<td>9.1×</td>
</tr>
<tr>
<td>Optimal Brain Surgery [39]</td>
<td>80.0%</td>
<td>0.2%</td>
<td>17.7×</td>
</tr>
<tr>
<td>Hoyer-Square [67]</td>
<td>80.2%</td>
<td>0.0%</td>
<td>21.3×</td>
</tr>
<tr>
<td>One-step ADMM [77]</td>
<td>80.2%</td>
<td>0.0%</td>
<td>21×</td>
</tr>
<tr>
<td>Progressive ADMM [85]</td>
<td>82.0%</td>
<td>0.4%</td>
<td>36×</td>
</tr>
<tr>
<td><strong>Our method (one-step)</strong></td>
<td><strong>82.0%</strong></td>
<td><strong>0.4%</strong></td>
<td><strong>37×</strong></td>
</tr>
<tr>
<td><strong>Our method</strong></td>
<td><strong>82.3%</strong></td>
<td><strong>0.1%</strong></td>
<td><strong>40×</strong></td>
</tr>
<tr>
<td><strong>Our method</strong></td>
<td><strong>82.0%</strong></td>
<td><strong>0.4%</strong></td>
<td><strong>45×</strong></td>
</tr>
</tbody>
</table>

AlexNet on ImageNet.

Table 3.2 shows the non-structured pruning results. In order to highlight the difference of the obtained accuracy by using different pruning methods, we use the relative accuracy loss against the baseline accuracy of each method. Note that the pruning rates of early works are less than or around 20×. A recent work [85] achieves 36× pruning rate with 82.0% top-5 accuracy. We use the same baseline as [85] and achieves 45× pruning rate with 82.0% top-5 accuracy.
Table 3.3: Structured pruning results on ResNet-18 for ImageNet dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Sparsity type</th>
<th>Pruning rate</th>
<th>Top-1/Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>N/A</td>
<td>1.0×</td>
<td>69.6%/89.0%</td>
</tr>
<tr>
<td>DCP [66]</td>
<td>filter &amp; shape</td>
<td>1.5×</td>
<td>69.2%/89.0%</td>
</tr>
<tr>
<td>Channel Gating [90]</td>
<td>filter &amp; shape</td>
<td>1.9×</td>
<td>68.8%/ N/A</td>
</tr>
<tr>
<td><strong>Our method</strong></td>
<td>shape</td>
<td><strong>3.0×</strong></td>
<td>69.2%/89.0%</td>
</tr>
<tr>
<td>Our method</td>
<td>filter &amp; shape</td>
<td><strong>4.2×</strong></td>
<td>68.7%/88.5%</td>
</tr>
</tbody>
</table>

Table 3.4: Structured pruning results on ResNet-50 for ImageNet dataset.

<table>
<thead>
<tr>
<th>Method</th>
<th>Sparsity type</th>
<th>Pruning rate</th>
<th>Top-1/Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>N/A</td>
<td>1.0×</td>
<td>75.7%/92.7%</td>
</tr>
<tr>
<td>Geometric Median [43]</td>
<td>filter &amp; shape</td>
<td>1.7×</td>
<td>74.8%/92.3%</td>
</tr>
<tr>
<td>DCP [66]</td>
<td>filter &amp; shape</td>
<td>2.0×</td>
<td>75.1%/92.3%</td>
</tr>
<tr>
<td>CNN-FCF [69]</td>
<td>filter &amp; shape</td>
<td>2.1×</td>
<td>74.6%/92.2%</td>
</tr>
<tr>
<td>AutoPrune [81]</td>
<td>filter &amp; shape</td>
<td>2.2×</td>
<td>74.5%/ N/A</td>
</tr>
<tr>
<td>Struct-ADMM [80]</td>
<td>filter &amp; shape</td>
<td>2.7×</td>
<td>N/A /92.3%</td>
</tr>
<tr>
<td><strong>Our method</strong></td>
<td>filter &amp; shape</td>
<td><strong>3.2×</strong></td>
<td>75.0%/92.3%</td>
</tr>
</tbody>
</table>

3.5.2 Structured pruning on ResNet-18, ResNet-50 and MobileNet-V2-1.0

ResNet-18 on ImageNet.

Table 3.3 shows the structured pruning results. DCP [66] only achieves 1.5× pruning rate without accuracy loss. Channel Gating [90] method achieves 1.9× pruning rate with minor accuracy loss. In our proposed method, we achieve 3.0× pruning rate without accuracy loss. We also implement filter-wise sparsity together with shape-wise sparsity on ResNet-18, and totally achieve 4.2× pruning rate with 0.5% accuracy loss.

Table 3.5: Structured pruning results on MobileNet-V2-1.0 for CIFAR-10 dataset

<table>
<thead>
<tr>
<th>Method</th>
<th>Sparsity type</th>
<th>Conv. pruning rate</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>N/A</td>
<td>1.0×</td>
<td>94.5%</td>
</tr>
<tr>
<td>DCP [66]</td>
<td>filter &amp; shape</td>
<td>1.4×</td>
<td>94.7%</td>
</tr>
<tr>
<td><strong>Our method</strong></td>
<td>filter &amp; shape</td>
<td><strong>7.2×</strong></td>
<td>94.6%</td>
</tr>
</tbody>
</table>
**ResNet-50 on ImageNet.**

Table 3.4 shows the structured pruning results. The recent works Geometric Median [43], Struct-ADMM [80], CNN-FCF [69] and DCP [66] achieve $1.7 \times$ to $2.2 \times$ pruning rate with minor or no accuracy loss. In our proposed method, we achieve $3.2 \times$ pruning rate with 92.3% Top-5 accuracy.

**MobileNet-V2-1.0 on CIFAR-10.**

We demonstrate the results of our structured pruning method on MobileNet-V2-1.0 for CIFAR-10 dataset in Table 3.5. We achieve $7.2 \times$ pruning rate without accuracy loss, which is significantly higher than $1.4 \times$ in DCP [66].

Overall, using the same training trails, our method can achieve higher pruning rate than the prior works. For small to large-scale dataset, our proposed method significantly outperforms the state-of-the-art in terms of pruning rate and accuracy, leading to light weight storage and computation.

### 3.6 Discussion on inference acceleration

The computations in the convolutional layers of DNNs are usually transformed to matrix multiplications by converting the weight tensors and feature map tensors to matrices [72], which is called general matrix multiplication (GEMM). In GEMM, filter-wise pruning corresponds to row pruning, and shape-wise pruning corresponds to column pruning. Combining filter-wise and shape-wise sparsity can directly reduce the dimension of weight matrix in GEMM by removing zero rows and columns, which can achieve notable speedup on CPU and GPU platforms. In this paper, we achieve higher overall pruning rate for the combination of filter-wise and shape-wise pruning compared with prior works, e.g. we achieve $3.2 \times$ structured pruning rate on the convolutional layers of ResNet-50 for ImageNet dataset, which is 45% higher than $2.2 \times$ in [81]. The sparse model with higher structured pruning rate can achieve higher speedup on CPU and GPU platforms.
3.7 Summary

In this paper, we propose a unified DNN weight pruning framework with dynamically updated regularization terms bounded by the designated constraint, which can generate both non-structured sparsity and different kinds of structured sparsity. In our proposed framework, we first use reweighted method to regularize the model, then remove the weights which are close to zero and mask the gradient of these weights to ensure that they no longer update, and we retrain the remaining non-zero weights to retrieve the accuracy. Experimental results demonstrate that we achieve higher pruning rate than state-of-the-arts for both non-structured and structured pruning with negligible accuracy degradation.
CHAPTER 4

GENERATION OF LOW DISTORTION
ADVERSARIAL ATTACKS VIA CONVEX
PROGRAMMING

4.1 Introduction

Deep neural networks (DNNs) continue to show extraordinary performance in a variety of tasks, such as image recognition [3–5], speech recognition [6,7], and natural language processing [91]. However, recent research shows that DNNs are vulnerable to adversarial attacks [23,92]. Adversarial attacks, also known as adversarial examples, are generated by incorporating imperceptible perturbations into the original input data in order to mislead the prediction of DNNs [93,94].

Research on the robustness of DNNs follows two directions in general. The first is to enhance the robustness of DNNs, in order to increase the degree of difficulty for adversarial attacks to fool DNNs [19–22]. The second is to design adversarial attack methods to test the robustness of DNNs [23–27]. These two aspects reciprocally benefit each other towards hardening DNNs, and our research in this chapter belongs to the latter one.

Adversarial attacks can be either untargeted or targeted. In untargeted attacks, adversarial
examples are generated to fool DNNs’ prediction towards a label other than the correct one [95].

In targeted attacks, adversarial examples are designed to force the DNNs to classify the data with a desired incorrect target label [23]. In this chapter, we focus on the problem of targeted attack generation, as such attacks are commonly regarded as being stronger [26].

Despite the fact that the loss functions of DNNs are non-convex, most adversarial attack generation problems in the literature are solved by gradient descent; for example [34] solves the C&W attack problem via ADAM. Recent papers on certifying the robustness of DNNs employ relaxations to formulate convex optimization problems [96, 97].

In contrast to these methods, in this chapter we first formulate the adversarial attack generation problem as one with a convex objective function but non-convex constraints. We then design an algorithm which iteratively solves a related convex problem. We prove that upon convergence of our iterative algorithm, the obtained solution is feasible for the original (non-convex) problem. We achieve 100% attack success rate on both the original undefended models and the adversarially-trained models. Our distortions of the $\ell_\infty$ attack are respectively 31% and 18% lower than the C&W attack for the best case and average case on the CIFAR-10 data set.

4.2 Related work

4.2.1 Gradient descent based attack methods

**L-BFGS attack [23]**: The L-BFGS attack is the first attack based on optimization. It aims to minimize the cross-entropy loss of the adversarial example and the target label, while minimizing the $\ell_2$ distortion of the adversarial example and original data.

**FGM attack [24] & IFGM attack [92]**: The fast gradient method (FGM) attack uses the gradient of the loss function to find the direction in which the intensity of pixels should be changed. It is an attack that is designed to be fast rather than to pursue low distortion in the original data. The iterative fast gradient method (IFGM) attack is a refinement of the FGM attack which takes multiple smaller steps instead of a single step on gradient descent.
C&W attack [26]: Based on the basic ideas of L-BFGS attack, C&W attack design their own objective functions instead of cross-entropy loss, which help them achieve 100% attack successful rate. Besides on $\ell_2$ attack, C&W also design iterative methods for the $\ell_0$ and $\ell_\infty$ attack, in which the objective functions are non-differentiable. C&W attack is state-of-the-art in the adversarial attacks on DNNs.

4.2.2 Related work on convex programming and mixed integer linear programming

Robustness certification of DNNs: Recently, convex optimization methods have been used to certify the robustness of DNNs rather than to generate adversarial attacks. Examples include the use of linear programming in [96], quadratic programming in [97], and semidefinite programming in [98].

Binarized neural networks attack: The paper [99] presents a new method based on mixed integer linear programming to attack binarized neural networks. The generation of low distortion attacks on binarized neural networks is a non-convex problem, where the binary nature of the activation functions is responsible for the lack of convexity. The authors use the property that the output of every layer is composed of zeros and ones to translate the lack of convexity into binary constraints. This presents a special case in which the non-convex problem can be solved by mixed integer linear programming.

4.2.3 Representative defense method

Defensive distillation [25]: Defensive distillation uses distillation for the purpose of improving the robustness of a neural network. In the defensive distillation method, we need to train a teacher network model at “high temperature” at first and then employ the teacher network to produce soft labels for the training data set. Later, the created soft labels are used to train a distilled model. Finally, we reduce the temperature to low values when we test the accuracy of the distilled model.
Adversarial training [100]: In adversarial training, adversarial examples with correct labels are mixed into the training data set. The neural network is then retrained to increase its robustness.

4.3 Problem statement

Consider an \( N \)-layer DNN, where the weights and biases in the \( i \)-th layer are respectively denoted by \( W_i \) and \( b_i \), and that all the layers in the DNN are fully connected. Assume that \( x_0 \) is a vector representation of an image in the test set, and that \( x \) is an adversarial example that we wish to generate. The example \( x \) has the property that it is a small perturbation of \( x_0 \) but is classified as belonging to the incorrect target class \( t \) by the DNN.

The output of the first layer of the DNN to input \( x \) is

\[
y_1 = \sigma(W_1x + b_1).
\]

Here, \( y_1 \) and \( b_1 \) are vectors, and \( \sigma(\cdot) \) is the non-linear activation function which acts elementwise on its vector argument. This function is generally chosen to be the ReLU function [70] in state-of-the-art DNNs, which is defined as

\[
\sigma(\tau) = \begin{cases} 
\tau & \text{if } \tau \geq 0, \\
0 & \text{if } \tau < 0.
\end{cases}
\]

In a DNN the output of one layer is the input to the next, and thus the output of the \( i \)-th layer for \( i = 2, \ldots, N - 1 \) is

\[
y_i = \sigma(W_iy_{i-1} + b_i).
\]

The output before the softmax function (the collection of logits) is

\[
z = W_Ny_{N-1} + b_N.
\]
The logits are input into the softmax function to calculate the scores of different classes. The class with the highest score will determine the classification made by the DNN. Since the softmax function is an increasing function, the class with the highest logit will achieve the highest score and become the classification result. For a targeted adversarial attack, the target class \( t \) should have the highest logit, which means

\[
(z)_t = \max(z),
\]

where \( (z)_t \) is the \( t \)-th element in the vector \( z \). The above equation can be equivalently rewritten as

\[
z \leq (z)_t \mathbb{1},
\]

where \( \mathbb{1} \) is the column vector of all ones. The above inequality ensures the success of the targeted attack. To ensure that \( x \) is an imperceptible perturbation of \( x_0 \) we minimize the \( L_p \) distortion between the adversarial example and the original data. Namely, we minimize

\[
\| x - x_0 \|_p,
\]

which is a convex function of \( x \) for \( p \geq 1 \). Also, to ensure the adversarial example yields a valid image we impose the constraint

\[
0 \leq x \leq 1.
\]

We can now formulate the adversarial attack problem as

\[
\begin{align*}
\text{minimize} & \quad \| x - x_0 \|_p \\
\text{subject to} & \quad y_1 = \sigma(W_1 x + b_1) \\
& \quad y_i = \sigma(W_i y_{i-1} + b_i), \quad i = 2, \ldots, N - 1 \\
& \quad z = W_N y_{N-1} + b_N \\
& \quad z \leq z_t \mathbb{1}, \quad 0 \leq x \leq 1.
\end{align*}
\]
This optimization problem has a convex objective and convex inequality constraints. However, \( \sigma(\cdot) \) is a nonlinear function which renders the equality constraints, and therefore the optimization problem as a whole, non-convex.

### 4.4 Problem relaxation and proposed algorithms

In this section, we propose an algorithm which iteratively solves a convex relaxation of (4.1) to obtain an approximate solution. This approximate solution is feasible in the sense that it satisfies all the constraints in (4.1).

Since \( \sigma(\cdot) \) acts elementwise on its argument, we can consider the effect of \( \sigma(\cdot) \) as an elementwise multiplication of the input vector with a binary vector \( a_i \) whose elements are zero/one based on the sign of the elements of the vectors \( W_1 x + b_1 \) and \( W_i y_{i-1} + b_i \),

\[
\begin{align*}
y_1 &= a_1 \circ (W_1 x + b_1), \\
y_i &= a_i \circ (W_i y_{i-1} + b_i), \quad i = 2, \ldots, N - 1,
\end{align*}
\]

where \( \circ \) denotes elementwise vector multiplication.

Due to the dependence of \( a_i \) on the sign of \( W_i y_{i-1} + b_i \), the equality constraint \( y_i = a_i \circ (W_i y_{i-1} + b_i) \) is still non-convex. We break this dependence by using an iterative procedure in which the sign of \( W_i y_{i-1} + b_i \), computed from the solution of the previous iteration, is used to form \( a_i \) in the current iteration. Concretely, rather than solve the non-convex problem (4.1), we
solve for $k = 0, 1, \ldots, T$ the convex problem

$$
\begin{align*}
\text{minimize} & \quad \|x - x_0\|_p + \lambda\|x - x^{(k)}\|_2 \\
\text{subject to} & \quad y_1 = a_1^{(k)} \circ (W_1x + b_1) \\
& \quad y_i = a_i^{(k)} \circ (W_iy_{i-1} + b_i), \quad i = 2, \ldots, N - 1 \\
& \quad z = W_Ny_{N-1} + b_N \\
& \quad z \leq z_t 1, \quad 0 \leq x \leq 1.
\end{align*}
$$

(4.2)

We denote by $x^{(k+1)}$, $y_i^{(k+1)}$, $z^{(k+1)}$ the solution of problem (4.2) at iteration $k$, and let $x^{(0)} = x_0$. We set the value of $a_1^{(k)}$ according to

$$
(a_1^{(k)})_j = \begin{cases} 
0 & \text{if } (W_1x^{(k)} + b_1)_j < 0, \\
1 & \text{if } (W_1x^{(k)} + b_1)_j \geq 0,
\end{cases}
$$

(4.3)

where $(v)_j$ denotes the $j$th element of the vector $v$ and $j$ takes all values between one and the dimension of the vector $a_1$. We employ a special procedure to compute the values of $a_i^{(k)}$. Rather than use $y_i^{(k)}$ from the previous iteration, we propagate forward through the layers the value $x^{(k)}$ of $x$ from the previous iteration and denote the resulting values by $y_i^{[k]}$. To make this precise, we find $a_1^{(k)}$ from (4.3) and set $y_1^{[k]} = a_1^{(k)} \circ (W_1x^{(k)} + b_1)$. We then find $a_2^{(k)}$ from

$$
(a_2^{(k)})_j = \begin{cases} 
0 & \text{if } (W_2y_1^{[k]} + b_2)_j < 0, \\
1 & \text{if } (W_2y_1^{[k]} + b_2)_j \geq 0,
\end{cases}
$$

and set $y_2^{[k]} = a_2^{(k)} \circ (W_2y_1^{[k]} + b_2)$. We continue this procedure so that for $i = 2, \ldots, N - 1,$

$$
(a_i^{(k)})_j = \begin{cases} 
0 & \text{if } (W_iy_{i-1}^{[k]} + b_i)_j < 0, \\
1 & \text{if } (W_iy_{i-1}^{[k]} + b_i)_j \geq 0,
\end{cases}
$$

(4.4)
We emphasize that problem (4.2) is convex and can therefore be solved efficiently using convex optimization tools.

This motivates Algorithm 3. We iteratively solve (4.2), using (4.3) and (4.4) to update $a_i^{(k+1)}$, until the condition $a_i^{(k+1)} = a_i^{(k)}$, $i = 1, \ldots, N - 1$ is satisfied. We initialize the algorithm by setting $x^{(0)} = x_0$.

**Proposition 1.** If for some $k$ we have

$$a_i^{(k+1)} = a_i^{(k)}, \quad i = 1, \ldots, N - 1, \quad (4.5)$$

then the solution $x$ of (4.2), denoted by $x^{(k+1)}$, is a feasible solution of problem (4.1).

**PROOF:** See Appendix A.1. 

The parameter $\lambda$ characterizes the relative importance of the two terms in the objective function of (4.2): A small value of $\lambda$ de-emphasizes the second norm, which results in a solution with lower distortion and therefore better performance; a large value of $\lambda$ emphasizes the second norm, which helps achieve convergence (at the expense of performance) when (4.2) is solved iteratively by penalizing the difference of the optimal $x$ between two consecutive iterations.

This motivates Algorithm 4. We choose a small value of $\lambda$ and check the convergence of Algorithm 3. If convergence, as determined by the satisfaction of condition (4.5), is not achieved then we increase the value of $\lambda$ and apply Algorithm 3 again; if convergence is achieved then we have found a value of $\lambda$ that results in a feasible solution. The advantage of this process is that when $\lambda$ is small, the optimization problem (4.2) is allowed to explore the $x$-space for a solution with small distortion. Therefore, our aim is to find the smallest value of $\lambda$ that results in convergence (in our experiments such a value of $\lambda$ could always be found); we refer to this value as $\hat{\lambda}$, and refer to the solution of Algorithm 3 with $\lambda = \hat{\lambda}$ as $\hat{x}$.

**Remark:** Once $\hat{\lambda}$ is obtained, we may explore whether solutions with lower distortion than $\hat{x}$ can be found as follows: We start from $\lambda = \hat{\lambda}$ and apply Algorithm 3 with the important difference that rather than setting $x^{(0)} = x_0$ we take $x^{(0)} = \hat{x}$. We then iteratively reapply Algorithm 3.
\textbf{Algorithm 3} Find approximate solution of (4.1) by iteratively solving (4.2)

\textbf{Input:} image $x_0$, weights $W_i$, biases $b_i$, parameter $\lambda$

Set $x^{(0)} = x_0$
Calculate $a_i^{(0)}$ according to (4.3) and (4.4)
Set $k = 0$
\textbf{for} $k \leq T$ \textbf{do}
\hspace{1em} Solve problem (4.2) to obtain $x^{(k+1)}$
\hspace{1em} Update $a_i^{(k+1)}$ according to (4.3) and (4.4)
\hspace{1em} \textbf{if} Condition (4.5) is satisfied \textbf{then}
\hspace{2em} Break for loop
\hspace{1em} \textbf{end if}
\hspace{1em} Set $k = k + 1$
\textbf{end for}

\textbf{Algorithm 4} Iterative method to guarantee convergence of Algorithm 3

\textbf{Input:} image $x_0$, weights $W_i$, biases $b_i$

Set parameter $\lambda$
\textbf{repeat}
\hspace{1em} Apply Algorithm 3
\hspace{1em} \textbf{if} Condition (4.5) is not satisfied \textbf{then}
\hspace{2em} Increase value of $\lambda$
\hspace{1em} \textbf{end if}
\textbf{until} Condition (4.5) is satisfied

Set $\hat{x}$ to solution of Algorithm 3
Set $\hat{\lambda} = \lambda$

we decrease $\lambda$ if convergence is achieved and otherwise increase $\lambda$, each time setting $x^{(0)}$ to be the solution of Algorithm 3 from the previous iteration. In our experiments we find that applying this methods for several iterations usually helps us find a solution with lower distortion than just applying Algorithms 3 and 4.

\subsection{4.5 Numerical results}

We compare our proposed method with the IFGM attack \cite{92} and the C&W attack \cite{92}, in which the C&W attack is state-of-the-art adversarial attack on DNNs. In the C&W attack, the authors proposed their method for $\ell_0$, $\ell_2$ and $\ell_\infty$ attacks, since $\ell_0$ norm is non-convex, it is not applicable for convex programming. Thus we compare our $\ell_2$ and $\ell_\infty$ attacks with other two works. Our ex-
experimental results demonstrate that the adversarial examples generated by our method have lower distortion than the IFGM attack and the C&W attack on the MNIST [4] and CIFAR-10 [101] data sets.

4.5.1 Experiment setup

We evaluate the performance of different attack methods on the LeNet-300-100 [4]. In this network, the number of neurons in the two hidden layers are 300 and 100, respectively. The activation functions after the hidden layers are chosen to be ReLU. The test accuracy of LeNet-300-100 on the MNIST and CIFAR-10 data sets are around 98% and 57%, respectively. In our proposed algorithm, we solve the convex problem by CVXPY [102, 103], which is a tool for convex programming in Python.

4.5.2 Attack success rate and distortion for the $\ell_2$ attack

We test the $\ell_2$ attack of our proposed method, the IFGM attack method and the C&W attack method on the first 500 images in the test sets of the MNIST and CIFAR-10 data sets. For every image we implement targeted attacks on its 9 incorrect labels. In the 4500 adversarial attacks in each data set, both of the methods achieve 100% attack success rate (ASR), and the $\ell_2$ distortion of different attack methods on CIFAR-10 are shown in Table 4.1.

In both of the data sets, the performance of our method and the C&W attack are much better than the IFGM attack. In the MNIST data set, our results are close to the C&W attack. While in the larger data set CIFAR-10, we achieve lower distortion than the C&W attack on both of the three cases.

4.5.3 Attack success rate and distortion for the $\ell_\infty$ attack

The data sets setup for the $\ell_\infty$ attack test is the same as the $\ell_2$ attack. The results of different $\ell_\infty$ attack methods are shown in Table 4.2.
Table 4.1: Comparisons of different $\ell_2$ attacks for MNIST and CIFAR-10 data sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Attack Method</th>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ASR $\ell_2$</td>
<td>ASR $\ell_2$</td>
<td>ASR $\ell_2$</td>
</tr>
<tr>
<td>MNIST</td>
<td>IFGM ($\ell_2$)</td>
<td>100 1.14</td>
<td>100 1.93</td>
<td>100 2.83</td>
</tr>
<tr>
<td></td>
<td>C&amp;W ($\ell_2$)</td>
<td>100 1.10</td>
<td>100 1.73</td>
<td>100 2.35</td>
</tr>
<tr>
<td></td>
<td>Convex Programming ($\ell_2$)</td>
<td>100 1.09</td>
<td>100 1.73</td>
<td>100 2.35</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>IFGM ($\ell_2$)</td>
<td>100 0.168</td>
<td>100 0.740</td>
<td>100 1.339</td>
</tr>
<tr>
<td></td>
<td>C&amp;W ($\ell_2$)</td>
<td>100 0.158</td>
<td>100 0.648</td>
<td>100 1.114</td>
</tr>
<tr>
<td></td>
<td>Convex Programming ($\ell_2$)</td>
<td>100 0.154</td>
<td>100 0.645</td>
<td>100 1.112</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of different $\ell_{\infty}$ attacks for MNIST and CIFAR-10 data sets

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Attack Method</th>
<th>Best Case</th>
<th>Average Case</th>
<th>Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ASR $\ell_{\infty}$</td>
<td>ASR $\ell_{\infty}$</td>
<td>ASR $\ell_{\infty}$</td>
</tr>
<tr>
<td>MNIST</td>
<td>IFGM ($\ell_{\infty}$)</td>
<td>100 0.081</td>
<td>100 0.134</td>
<td>100 0.197</td>
</tr>
<tr>
<td></td>
<td>C&amp;W ($\ell_{\infty}$)</td>
<td>100 0.076</td>
<td>100 0.117</td>
<td>100 0.156</td>
</tr>
<tr>
<td></td>
<td>Convex Programming ($\ell_{\infty}$)</td>
<td>100 0.074</td>
<td>100 0.114</td>
<td>100 0.152</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>IFGM ($\ell_{\infty}$)</td>
<td>100 0.0046</td>
<td>100 0.0198</td>
<td>100 0.0379</td>
</tr>
<tr>
<td></td>
<td>C&amp;W ($\ell_{\infty}$)</td>
<td>100 0.0051</td>
<td>100 0.0181</td>
<td>100 0.0299</td>
</tr>
<tr>
<td></td>
<td>Convex Programming ($\ell_{\infty}$)</td>
<td>100 0.0035</td>
<td>100 0.0149</td>
<td>100 0.0256</td>
</tr>
</tbody>
</table>

On the $\ell_{\infty}$ attack, we achieve a notable improvement compared with the results of the C&W attack. In the CIFAR-10 data set, we respectively reduce the $\ell_{\infty}$ distortion by 31% and 18% for the best case and average case compared with the C&W attack on the $\ell_{\infty}$ attack.

4.5.4 Attack against adversarial training

We test the performance of our method under adversarial training [100] using data augmentation, where we add 4500 adversarial examples with correct labels into the training data set. For these adversarially-trained models, we find that the adversarial examples generated by our $\ell_{\infty}$ method consistently have lower distortion compared with the adversarial examples generated by the C&W method. We elaborate on these results for the $\ell_{\infty}$ case below.

In our first group of experiments, we generate adversarial examples, perform adversarial training, and attack adversarially-trained models, all using the same method (i.e., our convex programming method versus the C&W method), and then compare their distortions against each other. On MNIST, the average distortion of our method is 0.124, which is lower than 0.129 for the C&W
method. On CIFAR-10, the average distortion of our method is 0.0231, which is 12% lower than 0.0262 for the C&W method.

In our second group of experiments, we implement adversarial training using adversarial examples generated by IFGM method, and then attack the adversarially-trained model using our method and the C&W method. On MNIST, the average distortion of our method is 0.126, which is lower than 0.132 for the C&W method. On CIFAR-10, the average distortion of our method is 0.0262, which is 11% lower than 0.0295 for the C&W method.

4.6 Summary

In this chapter, we propose an innovative method for generating adversarial examples via convex programming. Our method achieves a 100% attack success rate on both the original undefended models and the adversarially-trained models. We also decrease the distortion (on both original undefended models and adversarially-trained models) of adversarial examples compared with state-of-the-art attack methods.
CHAPTER 5
A UNIFIED MIN-MAX FRAMEWORK FOR
ADVERSARIAL EXPLORATION AND
ROBUSTNESS

5.1 Introduction

Training a machine learning model that is capable of assuring its worst-case performance against all possible adversaries given a specified threat model is a fundamental yet challenging problem, especially for deep neural networks (DNNs) [23, 24, 26]. A common practice to train an adversarially robust model is based on a specific form of min-max training, known as adversarial training (AT) [24, 104], where the minimization step learns model weights under the adversarial loss constructed at the maximization step in an alternative training fashion. In practice, AT has achieved the state-of-the-art defense performance against $\ell_p$-norm-ball input perturbations [105].

Although the min-max principle is widely used in AT and its variants [31, 104, 106, 107], few works have studied its power in attack generation. Thus, we ask: Beyond AT, can other types of min-max formulation and optimization techniques advance the research in both adversarial attack and robustness exploration? In this chapter, we give an affirmative answer corroborated by the
substantial performance gain and the ability of self-learned risk interpretation using our proposed min-max framework on several tasks for adversarial attack and defense.

We demonstrate the utility of a general formulation for minimizing the maximal loss induced from a set of risk sources (domains). Our considered min-max formulation is fundamentally different from AT, as our maximization step is taken over the probability simplex of the set of domains. Moreover, we show that many problem setups in adversarial attacks and defenses can in fact be reformulated under this general min-max framework, including attacking model ensembles [28,29], devising universal perturbation to input samples [30] and generalized AT over multiple types of threat models [31,32]. However, current methods for solving these tasks often rely on simple heuristics (e.g., uniform averaging), resulting in significant performance drops when compared to our proposed min-max optimization framework.

Contributions

(i) We indicate the utility of min-max optimization beyond AT by proposing a general and theoretically grounded framework on adversarial attack and defense. As a byproduct and an exclusive feature, by tracking the learnable weighting factors associated with multiple domains, our method can provide tools for self-adjusted importance assessment on the mixed learning tasks.

(ii) With the aid of min-max optimization, we propose a unified alternating one-step projected gradient descent (APGD) attack method, which can readily be specified to generate model ensemble attack, and universal attack over multiple images. In theory, we show that APGD has an $O(1/T)$ convergence rate, where $T$ is the number of iterations. In practice, we show that APGD obtains 17.48%, 35.21% and 9.39% improvement on average compared with PGD attack on CIFAR-10.

(iii) We propose a generalized AT scheme under mixed types of adversarial attacks and demonstrate that the diversified attack ensemble helps adversarial robustness. Compared with vanilla AT, our new training scheme leads to better worst-case robustness even if the defender lacks prior knowledge of the strengths of attacks.
5.2 Related work

Recent studies have identified that DNNs are highly vulnerable to adversarial manipulations in various applications [23, 94, 108–115], thus leading to an arms race between adversarial attacks [26, 27, 95, 105, 116–118] and defenses [25, 31, 104, 106, 107, 119–121]. One intriguing property of adversarial examples is the transferability across multiple domains [122–125], which indicates a more challenging yet promising research direction – devising universal adversarial perturbations over model ensembles [28, 29], and input samples [30, 126, 127]. However, current approaches suffer from a significant performance loss for resting on the uniform averaging strategy or heuristic weighting schemes [29, 127]. As a natural extension following min-max attack, we study the generalized AT under multiple perturbations [31, 32, 128, 129]. Finally, our min-max framework is adapted and inspired by previous literature on robust learning over multiple domains [130–133].

5.3 Preliminaries

Consider $K$ loss functions $\{F_i(v)\}$ (each of which is defined on a learning domain), the problem of robust learning over $K$ domains can be formulated as [130–132]

$$\min_{v \in V} \max_{w \in P} \sum_{i=1}^{K} w_i F_i(v),$$  \hspace{1cm} (5.1)

where $v$ and $w$ are optimization variables, $V$ is a constraint set, and $P$ denotes the probability simplex $P = \{w \mid 1^T w = 1, w_i \in [0, 1], \forall i\}$. Since the inner maximization problem in (5.1) is a linear function of $w$ over the probabilistic simplex, problem (5.1) is thus equivalent to

$$\min_{v \in V} \max_{i \in [K]} F_i(v),$$  \hspace{1cm} (5.2)

where $[K]$ denotes the integer set $\{1, 2, \ldots, K\}$. 
Benefit and challenge from (5.1). Compared to multi-task learning in a finite-sum formulation which minimizes $K$ losses on average, problem (5.1) provides consistently robust worst-case performance across all domains. This can be explained from the epigraph form of (5.2).

$$
\min_{v \in V, t} \max_{w \in P} \sum_{i=1}^{K} w_i F_i(v) - \frac{\gamma}{2} \|w - 1/K\|_2^2,
$$

(5.4)

where $t$ is an epigraph variable [134] that provides the $t$-level robustness at each domain.

In computation, the inner maximization problem of (5.1) always returns the one-hot value of $w$, namely, $w = e_i$, where $e_i$ is the $i$th standard basis vector, and $i = \arg \max_i \{F_i(v)\}$. However, this one-hot coding reduces the generalizability to other domains and induces instability of the learning procedure in practice. Such an issue is often mitigated by introducing a strongly concave regularizer in the inner maximization step to strike a balance between the average and the worst-case performance [130, 132].

Regularized formulation. Following [130], we penalize the distance between the worst-case loss and the average loss over $K$ domains. This yields

$$
\min_{v \in V} \max_{w \in P} \sum_{i=1}^{K} w_i F_i(v) - \frac{\gamma}{2} \|w - 1/K\|_2^2,
$$

(5.4)

where $\gamma > 0$ is a regularization parameter. As $\gamma \to 0$, problem (5.4) is equivalent to (5.1). By contrast, it becomes the finite-sum problem when $\gamma \to \infty$ since $w \to 1/K$. In this sense, the trainable $w$ provides an essential indicator on the importance level of each domain. The larger the weight is, the more important the domain is. We call $w$ domain weights in this chapter.

5.4 Min-max power in attack design

To the best of our knowledge, few works have studied the power of min-max in attack generation. In this section, we demonstrate how the unified min-max framework (5.4) fits into various attack settings. With the help of domain weights, our solution yields better empirical performance and
explainability. Finally, we present the min-max algorithm with convergence analysis to craft robust perturbations against multiple domains.

5.4.1 A unified framework for robust adversarial attacks

The general goal of adversarial attack is to craft an adversarial example \( x' = x_0 + \delta \in \mathbb{R}^d \) to mislead the prediction of machine learning (ML) or deep learning (DL) systems, where \( x_0 \) denotes the natural example with the true label \( t_0 \), and \( \delta \) is known as \textit{adversarial perturbation}, commonly subject to \( \ell_p \)-norm \((p \in \{0, 1, 2, \infty\})\) constraint \( X := \{ \delta \mid \| \delta \|_p \leq \epsilon, \ x_0 + \delta \in [0, 1]^d \} \) for a given small number \( \epsilon \). Here the \( \ell_p \) norm enforces the similarity between \( x' \) and \( x_0 \), and the input space of ML/DL systems is normalized to \([0, 1]^d\).

**Ensemble attack over multiple models.** Consider \( K \) ML/DL models \( \{\mathcal{M}_i\}_{i=1}^{K} \), the goal is to find robust adversarial examples that can fool all \( K \) models simultaneously. In this case, the notion of ‘domain’ in (5.4) is specified as ‘model’, and the objective function \( F_i \) in (5.4) signifies the attack loss \( f(\delta; x_0, y_0, \mathcal{M}_i) \) given the natural input \((x_0, y_0)\) and the model \( \mathcal{M}_i \). Thus, problem (5.4) becomes

\[
\begin{align*}
\min_{\delta \in X} \max_{w \in P} & \sum_{i=1}^{K} w_i f(\delta; x_0, y_0, \mathcal{M}_i) - \frac{\gamma}{2} \| w - 1/K \|_2^2, \\
\end{align*}
\]

where \( w \) encodes the difficulty level of attacking each model.

**Universal perturbation over multiple examples.** Consider \( K \) natural examples \( \{(x_i, y_i)\}_{i=1}^{K} \) and a single model \( \mathcal{M} \), our goal is to find the universal perturbation \( \delta \) so that all the corrupted \( K \) examples can fool \( \mathcal{M} \). In this case, the notion of ‘domain’ in (5.4) is specified as ‘example’, and problem (5.4) becomes

\[
\begin{align*}
\min_{\delta \in X} \max_{w \in P} & \sum_{i=1}^{K} w_i f(\delta; x_i, y_i, \mathcal{M}) - \frac{\gamma}{2} \| w - 1/K \|_2^2, \\
\end{align*}
\]

(5.6)
Algorithm 5 APGD to solve the min-max problem

1: Input: given \(\mathbf{w}^{(0)}\) and \(\delta^{(0)}\).
2: for \(t = 1, 2, \ldots, T\) do
3:     outer min.: fixing \(\mathbf{w} = \mathbf{w}^{(t-1)}\), call PGD (5.7) to update \(\delta^{(t)}\)
4:     inner max.: fixing \(\delta = \delta^{(t)}\), update \(\mathbf{w}^{(t)}\) with projected gradient ascent (5.11)
5: end for

where different from (5.5), \(\mathbf{w}\) encodes the difficulty level of attacking each example.

Benefits of min-max attack generation with learnable domain weights \(\mathbf{w}\). We can interpret (5.5)-(5.6) as finding the robust adversarial attack against the worst-case environment that an adversary encounters, e.g., multiple victim models, data samples, and input transformations. The proposed min-max design of adversarial attacks leads to two main benefits. First, compared to the heuristic weighting strategy (e.g., clipping thresholds on the importance of individual attack losses [127]), our proposal is free of supervised manual adjustment on domain weights. Even by carefully tuning the heuristic weighting strategy, we find that our approach with self-adjusted \(\mathbf{w}\) consistently outperforms the clipping strategy in [127]. Second, the learned domain weights can be used to assess the model robustness when facing different types of adversary.

5.4.2 Min-max algorithm for adversarial attack generation

We propose the alternating one-step projected gradient descent (APGD) method (Algorithm 5) to solve problem (5.4). For ease of presentation, we write problems (5.5), (5.6) into the general form

\[
\min_{\delta \in \mathcal{X}} \max_{\mathbf{w} \in \mathcal{P}} \sum_{i=1}^{K} w_i F_i(\delta) - \frac{\gamma}{2} \|\mathbf{w} - \mathbf{1}/K\|_2^2,
\]

where \(F_i\) denotes the \(i\)th individual attack loss. We show that at each iteration, APGD takes only one-step PGD for outer minimization and one-step projected gradient ascent for inner maximization.
**Outer minimization** Considering \( w = w^{(t-1)} \) and \( F(\delta) := \sum_{i=1}^{K} w_i^{(t-1)} F_i(\delta) \) in (5.4), we perform one-step PGD to update \( \delta \) at iteration \( t \),

\[
\delta^{(t)} = \text{proj}_X \left( \delta^{(t-1)} - \alpha \nabla_{\delta} F(\delta^{(t-1)}) \right),
\]

where \( \text{proj}(\cdot) \) denotes the Euclidean projection operator, i.e., \( \text{proj}_X(a) = \arg \min_{x \in X} \| x - a \|_2 \) at the point \( a \), \( \alpha > 0 \) is a given learning rate, and \( \nabla_{\delta} \) denotes the first-order gradient w.r.t. \( \delta \). If \( p = \infty \), then the projection function becomes the clip function. In Proposition 2, we derive the solution of \( \text{proj}_X(a) \) under different \( \ell_p \) norms for \( p \in \{0, 1, 2\} \).

**Proposition 2.** Given a point \( a \in \mathbb{R}^d \) and a constraint set \( X = \{ \delta \mid \| \delta \|_p \leq \epsilon, \hat{c} \leq \delta \leq \check{c} \} \), the Euclidean projection \( \delta^* = \text{proj}_X(a) \) has the closed-form solution when \( p \in \{0, 1, 2\} \).

1) If \( p = 1 \), then \( \delta^* \) is given by

\[
\delta^*_i = \begin{cases} 
P_{[\hat{c}_i, \check{c}_i]}(a_i) & \sum_{i=1}^d |P_{[\hat{c}_i, \check{c}_i]}(a_i)| \leq \epsilon \\
\text{proj}_{[\hat{c}_i, \check{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \lambda_1, 0\}) & \text{otherwise},
\end{cases}
\]

where \( x_i \) denotes the \( i \)th element of a vector \( x \); \( P_{[\hat{c}_i, \check{c}_i]}(\cdot) \) denotes the clip function over the interval \([\hat{c}_i, \check{c}_i]\); \( \text{sign}(x) = 1 \text{ if } x \geq 0, \text{ otherwise } 0 \); \( \lambda_1 \in (0, \max_i |a_i| - \epsilon/d] \) is the root of \( \sum_{i=1}^d |P_{[\hat{c}_i, \check{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \lambda_1, 0\})| = \epsilon \).

2) If \( p = 2 \), then \( \delta^* \) is given by

\[
\delta^*_i = \begin{cases} 
P_{[\hat{c}_i, \check{c}_i]}(a_i) & \sum_{i=1}^d (P_{[\hat{c}_i, \check{c}_i]}(a_i))^2 \leq \epsilon^2 \\
P_{[\hat{c}_i, \check{c}_i]}(a_i/(\lambda_2 + 1)) & \text{otherwise},
\end{cases}
\]

where \( \lambda_2 \in (0, \|a\|_2/\epsilon - 1] \) is the root of \( \sum_{i=1}^d (P_{[\hat{c}_i, \check{c}_i]}(a_i/(\lambda_2 + 1))^2 = \epsilon^2 \).
3) If \( p = 0 \) and \( \epsilon \in \mathbb{N}_+ \), then \( \delta^* \) is given by

\[
\delta_i^* = \begin{cases} 
\delta_i & \eta_i \geq [\eta]_\epsilon \\
0 & \text{otherwise},
\end{cases}
\eta_i = \begin{cases} 
\sqrt{2a_i\hat{c}_i - \hat{c}_i^2} & a_i < \hat{c}_i \\
\sqrt{2a_i\hat{c}_i - \hat{c}_i^2} & a_i > \hat{c}_i \\
|a_i| & \text{otherwise}.
\end{cases}
\tag{5.10}
\]

where \([\eta]_\epsilon\) denotes the \( \epsilon \)-th largest element of \( \eta \), and \( \delta_i' = P_{[c_i, \hat{c}_i]}(a_i) \).

**Proof:** See Appendix A.

**Inner maximization** By fixing \( \delta = \delta(t) \) and letting \( \psi(w) := \sum_{i=1}^{K} w_i F_i(\delta(t)) - \frac{\gamma}{2} \| w - 1/K \|_2^2 \) in problem (5.4), we then perform one-step PGD (w.r.t. \( -\psi \)) to update \( w \),

\[
w(t) = \text{proj}_P \left( w(t-1) + \beta \nabla_w \psi(w(t-1)) \right)
= (b - \mu 1)_+, \tag{5.11}
\]

where \( \beta > 0 \) is a given learning rate, \( \nabla_w \psi(w) = \phi(t) - \gamma(w - 1/K) \), and \( \phi(t) := [F_1(\delta(t)), \ldots, F_K(\delta(t))]^T \). In (5.11), the second equality holds due to the closed-form of projection operation onto the probabilistic simplex \( P \) \cite{135}, where \( (x)_+ = \max\{0, x\} \), and \( \mu \) is the root of the equation \( 1^T(b - \mu 1)_+ = 1 \). Since \( 1^T(b - \min_i\{b_i\} 1 + 1/K)_+ \geq 1^T1/K = 1 \), and \( 1^T(b - \max_i\{b_i\} 1 + 1/K)_+ \leq 1^T1/K = 1 \), the root \( \mu \) exists within the interval \([\min_i\{b_i\} - 1/K, \max_i\{b_i\} - 1/K]\) and can be found via the bisection method \cite{134}.

**Convergence analysis** We remark that APGD follows the gradient primal-dual optimization framework \cite{133}, and thus enjoys the same optimization guarantees.

**Theorem 1.** Suppose that in problem (5.4) \( F_i(\delta) \) has \( L \)-Lipschitz continuous gradients, and \( X \) is a convex compact set. Given learning rates \( \alpha \leq \frac{1}{L} \) and \( \beta < \frac{1}{\gamma} \), then the sequence \( \{\delta(t), w(t)\}_{t=1}^T \) generated by Algorithm 5 converges to a first-order stationary point\(^1\) in rate \( O(\frac{1}{T}) \).

\(^1\)The stationarity is measured by the \( \ell_2 \) norm of gradient of the objective in (5.4) w.r.t. \( (\delta, w) \).
5.5 Min-max power in defense

In this section, we show that the min-max principle can be used to generalize AT from a defender’s perspective. Different from promoting robustness of adversarial examples against the worst-case attacking environment (Sec. 5.4), the generalized AT promotes model’s robustness against the worst-case defending environment, given by the existence of diversified $\ell_p$ attacks.

One key challenge in generalized AT is that the multiple $\ell_p$ perturbations overlap largely thus weakening the diversity of inner threat models. To enhance the worst-case defending environment, we propose quantifying and regularizing the diversity of $\ell_p$ attacks (Sec. 5.5.2) to gain complementary robustness from defense against diversified attacks.

5.5.1 A unified framework for adversarial training under mixed types of adversarial attacks

Conventional AT is restricted to a single type of norm-ball constrained adversarial attack [104]. For example, AT under $\ell_\infty$ attack yields

$$\min_{\theta} \mathbb{E}_{(x, y) \in D} \max_{||\delta||_{\infty} \leq \epsilon} f_{tr}(\theta, \delta; x, y), \quad (5.12)$$

where $\theta \in \mathbb{R}^n$ denotes model parameters, $\delta$ denotes $\epsilon$-tolerant $\ell_\infty$ attack, and $f_{tr}(\theta, \delta; x, y)$ is the training loss under perturbed examples $\{(x + \delta, y)\}$. However, there possibly exist blind attacking spots across multiple types of adversarial attacks so that AT under one attack would not be strong enough against another attack [32]. Thus, an interesting question is how to generalize AT under multiple types of adversarial attacks. One possible way is to use the finite-sum formulation in the inner maximization problem of (5.12), namely, maximize $\epsilon_{\delta_i \in \mathcal{X}_i} \frac{1}{K} \sum_{i=1}^{K} f_{tr}(\theta, \delta; x, y)$, where $\delta_i \in \mathcal{X}_i$ is the $i$th type of adversarial perturbation defined on $\mathcal{X}_i$, e.g., different $\ell_p$ attacks.
However, one can also map ‘attack type’ to ‘domain’ considered in (5.1). We then generalize AT against the strongest adversarial attack across $K$ attack types in order to avoid blind attacking spots:

\[
\minimize_{\theta} \mathbb{E}_{(x, y) \in \mathcal{D}} \maximize_{i \in [K]} \maximize_{\delta_i \in \mathcal{X}_i} f_{tr}(\theta, \delta_i; x, y).
\] (5.13)

In Lemma 1 we show that problem (5.13) can be equivalently transformed into the min-max form.

**Lemma 1.** Problem (5.13) is equivalent to

\[
\minimize_{\theta} \mathbb{E}_{(x, y) \in \mathcal{D}} \maximize_{w \in \mathcal{P}, \{\delta_i \in \mathcal{X}_i\}} \sum_{i=1}^{K} w_i f_{tr}(\theta, \delta_i; x, y),
\] (5.14)

where $w \in \mathbb{R}^K$ represent domain weights, and $\mathcal{P}$ has been defined in (5.1).

**PROOF:** See Appendix C.

Similar to (5.4), a strongly concave regularizer $-\gamma/2\|w - 1/K\|_2^2$ can be added into the inner maximization problem of (5.14) for boosting the stability of the learning procedure and striking a balance between the max and the average attack performance. We finally remark that there was an independent work [31] which also generalized AT under multiple perturbations. However, our proposal is conceptually different from [31] as we generalize AT from the perspective of min-max optimization.

### 5.5.2 Improved robustness via diversified $\ell_p$ attacks

It was recently shown in [136, 137] that the diversity of multiple neural networks improves adversarial robustness of an ensemble model. Different from the previous work to promote model diversity, we measure the diversity between adversarial attacks under a single ML model. Such an attack diversity can be quantified through the similarity between perturbation directions, namely, input gradients $\{\nabla_{\delta_i} f_{tr}(\theta, \delta_i; x, y)\}_i$ in (5.14). Since different $\ell_p$ perturbations overlap largely, we examine whether or not the promotion of diversity among $\ell_p$ attacks is beneficial to adversarial
robustness. We enhance the diversity through

\[ h(\theta, \{\delta_i\}; x, y) := \log \det(G^T G), \tag{5.15} \]

where \( G \in \mathbb{R}^{d \times K} \) is a \( d \times K \) matrix, each column of which corresponds to a normalized input gradient \( \nabla_{\delta_i} f_{\text{tr}}(\theta, \delta_i; x, y) \) for \( i \in [K] \), and \( h(\theta, \{\delta_i\}; x, y) \) reaches the maximum value 0 as input gradients become orthogonal. Note that a diversity regularizer was used for defense in [137], but for promoting the diversity of ensemble models rather than perturbation directions under a single model. With the aid of (5.15) and a strongly concave regularizer, we modify problem (5.14) to

\[
\min_{\theta} \mathbb{E}_{(x, y) \in D} \max_{w \in \mathcal{P}, \{\delta_i \in \mathcal{X}_i\}} \psi(\theta, w, \{\delta_i\}) \\
\psi(\theta, w, \{\delta_i\}) := \sum_{i=1}^{K} w_i f_{\text{tr}}(\theta, \delta_i; x, y) - \frac{\gamma}{2} \|w - 1/K\|^2_2 + \lambda h(\theta, \{\delta_i\}; x, y) \tag{5.16}
\]

The rationale behind (5.16) is that the inner maximization enforces the worst-case defending environment: The adversary aims to enhance the effectiveness of attacks from diversified perturbation directions. However, during outer minimization, the defender enhances the capability of the model \( \theta \) to defend those diversified attacks.

### 5.5.3 Min-max algorithm for generalized adversarial training

We next propose the alternating multi-step projected gradient descent (AMPGD) method to solve the problem (5.16). We summarize AMPGD in Algorithm 6.

**Algorithm 6** AMPGD to solve problem (5.16)

```
1: Input: given \( \theta^{(0)}, w^{(0)}, \delta^{(0)} \) and \( K > 0 \).
2: for \( t = 1, 2, \ldots, T \) do
3:     given \( w^{(t-1)} \) and \( \delta^{(t-1)} \), perform SGD to update \( \theta^{(t)} \)
4:     given \( \theta^{(t)} \), perform \( R \)-step PGD to update \( w^{(t)} \) and \( \delta^{(t)} \)
5: end for
```

Problem (5.16) is in a more general non-convex non-concave min-max setting, where the inner
maximization involves both domain weights $w$ and adversarial perturbations $\{\delta_i\}$. It was shown in [138] that the multi-step PGD is required for inner maximization in order to approximate the near-optimal solution. This is also in the similar spirit of AT [104], which executed multi-step PGD attack during inner maximization. At step 4 of Algorithm 6 each PGD step to update $w$ and $\delta$ can be decomposed as

$$w_{r}^{(t)} = \text{proj}_P \left( w_{r-1}^{(t)} + \beta \nabla_{w} \psi(\theta^{(t)}, w_{r-1}^{(t)}, \{\delta_{i,r-1}^{(t)}\}) \right), \forall r \in [R],$$

$$\delta_{i,r}^{(t)} = \text{proj}_{X_i} \left( \delta_{i,r-1}^{(t)} + \beta \nabla_{\delta} \psi(\theta^{(t)}, w_{r-1}^{(t)}, \{\delta_{i,r-1}^{(t)}\}) \right), \forall r, i \in [R], [K]$$

where let $w_{1}^{(t)} := w^{(t-1)}$ and $\delta_{i,1}^{(t)} := \delta_{i}^{(t-1)}$. Here the superscript $t$ represents the iteration index of AMPGD, and the subscript $r$ denotes the iteration index of $R$-step PGD. Clearly, the above projection operations can be derived for closed-form expressions through (5.11) and Lemma 2. To the best of our knowledge, it is still an open question to build theoretical convergence guarantees for solving the general non-convex non-concave min-max problem like (5.16), except the work [138] which proposed $O(1/T)$ convergence rate if the objective function satisfies a strict Polyak-Łojasiewicz condition [139].

### 5.6 Numerical results

In this section, we first evaluate the proposed min-max optimization strategy on three attack tasks. We show that our approach leads to substantial improvement compared with state-of-the-art attack methods such as ensemble PGD [29] and expectation over transformation (EOT) [105, 140, 141].

We next demonstrate the effectiveness of the generalized AT for multiple types of adversarial perturbations. We show that the use of trainable domain weights in problem (5.16) can automatically adjust the risk level of different attacks during the training process even if the defender lacks prior knowledge on the strength of these attacks. We also show that the promotion of diversity of $\ell_p$ attacks help improve adversarial robustness further.
5.6.1 Min-max adversarial attacks

Fig. 5.1: Ensemble attack against four DNN models on MNIST. (a) & (b): Attack success rate of adversarial examples generated by average (ensemble PGD) or min-max (APGD) attack method. (c): Boxplot of weight \( w \) in APGD adversarial loss. Here we adopt the same \( \ell_\infty \)-attack as Table 5.1.

In what follows, we show the great strength of min-max optimization also lies at the side of attack generation. Note that problem formulations (5.5)-(5.6) are applicable to both untargeted and targeted attack. Here we focus on the former setting and use C&W loss function \([26, 104]\).

Ensemble attack over multiple models We craft adversarial examples against an ensemble of known classifiers. Recent work \([29]\) proposed an ensemble PGD attack, which assumed equal importance among different models, namely, \( w_i = 1/K \) in problem (5.5). Throughout this task, we measure the attack performance via ASR_{all} - the attack success rate (ASR) of fooling model ensembles simultaneously. Compared to the ensemble PGD attack, our approach results in 40.79% ASR_{all} improvement averaged over different \( \ell_p \)-norm constraints on MNIST, respectively. In what follows, we provide more detailed experiment results and analysis.

In Table 5.1, we show that our min-max APGD significantly outperforms ensemble PGD in ASR_{all}. Taking \( \ell_\infty \)-attack on MNIST as an example, our min-max attack leads to a 90.16% ASR_{all}, which largely outperforms 48.17% (ensemble PGD). The reason is that Model C, D are more difficult to attack, which can be observed from their higher test accuracy on adversarial examples. As a result, although the adversarial examples crafted by assigning equal weights over multiple models are able to attack \{A, B\} well, they achieve a much lower ASR in \{C, D\}. By contrast, APGD automatically handles the worst case \{C, D\} by slightly sacrificing the performance on \{A, B\}: 31.47% averaged ASR improvement on \{C, D\} versus 0.86% degradation on \{A, B\}. 
Table 5.1: Comparison of average and min-max (APGD) ensemble attack over four models on MNIST. Acc (%) represents the test accuracy of classifiers on adversarial examples.

<table>
<thead>
<tr>
<th>Box constraint</th>
<th>Opt.</th>
<th>Acc_A</th>
<th>Acc_B</th>
<th>Acc_C</th>
<th>Acc_D</th>
<th>ASR_all</th>
<th>Lift (↑)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\ell_0$ ($\epsilon = 30$)</td>
<td>avg.</td>
<td>7.03</td>
<td>1.51</td>
<td>11.27</td>
<td>2.48</td>
<td>84.03</td>
<td>91.97</td>
</tr>
<tr>
<td></td>
<td>min max</td>
<td>3.65</td>
<td>2.36</td>
<td>4.99</td>
<td>3.11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\ell_1$ ($\epsilon = 20$)</td>
<td>avg.</td>
<td>20.79</td>
<td>0.15</td>
<td>21.48</td>
<td>6.70</td>
<td>69.31</td>
<td>89.16</td>
</tr>
<tr>
<td></td>
<td>min max</td>
<td>6.12</td>
<td>2.53</td>
<td>8.43</td>
<td>5.11</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\ell_2$ ($\epsilon = 3.0$)</td>
<td>clip [127]</td>
<td>6.88</td>
<td>0.03</td>
<td>26.28</td>
<td>14.50</td>
<td>69.12</td>
<td>71.54</td>
</tr>
<tr>
<td></td>
<td>min max</td>
<td>0.66</td>
<td>0.03</td>
<td>23.43</td>
<td>13.23</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>clip [127]</td>
<td>1.51</td>
<td>0.89</td>
<td>3.50</td>
<td>2.06</td>
<td>95.31</td>
<td>71.54</td>
</tr>
<tr>
<td>$\ell_\infty$ ($\epsilon = 0.2$)</td>
<td>avg.</td>
<td>1.05</td>
<td>0.07</td>
<td>41.10</td>
<td>35.03</td>
<td>48.17</td>
<td>48.52%</td>
</tr>
<tr>
<td></td>
<td>clip [127]</td>
<td>0.66</td>
<td>0.03</td>
<td>23.43</td>
<td>13.23</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>min max</td>
<td>2.47</td>
<td>0.37</td>
<td>7.39</td>
<td>5.81</td>
<td>90.16</td>
<td>71.54</td>
</tr>
<tr>
<td></td>
<td>clip [127]</td>
<td>0.66</td>
<td>0.03</td>
<td>23.43</td>
<td>13.23</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Effectiveness of learnable domain weights  
Figure 5.1 depicts the ASR of four models under average/min-max attacks as well as the distribution of domain weights during attack generation. For ensemble PGD (Figure 5.1a), Model C and D are attacked insufficiently, leading to relatively low ASR and thus weak ensemble performance. By contrast, APGD (Figure 5.1b) will encode the difficulty level to attack different models based on the current attack loss. It dynamically adjusts the weight $w_i$ as shown in Figure 5.1c. For instance, the weight for Model D is first raised to 0.45 because D is difficult to attack initially. Then it decreases to 0.3 once Model D encounters the sufficient attack power and the corresponding attack performance is no longer improved. It is worth noticing that APGD is highly efficient because $w_i$ converges after a small number of iterations. Figure 5.1c also shows $w_c > w_d > w_a > w_b$ – indicating a decrease in model robustness for C, D, A and B, which is exactly verified by $\text{Acc}_C > \text{Acc}_D > \text{Acc}_A > \text{Acc}_B$ in the last row of Table 5.1 ($\ell_\infty$-norm). As the perturbation radius $\epsilon$ varies, we also observe that the ASR of min-max strategy is consistently better or on part with the average strategy.

Comparison with stronger heuristic baselines  
Apart from average strategy, we compare min-max framework with a stronger heuristic weighting scheme - loss clipping [127] in Table 5.1. Briefly, we achieve substantial improvement over baselines consistently. Also, we show that even
**Table 5.2:** Comparison of average and minmax optimization on universal perturbation over multiple input examples. The adversarial examples are generated by 20-step \(\ell_\infty\)-APGD.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>Setting</th>
<th>(K = 2)</th>
<th>(K = 4)</th>
<th>(K = 5)</th>
<th>(K = 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(K)</td>
<td>(\text{ASR}_{\text{avg}})</td>
<td>(\text{ASR}_{\text{gp}})</td>
<td>(\text{ASR}_{\text{avg}})</td>
<td>(\text{ASR}_{\text{gp}})</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>All-CNNs</td>
<td>avg.</td>
<td>91.09</td>
<td>83.08</td>
<td>-</td>
<td>85.66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min max</td>
<td>92.22</td>
<td>85.98</td>
<td>3.49%</td>
<td>87.63</td>
</tr>
<tr>
<td></td>
<td>LeNetV2</td>
<td>avg.</td>
<td>93.26</td>
<td>86.90</td>
<td>-</td>
<td>90.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min max</td>
<td>93.34</td>
<td>87.08</td>
<td>0.21%</td>
<td>91.91</td>
</tr>
<tr>
<td></td>
<td>VGG16</td>
<td>avg.</td>
<td>90.76</td>
<td>82.56</td>
<td>-</td>
<td>89.36</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min max</td>
<td>92.40</td>
<td>85.92</td>
<td>4.07%</td>
<td>90.04</td>
</tr>
<tr>
<td></td>
<td>GoogLeNet</td>
<td>avg.</td>
<td>85.02</td>
<td>72.48</td>
<td>-</td>
<td>75.20</td>
</tr>
<tr>
<td></td>
<td></td>
<td>min max</td>
<td>87.08</td>
<td>77.82</td>
<td>7.37%</td>
<td>77.05</td>
</tr>
</tbody>
</table>

Adopting converged min-max weights statically leads to a huge performance drop on attacking model ensembles, which again verifies the power of dynamically optimizing domain weights during attack generation process.

**Multi-image universal perturbation** We evaluate APGD in universal perturbation on MNIST and CIFAR-10, where 10,000 test images are randomly divided into equal-size groups (\(K\) images per group) for universal perturbation. We measure two types of ASR (%), \(\text{ASR}_{\text{avg}}\) and \(\text{ASR}_{\text{gp}}\). Here the former represents the ASR averaged over all images in all groups, and the latter signifies the ASR averaged over all groups but a successful attack is counted under a more restricted condition: images within each group must be successfully attacked simultaneously by universal perturbation. When \(K = 5\), our approach achieves 35.21% improvement over the averaging strategy under CIFAR-10.

In Table 5.2, we compare the proposed min-max strategy (APGD) with the averaging strategy on the attack performance of generated universal perturbations. As we can see, our method always achieves higher \(\text{ASR}_{\text{gp}}\) for different values of \(K\). The universal perturbation generated from APGD can successfully attack ‘hard’ images (on which the average-based PGD attack fails) by self-adjusting domain weights, and thus leads to a higher \(\text{ASR}_{\text{gp}}\). Besides, the min-max universal perturbation also offers interpretability of “image robustness” by associating domain weights with image visualization.
5.6.2 Robust training under multiple types of $\ell_p$ attacks

Table 5.3: Adversarial training on single attacks ($\ell_\infty$ and $\ell_2$) and multiple attacks (avg. and min max). The perturbation magnitude $\epsilon$ for $\ell_\infty$ and $\ell_2$ attacks are 0.2 and 2.0, respectively. (a) & (b) on MNIST; (c) & (d) on CIFAR-10.

<table>
<thead>
<tr>
<th></th>
<th>(a) MLP</th>
<th></th>
<th>(b) LeNet</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Opt.</td>
<td>Acc-$\ell_\infty$</td>
<td>Acc-$\ell_2$</td>
<td>Acc$^{\text{max}}_{\text{adv}}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acc-$\ell_\infty$</td>
<td>Acc-$\ell_2$</td>
<td>Acc$^{\text{max}}_{\text{adv}}$</td>
</tr>
<tr>
<td></td>
<td>natural</td>
<td>2.70</td>
<td>13.86</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>$\ell_\infty$</td>
<td>77.70</td>
<td>69.17</td>
<td>66.34</td>
</tr>
<tr>
<td></td>
<td>$\ell_2$</td>
<td>70.03</td>
<td>81.74</td>
<td>69.14</td>
</tr>
<tr>
<td></td>
<td>avg.</td>
<td>75.09</td>
<td>79.00</td>
<td>72.23</td>
</tr>
<tr>
<td></td>
<td>min max</td>
<td>75.96</td>
<td>79.15</td>
<td>73.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td>76.92</td>
<td>79.74</td>
<td>74.29</td>
</tr>
<tr>
<td></td>
<td>+ DPAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>avg.</td>
<td>89.21</td>
<td>85.98</td>
<td>84.82</td>
</tr>
<tr>
<td></td>
<td></td>
<td>90.19</td>
<td>86.47</td>
<td>85.47</td>
</tr>
<tr>
<td></td>
<td>min max</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ DPAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acc$^{\text{max}}_{\text{adv}}$</td>
<td>avg. + DPAR</td>
<td>min max + DPAR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acc$^{\text{max}}_{\text{adv}}$</td>
<td>32.63</td>
<td>33.97</td>
<td>34.71</td>
<td>35.22</td>
</tr>
<tr>
<td>Acc$^{\text{avg}}_{\text{adv}}$</td>
<td>35.98</td>
<td>36.71</td>
<td>37.53</td>
<td>38.60</td>
</tr>
</tbody>
</table>

Compared to vanilla AT, we show the generalized AT scheme produces models robust to multiple types of perturbation, thus leads to stronger “overall robustness”. We measure the training performance using two types of Acc ($\%$): Acc$^{\text{max}}_{\text{adv}}$ and Acc$^{\text{avg}}_{\text{adv}}$, where Acc$^{\text{max}}_{\text{adv}}$ denotes the test accuracy over examples with the strongest perturbation ($\ell_\infty$ or $\ell_2$), and Acc$^{\text{avg}}_{\text{adv}}$ denotes the averaged test accuracy over examples with all types of perturbations ($\ell_\infty$ and $\ell_2$). Moreover, we measure the
overall worst-case robustness $S_\epsilon$ in terms of the area under the curve ‘Acc$^{\text{max}}_{\text{adv}}$ vs. $\epsilon$’ (Figure 5.2b).

In Table 5.3 we present the test accuracy in different training schemes: a) natural training, b) single-norm: vanilla AT ($\ell_\infty$ or $\ell_2$), c) multi-norm: generalized AT ($\text{avg}$ and $\text{min max}$), and d) generalized AT with diversity-promoting attack regularization (DPAR, $\lambda = 0.1$ in problem (5.15)). If the adversary only performs single-type attack, training and testing on the same attack type leads to the best performance (diagonal of $\ell_\infty$-$\ell_2$ block). However, when facing $\ell_\infty$ and $\ell_2$ attacks simultaneously, multi-norm generalized AT achieves better Acc$^{\text{max}}_{\text{adv}}$ and Acc$^{\text{avg}}_{\text{adv}}$ than single-norm AT. In particular, the min-max strategy outperforms the averaging strategy under multiple perturbation norms. DPAR further boosts the adversarial test accuracy, which implies that the promotion of diversified $\ell_p$ attacks is a beneficial supplement to adversarial training. We also observed for ResNets on CIFAR-10, our approach results in larger benefits ($2 \sim 3\%$ improvement on Acc$^{\text{max}}_{\text{adv}}$).

**Effectiveness of domain weights and diversity promotion regularizer** In Figure 5.2 we offer deeper insights on the performance of generalized AT. During the training procedure we fix $\epsilon_{\ell_\infty}$ ($\epsilon$ for $\ell_\infty$ attack during training) as 0.2, and change $\epsilon_{\ell_2}$ from 0.2 to 5.6 ($\epsilon_{\ell_\infty} \times \sqrt{d}$) so that the $\ell_\infty$ and $\ell_2$ balls are not completely overlapped [32]. In Figure 5.2a, as $\epsilon_{\ell_2}$ increases, $\ell_2$-attack becomes stronger so the corresponding $w$ also increases, which is consistent with min-max spirit – defending the strongest attack. We remark that min max or $\text{avg}$ training does not always lead to the best performance on Acc$^{\text{max}}_{\text{adv}}$ and Acc$^{\text{avg}}_{\text{adv}}$, especially when the strengths of two attacks diverge greatly (see Figure 5.2a, 5.2b). However, Figure 5.2b shows that AMPGD is able to achieve a rather robust model no matter how $\epsilon$ changes (red lines), which empirically verifies the effectiveness of our proposed training scheme. In terms of area-under-the-curve $S_\epsilon$, AMPGD achieves the highest worst-case robustness: 6.27% and 17.64% improvement compared to the vanilla AT with $\ell_\infty$ and $\ell_2$ attacks. Lastly, we study the effectiveness of the proposed DPAR regularizer. Figure 5.2c shows how the diversity (gradient orthogonality) of multi-norm attacks alters with the increase of model robustness during the training procedure. Note that the larger the regularization term is, the more diverse the multiple $\ell_p$ attacks are. Consequently, DPAR boosts the model robustness via
promoting attack diversity (enforcing more diverse gradient directions).

5.7 Summary

In this chapter, we propose a general min-max framework applicable to both adversarial attack and defense settings. We show that many problem setups can be re-formulated under this general framework. Extensive experiments show that proposed algorithms lead to significant improvement on multiple attack and defense tasks compared with previous state-of-the-art approaches. Our min-max scheme also generalizes adversarial training (AT) for multiple types of adversarial attacks, attaining faster convergence and better robustness compared to the vanilla AT and the average strategy. Lastly, we propose to promote the ensemble diversity of input gradients (corresponding to different $\ell_p$ attacks) to further improve the defensive performance of the generalized AT.
CHAPTER 6
ON THE OPTIMAL INTERDICTION OF
TRANSPORTATION NETWORKS

6.1 Introduction

Transportation networks, also known as flow networks, are networks in which mass enters through source nodes and on-ramps, is routed through nodes/cells and directed links, and is removed at sink nodes and off-ramps. The flow of mass is subject to (i) conservation of mass constraints, and (ii) link capacity constraints. Traffic networks, water supply networks, and (routing of data packets in) computer networks are all examples of transportation networks.

The cell transmission model of mass transfer developed by Daganzo [142, 143] captures complex traffic behavior and transient phenomena, such as congestion effects and the propagation of shocks. Ziliaskopoulos [144] used the cell transmission model to formulate the optimum traffic assignment problem as a linear program. In an influential sequence of recent papers [145–148], Como et al. and Savla et al. analyzed the robustness and resilience of transportation networks under decentralized routing. In particular, they proposed routing policies that depend only on local information and maximally delay congestion effects under adversarial perturbations to the capacities of cells.
In this chapter, we study the interdiction or attack on transportation networks, which for the sake of concreteness we consider to be highway traffic networks. In this context it is of interest to find a small set of cells whose failure at time zero, amplified and propagated by the system’s dynamics, maximally disrupts the flow of traffic. This problem is combinatorial in nature and intractable in general. Our work follows [1] in formulating the optimal interdiction problem as a min-max optimization problem and subsequently employing duality to transform it to a standard bilinear optimization problem.

We demonstrate that even without an explicit promotion of sparsity in the formulation, the solution to the optimal interdiction problem is both sparse and binary. The solution is sparse in the sense that the attacker’s best use of resources is to find the small set of most consequential cells in the network, and it is binary in the sense that the attacker’s best choice is to fail these cells fully (as opposed to partially).

Furthermore, motivated by the block coordinate gradient descent (BCGD) and block coordinate descent (BCD) algorithms [149], we solve the bilinear problem by iteratively updating one set of variables through a gradient-based step and then finding the globally optimal solution in the other set of variables. Our numerical experiments demonstrate that our approach performs better in comparison with methods reported in earlier work [1], and we find the globally optimal solution in the small networks that we tested and for which the global optimum could be verified through exhaustive search.

6.2 Dynamic model of transportation networks & informal statement of optimal interdiction problem

In this section we first introduce the cell transmission model. We augment the model in a way that allows for the irreversible failure of cells through two mechanisms: being attacked by an adversary and reaching the jam threshold through the accumulation of mass. We then discuss a meaningful formulation of the optimal interdiction problem subject to attacker resource constraints.
The network is characterized by a directed graph, where we think of nodes as cells and of edges as allowing for flows between neighboring cells. (In this work we use the words cell and node interchangeably.) The temporal dynamics for the cell transmission model are governed in part by

\[ x_i(t) = x_i(t-1) + y_i(t-1) - z_i(t-1) \]  
(conservation of mass on cell \(i\))

\[ y_i(t) = v_i(t) + \sum_j f_{ji}(t) \]  
(total inflow to cell \(i\) = on-ramp flow + rerouted flow)

\[ z_i(t) = w_i(t) + \sum_j f_{ij}(t) \]  
(total outflow from cell \(i\) = off-ramp flow + rerouted flow)

for every \(i\) and \(t\), where \(x_i, y_i,\) and \(z_i\), respectively denote the mass (i.e., number of vehicles) on, the inflow to, and the outflow from, cell \(i\); \(v_i, w_i\) respectively denote the mass entering the network from on–ramp, and leaving the network from off–ramp, corresponding to cell \(i\); \(f_{ij}\) denotes the mass routed from cell \(i\) to adjacent cell \(j\). The dynamics are additionally constrained to

\[ x_i(t) \geq 0, \ v_i(t) \geq 0, \ w_i(t) \geq 0, \ f_{ij}(t) \geq 0 \]  
(positivity of mass)

\[ y_i(t) \leq \kappa_i, \ z_i(t) \leq \kappa_i \]  
(inflow, outflow cannot exceed flow-capacity of cell)

\[ y_i(t) \leq \phi_i - x_i(t), \ z_i(t) \leq x_i(t) \]  
(inflow cannot exceed remaining mass-capacity of cell, outflow cannot exceed mass on cell)
with the further restrictions that \( f_{ij}(\cdot) = 0 \) if cells \( i \) and \( j \) are not adjacent, \( w_i(\cdot) = 0 \) if cell \( i \) does not have an off-ramp, and \( v_i(\cdot) \) specified \textit{a priori}. The parameter \( \phi_i \) denotes the amount of mass that results in cell \( i \) being jammed. Inequalities (6.2b)–(6.2c) result from piecewise linear “supply” and “demand” functions [150].

We assume that all mass enters the network from on-ramps and possibly a source cell and that it leaves the network through a sink cell. The source and sink cells have very large capacities. Without loss of generality, we take the cell with the lowest index to be the source (when a source cell is present) and the cell with the largest index to be the sink.

As in [1], we further augment the dynamics (6.1)–(6.2) with

\[
\kappa_i \in \{0, \psi_i\},
\]

\[
z_i(t) \leq \phi_i - x_i(t),
\]

(6.3)

\( \psi_i \) denotes maximum mass can flow in or out of cell \( i \) during one time step, and \( \phi_i \) is the same jam mass as before. The constraints in (6.3), together with (6.2), capture two methods by which a cell irreversibly fails:

- at time \( 0 \) an attacker reduces the capacity of cell \( i \) to zero, \( \kappa_i = 0 \);

- at time \( t_0 \geq 1 \) and as a result of the network’s dynamics the accumulated mass on cell \( i \) reaches the jam threshold, \( x_i(t_0) = \phi_i \).

In both scenarios, once a cell has failed \textit{no mass can either enter or leave it} thereafter.

We assume that the attacker operates under a limited budget \( e \) and that the \( i \)th entry of the vector \( c \) characterizes the cost for the attacker of reducing \( \kappa_i \) from \( \psi_i \) to 0. This means that
The attacker is subject to the constraint
\[ c^T(1 - \kappa / \psi) \leq e, \]

\( l^T x(t) \) equals total mass at \( t \) on all cells except sink cell \( c \) denotes cost for the attacker of reducing \( \kappa_i \) from \( \psi_i \) to 0. where \( 1 \) is the column vector of all ones and division by a vector is element-wise.

The above inequality is equivalent to
\[ q^T \kappa \geq d \]

with \( q := c / \psi \) and \( d := 1^T c - e \).

Define \( l = [1, \ldots, 1, 0]^T \), problem in this work can be (informally) stated as follows \( x(t) \) as the vector whose \( i \)th entry is \( x_i(t) \), with similar definitions for vectors \( y(t), z(t), f(t), v(t), w(t), \kappa \), and taking \( l = [1, \ldots, 1, 0]^T \) so that \( l^T x(t) \) equals the total mass at time \( t \) on all cells except the sink cell, our main problem in this work can be (informally) stated as follows.

Optimal Interdiction Problem: Given the temporal evolution model (6.1)–(6.3) and a budget on the total amount of failures, for the time horizon \( 0, 1, \ldots, T \) find a sparse set of cells whose failure at time 0 maximizes the total travel time \( \sum_{t=0}^T l^T x(t) \).

We formulate the optimal interdiction problem as a game in which an attacker acts as the player who goes first and, subject to budget constraints, fails the most critical nodes at time 0 so as to maximize the total travel time of the mass. A centralized network operator then acts as the player who goes second and, subject to the dynamics (6.1)–(6.3), routes the mass so as to minimize its total travel time.

The optimal interdiction problem
\[
\max_{\kappa} \min_{x,y,z,f,w} \left( \sum_{t=0}^T l^T x(t) \right) - \gamma \card(\psi - \kappa),
\]

where the inner minimum is taken over the governing dynamics (6.1)–(6.3), given initial conditions \( x(0), y(0), z(0) \), and prescribed on-ramp flows. The outer maximization is performed over
\( \kappa_i \in \{0, \psi_i\} \) and \( q^T \kappa \geq d \). The cardinality function \( \text{card}(\cdot) \) counts the number of nonzero entries of its vector argument and \( \gamma \) is a non-negative scalar that characterizes the relative importance of the two terms in the objective. Here, the cardinality term serves to additionally enforce a sparse set of failures, i.e., to promote the sparsity of the vector \( \psi - \kappa \).

### 6.3 Formal statement of optimal interdiction & its reformulation as bilinear program

In this section we mathematically formulate optimal interdiction as a max-min problem with a linear objective and linear constraints. We then employ duality to obtain an equivalent formulation as a maximization problem with a bilinear objective and linear constraints.

Stacking the optimization variables \( x(t), y(t), z(t), f(t), w(t) \) into the vector \( u(t) \), and stacking \( u(1), u(2), \ldots, u(\bar{t}) \) to form the vector \( u \),

We state optimal interdiction problem as

\[
\begin{align*}
\text{maximize} & \quad \kappa \\
\text{minimize} & \quad u^T p \\
\text{subject to} & \quad Au = b, \quad Gu \leq H\kappa + h \\
& \quad 0 \leq \kappa \leq \psi, \quad q^T \kappa \geq d
\end{align*}
\]

where \( A, b, G, H, h, p \) are appropriately defined matrices and vectors. Here, we have relaxed each of the constraints \( \kappa_i \in \{0, \psi_i\} \) to \( 0 \leq \kappa_i \leq \psi_i \) and have eliminated the cardinality term that was present in (6.4). However, we will demonstrate in the next section that the solution to (6.5) is indeed sparse and that all but (at most) one of the \( \kappa_i \) belong to \( \{0, \psi_i\} \). This is the main theoretical contribution of this work.

We next employ duality as in [1] to turn the max-min problem (6.5) into a standard maximiza-
tion problem. Problem (6.5) is equivalent to

$$\begin{align*}
\text{maximize} \quad & -b^T \nu - h^T \lambda - \lambda^T H \kappa \\
\text{subject to} \quad & A^T \nu + G^T \lambda = -p, \quad \lambda \geq 0 \\
& 0 \leq \kappa \leq \psi, \quad q^T \kappa \geq d
\end{align*}$$

(6.6)

\(\nu\) : dual variables corresponding to the equality constraint and \(\lambda\) respectively are the dual variables corresponding to the equality and first inequality constraints in (6.5). The objective function in (6.6) is bilinear in the variables \(\lambda\) and \(\kappa\) and is therefore nonconcave. In general it is intractable to find the global maximum of a nonconcave function. In the next section we propose an effective numerical method to solve (6.6), which is the main algorithmic contribution of this work. For small examples, where an exhaustive search is feasible, we demonstrate that the solution found by our algorithm is the same as the globally optimal solution.

### 6.4 Guaranteed sparsity and binary property of failures

**Proposition 3.** When the budget is not enough to fail all cells, there is an optimal solution of

$$\begin{align*}
\text{maximize} \quad & -b^T \nu - h^T \lambda - \lambda^T H \kappa \\
\text{subject to} \quad & A^T \nu + G^T \lambda = -p, \quad \lambda \geq 0 \\
& 0 \leq \kappa \leq \psi, \quad q^T \kappa = d
\end{align*}$$

(6.7)

that solves (6.6). Furthermore, this solution has the property that all but (at most) one of the \(\kappa_i\) belong to \(\{0, \psi\}\).

**PROOF:** See Appendix D.

Proposition 3 demonstrates the sparsity of optimal failures. The attacker orders the nodes in terms of their importance, as determined by (A.20)–(A.21), and fails them fully in descending
order of importance until he has exhausted his budget. Since it is only meaningful that the attacker has limited resources/budget, this results in a sparse set of failed nodes.

6.5 Proposed method for solving problem (6.7)

This section contains our main algorithmic results. We solve the bilinear problem by iteratively updating one set of variables through a gradient-based step and then finding the globally optimal solution in the other set of variables. And our numerical experiments demonstrate that our approach performs better in comparison with methods reported in earlier work [1].

Although the objective function of problem (6.7) is bilinear in the variables, but the constraints on \( \kappa \) and \( \{ \lambda, \nu \} \) are independent. If we fix one set of variables and solve for the other, we can decompose problem (6.7) into two linear programs

\[
\begin{align*}
\text{maximize} \quad & -\lambda^T H \kappa \\
\text{subject to} \quad & 0 \leq \kappa \leq \psi, \quad q^T \kappa = d \quad (6.8)
\end{align*}
\]

and

\[
\begin{align*}
\text{maximize} \quad & -b^T \nu - h^T \lambda - \lambda^T H \kappa \\
\text{subject to} \quad & A^T \nu + G^T \lambda = -p, \quad \lambda \geq 0 \quad (6.9)
\end{align*}
\]

In our experiments we find that if we update the variables by iteratively solving linear programs (6.8) and (6.9) we rapidly converge to a sub-optimal solution of problem (6.7), which inhibits the search for the global optimum.

Motivated by block coordinate gradient descent (BCGD) and block coordinate descent (BCD) [149], we aim to solve problem (6.7) by iterating between updating \( \kappa \) using a gradient-based step and finding the globally optimal solution of \( \{ \lambda, \nu \} \). Generally, projected gradient descent is used as the one-step update for a constraint problem, which first employs gradient descent to update the variable and then obtains its Euclidean projection onto the constraint set. In this chapter we use
the adaptive moment estimation (Adam) algorithm [151] instead of gradient descent in the update of $\kappa$. We will refer to this procedure as the projected Adam algorithm.

The Adam algorithm adds bias-correction terms based on the root mean square prop (RMSProp) [152] and the adaptive gradient (Adagrad) [153] algorithms, and it is demonstrated to be robust and well-suited to a wide range of convex and non-convex problems [151]. In the case of solving (6.7), the momentum term in Adam helps us avoid early convergence to a sub-optimal point. In our experiments, we observe that when we use the projected Adam algorithm our solutions are much better than those found by projected gradient descent.

To begin, we use the fact that maximizing the objective function of problem (6.8) over $\kappa$ is equivalent to

$$\min_{\kappa} \lambda^T H \kappa.$$  \hspace{1cm} (6.10)

In our projected Adam algorithm, we first use Adam to update $\kappa$, for which we compute

$$g^{(k+1)} = \frac{\partial (\lambda^T H \kappa)}{\partial \kappa} = H^T \lambda,$$

$$\rho^{(k+1)} = \beta_1 \rho^{(k)} + (1 - \beta_1) g^{(k+1)},$$

$$\sigma^{(k+1)} = \beta_2 \sigma^{(k)} + (1 - \beta_2) g^{(k+1)} \circ g^{(k+1)},$$

$$\hat{\rho}^{(k+1)} = \frac{\rho^{(k+1)}}{1 - \beta_1^{k+1}}, \hat{\sigma}^{(k+1)} = \frac{\sigma^{(k+1)}}{1 - \beta_2^{k+1}},$$

with the update of $\kappa$ given by

$$\kappa^{(k+1)} = \kappa^{(k)} - \frac{\eta \hat{\rho}^{(k+1)}}{\sqrt{\hat{\sigma}^{(k+1)} + \epsilon} 1},$$  \hspace{1cm} (6.11)

where $\circ$ denotes element-wise vector multiplication and division by vectors is performed element-wise; $\epsilon$ is a parameter with small values to prevent division by zero (generally chosen to be $10^{-8}$), the initial values of $\rho^{(0)}$ and $\sigma^{(0)}$ is zero, and the parameters $\beta_1$ and $\beta_2$ respectively are chosen to be 0.9 and 0.999; $1$ is the column vector of all ones and $\eta$ is the step size of the Adam algorithm.
Next we find the Euclidean projection of $\kappa^{(k+1)}$ onto the constraint set by solving the linear program

$$\begin{align*}
\text{minimize} \quad & \|\kappa - \kappa^{(k+1)}\|_2^2 \\
\text{subject to} \quad & 0 \leq \kappa \leq \psi, \quad q^T\kappa = d.
\end{align*}$$ (6.12)

In every iteration, after solving (6.12) we use $\kappa^{(k+1)}$ to denote the solution of (6.12) rather than the result of (6.11). We then set $\kappa = \kappa^{(k+1)}$ in problem (6.9) and solve the linear program to obtain $\{\lambda^{(k+1)}, \nu^{(k+1)}\}$. This concludes one iteration of our algorithm.

**Algorithm 7** Our proposed algorithm to solve problem (6.7)

1:    initialize $\lambda^{(0)}$
2:    for $k = 0, 1, \ldots, k_{\text{max}}$ do
3:        Set $\lambda = \lambda^{(k)}$, find $\kappa^{(k+1)}$ using projected Adam algorithm (6.11)–(6.12).
4:        Set $\kappa = \kappa^{(k+1)}$, find $\{\lambda^{(k+1)}, \nu^{(k+1)}\}$ by solving linear program (6.9).
5:    end for

Algorithm 7 summarizes our proposed iterative method for solving problem (6.7). In our experiments we observe that the number of needed iterations is reduced if we initialize $\lambda^{(0)}$ using the sub-optimal solution found by solving linear programs (6.8) and (6.9) iteratively, instead of a using a random point in the constraint set. The details of the initialization of $\lambda^{(0)}$ is discussed in Algorithm 8.

**Algorithm 8** Initialization of $\lambda^{(0)}$

1:    given $\kappa^{(0)} = \psi$
2:    for $i = 0, 1, \ldots, i_{\text{max}}$ do
3:        Solve problem (6.9) to find $\lambda^{(i)}$ and $\nu^{(i)}$, set $J^{(i)}_1$ as value of objective function.
4:        Solve problem (6.8) to find $\kappa^{(i+1)}$, set $J^{(i)}_2$ as value of objective function.
5:        if $J^{(i)}_1 = J^{(i+1)}$ and $J^{(i)}_2 = J^{(i)}_1$ then
6:            Break for loop.
7:    end if
8:    end for
6.6 Numerical results

In this section we illustrate the utility of Algorithms 7 and 8 for solving problem (6.7). Our numerical experiments also validate the theoretical results in Section 6.4. For all linear programs we use CVXPY 154, 155, a tool for convex programming in Python. We implement Adam algorithm (6.11) in Tensorflow 156. We apply our approach to two different networks and compare the results with those in [1].

6.6.1 Example 1

We evaluate the utility of our approach using the network shown in Figure 6.1, which is taken from [1]. We prescribe that at each time step 2 units of mass enter nodes 1 and 9 through their respective on-ramps. Node 11 is the sink cell. We take \( \phi = 1.2 \psi \) and \( t = 12 \). The cost, flow-capacity, and initial mass vectors respectively are given by

\[
\begin{align*}
    c &= \begin{bmatrix} 3, 2, 1, 2, 1, 2, 1, 3, 2 \end{bmatrix}^T \\
    \psi &= \begin{bmatrix} 4, 3, 3/2, 3, 3, 3/2, 3, 3/2, 4, 3 \end{bmatrix}^T \\
    x(0) &= \begin{bmatrix} 2, 1, 1, 1, 1/2, 1/2, 1/2, 1/2, 2, 2 \end{bmatrix}^T.
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3,6,8</td>
<td>3,6,8</td>
<td>3,6,8</td>
<td>3,6,8</td>
</tr>
<tr>
<td>5</td>
<td>3,6,8,10</td>
<td>3,6,8,10</td>
<td>3,6,8,10</td>
<td>3,6,8,10</td>
</tr>
</tbody>
</table>

Table 6.1: Comparison of different numerical algorithms for network in Figure 6.1

Table 6.1 shows that for budgets \( e = 1, 3, 5 \) both our numerical method and that proposed in [1] successfully find the globally optimal attack (obtained through exhaustive search).
6.6.2 Example 2

As a second example we use the network shown in Figure 6.2 which is a traffic network adapted by [1] from [2]. We prescribe that at each time step 2 units of mass enter nodes 1,4 and 1 unit of mass enter node 3, all through their respective on-ramps. Node 18 is the sink cell. We take $\phi = 1.2 \psi$ and $\tau = 12$. The cost, flow-capacity, and initial mass vectors respectively are given by

$\begin{align*}
\mathbf{c} &= [3, 2, 2, 3, 2, 2, 3, 2, 2, 2, 2, 3, 1, 3, 2]^T \\
\mathbf{\psi} &= [6, 3, 3, 6, 3, 3, 5, 3, 3, 3, 3, 5, 5, 2, 5, 3]^T \\
\mathbf{x}(0) &= [2, 3, \frac{3}{2}, 3, 3, 3, 3, 3, 3, 2, 2, \frac{3}{2}, 2, 2, 2, 2]^T.
\end{align*}$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>15,17</td>
<td>15, (16)</td>
<td>15, (16)</td>
<td>15,17</td>
</tr>
<tr>
<td>4</td>
<td>15,17</td>
<td>15, (16)</td>
<td>15, (16)</td>
<td>15,17</td>
</tr>
<tr>
<td>6</td>
<td>15,16,17</td>
<td>15,16,17</td>
<td>15,16,17</td>
<td>15,16,17</td>
</tr>
</tbody>
</table>

Table 6.2: Comparison of different numerical algorithms for network in Figure 6.2

Table 6.2 shows that for budget $e = 6$ both our numerical method and that in [1] find the globally optimal attack (obtained through exhaustive search). In this case, the budget is enough to
fully block the network; it is clear that failing nodes 15, 16, 17 is optimal since these failures prevent any mass from leaving the network. For $e = 3, 4$ the problem is more challenging, as the budget is no longer enough to fully block the network. Still, our proposed approach successfully finds the globally optimal attack whereas the method in [1] does not.

### 6.7 Summary

In this chapter, we study the interdiction problem for transportation networks. We prove that the solution to the optimal interdiction problem is both sparse and binary even without any sparsity regularization or constraints in the formulation. We also propose a numerical method to solve the bilinear network interdiction problem and find globally optimal solutions in the small networks we tested.
CHAPTER 7
CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

7.1 Summary

In this dissertation, we discuss the mathematical optimization algorithms for model compression and adversarial learning on DNNs. For model compression, we introduce two different methods. The first method is based on hard constraint, in which the pruning rate in each layer need to be determined at the time when the problem is formulated. The second method is based on soft regularization, in which the pruning rate in each layer can be determined based on the distribution of weights after convergence. For the above two methods, the first one works better if the users have specific requirements on the size of part or all layers. And otherwise the second one is preferable. For adversarial learning, we propose an adversarial attack generation method via convex programming, which works for the DNNs with piecewise linear activation functions, such as ReLU. And we propose a unified min-max optimization framework for the adversarial attack and defense on DNNs over multiple domains.
7.2 Future research directions

7.2.1 Efficient and robust neural architecture search

Neural architecture search (NAS) is an approach for automating the design of artificial neural networks [157]. NAS has been applied to search for the architecture of DNNs which work well for specific tasks. It would be highly desirable to use NAS to search for the DNN architecture which is efficient for hardware implementation and robust to adversarial attacks. By combining the NAS for efficient and robust DNN architecture with weight pruning and adversarial training, we can further improve the implementation efficiency and adversarial robustness of DNNs.

7.2.2 Adversarial learning on graph neural networks

Graph neural networks which apply deep neural networks to graph data have achieved great performance on different tasks, such as traffic speed forecasting [158]. However, the prior works rarely discuss the robustness of graph neural networks. In our work, we propose to evaluate the robustness of graph neural networks on traffic speed prediction when some links in the graph of road networks are removed. First, we propose an algorithm to search the links which are sensitive to the traffic speed prediction. When we constraint the total number of links can be removed from the graph, removing the sensitive links searched by our method leads to much higher prediction error compared with randomly removing the same amount of links.
APPENDIX A

A.1 Proof of Proposition 1, Chapter 4

We need to demonstrate that when condition (4.5) holds, the optimal solution of the convex problem (4.2) satisfies all the constraints in problem (4.1).

Recall that $x^{(k+1)}$, $y^{(k)}_i$, $z^{(k+1)}$ denote the solution of (4.2) at iteration $k$. Since the solution satisfies the constraints, in particular we have

$$y^{(k+1)}_1 = a^{(k)}_1 \circ (W_1x^{(k+1)} + b_1),$$
$$y^{(k+1)}_i = a^{(k)}_i \circ (W_iy^{(k+1)}_{i-1} + b_i), \quad i = 2, \ldots, N - 1.$$  \hspace{1cm} (A.1)

From $a^{(k+1)}_i = a^{(k)}_i$, we conclude that

$$y^{(k+1)}_1 = a^{(k+1)}_1 \circ (W_1x^{(k+1)} + b_1),$$
$$y^{(k+1)}_i = a^{(k+1)}_i \circ (W_iy^{(k+1)}_{i-1} + b_i), \quad i = 2, \ldots, N - 1. \hspace{1cm} (A.2)$$

According to (A.1) and the definition of $y^{[k]}_1$ we can derive that $y^{(k+1)}_1 = y^{[k+1]}_1$. Similarly, from (A.2), the definition of $y^{[k]}_2$, and $y^{(k+1)}_1 = y^{[k+1]}_1$, we obtain $y^{(k+1)}_2 = y^{[k+1]}_2$. This procedure can
be continued to show that $y_i^{(k+1)} = y_i^{[k+1]}$ for $i = 1, 2, \ldots, N - 1$. Therefore, equation (A.2) is equivalent to

$$y_i^{(k+1)} = a_i^{(k+1)} \circ (W_i y_i^{[k+1]} + b_i), \quad i = 2, \ldots, N - 1. \quad (A.3)$$

Now, replacing $k$ with $k + 1$ in the definition of $a_i^{(k)}$, equations (A.1) and (A.3) are respectively equivalent to

$$y_1^{(k+1)} = \sigma(W_1 x^{(k+1)} + b_1),$$
$$y_i^{(k+1)} = \sigma(W_i y_i^{(k+1)} + b_i), \quad i = 2, \ldots, N - 1.$$  

Recalling that the constraints involving $z$ are the same in (4.1) and (4.2), the above argument implies that $x^{(k+1)}$, $y_i^{(k+1)}$, $z^{(k+1)}$ satisfy the constraints in (4.1) and therefore characterize a feasible point. This completes the proof of the proposition.  

\[\blacksquare\]

A Proof of Proposition 2, Chapter 5

\(\ell_1\) norm  When we find the Euclidean projection of \(a\) onto the set \(X\), we solve

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \| \delta - a \|^2 + I_{[\tilde{c}, \hat{c}]}(\delta) \\
\text{subject to} & \quad \| \delta \|_1 \leq \epsilon,
\end{align*}
\]

where \(I_{[\tilde{c}, \hat{c}]}(\cdot)\) is the indicator function of the set \([\tilde{c}, \hat{c}]\). The Langragian of this problem is

\[
L = \frac{1}{2} \| \delta - a \|^2 + I_{[\tilde{c}, \hat{c}]}(\delta) + \lambda_1 (\| \delta \|_1 - \epsilon) \\
= \sum_{i=1}^{d} \left( \frac{1}{2} (\delta_i - a_i)^2 + \lambda_1 |\delta_i| + I_{[\tilde{e}_i, \hat{e}_i]}(\delta_i) \right) - \lambda_1 \epsilon.
\]
The minimizer $\delta^*$ minimizes the Lagrangian, it is obtained by elementwise soft-thresholding

$$\delta_i^* = P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \lambda_1, 0\}).$$

where $x_i$ is the $i$th element of a vector $x$, $P_{[\hat{c}_i, \hat{c}_i]}(\cdot)$ is the clip function over the interval $[\hat{c}_i, \hat{c}_i]$.

The primal, dual feasibility and complementary slackness are

$$\lambda_1 = 0, \|\delta\|_1 = \sum_{i=1}^d |\delta_i| = \sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(a_i)| \leq \epsilon \quad \text{(A.7)}$$

$$\text{or } \lambda_1 > 0, \|\delta\|_1 = \sum_{i=1}^d |\delta_i| = \sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \lambda_1, 0\})| = \epsilon. \quad \text{(A.8)}$$

If $\sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(a_i)| \leq \epsilon$, $\delta_i^* = P_{[\hat{c}_i, \hat{c}_i]}(a_i)$. Otherwise $\delta_i^* = P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \lambda_1, 0\})$, where $\lambda_1$ is given by the root of the equation $\sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \lambda_1, 0\})| = \epsilon$. Bisection method can be used to solve the above equation for $\lambda_1$, starting with the initial interval $(0, \max_i |a_i| - \epsilon/d]$. Since $\sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i) \max \{|a_i| - 0, 0\})| = \sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(a_i)| > \epsilon$ in this case, and $\sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i) \max \{|a_i| - \max_i |a_i| + \epsilon/d, 0\})| \leq \sum_{i=1}^d |P_{[\hat{c}_i, \hat{c}_i]}(\text{sign}(a_i)(\epsilon/d))| \leq \sum_{i=1}^d (\epsilon/d) = \epsilon.$

**$\ell_2$ norm** When we find the Euclidean projection of $a$ onto the set $\mathcal{X}$, we solve

$$\begin{align*}
\min_{\delta} & \quad \|\delta - a\|_2^2 + I_{[\hat{c}, \hat{c}]}(\delta) \\
\text{subject to} & \quad \|\delta\|_2^2 \leq \epsilon^2,
\end{align*} \quad \text{(A.9)}$$

where $I_{[\hat{c}, \hat{c}]}(\cdot)$ is the indicator function of the set $[\hat{c}, \hat{c}]$. The Langragian of this problem is

$$L = \|\delta - a\|_2^2 + I_{[\hat{c}, \hat{c}]}(\delta) + \lambda_2 (\|\delta\|_2^2 - \epsilon^2) \quad \text{(A.10)}$$

$$= \sum_{i=1}^d ((\delta_i - a_i)^2 + \lambda_2 \delta_i^2 + I_{[\hat{c}_i, \hat{c}_i]}(\delta_i)) - \lambda_2 \epsilon^2. \quad \text{(A.11)}$$
The minimizer \( \delta^* \) minimizes the Lagrangian, it is

\[
\delta^*_i = P_{[\check{c}_i, \hat{c}_i]} \left( \frac{1}{\lambda_2 + 1} a_i \right).
\]

The primal, dual feasibility and complementary slackness are

\[
\lambda_2 = 0, \quad \|\delta\|_2^2 = \sum_{i=1}^{d} \delta_i^2 = \sum_{i=1}^{d} (P_{[\check{c}_i, \hat{c}_i]}(a_i))^2 \leq \epsilon^2 \tag{A.12}
\]

or \( \lambda_2 > 0, \quad \|\delta\|_2^2 = \sum_{i=1}^{d} \delta_i^2 = (P_{[\check{c}_i, \hat{c}_i]}(\frac{1}{\lambda_2 + 1} a_i))^2 = \epsilon^2. \tag{A.13} \)

If \( \sum_{i=1}^{d} (P_{[\check{c}_i, \hat{c}_i]}(a_i))^2 \leq \epsilon^2, \quad \delta_i^* = P_{[\check{c}_i, \hat{c}_i]}(a_i). \) Otherwise \( \delta_i^* = P_{[\check{c}_i, \hat{c}_i]} \left( \frac{1}{\lambda_2 + 1} a_i \right), \) where \( \lambda_2 \) is given by the root of the equation \( \sum_{i=1}^{d} (P_{[\check{c}_i, \hat{c}_i]}(\frac{1}{\lambda_2 + 1} a_i))^2 = \epsilon^2. \) Bisection method can be used to solve the above equation for \( \lambda_2, \) starting with the initial interval \( (0, \sqrt{\sum_{i=1}^{d} (a_i)^2 / \epsilon - 1}). \) Since \( \sum_{i=1}^{d} (P_{[\check{c}_i, \hat{c}_i]}(\epsilon a_i / \sqrt{\sum_{i=1}^{d} (a_i)^2}))^2 \leq \epsilon^2 \sum_{i=1}^{d} (a_i)^2 / (\sqrt{\sum_{i=1}^{d} (a_i)^2})^2 = \epsilon^2. \)

**\( \ell_0 \) norm** For \( \ell_0 \) norm in \( \mathcal{X}, \) it is independent to the box constraint. So we can clip \( a \) to the box constraint first, which is \( \delta_i' = P_{[\check{c}_i, \hat{c}_i]}(a_i), \) and then project it onto \( \ell_0 \) norm.

We find the additional Euclidean distance of every element in \( a \) and zero after they are clipped to the box constraint, which is

\[
\eta_i = \begin{cases} 
\sqrt{a_i^2 - (a_i - \hat{c}_i)^2} & a_i < \hat{c}_i \\
\sqrt{a_i^2 - (a_i - \check{c}_i)^2} & a_i > \check{c}_i \\
|a_i| & \text{otherwise}
\end{cases} \tag{A.14}
\]
It can be equivalently written as

\[
\eta_i = \begin{cases} 
\sqrt{2a_i \hat{c}_i - \hat{c}_i^2} & a_i < \hat{c}_i \\
\sqrt{2a_i \hat{c}_i - \hat{c}_i^2} & a_i > \hat{c}_i \\
|a_i| & \text{otherwise}
\end{cases}
\] (A.15)

To derive the Euclidean projection onto \(\ell_0\) norm, we find the \(\epsilon\)-th largest element in \(\eta\) and call it \([\eta]_\epsilon\). We keep the elements whose corresponding \(\eta_i\) is above or equals to \(\epsilon\)-th, and set rest to zeros. The closed-form solution is given by

\[
\delta_i^* = \begin{cases} 
\delta_i' & \eta_i \geq [\eta]_\epsilon \\
0 & \text{otherwise}
\end{cases}
\] (A.16)

\[ \blacksquare \]

**B Proof of Theorem 1, Chapter 5**

Note that the objective function of problem (5.4) is strongly concave w.r.t. \(w\) with parameter \(\gamma\), and has \(\gamma\)-Lipschitz continuous gradients. Moreover, we have \(\|w\|_2 \leq 1\) due to \(w \in \mathcal{P}\). Using these facts and [133, Theorem 1] or [159, Theorem 1] completes the proof. \[ \blacksquare \]

**C Proof of Lemma 1, Chapter 5**

Similar to (5.1), problem (5.13) is equivalent to

\[
\min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{D}} \maximize w \in \mathcal{P} \sum_{i=1}^{K} w_i F_i(\theta).
\] (A.17)

Recall that \(F_i(\theta) := \maximize_{\delta_i \in \mathcal{X}_i} f_{\text{tr}}(\theta, \delta_i; x, y)\), problem can then be written as

\[
\min_{\theta} \mathbb{E}_{(x,y) \in \mathcal{D}} \maximize \sum_{i=1}^{K} [w_i \maximize_{\delta_i \in \mathcal{X}_i} f_{\text{tr}}(\theta, \delta_i; x, y)].
\] (A.18)
According to proof by contradiction, it is clear that problem (A.18) is equivalent to

\[
\min_{\theta} \mathbb{E}_{(x,y) \in D} \max_{w \in \mathcal{P}, \{\delta_i \in \mathcal{X}_i\}} \sum_{i=1}^{K} w_i f_{\text{fl}}(\theta, \delta_i; x, y). \tag{A.19}
\]

\[\square\]

D Proof of Proposition 3, Chapter 5.7

Problem (6.6) is equivalent to

\[
\begin{align*}
\max_{\lambda, \nu} & \max_{\kappa} \quad -b^T \nu - h^T \lambda - \lambda^T H \kappa \\
\text{subject to} & \quad A^T \nu + G^T \lambda = -p, \quad \lambda \geq 0 \\
& \quad 0 \leq \kappa \leq \psi, \quad c^T (1 - \kappa / \psi) \leq e
\end{align*}
\]

in which the inner maximization problem is

\[
\begin{align*}
\max_{\kappa} & \quad -\lambda^T H \kappa \\
\text{subject to} & \quad 0 \leq \kappa \leq \psi, \quad c^T (1 - \kappa / \psi) \leq e.
\end{align*} \tag{A.20}
\]

Setting \( \omega^T = \lambda^T H \), and denoting the \( i \)th element of a vector \( a \) by \( a_i \), the last problem becomes

\[
\begin{align*}
\max_{\kappa} & \quad -\sum_i \omega_i \kappa_i \\
\text{subject to} & \quad 0 \leq \kappa_i \leq \psi_i, \quad i = 1, 2, \ldots \\
& \quad \sum_i c_i (1 - \kappa_i / \psi_i) \leq e
\end{align*}
\]
which is further equivalent to

\[
\begin{align*}
\text{maximize} & \quad \sum_{i} \omega_i (\psi_i - \kappa_i) \\
\text{subject to} & \quad 0 \leq \psi_i - \kappa_i \leq \psi_i, \quad i = 1, 2, \ldots \\
& \quad \sum_i (c_i/\psi_i)(\psi_i - \kappa_i) \leq e.
\end{align*}
\]

Setting \( \theta_i = c_i/\psi_i \) and \( \mu_i = \psi_i - \kappa_i \) the above problem can be rewritten as

\[
\begin{align*}
\text{maximize} & \quad \sum_{i} \omega_i \mu_i \\
\text{subject to} & \quad 0 \leq \mu_i \leq \psi_i, \quad i = 1, 2, \ldots \\
& \quad \sum_i \theta_i \mu_i \leq e.
\end{align*}
\]

Since both \( c_i \) and \( \psi_i \) are positive for every \( i \) then \( \theta_i \) is positive for \( i = 1, 2, \ldots \) and the last problem is equivalent to

\[
\begin{align*}
\text{maximize} & \quad \sum_{i} (\omega_i/\theta_i) \theta_i \mu_i \\
\text{subject to} & \quad 0 \leq \theta_i \mu_i \leq \theta_i \psi_i, \quad i = 1, 2, \ldots \\
& \quad \sum_i \theta_i \mu_i \leq e.
\end{align*}
\]

Setting \( \tau_i = \theta_i \mu_i \) and recalling that \( \theta_i \psi_i = c_i \), the above problem can be rewritten as

\[
\begin{align*}
\text{maximize} & \quad \sum_{i} (\omega_i/\theta_i) \tau_i \\
\text{subject to} & \quad 0 \leq \tau_i \leq c_i, \quad i = 1, 2, \ldots \\
& \quad \sum_i \tau_i \leq e.
\end{align*}
\]

(A.21)

It can be shown that all the elements of \( H \) in (6.5) are non-negative. Since the elements of \( \lambda \) also are non-negative and \( \omega^T = \lambda^T H \), it follows that the elements of \( \omega \) are non-negative and therefore \( \omega_i/\theta_i \) is non-negative for \( i = 1, 2, \ldots \). This implies that the objective function in (A.21) is monotonically non-decreasing in every \( \tau_i \).
When the attack budget is not enough to fail all cells in the network, $e < \sum_i c_i$, from the monotonically non-decreasing property we conclude that there is a solution of (A.21) that satisfies $\sum_i \tau_i = e$. Clearly this implies that there is a solution of (A.20) which satisfies $c^T(1 - \kappa/\psi) = e$, or equivalently $q^T\kappa = d$. This proves that there is an optimal solution of (6.7) that solves (6.6).

Moreover, when $e < \sum_i c_i$, we solve

$$\max_{\tau} \sum_i (\omega_i/\theta_i)\tau_i$$

subject to $0 \leq \tau_i \leq c_i, \ i = 1, 2, \ldots$

$$\sum_i \tau_i = e$$

is given by finding the index $i_1$ for which $\omega_i/\theta_i$ is largest among all $i$ and setting $\tau_{i_1} = c_{i_1}$ if $c_{i_1} \leq e$ and $\tau_{i_1} = e$ if $c_{i_1} > e$. If $c_{i_1} \leq e$ we proceed by finding the index $i_2$ for which $\omega_i/\theta_i$ is second-largest among all $i$ and setting $\tau_{i_2} = c_{i_2}$ if $c_{i_1} + c_{i_2} \leq e$ and $\tau_{i_2} = e - c_{i_1}$ if $c_{i_1} + c_{i_2} > e$. This procedure is repeated until $\sum_i \tau_i = e$. Thus all but (at most) one of the $\tau_i$ belong to $\{0, c_i\}$, with those $\tau_i$ corresponding to the largest values of $\omega_i/\theta_i$ equal to $c_i$ and those $\tau_i$ corresponding to the smallest values of $\omega_i/\theta_i$ equal to 0. This implies that for every $\lambda \geq 0$ there is a solution of

$$\max_{\kappa} -\lambda^T H\kappa$$

subject to $0 \leq \kappa \leq \psi, \ c^T(1 - \kappa/\psi) = e$.

with the property that all but (at most) one of the $\kappa_i$ belong to $\{0, \psi_i\}$, which in turn proves the same property for problem (6.7). The proof of the proposition is now complete.
REFERENCES


NAME OF AUTHOR: Tianyun Zhang

MAJOR: Electrical and Computer Engineering

PLACE OF BIRTH: Zhuzhou, Hunan, China

DATE OF BIRTH: May 27, 1993

DEGREE AWARDED:

B.E., 2015, Huazhong University of Science and Technology, Wuhan, China