8-31-2021

# Gradient free sign activation zero one loss neural networks for adversarially robust classification

Yunzhe Xue
*New Jersey Institute of Technology*

# Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

## ABSTRACT

## GRADIENT FREE SIGN ACTIVATION ZERO ONE LOSS NEURAL NETWORKS FOR ADVERSARIALLY ROBUST CLASSIFICATION

**by**
**Yunzhe Xue**

The zero-one loss function is less sensitive to outliers than convex surrogate losses such as hinge and cross-entropy. However, as a non-convex function, it has a large number of local minima, and its undifferentiable attribute makes it impossible to use backpropagation, a method widely used in training current state-of-the-art neural networks. When zero-one loss is applied to deep neural networks, the entire training process becomes challenging. On the other hand, a massive non-unique solution probably also brings different decision boundaries when optimizing zero-one loss, making it possible to fight against transferable adversarial examples, which is a common weakness in deep learning neural network models.

This dissertation introduces a stochastic coordinate descent to optimize the linear classification model based on zero-one loss. Moreover, its variants are successfully applied to multi-layer neural networks using sign activation and multi-layer convolutional neural networks to obtain higher image classification performance. In some image benchmark tests, the stochastic coordinate descent method achieves accuracy close to that of the stochastic gradient descent method. At the same time, some heuristic techniques are used, such as random node optimization, feature pool, warm start, step training, additional backpropagation penetration, and other methods to speed up training and save memory usage. Furthermore, the model's adversarial robustness is analyzed by conducting white-box attacks, decision boundary attacks, and comparing zero-one loss models to those using more traditional loss functions such as cross-entropy.

# GRADIENT FREE SIGN ACTIVATION ZERO ONE LOSS NEURAL NETWORKS FOR ADVERSARIALLY ROBUST CLASSIFICATION

**by**
**Yunzhe Xue**

**A Dissertation**
**Submitted to the Faculty of**
**New Jersey Institute of Technology**
**in Partial Fulfillment of the Requirements for the Degree of**
**Doctor of Philosophy in Computer Science**

**Department of Computer Science**

**August 2021**

## APPROVAL PAGE

## GRADIENT FREE SIGN ACTIVATION ZERO ONE LOSS NEURAL NETWORKS FOR ADVERSARIALLY ROBUST CLASSIFICATION

**Yunzhe Xue**

---

Dr. Usman W. Roshan, Dissertation Advisor — Date
Associate Professor of Computer Science, NJIT

---

Dr. Zhi Wei, Committee Member — Date
Professor of Computer Science, NJIT

---

Dr. Iannis Koutis, Committee Member — Date
Associate Professor of Computer Science, NJIT

---

Dr. Hai Phan, Committee Member — Date
Assistant Professor of Informatics, NJIT

---

Dr. Ji Meng Loh, Committee Member — Date
Associate Professor of Mathematical Sciences, NJIT

---

Dr. William Graves, Committee Member — Date
Associate Professor of Psychology, Rutgers University - Newark

## BIOGRAPHICAL SKETCH

**Author:** Yunzhe Xue

**Degree:** Doctor of Philosophy

**Date:** August 2021

**Undergraduate and Graduate Education:**

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2021
- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2015
- Bachelor of Software Engineering
  East China Jiaotong University, Nanchang, Jiangxi, China, 2013

**Major:** Computer Science

**Presentations and Publications:**

Yunzhe Xue, Meiyan Xie, Usman Roshan, "Defending against substitute model black box adversarial attacks with the 01 loss," *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020.

Yunzhe Xue, Meiyan Xie, Usman Roshan, "On the transferability of adversarial examples between convex and 01 loss models," *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020.

Yunzhe Xue, Meiyan Xie, Usman Roshan, "Towards adversarial robustness with 01 loss neural networks," *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020.

Zhibo Yang, Yanan Yang, Yunzhe Xue, Frank Y Shih, Justin Ady, Usman Roshan, "Accurate and adversarially robust classification of medical images and ECG time-series with gradient-free trained sign activation neural networks," *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2020.

Abdulrhman Aljouie, Yunzhe Xue, Meiyan Xie, Usman Roshan, "Challenges in predicting glioma survival time in multi-modal deep networks," *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 2020.

Meiyan Xie, Yunzhu Li, Yunzhe Xue, Lauren Huntress, William Beckerman, Saum A Rahimi, Justin W Ady, Usman W Roshan, "Two-stage and dual-decoder convolutional U-Net ensembles for reliable vessel and plaque segmentation in carotid ultrasound images," *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, 2020.

Yanan Yang, Fadi G Farhat, Yunzhe Xue, Frank Y Shih, Usman Roshan, "Classification of Histopathology Images with Random Depthwise Convolutional Neural Networks," *7th International Conference on Bioinformatics Research and Applications*, 2020.

Yunzhe Xue, Yanan Yang, Fadi Farhat, Frank Y. Shih, Olga Boukrina, A. M. Barrett, Jeffrey R. Binder, William W. Graves, and Usman W. Roshan, "Brain tumor classification with tumor segmentations and a dual path residual convolutional neural network from MRI and pathology images," *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, Vol, 11993, pp 360-367, 2020.

Yunzhe Xue, Yanan Yang, Fadi Farhat, Frank Y. Shih, Olga Boukrina, A. M. Barrett, Jeffrey R. Binder, William W. Graves, and Usman W. Roshan, "A Multi-path Decoder Network for Brain Tumor Segmentation," *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries*, Vol, 11993, pp 255-265, 2020.

Meiyan Xie, Yunzhu Li, Yunzhe Xue, Randy Shafritz, Saum A. Rahimi, Justin W. Ady, and Usman W. Roshan, "Vessel lumen segmentation in internal carotid artery ultrasounds with deep convolutional neural networks," *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp 492-506, 2020.

Yunzhe Xue, Yanan Yang, Fadi Farhat, Frank Y. Shih, Olga Boukrina, A. M. Barrett, Jeffrey R. Binder, Usman W. Roshan, and William W. Graves, "A multi-path 2.5 dimensional convolutional neural network system for segmenting stroke lesions in brain MRI images," *NeuroImage: Clinical*, Vol, 25, 2019.

Meiyan Xie, Yunzhe Xue, and Usman Roshan, "Stochastic coordinate descent for 0/1 loss and its sensitivity to adversarial attacks," *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019.

Yunzhe Xue, Yanan Yang, Fadi Farhat, Frank Y. Shih, Olga Boukrina, A. M. Barrett, Jeffrey R. Binder, William W. Graves, and Usman W. Roshan, "A fully 3D multi-path convolutional neural network with feature fusion and feature weighting for automatic lesion identification in brain MRI images," *ML4H: Machine Learning for Health, Workshop at NeurIPS*, 2019.

Yunzhe Xue and Usman Roshan, "Image classification and retrieval with random depthwise signed convolutional neural networks," *Lecture Notes in Computer Science*, pp 492-506, 2019.

*To my family.*

## ACKNOWLEDGMENT

It is a genuine pleasure to express my deep sense of thanks and gratitude to my advisor, colleague, and my friend forever, Dr. Usman Roshan. Your rigorousness, passion, and perseverance in the research process has shown me the outstanding character of a scientific researcher. Behind is the story of us working together towards the same goal, bravely challenging problems, and committing impactful work.

I want to thank the dissertation committee members: Zhi Wei, Iannis Koutis, Hai Phan, Ji Meng Loh, and William Graves. Thank you for your guidance on my research. I truly enjoy working with you.

I would like to thank the Computer Science Department for providing me a teaching assistantship. I would also like to thank the Vertex Inc. for providing me research assistantship. These support are greatly appreciated.

In addition, I would like to express my sincere gratitude to my labmates: Meiyan Xie, Yanan Yang, and Fadi Farhat for your technique supports and encouragement during these four years. I would like to thank NJIT IST Academic and Research Computing Systems (ARCS) administrator for providing reliable service and high performance computing resources as well.

I heartly appreciate my family, Mr. Jian Xue and Mrs. Lianxiang Wu, my best friend, Mr. Xinglei Jiang, and Ms. Min Zhang. You are my strong backing, the source of my strength, and my harbor of tenderness.

## TABLE OF CONTENTS

**TABLE OF CONTENTS**
**(Continued)**

**Chapter** **Page**

**TABLE OF CONTENTS**
**(Continued)**

**Chapter** **Page**

## LIST OF TABLES

## LIST OF TABLES
## (Continued)

**Table** **Page**

## LIST OF TABLES
## (Continued)

**Table** **Page**

## LIST OF FIGURES

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

## LIST OF FIGURES
## (Continued)

**Figure** **Page**

# CHAPTER 1

# INTRODUCTION

Machine learning has been widely used in many fields such as image recognition and detection, face recognition, autonomous driving, speech recognition and translation. Similar to the situation faced by traditional rule-based programming software systems, these advanced machine learning applications inevitably have unique loopholes, namely adversarial examples. Unlike traditional vulnerabilities, these adversarial examples are not caused by incorrect programming of machine learning models. Researchers expect that machine learning models can learn potential data rules based on our given data. Nevertheless, due to the lack of a large amount of training data and human logical thinking as a guide, it is difficult for machine learning models and even for deep learning models to obtain human-level thinking modes. In a small range of data, some interference that would not affect human's understanding or recognition of the subject can make a trained machine learning model to conduct misjudgments, even though its recognition accuracy on clean data has reached the human level. Based on this phenomenon, we can conclude that in the future, some groups of people who have a good understanding of machine learning models will be able to attack commercialized models and gain illegal profits.

Although there are published papers on counter attack defense every year, most of their algorithms can only play a defensive role against one or a certain type of attack. With the emergence of new attack methods, the problem of machine learning model's weak robustness to adversarial samples has not been fundamentally solved. In other words, if a model is only trained on clean data, but without having adversarial training against adversarial samples conducted, it is likely that it will not be resistant to most attack methods. However, through adversarial training, they are likely to be

able to defend against a certain type of attack method, and their robustness will be greatly improved [57, 24].

There are many machine learning algorithms, including but not limited to, linear regression, logistic regression, support vector machine, multiple layers perceptron, and tree methods such as decision tree and random forest. Most traditional machine learning methods do not have complex logical structures and nonlinear mappings. In recent years, the rapid development of deep learning has enabled multi-layer models to achieve excellent performance. Such models have complex logical structures and multiple non-linear mappings.

There are two main common types of adversarial attacks: black-box attacks and white-box attacks. A white-box attack is an attack ran when all the information of the target model is available. A black-box attack runs when the target model information cannot be obtained, and the output can only be obtained through feeding input data in the model. White-box attacks mainly use the deep learning model mapping function's derivability and inject small changes to the source data to increase the loss function value on the data to obtain incorrect solutions. When the mapping function is not differentiable, the derivative can be solved by an approximate method. The mainstream situation of black-box attacks is divided into two categories, one is related to transferability, and another one is to obtain the minimum distortion distance or the minimum number of queries. Deep learning models are generally based on backpropagation methods to solve similar loss functions. If these models are trained on similar tasks or similar datasets, their adversarial examples are likely to be transferable between each other. For example, multiple models with similar structures or different models are trained for the same classification task, and a certain attack method can be used to synthesize adversarial samples for a certain model. These adversarial samples can not only deceive the source model, but also are likely to be effective for other models. The reason is that based on the same loss function, the

approximate optimal solution obtained by backpropagation is expected to be similar. That is, the mapping function from the source data to the probability output of different categories is similar. Based on this assumption, if there is a certain loss function, or a solution method, by which solving a certain task can result in multiple inconsistent or dissimilar solutions, then it can alleviate this problem to a certain extent. 01 loss is such a loss function. For black-box attacks, common attack methods require constant access to the model to test on the decision boundary. If the mapping function of the target model is not smooth, it will inevitably increase the difficulty of the search, so the robustness can be improved [45]. The 01 activation function causes a large number of local minima in the solving process, and the mapping function is extremely unsmooth. Based on the speculation mentioned above, the model using Sign activation function and zero-one loss can naturally possess better robustness than the derivative activation function and convex function loss model. However, this combination makes the solving process noticeably difficult, especially when the deep model is widely used now. Optimizing the deep model to have an accuracy close to that of the differentiable deep model becomes a challenge.

In the second chapter of this paper, we first introduced several commonly used loss functions, activation functions, and optimization methods in machine learning or deep learning. Furthermore, we introduced what adversarial examples and the definition of model robustness are. Lastly, the classic white-box attack and black-box attack methods are introduced.

In the third chapter, we discussed the application of 01 loss in linear and non-linear classifier, and the loss function is solved by stochastic coordinate descent. In experiments, we evaluated the effect of several hyperparameters on accuracy in the stochastic coordinate descent experiment. The performance of stochastic coordinate descent and stochastic gradient descent under the same network structure, and the

influence of activation on the model are compared. This is a trial of stochastic coordinate descent on a shallow neural network.

Chapter 4 introduces the problems that arise from directly applying stochastic coordinate descent to convolutional neural networks (CNN). In response to these problems, the training method has been improved. The main changes are cancellation of bias, multi-phase training, and additional backpropagation penetration strategy. The model performance is shown on CIFAR10 and STL10 datasets. It is verified that the modified stochastic coordinate descent can be used to optimize deeper convolutional neural networks.

Chapter 5 shows the robustness of Sign activation Multiple Layers Perceptron (MLP) and deep CNN optimized by stochastic coordinate descent on image datasets. Experiments include white-box attacks based on Fast Gradient Sign Method (FGSM) and Project Gradient Descent (PGD), transferability of adversarial samples from different models, and decision boundary attacks.

The previous five chapters are summarized in Chapter 6.

# CHAPTER 2

# BACKGROUND

A loss function is a kind of function in mathematical optimization theory that maps one or more value or variables to another real number [49], which intuitively represents some"cost" associated with the event. "An optimization method seeks to reduce loss calculated from a given loss function. An objective function can be a loss function (in a specific domain, it is called a return function, a profit function, a utility function, a fitness function, etc.). In this case, it is necessary to be maximized." [1] In machine learning, the loss function is often used to evaluate the degree of difference between the predicted value of the model and the true value. The better the loss function, the stronger the performance of the model will be obtained after training. In classification tasks, common loss functions and their advantages and disadvantages are as follows.

## 2.1 Loss Function

### 2.1.1 Zero-One Loss

zero-one loss means that the loss is 1 when the predicted value is not equal to the true value, otherwise it is 0. It is often used to directly judge the number of errors in classification prediction. As a non-convex function, it is very difficult to solve [34].

$$L(y, f(x)) = \begin{cases} 1, & y \neq f(x) \\ 0, & y = f(x) \end{cases} \tag{2.1}$$

### 2.1.2 Cross Entropy Loss

$$loss = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] \tag{2.2}$$

[1] https://en.wikipedia.org/wiki/Loss_functioncite_note-Raschka_2019_p.-1

In the standard formula, $x$ represents the sample, $y$ represents the actual category, $a$ represents the probability predicted by the model, and $n$ represents the total number of samples. In the binary classification, the probability is usually obtained by the sigmoid function, in the multi-classification, the probability is usually obtained by the softmax [17], and the loss function is

$$loss = -\frac{1}{n}\sum_{i} y_i \ln a_i \tag{2.3}$$

### 2.1.3 Hinge Loss

Hinge loss means that if the data is classified correctly, the loss is 0, otherwise it is $1 - yf(x)$. SVM often uses this loss function. Hinge loss formula is below

$$L(y, f(x)) = max(0, 1 - yf(x)) \tag{2.4}$$

## 2.2 Activation Function

**Activation Function** is a function that runs on the neurons of the artificial neural network and serves to map the input of the neuron to the output.

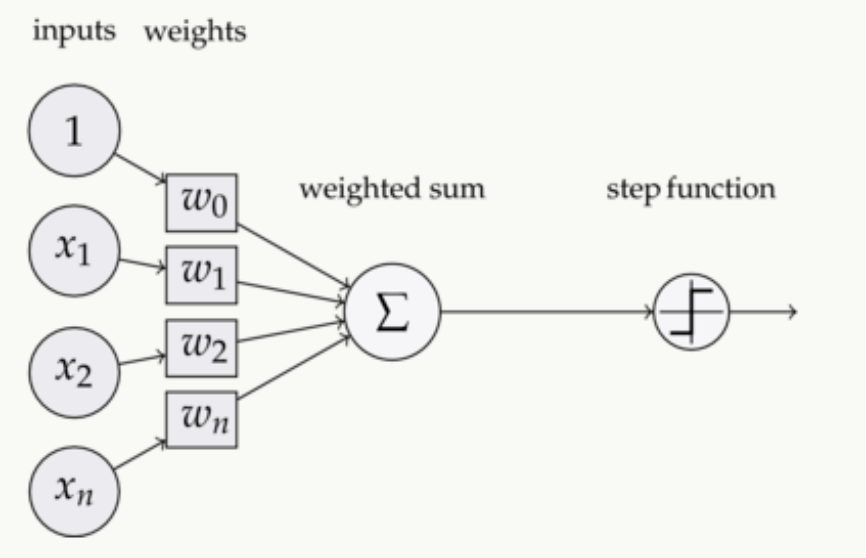


**Figure 2.1** Activation function.

Activation functions are essential for learning and understanding complex non-linear functions in artificial neural networks models. Using an activation function is a major way in which neurons in an artificial neural network roughly approximate biological neurons. As shown in Figure 2.1, the inputs are weighted and summed in the neuron, and then a function is applied. This function is the activation function, which is introduced to increase the nonlinearity of the neural network model. Each layer without activation function is equivalent to matrix multiplication. If the activation function is not used, the output of each layer is a linear mapping of the upper layer input. No matter how many layers of the neural network there are, the output is a linear combination of inputs. This situation is the most primitive Perceptron. If used, the activation function introduces a non-linear factor to the neuron. The neural network can approximate any non-linear function arbitrarily [39].

### 2.2.1 ReLU (Recitified Linear Unit)

Rectifying activation functions (ReLU) were used to separate specific excitation and unspecific inhibition in the neural abstraction pyramid, which was trained in a supervised way to learn several computer vision tasks. In 2011, the apply of the rectifier as a non-linearity activation has been shown to effectively train deep supervised neural networks without requiring unsupervised pre-training. Rectified linear units [26], compared to sigmoid function or similar activation functions, allow faster and effective training of deep neural architectures on large and complex datasets. The function is

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.5}$$

and derivative form is

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.6}$$

### 2.2.2 Sign

Sign function or binary step function is a kind of threshold-based activation function which activate neuron if projection bigger than a certain threshold and deactivate the neuron if projection below the threshold [21]. The function is

$$f(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{2.7}$$

and derivative form is

$$f(x) = \begin{cases} 0, & x \neq 0 \\ ?, & x = 0 \end{cases} \tag{2.8}$$

This step function activation can hardly be applied if there are multiple classes to deal with, but works well in binary classification.

## 2.3 Optimization Method

### 2.3.1 Gradient Descent

Gradient descent is one of iterative methods that can be applied to solve linear or nonlinear least squares problems. When searching the model parameters of machine learning algorithms, that is, unconstrained optimization problems, Gradient Descent is widely used, and another commonly used method is the least squares method. The gradient descent method could be applied to acquire the minimum value of loss function iteratively, to obtain the minimized loss function and model parameter

values. Conversely, if we need to achieve the maximum value of loss functions, then we need to use the gradient ascent method. In machine learning, two gradient descent methods have been developed based on the basic gradient descent method, namely stochastic gradient descent method [6, 69] and batch gradient descent method.

### 2.3.2 Coordinate Descent

Coordinate Descent is a optimization algorithm does not require gradient information. In each iteration of the optimization process, searching in one dimension is performed along a coordinate direction at the current position to find a local minimum of a given function. In the whole process, different coordinate directions are searched cyclically. For inseparable functions, Coordinate Descent may hardly find the optimal solution in a small number of iterations. In order to accelerate the convergence, an appropriate coordinate system can be used. For example, a new coordinate system with as little correlation as possible between the coordinates is obtained through principal component analysis [61, 22, 59].

## 2.4 Adversarial Attack

In recent years, many works [28, 30, 56] shows that deep neural network (DNN) are vulnerable to adversarial examples, not matter in image, text, audio classification, or graph application area [18, 33, 70, 52]. As a result, the existence of adversarial attacks has admonished researchers against directly adopting DNNs in safety-critical tasks in all machine learning application fields. Meanwhile, many studies tried to find countermeasures for preventing deep neural network from these adversarial examples' threat, such as *Gradient Masking* [48, 2], *Robust Optimization* [41, 36], and *Adversary Detection* [9, 66]. In other words, studying adversarial examples and their countermeasures is helpful for us to understand and improve DNN consequently.

There are works [28, 32] could help us earn more insight into deep neural networks by explaining and interpreting the existence of adversarial examples of DNNs [65].

### 2.4.1 Adversarial Examples

Adversarial example is a case with slightly intentional feature perturbations that cause a machine learning or deep learning model to make incorrect predictions [28]. Adversarial examples could cause deep learning models vulnerable to attacks, as in the following scenarios: A autonomous car crashed into another vehicle because it ignored a stop sign. The reason that it was ignored was because a picture was placed over the sign, which made it look like a little dirt for humans like Figure 2.3 shows, on the purpose of faking it to appear like a parking prohibition sign in the view of sign recognition software of the car. A spam detector failed to classify an email as spam. The spam email was designed to resemble a normal email, but with the intention of cheating the recipient.

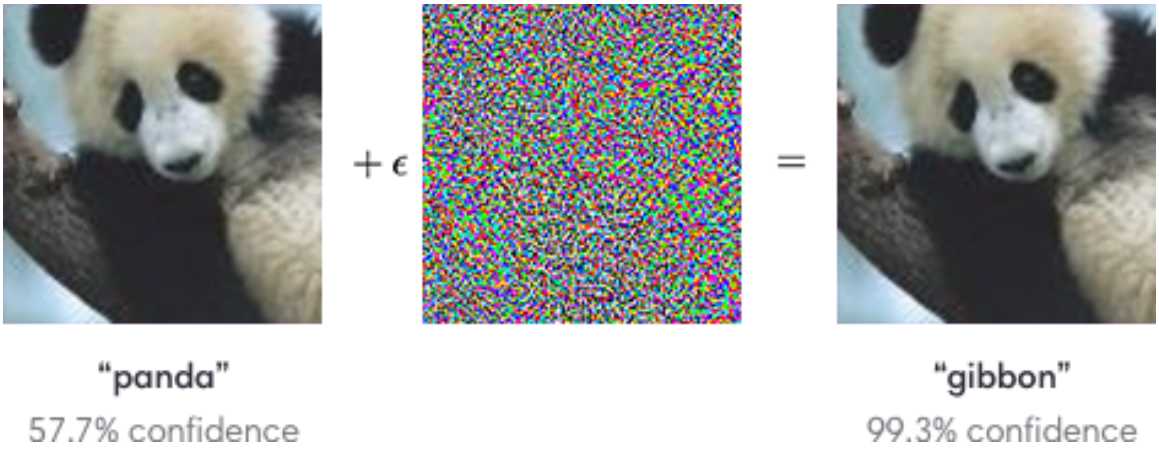


**Figure 2.2** An adversarial input, overlaid on a typical image, can cause a classifier to miscategorize a panda as a gibbon.
*Source: [28].*


**Figure 2.3** Sample of physical adversarial examples against deep neural networks. All the STOP signs were misclassified as speed limit 45 sign, and right turn signs were misclassified as STOP sign.
*Source: [20].*

### 2.4.2 Adversarial Attack

The adversarial attack is injecting small distortion into source data to fool a machine learning system. Suppose there is a classifier and $F$ represents the projection from source data $x$ to corresponding output $y$, as well as $F(x) = y$. We want to inject some noise to $x$, and let the new fake data to be $x'$. The euclidean distance between $x'$ and $x$ should be smaller than a threshold $\epsilon$ to avoid the injection been detected. Finally, the classifier will predict the data $x'$ as $y$‘ but $y' \neq y$.

$$\begin{aligned} &\text{find } x' \text{ satisfying } ||x' - x|| \leq \epsilon \\ &\text{such that } F(x') \neq y \end{aligned} \tag{2.9}$$

An ideal adversarial attack method could generate a lot of adversarial examples with minor distortion undetected by the human in a short time [46]. So its properties should be high attack success rate, fast generating speed, and small distortion.

## 2.5 Conclusion

In this chapter, we introduced the basic information of loss functions, activation functions, and optimization methods that will be involved in our experiments in later chapters. In addition, we described what an adversarial example is and the definition of the adversarial attack.

# CHAPTER 3

# OPTIMIZING LINEAR CLASSIFIER AND MULTIPLE LAYERS PERCEPTRON BY STOCHASTIC COORDINATE DESCENT

## 3.1 Loss Function

### 3.1.1 Loss Function For Linear Classifier

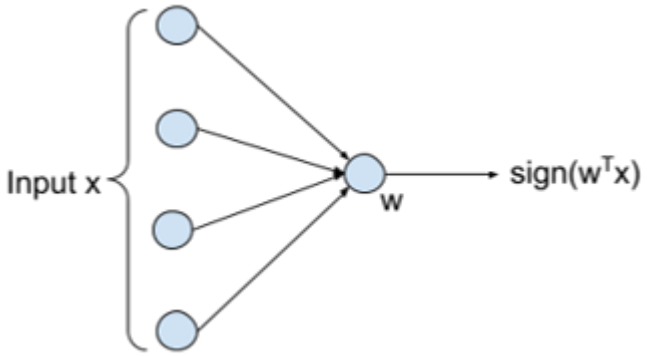


**Figure 3.1** Zero-one loss for linear classifier.

To decide the hyperplane with minimum number of misclassifications in a binary classification task is known to be NP-hard [4]. In mainstream machine learning literature this is called minimizing the zero-one loss [51] given in Objective 3.1,

$$\frac{1}{2n} \arg\min_{w,w_0} \sum_{i}^{n} (1 - sign(y_i(w^T x_i + w_0))) \tag{3.1}$$

where $w \in R^d$, $w_0 \in R$ is the hyperplane, and $x_i \in R^d, y_i \in \{+1, -1\}. \forall i = 0...n-1$ are training data. Popular linear classifiers such as the linear support vector machine, perceptron, and logistic regression [1] can be considered as convex approximations to this problem that yield fast gradient descent solutions [3]. However, they are also more sensitive to outliers than the zero-one loss [3, 44, 64] and more prone to mislabeled data than zero-one loss [42, 25, 40].

### 3.1.2 Loss Function For Multiple Layers Perceptron

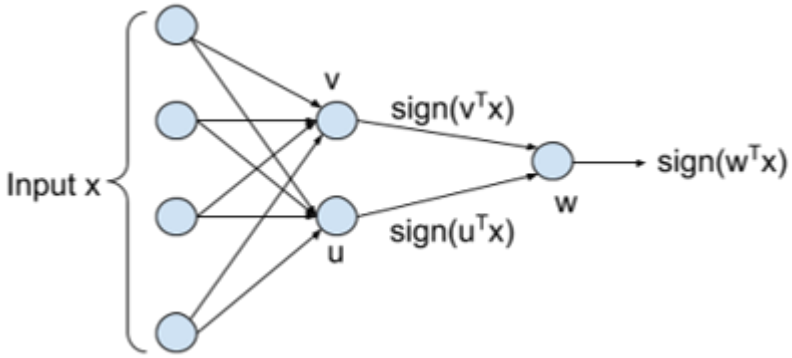


**Figure 3.2** Zero-one loss for two layers neural network.

Here, zero-one loss is extended to a simple two-layer neural network with $k$ hidden nodes, and sign activation that we call the MLP01loss. This objective for binary classification can be given as

$$\frac{1}{2n} \underset{W,W_0,w,w_0}{\arg\min} \sum_{i}^{n} (1 - sign(y_i(w^T(sign(W^T x_i + W_0)) + w_0))) \tag{3.2}$$

where $W \in R^{d \times k}$, $W_0 \in R^k$ are the hidden layer parameters, $w \in R^k, w_0 \in R$ are the final layer node parameters, $x_i \in R^d, y_i \in \{+1, -1\}. \forall i = 0...n-1$ are the training data, and $sign(v \in R^k) = (sign(v_0), sign(v_1), ..., sign(v_{k-1}))$. While this is a straightforward model to define, optimizing it is a different story altogether. Optimizing even a single node is NP-hard which makes optimizing this network much harder. Note that our weights are real numbers as opposed to binarized neural networks whose weights are constrained to be +1 and -1 or 1 and 0 [23, 14, 50].

## 3.2 Algorithm Description

### 3.2.1 Stochastic Coordinate Descent For Linear Classifier

**Algorithm 1** is our core coordinate descent algorithm. We perform just one iterative update instead of convergence. We find this to be more accurate and faster [68].

**Algorithm 1** Stochastic Coordinate Descent

**Input:** Data (feature vectors) $x_i \in R^d$ for $i = 0..n-1$ with labels $y_i \in \{+1, -1\}$, $w_{inc} \in R$, size of pooled features to update $k$, vector $w \in R^d$ and $w_0 \in R$

**Output:** Vector $w \in R^d$ and $w_0 \in R$

**Procedure:**

1. Initialization: If $w$ is null then let each feature $w_i$ of $w$ be normally drawn from $N(0, 1)$. We set $\|w\| = 1$ and throughout our search ensure that $\|w\| = 1$ by renormalizing each time $w$ changes.

2. Let the number of misclassified points with negative $w^T x_i$ be $errorminus = 0$ and those with positive $w^T x_i$ be $errorplus = 0$. These are later used in the Optimal Threshold algorithm called $Opt$ (see below) for fast update of our objective.

3. Compute the initial data projection $w^T x_i, \forall i = 0..n-1$, sort the projection with insertion sort, and initialize $(w_0, obj) = Opt(w^T x, y, 0, n-1)$. We also record the value of $j$ for the optimal $w_0 = (w^T x_j + w^T x_{j+1})/2$.

4. Set $prevobj = \infty$, $done = 0$.

**while** done != 1 **do**

Set $prevobj = obj$

Randomly pick $k$ of the $d$ feature indices.

**for all** selected features $w_i$ we update them **do**

1. Assume the optimal $w_0 = (w^T x_j + w^T x_{j+1})/2$

2. Set $start = w^T x_{j-10}$ and $end = w^T x_{j+10}$

3. Modify coordinate $w_i$ by $w_{inc}$, compute data projection $w^T x_i \forall i = 0..n-1$, and sort the projection with insertion sort

4. Set $(w_0, obj) = Opt(w^T x, y, start, end)$ and record this value for feature $w_i$

5. Reset $w_0$ to try the next coordinate

**end for**

Pick the coordinate whose update gives the largest decrease in the objective and set $(w_0, obj)$ to the values given by the best coordinate with ties decided randomly.

Set $done = 1$

**end while**=0

**Algorithm 2** Optimal Threshold $w_0$ and Zero-One Loss Objective Value

**Input:** $w^T x_i \in R^d$ for $i = 0..n-1$ with labels $y_i \in \{+1, -1\}$, *start*, *end*

**Output:** Optimal $w_0 \in R$ with minimum (balanced) 01 loss and the loss value *obj*

**Procedure:**

1: **for** $i = start$ to $end - 1$ **do**

2: $w_0' = \frac{w^T x_i + w^T x_{i+1}}{2}$

3: **if** $y_i(w^T x_i + w_0') == 0$ **then**

4: If $y_i == 1$ then errorplus++

5: **else if** $y_i(w^T x_i + w_0') > 0$ **then**

6: If $y_i == 1$ then errorplus−− else errorminus−−

7: **else if** $y_i(w^T x_i + w_0') < 0$ **then**

8: If $y_i == 1$ then errorplus++ else errorminus++

9: **end if**

10: If $obj' = \frac{errorplus + errorminus}{n}$ is lower than current best objective *obj* then $obj = obj'$ and $w_0 = w_0'$.

11: **end for**

12: **return** $(w_0, obj)$ =0

**Algorithm 2** is our fast algorithm to update $w_0$ and the model objective. Once we have the objective for $w_0 = \frac{w^T x_i + w^T x_{i+1}}{2}$ we can calculate it for $w_0 = \frac{w^T x_{i+1} + w^T x_{i+2}}{2}$ in constant time.

**Algorithm 3** Stochastic Coordinate Descent For Linear Zero-One Loss

**Input:** Data (feature vectors) $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, number of votes $rr \in N$ (Natural numbers), number of iterations per votet $it \in N$ (Natural numbers), batch size as a percent of training data $p \in [0, 1]$, and $w_{inc} \in R$

**Output:** Total of $rr$ pairs of ($bestw \in R^d$, $bestw_0 \in R$) after each vote

**Procedure:**

Set $j = 0$

**while** $j < rr$ **do**

1. Set $bestw = null, bestw_0 = null, bestloss = \infty$

**for** $i = 0$ to $it$ **do**

1. Randomly pick $p$ percent of rows as input training data to the coordinate descent algorithm and run it to completion starting with the values of $w$ and $w_0$ from the previous call to it (if $i == 0$ we set $w = null, w_0 = null$).

2. In the next step we calculate the linear 01 loss objective on the full input training set

**if** $objective(w, w_0) < objective(bestw, bestw_0)$ **then**

Set $bestw = w$, $bestw_0 = w_0$, and $bestloss = objective(w, w_0)$

**end if**

**end for**

2. Output $bestw$ and $bestw_0$

3. Set $j = j + 1$.

**end while**

We output all $(bestw, bestw_0)$ pairs across the votes. We can use the pair with the lowest objective or the majority vote of all pairs for prediction. =0

**Algorithm 3** is our stochastic descent search performs coordinate descent for the model parameters $w, w_0$. We keep track of the best parameters across iterations by evaluating the model objective on the full dataset after each iteration.

### 3.2.2 Stochastic Coordinate Descent For Multiple Layers Perceptron

**Algorithm 4** is our method to optimize two layer neural network. Our stochastic descent search performs coordinate descent on the final node and then a random

**Algorithm 4** Stochastic Coordinate Descent for Two Layer Zero-One Loss Network

**Input:** Data (feature vectors) $x_i \in R^d$ with labels $y_i \in \{+1, -1\}$, number of hidden nodes $h$, number of votes $rr \in N$ (Natural numbers), number of iterations per vote $it \in N$, batch size as a percent of training data $p \in [0, 1]$, $w_{inc} \in R$ and $w_{inc2} \in R$

**Output:** Total of $rr$ sets of $(bestW \in R^{k \times d}, bestW_0 \in R^k, bestw \in R^k, bestw_0 \in R)$ after each vote

**Procedure:**

1. Initialize all network weights $W, w$ to random values from the Normal distribution $N(0, 1)$.

2. Set network thresholds $W_0$ to the median projection value on their corresponding weight vectors and $w_0$ to the projection value that minimizes our network objective.

**while** $j < rr$ **do**

Set $bestW = null, bestW_0 = null, bestw = null, bestw_0 = null, bestloss = \infty$

**for** $i = 0$ to $it$ **do**

Randomly pick $p$ percent of rows as input training data.

Run the Coordinate Descent Algorithm 1 on the final output node $w$ to completion starting with the values of $w$ and $w_0$ from the previous call to it (if $i == 0$ we set $w = null$). We use learning rate $w_{inc2}$ in the coordinate descent.

Run the Coordinate Descent Algorithm 1 on a randomly selected hidden node $w_k$ ($k^{th}$ column in $W$) starting with the values of $w_k$ and $w_{k0}$ ($k^{th}$ entry in $W_0$) from the previous call to it (if $i == 0$ we set $w_k = null$). We use learning rate $w_{inc}$ in the coordinate descent for the hidden nodes.

Calculate the two layer network 01 loss objective on the full input training set

**if** $objective(W, W_0, w, w_0) < objective(bestW, bestW_0, bestw, bestw_0)$ **then**

Set $bestW = W$, $bestW_0 = W_0$, $bestw = w$, $bestw_0 = w_0$, and $bestloss = objective(bestW, bestW_0, bestw, bestw_0)$

**end if**

**end for**

Output ($bestW$, $bestW_0$, $bestw$, $bestw_0$)

Set $j = j + 1$.

**end while**

We output all sets of $(bestW, bestW_0, bestw, bestw_0)$ across the votes. We can use the first set or the majority vote of all sets for predictions.

=0

hidden node in each iteration. We keep track of the best parameters across iterations by evaluating the model objective on the full dataset after each iteration.

## 3.3 Experiments

We built a two-layer neural network as a baseline model. This neural network uses zero-one loss as the loss function and contains 20 nodes in the hidden layer - all of the nodes use Sign as the activation function. Since image data is currently the primary research object of machine learning model adversarial robustness, we selected class 0 and class 1 from CIFAR10 to train our model. We evaluated the impact of multiple key hyperparameters, such as step size, batch size, pool size, when optimizing zero-one loss through stochastic gradient descent. Finally, we evaluated the baseline model's performance on the CIFAR10 and STL10 data sets, and both are ten class benchmarks. Our algorithm focuses on solving binary classification problems, so we split the 10 class datasets into sub-datasets composed of two classes in each of CIFAR10 and STL10. In other words, each dataset has a total of 45 sub-datasets. In addition, we also trained a ReLU activation MLP, a Sign activation MLP optimized using an approximated gradient method, and a ReLU activation MLP optimized using stochastic coordinate descent as references to evaluate our performance algorithm.

### 3.3.1 Datasets

**CIFAR10** "The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another.

Between them, the training batches contain exactly 5000 images from each class". [1][35].

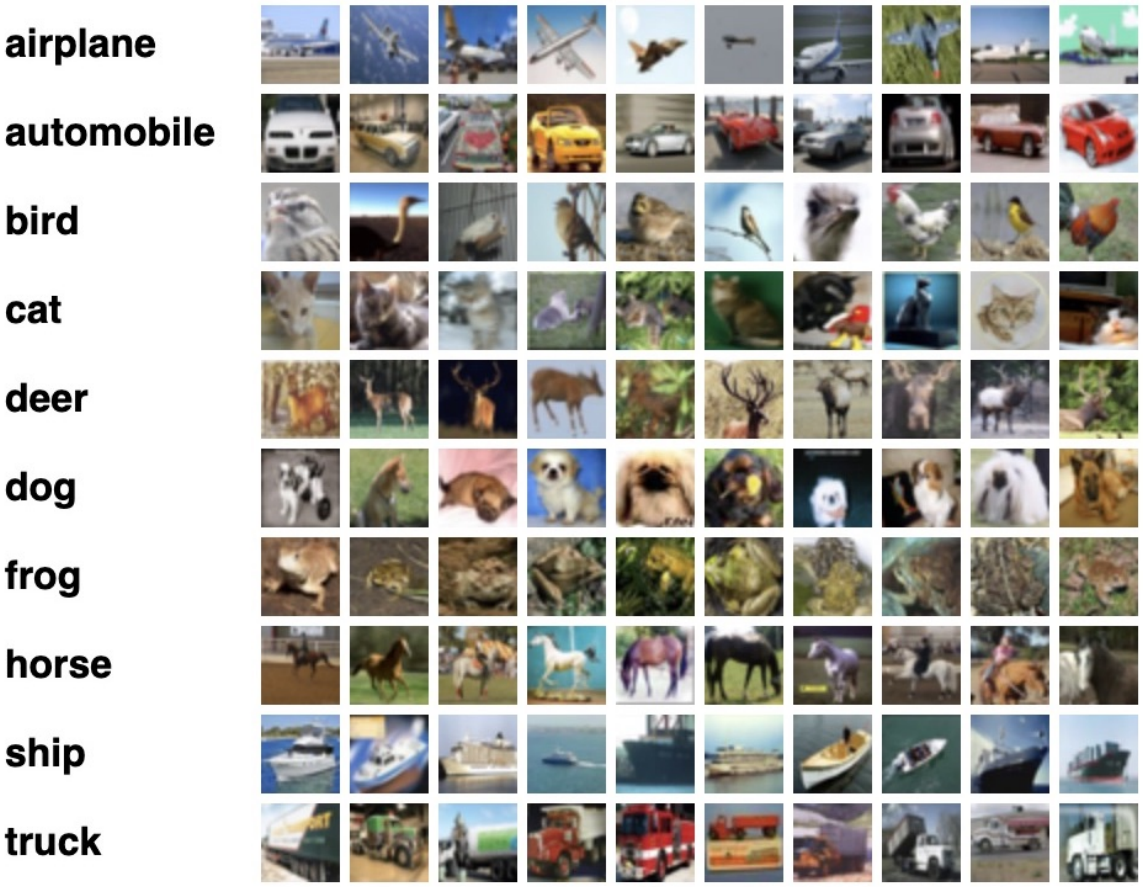


**Figure 3.3** CIFAR10 dataset. Here are the classes in the dataset, as well as 10 random images from each.

**STL10** "The STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the CIFAR-10 dataset but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from

[1]see https://www.cs.toronto.edu/ kriz/cifar.html

a similar but different distribution from the labeled data) to build a useful prior. We also expect that the higher resolution of this dataset (96x96) will make it a challenging benchmark for developing more scalable unsupervised learning methods. [13].10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. Images are 96x96 pixels, color. 500 training images (10 pre-defined folds), 800 test images per class. 100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in addition to the ones in the labeled set. Images were acquired from labeled examples on ImageNet".[15] [2]

### 3.3.2 Hyperparameters

Stochastic Coordinate Descent (SCD) is a heuristic search algorithm. Its purpose is to update one coordinate, which is the weight corresponding to a feature, at a time, and obtain the lowest loss value available in each iteration. Therefore, the choice of hyperparameters will have a significant impact on the search effect. A good set of parameters can help searching to achieve higher accuracy upper bound, and bring faster loss convergence speed. Due to significantly different feature dimensions, distributions, and inconsistent classification difficulties in various datasets, the optimal parameter set found in one dataset may not be the best in other datasets. This situation is similar to Stochastic Gradient Descent (the batch size depends on the size of the entire dataset, batch size and learning rate will affect each other) [6]. However, the optimal hyperparameters picked from different datasets are normally similar in our algorithm, saving researchers time spent on tuning parameters.

**Step size** controls how far each coordinate update moves. From the Figure 3.5 we can see bigger step size achieve higher accuracy/lower loss, 0.2 is a optimal one.

[2]see https://cs.stanford.edu/ acoates/stl10/

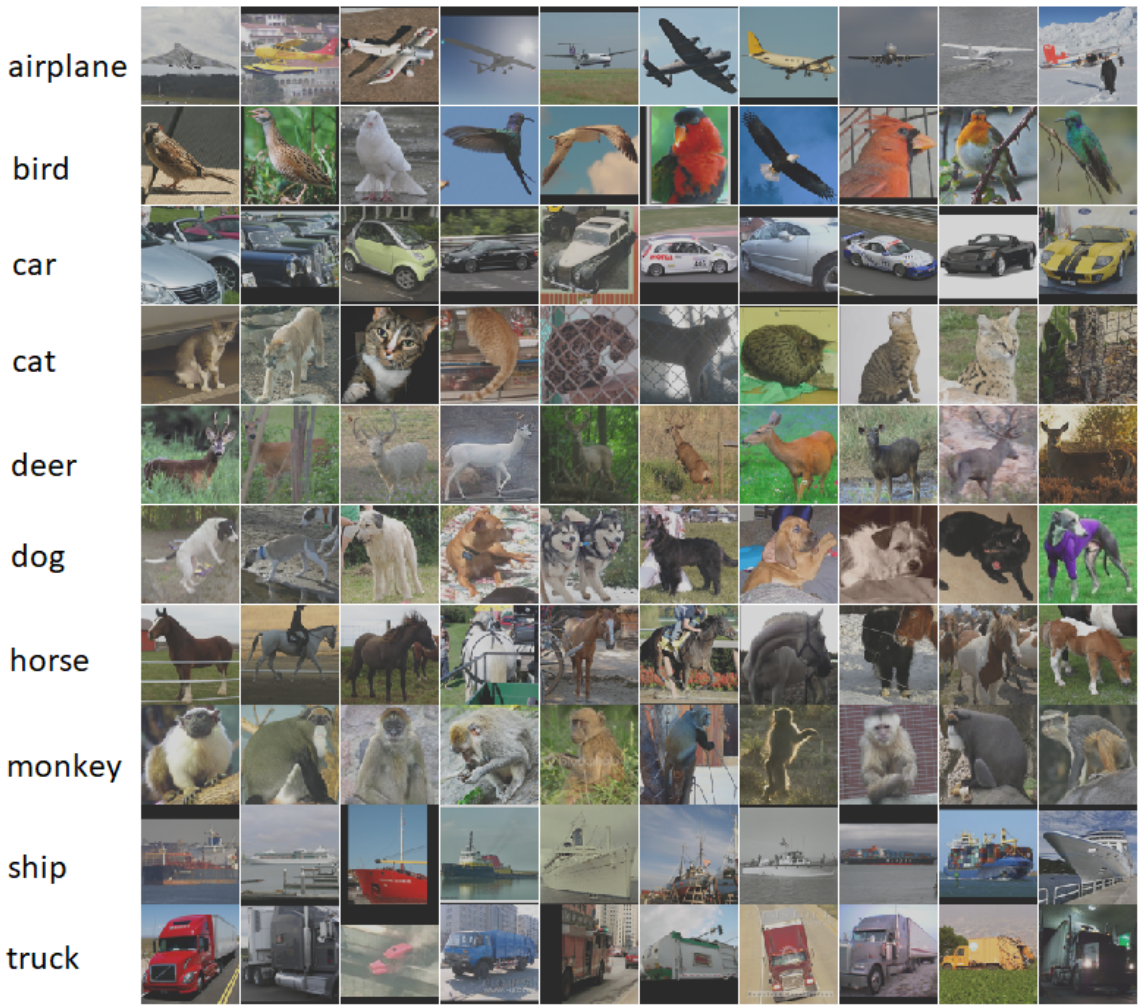


**Figure 3.4** Randomly picked picture samples from STL10 dataset.

Because we normalize weights after each move, the actual step size will be $\frac{w_i + w_{inc}}{||w||}$. $w_i$ is the coordinate to be update, $w_{inc}$ is the step size, $||w||$ is the length of the weight vector. Bigger step size will make the overall weights more sparse after normalization. We recommend smaller step size for higher dimension.

**Iterations** is the total number of iterations in training. More iterations will increase the training time linearly. During the training, we reserve the weights to achieve the highest training accuracy. In Figure 3.6, after 1000 iterations, the training accuracy is still increasing but the test accuracy curve begins to oscillate. Considering

**Table 3.1** Hyperparameters in Zero-One Loss Stochastic Coordinate Descent

| **Hyperparameters** | **Description** |
|---|---|
| Step size | $w_{inc}$ in Algorithm 1 |
| Iterations | The number of iterations, $it$ in Algorithm 3 |
| Batch size | The ratio of training data sampling in each iteration |
| Pool size | The number of coordinates/features will be considered and updated in each inner loop |
| Interval | The number of neighbours considered in best bias searching |
| Votes | The number of models used in majority votes |

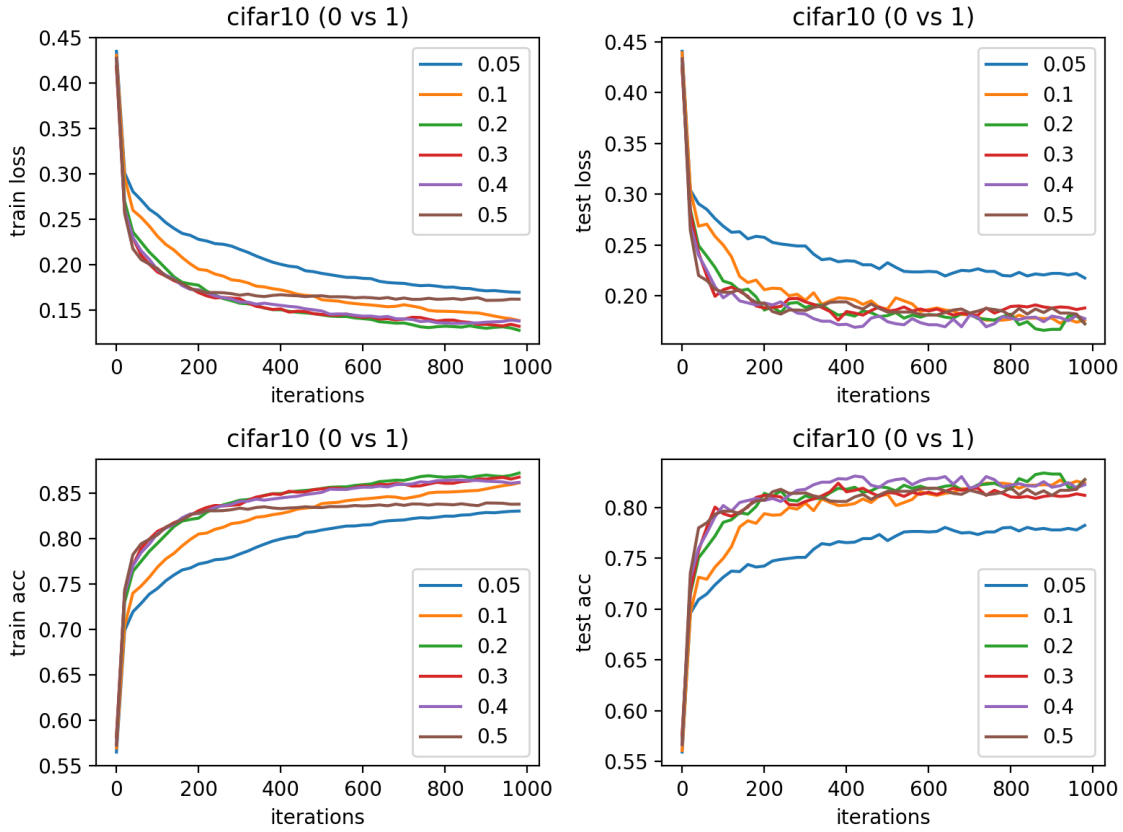


**Figure 3.5** Different step size affect training. Step size includes 0.05, 0.1, 0.2, 0.3, 0.4, 0.5. X-axis is iteration, Y-axis is loss or accuracy.

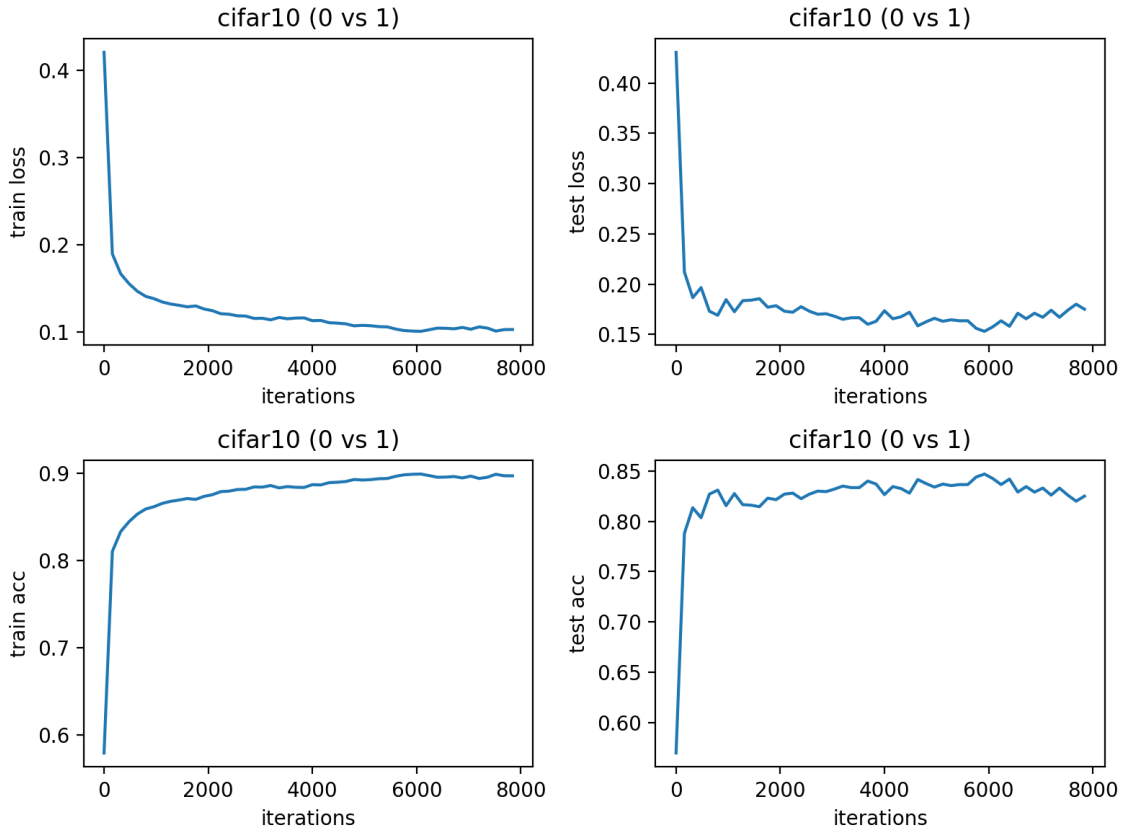


**Figure 3.6** Different iterations affect training process. X-axis is iteration, Y-axis is loss or accuracy.

that we usually use majority vote on multiple diverse weaker models to obtain a stable accuracy, We suggest to pick a good checkpoint in an earlier iteration to save time. In Figure 3.6, we prefer training stop at the 1000 iteration because later ones are not significantly better than it.

**Batch size** In each iteration, we pick $p$ of data to train the model. Figure 3.7 shows that bigger batch size bring higher accuracy and more stable converge curve. This is because a smaller batch can help the searching jump off the local minimum, but it cannot represent the distribution of the whole dataset. As the sampling ratio increases, that batch's data distribution will be more similar to the whole dataset. In other words, the best weight in a bigger batch has a higher probability of being

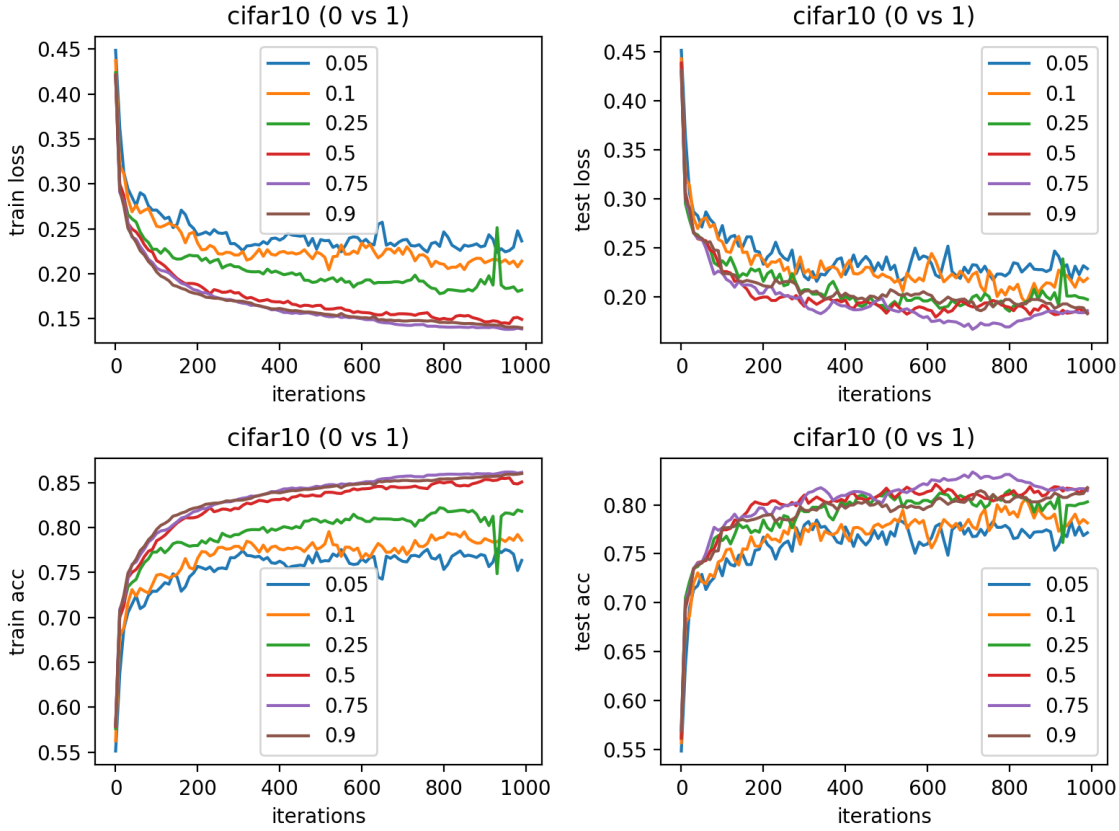


**Figure 3.7** Different batch size affect training. Batch size includes 0.05, 0.1, 0.25, 0.5, 0.75, 0.9. X-axis is iteration, Y-axis is loss or accuracy.

the best for the whole dataset. Nevertheless, we cannot use all data in each iteration because of the local minimum problem. Sampling ratios of 0.5, 0.75, and 0.9 achieve similar performance because of diminishing marginal effect. We prefer the ratio of 0.75 as an optimal one because it performs better than the ratio of 0.5 in test data, and takes less time to train the model than the ratio of 0.9.

**Pool size** is the maximum number of features considered to decide which coordinate should be updated. Once we have a new batch of data and a corresponding global bias, we will pick $k$ features from the previous layer and select the best update on weight coordinate of those features. In general, we would not go through all features because computing complexity of this part is $O(nk)$. If all features' dimension

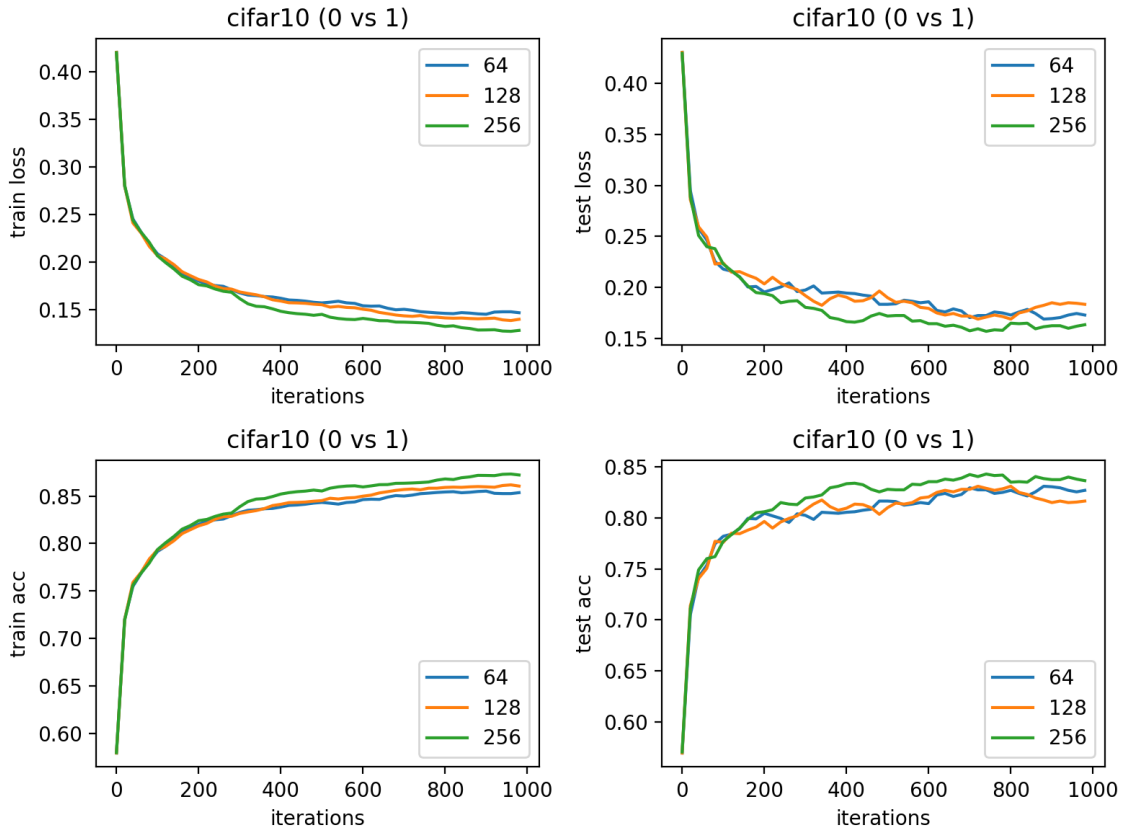


**Figure 3.8** Different pool size affect training. Intervals include 64, 128, 256. X-axis is iteration, Y-axis is loss or accuracy.

is $t$ and $t >> k$, the complexity can be approximately treated as $O(n)$. In practice, selecting a good pool size can save much computing time without losing accuracy. In Figure 3.8, we see the difference between pool sizes of 64, 128, 256 is small, and the difference will almost disappear after the majority of votes. CIFAR10 dataset's image feature dimension is 3072(=32*32*3). However, if we trained the model on ImageNet (the general dimension is more than 150,000), we would need to increase pool size since the dimension is too big.

**Interval** After we get a global bias for a new batch during the training, we should review some coordinates updating and their corresponding bias. Nevertheless, it is unnecessary to search the bias for all unique projections again because the coordinates updating would not affect most data points' signs. We only review the

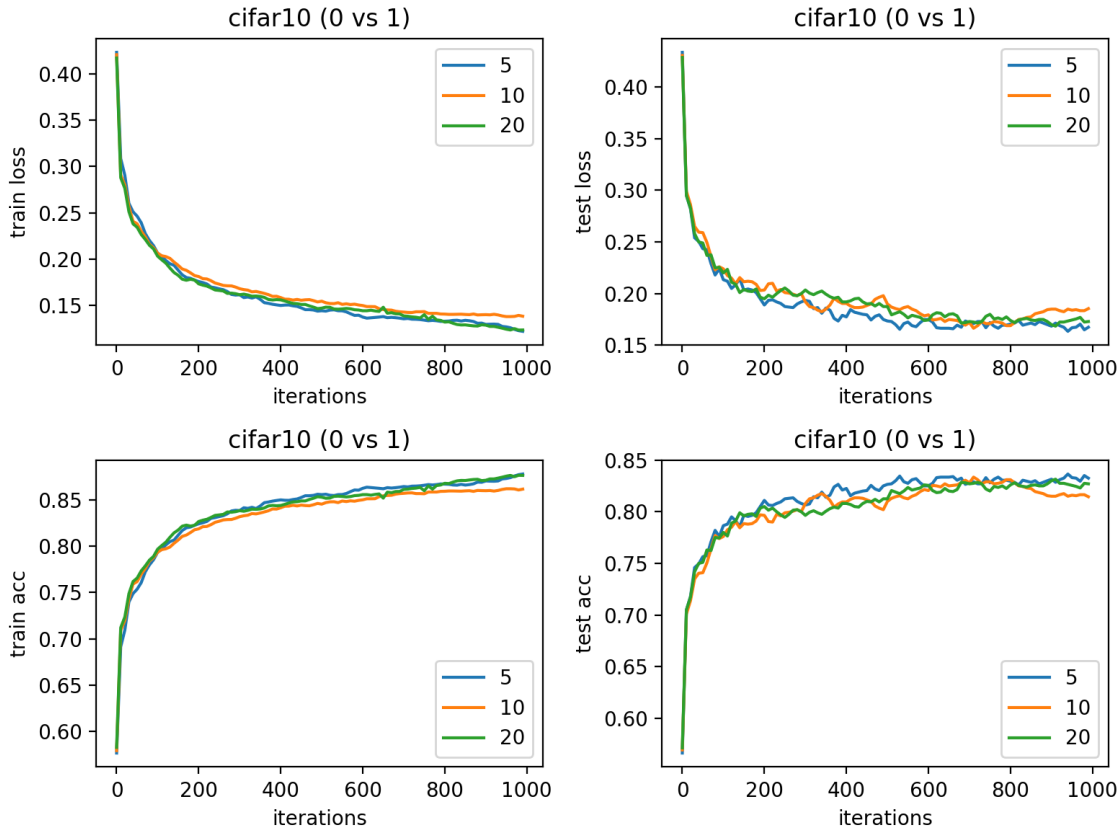


**Figure 3.9** Different intervals affect training. Intervals include 5, 10, 20. X-axis is iteration, Y-axis is loss or accuracy.

neighbor data points whose projection is around the global bias we found. Interval decides how many data points we should look at on each side of the global bias. If the interval is $k$, and global bias $B = \frac{p_i + p_{i+1}}{2}$, we will only search the bias $b$ across projections $[p_{i-k}, p_{i-k+1}, ..., p_{i+k-2}, p_{i+k}]$. In Figure 3.9, we see there is no significant difference among intervals of 5, 10, 20. Actually, we found that interval of 20 is more stable, and it does not take too much longer on computing than interval of 10.

### 3.3.3 Comparison

We compared our models performance to convex models, and training hyperparameters are listed in Table 3.2. Image data will be scaled to range [0, 1] by divided by 255.

1. **MLP-BCE-BP** is a binary cross-entropy loss two layer ReLu activation neural network with 20 hidden nodes optimized by stochastic gradient descent.

2. **MLP-BCE-BAN (Binary Activated Network)** is a binary cross-entropy loss two layer sign activation neural network with 20 hidden nodes optimized by stochastic gradient descent. Because sign activation is non-differentiable, we approximate the gradient like linear activation, but clip the gradient if the activation projection value more than mean in this batch.

3. **MLP-01-SCD** is a zero-one loss two layer Sign activation neural network with 20 hidden nodes optimized by our stochastic coordinate descent.

4. **MLP-BCE-SCD** is a binary cross-entropy loss two layer ReLu activation neural network with 20 hidden nodes optimized by our stochastic coordinate descent. We show that our stochastic coordinate descent can achieve similar performance when the network structure is the same as MLP-BCE-BP. In other words, Sign activation is the major reason cause performance dropped on MLP-01-SCD compared to MLP-BCE-BP.

**Table 3.2** Models' Hyperparameters Setting in Training

| **Model** | **MLP-BCE-BP/BAN** | **MLP-01/BCE-SCD** |
|---|---|---|
| **Learning rate** | 0.01 | 0.17 |
| **Batch size** | 200 | 0.75 |
| **Epoch** | 200 | 1000 |
| **Optimizer** | SGD | / |
| **weight_decay** | 1.00E-04 | / |
| **Nesterov** | TRUE | / |
| **Pool size** | / | 128 |

Tables 3.3 and 3.4 show the accuracy of the four basic models in the 45 subgroups of CIFAR10 and STL10. Table 3.5 summarizes the results and compares the performance of the benchmark model with MLP-BCE-BP. We can see that

**Table 3.3** Training Accuracy (Testing Accuracy) of MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD on CIFAR10 and STL10, Each Model is a Ensembling of Eight Votes (A)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| 0 vs 1 | 95.04 (90.75) | 97.26 (87.90) | 90.24 (85.40) | 92.92 (89.45) | 92.20 (81.19) | 95.50 (81.94) | 96.30 (80.19) | 100.00 (84.06) |
| 0 vs 2 | 90.40 (85.25) | 92.34 (84.20) | 85.52 (80.80) | 90.28 (85.25) | 96.60 (85.50) | 97.20 (83.94) | 97.50 (86.81) | 100.00 (88.50) |
| 0 vs 3 | 97.25 (90.35) | 95.65 (88.45) | 89.56 (85.55) | 93.53 (88.95) | 93.30 (85.12) | 94.10 (84.88) | 96.50 (84.81) | 100.00 (86.81) |
| 0 vs 4 | 95.20 (89.80) | 94.52 (88.30) | 88.87 (86.00) | 92.98 (88.70) | 92.50 (85.69) | 96.30 (85.25) | 97.20 (84.81) | 100.00 (85.25) |
| 0 vs 5 | 97.45 (90.90) | 97.11 (89.25) | 91.02 (86.30) | 94.66 (90.65) | 91.50 (85.38) | 94.70 (84.81) | 97.20 (85.81) | 100.00 (87.94) |
| 0 vs 6 | 97.18 (93.40) | 97.28 (92.00) | 93.51 (90.70) | 96.22 (92.70) | 94.40 (86.75) | 95.20 (86.44) | 96.90 (87.56) | 100.00 (88.88) |
| 0 vs 7 | 96.65 (90.25) | 97.27 (88.20) | 89.90 (85.40) | 93.80 (89.65) | 93.20 (87.56) | 92.10 (87.38) | 97.60 (88.19) | 100.00 (90.50) |
| 0 vs 8 | 92.28 (83.25) | 92.66 (81.75) | 84.27 (78.90) | 88.45 (82.95) | 86.90 (78.19) | 93.20 (78.69) | 94.30 (75.44) | 100.00 (80.56) |
| 0 vs 9 | 93.83 (88.55) | 96.27 (85.05) | 89.69 (84.85) | 92.75 (87.25) | 95.20 (81.50) | 94.90 (80.69) | 96.30 (83.38) | 100.00 (84.38) |
| 1 vs 2 | 98.32 (92.25) | 98.05 (89.55) | 93.67 (89.45) | 96.48 (92.00) | 97.60 (88.31) | 95.60 (86.75) | 98.60 (86.94) | 100.00 (91.00) |
| 1 vs 3 | 96.63 (91.10) | 98.04 (89.45) | 92.00 (87.35) | 96.17 (91.15) | 77.80 (65.06) | 89.80 (66.56) | 95.90 (64.75) | 100.00 (67.00) |
| 1 vs 4 | 98.65 (93.85) | 98.32 (91.75) | 94.37 (90.50) | 97.32 (93.50) | 78.50 (68.44) | 93.00 (70.31) | 96.60 (67.94) | 100.00 (70.75) |
| 1 vs 5 | 99.52 (93.10) | 98.32 (90.35) | 92.64 (88.10) | 96.72 (92.50) | 72.90 (62.94) | 87.70 (64.06) | 96.50 (63.69) | 100.00 (64.75) |
| 1 vs 6 | 99.33 (94.60) | 98.82 (92.10) | 95.66 (92.65) | 98.04 (94.50) | 82.40 (72.81) | 92.50 (73.19) | 97.50 (69.75) | 100.00 (77.00) |
| 1 vs 7 | 98.77 (92.65) | 98.70 (89.80) | 92.16 (88.50) | 96.63 (92.55) | 77.20 (63.06) | 88.90 (62.69) | 95.10 (61.12) | 100.00 (63.50) |
| 1 vs 8 | 93.73 (87.40) | 96.20 (86.35) | 89.44 (84.10) | 93.50 (86.95) | 97.70 (89.94) | 97.20 (88.50) | 97.00 (87.81) | 100.00 (90.12) |
| 1 vs 9 | 88.31 (77.50) | 91.02 (76.20) | 78.70 (70.75) | 85.33 (76.60) | 98.20 (87.50) | 95.90 (85.62) | 97.80 (85.62) | 100.00 (88.69) |
| 2 vs 3 | 86.88 (77.20) | 87.00 (75.95) | 83.73 (74.90) | 84.73 (77.65) | 98.60 (86.56) | 97.00 (84.88) | 99.00 (83.44) | 100.00 (87.38) |
| 2 vs 4 | 87.81 (74.25) | 88.97 (72.90) | 76.51 (66.65) | 81.81 (73.40) | 99.00 (89.12) | 96.70 (87.44) | 98.30 (88.75) | 100.00 (90.00) |
| 2 vs 5 | 85.45 (77.95) | 90.29 (77.15) | 83.10 (73.80) | 85.30 (77.65) | 98.60 (88.44) | 96.60 (86.62) | 97.40 (86.06) | 100.00 (89.12) |
| 2 vs 6 | 91.45 (83.20) | 92.94 (81.65) | 83.16 (76.90) | 87.18 (82.25) | 98.80 (88.62) | 97.00 (87.12) | 98.70 (86.12) | 100.00 (88.75) |
| 2 vs 7 | 90.20 (83.95) | 93.66 (82.25) | 86.27 (79.85) | 89.27 (83.45) | 99.60 (88.94) | 96.50 (86.75) | 99.20 (85.88) | 100.00 (89.44) |
| 2 vs 8 | 97.29 (91.65) | 97.33 (90.65) | 91.70 (86.95) | 95.15 (91.75) | 94.30 (81.88) | 95.80 (81.38) | 96.40 (82.94) | 100.00 (84.94) |
| 2 vs 9 | 96.86 (91.80) | 97.24 (89.80) | 93.02 (89.35) | 94.95 (91.85) | 84.60 (70.31) | 92.20 (70.50) | 96.60 (67.75) | 100.00 (69.88) |

MLP-BCE-BP has the best performance in CIFAR10. The performance of the equivalent structure network optimized by scd (MLP-BCE-SCD) is very close to the benchmark model, and the performance loss is as small as 1%. This shows that using the same loss function, the search accuracy of SCD can be close to SGD. In STL10, MLP-BCE-SCD is 2% better than MLP-BCE-BP. After replacing ReLu in the network with Sign, the accurate gradient cannot be obtained due to non-differentiable activation function. Backpropagation cannot be used directly to solve the problem, but the approximate gradient can be obtained. Limited by the

**Table 3.4** Training Accuracy (Testing Accuracy) of MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD on CIFAR10 and STL10, Each Model is a Ensembling of Eight Votes (B)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| 3 vs 4 | 92.30 (80.50) | 93.13 (79.40) | 85.92 (78.85) | 88.19 (80.55) | 76.70 (68.94) | 90.00 (70.50) | 97.00 (66.88) | 100.00 (68.25) |
| 3 vs 5 | 79.13 (65.20) | 74.02 (63.50) | 70.53 (62.60) | 76.88 (65.70) | 66.80 (59.69) | 86.90 (60.25) | 97.30 (56.88) | 100.00 (58.94) |
| 3 vs 6 | 85.67 (80.45) | 90.05 (79.50) | 83.38 (77.80) | 86.10 (80.65) | 81.60 (70.62) | 93.50 (71.44) | 96.90 (69.38) | 100.00 (72.06) |
| 3 vs 7 | 92.43 (83.10) | 95.59 (81.90) | 85.35 (79.25) | 89.64 (83.05) | 77.60 (64.81) | 88.70 (65.81) | 95.90 (59.38) | 100.00 (63.38) |
| 3 vs 8 | 95.55 (91.35) | 97.32 (89.65) | 92.25 (87.55) | 95.40 (91.45) | 96.00 (89.44) | 96.80 (89.44) | 96.90 (88.31) | 100.00 (90.06) |
| 3 vs 9 | 97.75 (90.25) | 96.82 (87.75) | 89.94 (84.55) | 93.94 (88.85) | 93.10 (86.75) | 95.70 (84.88) | 97.90 (85.38) | 100.00 (87.62) |
| 4 vs 5 | 91.36 (80.95) | 93.27 (78.20) | 85.42 (78.45) | 88.22 (80.15) | 79.70 (68.62) | 91.50 (68.50) | 95.60 (68.75) | 100.00 (70.75) |
| 4 vs 6 | 92.61 (82.05) | 93.63 (80.55) | 83.20 (76.20) | 87.62 (79.90) | 76.80 (68.25) | 91.10 (71.38) | 96.40 (73.31) | 100.00 (77.06) |
| 4 vs 7 | 90.97 (82.80) | 93.21 (80.90) | 84.64 (78.45) | 89.00 (82.70) | 83.50 (71.19) | 92.20 (71.62) | 96.50 (68.19) | 100.00 (73.69) |
| 4 vs 8 | 94.35 (91.40) | 95.86 (90.70) | 93.13 (88.65) | 95.15 (91.25) | 97.50 (90.69) | 97.80 (90.75) | 97.70 (89.19) | 100.00 (91.62) |
| 4 vs 9 | 97.67 (92.60) | 97.33 (90.55) | 93.21 (88.65) | 95.67 (92.60) | 97.60 (88.00) | 97.00 (86.69) | 98.20 (87.44) | 100.00 (89.50) |
| 5 vs 6 | 94.08 (84.20) | 93.14 (81.85) | 86.01 (80.30) | 89.55 (84.50) | 78.30 (65.12) | 88.70 (65.50) | 96.10 (61.75) | 100.00 (66.69) |
| 5 vs 7 | 90.73 (82.85) | 92.64 (80.80) | 84.78 (76.70) | 88.55 (82.80) | 75.40 (62.94) | 90.50 (64.81) | 95.60 (59.88) | 100.00 (64.25) |
| 5 vs 8 | 98.73 (92.05) | 97.65 (90.20) | 92.84 (88.50) | 96.59 (92.10) | 94.50 (91.81) | 96.00 (91.44) | 97.90 (89.88) | 100.00 (91.56) |
| 5 vs 9 | 96.77 (90.75) | 96.52 (89.00) | 91.59 (86.70) | 95.17 (91.00) | 92.70 (87.06) | 95.60 (85.94) | 97.70 (87.00) | 100.00 (88.38) |
| 6 vs 7 | 96.95 (91.30) | 97.60 (89.45) | 91.22 (86.40) | 94.39 (89.75) | 83.00 (70.56) | 93.90 (70.81) | 96.60 (65.50) | 100.00 (72.62) |
| 6 vs 8 | 98.33 (94.85) | 98.36 (93.95) | 96.16 (92.70) | 97.82 (95.05) | 96.40 (91.50) | 98.40 (90.94) | 97.80 (90.12) | 100.00 (90.62) |
| 6 vs 9 | 99.10 (93.25) | 97.89 (90.90) | 93.99 (90.15) | 96.69 (93.20) | 95.70 (86.31) | 95.80 (85.69) | 97.10 (85.19) | 100.00 (87.25) |
| 7 vs 8 | 97.79 (92.40) | 98.23 (90.95) | 93.97 (88.95) | 96.46 (92.55) | 97.60 (91.62) | 97.50 (91.31) | 97.20 (89.94) | 100.00 (92.38) |
| 7 vs 9 | 96.76 (89.50) | 96.28 (87.35) | 90.64 (85.70) | 93.84 (89.45) | 95.10 (87.38) | 96.40 (86.69) | 97.10 (85.00) | 100.00 (89.50) |
| 8 vs 9 | 95.52 (86.95) | 95.89 (86.10) | 90.46 (83.15) | 92.82 (87.10) | 89.60 (74.31) | 94.80 (75.06) | 96.10 (75.94) | 100.00 (77.25) |

binary representation of the activation value of 0 and 1, the model performance is slightly reduced (approximately equal to 2). MLP-01-SCD has the worst performance, with a performance loss of about 4. We were considering the difficulty of optimization of zero-one loss and the binary representation of the middle layer. This performance loss is acceptable considering that training a undifferentiable neural network with non-convex loss by gradient-free method is much harder than training a differentiable network by back-propagation.

In the performance part, we pick MLP-BCE-BP's accuracy in each class pair as baseline, compare the other three models' accuracy to MLP-BCE-BP's, and

**Table 3.5** Summary Results of MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD on CIFAR10 and STL10, Each Model is a Ensembling of Eight Votes

| | **Accuracy (mean of all 45 pairs)** | | | | |
|---|---|---|---|---|---|
| **Dataset** | **Model** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| CIFAR10 | Train | 94.2 | 94.88 | 88.6 | 92.4 |
| | Test | 87.2 | 85.42 | 83.2 | 86.8 |
| STL10 | Train | 89.26 | 94.1 | 97.02 | 100 |
| | Test | 79.65 | 79.46 | 78.42 | 81.17 |
| | **Performance (mlpbcebp is the baseline)** | | | | |
| **Dataset** | **Model** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| CIFAR10 | Train | 100.00% | 102.34% | 94.18% | 97.77% |
| | Test | 100.00% | 96.86% | 94.10% | 98.57% |
| STL10 | Train | 100.00% | 103.58% | 104.45% | 108.46% |
| | Test | 100.00% | 100.92% | 98.77% | 103.54% |
| | **Runtime (seconds)** | | | | |
| **Dataset** | **#Samples** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| CIFAR10 | 10000 | 40 | 46 | 64 | 56 |
| STL10 | 1600 | 34 | 39 | 17.46 | 17.3 |

then average the percentage. We can see under the same activation (MLP-01-SCD compared to MLP-BCE-BAN, MLP-BCE-SCD compared to MLP-BCE-BP), the performance of models optimized by SCD is about $1\% - 3\%$ different from models optimized by SGD.

In the runtime part, SCD algorithm takes a longer time to train on the CIFAR10 dataset than SGD algorithm, but much faster in the STL10 dataset. The reason is that SCD always performs better with bigger batch size and more data cost longer

in computing. STL10 dataset contains only 1600 samples in each subset, which is smaller than 10000 samples in each subset in CIFAR10.

## 3.4 Conclusion

This study shows a novel Stochastic Coordinate Descent (SCD) method to optimize the zero-one loss function. In practice, a fixed step size plus weight normalization method adjusts the rotation angle of the hyperplane. In addition, the setting of the feature pool can reduce the number of coordinates to be filtered, and the interval can help to avoid redundant projection calculations. These modifications can save calculating time and memory in multi-core parallel operations, and will not cause excessive adverse effects on the final performance. Finally, we verified the actual performance of SCD on multiple image data sets and analyzed the impact of different hyperparameters on training. Objectively speaking, the shortcomings of SCD are more prominent. As the data dimension increases, SCD often requires many more iterations to achieve convergence.

Comparing with SGD, our implementation has not been deeply optimized, and its running time is longer than SGD. We only verified the performance of SCD on the shallow model network, which does not mean that it can be perfectly applied to the deep network model. As the network model deepens, the local minimum problem of zero-one loss will become more significant. Sign activation is not sensitive to input changes, zero-one loss is challenging to optimize (in the case of no data points passing through the hyperplane, whether the changed hyperplane is better or worse cannot be reflected in zero-one loss, but the cross entropy can). In addition, we only change one coordinate each time, which means the change of the output feature may not necessarily affect the subsequent network layer, when training a certain layer of the network. In other words, as the network deepens, SCD is very likely to fall into a

particular local minimum and not able to escape. Solving this series of problem is the key to applying SCD to deep neural networks successfully.

# CHAPTER 4

# OPTIMIZING CONVOLUTIONAL NEURAL NETWORKS BY STOCHASTIC COORDINATE DESCENT

## 4.1 Background

### 4.1.1 Convolutional Neural Network

A convolutional neural network is an extended form of multiple layers perceptron, which is a representative neural network architecture of deep learning [27, 29]. The research on convolutional neural networks began in the 1980s and 1990s. Time delay neural networks and LeNet-5 were the earliest convolutional neural networks [37]. After the twentieth century, with the introduction of deep learning theory and the use of enormous computing resources, brain neural networks have been developed rapidly and widely used in computer vision and natural language processing [29].

In general, the hidden layers of convolutional neural networks include convolutional layers, pooling layers, and fully connected layers [38]. Modern algorithms may contain residual layers, inception, and other more complex structures [30, 55].

The convolutional layer aims at extracting features from the input data. It contains multiple convolution kernels. Each convolution kernel contains a weight coefficient and a bias vector, similar to a neuron in a feedforward neural network. Each kernel in the convolutional layer connects to multiple kernels in a region close to the previous layer. The region's size depends on the size of the convolution kernel, which is called the "receptive field" which is compared to the receptive field of visual cortex cells in the literature [29]. The convolution kernel will scan the input features regularly, do matrix element multiplication and summation of the input features in the receptive field and superimpose the deviation [27].

Subsequent to feature extraction in the convolutional layer, the output feature map will be passed to the pooling layer for feature selection and information filtering.

The pooling layer contains a preset pooling function, whose function is to replace the result of a single point in the feature map with the feature map statistics of its neighboring regions. The pooling layer selects the pooling area in the same steps as the convolution kernel scanning feature map, which is controlled by the pooling size, step size, and padding [27, 31, 19, 8].

The fully connected layer in the convolutional neural network is equivalent to the hidden layer in the traditional feedforward neural network. The fully connected layer is located at the last part of the hidden layer of the convolutional neural network and only transmits signals to other fully connected layers. The feature map will lose the spatial topology in the fully connected layer and will be expanded into a vector and passed through the activation function [29].

### 4.1.2 Training Convolutional Neural Network

Although the primary strategy of applying Stochastic Coordinate Descent (SCD) to Convolutional Neural Network (CNN) and MLP does not differ a lot, it is still optimized layer by layer, optimizing one node at a time. However, the shared weight mechanism of CNN and the deeper network structure will significantly impact the original SCD algorithm, mainly focusing on massive optional biases, which results to a substantial increase in calculation volume. Furthermore, more layers of sign activation layers would cause the loss at the back of the network not responding to weights' change in the front of the network. In other words, SCD searching falls into a local minimum.

## 4.2 Method

### 4.2.1 CNN01 Architecture

Figure 4.1 shows an example of the structure of the Sign activation convolutional neural network. This structure consists of three convolutional blocks in which there is

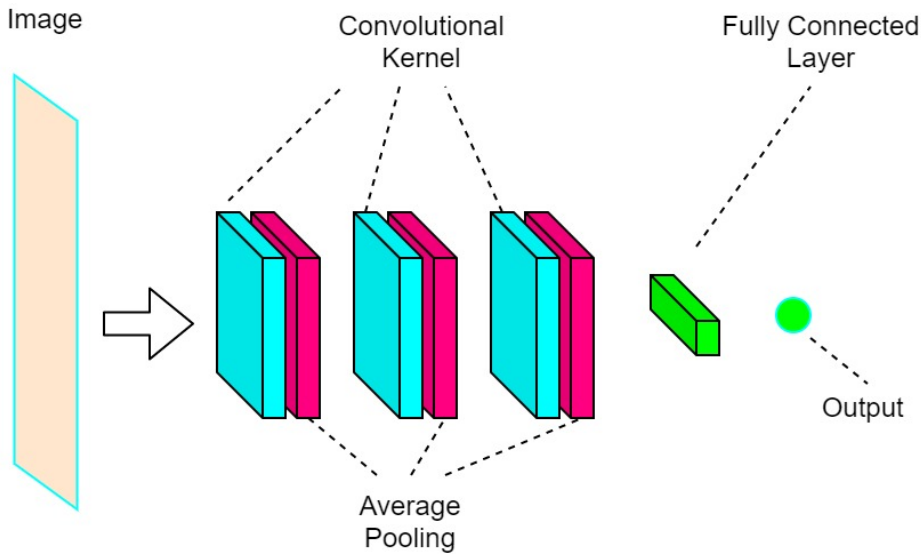


**Figure 4.1** CNN01 architecture for image classification.

a convolutional kernel and an average pooling layer. A fully connected layer followed the last convolutional block, then an output layer. When the image data passes through the first convolution module, the convolution kernel scans the entire image and outputs real-number features. After passing sign activation, the feature consists of binary values of 0 and 1 only and then feeds into the pooling layer for down-sampling. After passing through the second and third convolution modules, the features already contain the high-level semantics of the input image. Finally, after being reshaped into a flatten vector, the features go through the fully connected layer and then generate the final prediction. The whole process is no different from being in a typical CNN.

### 4.2.2 Multiple Phases Training And Temporary Linearization

Directly applying the original training strategy to CNN will easily fall into a local minimum, and this negative effect will gradually become apparent as the network deepens. Therefore, how to solve this problem has become the key to applying SCD to deep neural networks. Our solution is to follow an earlier autoencoder training method and train the network layer by layer. For the training of the deep model,

the BP algorithm is usually used to update the network parameters. However, the network weights need to be initialized very carefully to prevent the network from falling into a local minimum during the training [54]. Of course, there are already many ways to initialize network weights or other deep neural network processing techniques such as ReLU activation function, Batch Normalization, and Residual Connection, which can avoid the network falling into a local minimum in the training process or gradient vanishing and degrading problems [55, 30].

**No Bias** The SCD Algorithm 2 described in the previous chapter mentioned that when we update a node, whenever we obtain a new batch or update to coordinate candidates, the bias should be re-calculated based on all unique projections in that node. Therefore, the time complexity of obtaining bias is $O(N)$, where $N$ is the number of samples in the batch. However, in calculating the convolution kernel, the number of unique projections that appear is related to the output feature dimension in the kernel. Therefore, the time complexity of obtaining the bias will become $O(NK)$, where $K$ is the output dimension in each kernel. When the batch size remains the same, the time complexity will be expanded by K times. For example, in a simple CNN designed for CIFAR10 dataset (image size is $32*32*3$), the output dimension of the first convolutional layer (with 16 convolution kernels) is $N*32*32*16$. Regardless of repeated projection values, the number of candidate bias is $32 * 32 = 1024$ times more than the first layer of MLP. When the input image size is larger, each layer's feature dimension will also increase. Therefore, the time cost to calculate the bias will greatly increase. In training, we constantly reduce the amount of calculation and save space by not including bias. Through experimental observations, even if there is no bias in CNN, the impact on the final model performance is very small, and the accuracy loss is $< 0.5\%$.

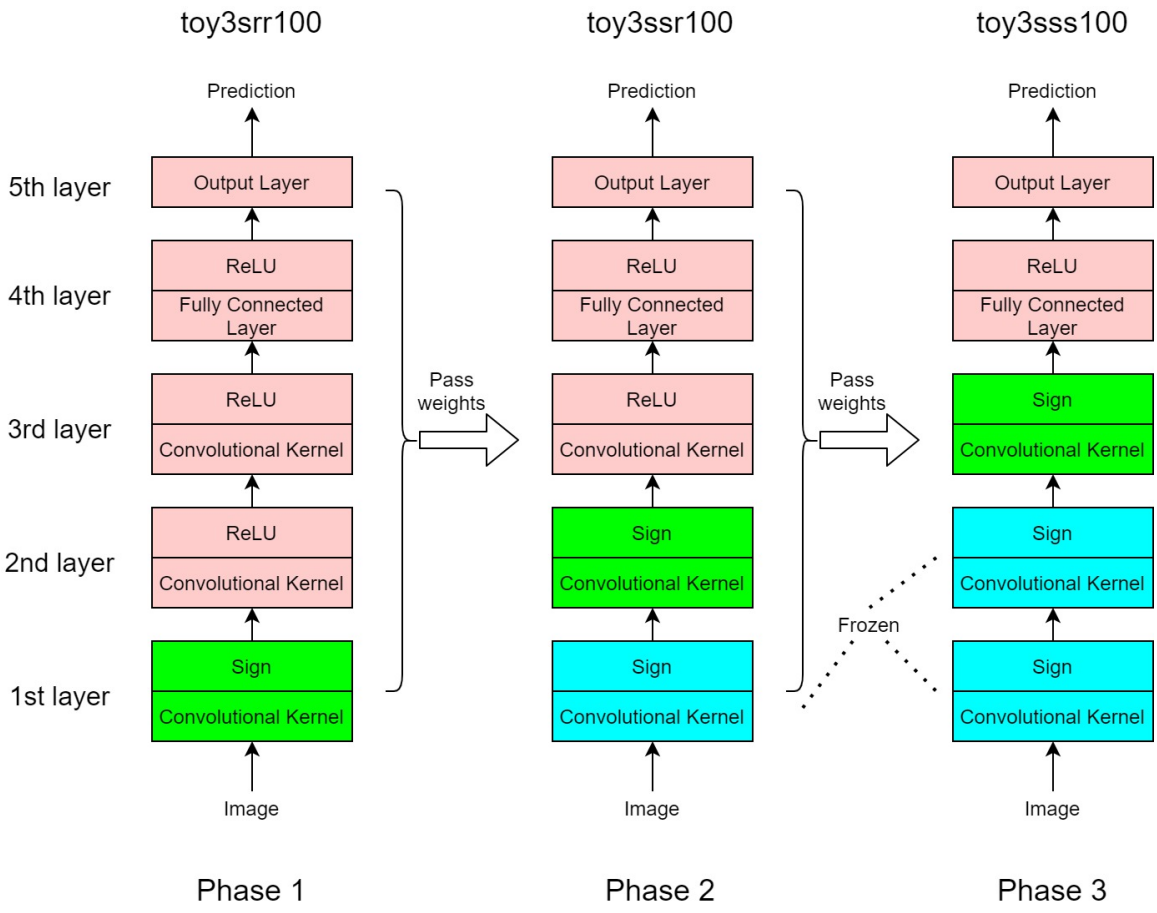


**Figure 4.2** Three phase training for CNN01. toy3srr100, toy3ssr100, toy3sss100 are models' name in each phase. In the first phase, model's weights are initialized randomly, in the other phases, the model will load weights from previous phase.

**Multi-phase Training and Switch Activation** Figure 4.2 shows the process of training a convolutional neural network in three phases. This network is designed for training on the CIFAR10 dataset. It includes three convolution modules, each of which contains three components: a convolution layer, an activation function, and a pooling layer. Passing through the last convolutional module and then a fully connected layer, the last layer finally outputs predicted probabilities for each category. In the first stage, only the first convolutional layer uses Sign activation, and all other layers use ReLU activation. The weights of all layers are randomly initialized. Then the network is trained by SCD in reverse order, from the output layer to the first layer. The main goal of this phase is to train the Sign activated layer, the first layer. Once the loss is converged, the activation function ReLU in the second layer will be changed to Sign. Since the first layer has been trained in the previous stage, it is not involved in training in this second stage. The goal of the second stage is to train the second layer. The training process of the third stage is similar to that of the second stage. The activation function of the third layer will be changed to Sign, and then the second layer trained previously is also frozen in the current round. Since we found that if we set Sign activation in the fully connected layer, the model's overall performance will drop significantly, so the ReLU activation function of the fully connected layer is retained. After three phases of training, we will get a trained convolutional neural network, toy3sss100 with three Sign activated convolutional layers. This training strategy will greatly improve the performance compared to the method of directly training the same structure, as Figure 4.3 shows. Since we will freeze the trained layers in the later stages, the training time required in the later stages will be greatly reduced.

**Learning Rate** Different from using the same learning rate for each layer when training MLP, we found that in CNN training, as the training iteration increases, it

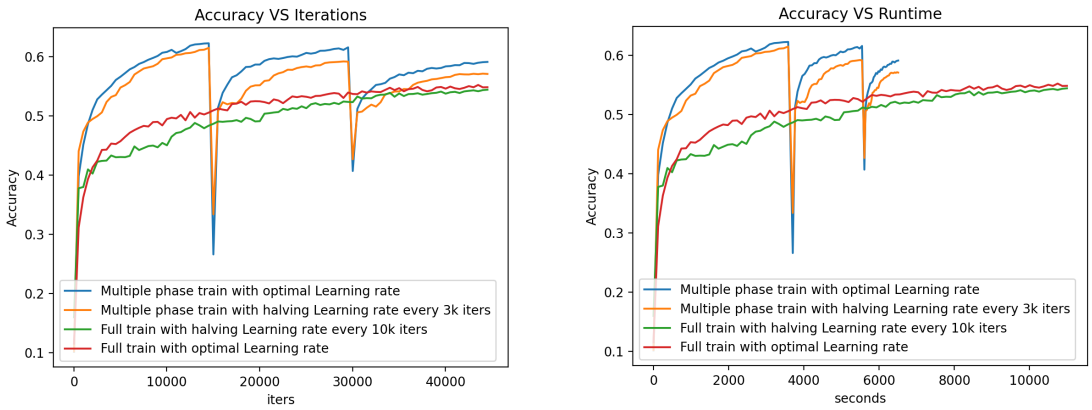


**Figure 4.3** Accuracy curve of three-phase training strategy compared to directly training strategy on CIFAR10 dataset. X-axis is iteration or seconds, Y-axis is accuracy.

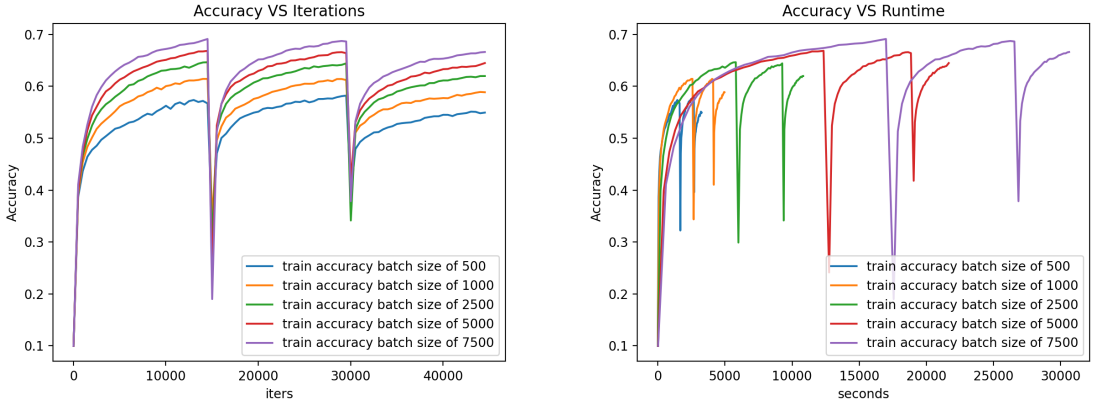


**Figure 4.4** Accuracy curve of different batch size setting in three-phase training strategy. X-axis is iteration or seconds, Y-axis is accuracy.

is better to use a lower learning rate. This situation is similar to conducting learning rate decay in Backpropagation to obtain lower loss. In SCD training, there are two main learning rate adjustment strategies: 1. We try different learning rates on models in each training phase and choose the best one as the final model. 2. Following the strategy of learning rate decay, we halve the learning rate after a certain number of

iterations. The result is shown in the Figure 4.3. It would be better to apply a fixed learning rate at each stage than the learning rate decay strategy.

**Batch Size** As shown in the Figure 4.4, a model trained with a larger batch size results in a higher accuracy. But considering that a larger batch size will bring a huge memory workload and longer training time, we will not always use a large batch size in actual training. Here we have a strategy: every time 3000 iterations are passed, the batch size would be doubled. The batch size resets at the beginning of each stage. In a single-stage with only 15000 iterations, we increase the batch size by up to 8 times, located at 3000th, 6000th, 9000th, and 12000th, respectively. This strategy can significantly reduce training time. Its success is because the small batch size can serve as a warm start, and the larger batch size is used to break the convergence bottleneck of the small batch size training.

### 4.2.3 Additional Backpropagation Penetration (ABP)

Although multi-phase training can relieve symptoms of SCD search falling into the local minimum during training multi-layer convolutional neural networks, the extra training volume greatly extends the training time. In addition, changing only one coordinate in later layers at a time makes loss finitely propagating back to the current trained layer. Considering that backpropagation is an effective training method, we add backpropagation into multi-stage training to better pass the loss back to the previous layer. We call this method Additional Backpropagation Penetration.

**Process** The entire process of additional backpropagation penetration (ABP) is partially similar to the multi-stage training. A three-phase ABP process is shown in the Figure 4.5. In the first stage, all layers' weights are initialized randomly. The first convolutional layer uses Sign activation, and all subsequent layers use ReLU activation. In each iteration, firstly cross-entropy loss for the second to fifth layers by

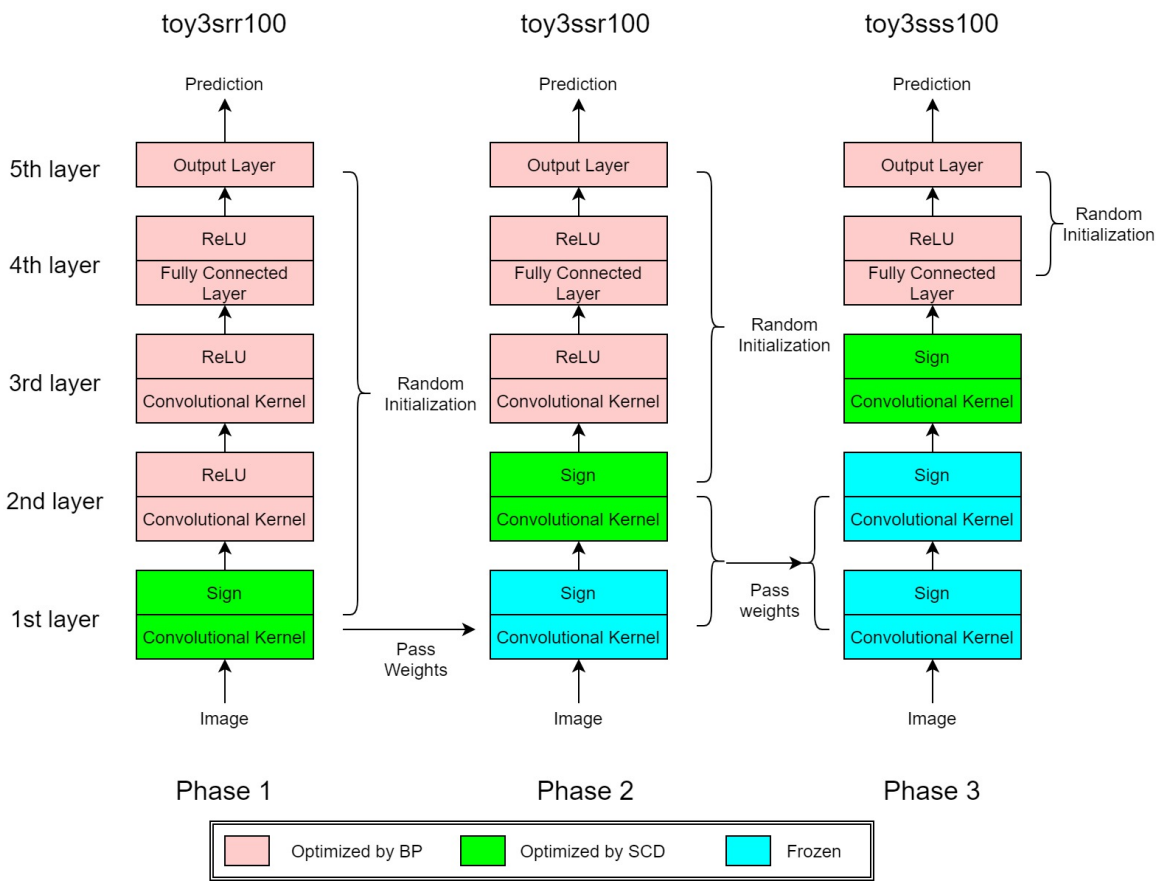


**Figure 4.5** Three phase ABP training for CNN01. toy3srr100, toy3ssr100, toy3sss100 are models' name in each phase.

backpropagation is optimized, then the loss function is reset to zero-one loss and the first layer is updated through SCD. After meeting the training iteration threshold, the second stage begins, and the first layer will no longer participate in subsequent training phases. In the second stage, the activation function of the second layer is changed to Sign, and the weight of that layer is reinitialized. In each iteration, we would use backpropagation to update the third to fifth layers and use SCD to update the second layer. In the third stage, the second layer is frozen, and the third layer's activation function is set to Sign. Backpropagation is applied to update the fourth to fifth layers and SCD is utilized to optimize the third layer. After these three stages are completed, we get the final model, toy3sss100.

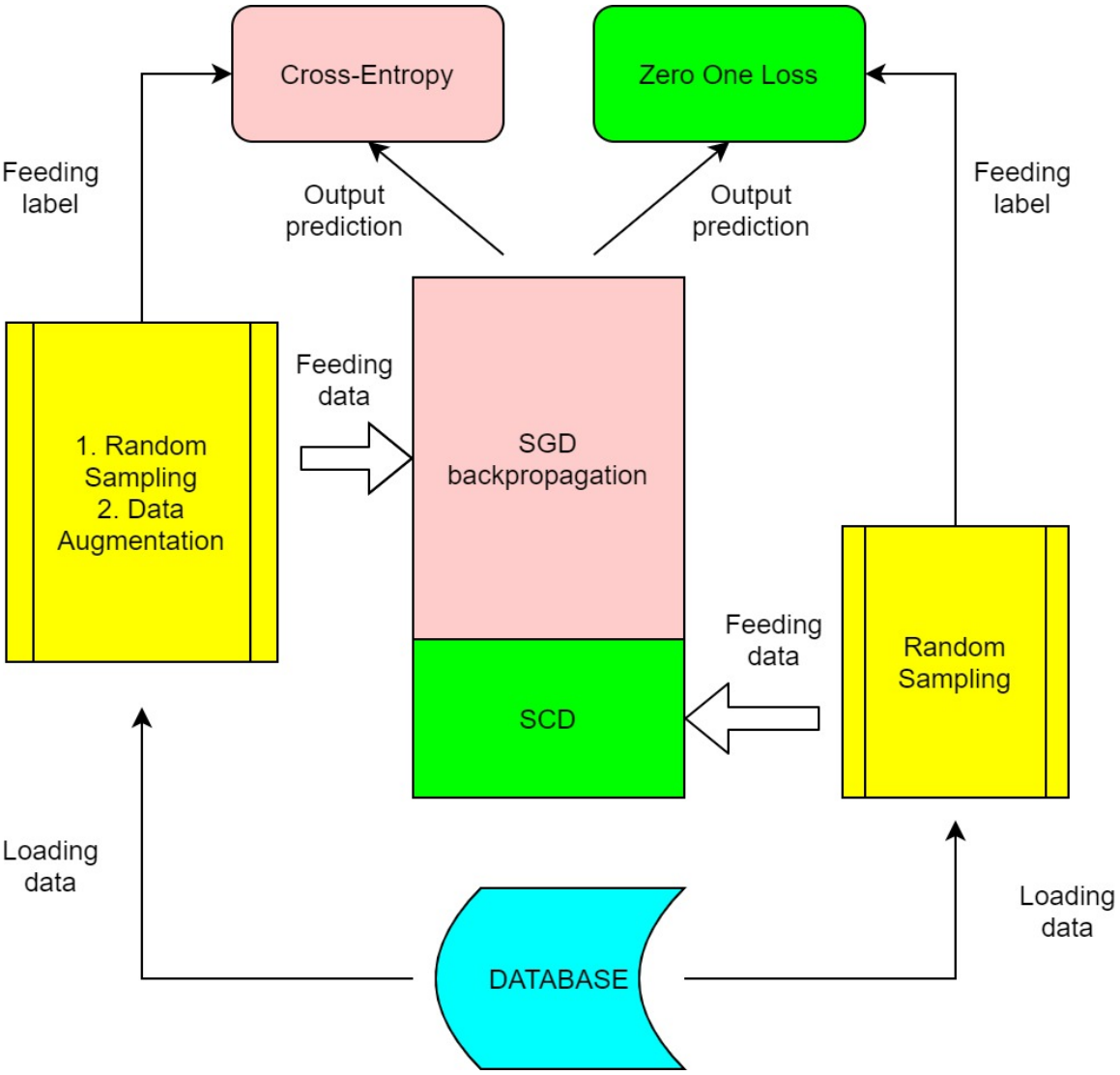


**Figure 4.6** Two independent data samplers for ABP training process.

**Independent Data Sampler** As Figure 4.6 shows, there are two independent data samplers in ABP training process, which is different from general multi-phase training mentioned above. One sampler is responsible for feeding data for ReLU layers during Stochastic Gradient Descent (SGD) backpropagation, and the other one provides data for Sign layers during Stochastic Coordinate Descent (SCD) optimization. For the SGD backpropagation part, additional data augmentation methods like mirror flipping and random cropping will be included in the data processing. Data

augmentation is an effective way of data processing to avoid the overfitting problem while training deep convolutional neural networks, enabling the model to achieve much higher level of accuracy in the test dataset. However, data augmentation can not be as beneficial to our SCD algorithm currently as SGD can, so we only apply data augmentation in the data sampler in SGD part.

**Re-initialization Weights** In the ABP training process, we consistently re-initialize the weights in layers with ReLU activation in the SGD backpropagation part, as Figure 4.5 shows. In general multi-phase training, we don't re-initialize the weights because we use SCD to optimize the whole neural network. Even though the optimized weights do not work after switching the activation function from ReLU to Sign, pre-trained weights work better than random initialization. To avoid the situation that SGD gives similar mapping for the later layers, we would initialize all layers, where SGD will optimize at the beginning of each round. This operation would not affect regular training for accuracy but will affect the transferability of the adversaries during adversary attacks.

### 4.2.4 Multiple Classes Classification

It is a huge challenge to apply zero-one loss to multi-classification tasks directly. zero-one loss can only reflect the error rate of a hyperplane but cannot indicate the distance of a misclassified point from the hyperplane like Cross Entropy does. Therefore, in the optimization process, multiple classifications will bring more local minima than binary classification. We cannot apply zero-one loss directly to the last layer as the final loss function in a multi-class classification task. But we can intergrate zero-one loss into multi-stage training. While applying Additional Backpropagation Penetration (ABP), we use zero-one loss as the loss function to train the Sign activation layer with SCD. However, We switch the loss function back to Cross-Entropy to train the ReLU activation layer and the last layer by backpropagation.

## 4.3 Experiments

### 4.3.1 Datasets

**CIFAR10** "The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class". [1][35].

**STL10** "The STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the CIFAR-10 dataset but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from a similar but different distribution from the labeled data) to build a useful prior. We also expect that the higher resolution of this dataset (96x96) will make it a challenging benchmark for developing more scalable unsupervised learning methods. [13].10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck. Images are 96x96 pixels, color. 500 training images (10 pre-defined folds), 800 test images per class. 100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in

[1]see https://www.cs.toronto.edu/ kriz/cifar.html

addition to the ones in the labeled set. Images were acquired from labeled examples on ImageNet".[15] [2]

### 4.3.2 Models

We built an eight layers convolutional neural network (CNN) for CIFAR10 and a ten layers convolutional neural network "CNN-01-ABP" for STL10 as baseline models. Our algorithm focuses on solving binary classification problems, so we split the 10 class datasets into sub-datasets composed of two classes in each of CIFAR10 and STL10. In other words, each dataset has a total of 45 sub-datasets. In addition, we also trained a ReLU activation CNN "CNN-BCE-BP" optimized by stochastic gradient descent, a Sign activation CNN "CNN-BCE-BAN" optimized by using an approximated gradient methods to evaluate our performance algorithm. Training details listed in Table 4.2. Image data will be scaled to range [0, 1] by divided by 255.

**CNN-BCE-BP** is a standard convolutional neural network, using ReLU activation for each layer, binary cross-entropy as loss function, optimization method is Stochastic Gradient descent. Parameters list in Table 4.1.

**CNN-BCE-BAN** is a convolutional neural network, using Sign activation for convolutional layer and ReLU for fully connected layer, binary cross-entropy as loss function, optimization method is Stochastic Gradient descent. Because sign activation is non-differentiable, we approximate the gradient like linear activation, but clip the gradient if the activation projection value more than quarter of mean in this batch. Parameters list in Table 4.1.

**CNN-01-ABP** is a convolutional neural network, using Sign activation for convolutional layer and ReLU for fully connected layer, zero-one loss as loss function,

[2] see https://cs.stanford.edu/ acoates/stl10/

**Table 4.1** Models' Structure, Parameters, Output Feature Dimension in Each Layer

| **CIFAR10** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Structure** | | **Parameters** | | | | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| **Layer index** | **Layer name** | **Output Feature Dimension** | **Kernel Size** | **Padding** | **Stride** | **#Kernels** | **Activation** | **Activation** | **Activation** |
| 1 | Convolutional Layer | 16x32x32 | 3x3 | 1 | 1 | 16 | ReLU | Sign | Sign |
| 2 | Pooling Layer | 16x16x16 | 2x2 | | 2 | | | | |
| 3 | Convolutional Layer | 32x16x16 | 3x3 | 1 | 1 | 32 | ReLU | Sign | Sign |
| 4 | Pooling Layer | 32x8x8 | 2x2 | | 2 | | | | |
| 5 | Convolutional Layer | 64x8x8 | 3x3 | 1 | 1 | 64 | ReLU | Sign | Sign |
| 6 | Pooling Layer | 64x4x4 | 2x2 | | 2 | | | | |
| 7 | Fully Connected Layer | 100 | | | | 100 | ReLU | ReLU | ReLU |
| 8 | Fully Connected Layer | 1 | | | | 1 | Sigmoid | Sigmoid | Sign |
| **STL10** | | | | | | | | | |
| | **Structure** | | **Parameters** | | | | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| **Layer index** | **Layer name** | **Output Feature Dimension** | **Kernel Size** | **Padding** | **Stride** | **#Kernels** | **Activation** | **Activation** | **Activation** |
| 1 | Convolutional Layer | 16x96x96 | 3x3 | 1 | 1 | 16 | ReLU | Sign | Sign |
| 2 | Pooling Layer | 16x48x48 | 2x2 | | 2 | | | | |
| 3 | Convolutional Layer | 32x48x48 | 3x3 | 1 | 1 | 32 | ReLU | Sign | Sign |
| 4 | Pooling Layer | 32x24x24 | 2x2 | | 2 | | | | |
| 5 | Convolutional Layer | 64x24x24 | 3x3 | 1 | 1 | 64 | ReLU | Sign | Sign |
| 6 | Pooling Layer | 64x12x12 | 2x2 | | 2 | | | | |
| 7 | Convolutional Layer | 128x12x12 | 3x3 | 1 | 1 | 128 | ReLU | Sign | Sign |
| 8 | Pooling Layer | 128x6x6 | 2x2 | | 2 | | | | |
| 9 | Fully Connected Layer | 100 | | | | 100 | ReLU | ReLU | ReLU |
| 10 | Fully Connected Layer | 1 | | | | 1 | Sigmoid | Sigmoid | Sign |

optimization method is stochastic coordinate descent with additional backpropagation penetration. Parameters list in Table 4.1.

### 4.3.3 Comparison

Tables 4.3 and 4.4 show the accuracy of the three basic models in the 45 subgroups of CIFAR10 and STL10. Table 4.5 summarizes the results and compares the performance of the benchmark model with CNN-BCE-BP. Compared to Table 3.5, all version of CNN achieve much higher accuracy than MLP in both CIFAR10 and STL10 datasets.

We can see that CNN-BCE-BP has the best performance in CIFAR10. After replacing ReLU in convolutional modules in the network with Sign, the accurate gradient cannot be obtained due to non-differentiable activation function. Backpropagation cannot be used directly to solve the optimization problem accurately, but the approximate gradient can be obtained instead. Limited by the binary representation of the activation value of 0 and 1, the model performance is slightly reduced (approximately equal to 0.6%). The performance of the equivalent structure network (CNN-01-ABP) optimized by SCD with Additional Backpropagation Penetration (ABP) is very close to the benchmark model, and the performance loss is as small as 0.8%, compared to CNN-BCE-BAN, the model optimized by backpropagation with approximating gradient, the accuracy is only 0.2% lower. This reaveals that SCD does not only work in MLP structure, it could also achieve the same level accuracy in convolutional neural network, a deeper neural network structure. In STL10, CNN-01-ABP's average accuracy is 1.5% lower than CNN-BCE-BP's.

In the performance part, we pick CNN-BCE-BP's accuracy in each class pair as a baseline, compare the other three models' accuracy to CNN-BCE-BP's, and then average the percentage. We can see under the same activation ("CNN-01-ABP" compared to "CNN-BCE-BAN"), the performance of models optimized by SCD is

**Table 4.2** Models' Hyperparameters Setting in Training

| **Model** | **CNN-BCE-BP/BAN** | **CNN-01-ABP** | | | |
|---|---|---|---|---|---|
| **Sub-phase** | / | **toy3srr100/toy4srr100** | **toy3ssr100/toy4ssr100** | **toy3sss100/toy4sss100** | **toy4ssss100** |
| **Epoch** | 200 | 300 | 300 | 300 | 300 |
| **Batch size** | 128 | 200 | 200 | 200 | 200 |
| **BP Learning rate** | 0.001 | 0.001 | 0.001 | 0.001 | 0.001 |
| **BP Optimizer** | Adam | Adam | Adam | Adam | Adam |
| **eps** | 1.00E-08 | 1.00E-08 | 1.00E-08 | 1.00E-08 | 1.00E-08 |
| **Conv Pool size** | / | 32 | 32 | 32 | 32 |
| **FC Pool size** | / | 128 | 128 | 128 | 128 |
| **Conv Learning rate** | / | 0.025 | 0.025 | 0.05 | 0.05 |
| **FC Learning rate** | / | 0.1 | 0.05 | 0.05 | 0.1 |

**Table 4.3** Training Accuracy (Testing Accuracy) of CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP on CIFAR10 and STL10, Each Model is a Ensembling of Eight Votes (A)

| | CIFAR10 | | | STL10 | | |
|---|---|---|---|---|---|---|
| **classes** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| 0 vs 1 | 98.23 (97.60) | 98.00 (96.90) | 99.78 (96.45) | 100.00 (94.88) | 99.90 (95.00) | 100.00 (94.38) |
| 0 vs 2 | 94.39 (92.35) | 93.58 (92.75) | 99.08 (92.05) | 100.00 (95.00) | 99.80 (95.50) | 100.00 (93.81) |
| 0 vs 3 | 97.09 (96.00) | 96.42 (95.70) | 99.87 (95.55) | 99.80 (98.19) | 99.90 (97.81) | 100.00 (97.06) |
| 0 vs 4 | 97.15 (96.35) | 96.89 (96.10) | 99.92 (95.80) | 100.00 (98.19) | 99.80 (97.81) | 100.00 (97.19) |
| 0 vs 5 | 97.95 (97.10) | 97.71 (96.80) | 99.96 (96.70) | 99.90 (98.12) | 100.00 (98.50) | 100.00 (97.56) |
| 0 vs 6 | 98.12 (98.05) | 97.83 (97.65) | 99.98 (97.85) | 99.90 (98.62) | 100.00 (98.94) | 100.00 (97.75) |
| 0 vs 7 | 97.84 (96.45) | 97.22 (96.65) | 99.96 (96.20) | 99.70 (98.81) | 99.80 (98.88) | 100.00 (98.31) |
| 0 vs 8 | 96.63 (94.35) | 94.71 (93.25) | 99.48 (92.85) | 99.90 (93.38) | 100.00 (92.50) | 100.00 (89.44) |
| 0 vs 9 | 96.95 (95.55) | 96.50 (94.90) | 99.80 (95.10) | 99.90 (94.12) | 99.60 (94.25) | 100.00 (92.75) |
| 1 vs 2 | 98.81 (98.35) | 98.67 (98.60) | 99.98 (98.25) | 100.00 (97.12) | 100.00 (96.94) | 100.00 (96.00) |
| 1 vs 3 | 98.40 (97.70) | 98.65 (97.95) | 99.96 (97.00) | 100.00 (86.44) | 99.90 (82.44) | 100.00 (83.69) |
| 1 vs 4 | 99.08 (98.70) | 99.23 (98.85) | 100.00 (98.75) | 100.00 (92.12) | 99.70 (90.38) | 100.00 (88.31) |
| 1 vs 5 | 98.81 (98.50) | 99.23 (99.05) | 99.99 (98.45) | 99.90 (89.94) | 99.90 (88.56) | 100.00 (86.81) |
| 1 vs 6 | 98.77 (98.85) | 98.75 (98.55) | 99.97 (98.55) | 99.90 (94.50) | 99.90 (93.62) | 100.00 (93.12) |
| 1 vs 7 | 99.25 (98.95) | 99.17 (99.05) | 100.00 (98.60) | 99.90 (89.44) | 99.50 (87.94) | 100.00 (85.94) |
| 1 vs 8 | 97.61 (97.35) | 97.35 (96.85) | 99.90 (95.40) | 100.00 (96.88) | 100.00 (97.69) | 100.00 (97.00) |
| 1 vs 9 | 96.13 (94.25) | 95.04 (94.10) | 99.21 (92.65) | 100.00 (96.62) | 99.80 (96.56) | 100.00 (95.44) |
| 2 vs 3 | 91.91 (88.65) | 89.45 (86.80) | 99.05 (87.55) | 100.00 (97.25) | 100.00 (97.56) | 100.00 (96.56) |
| 2 vs 4 | 93.84 (90.65) | 90.26 (87.30) | 98.46 (87.95) | 100.00 (97.25) | 100.00 (98.00) | 100.00 (97.19) |
| 2 vs 5 | 93.41 (90.90) | 91.25 (88.85) | 99.28 (89.80) | 99.60 (97.69) | 100.00 (98.06) | 100.00 (97.12) |
| 2 vs 6 | 93.98 (93.25) | 92.55 (91.25) | 99.07 (92.25) | 100.00 (97.00) | 100.00 (97.19) | 100.00 (96.31) |
| 2 vs 7 | 95.52 (93.60) | 94.56 (92.75) | 99.73 (93.60) | 100.00 (98.12) | 100.00 (98.50) | 100.00 (97.75) |
| 2 vs 8 | 97.96 (96.95) | 97.33 (96.45) | 99.97 (96.25) | 100.00 (95.00) | 100.00 (94.88) | 100.00 (94.25) |
| 2 vs 9 | 98.08 (97.45) | 98.33 (97.40) | 99.93 (96.85) | 100.00 (89.50) | 99.90 (90.19) | 100.00 (88.44) |

about $0.4\% - 0.5\%$ lower than models optimized by SGD. This indicate that multi-phase training with ABP redeem the weakness of SCD in training a deep neural network.

In the runtime part, we see SCD algorithm takes much longer to train the same network than backpropagation. The main reason is we need to train the network layer by layer in multi-phase training, time complexity increase as the network structure

**Table 4.4** Training Accuracy (Testing Accuracy) of CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP on CIFAR10 and STL10, Each Model is a Ensembling of Eight Votes (B)

| | **CIFAR10** | | | **STL10** | | |
|---|---|---|---|---|---|---|
| **classes** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| 2 vs 9 | 98.08 (97.45) | 98.33 (97.40) | 99.93 (96.85) | 100.00 (89.50) | 99.90 (90.19) | 100.00 (88.44) |
| 3 vs 4 | 93.18 (91.70) | 91.82 (90.05) | 99.27 (89.95) | 98.90 (85.12) | 100.00 (84.81) | 100.00 (81.44) |
| 3 vs 5 | 87.58 (83.25) | 79.97 (78.65) | 96.51 (78.80) | 99.50 (80.44) | 99.50 (80.56) | 100.00 (76.50) |
| 3 vs 6 | 92.89 (91.90) | 91.75 (91.20) | 98.72 (92.10) | 100.00 (93.38) | 100.00 (92.38) | 100.00 (91.38) |
| 3 vs 7 | 95.23 (93.00) | 94.10 (92.20) | 99.45 (92.35) | 99.40 (83.75) | 99.70 (84.38) | 100.00 (80.06) |
| 3 vs 8 | 97.82 (96.80) | 97.39 (96.90) | 99.98 (96.70) | 99.80 (97.75) | 100.00 (97.56) | 100.00 (97.31) |
| 3 vs 9 | 97.44 (96.40) | 97.95 (97.00) | 99.77 (95.65) | 99.80 (96.12) | 99.80 (96.50) | 100.00 (95.62) |
| 4 vs 5 | 93.92 (92.50) | 92.02 (90.10) | 99.14 (91.30) | 99.70 (86.50) | 99.90 (84.75) | 100.00 (84.62) |
| 4 vs 6 | 95.93 (94.45) | 95.04 (94.50) | 99.70 (92.95) | 99.90 (90.56) | 99.60 (90.38) | 100.00 (88.94) |
| 4 vs 7 | 94.52 (92.10) | 92.72 (91.20) | 99.07 (91.60) | 99.30 (89.06) | 100.00 (87.31) | 100.00 (86.06) |
| 4 vs 8 | 98.44 (97.65) | 97.98 (97.50) | 100.00 (97.30) | 100.00 (98.12) | 100.00 (98.00) | 100.00 (97.75) |
| 4 vs 9 | 98.78 (98.40) | 98.47 (97.80) | 99.98 (98.15) | 100.00 (96.94) | 99.90 (96.62) | 100.00 (95.81) |
| 5 vs 6 | 95.78 (94.90) | 95.54 (94.60) | 99.62 (94.40) | 99.70 (85.50) | 100.00 (83.81) | 100.00 (81.31) |
| 5 vs 7 | 94.67 (93.25) | 92.87 (91.65) | 99.57 (91.15) | 99.50 (80.56) | 99.70 (79.38) | 100.00 (78.25) |
| 5 vs 8 | 98.56 (97.70) | 98.33 (97.60) | 100.00 (97.40) | 99.80 (98.50) | 100.00 (98.44) | 100.00 (97.81) |
| 5 vs 9 | 97.89 (97.85) | 98.44 (97.65) | 99.93 (97.00) | 100.00 (97.25) | 99.90 (97.44) | 100.00 (96.19) |
| 6 vs 7 | 98.20 (97.35) | 98.42 (97.90) | 99.93 (97.35) | 100.00 (91.00) | 100.00 (90.12) | 100.00 (89.19) |
| 6 vs 8 | 98.70 (98.25) | 98.63 (98.15) | 100.00 (97.95) | 99.80 (98.62) | 100.00 (98.12) | 100.00 (98.00) |
| 6 vs 9 | 98.80 (98.85) | 98.82 (98.55) | 99.93 (98.30) | 100.00 (96.62) | 100.00 (96.50) | 100.00 (95.81) |
| 7 vs 8 | 99.03 (98.55) | 98.52 (98.00) | 100.00 (97.90) | 99.90 (98.56) | 100.00 (98.94) | 100.00 (98.50) |
| 7 vs 9 | 98.46 (98.00) | 98.17 (97.90) | 99.82 (96.95) | 99.90 (97.62) | 100.00 (97.88) | 100.00 (96.44) |
| 8 vs 9 | 97.28 (96.30) | 97.18 (96.10) | 99.89 (95.15) | 99.90 (91.50) | 99.90 (91.50) | 100.00 (89.81) |

becomes deeper. Except additional phase training, the SCD search implementation is not optimized as good as Stochastic Gradient Descent overall.

**Table 4.5** Summary Results of CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP on CIFAR10 and STL10, Each Model is a Ensembling of Eight Votes

| | **Accuracy (mean of 45 pairs)** | | | |
|---|---|---|---|---|
| **Dataset** | **Model** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| CIFAR10 | Train | 96.64 | 95.84 | 99.61 |
| | Test | 95.49 | 94.88 | 94.69 |
| STL10 | Train | 99.85 | 99.90 | 100.00 |
| | Test | 93.73 | 93.40 | 92.20 |
| | **Performance (CNN-BCE-BP is the baseline)** | | | |
| **Dataset** | **Model** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| CIFAR10 | Train | 100.00% | 99.77% | 101.58% |
| | Test | 100.00% | 99.28% | 98.82% |
| STL10 | Train | 100.00% | 99.90% | 100.00% |
| | Test | 100.00% | 100.13% | 99.47% |
| | **Runtime (seconds)** | | | |
| **Dataset** | **#Samples** | **CNN-BCE-BP** | **CNN-BCE-BAN** | **CNN-01-ABP** |
| CIFAR10 | 10000 | 226 | 242 | 4285 |
| STL10 | 1600 | 269 | 312 | 5587 |

## 4.4 Conclusion

Convolutional Neural Network (CNN) is an extension of Multiple Layers Perceptron to deal with spatial information. CNN could avoid excessive parameters problems under the shared weights mechanism compared to MLP when the network structure goes deeper. Moreover, the square kernel helps extract spatial information from the image, which is more effective and flexible than flattening the source image into a vector and then feeding it into MLP. Training deep neural network is a big challenge, to overcome gradient vanishing, exploring, and degrading problems, researchers developed ReLU, Batch Normalization, Residual connections to make network can be trained easier.

Training deep sign activation neural network with gradient-free optimization method will meet the similar problems as SGD did. Here, we proposed a multi-phase methodand, which train the network layer by layer, and additional backpropagation penetration strategy to solve the problem of that loss can not be passed back well to the front layers. Through experiments on CIFAR10 and STL10 dataset, we see our SCD algorithm assisted with training strategies above could achieve similar accuracy compared to backpropagation. Unfortunately, only update one coordinate each time and multple phases training makes training takes much longer than SGD, and applying zero-one loss in multi-class classification is long way to go.

In next chapter, we will discuss the robustness of zero-one loss Sign activated models.

# CHAPTER 5

# ADVERSARIAL ROBUSTNESS OF ZERO ONE LOSS SIGN ACTIVATED MODELS

## 5.1 Introduction

Ever since the first paper introducing the incredible power of adversary examples was published, the adversarial robustness of machine learning models has gained widespread attention. The attack methods originated from the gradient-based white-box attacks, then developed to attacks searching minimum distortion and black-box attacks based on adversary transferability. In order to evaluate the adversarial robustness of our zero-one loss neural network, we applied Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) for white-box attacks and evaluated the adversaries' transferability among the same type of models and between different types of models. In addition, we conducted minimun distortion attack such as Decision Boundary Attack to get each model's $l_2$ minimum distance between adversaries and clean data. Finally, we compared the adversarial robustness difference between the zero-one loss neural network optimized by SCD and the cross-entropy neural network optimized by SGD with the same structure.

## 5.2 White-box Attack

In a white-box setting, the users have access to all the information of the target neural network, including its architecture, parameters, gradients. The users can make full use of the network information to carefully craft adversarial examples. White-box attacks have been extensively studied because the disclosure of model architecture and parameters helps people understand the weakness of DNN models clearly, and it can be analyzed mathematically. As stated by Tram'er et. al. [57], security against white-box attacks is the property that we desire ML models to have. Commonly used

white-box attack algorithm includes Biggio's attack [5], L-BFGS attack [56], Fast Gradient Sign Method [28], Deep Fool [43], Jacobian-based Saliency Map Attack [47], Projected Gradient Descent Attack [36], Carlini & Wagner's Attack, Ground Truth Attack [10]. Furthermore, other $l_p$ attacks try finding the minimum distortion to fool the model [53]. Here, we only introduce Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD) attack, which we used in the experiments.

### 5.2.1 Fast Gradient Sign Method (FGSM)

FGSM [28] is a one-step method introduced to rapidly generate adversarial examples. The formulation is:

$$\begin{aligned} x' &= x + \epsilon \text{sign}(\nabla_x \mathcal{L}(\theta, x, y)) \quad \text{non-target} \\ x' &= x - \epsilon \text{sign}(\nabla_x \mathcal{L}(\theta, x, t)) \quad \text{target on t} \end{aligned} \tag{5.1}$$

For non-target attack, $x$ is the source image. $\epsilon$ is the distortion value, we often set $\epsilon$ as $\frac{n}{256}$ ($n$ is pixel shifting value, 256 is the maximum value in RGB image). $\mathcal{L}$ is the loss of the model on input $x$ if the target is $y$, $\nabla_x \mathcal{L}(\theta, x, y)$ is $x$'s gradient based on the given loss, and we take the sign of the gradient times $\epsilon$, and add it to $x$. Before processing this, we will normalize $x$ to a range of $[0, 1]$. After injection, we will clip $x'$'s range into $[0, 1]$ as well. $+$ will let the distortion to make $x$ move to a place away from the class $y$'s region.(After moving, the $\mathcal{L}(\theta, x, y)$ will increase.) For target attack, we would replace target class $t$ (the class we hope the model to predict the $x$ as) to $y$ in the formula. And inverse the $+$ to $-$, in order to let the $x$ to move to a place close to class $t$'s region. (After moving, the $\mathcal{L}(\theta, x, t)$ will decrease.) Other processes are the same as in untargeted attack. This formulation can be seen as a one-step of gradient descent to solve the problem:

$$\begin{aligned} &\text{minimize } \mathcal{L}(\theta, x', t) \\ &\text{subject to } ||x' - x||_\infty \leq \epsilon \text{ and } x' \in [0, 1]^m \end{aligned} \tag{5.2}$$

The objective function in (5.2) searches the point which has the minimum loss value to label $t$ in $x$'s $\epsilon$-neighbor ball, which is the location where model $F$ is most likely to predict it to the target class $t$. In this way, the one-step generated sample $x'$ is also likely to fool the model. A toy example of adversarial attack is shown in Figure 5.1.

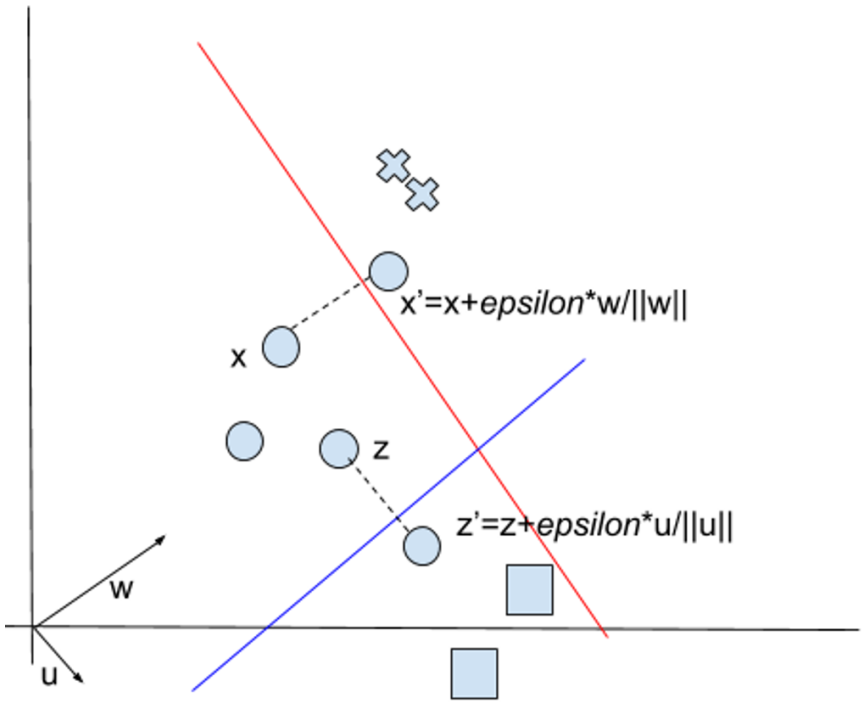


**Figure 5.1** Targeted adversarial attack on multi-class linear classifier.

### 5.2.2 Results of FGSM Attacks on Baseline Models

Tables 5.1 and 5.2 shows our four baseline models' performance on their adversaries generated from the FGSM attack on CIFAR10 and STL10 datasets (test fold). The number in parenthesis is the accuracy on **test data** and the number out of parenthesis is the accuracy on the adversarial version of **test data**. For example, there are 2000 images in classes pair 0 vs. 1, and MLP-BCE-BP's original accuracy in this pair is 90.75%. We do the FGSM attack for MLP-BCE-BP and generate adversary for each image in this pair based on MLP-BCE-BP's information, and then we get a total number of 2000 adversaries. The MLP-BCE-BP's accuracy on these 2000 adversaries

**Table 5.1** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Accuracy on Adversaries (Accuracy on Clean Data) of Test Dataset of CIFAR10 and STL10 Generated by FGSM Attacks ($\epsilon = 16/255$), Each Model is a Ensembling of Eight Votes (A)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd |
| **0 vs 1** | 0.4 (90.75) | 0.95 (87.90) | **36.35 (85.40)** | 10.3 (89.45) | 9 (81.19) | 2.75 (81.94) | **10.81 (80.19)** | 33.56 (84.06) |
| **0 vs 2** | 0.05 (85.25) | 0.15 (84.20) | **49.05 (80.80)** | 2.15 (85.25) | 13.81 (85.50) | 3.5 (83.94) | **15.75 (86.81)** | 39.87 (88.50) |
| **0 vs 3** | 0.9 (90.35) | 2.95 (88.45) | **56.6 (85.55)** | 9.05 (88.95) | 29 (85.13) | 13.63 (84.88) | **17.31 (84.81)** | 40.13 (86.81) |
| **0 vs 4** | 0.3 (89.80) | 0.5 (88.30) | **42.95 (86.00)** | 4.6 (88.70) | 37.75 (85.69) | 14.44 (85.25) | **20.81 (84.81)** | 43.94 (85.25) |
| **0 vs 5** | 0.95 (90.90) | 1.55 (89.25) | **65.6 (86.30)** | 13.75 (90.65) | 43.12 (85.38) | 8.44 (84.81) | **8.94 (85.81)** | 39.19 (87.94) |
| **0 vs 6** | 0.85 (93.40) | 5.6 (92.00) | **51.45 (90.70)** | 17.6 (92.70) | 22.5 (86.75) | 9.19 (86.44) | **20.13 (87.56)** | 41.62 (88.88) |
| **0 vs 7** | 0.8 (90.25) | 0.35 (88.20) | **45.65 (85.40)** | 10.15 (89.65) | 33.69 (87.56) | 15.12 (87.38) | **7.06 (88.19)** | 45.37 (90.50) |
| **0 vs 8** | 0.05 (83.25) | 0.05 (81.75) | **19.5 (78.90)** | 1.95 (82.95) | 17.94 (78.19) | 1.19 (78.69) | **12.31 (75.44)** | 30 (80.56) |
| **0 vs 9** | 2.8 (88.55) | 0.7 (85.05) | **47.6 (84.85)** | 9.45 (87.25) | 14.44 (81.50) | 1.87 (80.69) | **8.87 (83.38)** | 33.19 (84.38) |
| **1 vs 2** | 1.05 (92.25) | 1.45 (89.55) | **39.95 (89.45)** | 19 (92.00) | 15.62 (88.31) | 21.06 (86.75) | **20.06 (86.94)** | 42.37 (91.00) |
| **1 vs 3** | 1.55 (91.10) | 1.4 (89.45) | **47.6 (87.35)** | 18.5 (91.15) | 2.31 (65.06) | 0.06 (66.56) | **2.56 (64.75)** | 14.75 (67.00) |
| **1 vs 4** | 1.1 (93.85) | 0.9 (91.75) | **40.2 (90.50)** | 23.05 (93.50) | 11.75 (68.44) | 0.19 (70.31) | **9.69 (67.94)** | 14.56 (70.75) |
| **1 vs 5** | 1.05 (93.10) | 1.5 (90.35) | **42.9 (88.10)** | 22.75 (92.50) | 5.62 (62.94) | 0.25 (64.06) | **3.62 (63.69)** | 12 (64.75) |
| **1 vs 6** | 0.8 (94.60) | 2.5 (92.10) | **41.1 (92.65)** | 28.25 (94.50) | 3 (72.81) | 0.19 (73.19) | **7.37 (69.75)** | 20.31 (77.00) |
| **1 vs 7** | 1.5 (92.65) | 1.7 (89.80) | **42.5 (88.50)** | 22.2 (92.55) | 4.69 (63.06) | 0.12 (62.69) | **1.69 (61.13)** | 13.75 (63.50) |
| **1 vs 8** | 2.3 (87.40) | 0.7 (86.35) | **46.7 (84.10)** | 7.45 (86.95) | 17.25 (89.94) | 10.81 (88.50) | **30.88 (87.81)** | 46.37 (90.13) |
| **1 vs 9** | 0.1 (77.50) | 0 (76.20) | **38.95 (70.75)** | 0.4 (76.60) | 15.81 (87.50) | 13.5 (85.63) | **27.5 (85.63)** | 47.94 (88.69) |
| **2 vs 3** | 0.45 (77.20) | 1.5 (75.95) | **18.95 (74.90)** | 1.35 (77.65) | 8.94 (86.56) | 18.63 (84.88) | **33.25 (83.44)** | 41.06 (87.38) |
| **2 vs 4** | 0.1 (74.25) | 0 (72.90) | **22.3 (66.65)** | 0.55 (73.40) | 18.37 (89.13) | 19.75 (87.44) | **39.12 (88.75)** | 41.44 (90.00) |
| **2 vs 5** | 0.25 (77.95) | 0.35 (77.15) | **23.8 (73.80)** | 1.4 (77.65) | 12.87 (88.44) | 19.81 (86.63) | **34.13 (86.06)** | 45.19 (89.13) |
| **2 vs 6** | 0.35 (83.20) | 0.25 (81.65) | **35.45 (76.90)** | 2.25 (82.25) | 16.5 (88.63) | 29 (87.13) | **37.56 (86.13)** | 47 (88.75) |
| **2 vs 7** | 0.05 (83.95) | 0.2 (82.25) | **29.35 (79.85)** | 1.85 (83.45) | 18.44 (88.94) | 30.06 (86.75) | **33.12 (85.88)** | 42 (89.44) |
| **2 vs 8** | 0.9 (91.65) | 1.35 (90.65) | **71.75 (86.95)** | 10.3 (91.75) | 6.44 (81.88) | 0.56 (81.38) | **24.81 (82.94)** | 35.25 (84.94) |

is 0.4%, more than 90% drops from the accuracy on clean data (90.75%). We do the same thing for the other three models, MLP-BCE-BAN's accuracy drops from 87.9% to 0.95%, MLP-01-SCD's accuracy drops from 85.4% to 36.35%, MLP-BCE-SCD's accuracy drops from 89.45% to 10.3%. We do the attack on all 45 pairs each from both CIFAR10 and STL10 for the four baseline models and summary results in Tables 5.1 and 5.2.

Figure 5.2 shows the overall adversaries' accuracy on each pair from CIFAR10 dataset. We can see that MLP-01-SCD (green line) performs much better than the

**Table 5.2** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Accuracy on Adversaries (Accuracy on Clean Data) of Test Dataset of CIFAR10 and STL10 Generated by FGSM Attacks ($\epsilon = 16/255$), Each Model is a Ensembling of Eight Votes (B)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd |
| **2 vs 9** | 0.95 (91.80) | 1.85 (89.80) | **42.7 (89.35)** | 12.55 (91.85) | 2.44 (70.31) | 0 (70.50) | **6.19 (67.75)** | 21.44 (69.88) |
| **3 vs 4** | 0.2 (80.50) | 0.45 (79.40) | **26.9 (78.85)** | 2.7 (80.55) | 2.56 (68.94) | 0.06 (70.50) | **1 (66.88)** | 11.63 (68.25) |
| **3 vs 5** | 0.05 (65.20) | 0.1 (63.50) | **20.25 (62.60)** | 0.1 (65.70) | 0.37 (59.69) | 0 (60.25) | **2.5 (56.88)** | 8.37 (58.94) |
| **3 vs 6** | 0.55 (80.45) | 0.3 (79.50) | **32.05 (77.80)** | 2.15 (80.65) | 2.25 (70.63) | 0.12 (71.44) | **14.37 (69.38)** | 17.81 (72.06) |
| **3 vs 7** | 0.2 (83.10) | 0.55 (81.90) | **35.05 (79.25)** | 1.9 (83.05) | 0.88 (64.81) | 0 (65.81) | **2.37 (59.38)** | 10.31 (63.38) |
| **3 vs 8** | 2.2 (91.35) | 3.2 (89.65) | **70.5 (87.55)** | 14.9 (91.45) | 26.62 (89.44) | 25.06 (89.44) | **48.69 (88.31)** | 41.44 (90.06) |
| **3 vs 9** | 0.8 (90.25) | 1.3 (87.75) | **39.3 (84.55)** | 11.6 (88.85) | 21.19 (86.75) | 11.75 (84.88) | **23.31 (85.38)** | 41.25 (87.63) |
| **4 vs 5** | 0.8 (80.95) | 1.3 (78.20) | **22.1 (78.45)** | 4.9 (80.15) | 22.25 (68.63) | 0.94 (68.50) | **4.69 (68.75)** | 19.25 (70.75) |
| **4 vs 6** | 0.05 (82.05) | 0.2 (80.55) | **37.95 (76.20)** | 1.6 (79.90) | 6.5 (68.25) | 0.19 (71.38) | **8.81 (73.31)** | 16.37 (77.06) |
| **4 vs 7** | 0.1 (82.80) | 0.2 (80.90) | **22.7 (78.45)** | 2.85 (82.70) | 2.62 (71.19) | 0.44 (71.63) | **10.25 (68.19)** | 18.63 (73.69) |
| **4 vs 8** | 1.4 (91.40) | 4.4 (90.70) | **70.25 (88.65)** | 8.1 (91.25) | 32.5 (90.69) | 33.5 (90.75) | **45.87 (89.19)** | 55.5 (91.63) |
| **4 vs 9** | 1.15 (92.60) | 1.95 (90.55) | **44.85 (88.65)** | 13.95 (92.60) | 15.5 (88.00) | 15.31 (86.69) | **26.56 (87.44)** | 44 (89.50) |
| **5 vs 6** | 0.2 (84.20) | 0.15 (81.85) | **37.7 (80.30)** | 5.55 (84.50) | 1.25 (65.13) | 0.06 (65.50) | **1.31 (61.75)** | 11.44 (66.69) |
| **5 vs 7** | 0.05 (82.85) | 0.05 (80.80) | **40.25 (76.70)** | 1.2 (82.80) | 2.37 (62.94) | 0 (64.81) | **4.37 (59.88)** | 9.44 (64.25) |
| **5 vs 8** | 1.1 (92.05) | 3.8 (90.20) | **64.25 (88.50)** | 20 (92.10) | 36.06 (91.81) | 31.69 (91.44) | **65.87 (89.88)** | 47.94 (91.56) |
| **5 vs 9** | 1.15 (90.75) | 2.5 (89.00) | **74.1 (86.70)** | 15.15 (91.00) | 23 (87.06) | 14.5 (85.94) | **25.5 (87.00)** | 46 (88.38) |
| **6 vs 7** | 0.55 (91.30) | 0.55 (89.45) | **50.7 (86.40)** | 7.7 (89.75) | 5.25 (70.56) | 0.19 (70.81) | **2 (65.50)** | 19.37 (72.63) |
| **6 vs 8** | 3.25 (94.85) | 5.85 (93.95) | **84.75 (92.70)** | 25.2 (95.05) | 30.56 (91.50) | 23.38 (90.94) | **69.75 (90.13)** | 53.31 (90.63) |
| **6 vs 9** | 0.6 (93.25) | 2.15 (90.90) | **41.45 (90.15)** | 19.75 (93.20) | 20.81 (86.31) | 16.88 (85.69) | **30.63 (85.19)** | 47.56 (87.25) |
| **7 vs 8** | 1.75 (92.40) | 2.25 (90.95) | **70.35 (88.95)** | 12.45 (92.55) | 27.69 (91.63) | 26.87 (91.31) | **55.94 (89.94)** | 54 (92.38) |
| **7 vs 9** | 1.1 (89.50) | 1.9 (87.35) | **59.4 (85.70)** | 8.25 (89.45) | 22.37 (87.38) | 20.75 (86.69) | **30.63 (85.00)** | 50.56 (89.50) |
| **8 vs 9** | 0.35 (86.95) | 0.75 (86.10) | **44.25 (83.15)** | 5.9 (87.10) | 5.56 (74.31) | 0 (75.06) | **8.62 (75.94)** | 21.75 (77.25) |

others' model, MLP-BCE-SCD (yellow line) is the second-best one. MLP-BCE-BP and MLP-BCE-BAN's accuracy are close to 0% for each pair. This indicated that the model optimized by our Stochastic Coordinate Descent (SCD) algorithm could defend the FGSM attack better than Stochastic Gradient Descent (SGD) in this structure on CIFAR10 dataset.

Figure 5.3 shows the overall adversaries' accuracy on each pair from STL10 dataset. The performance difference between SCD models and SGD models is not as big as in the CIFAR10 dataset, but we can still see that SCD models perform better

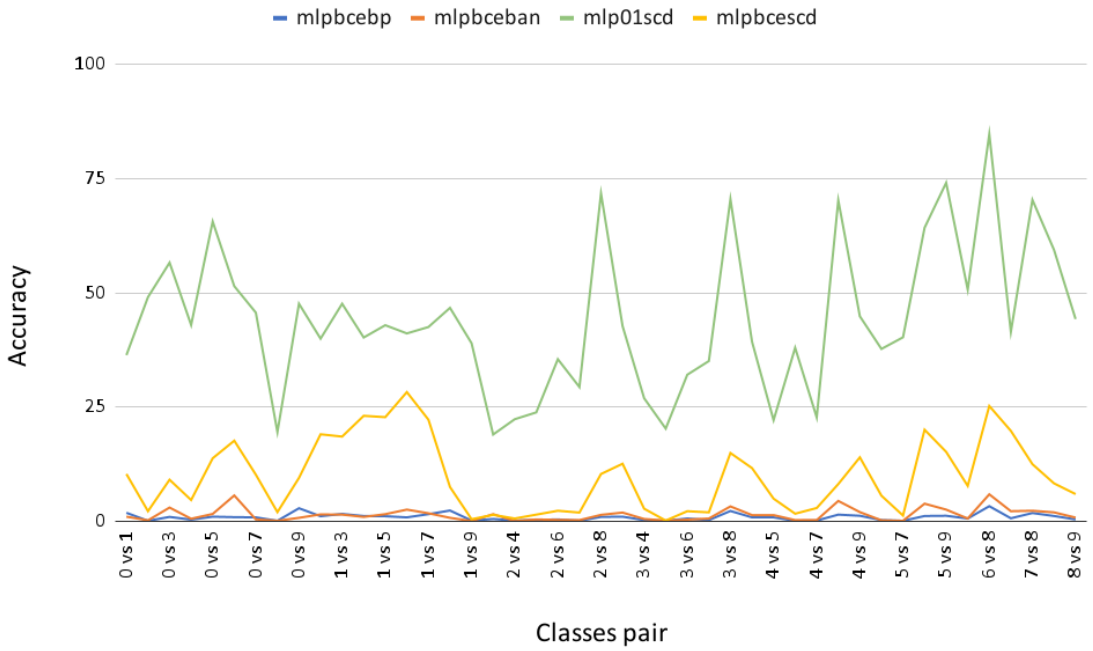


**Figure 5.2** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's accuracy on adversaries of CIFAR10 generated by FGSM attack ($\epsilon = 16/255$). X-axis is classes pair, Y-axis is accuracy on adversaries.

than SGD models in most cases. MLP-01-SCD performs worse than SGD models in some pairs, probably because the structure does not fit the data dimension. The image's dimension in STL10 is 27,648 (96×96×3), which is much bigger than the image's dimension, 3072 (32×32×3) in CIFAR10. The MLP structure with 20 nodes is hard to handle this kind of big image, especially for MLP-01-SCD, whose activation function is **Sign**. Nevertheless, in MLP-BCE-SCD, in which activation is ReLu, it performs stably better than SGD models.

As we knew, Convolutional Neural Network is a much better structure than Multiple Layers Perceptron in extracting spatial information from image data. So we also evaluate our CNN baseline models' performance on adversaries of CIFAR10 generated by FGSM attack, the results listed in Table 5.3. From Figure 5.4, we can see FGSM is not powerful enough to attack CNN structure and make their accuracy

**Table 5.3** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's Accuracy on Adversaries (Accuracy on Clean Data) of Test Dataset of CIFAR10 Generated by FGSM Attacks ($\epsilon = 16/255$), Each Model is a Ensembling of Eight Votes

| **classes** | cnnbcebp | cnnbceban | **cnn01abp** | **classes** | cnnbcebp | cnnbceban | **cnn01abp** |
|---|---|---|---|---|---|---|---|
| **0 vs 1** | 32.4 (97.60) | 53.4 (96.90) | **79.15 (96.45)** | **2 vs 9** | 22.25 (97.45) | 50.35 (97.40) | **77.65 (96.85)** |
| **0 vs 2** | 9.35 (92.35) | 29.4 (92.75) | **54.6 (92.05)** | **3 vs 4** | 4.8 (91.70) | 15.25 (90.05) | **43.35 (89.95)** |
| **0 vs 3** | 24.35 (96.00) | 48 (95.70) | **74.7 (95.55)** | **3 vs 5** | 0.35 (83.25) | 2 (78.65) | **16.7 (78.80)** |
| **0 vs 4** | 9.7 (96.35) | 26.35 (96.10) | **62 (95.80)** | **3 vs 6** | 5.3 (91.90) | 18.1 (91.20) | **45.05 (92.10)** |
| **0 vs 5** | 25.85 (97.10) | 46.45 (96.80) | **78 (96.70)** | **3 vs 7** | 2.05 (93.00) | 8.05 (92.20) | **38.05 (92.35)** |
| **0 vs 6** | 40.45 (98.05) | 57.05 (97.65) | **82.05 (97.85)** | **3 vs 8** | 28.15 (96.80) | 46.4 (96.90) | **80.25 (96.70)** |
| **0 vs 7** | 19.55 (96.45) | 40.95 (96.65) | **74.3 (96.20)** | **3 vs 9** | 13.65 (96.40) | 40.8 (97.00) | **67.35 (95.65)** |
| **0 vs 8** | 7.65 (94.35) | 19.75 (93.25) | **50.6 (92.85)** | **4 vs 5** | 9.85 (92.50) | 16.95 (90.10) | **48.4 (91.30)** |
| **0 vs 9** | 26.7 (95.55) | 43.5 (94.90) | **73.1 (95.10)** | **4 vs 6** | 1.75 (94.45) | 8.3 (94.50) | **42.1 (92.95)** |
| **1 vs 2** | 27.85 (98.35) | 62 (98.60) | **80.75 (98.25)** | **4 vs 7** | 1.65 (92.10) | 8.6 (91.20) | **40.2 (91.60)** |
| **1 vs 3** | 24.65 (97.70) | 55.05 (97.95) | **76.15 (97.00)** | **4 vs 8** | 21.3 (97.65) | 48.75 (97.50) | **74.2 (97.30)** |
| **1 vs 4** | 18.65 (98.70) | 47.95 (98.85) | **78.35 (98.75)** | **4 vs 9** | 16 (98.40) | 47.05 (97.80) | **74.35 (98.15)** |
| **1 vs 5** | 25.45 (98.50) | 57.1 (99.05) | **81.75 (98.45)** | **5 vs 6** | 9.55 (94.90) | 19.05 (94.60) | **50.35 (94.40)** |
| **1 vs 6** | 25.25 (98.85) | 49.35 (98.55) | **81.3 (98.55)** | **5 vs 7** | 2.15 (93.25) | 7.45 (91.65) | **41.95 (91.15)** |
| **1 vs 7** | 25.05 (98.95) | 55.15 (99.05) | **83.95 (98.60)** | **5 vs 8** | 41.1 (97.70) | 54.2 (97.60) | **83.35 (97.40)** |
| **1 vs 8** | 12.9 (97.35) | 32.3 (96.85) | **68.25 (95.40)** | **5 vs 9** | 22 (97.85) | 45.95 (97.65) | **75 (97.00)** |
| **1 vs 9** | 4.3 (94.25) | 14.35 (94.10) | **50.55 (92.65)** | **6 vs 7** | 7.45 (97.35) | 28.3 (97.90) | **62.05 (97.35)** |
| **2 vs 3** | 2.25 (88.65) | 9.85 (86.80) | **35.9 (87.55)** | **6 vs 8** | 40.5 (98.25) | 59.05 (98.15) | **83 (97.95)** |
| **2 vs 4** | 2.25 (90.65) | 6.5 (87.30) | **27.6 (87.95)** | **6 vs 9** | 19.55 (98.85) | 39.8 (98.55) | **77.1 (98.30)** |
| **2 vs 5** | 5.05 (90.90) | 11.55 (88.85) | **43.2 (89.80)** | **7 vs 8** | 26.05 (98.55) | 53.95 (98.00) | **82.25 (97.90)** |
| **2 vs 6** | 6.4 (93.25) | 17.5 (91.25) | **48.3 (92.25)** | **7 vs 9** | 12.6 (98.00) | 39.45 (97.90) | **73.75 (96.95)** |
| **2 vs 7** | 6.35 (93.60) | 18.8 (92.75) | **51.3 (93.60)** | **8 vs 9** | 15.65 (96.30) | 31.95 (96.10) | **67 (95.15)** |
| **2 vs 8** | 21.9 (96.95) | 43.5 (96.45) | **75.7 (96.25)** | | | | |

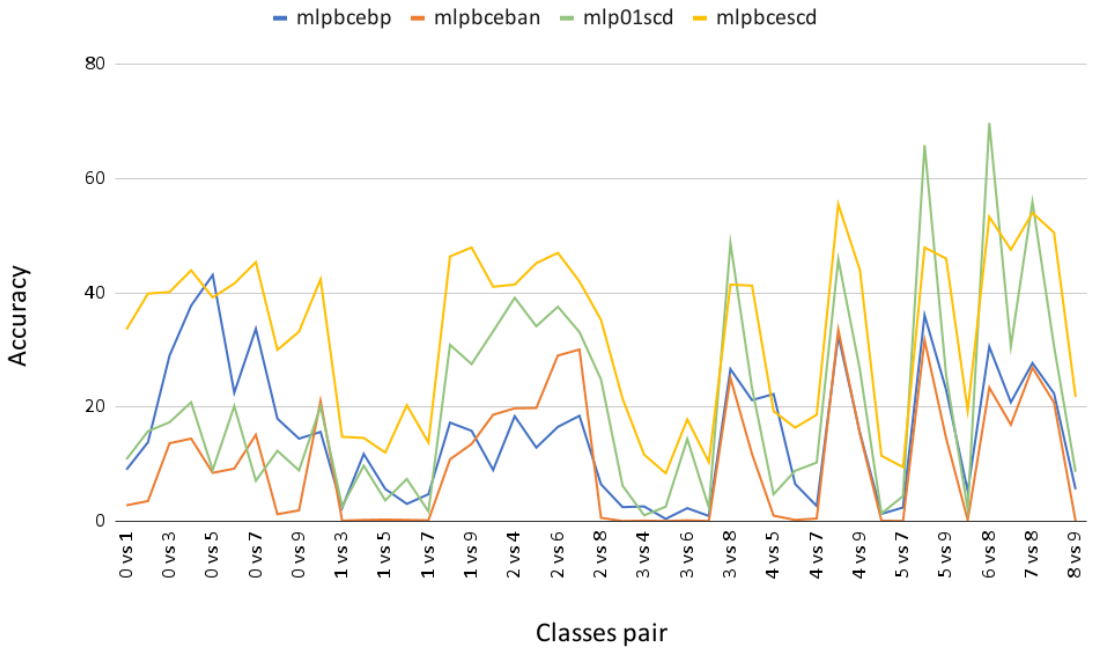


**Figure 5.3** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's accuracy on adversaries of STL10 generated by FGSM attack ($\epsilon = 16/255$). X-axis is classes pair, Y-axis is accuracy on adversaries.

drop to 0% on adversaries. Aleksander Madry [41] claims that *FGSM is a one-time attack method. That is, adding a gradient to data only increases the gradient once. However, if the target is a complex nonlinear model, such a method may not be able to attack successfully.* Even though the FGSM is not a powerful attack on deep neural networks or complicated models, we can also see that CNN-01-ABP performs more robust than CNN-BCE-BP and CNN-BCE-BAN in all pairs. In the next section, we introduce a stronger gradient-based white-box attack method, PGD attack.

### 5.2.3 Projected Gradient Descent

The Basic Iterative Method was first introduced by Alexey Kurakin [36]. It is an iterative version of the one-step attack *FGSM*. In a non-targeted setting, it gives an

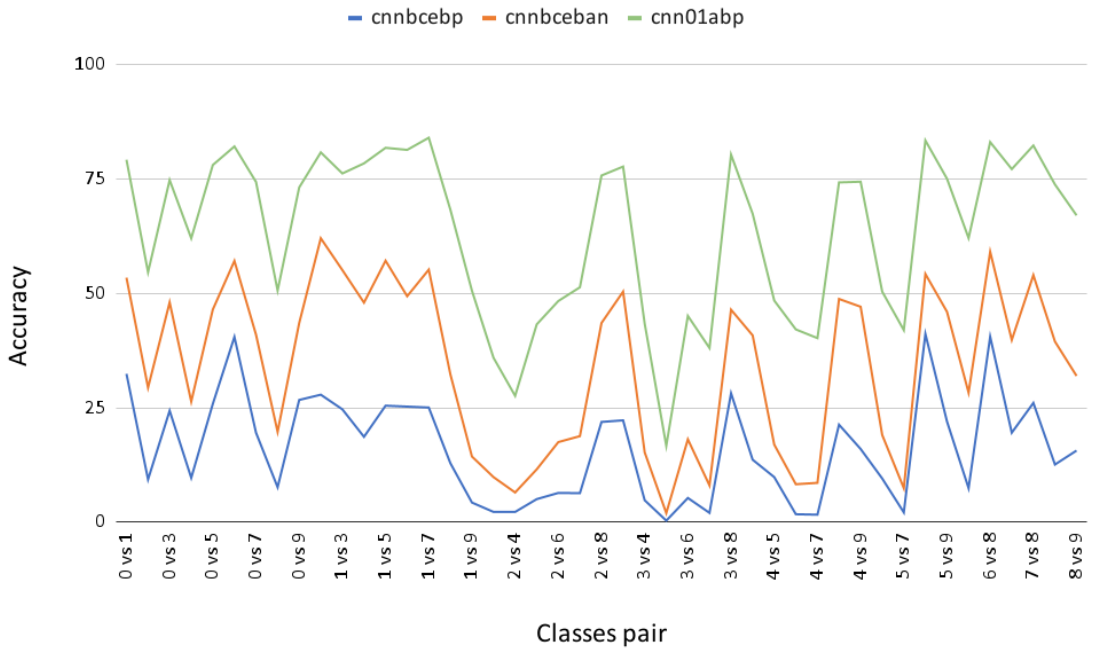


**Figure 5.4** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's accuracy on adversaries of CIFAR10 generated by FGSM attack ($\epsilon = 16/255$). X-axis is classes pair, Y-axis is accuracy on adversaries.

iterative formulation to craft $x'$:

$$\begin{aligned} x_0 &= x \\ x^{t+1} &= Clip_{x,\epsilon}(x^t + \alpha \text{sign}(\nabla_x \mathcal{L}(\theta, x^t, y))) \end{aligned} \tag{5.3}$$

Here, *Clip* restricts the function not passing the surface of $x$'s $\epsilon$-neighbor ball $B_\epsilon(x) : \{x' : ||x' - x||_\infty \leq \epsilon\}$. The step size $\alpha$ usually is a small number (e.g. 1 unit of pixel change for each step), and step numbers ensure that the distortion can reach the border (e.g. $step = 2 \times \epsilon/\alpha$). Projected Gradient Method (PGD) is a variant of BIM, where the distortion start from a random projection on $x + \theta$, instead of the original $x$ [41].

PGD attempts to acquire a perturbation that enlarge the loss of a given model on a specific data sample, while maintain the distortion smaller than a threshold $\epsilon$.

Usually, to make sure the adversary looks imperceptibly different to humans, $L_2$ or $L_\infty$ norm of the distortion will be measured when it adds to the clean sample.

This PGD attack heuristically searches adversary $x'$ achieves largest loss value without going out of the $l_\infty$ ball limitation. These "most-adversarial" examples are most aggressive to fool the classifiers, when the perturbation intensity (its $l_p$ norm) is limited. Finding these adversarial examples help researchers to figure out the weaknesses of machine learning models.

### 5.2.4 Results of PGD Attacks on Baseline Models

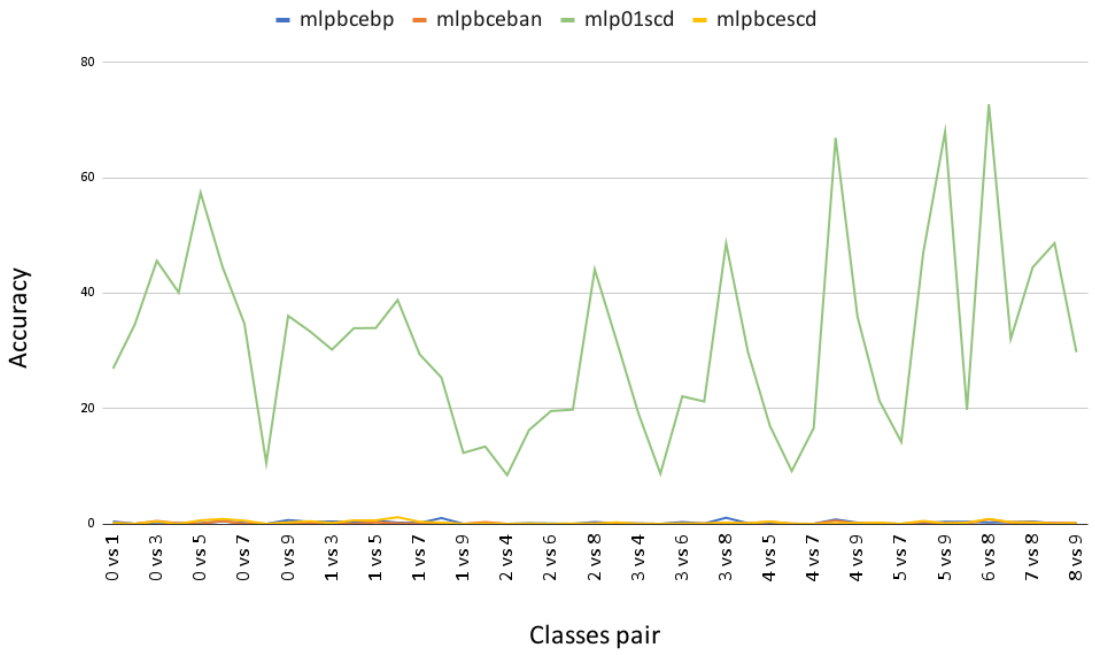


**Figure 5.5** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's accuracy on adversaries of CIFAR10 generated by PGD attack ($\epsilon = 16/255, \alpha = 2/255, steps = 20$). X-axis is classes pair, Y-axis is accuracy on adversaries.

Tables 5.4 and 5.5 shows our four baseline models' performance on their adversaries generated from the PGD attack on CIFAR10 and STL10 datasets (test fold). The number in parenthesis is the accuracy on **test data** and the number

**Table 5.4** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Accuracy on Adversaries (Accuracy on Clean Data) of Test Dataset of CIFAR10 and STL10 Generated by PGD Attacks ($\epsilon = 16/255, \alpha = 2/255, steps = 20$), Each Model is a Ensembling of Eight Votes (A)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd |
| **0 vs 1** | 0.4 (90.75) | 0.1 (87.90) | **26.9 (85.40)** | 0.15 (89.45) | 3.12 (81.19) | 1.25 (81.94) | **0.19 (80.19)** | 0.25 (84.06) |
| **0 vs 2** | 0 (85.25) | 0 (84.20) | **34.6 (80.80)** | 0 (85.25) | 5.31 (85.50) | 1.81 (83.94) | **10.25 (86.81)** | 0.37 (88.50) |
| **0 vs 3** | 0.25 (90.35) | 0.5 (88.45) | **45.6 (85.55)** | 0.4 (88.95) | 15 (85.13) | 10.44 (84.88) | **5 (84.81)** | 0.19 (86.81) |
| **0 vs 4** | 0.1 (89.80) | 0.1 (88.30) | **40.1 (86.00)** | 0 (88.70) | 19.5 (85.69) | 7.12 (85.25) | **5.37 (84.81)** | 2.56 (85.25) |
| **0 vs 5** | 0.1 (90.90) | 0.05 (89.25) | **57.4 (86.30)** | 0.6 (90.65) | 24.94 (85.38) | 7.69 (84.81) | **1.81 (85.81)** | 0.19 (87.94) |
| **0 vs 6** | 0.45 (93.40) | 0.5 (92.00) | **44.55 (90.70)** | 0.85 (92.70) | 13.63 (86.75) | 8.69 (86.44) | **10.56 (87.56)** | 0.37 (88.88) |
| **0 vs 7** | 0.15 (90.25) | 0 (88.20) | **34.7 (85.40)** | 0.55 (89.65) | 18.25 (87.56) | 14.94 (87.38) | **1.69 (88.19)** | 0.62 (90.50) |
| **0 vs 8** | 0 (83.25) | 0 (81.75) | **10.55 (78.90)** | 0 (82.95) | 7.37 (78.19) | 0.75 (78.69) | **1.75 (75.44)** | 0.5 (80.56) |
| **0 vs 9** | 0.65 (88.55) | 0.1 (85.05) | **36.05 (84.85)** | 0.25 (87.25) | 5.75 (81.50) | 1.81 (80.69) | **2.94 (83.38)** | 0.12 (84.38) |
| **1 vs 2** | 0.35 (92.25) | 0.1 (89.55) | **33.35 (89.45)** | 0.45 (92.00) | 5 (88.31) | 5.81 (86.75) | **13.12 (86.94)** | 0.88 (91.00) |
| **1 vs 3** | 0.4 (91.10) | 0.05 (89.45) | **30.2 (87.35)** | 0.15 (91.15) | 0.25 (65.06) | 0.06 (66.56) | **0.06 (64.75)** | 0.12 (67.00) |
| **1 vs 4** | 0.35 (93.85) | 0 (91.75) | **33.9 (90.50)** | 0.6 (93.50) | 3.81 (68.44) | 0 (70.31) | **1.06 (67.94)** | 0 (70.75) |
| **1 vs 5** | 0.55 (93.10) | 0.25 (90.35) | **33.95 (88.10)** | 0.6 (92.50) | 2.31 (62.94) | 0.06 (64.06) | **0.12 (63.69)** | 0 (64.75) |
| **1 vs 6** | 0.2 (94.60) | 0.1 (92.10) | **38.8 (92.65)** | 1.15 (94.50) | 0.94 (72.81) | 0.12 (73.19) | **0.44 (69.75)** | 0.19 (77.00) |
| **1 vs 7** | 0.15 (92.65) | 0 (89.80) | **29.4 (88.50)** | 0.35 (92.55) | 2.25 (63.06) | 0 (62.69) | **0 (61.13)** | 0 (63.50) |
| **1 vs 8** | 1 (87.40) | 0.05 (86.35) | **25.35 (84.10)** | 0.2 (86.95) | 7.06 (89.94) | 9.56 (88.50) | **9.56 (87.81)** | 4 (90.13) |
| **1 vs 9** | 0 (77.50) | 0 (76.20) | **12.3 (70.75)** | 0 (76.60) | 6.25 (87.50) | 7.56 (85.63) | **17.56 (85.63)** | 4.37 (88.69) |
| **2 vs 3** | 0.05 (77.20) | 0.3 (75.95) | **13.4 (74.90)** | 0.05 (77.65) | 3.37 (86.56) | 3.5 (84.88) | **25.56 (83.44)** | 0 (87.38) |
| **2 vs 4** | 0 (74.25) | 0 (72.90) | **8.45 (66.65)** | 0 (73.40) | 8.44 (89.13) | 4.69 (87.44) | **23.12 (88.75)** | 0.06 (90.00) |
| **2 vs 5** | 0.1 (77.95) | 0 (77.15) | **16.25 (73.80)** | 0.05 (77.65) | 4.75 (88.44) | 4 (86.63) | **20.81 (86.06)** | 0.19 (89.13) |
| **2 vs 6** | 0.05 (83.20) | 0 (81.65) | **19.55 (76.90)** | 0 (82.25) | 4.69 (88.63) | 6.88 (87.13) | **27.31 (86.13)** | 0 (88.75) |
| **2 vs 7** | 0 (83.95) | 0 (82.25) | **19.8 (79.85)** | 0.05 (83.45) | 6.94 (88.94) | 7.5 (86.75) | **22.12 (85.88)** | 0.12 (89.44) |
| **2 vs 8** | 0.3 (91.65) | 0.05 (90.65) | **44.1 (86.95)** | 0.05 (91.75) | 0.94 (81.88) | 0.12 (81.38) | **17.62 (82.94)** | 0.25 (84.94) |

out of parenthesis is the accuracy on the adversarial version of **test data**. For example, there are 2000 images in classes pair 0 vs. 1, and MLP-BCE-BP's original accuracy in this pair is 90.75%. We do the PGD attack for MLP-BCE-BP and generate adversary for each image in this pair based on MLP-BCE-BP's information, and then we get a total number of 2000 adversaries. The MLP-BCE-BP's accuracy on these 2000 adversaries is 0.4%, more than 90% drops from the accuracy on clean data (90.75%). We do the same thing for the other three models, MLP-BCE-BAN's

**Table 5.5** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Accuracy on Adversaries (Accuracy on Clean Data) of Test Dataset of CIFAR10 and STL10 Generated by PGD Attacks ($\epsilon = 16/255, \alpha = 2/255, steps = 20$), Each Model is a Ensembling of Eight Votes (B)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd | mlpbcebp | mlpbceban | **mlp01scd** | mlpbcescd |
| **2 vs 9** | 0.1 (91.80) | 0.05 (89.80) | **31.7 (89.35)** | 0.25 (91.85) | 0.31 (70.31) | 0 (70.50) | **0.19 (67.75)** | 0.06 (69.88) |
| **3 vs 4** | 0.05 (80.50) | 0.05 (79.40) | **19.1 (78.85)** | 0 (80.55) | 0.31 (68.94) | 0 (70.50) | **0 (66.88)** | 0 (68.25) |
| **3 vs 5** | 0 (65.20) | 0 (63.50) | **8.7 (62.60)** | 0 (65.70) | 0.06 (59.69) | 0 (60.25) | **0 (56.88)** | 0 (58.94) |
| **3 vs 6** | 0.3 (80.45) | 0.05 (79.50) | **22.1 (77.80)** | 0.05 (80.65) | 0.44 (70.63) | 0.12 (71.44) | **0.12 (69.38)** | 0.44 (72.06) |
| **3 vs 7** | 0.05 (83.10) | 0.05 (81.90) | **21.2 (79.25)** | 0.05 (83.05) | 0.25 (64.81) | 0 (65.81) | **0 (59.38)** | 0 (63.38) |
| **3 vs 8** | 1.05 (91.35) | 0.15 (89.65) | **48.6 (87.55)** | 0.2 (91.45) | 13.88 (89.44) | 11.19 (89.44) | **9.44 (88.31)** | 2.31 (90.06) |
| **3 vs 9** | 0.1 (90.25) | 0.1 (87.75) | **29.75 (84.55)** | 0.1 (88.85) | 11.12 (86.75) | 7.19 (84.88) | **15.25 (85.38)** | 3.75 (87.63) |
| **4 vs 5** | 0.05 (80.95) | 0.35 (78.20) | **17 (78.45)** | 0.4 (80.15) | 13.88 (68.63) | 0.37 (68.50) | **0.56 (68.75)** | 1.12 (70.75) |
| **4 vs 6** | 0 (82.05) | 0.05 (80.55) | **9.15 (76.20)** | 0 (79.90) | 2.87 (68.25) | 0.06 (71.38) | **3.44 (73.31)** | 0.31 (77.06) |
| **4 vs 7** | 0 (82.80) | 0 (80.90) | **16.6 (78.45)** | 0 (82.70) | 1 (71.19) | 0.31 (71.63) | **2.19 (68.19)** | 0.69 (73.69) |
| **4 vs 8** | 0.75 (91.40) | 0.55 (90.70) | **66.9 (88.65)** | 0.1 (91.25) | 18.81 (90.69) | 15.19 (90.75) | **18.37 (89.19)** | 9.31 (91.63) |
| **4 vs 9** | 0.2 (92.60) | 0.05 (90.55) | **35.9 (88.65)** | 0.15 (92.60) | 8.56 (88.00) | 2.94 (86.69) | **21.12 (87.44)** | 3.37 (89.50) |
| **5 vs 6** | 0.05 (84.20) | 0.05 (81.85) | **21.35 (80.30)** | 0.2 (84.50) | 0.12 (65.13) | 0.06 (65.50) | **0 (61.75)** | 0.06 (66.69) |
| **5 vs 7** | 0 (82.85) | 0 (80.80) | **14.2 (76.70)** | 0 (82.80) | 0.69 (62.94) | 0 (64.81) | **0.12 (59.88)** | 0 (64.25) |
| **5 vs 8** | 0.1 (92.05) | 0.3 (90.20) | **46.9 (88.50)** | 0.5 (92.10) | 24.31 (91.81) | 17 (91.44) | **19.44 (89.88)** | 4.19 (91.56) |
| **5 vs 9** | 0.35 (90.75) | 0.1 (89.00) | **67.95 (86.70)** | 0.1 (91.00) | 15.25 (87.06) | 5.37 (85.94) | **18 (87.00)** | 4.12 (88.38) |
| **6 vs 7** | 0.35 (91.30) | 0.05 (89.45) | **19.75 (86.40)** | 0.2 (89.75) | 2.5 (70.56) | 0.12 (70.81) | **0.19 (65.50)** | 0.19 (72.63) |
| **6 vs 8** | 0.25 (94.85) | 0.8 (93.95) | **72.75 (92.70)** | 0.85 (95.05) | 22.5 (91.50) | 9 (90.94) | **15.06 (90.13)** | 5.62 (90.63) |
| **6 vs 9** | 0.3 (93.25) | 0.2 (90.90) | **32.05 (90.15)** | 0.3 (93.20) | 11.44 (86.31) | 5.94 (85.69) | **11.44 (85.19)** | 5.19 (87.25) |
| **7 vs 8** | 0.4 (92.40) | 0.25 (90.95) | **44.45 (88.95)** | 0.25 (92.55) | 14.69 (91.63) | 15.25 (91.31) | **19.31 (89.94)** | 7.25 (92.38) |
| **7 vs 9** | 0.05 (89.50) | 0.15 (87.35) | **48.65 (85.70)** | 0 (89.45) | 11.75 (87.38) | 5.19 (86.69) | **4.56 (85.00)** | 6.88 (89.50) |
| **8 vs 9** | 0.1 (86.95) | 0.15 (86.10) | **29.75 (83.15)** | 0.15 (87.10) | 1.62 (74.31) | 0 (75.06) | **4.5 (75.94)** | 0.12 (77.25) |

accuracy drops from 87.9% to 0.1%, MLP-01-SCD's accuracy drops from 85.4% to 26.9%, MLP-BCE-SCD's accuracy drops from 89.45% to 0.15%. We do the attack on all 45 pairs each from both CIFAR10 and STL10 for the four baseline models and summary results in Table 5.4 and 5.5. Compared to Table 5.1 and 5.2, we can see the overall accuracy on adversaries after PGD attack are lower than FGSM attack, this means PGD attack is more powerful in non-linear classifier because of multiple steps gradient is more accurate to approximate the target models' decision boundary.

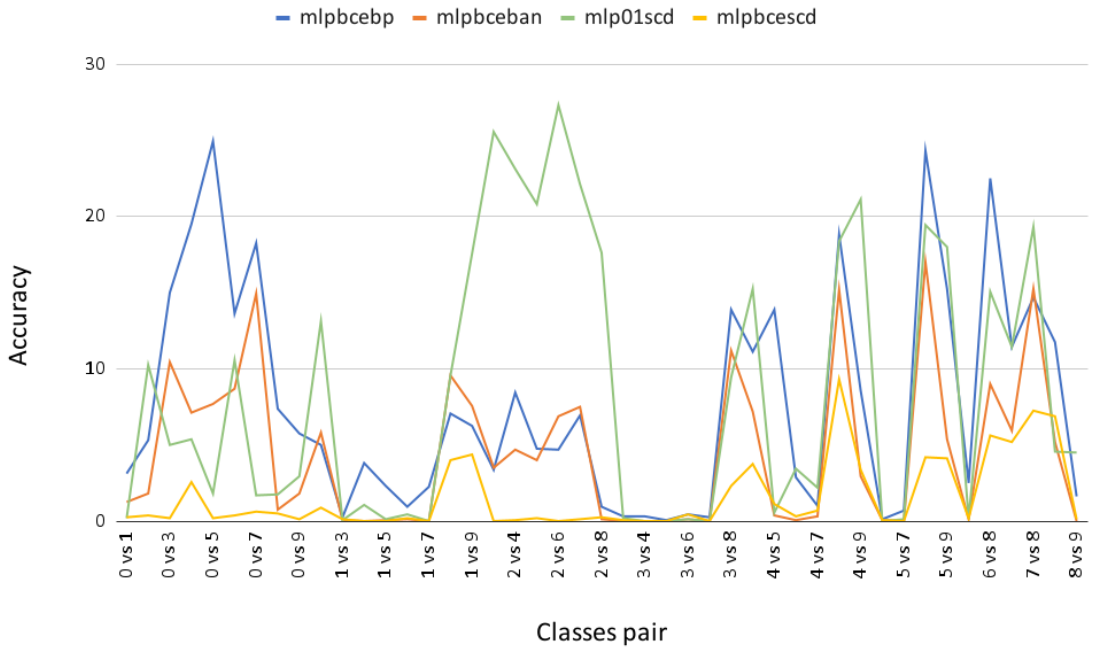


**Figure 5.6** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's accuracy on adversaries of STL10 generated by PGD attack ($\epsilon = 16/255, \alpha = 2/255, steps = 20$). X-axis is classes pair, Y-axis is accuracy on adversaries.

Figure 5.5 shows the overall adversaries' accuracy on each pair from CIFAR10 dataset. We can see that MLP-01-SCD (green line) performs clearly better than the others' model. MLP-BCE-BP, MLP-BCE-BAN, and MLP-BCE-SCD's accuracy are close to 0% for each pair.

Figure 5.6 shows the overall adversaries' accuracy on each pair from STL10 dataset. The performance difference between SCD models and SGD models is not as big as in the CIFAR10 dataset. Overall speaking the MLP-01-SCD model's robustness is not bad.

In the previous section we see FGSM is not very effective in attacking CNN, becasue one step attack can hardly attack structure with complicated non-linear projection. After doing PGD attack for all four baseline models, all models' accuracy

**Table 5.6** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's Accuracy on Adversaries (Accuracy on Clean Data) of Test Dataset of CIFAR10 Generated by PGD Attacks ($\epsilon = 16/255, \alpha = 2/255, steps = 20$), Each Model is a Ensembling of Eight Votes

| **classes** | cnnbcebp | cnnbceban | **cnn01abp** | **classes** | cnnbcebp | cnnbceban | **cnn01abp** |
|---|---|---|---|---|---|---|---|
| **0 vs 1** | 6.95 (97.60) | 8.7 (96.90) | **19.95 (96.45)** | **2 vs 9** | 5.05 (97.45) | 4.3 (97.40) | **15.5 (96.85)** |
| **0 vs 2** | 0.7 (92.35) | 2.1 (92.75) | **2.85 (92.05)** | **3 vs 4** | 0.35 (91.70) | 0.1 (90.05) | **0.7 (89.95)** |
| **0 vs 3** | 6.85 (96.00) | 10.25 (95.70) | **15.05 (95.55)** | **3 vs 5** | 0 (83.25) | 0 (78.65) | **0 (78.80)** |
| **0 vs 4** | 1.15 (96.35) | 1.3 (96.10) | **3.4 (95.80)** | **3 vs 6** | 0.25 (91.90) | 0.1 (91.20) | **0.4 (92.10)** |
| **0 vs 5** | 6.9 (97.10) | 7.4 (96.80) | **14.15 (96.70)** | **3 vs 7** | 0.7 (93.00) | 0.15 (92.20) | **0.95 (92.35)** |
| **0 vs 6** | 11.05 (98.05) | 21.45 (97.65) | **15.9 (97.85)** | **3 vs 8** | 7.8 (96.80) | 5.8 (96.90) | **16.5 (96.70)** |
| **0 vs 7** | 6.45 (96.45) | 11.6 (96.65) | **15.8 (96.20)** | **3 vs 9** | 2.15 (96.40) | 0.95 (97.00) | **6.2 (95.65)** |
| **0 vs 8** | 1.05 (94.35) | 0.75 (93.25) | **1.4 (92.85)** | **4 vs 5** | 0.65 (92.50) | 1.5 (90.10) | **1.45 (91.30)** |
| **0 vs 9** | 6.5 (95.55) | 8.3 (94.90) | **15.65 (95.10)** | **4 vs 6** | 0 (94.45) | 0 (94.50) | **0.35 (92.95)** |
| **1 vs 2** | 8.65 (98.35) | 10.55 (98.60) | **22.3 (98.25)** | **4 vs 7** | 0.2 (92.10) | 0.35 (91.20) | **1 (91.60)** |
| **1 vs 3** | 6.45 (97.70) | 6.25 (97.95) | **14.95 (97.00)** | **4 vs 8** | 3.55 (97.65) | 7.95 (97.50) | **15.8 (97.30)** |
| **1 vs 4** | 6.2 (98.70) | 5.5 (98.85) | **15.35 (98.75)** | **4 vs 9** | 3.4 (98.40) | 5.75 (97.80) | **13.25 (98.15)** |
| **1 vs 5** | 5.75 (98.50) | 5.8 (99.05) | **16.6 (98.45)** | **5 vs 6** | 0.85 (94.90) | 0.95 (94.60) | **1.25 (94.40)** |
| **1 vs 6** | 7.1 (98.85) | 5.05 (98.55) | **12.4 (98.55)** | **5 vs 7** | 0.2 (93.25) | 0.2 (91.65) | **0.65 (91.15)** |
| **1 vs 7** | 8.3 (98.95) | 5.5 (99.05) | **14.65 (98.60)** | **5 vs 8** | 12.75 (97.70) | 8.7 (97.60) | **21.45 (97.40)** |
| **1 vs 8** | 1.9 (97.35) | 2 (96.85) | **8.7 (95.40)** | **5 vs 9** | 2.5 (97.85) | 1.15 (97.65) | **6.9 (97.00)** |
| **1 vs 9** | 0 (94.25) | 0.05 (94.10) | **0.45 (92.65)** | **6 vs 7** | 1 (97.35) | 0.7 (97.90) | **2.85 (97.35)** |
| **2 vs 3** | 0.35 (88.65) | 0.3 (86.80) | **0.5 (87.55)** | **6 vs 8** | 11.85 (98.25) | 13.8 (98.15) | **23.15 (97.95)** |
| **2 vs 4** | 0.05 (90.65) | 0 (87.30) | **0.05 (87.95)** | **6 vs 9** | 2.5 (98.85) | 1.15 (98.55) | **6.25 (98.30)** |
| **2 vs 5** | 0.5 (90.90) | 0.3 (88.85) | **1.05 (89.80)** | **7 vs 8** | 6.7 (98.55) | 15.75 (98.00) | **23.95 (97.90)** |
| **2 vs 6** | 0.55 (93.25) | 0.4 (91.25) | **1.4 (92.25)** | **7 vs 9** | 2.35 (98.00) | 2.75 (97.90) | **8.9 (96.95)** |
| **2 vs 7** | 0.6 (93.60) | 0.7 (92.75) | **2.1 (93.60)** | **8 vs 9** | 1.3 (96.30) | 0.7 (96.10) | **7.65 (95.15)** |
| **2 vs 8** | 4.15 (96.95) | 8.2 (96.45) | **14 (96.25)** | | | | |

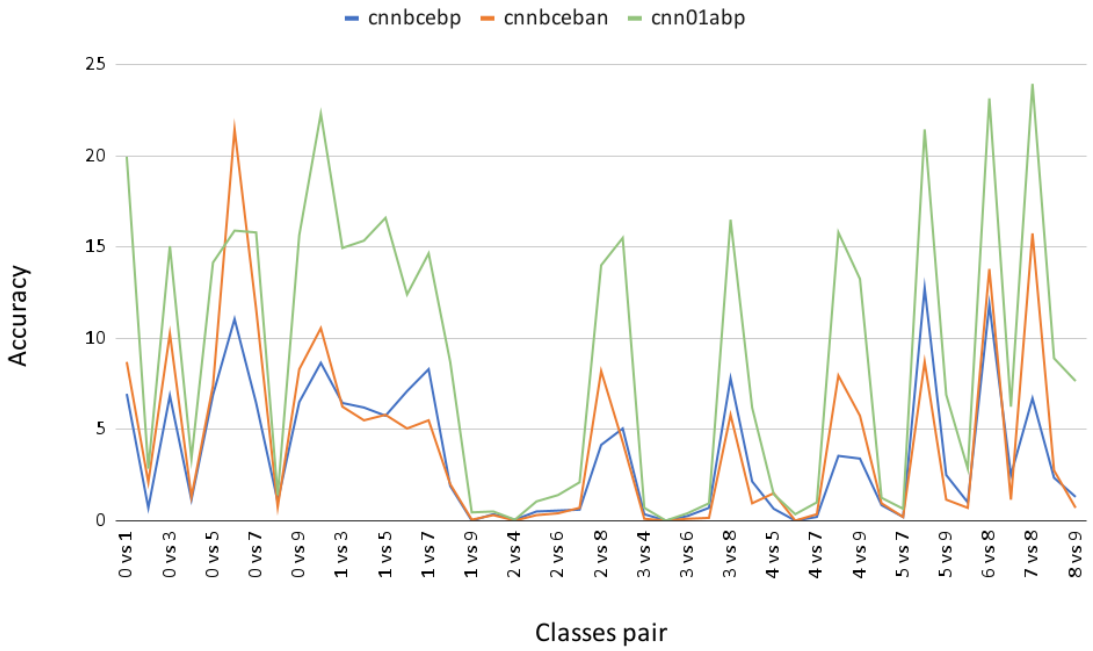


**Figure 5.7** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's accuracy on adversaries of CIFAR10 generated by PGD attack ($\epsilon = 16/255, \alpha = 2/255, steps = 20$). X-axis is classes pair, Y-axis is accuracy on adversaries.

on adversaries stuck in a position under 20%. But we see CNN-01-ABP is still the best one.

### 5.3 Adversaries' Transferability

Some researchers claimed that there is a large-dimensional continuous subspace in the adversarial sample space. This subspace is a very important part of the adversarial space and is shared by different models to achieve mobility. The dimension of the transferable adversarial subspace means that the learning boundaries learned by different models are incredibly close in the input domain, and these boundaries are far away from the data points in the adversarial direction [58]. This phenomenon happens in object detection [60] also.

Many works showed that adversaries' transferability is a key to do black-box attack successfully. In real world, most application service based on machine learning or deep learning techniques would not release the model or parameter details to users, they only allow users access to the input and output. Without the full information of a target model, powerful white-box attack would not be possibly applied to hack the system. In other worlds, doing hack on machine learning application through black-box attack, which a kind of attack not rely on model's information is more practical than doing white-box attack in real world. How to increase adversaries' transferability becomes a hot topic in black-box attack field. As the same time, many works showed their ability on improving adversaries' transferability [16, 63, 62].

In this section, we mainly focus on inner transferability and external transferability. Here, we define inner transferability is the transferability of adversaries among the same type of structure with the same loss function. These models have same structure, same optimization method and training detials, except that they are under different initialization. For example, in previous section we see baseline model *MLP-BCE-BP*'s accuracy on CIFAR10 and STL10. This accuracy is not a single model's performance but a majority vote results of 8 models. These eight models' training under different random seed setting and different initialized weights, other training details are the same. We evaluate each single vote's adversaries' transferability between each other, and show an example in Table 5.7. For external transferability, we defined it as the transferability of adversaries from models with different type of structure or loss function. The external adversaries' transferability results example is showed in Table 5.15. These attacks parameters' setting are the same as in the previous section.

**Table 5.7** Adversaries' Transferability of MLP-BCE-BP and MLP-01-SCD on Class Pair of 2 vs 4 From CIFAR10 Dataset

| **MLP-BCE-BP** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **source\target** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0** | **0.85** | 2.4 | 2.7 | 2.45 | 2.2 | 1.7 | 1.8 | 3.35 |
| **1** | 2.8 | **0.95** | 4.25 | 3.2 | 3.4 | 2.4 | 2.8 | 3.8 |
| **2** | 0.9 | 0.7 | **0.15** | 0.6 | 0.5 | 0.35 | 0.35 | 0.95 |
| **3** | 0.95 | 0.95 | 1.3 | **0.75** | 0.85 | 0.65 | 0.85 | 1.85 |
| **4** | 0.25 | 0.1 | 0.45 | 0.1 | **0.1** | 0.05 | 0.2 | 0.55 |
| **5** | 0.35 | 0.15 | 0.6 | 0.1 | 0.1 | **0.05** | 0.25 | 0.95 |
| **6** | 3.6 | 4.3 | 6 | 3.15 | 2.7 | 2.05 | **0.9** | 5.7 |
| **7** | 0.85 | 1.1 | 0.85 | 0.9 | 0.75 | 0.35 | 0.6 | **0.15** |
| **MLP-01-SCD** | | | | | | | | |
| **source\target** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0** | **0.3** | 61.4 | 67.8 | 57.85 | 60.45 | 63.3 | 63.65 | 57.85 |
| **1** | 65.85 | **0.3** | 64.3 | 61.35 | 61 | 64.1 | 70.85 | 58.45 |
| **2** | 60.75 | 62.25 | **0.15** | 58.65 | 60.05 | 55.95 | 66.7 | 57.3 |
| **3** | 59.5 | 55.2 | 62.3 | **0.2** | 64.6 | 54.6 | 65.65 | 61.9 |
| **4** | 63.1 | 62.2 | 60.7 | 65.2 | **0** | 56.15 | 52.75 | 60.25 |
| **5** | 61.65 | 64.65 | 60.55 | 58.55 | 59.5 | **0** | 59.65 | 57.2 |
| **6** | 64.25 | 70.1 | 56.9 | 64.7 | 58.65 | 64.1 | **0.4** | 57 |
| **7** | 59 | 69.45 | 59.15 | 64.95 | 60.05 | 63.9 | 54.4 | **0.3** |

**Table 5.8** Adversaries' Transferability of CNN-BCE-BP and CNN-01-ABP on Class Pair of 2 vs 4 From CIFAR10 Dataset

| **CNN-BCE-BP** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **source\target** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0** | **0** | 3.15 | 3.35 | 3.1 | 2.3 | 2.85 | 6.35 | 3.65 |
| **1** | 2.3 | **0** | 3.05 | 2.1 | 2 | 2.25 | 3.2 | 1.45 |
| **2** | 5.8 | 6.95 | **0** | 5.1 | 3.55 | 7.6 | 5.2 | 9.35 |
| **3** | 2.1 | 1.75 | 2.15 | **0** | 2.1 | 1.95 | 3.45 | 2 |
| **4** | 1.85 | 2.95 | 1.95 | 2.45 | **0** | 1.85 | 2.05 | 1.7 |
| **5** | 1.4 | 1.95 | 2.45 | 1.75 | 1.1 | **0** | 3.55 | 1.75 |
| **6** | 4.1 | 3.65 | 2.3 | 3.7 | 1.3 | 4.35 | **0** | 4.3 |
| **7** | 1.5 | 1.45 | 3.2 | 1.85 | 1 | 1.35 | 3.1 | **0** |
| **CNN-01-ABP** | | | | | | | | |
| **source\target** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| **0** | **0** | 62.75 | 57.5 | 59.65 | 51.65 | 57.4 | 60.6 | 57.15 |
| **1** | 51.2 | **0** | 48.8 | 53.85 | 50.15 | 53.95 | 51.35 | 53.4 |
| **2** | 50.65 | 55.2 | **0** | 55.25 | 46.85 | 51.05 | 50.25 | 50.65 |
| **3** | 47.45 | 53 | 49.65 | **0** | 43.75 | 52 | 53.6 | 51.4 |
| **4** | 57.4 | 65.75 | 59 | 61.35 | **0** | 59.55 | 63.55 | 59.45 |
| **5** | 51.85 | 59.4 | 51.05 | 55.35 | 48.5 | **0** | 47.6 | 53.5 |
| **6** | 56.6 | 59.85 | 51.9 | 59.4 | 53.25 | 51.05 | **0** | 53.6 |
| **7** | 56.1 | 61.35 | 57.2 | 59.15 | 53.6 | 58.5 | 56.25 | **0** |

### 5.3.1 Adversaries' Inner Transferability

Table 5.7 and 5.8 is an example of MLP's and CNN's inner adversaries' transferability on class pair 2 vs 4 of CIFAR10. For example, we trained 8 MLP-BCE-BP models $\{M_0, M_1, ..., M_7\}$ on class pair 2 vs 4 of CIFAR10 $\{X, Y\}$ with different random seed $\{rd_0, rd_1, ..., rd_7\}$, and then do PGD attack for each model to get their adversaries $\{adv_0, adv_1, ..., adv_7\}$. So in the table, we record accuracy $Acc$ at position $row\_i$, $col\_j$ as $M_j$'s accuracy on adversaries $adv_i$. In other words, we defined the accuracy in formula,

$$\begin{aligned} Acc_{ij} &= \frac{1}{n}\sum_{k}^{n}(M_j(x_k) = y_k) \\ &\text{subject to } x_k \in adv_i, y_k \in Y \end{aligned} \tag{5.4}$$

In the Table 5.7, for MLP-BCE-BP, $M_1$'s accuracy on $M_0$'s adversaries is 2.4%, $M_0$'s accuracy on $M_1$'s adversaries is 2.8%, and $M_0$'s accuracy on the adversaries from itself is 0.85%. This means $M_0$ and $M_1$'s adversaries could be shared between each other, and this phenomenon is consistent with the argument that *the decision boundaries learned by different models are incredibly close in the input domain* [57]. For MLP-01-SCD, $M_1$'s accuracy on $M_0$'s adversaries is 61.4%, $M_0$'s accuracy on $M_1$'s adversaries is 65.85%, and $M_0$'s accuracy on the adversaries from itself is 0.3%. Compared to MLP-BCE-BP's adversaries transferability, we can see MLP-01-SCD's adversaries transferability is obviously lower than MLP-BCE-BP's in this class pair. Higher accuracy on other models' adversaries means MLP-01-SCD's adversaries could hardly transfer to models with the same structure, indicating that the transferability attack on MLP-01-SCD would be ineffective.

In the Table 5.8, for CNN-BCE-BP, $M_1$'s accuracy on $M_0$'s adversaries is 3.15%, $M_0$'s accuracy on $M_1$'s adversaries is 2.3%, and $M_0$'s accuracy on the adversaries from itself is 0%. For CNN-01-ABP, $M_1$'s accuracy on $M_0$'s adversaries is 62.75%, $M_0$'s

accuracy on $M_1$'s adversaries is 51.2%, and $M_0$'s accuracy on the adversaries from itself is 0%. The situation is similar as we see in MLP's results, which means our zero-one loss models' adversarial transferability attribute does not change as network structure changes from MLP to CNN and even becomes deeper.

### 5.3.2 Inner Adversarial Transferability Results

We evaluated our baseline models' inner adversaries transferability under FGSM and PGD attack, averaged accuracy like in Table 5.7 and 5.8 exclude accuracy on the diagnoal (white-box attack), averaged accuracy for each class pair and recorded them in Table 5.8, 5.9, 5.10, 5.12, 5.13, and 5.14 and their corresponding curve in Figures 5.9, 5.10, 5.11, 5.11, 5.12, and 5.13. From these figures we can see MLP-01-SCD and CNN-01-ABP's adversaries' transferability is the weakest in their corresponding groups. Optimizing zero-one loss will give non-unique solutions or decision boundary, which makes each models' adversaries generated by FGSM or PGD can only succeed in itself rather than other models. This is different from that cross-entropy loss optimized by SGD probably bring similar decision boundary for models if they are under the same structure and trained on the same dataset.

**Table 5.9** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Average Accuracy of Inner Adversarial Transferability Results Under FGSM Attacks (A)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** |
| 0 vs 1 | 6.02 | 4.98 | 79.81 | 69.76 | 41.08 | 8.06 | 70.93 | 74.74 |
| 0 vs 2 | 1.45 | 3.35 | 77.00 | 56.63 | 40.91 | 26.05 | 81.18 | 83.11 |
| 0 vs 3 | 4.50 | 14.43 | 80.98 | 70.92 | 48.80 | 34.38 | 78.77 | 81.10 |
| 0 vs 4 | 0.87 | 5.32 | 82.55 | 63.48 | 62.91 | 43.65 | 77.82 | 78.90 |
| 0 vs 5 | 3.99 | 10.82 | 82.59 | 72.78 | 60.59 | 20.36 | 79.11 | 83.12 |
| 0 vs 6 | 3.81 | 16.30 | 87.00 | 73.26 | 52.71 | 21.06 | 79.17 | 81.62 |
| 0 vs 7 | 1.75 | 3.52 | 80.56 | 63.70 | 63.93 | 56.38 | 77.98 | 84.08 |
| 0 vs 8 | 1.27 | 2.29 | 69.24 | 44.62 | 47.49 | 4.49 | 71.39 | 70.48 |
| 0 vs 9 | 10.59 | 5.27 | 75.77 | 62.50 | 48.98 | 14.06 | 69.56 | 77.63 |
| 1 vs 2 | 3.46 | 7.62 | 82.41 | 72.51 | 38.69 | 52.86 | 78.18 | 85.11 |
| 1 vs 3 | 6.62 | 7.68 | 80.03 | 71.85 | 28.94 | 9.35 | 58.16 | 56.52 |
| 1 vs 4 | 2.67 | 5.29 | 83.61 | 75.97 | 45.95 | 3.11 | 58.75 | 57.83 |
| 1 vs 5 | 2.69 | 7.03 | 82.28 | 74.38 | 41.59 | 19.74 | 55.97 | 52.03 |
| 1 vs 6 | 1.65 | 8.60 | 85.38 | 76.44 | 33.04 | 1.28 | 61.74 | 65.57 |
| 1 vs 7 | 3.92 | 7.03 | 81.96 | 72.12 | 43.04 | 5.23 | 56.76 | 58.30 |
| 1 vs 8 | 6.90 | 4.05 | 76.78 | 57.66 | 40.58 | 33.99 | 72.95 | 85.17 |
| 1 vs 9 | 0.96 | 2.89 | 65.35 | 42.62 | 32.34 | 47.01 | 75.70 | 83.63 |
| 2 vs 3 | 3.48 | 12.05 | 65.74 | 51.54 | 28.26 | 44.79 | 77.39 | 82.73 |
| 2 vs 4 | 0.67 | 0.87 | 60.46 | 32.24 | 44.22 | 49.93 | 82.28 | 86.33 |
| 2 vs 5 | 3.52 | 6.32 | 66.89 | 48.79 | 31.96 | 49.70 | 74.20 | 84.69 |
| 2 vs 6 | 1.33 | 2.44 | 70.54 | 43.77 | 30.88 | 62.44 | 76.10 | 83.35 |
| 2 vs 7 | 0.71 | 2.37 | 72.26 | 45.59 | 25.89 | 59.53 | 78.51 | 85.01 |
| 2 vs 8 | 2.73 | 10.21 | 82.48 | 67.47 | 24.81 | 11.20 | 73.28 | 77.53 |

**Table 5.10** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Average Accuracy of Inner Adversarial Transferability Results Under FGSM Attacks (B)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** |
| 2 vs 9 | 4.87 | 11.66 | 81.88 | 67.40 | 26.68 | 7.72 | 60.03 | 60.39 |
| 3 vs 4 | 1.12 | 3.09 | 71.33 | 51.62 | 44.95 | 15.51 | 59.79 | 53.85 |
| 3 vs 5 | 1.11 | 12.28 | 57.94 | 31.39 | 38.89 | 10.90 | 52.21 | 50.20 |
| 3 vs 6 | 3.33 | 4.99 | 70.36 | 50.64 | 28.62 | 1.50 | 61.59 | 60.00 |
| 3 vs 7 | 1.12 | 1.90 | 70.47 | 38.47 | 19.12 | 1.81 | 55.56 | 52.37 |
| 3 vs 8 | 9.11 | 14.43 | 83.17 | 73.69 | 52.44 | 60.33 | 81.19 | 85.39 |
| 3 vs 9 | 3.51 | 8.45 | 78.61 | 66.22 | 46.41 | 45.93 | 75.78 | 80.50 |
| 4 vs 5 | 3.80 | 6.17 | 71.23 | 55.71 | 53.38 | 11.08 | 59.03 | 59.97 |
| 4 vs 6 | 0.50 | 2.28 | 67.41 | 37.32 | 45.48 | 2.57 | 64.64 | 63.33 |
| 4 vs 7 | 0.45 | 1.82 | 67.09 | 44.28 | 34.07 | 4.06 | 61.24 | 62.70 |
| 4 vs 8 | 6.55 | 16.64 | 84.29 | 65.31 | 49.92 | 68.37 | 82.98 | 87.65 |
| 4 vs 9 | 4.08 | 10.95 | 82.44 | 69.21 | 30.60 | 42.87 | 81.00 | 84.07 |
| 5 vs 6 | 1.24 | 3.09 | 73.55 | 54.90 | 33.49 | 3.69 | 55.47 | 51.99 |
| 5 vs 7 | 1.31 | 3.02 | 69.91 | 41.76 | 29.38 | 0.62 | 55.51 | 52.14 |
| 5 vs 8 | 5.16 | 16.88 | 84.06 | 75.13 | 59.17 | 71.89 | 80.07 | 86.72 |
| 5 vs 9 | 4.59 | 13.99 | 82.22 | 71.83 | 47.15 | 48.44 | 79.46 | 82.36 |
| 6 vs 7 | 1.98 | 2.43 | 78.11 | 51.57 | 34.94 | 1.96 | 60.77 | 61.16 |
| 6 vs 8 | 11.43 | 19.68 | 89.06 | 73.48 | 59.15 | 60.18 | 84.30 | 86.24 |
| 6 vs 9 | 3.37 | 10.81 | 83.52 | 71.86 | 39.39 | 55.69 | 77.64 | 81.36 |
| 7 vs 8 | 5.88 | 8.15 | 83.90 | 66.43 | 50.35 | 63.09 | 83.07 | 88.89 |
| 7 vs 9 | 3.69 | 9.60 | 78.68 | 60.91 | 46.26 | 49.90 | 79.12 | 84.06 |
| 8 vs 9 | 1.92 | 4.88 | 74.90 | 55.88 | 42.06 | 3.69 | 61.68 | 67.21 |

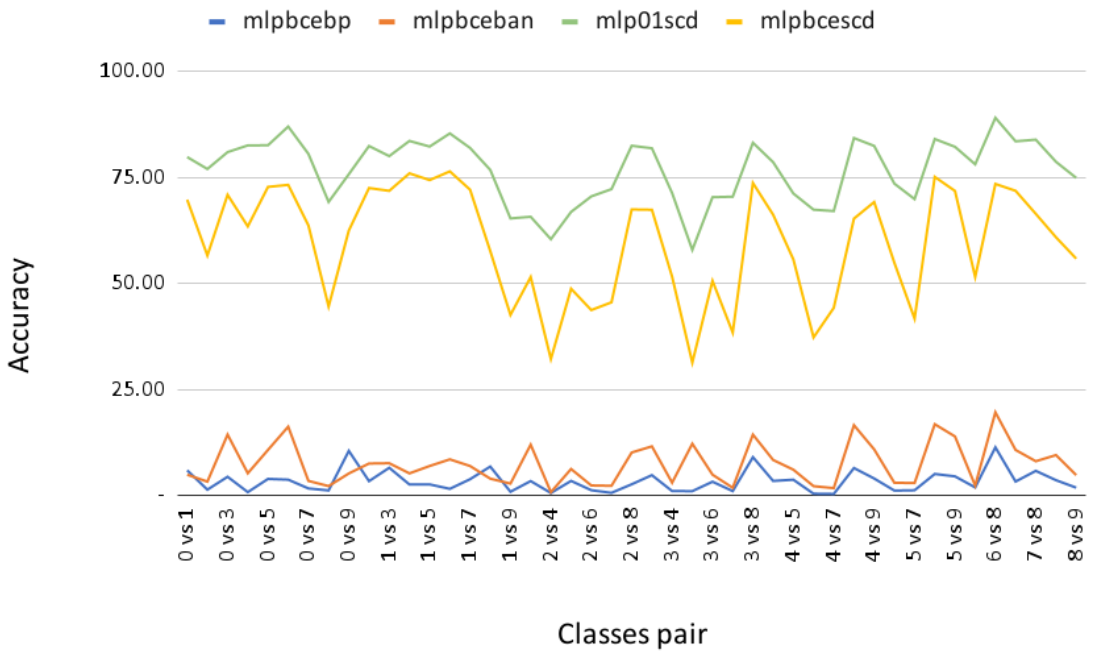


**Figure 5.8** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's average accuracy of inner adversarial transferability results exclude white-box attack part. Adversaries are test dataset of CIFAR10 by FGSM attacks. X-axis is classes pair, Y-axis is accuracy.

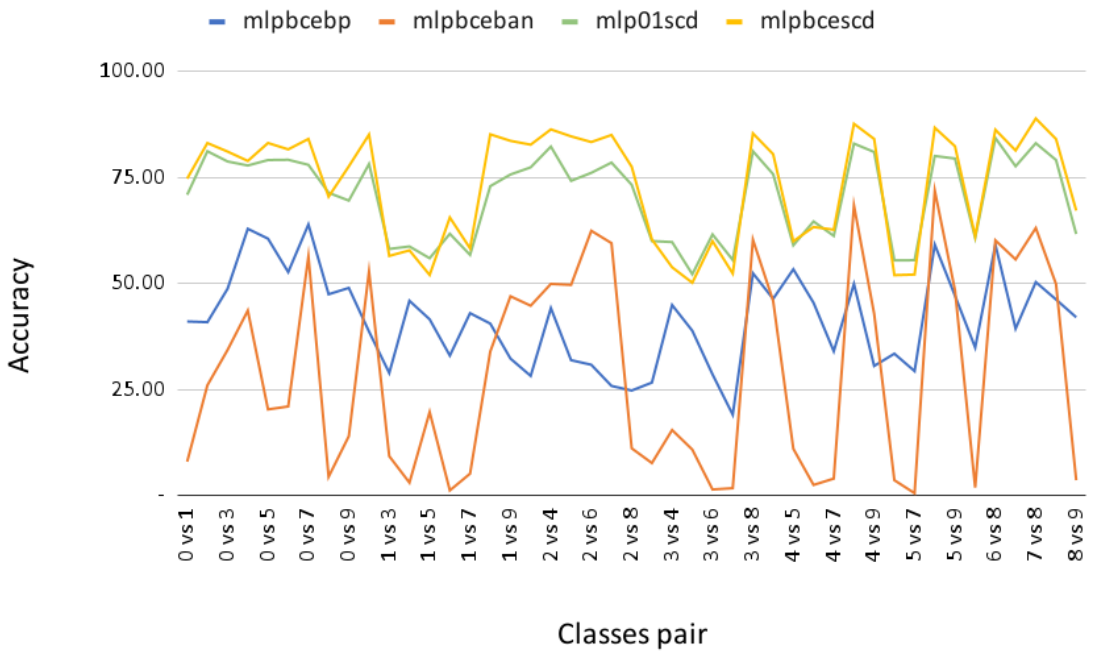


**Figure 5.9** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's average accuracy of inner adversarial transferability results exclude white-box attack part. Adversaries are test dataset of STL10 by FGSM attacks. X-axis is classes pair, Y-axis is accuracy.

**Table 5.11** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's Average Accuracy of Inner Adversarial Transferability Results Under FGSM Attacks

| **classes** | cnnbcebp | cnnbceban | **cnn01abp** | **classes** | cnnbcebp | cnnbceban | **cnn01abp** |
|---|---|---|---|---|---|---|---|
| **0 vs 1** | 57.07 | 78.99 | **87.39** | **2 vs 9** | 50.7 | 78.67 | **85.53** |
| **0 vs 2** | 26.86 | 62.48 | **72.08** | **3 vs 4** | 18.17 | 39.12 | **66.14** |
| **0 vs 3** | 48.28 | 75.06 | **84.3** | **3 vs 5** | 6.28 | 21.85 | **54.58** |
| **0 vs 4** | 25.44 | 54.68 | **75.3** | **3 vs 6** | 28.31 | 51.44 | **66.81** |
| **0 vs 5** | 49.51 | 73.4 | **86.52** | **3 vs 7** | 9.57 | 36.73 | **63.34** |
| **0 vs 6** | 67.58 | 79.55 | **86.47** | **3 vs 8** | 54.06 | 74.34 | **87.26** |
| **0 vs 7** | 39.04 | 68.86 | **83.98** | **3 vs 9** | 42.83 | 72.06 | **79.37** |
| **0 vs 8** | 20.82 | 51.07 | **71.73** | **4 vs 5** | 28.92 | 49.71 | **70.48** |
| **0 vs 9** | 53.12 | 72.71 | **83.23** | **4 vs 6** | 11.65 | 42.63 | **64.32** |
| **1 vs 2** | 51.2 | 83.75 | **87.62** | **4 vs 7** | 9.08 | 41.99 | **65.5** |
| **1 vs 3** | 45.96 | 80.14 | **84.54** | **4 vs 8** | 46.53 | 71.59 | **83.92** |
| **1 vs 4** | 35.86 | 75.78 | **86.99** | **4 vs 9** | 38.19 | 76.27 | **83.66** |
| **1 vs 5** | 48.19 | 82.31 | **88.85** | **5 vs 6** | 32.54 | 48.53 | **69.05** |
| **1 vs 6** | 53.15 | 78.56 | **88.8** | **5 vs 7** | 13.49 | 37.94 | **67.5** |
| **1 vs 7** | 50.08 | 82.3 | **89.6** | **5 vs 8** | 66.38 | 78.36 | **89.74** |
| **1 vs 8** | 31.66 | 66.79 | **80.29** | **5 vs 9** | 50.91 | 76.23 | **83.24** |
| **1 vs 9** | 24.1 | 53.1 | **73.26** | **6 vs 7** | 24.22 | 59.97 | **77.27** |
| **2 vs 3** | 15.27 | 38.37 | **62.32** | **6 vs 8** | 63.45 | 79.81 | **88.81** |
| **2 vs 4** | 9.18 | 30.16 | **57.3** | **6 vs 9** | 50.93 | 76.08 | **86.76** |
| **2 vs 5** | 21.46 | 43.17 | **67.41** | **7 vs 8** | 50.72 | 73.3 | **89.19** |
| **2 vs 6** | 27.12 | 54.72 | **69.07** | **7 vs 9** | 37.52 | 69.75 | **83.7** |
| **2 vs 7** | 22.43 | 51.53 | **71.75** | **8 vs 9** | 39.1 | 64.47 | **77.41** |
| **2 vs 8** | 45.06 | 73.18 | **84.27** | | | | |

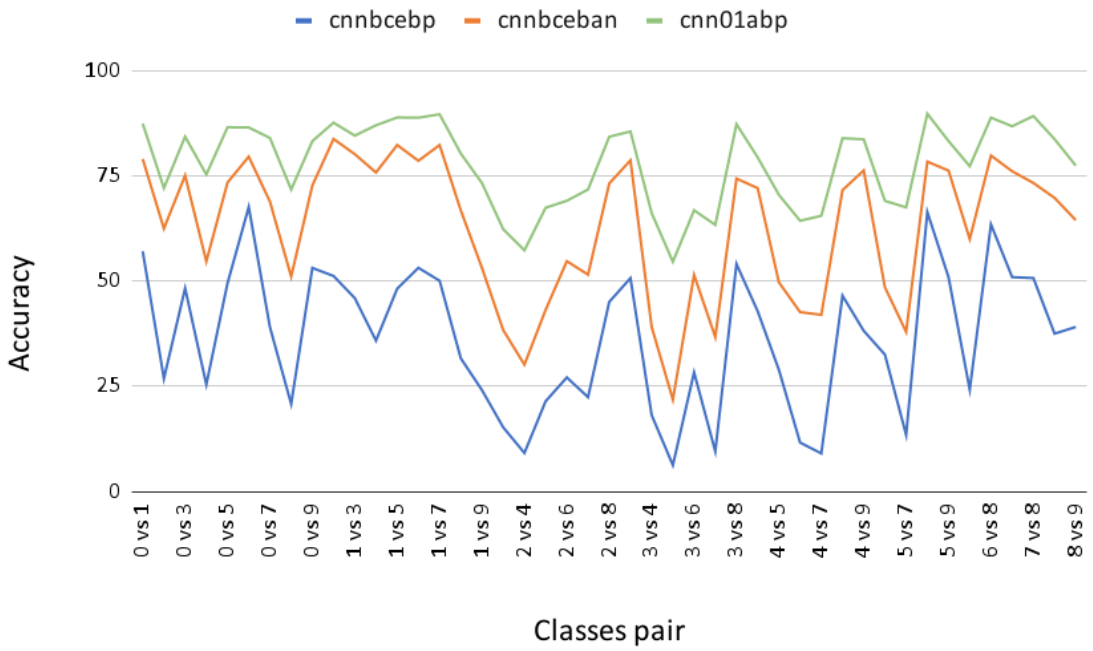


**Figure 5.10** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's average accuracy of inner adversarial transferability results exclude white-box attack part. Adversaries are test dataset of CIFAR10 by FGSM attacks. X-axis is classes pair, Y-axis is accuracy.

**Table 5.12** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Average Accuracy of Inner Adversarial Transferability Results Under PGD Attacks (A)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** |
| 0 vs 1 | 5.30 | 2.48 | 80.04 | 76.04 | 50.74 | 6.10 | 71.36 | 78.03 |
| 0 vs 2 | 1.16 | 1.57 | 76.48 | 64.70 | 53.89 | 25.42 | 77.24 | 84.60 |
| 0 vs 3 | 4.82 | 8.56 | 80.58 | 76.22 | 53.93 | 33.79 | 79.48 | 83.18 |
| 0 vs 4 | 1.06 | 1.85 | 82.09 | 71.67 | 68.34 | 42.14 | 77.77 | 81.76 |
| 0 vs 5 | 4.50 | 6.21 | 82.57 | 78.30 | 67.75 | 17.39 | 78.78 | 84.28 |
| 0 vs 6 | 4.20 | 11.52 | 86.15 | 79.19 | 64.18 | 20.27 | 80.64 | 84.30 |
| 0 vs 7 | 2.02 | 1.13 | 79.57 | 72.89 | 76.23 | 58.84 | 78.40 | 85.95 |
| 0 vs 8 | 1.76 | 1.06 | 68.79 | 54.90 | 54.73 | 3.18 | 70.02 | 75.20 |
| 0 vs 9 | 9.53 | 2.61 | 75.38 | 69.38 | 58.76 | 12.63 | 69.50 | 79.24 |
| 1 vs 2 | 3.40 | 3.38 | 82.14 | 78.78 | 51.22 | 56.87 | 73.60 | 86.63 |
| 1 vs 3 | 6.06 | 3.74 | 79.52 | 77.44 | 38.81 | 9.99 | 58.24 | 61.29 |
| 1 vs 4 | 2.26 | 1.41 | 83.96 | 80.79 | 54.40 | 2.76 | 58.46 | 63.85 |
| 1 vs 5 | 2.40 | 2.58 | 81.74 | 80.21 | 50.63 | 20.83 | 55.58 | 58.11 |
| 1 vs 6 | 1.48 | 2.87 | 85.37 | 82.27 | 46.81 | 1.00 | 62.11 | 69.38 |
| 1 vs 7 | 2.77 | 1.57 | 81.44 | 78.83 | 52.07 | 5.67 | 56.88 | 60.12 |
| 1 vs 8 | 6.37 | 2.30 | 76.31 | 64.42 | 51.78 | 32.39 | 73.07 | 86.86 |
| 1 vs 9 | 1.34 | 2.10 | 64.70 | 49.70 | 42.45 | 49.24 | 75.50 | 85.37 |
| 2 vs 3 | 4.01 | 10.05 | 65.05 | 57.56 | 37.74 | 45.12 | 75.88 | 84.44 |
| 2 vs 4 | 1.57 | 0.35 | 61.29 | 40.69 | 51.63 | 53.06 | 76.16 | 87.48 |
| 2 vs 5 | 4.55 | 5.18 | 67.25 | 56.48 | 35.83 | 52.35 | 77.45 | 86.15 |
| 2 vs 6 | 1.66 | 1.05 | 68.91 | 53.98 | 33.84 | 65.61 | 76.06 | 85.52 |
| 2 vs 7 | 0.62 | 1.09 | 71.19 | 57.87 | 34.24 | 62.50 | 79.04 | 86.52 |
| 2 vs 8 | 2.39 | 4.55 | 82.09 | 76.38 | 33.65 | 11.20 | 73.66 | 80.03 |

**Table 5.13** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's Average Accuracy of Inner Adversarial Transferability Results Under PGD Attacks (B)

| | **CIFAR10** | | | | **STL10** | | | |
|---|---|---|---|---|---|---|---|---|
| **classes** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** | mlpbcebp | mlpbceban | **mlp01scd** | **mlpbcescd** |
| 2 vs 9 | 4.37 | 5.79 | 82.41 | 76.78 | 38.11 | 7.73 | 60.11 | 64.23 |
| 3 vs 4 | 1.44 | 1.62 | 71.19 | 57.85 | 53.34 | 18.04 | 59.96 | 61.09 |
| 3 vs 5 | 2.01 | 13.69 | 57.10 | 36.90 | 45.04 | 12.28 | 53.43 | 55.23 |
| 3 vs 6 | 3.14 | 3.53 | 69.49 | 56.82 | 40.44 | 1.21 | 60.56 | 66.00 |
| 3 vs 7 | 0.93 | 0.36 | 69.23 | 52.45 | 27.08 | 1.66 | 55.20 | 58.10 |
| 3 vs 8 | 8.26 | 7.52 | 82.65 | 79.80 | 58.47 | 63.68 | 81.26 | 87.14 |
| 3 vs 9 | 3.32 | 3.71 | 78.36 | 74.56 | 51.67 | 48.02 | 76.35 | 83.04 |
| 4 vs 5 | 4.22 | 4.63 | 68.66 | 64.34 | 58.02 | 11.43 | 58.23 | 64.94 |
| 4 vs 6 | 0.80 | 1.22 | 66.98 | 50.72 | 56.90 | 1.35 | 63.91 | 68.96 |
| 4 vs 7 | 0.42 | 1.15 | 70.35 | 58.15 | 47.70 | 2.70 | 61.36 | 68.01 |
| 4 vs 8 | 5.91 | 10.13 | 84.13 | 74.76 | 53.66 | 72.46 | 81.94 | 89.13 |
| 4 vs 9 | 4.06 | 5.39 | 82.62 | 78.99 | 37.01 | 44.93 | 81.08 | 85.95 |
| 5 vs 6 | 1.32 | 1.73 | 72.73 | 61.16 | 42.98 | 4.24 | 55.38 | 59.28 |
| 5 vs 7 | 1.11 | 1.32 | 68.67 | 53.14 | 41.35 | 0.46 | 56.29 | 57.48 |
| 5 vs 8 | 4.95 | 9.00 | 83.77 | 81.29 | 62.84 | 75.54 | 82.94 | 88.52 |
| 5 vs 9 | 3.98 | 7.49 | 81.94 | 78.73 | 52.88 | 52.72 | 77.70 | 84.43 |
| 6 vs 7 | 1.94 | 0.49 | 77.29 | 67.19 | 47.72 | 1.24 | 59.58 | 65.79 |
| 6 vs 8 | 10.70 | 9.56 | 88.43 | 82.74 | 64.07 | 63.61 | 84.55 | 88.50 |
| 6 vs 9 | 3.33 | 4.96 | 83.46 | 81.13 | 48.91 | 59.71 | 76.41 | 83.35 |
| 7 vs 8 | 5.11 | 2.02 | 83.65 | 75.92 | 60.09 | 66.58 | 82.45 | 90.01 |
| 7 vs 9 | 3.57 | 3.89 | 77.99 | 70.31 | 56.72 | 55.08 | 79.39 | 85.88 |
| 8 vs 9 | 1.82 | 2.35 | 74.60 | 66.69 | 51.97 | 3.47 | 62.42 | 70.75 |

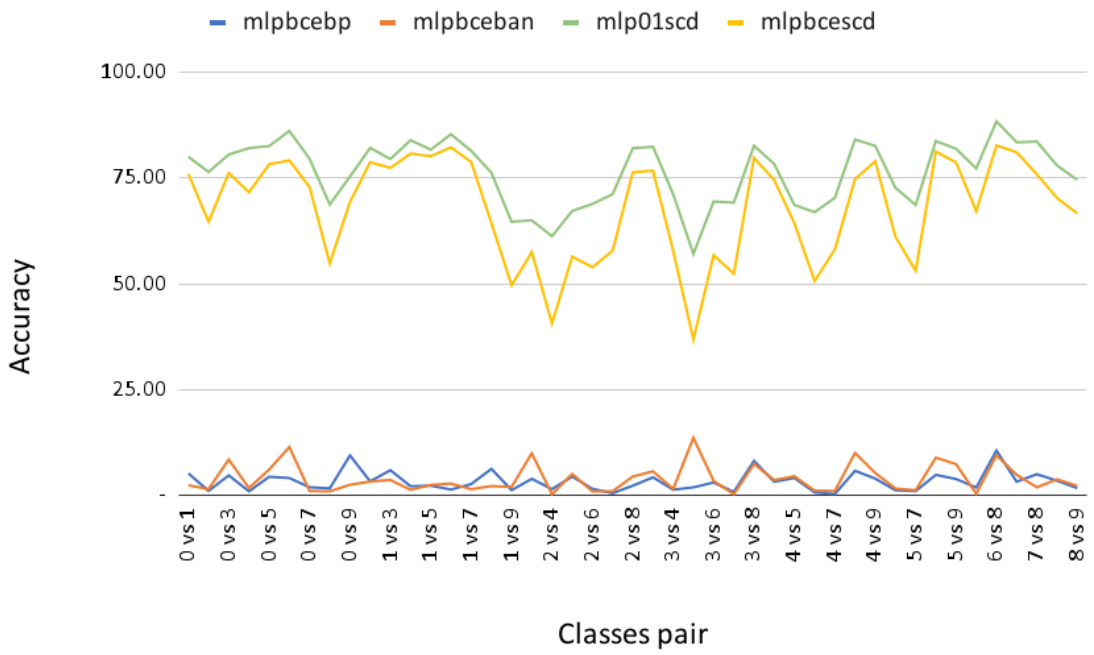


**Figure 5.11** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's average accuracy of inner adversarial transferability results exclude white-box attack part. Adversaries are test dataset of CIFAR10 by PGD attacks. X-axis is classes pair, Y-axis is accuracy.

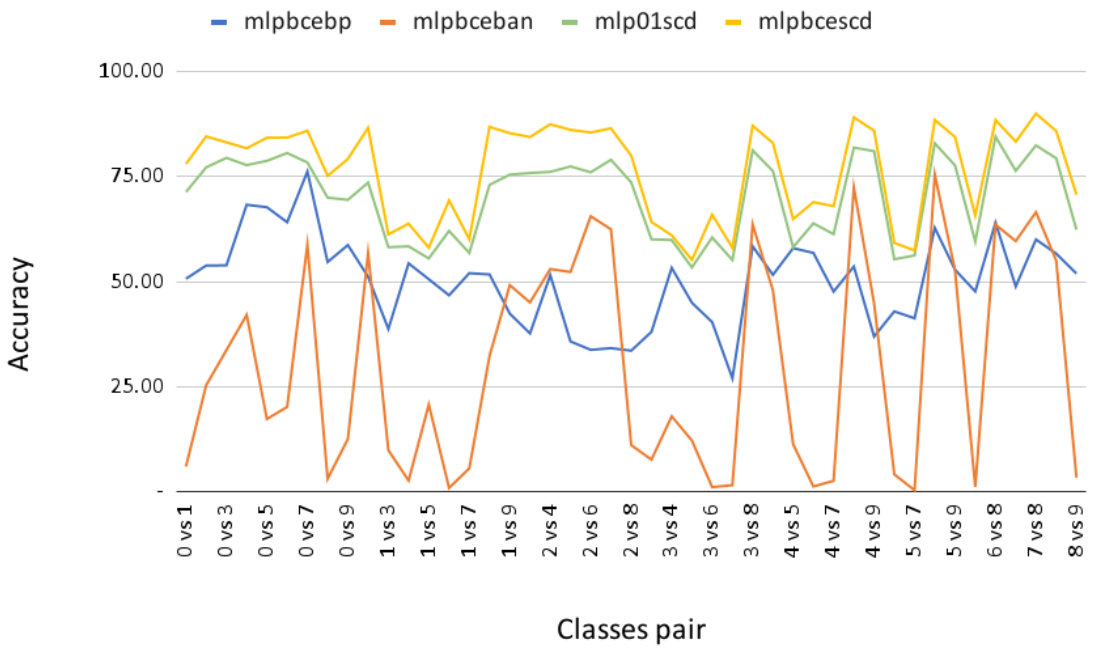


**Figure 5.12** MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, MLP-BCE-SCD's average accuracy of inner adversarial transferability results exclude white-box attack part. Adversaries are test dataset of STL10 by PGD attacks. X-axis is classes pair, Y-axis is accuracy.

**Table 5.14** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's Average Accuracy of Inner Adversarial Transferability Results Under PGD Attacks

| **classes** | cnnbcebp | cnnbceban | **cnn01abp** | **classes** | cnnbcebp | cnnbceban | **cnn01abp** |
|---|---|---|---|---|---|---|---|
| **0 vs 1** | 41.79 | 72.25 | **89.09** | **2 vs 9** | 37.84 | 74.13 | **88.27** |
| **0 vs 2** | 16.93 | 53.52 | **72.05** | **3 vs 4** | 7.21 | 22.14 | **66.71** |
| **0 vs 3** | 37.53 | 65.43 | **84.76** | **3 vs 5** | 2.35 | 16.82 | **52.62** |
| **0 vs 4** | 16.09 | 42.96 | **78.66** | **3 vs 6** | 14.62 | 39.85 | **69.73** |
| **0 vs 5** | 37.39 | 62.75 | **88.06** | **3 vs 7** | 5.34 | 25.13 | **66.29** |
| **0 vs 6** | 53.79 | 72.48 | **90.17** | **3 vs 8** | 42.71 | 63.09 | **89.02** |
| **0 vs 7** | 30.86 | 56.62 | **86.60** | **3 vs 9** | 26.80 | 69.32 | **84.24** |
| **0 vs 8** | 12.16 | 38.48 | **70.47** | **4 vs 5** | 14.35 | 37.01 | **69.99** |
| **0 vs 9** | 36.27 | 65.88 | **86.04** | **4 vs 6** | 4.36 | 25.83 | **67.37** |
| **1 vs 2** | 39.78 | 81.45 | **88.25** | **4 vs 7** | 4.37 | 35.69 | **63.99** |
| **1 vs 3** | 31.47 | 77.80 | **85.33** | **4 vs 8** | 31.97 | 61.70 | **84.60** |
| **1 vs 4** | 27.03 | 70.17 | **88.32** | **4 vs 9** | 26.55 | 72.34 | **88.45** |
| **1 vs 5** | 35.35 | 77.75 | **90.21** | **5 vs 6** | 13.88 | 36.44 | **74.48** |
| **1 vs 6** | 35.62 | 70.96 | **90.73** | **5 vs 7** | 6.30 | 26.11 | **69.31** |
| **1 vs 7** | 39.67 | 75.82 | **91.12** | **5 vs 8** | 53.09 | 66.72 | **90.38** |
| **1 vs 8** | 19.98 | 58.27 | **82.38** | **5 vs 9** | 36.79 | 72.53 | **88.42** |
| **1 vs 9** | 12.23 | 43.94 | **75.87** | **6 vs 7** | 13.40 | 51.77 | **78.30** |
| **2 vs 3** | 6.38 | 23.85 | **60.74** | **6 vs 8** | 50.18 | 72.07 | **89.99** |
| **2 vs 4** | 2.97 | 12.25 | **54.67** | **6 vs 9** | 34.81 | 68.78 | **88.67** |
| **2 vs 5** | 12.24 | 30.98 | **67.02** | **7 vs 8** | 37.29 | 59.91 | **89.96** |
| **2 vs 6** | 13.69 | 44.98 | **71.40** | **7 vs 9** | 27.31 | 63.14 | **86.48** |
| **2 vs 7** | 14.60 | 41.87 | **72.68** | **8 vs 9** | 22.37 | 55.13 | **81.85** |
| **2 vs 8** | 30.63 | 63.53 | **84.81** | | | | |

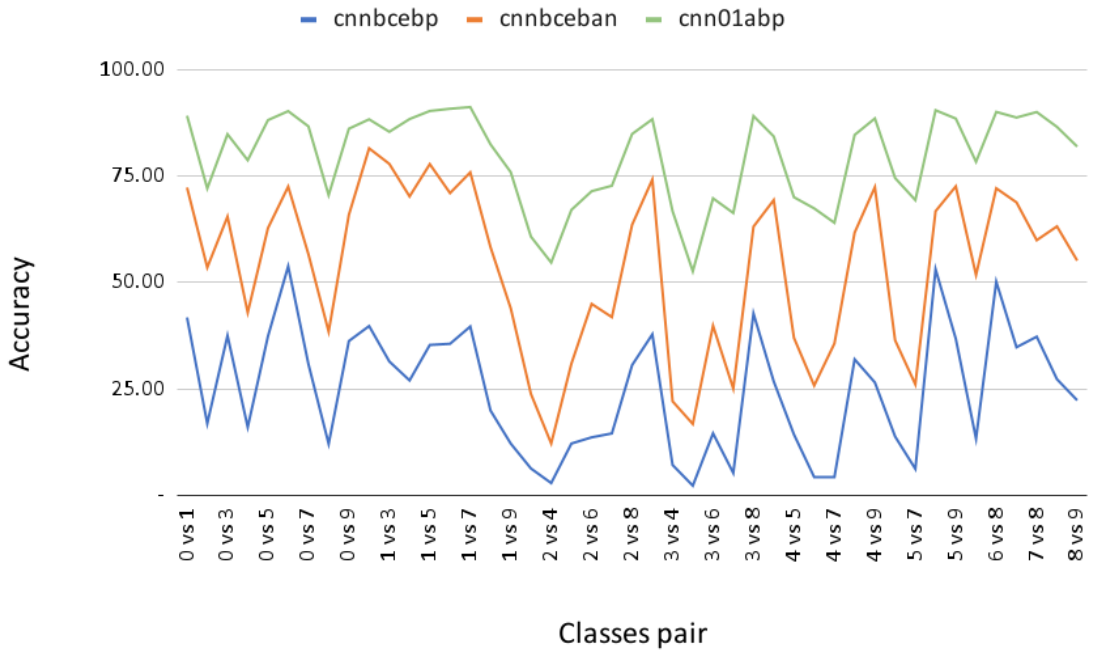


**Figure 5.13** CNN-BCE-BP, CNN-BCE-BAN, CNN-01-ABP's average accuracy of inner adversarial transferability results exclude white-box attack part. Adversaries are test dataset of CIFAR10 by PGD attacks. X-axis is classes pair, Y-axis is accuracy.

### 5.3.3 Adversaries' External Transferability

**Table 5.15** Adversaries' Transferability Between MLP-BCE-BP, MLP-BCE-BAN, MLP-01-SCD, and MLP-BCE-SCD on CIFAR10 Dataset

| **0 vs 1** | | | | |
|---|---|---|---|---|
| **model** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| **mlpbcebp** | 0.4 | 0.45 | 66.45 | 3.7 |
| **mlpbceban** | 1.45 | 0.1 | 69.95 | 4.25 |
| **mlp01scd** | 81.95 | 75.6 | 26.9 | 71.5 |
| **mlpbcescd** | 26.05 | 14.95 | 65.45 | 0.15 |
| **0 vs 2** | | | | |
| **model** | **mlpbcebp** | **mlpbceban** | **mlp01scd** | **mlpbcescd** |
| **mlpbcebp** | 0 | 0 | 66.2 | 0.2 |
| **mlpbceban** | 0.15 | 0 | 66.45 | 0.35 |
| **mlp01scd** | 80.35 | 79.85 | 34.6 | 76.6 |
| **mlpbcescd** | 11.95 | 7.95 | 61.2 | 0 |

Tables 5.16 shows an example of adversaries' external transferability between CNN baseline models in class pair 0 vs 1 and class pair 0 vs 2 on the CIFAR10 dataset. We generate adversaries through PGD attack for models list in the first column and then evaluate the accuracy of models list in the first row. We only evaluate adversarial transferability under PGD attacksFor example, in class pair 0 vs 1, MLP-BCE-BAN's accuracy on MLP-BCE-BP's adversaries is 0.45%, almost the same as MLP-BCE-BP's. This means MLP-BCE-BP's adversaries can be transferred to MLP-BCE-BAN. In other words, if we want to generate adversaries for attacking MLP-BCE-BAN, we do not need accurate weights' information from MLP-BCE-BAN. We could instead generate adversaries for attacking an MLP-BCE-BP model trained on the same dataset to succeed. However, MLP-01-SCD's accuracy evaluated on MLP-BCE-

**Table 5.16** Adversaries' Transferability Between CNN-BCE-BP, CNN-BCE-BAN, and CNN-01-ABP on CIFAR10 Dataset

| **0 vs 1** | | | |
|---|---|---|---|
| **model** | **cnnbcebp** | **cnnbceban** | **cnn01abp** |
| **cnnbcebp** | 6.95 | 17.35 | 47.9 |
| **cnnbceban** | 40.85 | 8.7 | 60.8 |
| **cnn01abp** | 83.6 | 80.15 | 19.95 |
| **0 vs 2** | | | |
| **model** | **cnnbcebp** | **cnnbceban** | **cnn01abp** |
| **cnnbcebp** | 0.7 | 12.8 | 15.4 |
| **cnnbceban** | 19.8 | 2.1 | 22.8 |
| **cnn01abp** | 51.55 | 51.85 | 2.85 |

BP's adversaries is 66.45%, much higher than 0.45%, indicating adversaries transfer from model MLP-BCE-BP to MLP-01-SCD is harder than transfer to MLP-BCE-BAN. MLP-BCE-Bp's accuracy evaluated on MLP-BCE-BAN's adversaries is 1.45%, which is 68% lower than MLP-01-SCD's accuracy evaluated on the same adversaries MLP-BCE-BAN's adversaries are also harder to transfer to MLP-01-SCD than MLP-BCE-BP. In addition, MLP-01-SCD's adversaries cannot transfer to either MLP-BCE-BP (81.95%) or MLP-BCE-BAN (75.6%) means MLP-01-SCD's adversaries' transferability are very low. The same phenomenon happens in class pair 0 vs 2.

Tables 5.16 shows an example of adversaries' external transferability between MLP baseline models in class pair 0 vs 1 and class pair 0 vs 2 on CIFAR10 dataset. We generate adversaries through PGD attack for models list in the first column and then evaluate the accuracy of models list in the first row. For example, in class pair 0 vs 1, CNN-BCE-BAN's accuracy on MLP-BCE-BP's adversaries is 17.35%, which is only 10% higher than MLP-BCE-BP's but 30% lower than CNN-01-ABP's. This means

CNN-BCE-BP's adversaries can transfer to CNN-BCE-BAN easier than transfer to CNN-01-ABP. However, CNN-BCE-BP (40.85%) and CNN-01-ABP (60.8%) perform well in defending adversaries generated from CNN-BCE-BAN. Additionally, CNN-BCE-BP (83.6%) and CNN-BCE-BAN (80.15%) are robust in defending CNN-01-ABP's adversaries. In class pair 0 vs 2, the overall robustness for all models declined, and CNN-01-ABP does not perform significantly better than the other two models on defending adversaries.

### 5.3.4 External Adversarial Transferability Results

Tables 5.17, 5.18, 5.19, and 5.20 show adversaries transferability between MLP baseline models in each class pair of CIFAR10 and STL10. For defending adversaries generated from either MLP-BCE-BP or MLP-BCE-BAN, MLP-01-SCD is significantly better than another method with p-values below 0.05 ($< 0.05$) under Equal Variance T-Test. MLP-BCE-BP and MLP-BCE-BAN are robust on defending MLP-01-SCD's adversaries.

Tables 5.21 and 5.22 show adversaries transferability between CNN baseline models in each class pair of CIFAR10. CNN-01-ABP is more robust than CNN-BCE-BAN when defending CNN-BCE-BP's adversaries with p-value of 0.00001 ($< 0.05$) under Equal Variance T-Test. However, CNN-01-ABP is not significantly more robust than CNN-BCE-BP when defending CNN-BCE-BAN's adversaries with p-value of 0.06 ($> 0.05$) under Equal Variance T-Test. Both CNN-BCE-BP and CNN-BCE-BAN are robust in defending CNN-01-ABP's adversaries.

**Table 5.17** Adversaries' Transferability Between MLP-BCE-BP, MLP-BCE-BAN, and MLP-01-SCD on CIFAR10 Dataset Generated by PGD Attack (A)

| **Source** | mlpbcebp | | mlpbceban | | mlp01scd | |
|---|---|---|---|---|---|---|
| **target** | mlpbceban | **mlp01scd** | mlpbcebp | **mlp01scd** | mlpbcebp | mlpbceban |
| **0 vs 1** | 0.45 | **66.45** | 1.45 | **69.95** | 81.95 | 75.6 |
| **0 vs 2** | 0 | **66.2** | 0.15 | **66.45** | 80.35 | 79.85 |
| **0 vs 3** | 3 | **73.45** | 1.25 | **72.7** | 87.4 | 84.35 |
| **0 vs 4** | 0.55 | **75.5** | 0.35 | **74.9** | 83.95 | 83.1 |
| **0 vs 5** | 1.2 | **73.85** | 0.75 | **74.5** | 89.6 | 87.1 |
| **0 vs 6** | 1.85 | **73.15** | 1.7 | **72.35** | 88.4 | 85.85 |
| **0 vs 7** | 0.15 | **71.9** | 0.5 | **73.8** | 82.75 | 80.2 |
| **0 vs 8** | 0 | **46.7** | 0.15 | **52.65** | 68.25 | 68.4 |
| **0 vs 9** | 0.45 | **66.25** | 2.55 | **69.1** | 83.4 | 78.85 |
| **1 vs 2** | 0.55 | **70.55** | 0.9 | **71.9** | 87.45 | 83.25 |
| **1 vs 3** | 0.35 | **54.1** | 2.15 | **60.55** | 84.25 | 78.95 |
| **1 vs 4** | 0.35 | **70.1** | 1.75 | **71.8** | 83.65 | 75.1 |
| **1 vs 5** | 1.4 | **70.55** | 1.2 | **66.35** | 86.45 | 80.8 |
| **1 vs 6** | 1 | **65.15** | 1.05 | **66.6** | 87.9 | 82.85 |
| **1 vs 7** | 0.25 | **63.05** | 1.35 | **66.75** | 87.6 | 82.45 |
| **1 vs 8** | 0.65 | **54.45** | 2.95 | **61** | 76.6 | 71.7 |
| **1 vs 9** | 0 | **44.1** | 0.05 | **46.45** | 71.05 | 67.15 |
| **2 vs 3** | 1.35 | **44.8** | 0.7 | **42.45** | 66.85 | 61.55 |
| **2 vs 4** | 0 | **35.2** | 0 | **38.65** | 63.75 | 61.95 |
| **2 vs 5** | 0.2 | **44.6** | 0.25 | **51.95** | 71 | 66.8 |
| **2 vs 6** | 0.2 | **45.95** | 0.35 | **51.35** | 73.2 | 70.35 |
| **2 vs 7** | 0 | **45.65** | 0.05 | **48.85** | 70.75 | 65.35 |
| **2 vs 8** | 0.55 | **71.5** | 0.95 | **72.2** | 88.55 | 86.05 |

**Table 5.18** Adversaries' Transferability Between MLP-BCE-BP, MLP-BCE-BAN, and MLP-01-SCD on CIFAR10 Dataset Generated by PGD Attack (B)

| **Source** | mlpbcebp | | mlpbceban | | mlp01scd | |
|---|---|---|---|---|---|---|
| **target** | mlpbceban | **mlp01scd** | mlpbcebp | **mlp01scd** | mlpbcebp | mlpbceban |
| **2 vs 9** | 0.35 | **68.7** | 0.65 | **70.55** | 84.5 | 80.4 |
| **3 vs 4** | 0.1 | **47.6** | 0.2 | **48.6** | 66.7 | 63.5 |
| **3 vs 5** | 0.15 | **34.65** | 0.05 | **35.1** | 57.9 | 56.95 |
| **3 vs 6** | 0.1 | **36.25** | 0.65 | **39.7** | 67.2 | 62.5 |
| **3 vs 7** | 0.05 | **38.9** | 0.1 | **41.7** | 70.3 | 66.55 |
| **3 vs 8** | 1.1 | **73.95** | 3.1 | **76.05** | 88.2 | 84.5 |
| **3 vs 9** | 0.65 | **68** | 0.85 | **65.05** | 79.55 | 72.9 |
| **4 vs 5** | 0.65 | **48.4** | 0.55 | **52.65** | 68.8 | 62.2 |
| **4 vs 6** | 0.1 | **38.75** | 0.05 | **38.95** | 66.5 | 62.6 |
| **4 vs 7** | 0.05 | **45.85** | 0 | **44.75** | 60.25 | 54.7 |
| **4 vs 8** | 1.45 | **74.55** | 2.25 | **74.3** | 87.5 | 85.7 |
| **4 vs 9** | 0.3 | **69.15** | 0.75 | **69.45** | 85.9 | 80.55 |
| **5 vs 6** | 0.05 | **49.2** | 0.25 | **50.55** | 71.15 | 66.95 |
| **5 vs 7** | 0 | **44.5** | 0.05 | **43.35** | 73.65 | 70.2 |
| **5 vs 8** | 2.45 | **78.85** | 0.9 | **76.8** | 89.25 | 86.15 |
| **5 vs 9** | 1.05 | **71.4** | 1.3 | **71.4** | 88.65 | 85.35 |
| **6 vs 7** | 0.35 | **46.75** | 0.3 | **48.15** | 73.75 | 67.75 |
| **6 vs 8** | 2.85 | **76.9** | 1.4 | **81.2** | 90.9 | 87.75 |
| **6 vs 9** | 1 | **74.4** | 0.4 | **69.6** | 86.45 | 83.1 |
| **7 vs 8** | 0.45 | **69.55** | 1.35 | **71.4** | 87.5 | 83.3 |
| **7 vs 9** | 0.65 | **58.95** | 0.6 | **57.55** | 85.45 | 82.2 |
| **8 vs 9** | 0.5 | **61.65** | 0.2 | **61.8** | 77.3 | 73.6 |

**Table 5.19** Adversaries' Transferability Between MLP-BCE-BP, MLP-BCE-BAN, and MLP-01-SCD on STL10 Dataset Generated by PGD Attack (A)

| **Source** | mlpbcebp | | mlpbceban | | mlp01scd | |
|---|---|---|---|---|---|---|
| **target** | mlpbceban | **mlp01scd** | mlpbcebp | **mlp01scd** | mlpbcebp | mlpbceban |
| **0 vs 1** | 4.94 | **54.81** | 4.19 | **50.69** | 80.94 | 81.69 |
| **0 vs 2** | 5.87 | **76.5** | 9.56 | **76.88** | 84.94 | 82.94 |
| **0 vs 3** | 16.19 | **59** | 16.12 | **60** | 84.25 | 83.94 |
| **0 vs 4** | 17.56 | **61.62** | 27.25 | **63.62** | 84.62 | 84.5 |
| **0 vs 5** | 23.25 | **61.81** | 23.75 | **58** | 82.94 | 82.81 |
| **0 vs 6** | 16.81 | **65.19** | 10.94 | **62.31** | 85.12 | 84.31 |
| **0 vs 7** | 32.31 | **67.87** | 22.19 | **68.87** | 86.31 | 86 |
| **0 vs 8** | 5.5 | **56** | 9.75 | **54.06** | 77.56 | 76.5 |
| **0 vs 9** | 7.12 | **69.37** | 9.25 | **68.12** | 79.06 | 78.31 |
| **1 vs 2** | 18.69 | **69.06** | 12.62 | **70.06** | 87.44 | 85.19 |
| **1 vs 3** | 0.31 | **25.06** | 1.56 | **23.69** | 64.19 | 64.88 |
| **1 vs 4** | 3.19 | **36.5** | 7.87 | **33.25** | 67.62 | 67.56 |
| **1 vs 5** | 7.69 | **35.62** | 6.5 | **24.69** | 62.12 | 62.12 |
| **1 vs 6** | 0.62 | **36.81** | 2.19 | **32.81** | 71.31 | 71 |
| **1 vs 7** | 4.25 | **35.81** | 4.63 | **23.12** | 61.94 | 60.69 |
| **1 vs 8** | 17.62 | **69.81** | 12.62 | **68.94** | 89.31 | 87.62 |
| **1 vs 9** | 15.75 | **67.44** | 12.12 | **68.12** | 86.87 | 84.87 |
| **2 vs 3** | 9.69 | **67.94** | 8.19 | **70.31** | 86 | 84.25 |
| **2 vs 4** | 14.25 | **74.31** | 16.12 | **78.06** | 88.31 | 86.62 |
| **2 vs 5** | 14.75 | **67.44** | 14.75 | **71.75** | 86.81 | 85.37 |
| **2 vs 6** | 20.37 | **66.75** | 16.62 | **71.44** | 87.69 | 86.37 |
| **2 vs 7** | 23.5 | **67.56** | 14.62 | **69.75** | 87.81 | 86.06 |
| **2 vs 8** | 0.88 | **67.06** | 1.31 | **66.31** | 80.5 | 78.62 |

**Table 5.20** Adversaries' Transferability Between MLP-BCE-BP, MLP-BCE-BAN, and MLP-01-SCD on STL10 Dataset Generated by PGD Attack (B)

| **Source** | mlpbcebp | | mlpbceban | | mlp01scd | |
|---|---|---|---|---|---|---|
| **target** | mlpbceban | **mlp01scd** | mlpbcebp | **mlp01scd** | mlpbcebp | mlpbceban |
| **2 vs 9** | 0.12 | **32.87** | 1.25 | **37** | 68.56 | 68.44 |
| **3 vs 4** | 2.19 | **35.56** | 4.25 | **26.37** | 67.12 | 68.06 |
| **3 vs 5** | 0.56 | **28.06** | 2.19 | **18.63** | 58.94 | 58.56 |
| **3 vs 6** | 0.25 | **27** | 1.25 | **25.56** | 68.94 | 69.31 |
| **3 vs 7** | 0.06 | **21.37** | 1.06 | **19.94** | 63.19 | 61.75 |
| **3 vs 8** | 25.5 | **69.31** | 25.87 | **71.06** | 89 | 88.81 |
| **3 vs 9** | 16.62 | **66.75** | 20.06 | **69.94** | 86 | 84 |
| **4 vs 5** | 19 | **48.06** | 9.06 | **34** | 67.62 | 65.56 |
| **4 vs 6** | 9.81 | **47.19** | 6.69 | **39.5** | 68 | 69.62 |
| **4 vs 7** | 2.56 | **32.81** | 1.37 | **22.44** | 69.19 | 69.12 |
| **4 vs 8** | 34.25 | **68.62** | 32.69 | **71.63** | 90.19 | 90.19 |
| **4 vs 9** | 10.69 | **69.75** | 16.88 | **71.88** | 87 | 85.75 |
| **5 vs 6** | 0.44 | **25.87** | 1.06 | **25.69** | 63.25 | 63.75 |
| **5 vs 7** | 0.75 | **28.81** | 1.06 | **20.81** | 61 | 62.19 |
| **5 vs 8** | 34.56 | **69.12** | 39.69 | **72.94** | 91.06 | 90.62 |
| **5 vs 9** | 15 | **66.62** | 25.19 | **69.12** | 86.44 | 85 |
| **6 vs 7** | 3.44 | **28.94** | 3.25 | **27.5** | 68.56 | 68.75 |
| **6 vs 8** | 22.94 | **76.44** | 39 | **77.56** | 90.44 | 89.69 |
| **6 vs 9** | 18.44 | **67.94** | 21.69 | **67.81** | 85.62 | 84.56 |
| **7 vs 8** | 30.94 | **75.94** | 24.44 | **74.44** | 91.12 | 90.69 |
| **7 vs 9** | 15.31 | **65.87** | 19.94 | **67.87** | 86.19 | 85.37 |
| **8 vs 9** | 1.62 | **55.06** | 0.31 | **54.44** | 72.31 | 72.37 |

**Table 5.21** Adversaries' Transferability Between CNN-BCE-BP, CNN-BCE-BAN, and CNN-01-ABP on CIFAR10 Dataset Generated by PGD Attack (A)

| **source** | cnnbcebp | | cnnbceban | | cnn01abp | |
|---|---|---|---|---|---|---|
| **target** | cnnbceban | **cnn01abp** | cnnbcebp | **cnn01abp** | cnnbcebp | cnnbceban |
| **0 vs 1** | 17.35 | **47.9** | 40.85 | **60.8** | 83.6 | 80.15 |
| **0 vs 2** | 12.8 | **15.4** | 19.8 | **22.8** | 51.55 | 51.85 |
| **0 vs 3** | 25.75 | **36.25** | 38.65 | **41.3** | 73.3 | 68.65 |
| **0 vs 4** | 5.35 | **11.4** | 10.95 | **16.95** | 52.95 | 48.4 |
| **0 vs 5** | 20.5 | **32.2** | 34.3 | **42.45** | 76.45 | 72.55 |
| **0 vs 6** | 29.3 | **31.85** | 44.55 | **43.6** | 82.8 | 80.35 |
| **0 vs 7** | 17.35 | **27.2** | 23.65 | **36.1** | 70.1 | 68.75 |
| **0 vs 8** | 5.35 | **13.7** | 14.15 | **17.6** | 52.35 | 41.9 |
| **0 vs 9** | 17.95 | **40.9** | 33.6 | **49.3** | 77.35 | 72.25 |
| **1 vs 2** | 26.1 | **35.1** | 56.25 | **60.15** | 80.85 | 76.8 |
| **1 vs 3** | 19.65 | **31.15** | 47.9 | **50.3** | 74.35 | 68.95 |
| **1 vs 4** | 14.55 | **24.8** | 34.25 | **41.8** | 71.8 | 67.1 |
| **1 vs 5** | 20 | **31.45** | 47.75 | **49.85** | 77.3 | 74.65 |
| **1 vs 6** | 13.35 | **23.85** | 37.25 | **39.7** | 77.1 | 71.4 |
| **1 vs 7** | 15.65 | **32.4** | 46.7 | **51.5** | 81.6 | 77.05 |
| **1 vs 8** | 7.1 | **20.5** | 22.55 | **33.55** | 65.35 | 56.95 |
| **1 vs 9** | 1.2 | **17.6** | 6.95 | **29.75** | 65.1 | 57.65 |
| **2 vs 3** | 2.1 | **7.85** | 4 | **10.45** | 38 | 31.05 |
| **2 vs 4** | 0.1 | **1.9** | 2.05 | **3.4** | 26.8 | 17.5 |
| **2 vs 5** | 2.7 | **9.75** | 8.05 | **15.3** | 45.8 | 39.05 |
| **2 vs 6** | 3.95 | **10.55** | 12.4 | **20.75** | 48.55 | 42.75 |
| **2 vs 7** | 3.55 | **13.7** | 15.95 | **23.05** | 55.55 | 43.9 |
| **2 vs 8** | 20.45 | **32.95** | 30.3 | **40.6** | 73.85 | 71.45 |

**Table 5.22** Adversaries' Transferability Between CNN-BCE-BP, CNN-BCE-BAN, and CNN-01-ABP on CIFAR10 Dataset Generated by PGD Attack (B)

| **source** | cnnbcebp | | cnnbceban | | cnn01abp | |
|---|---|---|---|---|---|---|
| **target** | cnnbceban | **cnn01abp** | cnnbcebp | **cnn01abp** | cnnbcebp | cnnbceban |
| **2 vs 9** | 15.75 | **39.75** | 45.35 | **57.25** | 77.2 | 69.6 |
| **3 vs 4** | 2.8 | **8.4** | 4.9 | **11.55** | 37.95 | 29.55 |
| **3 vs 5** | 1.3 | **3.8** | 2.85 | **4.6** | 32.3 | 28 |
| **3 vs 6** | 2.3 | **7.55** | 9.9 | **12.5** | 45.15 | 33.7 |
| **3 vs 7** | 0.95 | **5.75** | 5.1 | **9.45** | 34.95 | 23.8 |
| **3 vs 8** | 21.05 | **36.7** | 39.55 | **42.55** | 81.15 | 73.45 |
| **3 vs 9** | 8.6 | **23.65** | 37.7 | **44.25** | 66.5 | 57.75 |
| **4 vs 5** | 5.85 | **9.5** | 9.7 | **13.8** | 52.2 | 46.95 |
| **4 vs 6** | 0.2 | **2.95** | 3.55 | **7.75** | 34.05 | 25.45 |
| **4 vs 7** | 1.15 | **6.25** | 4.35 | **11.2** | 38.95 | 32.35 |
| **4 vs 8** | 17.5 | **28.6** | 31.4 | **37.25** | 73.05 | 64.65 |
| **4 vs 9** | 12.55 | **24.25** | 31.65 | **41.55** | 68.4 | 63.45 |
| **5 vs 6** | 2.65 | **6.95** | 7.75 | **10.7** | 44.1 | 39.95 |
| **5 vs 7** | 1.05 | **9.25** | 4.75 | **10.45** | 49.7 | 35.05 |
| **5 vs 8** | 24.8 | **41.25** | 46.4 | **48.6** | 85.05 | 79.05 |
| **5 vs 9** | 9.75 | **27.05** | 40.05 | **44.5** | 75.5 | 67.9 |
| **6 vs 7** | 2.25 | **9.85** | 15.05 | **21.3** | 55.85 | 44.05 |
| **6 vs 8** | 29.45 | **37.1** | 46.55 | **44.1** | 83.15 | 78.45 |
| **6 vs 9** | 6 | **15.75** | 25.7 | **25.5** | 72.45 | 62.6 |
| **7 vs 8** | 25.05 | **38** | 33.5 | **46.7** | 78.75 | 75.95 |
| **7 vs 9** | 8.2 | **28.3** | 23.2 | **39.05** | 70.35 | 65.4 |
| **8 vs 9** | 6.8 | **23.4** | 23.65 | **33.4** | 63.35 | 54.7 |

## 5.4 Decision Based Attack

Except for transferability attacks, there is another type of black-box attack method, Decision-based attack. Considering that the users generally can access the neural networks' predicted category only, rather than accurate probability in the real world. score-based attack such as variants of JSMA [47], Carlini & Wagner attack [10], and ZOO [12] can not be deployed to attack DNN successfully.

Brendel proposed a decision-based adversarial attack, which achieves similar performance as white-box attack[7]. But relying on the huge number of model queries prevents it from being practical in real-world applications. Jianbo [11] propose a query-efficient decision-based attack in 2019, which requires significantly fewer queries from the target model to acquire competitive performance in attack.

We deployed Decision Based Attack and HopSkipJump Attack to our baseline models and evaluated their robustness in defending this kind of black-box attack method in this section.

### 5.4.1 Decision Based Attack Results

We applied Decision-based Attack[1] implemented in IBM Adversarial Robustness 360 Toolbox package[2]. Hyperparameters setting: untargetted $L_2$ distance attack, sample_size=10000, delta=0.01, epsilon=0.01, step_adapt=0.02, num_trail=100, init_eval=100, max_iters=40, train_size=100.

In each class pair from CIFAR10, we randomly pick 100 samples that are correctly classified by all four baseline models for synthesizing adversaries by Decision-based Attack. We take the median $L_2$ distance of each pair and record the results in Tables 5.23 and 5.24. Figures 5.14 and 5.15 show summary curve of results in Tables 5.23 and 5.24.

[1]https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/attacks/evasion.htmlDecision-based-attack
[2]https://github.com/Trusted-AI/adversarial-robustness-toolbox

For MLP baseline models' results, MLP-01-SCD's median $L_2$ distance is not better than MLP-BCE-BP with p-value of 0.026 ($> 0.05$) under Equal Variance T-Test. But MLP-BCE-BAN performs significantly better than MLP-BCE-BP with p-value of 0.01 ($< 0.05$) under Equal Variance T-Test.

For CNN baseline models' results, CNN-01-ABP's median $L_2$ distance is significantly better than CNN-BCE-BP with p-value of 0 ($< 0.05$) under Equal Variance T-Test. But CNN-BCE-BAN performs similar as CNN-01-ABP with p-value of 0.93 ($> 0.05$) under Equal Variance T-Test.

The Sign activation model's overall $L_2$ distance under Decision-based attack is larger than ReLU activation models, indicating Sign activation prevents Decision-based attack from searching a minimum distortion in a few queries. We do not know if these model will achieve a similar $L_2$ distance after a large number of queries or iterations run in Decision-based attack, but with limited number of queries available, Sign activation looks more robust in defending Decision-based attack.

**Table 5.23** Median $L_2$ Distance Between Adversary and Clean Data of Decision-based Attack for Attacking MLP Baseline Models

| class | MLP-BCE-BP | MLP-BCE-BAN | MLP-01-SCD | MLP-BCE-SCD | class | MLP-BCE-BP | MLP-BCE-BAN | MLP-01-SCD | MLP-BCE-SCD |
|---|---|---|---|---|---|---|---|---|---|
| **0 vs 1** | 3.231 | 3.114 | 2.727 | 2.185 | **2 vs 9** | 3.038 | 3.958 | 2.659 | 2.024 |
| **0 vs 2** | 2.642 | 2.712 | 2.970 | 1.595 | **3 vs 4** | 2.029 | 2.824 | 1.562 | 1.620 |
| **0 vs 3** | 3.024 | 3.809 | 3.117 | 2.388 | **3 vs 5** | 1.436 | 2.042 | 0.967 | 0.833 |
| **0 vs 4** | 2.653 | 3.537 | 3.217 | 1.946 | **3 vs 6** | 2.806 | 2.508 | 1.475 | 1.798 |
| **0 vs 5** | 3.010 | 3.672 | 3.129 | 2.191 | **3 vs 7** | 2.055 | 1.940 | 1.361 | 1.416 |
| **0 vs 6** | 3.375 | 4.323 | 4.724 | 2.313 | **3 vs 8** | 4.081 | 4.904 | 3.907 | 2.464 |
| **0 vs 7** | 2.786 | 2.801 | 3.021 | 2.085 | **3 vs 9** | 2.791 | 3.265 | 2.207 | 2.105 |
| **0 vs 8** | 2.104 | 2.265 | 1.622 | 1.650 | **4 vs 5** | 2.353 | 2.060 | 1.297 | 1.648 |
| **0 vs 9** | 3.690 | 3.421 | 2.549 | 2.048 | **4 vs 6** | 1.737 | 1.656 | 1.262 | 1.272 |
| **1 vs 2** | 3.113 | 3.359 | 2.274 | 1.957 | **4 vs 7** | 2.018 | 2.076 | 1.254 | 1.543 |
| **1 vs 3** | 3.458 | 3.377 | 2.044 | 2.207 | **4 vs 8** | 3.491 | 4.679 | 3.783 | 2.064 |
| **1 vs 4** | 2.828 | 3.217 | 2.467 | 1.995 | **4 vs 9** | 2.859 | 4.256 | 3.182 | 2.215 |
| **1 vs 5** | 2.828 | 3.280 | 2.798 | 2.068 | **5 vs 6** | 2.236 | 2.699 | 1.922 | 1.864 |
| **1 vs 6** | 3.112 | 3.731 | 2.374 | 2.389 | **5 vs 7** | 1.963 | 2.194 | 1.590 | 1.405 |
| **1 vs 7** | 2.902 | 3.301 | 2.621 | 2.041 | **5 vs 8** | 3.057 | 4.447 | 3.492 | 2.239 |
| **1 vs 8** | 2.749 | 2.461 | 2.301 | 1.884 | **5 vs 9** | 3.164 | 4.235 | 2.843 | 2.162 |
| **1 vs 9** | 2.219 | 2.355 | 1.467 | 1.462 | **6 vs 7** | 2.097 | 2.037 | 1.876 | 1.836 |
| **2 vs 3** | 2.271 | 2.823 | 1.333 | 1.476 | **6 vs 8** | 3.739 | 5.412 | 4.983 | 2.664 |
| **2 vs 4** | 1.240 | 0.992 | 0.852 | 1.023 | **6 vs 9** | 2.584 | 4.169 | 2.460 | 2.146 |
| **2 vs 5** | 2.704 | 2.823 | 1.345 | 1.460 | **7 vs 8** | 3.537 | 3.868 | 3.946 | 2.401 |
| **2 vs 6** | 1.699 | 2.060 | 1.643 | 1.438 | **7 vs 9** | 2.583 | 3.353 | 2.750 | 1.869 |
| **2 vs 7** | 2.341 | 2.142 | 1.764 | 1.584 | **8 vs 9** | 2.565 | 3.199 | 2.589 | 1.977 |
| **2 vs 8** | 3.125 | 4.032 | 3.063 | 2.021 | | | | | |

**Table 5.24** Median $L_2$ Distance Between Adversary and Clean Data of Decision-based Attack for Attacking CNN Baseline Models

| class | CNN-BCE-BP | CNN-BCE-BAN | CNN-01-ABP | class | CNN-BCE-BP | CNN-BCE-BAN | CNN-01-ABP |
|---|---|---|---|---|---|---|---|
| **0 vs 1** | 6.871 | 7.454 | 7.097 | **2 vs 9** | 6.527 | 7.407 | 7.484 |
| **0 vs 2** | 5.769 | 7.478 | 7.020 | **3 vs 4** | 4.059 | 4.766 | 5.658 |
| **0 vs 3** | 6.475 | 8.497 | 8.799 | **3 vs 5** | 3.559 | 5.656 | 4.033 |
| **0 vs 4** | 5.220 | 6.251 | 6.361 | **3 vs 6** | 4.365 | 5.607 | 4.551 |
| **0 vs 5** | 6.432 | 8.298 | 8.627 | **3 vs 7** | 4.221 | 5.439 | 6.426 |
| **0 vs 6** | 7.677 | 8.011 | 8.287 | **3 vs 8** | 6.724 | 8.592 | 8.394 |
| **0 vs 7** | 6.825 | 9.567 | 8.432 | **3 vs 9** | 5.372 | 7.279 | 7.020 |
| **0 vs 8** | 5.291 | 7.717 | 8.108 | **4 vs 5** | 5.938 | 9.562 | 8.576 |
| **0 vs 9** | 5.830 | 5.883 | 5.655 | **4 vs 6** | 4.172 | 5.147 | 4.745 |
| **1 vs 2** | 6.458 | 8.138 | 7.543 | **4 vs 7** | 4.001 | 6.703 | 6.402 |
| **1 vs 3** | 5.110 | 5.770 | 6.903 | **4 vs 8** | 7.523 | 9.948 | 8.748 |
| **1 vs 4** | 6.625 | 7.445 | 6.745 | **4 vs 9** | 6.533 | 8.031 | 7.039 |
| **1 vs 5** | 6.246 | 6.986 | 6.966 | **5 vs 6** | 5.526 | 5.968 | 6.475 |
| **1 vs 6** | 7.521 | 8.286 | 8.358 | **5 vs 7** | 4.388 | 6.104 | 5.900 |
| **1 vs 7** | 7.049 | 8.167 | 8.077 | **5 vs 8** | 7.297 | 8.056 | 9.074 |
| **1 vs 8** | 5.687 | 6.491 | 6.833 | **5 vs 9** | 6.264 | 7.417 | 7.258 |
| **1 vs 9** | 4.624 | 6.485 | 6.694 | **6 vs 7** | 5.710 | 7.134 | 6.527 |
| **2 vs 3** | 3.840 | 5.302 | 5.658 | **6 vs 8** | 7.645 | 7.940 | 7.692 |
| **2 vs 4** | 3.259 | 4.459 | 8.762 | **6 vs 9** | 6.559 | 7.803 | 7.780 |
| **2 vs 5** | 4.634 | 5.323 | 5.734 | **7 vs 8** | 6.601 | 9.663 | 9.224 |
| **2 vs 6** | 5.658 | 5.750 | 5.355 | **7 vs 9** | 5.795 | 6.970 | 7.172 |
| **2 vs 7** | 5.452 | 6.805 | 6.886 | **8 vs 9** | 5.295 | 5.741 | 5.699 |
| **2 vs 8** | 6.928 | 7.839 | 7.531 | | | | |

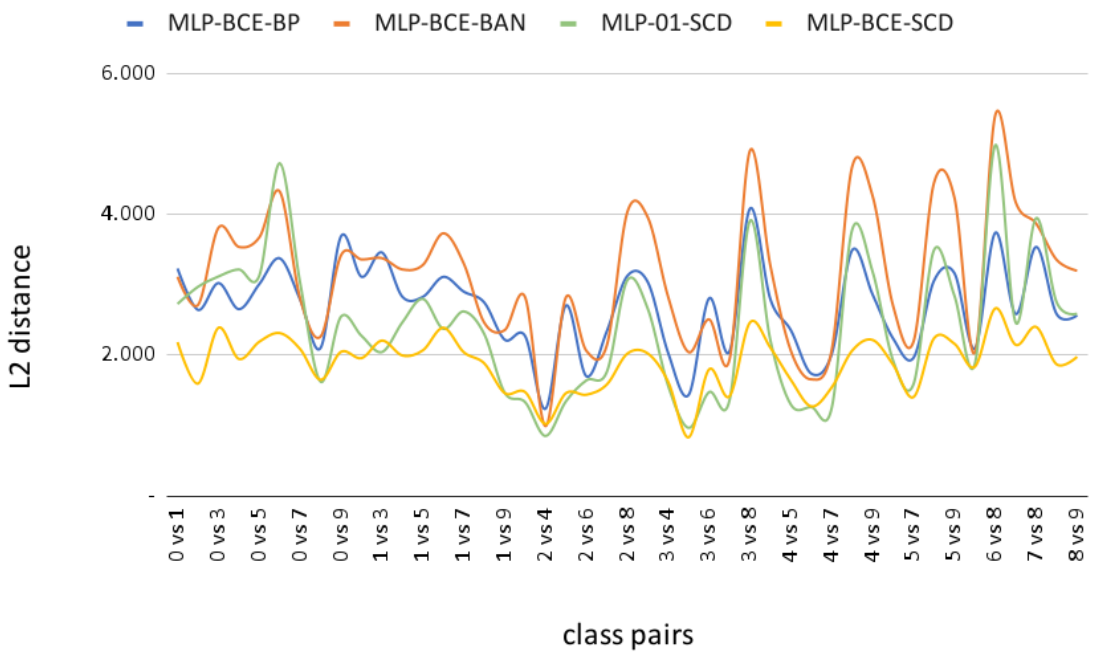


**Figure 5.14** Median $L_2$ distance between adversary and clean data of Decision-based attack for MLP baseline models.

### 5.4.2 HopSkipJump Attack Results

We applied HopSkipJump Attack[3] implemented in IBM Adversarial Robustness 360 Toolbox package[4]. Hyperparameters setting: untargetted $L_2$ distance attack, max_eval=10000, init_eval=100, max_iters=40, train_size=100.

In each class pair from CIFAR10, we randomly pick 100 samples that are correctly classified by all four baseline models for synthesizing adversaries by HopSkipJump Attack. We take the median $L_2$ distance of each pair and record the results in Tables 5.25 and 5.26. Figures 5.16 and 5.17 show summary curve of results in Tables 5.25 and 5.26.

For MLP baseline models' results, MLP-01-SCD's median $L_2$ distance is significantly better than MLP-BCE-BP with p-value of 0.026 ($< 0.05$) under Equal

[3]https://adversarial-robustness-toolbox.readthedocs.io/en/latest/modules/attacks/evasion.htmlhopskipjump-attack
[4]https://github.com/Trusted-AI/adversarial-robustness-toolbox

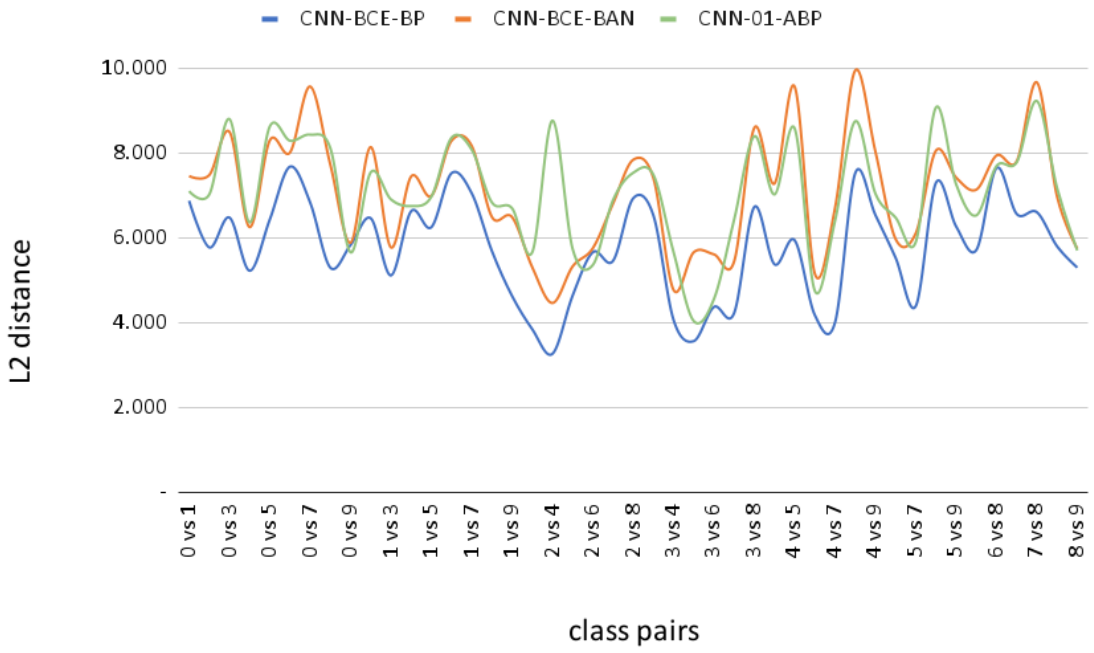


**Figure 5.15** Median $L_2$ distance between adversary and clean data of Decision-based attack for CNN baseline models.

Variance T-Test. But MLP-BCE-BAN performs similar as MLP-01-SCD with p-value of 0.34 ($> 0.05$) under Equal Variance T-Test.

For CNN baseline models' results, CNN-01-ABP's median $L_2$ distance is significantly better than CNN-BCE-BP with p-value of 0 ($< 0.05$) under Equal Variance T-Test. But CNN-BCE-BAN performs similar as CNN-01-ABP with p-value of 0.09 ($> 0.05$) under Equal Variance T-Test.

The Sign activation model's overall $L_2$ distance under HopSkipJump attack is larger than ReLU activation models, indicating Sign activation prevent HopSkipJump attack from searching a minimum distortion in a few queries. We do not know if these model will achieve a similar $L_2$ distance after a large number of queries or iterations run in HopSkipJump attack, but with limited number of queries available, Sign activation looks more robust in defending HopSkipJump attack. Another reason is

each votes' adversaries could hardly transfer to each other, results that the ensembling becomes more robust as the number of votes increased [67].

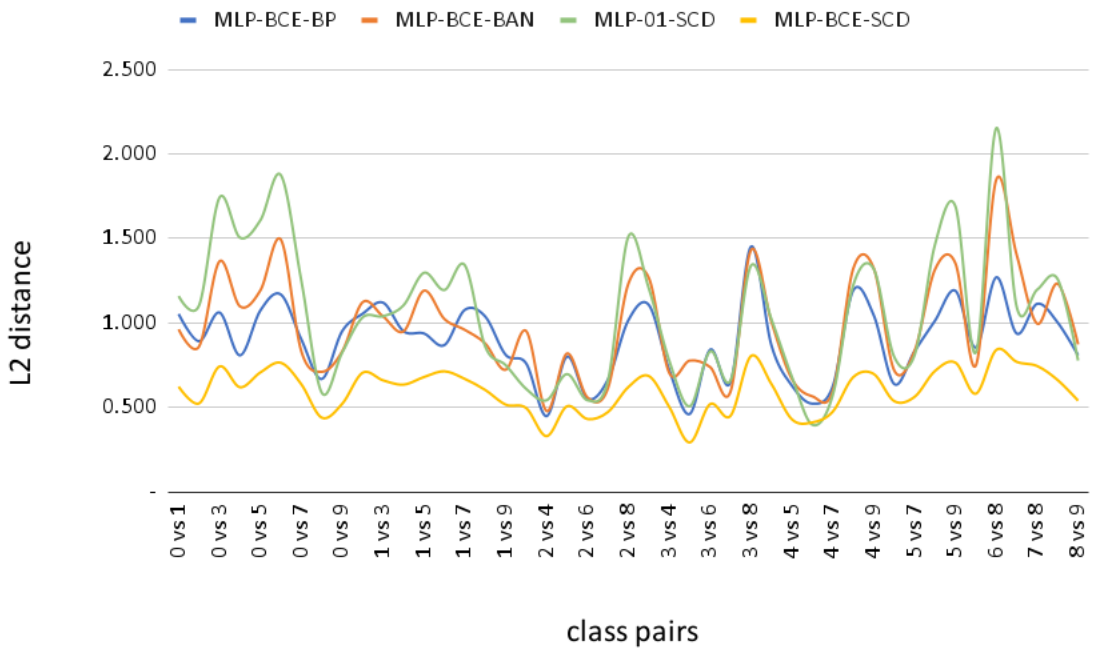


**Figure 5.16** Median $L_2$ distance between adversary and clean data of HopSkipJump attack for MLP baseline models.

**Table 5.25** Median $L_2$ Distance Between Adversary and Clean Data of HopSkipJump Attack for Attacking MLP Baseline Models

| class | MLP-BCE-BP | MLP-BCE-BAN | MLP-01-SCD | MLP-BCE-SCD | class | MLP-BCE-BP | MLP-BCE-BAN | MLP-01-SCD | MLP-BCE-SCD |
|---|---|---|---|---|---|---|---|---|---|
| **0 vs 1** | 1.054 | 0.963 | 1.159 | 0.624 | **2 vs 9** | 1.107 | 1.267 | 1.205 | 0.686 |
| **0 vs 2** | 0.892 | 0.861 | 1.104 | 0.525 | **3 vs 4** | 0.723 | 0.705 | 0.777 | 0.504 |
| **0 vs 3** | 1.062 | 1.365 | 1.742 | 0.743 | **3 vs 5** | 0.462 | 0.778 | 0.508 | 0.294 |
| **0 vs 4** | 0.809 | 1.098 | 1.505 | 0.620 | **3 vs 6** | 0.842 | 0.739 | 0.833 | 0.520 |
| **0 vs 5** | 1.075 | 1.193 | 1.608 | 0.708 | **3 vs 7** | 0.651 | 0.595 | 0.662 | 0.451 |
| **0 vs 6** | 1.168 | 1.493 | 1.874 | 0.765 | **3 vs 8** | 1.451 | 1.431 | 1.339 | 0.805 |
| **0 vs 7** | 0.903 | 0.833 | 1.251 | 0.638 | **3 vs 9** | 0.867 | 1.007 | 1.022 | 0.639 |
| **0 vs 8** | 0.669 | 0.711 | 0.588 | 0.440 | **4 vs 5** | 0.629 | 0.669 | 0.682 | 0.430 |
| **0 vs 9** | 0.952 | 0.832 | 0.825 | 0.522 | **4 vs 6** | 0.523 | 0.566 | 0.398 | 0.412 |
| **1 vs 2** | 1.055 | 1.123 | 1.033 | 0.706 | **4 vs 7** | 0.625 | 0.597 | 0.574 | 0.476 |
| **1 vs 3** | 1.120 | 1.039 | 1.040 | 0.661 | **4 vs 8** | 1.193 | 1.321 | 1.224 | 0.680 |
| **1 vs 4** | 0.951 | 0.950 | 1.104 | 0.636 | **4 vs 9** | 1.047 | 1.325 | 1.320 | 0.699 |
| **1 vs 5** | 0.938 | 1.191 | 1.297 | 0.680 | **5 vs 6** | 0.640 | 0.721 | 0.801 | 0.538 |
| **1 vs 6** | 0.869 | 1.024 | 1.195 | 0.714 | **5 vs 7** | 0.833 | 0.835 | 0.809 | 0.563 |
| **1 vs 7** | 1.078 | 0.959 | 1.342 | 0.669 | **5 vs 8** | 1.014 | 1.316 | 1.464 | 0.716 |
| **1 vs 8** | 1.038 | 0.878 | 0.860 | 0.604 | **5 vs 9** | 1.191 | 1.353 | 1.690 | 0.766 |
| **1 vs 9** | 0.812 | 0.724 | 0.751 | 0.517 | **6 vs 7** | 0.855 | 0.749 | 0.826 | 0.582 |
| **2 vs 3** | 0.761 | 0.953 | 0.610 | 0.497 | **6 vs 8** | 1.270 | 1.848 | 2.152 | 0.841 |
| **2 vs 4** | 0.449 | 0.479 | 0.543 | 0.331 | **6 vs 9** | 0.937 | 1.401 | 1.094 | 0.770 |
| **2 vs 5** | 0.801 | 0.820 | 0.698 | 0.508 | **7 vs 8** | 1.114 | 0.996 | 1.197 | 0.745 |
| **2 vs 6** | 0.553 | 0.559 | 0.543 | 0.432 | **7 vs 9** | 1.003 | 1.232 | 1.260 | 0.663 |
| **2 vs 7** | 0.665 | 0.614 | 0.659 | 0.473 | **8 vs 9** | 0.809 | 0.874 | 0.778 | 0.541 |
| **2 vs 8** | 1.016 | 1.232 | 1.511 | 0.621 | | | | | |

**Table 5.26** Median $L_2$ Distance Between Adversary and Clean Data of HopSkipJump Attack for Attacking CNN Baseline Models

| class | CNN-BCE-BP | CNN-BCE-BAN | CNN-01-ABP | class | CNN-BCE-BP | CNN-BCE-BAN | CNN-01-ABP |
|---|---|---|---|---|---|---|---|
| **0 vs 1** | 1.652 | 5.919 | 5.167 | **2 vs 9** | 1.470 | 5.972 | 5.053 |
| **0 vs 2** | 1.109 | 5.802 | 5.163 | **3 vs 4** | 0.740 | 3.761 | 4.044 |
| **0 vs 3** | 1.715 | 7.043 | 5.785 | **3 vs 5** | 0.583 | 3.607 | 2.576 |
| **0 vs 4** | 1.176 | 5.015 | 4.464 | **3 vs 6** | 0.951 | 3.486 | 3.224 |
| **0 vs 5** | 1.570 | 6.668 | 6.055 | **3 vs 7** | 0.824 | 4.423 | 4.682 |
| **0 vs 6** | 1.918 | 5.655 | 5.110 | **3 vs 8** | 1.806 | 6.630 | 5.876 |
| **0 vs 7** | 1.628 | 7.621 | 6.627 | **3 vs 9** | 1.406 | 4.986 | 4.882 |
| **0 vs 8** | 1.038 | 5.627 | 5.163 | **4 vs 5** | 1.068 | 4.588 | 4.357 |
| **0 vs 9** | 1.294 | 4.607 | 3.813 | **4 vs 6** | 0.781 | 3.451 | 3.367 |
| **1 vs 2** | 1.641 | 5.940 | 5.048 | **4 vs 7** | 0.700 | 3.845 | 3.902 |
| **1 vs 3** | 1.387 | 4.872 | 5.057 | **4 vs 8** | 1.546 | 6.745 | 6.611 |
| **1 vs 4** | 1.370 | 6.207 | 5.503 | **4 vs 9** | 1.365 | 6.329 | 5.588 |
| **1 vs 5** | 1.732 | 5.476 | 5.197 | **5 vs 6** | 0.979 | 4.039 | 3.222 |
| **1 vs 6** | 1.810 | 6.761 | 6.396 | **5 vs 7** | 0.860 | 4.756 | 4.590 |
| **1 vs 7** | 1.685 | 6.385 | 6.574 | **5 vs 8** | 1.918 | 6.437 | 5.824 |
| **1 vs 8** | 1.207 | 5.351 | 4.907 | **5 vs 9** | 1.679 | 5.410 | 4.749 |
| **1 vs 9** | 0.907 | 4.123 | 3.970 | **6 vs 7** | 1.234 | 4.307 | 4.049 |
| **2 vs 3** | 0.788 | 3.600 | 2.879 | **6 vs 8** | 2.008 | 5.523 | 5.350 |
| **2 vs 4** | 0.583 | 3.544 | 3.211 | **6 vs 9** | 1.506 | 5.225 | 5.050 |
| **2 vs 5** | 0.792 | 3.817 | 3.764 | **7 vs 8** | 1.621 | 6.466 | 6.460 |
| **2 vs 6** | 0.881 | 3.388 | 3.009 | **7 vs 9** | 1.159 | 5.246 | 5.277 |
| **2 vs 7** | 1.059 | 4.346 | 4.673 | **8 vs 9** | 1.192 | 4.180 | 3.768 |
| **2 vs 8** | 1.606 | 6.397 | 5.580 | | | | |

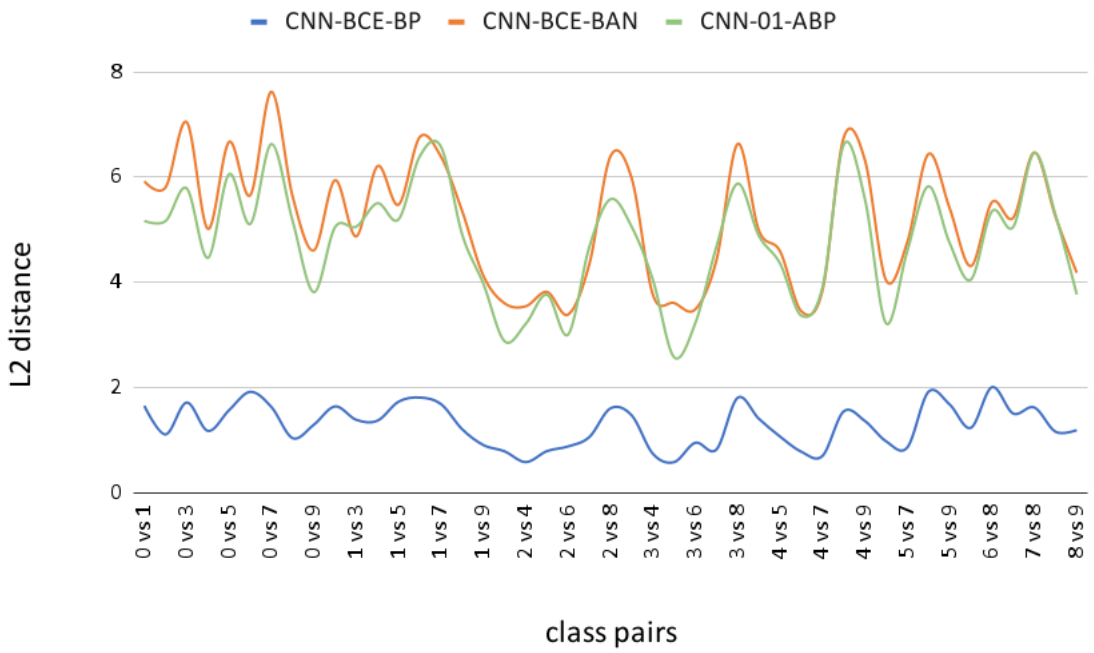


**Figure 5.17** Median $L_2$ distance between adversary and clean data of HopSkipJump attack for CNN baseline models.

## 5.5 Conclusion

This chapter conducted two gradient-based attack methods, FGSM and PGD, for white-box attacking our MLP and CNN baseline models. After that, we evaluated the inner and external adversarial transferability between each model.

Based on the observation in white-box attack section, we could say that Stochatic Coordinate Descent optimization helps model to be more robust in defending FGSM attacks. Because gradient-free method makes models' gradient be hardly estimated for one-step gradient-based attack. After doing PGD attack, a stronger multiple steps gradient estimating attack, the advantage of SCD disappears on MLP-BCE-SCD. However, MLP-01-SCD is still performing well, which means zero-one loss helps defend this kind of white-box attack. The robustness is not because of Sign activation only, since MLP-BCE-BAN is hard to defend either FGSM and PGD, even though its activation is Sign. And the adversary attack results of CNN is consistent with MLP's. It reveals that zero-one loss works on improving the robustness of either a shallow model like MLP or a deeper model like CNN. Even though backpropagation is included in ABP training strategy, backpropagation does not lower zero-one loss models' robustness too much in defending white-box attacks.

In adversarial transferability section, we noticed that MLP-01-SCD and CNN-01-ABP are outstandingly robust in defending inner adversaries, which synthesized from the network with the same structure and loss function. Benefits from non-unique solutions for solving zero-one loss problem, zero-one loss models' hyperplane might be different in each single run. However, cross-entropy loss optimized by SGD would results in similar hyperplanes in each run, the main reason that cause adversaries transfer easily between each other. Nevertheless, zero-one loss does not enable CNN models robust in external adversarial transferability, even though MLP-01-SCD performs better than other structures in MLP experiment group.

Decision-based attack and HopSkipJump attacks' results reveal that Sign activation in a neural network might decline these attacks' power. Nevertheless, zero-one loss does not affect the robustness clearly.

# CHAPTER 6

# CONCLUSION AND FUTURE WORK

## 6.1 Conclusion

This dissertation designed a Sign activation neural network framework, which used zero-one loss as loss function. This dissertation implemented a novel method, Stochastic Coordinate Descent (SCD), to optimize zero-one loss function for MLP structure series. Compared to Stochastic Gradient Descent (SGD), SCD can achieve similar performance in optimizing the same type of neural network on the CIFAR10 and STL10 datasets. In order to train Convolutional Neural Network (CNN) effectively and efficiently, a variant of SCD that ignoring the bias is developed associated with corresponding training strategy such as multi-phase training and Additional Backpropagation Penetration (ABP). These modifications on SCD significantly speed up the convergence of zero-one loss in training a deep neural network. Lastly, several commonly used adversarial attack methods are applied to measure the adversarial robustness differences between zero-one loss model and cross-entropy model, covering white-box attack, decision-based attack, and adversaries' transferability.

## 6.2 Future Work

Even though the SCD algorithm works well on both MLP and CNN structure, it is necessary to develop a more efficient optimization method for training deeper zero one loss Sign activated neural network. SCD's feature that updating one coordinate in each iteration can hardly bring optimal weights, which are very different from what they look like initially. Reducing the number of training iterations and making weights update more effective in each step are keys to the success of training a deeper neural network. Currently, only binary classification results are showed in this dissertation. Because of that zero-one loss being directly applied in multi-class problems will result

in massive local minima during training. Looking forward to finding a better multi-class version of zero-one loss is an excellent way to solve multi-class classification problems. This dissertation does not include adversarial training, which is a popular method to improve deep neural networks' adversarial robustness. Training on clean data and adversaries synthesized by gradient-based attacks like FGSM and PGD makes the model more robust to their adversaries. However, this process does not work in the SCD algorithm. So developing a new adversarial training method is worthy to enable zero one loss models to become more robust.

## REFERENCES


[1] Ethem Alpaydin. *Machine Learning*. Cambridge, MA: MIT Press, 2004.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, pages 274–283. PMLR, 2018.

[3] Peter L Bartlett, Michael I Jordan, and Jon D McAuliffe. Large margin classifiers: Convex loss, low noise, and convergence rates. In *Neural Information Processing Systems*, pages 1173–1180. Citeseer, 2003.

[4] Shai Ben-David, Nadav Eiron, and Philip M Long. On the difficulty of approximately maximizing agreements. *Journal of Computer and System Sciences*, 66(3):496–514, 2003.

[5] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *29th International Conference on Machine Learning*, pages 1467–1474, 2012.

[6] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[7] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

[8] Jameson Cahill, Peter G Casazza, Jesse Peterson, and Lindsey Woodland. Phase retrieval by projections. *arXiv preprint arXiv:1305.6226*, 2013.

[9] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14, 2017.

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

[11] Jianbo Chen, Michael I Jordan, and Martin J Wainwright. Hopskipjumpattack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1277–1294, 2020.

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

[13] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *14th International Conference on Artificial Intelligence and Statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[14] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[16] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *2018 IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[17] Rob A Dunne and Norm A Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer, 1997.

[18] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. Hotflip: White-box adversarial examples for text classification. In *56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 31–36, 2018.

[19] Joan Bruna Estrach, Arthur Szlam, and Yann LeCun. Signal recovery from pooling representations. In *International Conference on Machine Learning*, pages 307–315. PMLR, 2014.

[20] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *2018 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1625–1634, 2018.

[21] Yoav Freund and Robert E Schapire. Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296, 1999.

[22] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010.

[23] Angus Galloway, Graham W Taylor, and Medhat Moussa. Attacking binarized neural networks. *arXiv preprint arXiv:1711.00449*, 2017.

[24] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[25] Aritra Ghosh, Naresh Manwani, and PS Sastry. Making risk minimization tolerant to label noise. *Neurocomputing*, 160:93–107, 2015.

[26] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *14th International Conference on Artificial Intelligence and Statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. Cambridge, MA: MIT Press, 2016.

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

[29] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[31] Aapo Hyvärinen and Urs Köster. Complex cell pooling and the statistics of natural images. *Network: Computation in Neural Systems*, 18(2):81–100, 2007.

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

[33] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is bert really robust? a strong baseline for natural language attack on text classification and entailment. In *34th AAAI Conference on Artificial Intelligence*, volume 34, pages 8018–8025, 2020.

[34] Ron Kohavi, David H Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *International Conference on Machine Learning*, volume 96, pages 275–83, 1996.

[35] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

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

[37] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

[38] Yann LeCun, Koray Kavukcuoglu, and Clément Farabet. Convolutional networks and applications in vision. In *2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, 2010.

[39] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[40] Yueming Lyu and Ivor W Tsang. Curriculum loss: Robust learning and generalization against label corruption. In *International Conference on Learning Representations*, 2019.

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

[42] Naresh Manwani and PS Sastry. Noise tolerance under risk minimization. *IEEE Transactions on Cybernetics*, 43(3):1146–1151, 2013.

[43] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[44] Tan Nguyen and Scott Sanner. Algorithms for direct 0–1 loss optimization in binary classification. In *International Conference on Machine Learning*, pages 1085–1093. PMLR, 2013.

[45] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning*, pages 4970–4979. PMLR, 2019.

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

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

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

[49] Sebastian Raschka and Vahid Mirjalili. Python machine learning: Machine learning and deep learning with python. *Scikit-Learn, and TensorFlow. Second edition ed*, 2017.

[50] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[51] Shai Shalev-Shwartz, Ohad Shamir, and Karthik Sridharan. Learning linear and kernel predictors with the 0-1 loss function. In *22th International Joint Conference on Artificial Intelligence*, 2011.

[52] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *2016 ACM Sigsac Conference on Computer and Communications Security*, pages 1528–1540, 2016.

[53] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019.

[54] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147. PMLR, 2013.

[55] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *31th AAAI Conference on Artificial Intelligence*, volume 31, 2017.

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

[57] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[58] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453*, 2017.

[59] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of Optimization Theory and Applications*, 109(3):475–494, 2001.

[60] Xingxing Wei, Siyuan Liang, Ning Chen, and Xiaochun Cao. Transferable adversarial attacks for image and video object detection. *arXiv preprint arXiv:1811.12641*, 2018.

[61] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[62] Dongxian Wu, Yisen Wang, Shu-Tao Xia, James Bailey, and Xingjun Ma. Skip connections matter: On the transferability of adversarial examples generated with resnets. In *International Conference on Learning Representations*, 2019.

[63] Cihang Xie, Zhishuai Zhang, Yuyin Zhou, Song Bai, Jianyu Wang, Zhou Ren, and Alan L Yuille. Improving transferability of adversarial examples with input diversity. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2730–2739, 2019.

[64] Meiyan Xie, Yunzhe Xue, and Usman Roshan. Stochastic coordinate descent for 01 loss and its sensitivity to adversarial attacks. In *18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 299–304, 2019.

[65] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178, 2020.

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

[67] Yunzhe Xue, Meiyan Xie, and Usman Roshan. On the transferability of adversarial examples between convex and 01 loss models. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1460–1467, 2020.

[68] Yunzhe Xue, Meiyan Xie, and Usman Roshan. Towards adversarial robustness with 01 loss neural networks. In *19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1304–1309, 2020.

[69] Martin Zinkevich, Markus Weimer, Alexander J Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Neural Information Processing Systems*, volume 4, page 4. Citeseer, 2010.

[70] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2847–2856, 2018.