On Mixup Training of Neural Networks

Zixuan Liu

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for
Master of Applied Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Examining Committee

Supervisor:	Yongyi Mao Professor, School of Electrical Engineering & Computer Science University of Ottawa
Internal Member:	Diana Inkpen Professor, School of Electrical Engineering & Computer Science University of Ottawa
Carleton Member:	Yuhong Guo Professor, School of Computer Science Carleton Univeristy

Declaration of Authorship

I hereby certify that this thesis is entirely my own original work except where otherwise indicated. I am aware of the University of Ottawa regulations concerning plagiarism, including those regarding consequent disciplinary actions. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

The content about theoretical explanations in Chapter 5 is accomplished by my collaborator Ziqiao Wang.

Abstract

Deep neural networks are powerful tools of machine learning. Despite their capabilities of fitting the training data, they tend to perform undesirably on the unseen data. To improve the generalization of the deep neural networks, a variety of regularization techniques have been proposed. This thesis studies a simple yet effective regularization scheme, Mixup [44], which has been proposed recently. Briefly speaking, Mixup creates synthetic examples by linearly interpolating random pairs of the real examples and uses the synthetic examples for training. Although Mixup has been empirically shown to be effective on various classification tasks for neural network models, its working mechanism and possible limitations have not been well understood.

One potential problem of Mixup is known as manifold intrusion [16], in which the synthetic examples "intrude" the data manifolds of the real data, resulting in the conflicts between the synthetic labels and the ground-truth labels of the synthetic examples. The first part of this thesis investigates the strategies for resolving the manifold intrusion problem. We focus on two strategies. The first strategy, which we call "relabelling", attempts to find better labels for the synthetic data; the second strategy, which we call "cautious mixing", carefully selects the interpolating parameters to generate the synthetic examples. Through extensive experiments over several design choices, we observe that the "cautious mixing" strategy appears to perform better.

The second part of this thesis reports a previously unobserved phenomenon in Mixup training: on a number of standard datasets, the performance of the Mixup-trained models starts to decay after training for a large number of epochs, giving rise to a U-shaped generalization curve. This behavior is further aggravated when the size of the original dataset is reduced. To help understand such a behavior of Mixup, we show theoretically that Mixup training may introduce undesired data-dependent label noises to the synthetic data. Via analyzing a least-square regression problem with a random feature model, we explain why noisy labels may cause the U-shaped curve to occur: Mixup improves generalization through fitting the clean patterns at the early training stage, but as training progresses, the model becomes over-fitting to the noise in the synthetic data. Extensive experiments are performed on a variety of benchmark datasets, validating this explanation.

Acknowledgements

I would like to express my appreciation to my supervisor, Professor Yongyi Mao. During my graduate study, he patiently and professionally provided me with the academic guidance. Moreover, he also taught me how to have a right scientific research attitude and habits. He taught me not only how to be a good scholar, but also how to be a good person.

I would like to thank my collaborator Ziqiao Wang. During the research work of this thesis, he provided sufficient support on the theoretical content. During the pandemic, he also helped me to get familiar with the Ottawa city and University of Ottawa.

I would like to thank all my peers and colleagues in Professor Mao's group. I have gained a lot of happiness and ideological exchanges through getting along with them over the past year and a half.

I would like to specially thank my parents. They have always supported me on my graduate study and my overseas living, both financially and emotionally. Their support helped me to go through a lot of hardships.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mixup [44], a simple interpolation-based regularization technique, has empirically shown its effectiveness in improving the generalization and robustness of the deep classification models [16, 38, 44, 47]. Unlike the vanilla empirical risk minimization (ERM) in which networks are trained using the original training sets, Mixup trains the networks with synthetic examples. These examples are created by linearly interpolating both the input features and the one-hot labels of random instance pairs. More specifically, if we randomly draw two examples $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}', \mathbf{y}')$ from a training set, then we can formulate a synthetic example in the following way:

$$\tilde{\mathbf{x}} = \lambda \mathbf{x} + (1 - \lambda)\mathbf{x}', \qquad \tilde{\mathbf{y}} = \lambda \mathbf{y} + (1 - \lambda)\mathbf{y}' \tag{1.1}$$

where $\lambda \in [0, 1]$ and is called the interpolation coeffecient.

Owning to Mixup's simplicity and effectiveness in boosting the testing accuracy and the robustness of the deep classification models, there has been a recent surge of interest attempting to better understand Mixup's working mechanism, training characteristics, regularization potential, and possible limitations. For example, [38] empirically shows that Mixup also helps improve the calibration of the trained networks. [16] identifies the manifold intrusion issue in Mixup, where the synthetic data "intrudes" the data manifolds of the real data. [46] theoretically explains the effectiveness of Mixup via analyzing an upper bound of the loss function used in Mixup training. [47] suggests that the calibration effect of Mixup is correlated with the capacity of the networks. In this thesis, we carry out the explorations along these research lines.

In the first part of this thesis, we investigate the strategies of solving manifold intrusion [16]. As mentioned above, manifold intrusion is one of the potential problems of Mixup. It

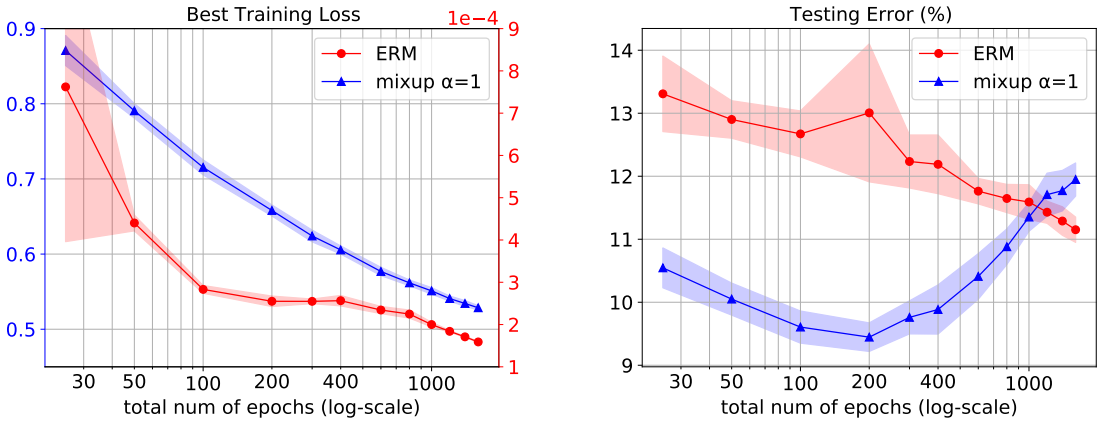
refers to a phenomenon in Mixup training where the synthetic examples "intrude" the data manifolds of the real data, resulting in the conflicts between the synthetic labels and the ground-truth labels of the synthetic data. The raising of this issue has inspired a variety of research works aiming to solve it. We first categorise the majority of the proposed arts of solving manifold intrusion into two strategies: the "relabelling" strategy and the "cautious mixing" strategy. The former one refers to a series of approaches that attempt to assign the correct training targets for the synthetic examples, whereas the latter one refers to the approaches that control the generation of the synthetic examples so that manifold intrusion can be prevented at the very first place.

We then propose two new relabelling algorithms of solving manifold intrusion: **pseudo relabelling** and **reference relabelling**. Pseudo relabelling is inspired by the pseudo labelling algorithm in semi-supervised learning. By adopting pseudo relabelling, each synthetic example is relabelled with the weighted average of its synthetic label and its pseudo label. Reference relabelling on the other hand, draws some extra examples from the training set as the reference to help relabel each synthetic example. It hypothesizes an instance-wise Gaussian generative model with respect to each of the real examples, and all such generative models together produce a Gaussian mixture model with respect to the reference examples and the two interpolating examples. The reference relabelling algorithm relabels the synthetic example via considering it as a generated data point from the Gaussian mixture model mentioned above. We then conduct some experiments of our proposed relabelling algorithms on CIFAR10 and a toy dataset named "Spiral3-90". By comparing their testing results with the Mixup baselines, we show that the two algorithms both fail to properly solve manifold intrusion as expected.

We then investigate the effectiveness of the cautious mixing strategy through extensive experiments. We empirically show that on some datasets, we can prevent the negative impact of manifold intrusion by cautiously tuning the mixing hyper-parameters. We illustrate this behavior both from the testing accuracies of the trained models and the decision boundaries fitted by them. At the end of this part, we conclude that in general the cautious mixing strategy is more effective and efficient in preventing manifold intrusion than the relabelling strategy, and we put this conclusion forward as a suggestion for the future works that aim to solve this issue.

In the second part of this thesis, we further investigate the generalization properties of Mixup training. We first report a previously unobserved phenomenon in Mixup training. Through extensive experiments on various benchmark datasets and network architectures, we observe that over-training the networks with Mixup may result in significant degradation of the networks' generalization performance. In particular, after certain number of epochs, the longer we train a network with Mixup, the worse its testing accuracy becomes, although

the training loss continues to decrease. As a result, along with the training epochs, the generalization performance of the network measured by its testing error may exhibit a U-shaped curve. Figure 1.1 shows such a curve obtained from over-training ResNet18 [18] with Mixup on CIFAR10 [23]. As can be seen from Figure 1.1, when training with Mixup for a long time, both ERM and Mixup keep decreasing their training loss, but the testing accuracy of the Mixup-trained ResNet18 gradually reduces, while that of the ERM-trained ResNet18 keeps decreasing.



(a) Train loss (100%) (b) Test acc (100%)

Figure 1.1: Over-training ResNet18 on CIFAR10. We observe that over-training reduces both the ERM and the Mixup training losses, but the testing accuracy of the Mixup-trained ResNet18 gradually decreases, while that of the ERM-trained ResNet18 show no significant changes.

Motivated by this observation, we conduct a theoretical analysis, aiming to better understand the aforementioned behavior of Mixup training. We first show theoretically that Mixup training may introduce undesired data-dependent label noises to the synthetic data. Then by analyzing the gradient-descent dynamics of training a random feature model for a least-square regression problem, we explain why noisy labels may cause the U-shaped curve to occur: under label noise, the early phase of training is primarily driven by the clean data pattern, which moves the model parameter closer to the correct solution. But as training progresses, the effect of label noise accumulates through iterations and gradually over-weighs that of the clean pattern and dominates the training process. In this phase, the model parameter gradually moves away from the correct solution until it is sufficient apart and approaches a location depending on the noise realization.

Finally, we conduct experiments on a teacher-student toy example to explicitly validate the above theoretical explanation for classification tasks.

We summarize our contributions in this thesis as follows:

- We empirically show that the cautious mixing strategy is more effective and efficient than the relabelling strategy in solving manifold intrusion.
- We report a previously unobserved phenomenon in Mixup training: over-training with Mixup may give rise to a U-shaped generalization curve.
- We show theoretically that Mixup may introduce undesired data-dependent label noises to the synthetic data.
- We empirically show and theoretically prove that over-training the deep neural networks with Mixup may hurt the networks' generalization.

The outline of this thesis is as follows: in Chapter 2, we present the preliminaries about machine learning and the related mathematical background. In Chapter 3, we provide the introduction of the Mixup training scheme. In Chapter 4, we discuss our investigation on the two main strategies of solving manifold intrusion in details. In Chapter 5, we present our observation and investigation of the phenomenon that over-training with Mixup may hurt the models' generalization performance.

Chapter 2

Preliminaries

2.1 Artificial Intelligence

Primarily, the idea of **artificial intelligence** (AI) was concerned with endowing intelligence to artificial beings. In the 1950s, with the advents of electronic digital computers, scientists started to consider the possibility of constructing the complex machines that have the equivalent essence of intelligence as human beings using computer algorithms.

Based on the capabilities of the AI technologies, the field can be mainly categorised into two types: general AI and narrow AI. General AI refers to a type of the AI technologies that enable the machines to well replicate or even overwhelm human's perception and intelligence, whereas narrow AI indicates the technologies that can simulate humans' patterns of thinking when solving specific tasks such as image classification, speech recognition, *etc.* Currently, the majority of the research works in this field are about narrow AI.

2.2 Machine Learning

Machine learning is a cross-disciplinary subject involving the fields of probability theory, statistics, optimization, *etc.* It focuses on the techniques to perform narrow AI in reality. The fundamental idea of machine learning is to develop programs that enable the machines to learn the knowledge from the data. With the help of the computer algorithms, machine

learning has become one of the most promising subfields of AI. The machine learning problems can be mainly categorised into 4 classes as shown in the following sections.

2.2.1 Unsupervised Learning

Given an unlabelled dataset $S := \{\mathbf{x}^{(i)}\}_{i=1}^{n}$, we may presume that all the examples in it are drawn from an unknown distribution P_X, where "X" denotes the random variable corresponding to the input features. The goal of unsupervised learning is developing a program that learns P_X from S so that it can generate a new set of examples that is supported by the same statistical structure of X.

2.2.2 Supervised Learning

In supervised learning, the examples are labelled. Let's denote a labelled dataset by $S := \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, where X and Y denote the random variables with respect to the input features and the labels of the examples. The objective of supervised learning is to develop a program that learns the posterior distribution of Y given X: $P_{Y|X}$.

2.2.3 Semi-Supervised Learning

In these problems, a data set S is composed with both labelled data and unlabelled data. Here the unlabelled data usually plays the role as the assistant data that helps the programs to learn the knowledge from the labelled data. Semi-supervised learning can be seen as a scenario where we need to solve a supervised learning task with limited resource.

2.2.4 Reinforcement Learning

Suppose in a given problem, there exist a reward system and a set of rules following which some moves can be made. Then suppose that by taking some actions in accordance with these rules, a program can gain rewards or feedback from the system. The goal of reinforcement learning is to develop a program that maximizes the rewards it gains.

2.3 Parameterized Models

In machine learning, **models** can be seen as the carriers of the programs we intend to develop. In a more accurate way, a model is a restricted family $\mathcal{H}$ of hypotheses about the learning intent of a machine learning problem. The process of developing a desired program can be seen as the process of learning the hypotheses from H that can solve the problem with the best performance.

In machine learning, to facilitate the use of the computer algorithms, we generally use the **parameterized models**, which means each model can be fully represented by a finite set of parameters. Let's denote the parameters by θ. If the size of θ in a model is fixed regardless of the size of the dataset, we can further define the model as a **parametric model**, and θ can be seen as an element of a parameter space denoted by Θ: $\theta \in \Theta$. Therefore, the objective of a given task can be is to let the model learn the optimal solution θ'. Without additional explanations, the terms "models" appear in this thesis refer to the parametric models.

Additionally, we introduce two important concepts regarding the parametric models: the complexity and the capacity. The complexity of a model is usually determined by the number of its parameters, and the capacity of a model stands for the number of different solutions the model can fit with its parameters, which is also known as the expressive power of the model. Generally, the complexity and the capacity of a model is correlated.

2.4 Classification Problems

The classification problems are generally a type of the supervised learning problems. Each example $\mathbf{x}$ in the dataset S is assigned a label y which may only take value from a finite set of number: $Y \in \mathcal{Y}$, where $\mathcal{Y}$ is called the label space. Each possible value of y represents each of the classes that $\mathbf{x}$ may belong to, and the number of all its possible values is the number of classes of the problem. Our objective is to develop a classifier that predicts the label Y for any given input feature X.

2.4.1 Binary Classification

Given a dataset $S := \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, we suppose that each $(\mathbf{x}, y)$ is drawn from an unknown joint distribution $P_{X,Y}$. In a binary classification problem, the label variable Y is a binary variable, *i.e.* $\mathcal{Y} = \{0, 1\}$. We are interested in letting the classifier learn the optimal

hypothesis of $P_{Y|X}$ so that we can predict the label Y for any given X by the following rule:

$$Y = \begin{cases} 1 & p_{Y|X}(Y=1|X) > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

2.4.2 Multi-Class Classification

Multi-class classification refers to the classification problems that have more than 2 classes. For a C-class classification problem with a dataset: $S = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, we suppose that $\mathcal{Y} = \{1, 2, \cdots, C\}$. To solve the problem, there are two strategies: the **one-vs-one** strategy and the **one-vs-rest** strategy.

With the one-vs-one strategy, for each pair of the classes (i, j) where $i, j \in \mathcal{Y}$, we view Y as a binary variable taking values from i, j, then we construct a binary classifier for the two classes. Therefore, in total we need to develop $\frac{C \times (C-1)}{2}$ binary classifiers. Then for a given $\mathbf{x}$, we let every classifier predict Y for it. Eventually, we say $Y = i$ if i is predicted for Y by the majority of all the classifiers and the sum of the computed $p_{Y|X}(Y = i|X = \mathbf{x})$ over all the binary classifiers has the highest value.

With the one-vs-rest strategy, we only need to let the model learn the optimal posterior distribution of Y conditioning on X: $P_{Y|X}$. Then, to predict Y for a given $\mathbf{x}$, we can compute $[p_{Y|X}(Y = 1|X = \mathbf{x}), p_{Y|X}(Y = 2|X = \mathbf{x}), \ldots, p_{Y|X}(Y = C|X = \mathbf{x})]$ and make the prediction based on the following rule:

$$Y = \arg\max_{i} \left(p_{Y|X}(Y = i|X = \mathbf{x}) \right) \tag{2.2}$$

where $i \in \mathcal{Y}$.

2.4.3 Classification Accuracy

To let the model learn the optimal solution, we need a measurement that determines how well learned solution is. For classification problems, one of the most intuitive measurements is the classification accuracy. Given dataset $S = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, let's denote the model's prediction of each example $\mathbf{x}$ by $f_\theta(\mathbf{x})$. Then the classification accuracy of the model on S is defined as:

$$Acc_S := \frac{\sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in S} \mathbb{1}(f_\theta(\mathbf{x}^{(i)}) = y^{(i)})}{|S|} \tag{2.3}$$

where $|\cdot|$ stands for the cardinality of the set "$\cdot$", and $\mathbb{1}$ is the indicator function:

$$\mathbb{1}(a) := \begin{cases} 1 & \text{“}a\text{” is true} \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Similarly we can defined the classification error as:

$$\begin{aligned} Err_S &= \frac{\sum_{(\mathbf{x}^{(i)}, y^{(i)}) \in S} \mathbb{1}(f_\theta(\mathbf{x}^{(i)}) \neq y^{(i)})}{|S|} \\ &= 1 - Acc_S \end{aligned} \tag{2.5}$$

2.4.4 Common Datasets for Classification Problems

CIFAR10 and **CIFAR100** [23] are two of the most common datasets used for image classification problems. CIFAR10 contains 60000 colored images, each of which is in the shape $32 \times 32 \times 3$. The images are categorised into 10 classes in the balanced way, and among the 6000 images in each class there are 5000 images for training and 1000 images for testing. CIFAR100 contains the same images of CIFAR10, but they are categorised into 100 classes in CIFAR100, and in each class there are 500 images for training and 100 images for testing.

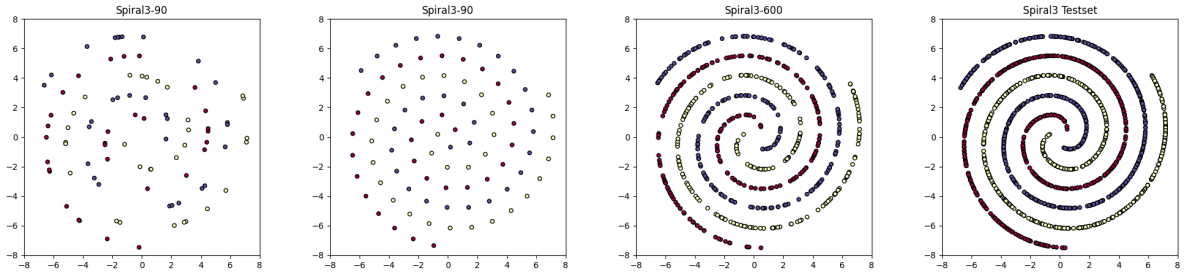
MNIST [11] contains in total 70000 images of handwritten digits which contains 60000 training images and 10000 testing images, and they are categorised into 10 classes, representing the 10 different digits. The images are all grayscale images, each of which is in the shape 28×28.

SVHN [33] contains the colored images of the digits that appear in the real world. The images are categorised into 10 classes like MNIST. The images includes 73257 regular training images, 26032 testing images and 531131 additional training images. There are two formats of the images in SVHN: in format 1 the images are kept their original formats and shapes in the Google street views from which the images are obtained, and thus there may be multiple labels for a single image since it may contain more than one digits. In format 2 the images are cropped into the shape $32 \times 32 \times 3$, and they are all centered on one digit which is also taken as the label of the image. In this thesis, we adopt the second format of the regular training images and all the testing images of SVHN in the experiments.

The **Spiral** datasets are a type of the toy datasets commonly used by the researchers when examining their algorithms or verifying their theories on simpler cases. In our thesis, we

construct our customized Spiral datasets in the following way: on the Cartesian coordinate system, we first draw a quarter-circle counterclockwise with radius r with the origin point as the center, then we move the center down by distance r, and we draw a quarter-circle with radius $2r$ following the previous arc; after that we move the center to the right by r and draw another quarter arc with radiu $3r$. By repeating the processes mentioned above, we can obtain a spiral curve, which we define as the data manifold of the data in one class. To create the data manifolds of the other classes, we only need to rotate the curve about the origin. Eventually, we can generate the examples by randomly sampling the data points on the curves.

In this thesis, if a $\{C\}$-class Spiral dataset contains $\{n\}$ examples in each class, we denote the dataset by "Spiral$\{C\}$-$\{n\}$". Additionally, we define that a Spiral dataset is said to be uniform if the distances between the adjacent examples in the same class are equal and this distance is the same in all the classes. In Figure 2.1 we have shown a few examples of the 3-class Spiral datasets.



(a) Spiral3-90 (b) uniform Spiral3-90 (c) Spiral3-600 (d) Spiral3 Testset

Figure 2.1: 3-class Spiral Datasets

2.5 Training

To enable the model to learn the optimal solution, we can design an algorithm which guides it to update its parameters automatically towards the optimal states by learning the knowledge from the data. This process is called **training** a model, and the data used to train the model is particularly called the training data.

2.5.1 Hyperparameters

Hyperparameters refer to group of parameters that are configured before training a model to control the training process. Unlike the model parameters, the hyperparameters are not trainable.

2.5.2 Setup of the Training Targets

In a classification problem, the setup of the training targets is essential in the preparation stage. The training targets are normally constructed by transforming the labels, and they describe the ground-truth posterior distributions of the labels for the given input features. Roughly speaking, the objective of training a model is to close the difference between the model's hypotheses and the training targets of the input features.

In a C-class classification problem with a dataset $S = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n}$, each y is primarily defined as the index label of the input feature. It serves simply as a sign of which class the corresponding input feature $\mathbf{x}$ belongs to.

Let's denote the space of the input features by $\mathcal{X}$ such that: $X \in \mathcal{X}$. Also, we can let $\mathcal{P}(\mathcal{Y})$ denote the space of the posterior distributions of Y such that: $P_{Y|X} \in \mathcal{P}(\mathcal{Y})$. Suppose we have a model parameterized by θ which predict $P_{Y|X}$, then the model can be represented by a function $f_\theta : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$. Given an input feature $\mathbf{x}$, we have:

$$f_\theta(\mathbf{x}) = [p_{Y|X}(Y = 0|X = \mathbf{x}), p_{Y|X}(Y = 1|X = \mathbf{x}), \ldots, p_{Y|X}(Y = k-1|X = \mathbf{x})]^{\mathrm{T}} \quad (2.6)$$

To let the training targets match the formats of the model's outputs, in practice we normally encode the index labels into **one-hot labels**. Given an example whose label is $y = c$ where $c \in \mathcal{Y}$, its corresponding one-hot label is a vector in the C-dimensional space. We denote the one-hot label by $\mathbf{y} := [y_1, y_2, \cdots, y_C]^{\mathrm{T}}$, which is defined as follows:

$$y_i = \begin{cases} 1 & i = c \\ 0 & \text{otherwise} \end{cases} \quad (2.7)$$

for $i = 1, 2, \cdots, C$. The one-hot label is also called the ground-truth of $\mathbf{x}$, since it exactly represents the true posterior distribution of Y given the input feature:

$$p_{Y|X}(Y = i|X = \mathbf{x}) = \begin{cases} 1 & i = c \\ 0 & \text{otherwise} \end{cases} \quad (2.8)$$

2.5.3 Loss Functions

The **loss functions** are the essential factors in training the models. The loss function in a classification problem serves as the criteria that reflects the difference between the model's outputs and the training targets of the training data. Given a model parameterized by θ and an example $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{P}(\mathcal{Y})$, we denote the corresponding loss function with respect to $f_\theta(\mathbf{x})$ and $\mathbf{y}$ by $\ell(\theta, \mathbf{x}, \mathbf{y})$ where $\theta \in \Theta$. Let $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^n$ be the training set, then the empirical form of the loss function on S is defined as follows:

$$\begin{aligned} \hat{R}_S(\theta) :&= \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\in S} \ell(\theta, \mathbf{x}, \mathbf{y}) \\ &= \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y})\in S} \ell(\theta, \mathbf{x}, \mathbf{y}) \end{aligned} \tag{2.9}$$

assuming all the examples $\mathbf{x}$'s have the identical prior probabilities. The loss function defined by Equation 2.9 is also known as the **empirical risk**. The objective of training the model is thereby to let it learn the optimal solution θ^* by minimizing the empirical risk on S:

$$\theta^* = \arg\min_{\theta\in\Theta} \hat{R}_S(\theta) \tag{2.10}$$

This training scheme is also known as **empirical risk minimization** (ERM) in the classification problems.

In this thesis, we mainly adopt two types of the loss functions: the **cross-entropy loss** and the **mean square error** (MSE). They are respectively defined as:

$$\ell_{\text{CE}}(\theta, \mathbf{x}, \mathbf{y}) = \sum_{i=1}^{C} y_i \log(f_\theta(\mathbf{x})_i) \tag{2.11}$$

$$\ell_{\text{MSE}}(\theta, \mathbf{x}, \mathbf{y}) = \frac{1}{C} \sum_{i=1}^{C} (y_i - f_\theta(\mathbf{x})_i)^2 \tag{2.12}$$

where we denote the i^{th} elements of $\mathbf{y}$ and $f_\theta(\mathbf{x})$ by y_i and $f_\theta(\mathbf{x})_i$ respectively.

2.5.4 Gradient Descent

Gradient descent is an elementary approach for updating the parameters of the models. In mathematics, the gradient of a function at a point indicates the slope of the function at

the point along each of its axes. It can be seen as a direction towards which if the point is moved, the function's output increases. With gradient descent, during the training process θ is updated iteratively based on the following policy:

1. Initialize θ randomly
2. Update θ by:
$$\theta^{new} = \theta^{old} - \eta \times \frac{1}{|S|} \sum_{(\mathbf{x},\mathbf{y})\in S} \nabla_\theta \ell(\theta^{old}, \mathbf{x}, \mathbf{y}) \tag{2.13}$$
3. Repeat step 2

where η is called the learning rate. The learning rate regulates the step to update θ. Note that for a classification problem, gradient descent can't necessarily lead the model to learn the exact solution of the optimization problem in Equation 2.9, since the parameters are updated discretely. However, with careful design of the algorithm, we can still enable the model to approximately fit the optimal solution.

2.5.5 Stochastic Gradient Descent

In **stochastic gradient descent** (SGD), the gradient at each parameters update is approximated from a single training example $(\mathbf{x}, \mathbf{y})$ instead of the entire training set. The algorithm works as follows:

1. Initialize θ randomly
2. Draw a random $(\mathbf{x}, \mathbf{y})$ from S
3. Update θ by:
$$\theta^{new} = \theta^{old} - \eta \times \nabla_\theta \ell(\theta^{old}, \mathbf{x}, \mathbf{y}) \tag{2.14}$$
4. Repeat steps 2 and 3.

2.5.6 Mini-Batch Gradient Descent

Due to the memory constraints in the computing resources like the **central processing units** (CPUs) and the **graphics processing units** (GPUs), sometimes we are unable to feed the entire dataset to the model to compute the gradients all at once. On the other hand, it can be inefficient to compute the gradient of only one example each time, since the GPUs' parallel processing capability is not even utilized. Therefore, in practice we normally adopt an approach called **mini-batch gradient descent**.

In mini-batch gradient descent, in each update we randomly draw a subset of S and compute the average of the gradients of all the examples in the subset. Such subsets are called the mini-batches. The overall algorithm is described as follows:

1. Initialize θ randomly
2. Draw a random mini-batch $\mathcal{B} \subset S$
3. Update θ by:
$$\theta^{new} = \theta^{old} - \eta \times \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{B}} \nabla_\theta \ell(\theta^{old}, \mathbf{x}, \mathbf{y}) \tag{2.15}$$
4. Repeat step 2 and 3

Additionally, we define the process of each batch being processed through the algorithm as an "iteration".

2.5.7 Pragmatical Implementation

When training the model for a classification problem, the mini-batch gradient descent is mostly used for parameters update. However, instead of closely following the standard definition of the algorithm introduced in Section 2.5.6, in practice a particular variation of it is more commonly adopted. In this variation algorithm, at each update a batch is randomly drawn from the training set S without replacement and the size of the batch is prefixed as one of the hyperparameters.

We define each process of all the examples in S being processed through the aforementioned algorithm as an **epoch**. The total number of epochs together with the batch size govern the duration of the training process. Algorithm 2.1 shows the pseudo code that describes the training scheme mentioned above.

Algorithm 2.1

Input: $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$, θ
  ne = Total Number of Epochs
  bs = Batch Size
  $nb = \lceil \frac{n}{bs} \rceil$[1] (Number of the Batches in Each Epoch)
  η = Learning Rate

  for *epoch* **in** $[1 : ne]$ **do**
    $S' \leftarrow$Randomly Shuffled S
    $idx = 1$
    for *batch* **in** $[1 : nb]$ **do**
      $\mathcal{B} = S'[idx : idx + bs - 1]$
      $\theta = \theta - \eta \times \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{B}} \nabla_\theta \ell(\theta, \mathbf{x}, \mathbf{y})$
      $idx = idx + bs$
    end for
  end for
Output: θ

2.6 Performance Evaluation

2.6.1 Underfitting

Underfitting is an issue in machine learning that the trained model fails to adapt to the training data. This usually happens when the capacity of the model is low or the training algorithm is not well designed, making the model unable to learn the optimal solution.

To solve the underfitting issue, normally we can increase the capacity of the model or extend the duration of the training process. Also, we may assign a relatively small value to the learning rate. If the learning rate is large, when updating the parameters the model may skip and miss the optimal solution, since the stride is too large.

[1] Given a real number a, $\lceil a \rceil$ stands for its ceiling, *i.e.* the smallest integer greater than or equal to a.

2.6.2 Testing

For a classification problem we normally evaluate the performance of the trained model by its classification accuracy. With cautious design of the model and the training algorithm, we generally can enable the model to perfectly fit the training data. In other words, we can achieve 100% accuracy. However, from the perspective of the model's generalization performance, we are also interested in evaluating the model's classification accuracy on unseen data. To do so, we can set up an extra dataset that serves as the testing set which contains different examples from the training set. The process of evaluating the model's performance on the testing set is thereby called testing. In this thesis, we use the terms "training accuracy" and "testing accuracy" to represent a model's classification accuracy on the training set and the testing set respectively.

2.6.3 Generalization

The **generalization** or generalizability of a trained model refers to its adaptability to the unseen data. It is normally measured by the gap between the model's testing accuracy and training accuracy. In practice, a trained model's testing accuracy is always not as good as its training accuracy. Such a difference is called the **generalization gap**.

2.6.4 Overfitting

Normally as the training process proceeds, the model's training loss converges to 0, and its training accuracy approaches 100% gradually, whereas its testing accuracy increases as well. However, occasionally at a certain time point in the training process, the model's testing accuracy may start to drop. Such a phenomenon where the generalization gap keeps becoming larger along with the training process is known as the **overfitting** issue. We believe that the main cause of this issue is that the model fits its classification boundaries too close to the training examples. As a consequence, it loses its regularities in the out-of-distribution data.

2.7 Regularization

2.7.1 Introduction

In machine learning, **regularization** refers to a process added in the training process to improve the generalization of the trained models and to avoid the overfitting issue.

2.7.2 Early Stopping

Early stopping is a heuristic strategy of regularization. It aims to prevent the model from being trained for too long so that the model doesn't fit the complex solutions. The criterion of when to stop training the model varies in different tasks, and a common one is to use the validation sets. Specifically, at the preparation stage we separate a part from the training set as the validation set. Then after the parameters are updated at each epoch in the training process, we can measure the performance of the model on the validation set, and we can stop training the model if the validation performance reaches a certain threshold which is prefixed.

2.7.3 Weight Decay

For a vector $\mathbf{v} = [v_1, v_2, \cdots, v_n]^{\mathrm{T}}$ and a positive integer $p \in [0, +\infty)$, we the L_p norm of $\mathbf{v}$ by $\|\mathbf{v}\|_p$, which is defined as follows:

$$\|\mathbf{v}\|_p := \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{2.16}$$

Particularly, the L_2 norm of a vector is also called the **euclidean distance** of the vector.

Weight decay regularizes the network by adding a constraint on the L_2 norm of the weights vector $\mathcal{W}$. It can be seen as adding a penalty term in the training loss as shown below:

$$\hat{R}_S^{\mathrm{wd}}(\mathcal{W}) := \hat{R}_S(\mathcal{W}) + \lambda \|\mathcal{W}\|_2^2 \tag{2.17}$$

where λ controls the strength of the L_2 norm constraint. In the rest of this thesis, the Greek letter λ has also been used as the notation of another abstract concept. To avoid the abuse of the notations, in this thesis we denote the aforementioned λ defined in Equation 2.17 simply by the text "weight decay".

2.7.4 Data Augmentation

Data augmentation is a data-dependent regularization technique. It performs certain types of transformations on the input features of the original training data along with the training process. Figure 2.2 shows some examples of data augmentation performed on the image of a truck in the CIFAR10 dataset. Note that the ways of data augmentation vary in different tasks. For example, the horizontal flips are commonly adopted on the CIFAR10 images, but it is not applicable on the MNIST images since the upside-down digits are invalid.

(a) (b) (c) (d)

Figure 2.2: Examples of Data Augmentation. (a): Original Image of a Truck; (b): Rotation; (c): Random Crop; (d): Horizontal Flip.

2.8 Artificial Neural Networks

2.8.1 Artificial Neurons

The primitive idea of AI is to enable the machines to imitate the thinking patterns of the human beings. An intuitive way to do so is to start by studying the working mechanisms of the humans' brains thinking.

In biology, the **neural networks** refer to the connection architecture of the neural cells, which is also known as the neurons. Such an architecture enables the bio-electrical signals to be transmitted between the neurons, which we believe is the fundamental of our thinking ability.

In the field of machines learning, the idea of the **artificial neural networks** is inspired by the aforementioned biological neural networks. The artificial neural networks refer to a

class of the parametric models composed of interconnected nodes, which is also known as the **artificial neurons**. The connecting structure of the artificial neurons in an artificial neural network is also defined as the topological structure of the network. To simplify the notations, in the rest this thesis the terms "neural networks" and "neurons" serve as the shortening of "artificial neural networks" and "artificial neurons".

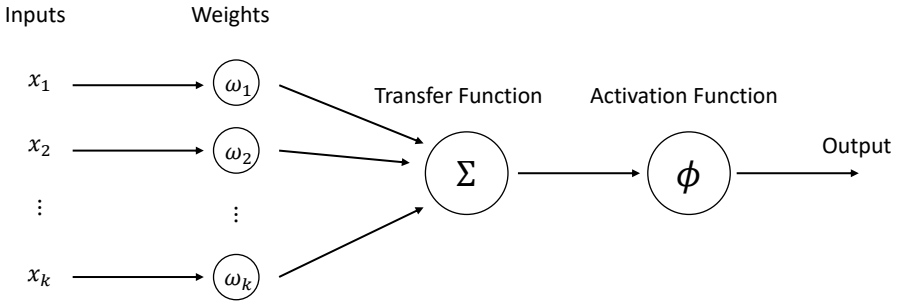


Figure 2.3: An Example of Artificial Neuron

Figure 2.3 shows the example of an artificial neuron. It consists of the inputs nodes, the output node, the weights, a transfer function and an activation function. For each input x, a weight w is assigned to it, where $x, w \in \mathbb{R}$. The transfer function, which we denote by $\sum$, is a linear function that sums up the products $w_i \cdot x_i$ for $i = 1, 2, \cdots, k$ and transmit the summation to the activation function. Normally the activation function is defined as a scalar function $\phi : \mathbb{R} \to \mathbb{R}$, which computes the final output of the neuron. Let's denote the output of the m^{th} neuron in the network by $o^{(m)}$, then we have:

$$o^{(m)} = \phi\left(\sum_{i=1}^{k} w_i^{(m)} x_i^{(m)}\right) \tag{2.18}$$

$o^{(k)}$ is then propagated to the next connected neurons as parts of their inputs.

Equation 2.18 shows that the computation process in each neuron can be defined as a function $o : \mathbb{R}^k \to \mathbb{R}$. Therefore, an artificial neural network can be seen as a computation graph that fits a function via assembling a group of some minor functions, and we treat the collection of the weights in all the network's neurons as the model's trainable parameters. In addition, the activation functions and the numbers of the neurons determine the expressive power of the network.

Additionally, the connections between the neurons in a neural network serve as the routes through which the information is transmitted. The **feedforward neural networks** refer

to a type of the neural networks in which the connections are all unidirectional. In other words, the information can only be transmitted in one direction through each of the connections in these networks. In this thesis, only adopt the feedforward neural networks as the models in our experiments and analyses.

2.8.2 Layered Structure

On the basis of the neurons construction, an artificial neural networks is constructed into the layered structure. In other words, a neural network is made up of several layers arranged in a preset order, and the neurons are arranged on the layers. Generally, each layer with the neurons is a local integral inside the neural network. The collection of the inputs on all the neurons in each layer serve as the input of the layer, and the outputs of all these neurons together serve as the layer's output. Normally, the number of the neurons in a layer is defined as the width of the layer.

The layers of a network can be mainly categorised into 3 types based on their positions in the network: the input layer, the output layer and the hidden layers . Let $\mathbf{x}$ denote the input feature of a given example. As the input of the network, $\mathbf{x}$ is transmitted from the input layer to the 1^{st} hidden layer directly. Notice that the width of the input layer must match the dimension of $\mathbf{x}$. Also, the output of a neural network is simply the output of the output layer. Thus for a C-class classification task, the width of the network's output layer ought to be C as well.

The hidden layers of a neural network carry all the network's trainable parameters, *i.e.* the weights of all the hidden neurons. Normally, we refer to a neural network consisting of in total L layers as an $L-1$-layer network (excluding the input layer since it doesn't contain any trainable weight), but the network's depth is defined as L, *i.e.* the total number of the layers including all the three types. Additionally, the width of a neural network is defined as the width of the widest layer of the network.

2.8.3 Activation Functions

As mentioned in Section 2.8.1, the activation function is an essential component of an artificial neuron. The activation functions are normally defined to be nonlinear and differentiable. Some commonly used activation functions and their derivatives are listed below:

Sigmoid Function (σ):

$$\sigma(x) := \frac{1}{1 + \exp(-x)} \tag{2.19}$$

$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x)) \tag{2.20}$$

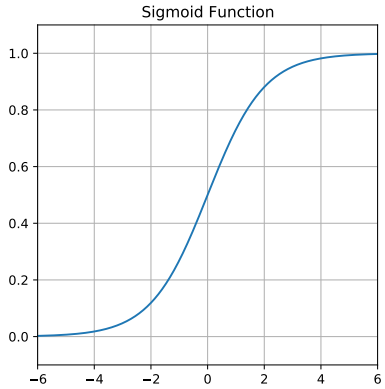


Figure 2.4: Sigmoid Function

Hyperbolic Tangent Function (tanh):

$$\tanh(x) := \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{2.21}$$

$$\tanh'(x) = 1 - \tanh^2(x) \tag{2.22}$$

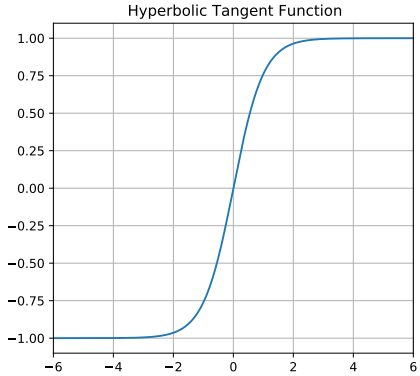


Figure 2.5: Hyperbolic Tangent Function

Rectified Linear Unit (ReLU):

$$\text{ReLU}(x) := \max\{x, 0\} \tag{2.23}$$

$$\text{ReLU}'(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.24}$$

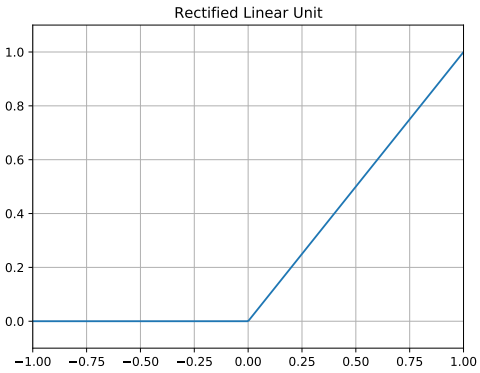


Figure 2.6: Rectified Linear Unit

In practice we normally arrange the same activation functions on all the neurons in each layer. Thus, for a hidden layer of width k we may define $\Phi : \mathbb{R}^k \to \mathbb{R}^k$ to be the layer's activation function which consists of all the neurons' activation functions.

For a C-class classification task, a vector-wise activation function $\Phi : \mathbb{R}^C \to \mathbb{R}^C$ called the **softmax** function is particularly assigned on the output layer of the network. It is defined as follows:

$$\textbf{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_{i=1}^{C} \exp(a_i)}, \frac{\exp(a_2)}{\sum_{i=1}^{C} \exp(a_i)}, \cdots, \frac{\exp(a_C)}{\sum_{i=1}^{C} \exp(a_i)} \right]^{\mathrm{T}} \tag{2.25}$$

The softmax function normalizes the initial output of the output layer while retaining the numerical relationships between the C elements in it. Therefore, it enables the network to fit the solutions that predict the posterior distributions of the label variable Y for any input feature $\mathbf{x}$.

2.8.4 Multilayer Perceptrons

A **multilayer perceptron** (MLP) is a feedforward layered neural network all of whose adjacent layers are fully connected. In other words, each neuron in any layer is connected with all the neurons in the neighboring layers. Figure 2.7 briefly shows the architecture of a 3-layer MLP.

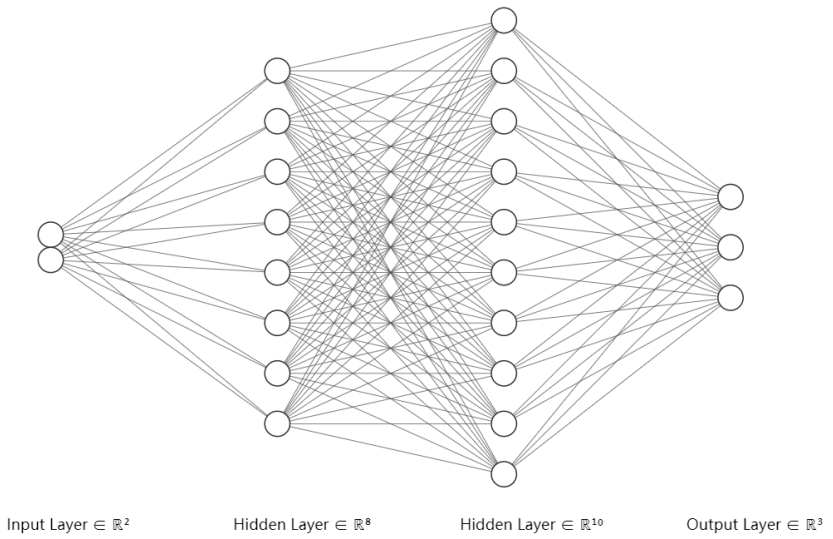


Figure 2.7: 3-Layer MLP [25]

For a L-layer MLP, let's denote the output of the input layer by $\mathbf{h}^{(0)}$, and let's denote the output of the l^{th} layer (excluding the input layer) by $\mathbf{h}^{(l)}$. Since the layers are fully connected, the l^{th} layer (excluding the input layer) carries in total $\mathbf{dim}(\mathbf{h}^{(l-1)}) \times \mathbf{dim}(\mathbf{h}^{(l)})$ many weights[2]. Let's denote the collection of these weights by $\mathbf{W}^{(l)}$, which can be seen as a matrix whose shape is $\mathbf{dim}(\mathbf{h}^{(l-1)}) \times \mathbf{dim}(\mathbf{h}^{(l)})$ denoted by. Then we can compute $\mathbf{h}^{l}$ as follows: $\mathbf{h}^{(l)} := \Phi^{(l)}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)})$, where $\Phi^{(l)} : \mathbb{R}^{\mathbf{dim}(\mathbf{h}^{(l)})} \to \mathbb{R}^{\mathbf{dim}(\mathbf{h}^{(l)})}$ denotes the activation function of the l^{th} layer. Suppose $\mathbf{h}^{(0)} = \mathbf{x}$ is the input of the MLP, and let's denote the output of the MLP by $f_{\mathcal{W}}(\mathbf{x})$, then we have:

$$f_{\mathcal{W}}(\mathbf{x}) = \mathbf{softmax}\Big(\mathbf{W}^{(L)} \cdot \Phi^{(L-1)}\Big(\mathbf{W}^{(L-1)} \cdot \Phi^{(L-2)}\Big(\mathbf{W}^{(L-2)} \cdot \ldots \cdot \Phi^{(1)}\Big(\mathbf{W}^{(1)} \cdot \mathbf{x}\Big)\Big)\Big)\Big) \quad (2.26)$$

[2] $\mathbf{dim}(\mathbf{a})$ represents the dimension of the vector space of vector $\mathbf{a}$.

where $\mathcal{W}$ denotes the collection of all the weights matrices. Thus given a training set $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$, the empirical risk $\hat{R}_S(\mathcal{W})$ can be computed following Equation 2.9.

2.9 Convolutional Neural Networks

The **convolutional neural networks** (CNNs) are a type of neural networks composed of the convolutional layers, the pooling layers and the fully connected layers. Owning to their abilities of processing the multidimensional tensors and their efficient utilization of the weights, the CNNs are widely used in the image classification tasks nowadays.

2.9.1 Tensors

Unlike an MLP whose inputs are in the vectors forms, a CNN takes the tensors as its inputs. Briefly speaking, the tensors can be seen as the generalized forms of the multidimensional arrays of values. Generally, we define the rank of a tensor as the number of its dimensions, and we define the shape of a tensor as a tuple which describes the sizes of all the dimensions of the tensor. In this thesis, to distinguish the notations of the tensors from those of the vectors, we refer to an n-D tensor as a tensor whose rank is n. For example, each image in the CIFAR10 dataset can be seen as a 3D tensor whose shape can be written as $(32, 32, 3)$. In fact, any vector and any matrix can also be simply seen as a 1D tensor and a 2D tensor respectively, which gives us great convenience in describing the mathematical forms of the networks' inputs.

2.9.2 Convolutional Layers

The **convolutional layers** are the essential components of a CNN. The inputs and the outputs of each convolutional layer are also called the input and output feature maps of the layer respectively. Unlike the neurons in the MLPs, in the CNNs each convolutional layer consists of a kernel, which can also be seen as a filter. On each convolutional layer, the kernel is constructed as a tensor, and it serves as a sliding window that moves on the input feature maps of the layer. At each position on the feature map the kernel slides to, the network computes the dot product between the kernel and the block in the input feature map that overlaps with the kernel. Such a computation process is also called the 2D convolution operation. The computation results of all the 2D convolutions on the input feature maps together compose the output feature map of the layer. In the CNNs we can

also define the activation functions for the convolutional layers to process their output feature maps. In practice, we normally adopt ReLU as the activation functions of the convolutional layers.

2.9.3 Max-Pooling Layers

The **max-pooling layers** are a common type of the pooling layers. They are normally arranged after the convolutional layers and are used to downsample the output feature maps of the convolutional layers. They partition the feature maps into several patches and extract the maximum value in each patch. The max-pooling layers and the convolutional layers together play the role to abstract the essential features from the CNNs' inputs.

2.9.4 Fully Connected Layers

The **fully connected layers** are also called the dense layers. From the perspective of the architecture, a fully connected layer can simply be seen as an MLP merged into a CNN as a part of it. In a CNN, the fully connected layer is normally arranged as the last component of the network, and it plays the major role in enabling the CNN to fit the solutions of the given task.

Figure 2.8 briefly illustrates the architecture of an instance CNN.

2.9.5 Residual Neural Networks

The **residual neural networks** (ResNets) [18] are a popular type of the CNNs. From the conventional understanding of the artificial neural networks, we believe that we can improve a network's expressive power by directly increasing its depth. However, by doing so we may also encounter the problem that the gradients correspond to the layers near the output layer of the network may vanish, making the gradient descent algorithms unable to work. This is formally called the vanishing gradient problem. To solve this issue, in [18] the authors have proposed the ResNets which consist of the residual blocks. In a ResNet, each residual block is composed of several convolutional layers and max-pooling layers. Aside of processing the input feature maps of the block in the regular way through the layers, the block also allows the input feature maps to be directly transmitted to the end of the block and be merged into the block's outputs.

In this thesis, we have mainly used ResNet18 and ResNet34 in our experiments, where the numbers indicate the depths of the networks.

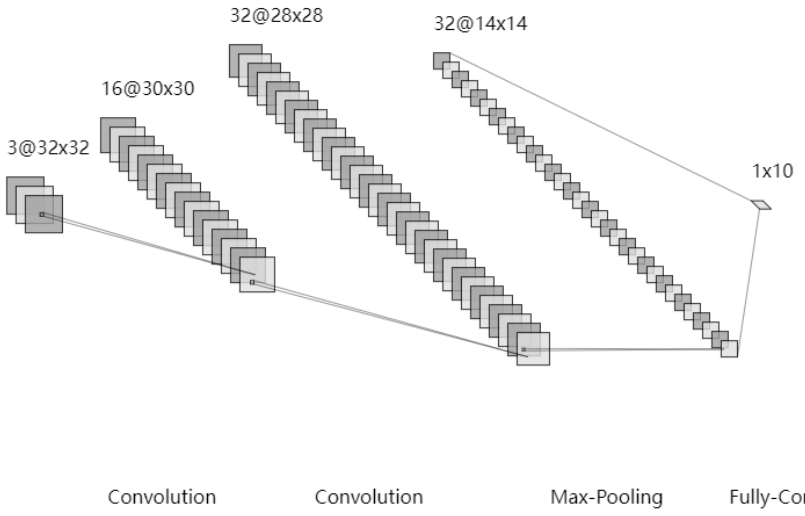
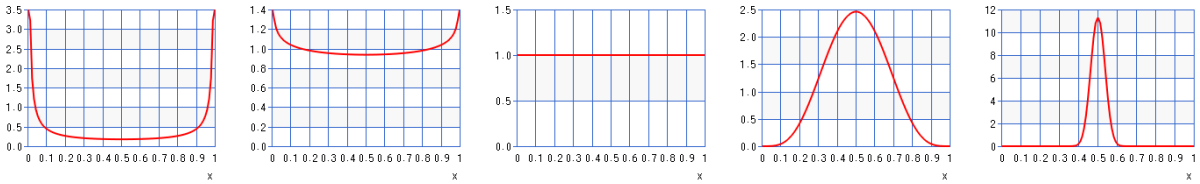


Figure 2.8: CNN [24]

2.10 Beta Distribution

The Beta distribution, denoted by Beta(α, β) for $\alpha, \beta > 0$, defines a family of the continuous probability distributions on the interval $[0, 1]$. Particularly, if $\alpha = \beta$, the distribution is axisymmetric with respect to the point 0.5 on its domain. Figure 2.9 illustrates the probability density functions (PDFs) of the Beta distribution Beta(α, α) with different α's.



(a) $\alpha = 0.1$ (b) $\alpha = 0.9$ (c) $\alpha = 1$ (d) $\alpha = 5$ (e) $\alpha = 100$

Figure 2.9: PDF of Beta(α, α) [10]

2.11 Multivariate Gaussian Distribution

The **multivariate Gaussian distribution** is a type of the continuous probability distributions defined for the multidimentional random vectors. It can be seen as the generalized form of the univariate Gaussian distribution, which is defined for the scalar random variables. If a K-dimensional random vector $\mathbf{X} = [X_1, X_2, \cdots, X_K]^T$ follows the K-variate Gaussian distribution, we may denote it by $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denote the mean vector and the covariance matrix of $\mathbf{X}$ respectively. They are defined as follows:

$$\begin{aligned} \boldsymbol{\mu} &:= \mathbb{E}(\mathbf{X}) \\ &= \big[\mathbb{E}(X_1), \mathbb{E}(X_2), \cdots, \mathbb{E}(X_K)\big]^{\mathrm{T}} \end{aligned} \tag{2.27}$$

$$\boldsymbol{\Sigma} := \begin{bmatrix} \operatorname{cov}(X_1, X_1) & \operatorname{cov}(X_1, X_2) & \cdots & \operatorname{cov}(X_1, X_K) \\ \operatorname{cov}(X_2, X_1) & \operatorname{cov}(X_2, X_2) & \cdots & \operatorname{cov}(X_2, X_K) \\ \vdots & \vdots & \ddots & \vdots \\ \operatorname{cov}(X_K, X_1) & \operatorname{cov}(X_K, X_2) & \cdots & \operatorname{cov}(X_K, X_K) \end{bmatrix} \tag{2.28}$$

where $\operatorname{cov}(X_i, X_j) := \mathbb{E}[(X_i - \mathbb{E}(X_i))(X_j - \mathbb{E}(X_j))]$ is defined as the covariance between X_i and X_j. Notice that $\operatorname{cov}(X_i, X_i)$ is simply the variance of X_i.

If $\boldsymbol{\Sigma}$ is positive definite, the PDF of the multivariate Gaussian distribution is defined as follows:

$$f_{\mathcal{N}}(\mathbf{x}) := \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right) \tag{2.29}$$

where $|\boldsymbol{\Sigma}|$ and $\boldsymbol{\Sigma}^{-1}$ refer to the determinant and the inverse of $\boldsymbol{\Sigma}$ respectively.

2.12 Student's *t*-test

Student's *t*-test is a statistical method used to evaluate the statistical difference between two sequences of values. For example, suppose we carry out two trials of a experiment, and each of them is repeated for n_1 and n_2 times respectively. We can measure the results of the experiments with the real numbers and record them into two arrays: $\mathbf{x}_1$ and $\mathbf{x}_2$. Let's denote the empirical averages of the two arrays by $\overline{\mathbf{x}_1}$ and $\overline{\mathbf{x}_2}$, and let's denote their standard deviations by s_1 and s_2. In the case that $\overline{\mathbf{x}_1}$ and $\overline{\mathbf{x}_2}$ are different, we wish to verify whether their difference is statistically significant. To do so, we can perform the independent Student's t-test. First, we can define our null hypothesis as $H_0 : \mu_1 = \mu_2$,

where μ_1 and μ_2 denote the mathematical means of the outcomes of the two experimental trials. This null hypothesis simply assumes that the two trials of the experiment show no significant differences. Also. we need to set up the significance level of the test, which is conventionally set as 0.05. Then, we can compute the t value of the two arrays as follows:

$$t := \frac{|\overline{\mathbf{x}_1} - \overline{\mathbf{x}_2}|}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{2.30}$$

In addition, $\mathrm{df} := n_1 + n_2 - 2$ is defined as the degree of freedom of the test. Thus, we can search on the t-test table with the computed t value and the degree of freedom to approximate the p value. If p is smaller than the prefixed significance level, we reject H_0 and conclude that the difference between the means of the two tested arrays are statistically significant. Otherwise, we maintain the null hypothesis.

The computation and searching processes in the Student's t-tests are cumbersome. However, in practice the development platforms like Python and Matlab have all provided the packages and the functions to perform the tests. Moreover, they can even help to automatically compute the p values with higher precision than the approximated ones.

2.13 Entropy

Consider a discrete probability distribution P defined on the discrete random variable Y which takes the values from the sample space $\mathcal{Y}$. We denote the entropy of Y by $\mathcal{H}(Y)$, and it is defined as follows:

$$\mathcal{H}(Y) := -\sum_{y \in \mathcal{Y}} P(y) \log P(y) \tag{2.31}$$

2.14 Relative Entropy

The **relative entropy** is also known as the **Kullback–Leibler Divergence** (KL divergence). Consider two discrete probability distributions P and Q defined with respect to the same sample space $\mathcal{Y}$. We denote the KL divergence from Q to P by $D_{\mathrm{KL}}(P\|Q)$, and it is defined as follows:

$$D_{\mathrm{KL}}(P\|Q) := \sum_{y \in \mathcal{Y}} P(y) \log \left(\frac{P(y)}{Q(y)} \right) \tag{2.32}$$

The KL divergence $D_{\mathrm{KL}}(P\|Q)$ is non-negative, and it equals 0 if and only if $P = Q$ holds true.

2.15 Total Variation

The **total variation** between two probability measures P and Q is defined as follows:

$$\mathrm{TV}(P, Q) := \sup_{E} |P(E) - Q(E)| \tag{2.33}$$

where the supremum is with respect to all measurable set E.

Lemma 2.15.1 ([26, Proposition 4.2]). *Let P and Q be two probability distributions on $\mathcal{X}$. If $\mathcal{X}$ is countable, then*

$$\mathrm{TV}(P, Q) = \frac{1}{2} \sum_{x \in \mathcal{X}} |P(x) - Q(x)| .$$

Lemma 2.15.2 (Coupling Inequality [26, Proposition 4.7]). *Given two random variables X and Y with probability distributions P and Q, any coupling $\hat{P}$ of P, Q satisfies*

$$\mathrm{TV}(P, Q) \leq \hat{P}(X \neq Y).$$

Chapter 3

Introduction of Mixup

3.1 Mixup as a Data-Dependent Regularizer

In Section 2.7 we have introduced several common regularization techniques. In 2018, a new regularization technique named **Mixup** was proposed in [44]. Different from the ERM training scheme, Mixup trains the networks on the synthetic data formulated via linearly interpolating the real data pairs randomly drawn from the training set.

Consider a C-class classification task with a training set $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$, where each label $\mathbf{y}$ is treated as the one-hot label of $\mathbf{x}$. Let $((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}'))$ denotes a pair of the training examples randomly drawn from S, and let λ denote a random real number in the interval $[0, 1]$. Then in the Mixup training scheme, we can formulate the synthetic example $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ as follows:

$$\begin{aligned} \tilde{\mathbf{x}} &= \lambda \mathbf{x} + (1 - \lambda)\mathbf{x}' \\ \tilde{\mathbf{y}} &= \lambda \mathbf{y} + (1 - \lambda)\mathbf{y}' \end{aligned} \tag{3.1}$$

where $\lambda \in [0, 1]$ is also called the interpolation coefficient.

In Mixup, instead of using the original training set S, the training is performed on a synthetic dataset $\widetilde{S}$ obtained by interpolating training examples in S. Without loss of generality, we consider the synthetic dataset for each interpolating parameter $\lambda \in [0, 1]$ as the set $\widetilde{S}_\lambda$, which is defined by:

$$\widetilde{S}_\lambda := \{(\lambda \mathbf{x} + (1 - \lambda)\mathbf{x}', \lambda \mathbf{y} + (1 - \lambda)\mathbf{y}') : (\mathbf{x}, \mathbf{y}) \in S, (\mathbf{x}', \mathbf{y})' \in S\} \tag{3.2}$$

Suppose the model is parameterized by θ, then the empirical Mixup loss can be defined as

follows:

$$\begin{aligned}\hat{R}_{\widetilde{S}_\lambda}(\theta) :&= \frac{1}{|\widetilde{S}_\lambda|}\sum_{(\tilde{\mathbf{x}},\tilde{\mathbf{y}})\in\widetilde{S}_\lambda}\ell(\theta,\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\ell(\theta,\lambda\mathbf{x}^{(i)}+(1-\lambda)\mathbf{x}^{(j)},\lambda\mathbf{y}^{(i)}+(1-\lambda)\mathbf{y}^{(j)})\end{aligned} \tag{3.3}$$

A suggested by the authors in [44], most often the interpolation coefficient λ is randomly drawn from a symmetric Beta distribution: $\text{Beta}(\alpha,\alpha)$, where $\alpha > 0$ and can be seen as a hyperparameter of the Mixup algorithm. In this case, we define the expectation of the empirical Mixup loss with respect to λ as the Mixup target loss, which we denote by $\hat{R}_{\widetilde{S}}(\theta,\alpha)$:

$$\hat{R}_{\widetilde{S}}(\theta,\alpha) := \mathbb{E}_\lambda \hat{R}_{\widetilde{S}_\lambda}(\theta) \tag{3.4}$$

where $\lambda \sim \text{Beta}(\alpha,\alpha)$. Thus, with α prefixed, the objective of Mixup training can be defined as learning the optimal solution θ^* that minimizes $\hat{R}_{\widetilde{S}}(\theta,\alpha)$:

$$\theta^* = \arg\min_{\theta\in\Theta}\hat{R}_{\widetilde{S}}(\theta,\alpha) \tag{3.5}$$

where Θ denotes the parameter space.

In addition, recall Equation 1.1 we can notice that since $(\mathbf{x},\mathbf{y})$ and $(\mathbf{x}',\mathbf{y}')$ are independently drawn from the training set, it is possible that they are the same example. In this case, $(\tilde{\mathbf{x}},\tilde{\mathbf{y}})$ is actually also a real example. Nevertheless, in this thesis we call all the examples formulated via Equation 1.1 as the synthetic examples. Moreover, in this thesis we define the synthesizing operations between the examples of the same class as the "in-class mixing", and we define those between the examples of different classes as the "cross-class mixing".

3.2 Pragmatic Implementation of Mixup

The main difference between ERM and Mixup is that the latter training scheme requires the data synthesis, which consists of two factors: generating the λ's and pairing the examples. If we rigidly follow Equation 3.5 to implement the Mixup training scheme, the algorithm complexity may be too high: the space complexity is squared compared to ERM, and the expectation can only be approximated via discretely sampling the λ's from $\text{Beta}(\alpha,\alpha)$.

Thus, as suggested by the authors of [44] in their source code, in practice we normally adopt a batch-wise way to implement Mixup. It is in accordance with the principle of the

mini-batch gradient descent algorithm. Given a batch $\mathcal{B}$, we first shuffle the examples in it, and we denote the shuffled copy of $\mathcal{B}$ by $\mathcal{B}'$. Also, we generate a value λ from Beta(α, α) as the interpolation coefficient. Then we can formulate the synthetic examples by mixing the examples in $\mathcal{B}$ and $\mathcal{B}'$ pair-wise: $\widetilde{\mathcal{B}} = \lambda\mathcal{B} + (1-\lambda)\mathcal{B}'$, where $\widetilde{\mathcal{B}}$ denotes the synthetic batch containing the synthetic examples. More specifically, suppose $\mathcal{B} := \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{m}$ and $\mathcal{B}' := \{(\mathbf{x}'^{(i)}, \mathbf{y}'^{(i)})\}_{i=1}^{m}$, then we have:

$$\begin{aligned} \widetilde{\mathcal{B}} :&= \{(\tilde{\mathbf{x}}^{(i)}, \tilde{\mathbf{y}}^{(i)})\}_{i=1}^{m} \\ &= \{(\lambda\mathbf{x}^{(i)} + (1-\lambda)\mathbf{x}'^{(i)}, (1-\lambda)\mathbf{y}^{(i)} + (1-\lambda)\mathbf{y}'^{(i)})\}_{i=1}^{m} \end{aligned} \tag{3.6}$$

Algorithm 3.1 shows the pseudo code of the complete training process of the Mixup policy.

Algorithm 3.1

Input: $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$, θ
  ne = Total Number of Epochs
  bs = Batch Size
  $nb = \lceil \frac{N}{bs} \rceil$ (Number of the Batches in Each Epoch)
  η = Learning Rate
  $\alpha \in (0, +\infty)$

  for *epoch* **in** $[1 : ne]$ **do**
    $S' \leftarrow$Shuffled S
    $idx = 1$
    for *batch* **in** $[1 : nb]$ **do**
      $\lambda \leftarrow \text{Beta}(\alpha, \alpha)$
      $\mathcal{B} = S'[idx, idx + bs - 1]$
      $\mathcal{B}' \leftarrow$ Shuffled $\mathcal{B}$
      $\widetilde{\mathcal{B}} = \lambda\mathcal{B} + (1-\lambda)\mathcal{B}'$
      $\theta = \theta - \eta \frac{1}{|\widetilde{\mathcal{B}}|} \sum_{(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \in \widetilde{\mathcal{B}}} \nabla_\theta \ell(\theta, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$
      $idx = idx + bs$
    end for
  end for
Output: θ

3.3 Related Works

After the initial work of [44], a series of the Mixup scheme's variants have also been proposed [14, 16, 17, 22, 36, 39, 42]. For example, *AdaMixup* [16] trains an extra network to dynamically determine the interpolation coefficient hyperparameter α. *Manifold Mixup* [39] performs the linear mixing on the hidden states of the neural networks. *CutMix* [42] mixes each pair of the training images by extracting a part from each of the two images and merging them together. *k-Mixup* [14] randomly draws two batches of k training examples and pairs the examples in a way that minimizes the Wasserstein distance between the two batches. *GenLabel* [36] reassigns the training targets for the synthetic data using an additional generative model trained on the training data.

Aside of the application aspects of Mixup, its working mechanism and the possible limitations are also being explored constantly. For example, in [46] the authors demonstrate that Mixup yields an upper bound on the Rademacher complexity of the class of the solutions that a network can fit, which in turn bounds the generalization error of the network. [38] shows that Mixup helps to improve the calibration of the trained networks. [47] theoretically demonstrates that the calibration effect of Mixup is correlated with the capacity of the network. [16] introduces the concept of manifold intrusion, which is an issue that may hurt the performance of Mixup. [36] demonstrates that the linear labelling policy of Mixup is sub-optimal.

Chapter 4

On the Manifold Intrusion

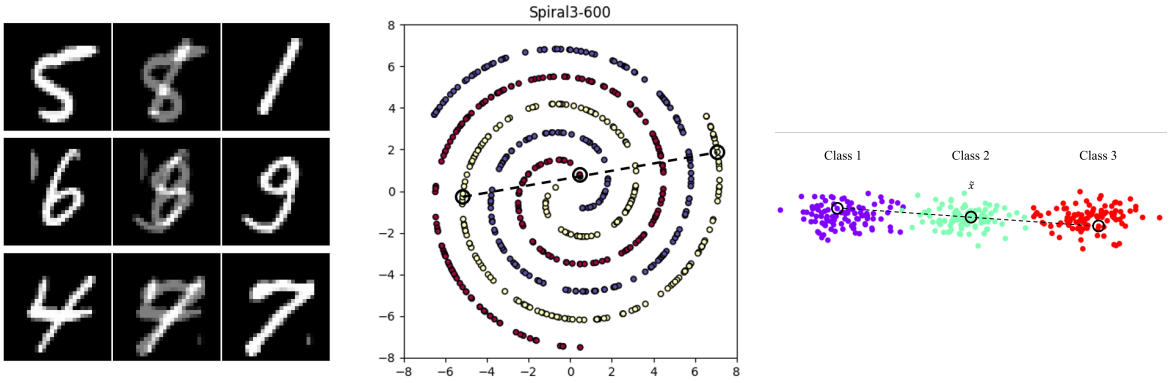
In this chapter, we first explain the concept of **manifold intrusion** proposed by [16] We summarize the majority of the current arts about solving manifold intrusion into two strategies: the **relabelling** strategy and the **cautious mixing** strategy. We then introduce two of our proposed relabelling algorithms, and we conduct extensive experiments to examine the algorithms' effectiveness. We also compare the performance of the two strategies through some of the state-of-the-art works and some of our own works. In the end, we conclude that the cautious mixing strategy is in general more effective and efficient than the relabelling strategy to solve manifold intrusion.

4.1 Manifold Intrusion

As a regularization technique, Mixup has been shown to be effective in improving the generalization and robustness of the deep networks on various tasks and datasets. While several research works have been aiming to theoretically explain "why" Mixup works, some others are also exploring "if" Mixup works universally.

In [16], the authors introduce an issue in Mixup training where the synthetic examples collide with the real examples. This issue is called **manifold intrusion**. In the classification problems, we may assume that the examples of different classes lie on different data manifolds. Under this hypothesis, manifold intrusion can be seen as a potential phenomenon in Mixup training where the synthetic examples "intrude" the data manifolds of the real examples, resulting in the conflicts between the synthetic labels and the ground-truth labels of the synthetic data. In fact, manifold intrusion has already been empirically shown to bring about the underfitting issues in the Mixup-trained models [16].

In Figure 4.1 we have provided some examples of manifold intrusion occurring in different datasets. Figure 4.1(a) shows that in the MNIST dataset, a synthetic digit may look like a real digit, which can even be different from both the real digits that formulate it. The similar behavior can also be seen in Figure 4.1(c) but on a 2-dimensional clustered dataset of 3 classes. This fact indicates that manifold intrusion may also occur both the low dimensional datasets. Additionally, in Figure 4.1(b) the synthetic example is formulated by a pair of the real examples in the yellow class. However, it is located in the data manifold of the red class, demonstrating that manifold intrusion may occur in both the cases of cross-class mixing and those of in-class mixing.



(a) MNIST (b) Spiral3-600 (c) 3-Class Clustered Dataset

Figure 4.1: Examples of Manifold Intrusion. (a): In each row, the digits on the two sides are the real images drawn from the MNIST dataset, and the one in the middle is the synthetic image. (b), (c): In each figure, the rings at the both ends on the dashed line segment indicate a pair of the real data, and the ring in the middle indicates the point of the synthetic data feature.

4.2 Relabelling Strategy

Driven by the raising of manifold intrusion, many researchers have been interested in exploring techniques that can solve manifold intrusion so as to prevent it from hurting the generalization performance of the trained models. We summarize a part of them as the **relabelling** strategy. For example, [36] proposes *GenLabel*, which trains an additional generative network to learn the statistical structure of the training data, and the synthetic examples are relabelled based on the generative network's predictions of them.

The relabelling strategy stems from the aforementioned belief that manifold intrusion causes the conflicts within the training targets of the synthetic data. For example, in Figure 4.1(a), the labels of the three synthetic images are all supposed to be "two-hot" according to the linear interpolation policy of Mixup. However, one may also reasonably assign them the one-hot labels since they look close to the real digits. Such contradictions may confuse the model to learn the real knowledge from the data, which we believe is the direct cause of the potential performance degradation. The primary objective of the relabelling strategy is to determine the correct training targets for the synthetic examples that fit its ground-truth labels.

In this thesis, we have designed and proposed two different relabelling schemes, which are explained in details in the following two sections respectively.

4.3 Pseudo Relabelling

4.3.1 Relabelling via Pseudo Labels

The idea of **pseudo relabelling** is inspired by the **pseudo labelling** algorithm. In the field of semi-supervised learning, pseudo labelling is a technique that enables us to utilize the unlabelled data to train a model. To adopt pseudo labelling, we can first train the model on the labelled training examples. Then we let the model make predictions on the unlabelled data, and we transform the its predictions into one-hot vectors. In other words, the positions of the peak values in the transformed one-hot vectors match those in the model's outputs. We assign these one-hot vectors as the training targets of the unlabelled data, and we denote them by the "pseudo labels". Therefore, the unlabelled examples can be treated as the labelled ones, and the training loss is then computed with respect to all the labelled data.

From the effectiveness of pseudo labelling in semi-supervised learning, we believe that even when a model is not well trained, it can still provide meaningful information of the unlabelled data. Recall that to deal with manifold intrusion, we hypothesize that the initial training targets of the synthetic data are not entirely correct. Therefore, if we consider the synthetic data in Mixup training as unlabelled, we may adopt the similar principles of pseudo labelling to help to relabel them.

However, in the conventional scenarios of pseudo labelling, the training examples are guaranteed to be real whether labelled or not, which in Mixup training is not necessarily the case. In Mixup training we may consider only the examples that commit manifold

intrusion as being "approximately real". As for the rest, we may assume that they are located in the void space outside the real data manifolds. Based on this assumption, we propose the pseudo relabelling algorithm. Given a synthetic example, pseudo relabelling enables us to take the weighted average of its synthetic label and pseudo label as its training target.

One of the essential factors of pseudo relabelling is a criterion to determine whether manifold intrusion has occurred with respect to each synthetic example. As mentioned in Section 2.4.2, we understand that a classifier's outputs represent the posterior distributions of the label variables with respect to the given input features. Thus, we can define the peak value in an output of the model as the **winning score**. The winning score reflects the model's confidence level of predicting the corresponding class for the given input.

We have empirically observed that the winning scores in the ERM-trained models' outputs are most often close to 1 no matter the predictions are correct or not. This observation can be seen as an implication that the ERM-trained models tend to behave overconfident, which may in turn result in the overfitting issues. Thus in our proposed pseudo relabelling algorithm, we can set up a threshold on the winning scores. Then, a synthetic example can be considered to be real if the corresponding winning score exceeds the preset threshold.

Additionally, it is necessary that we take the interpolation coefficient λ into consideration to determine the likelihood of manifold intrusion's occurrence. Consider a synthetic input $\tilde{\mathbf{x}} = \lambda\mathbf{x}+(1-\lambda)\mathbf{x}'$. We notice that with λ being shifted either from 1 to 0.5 or from 0 to 0.5, the data point of $\tilde{\mathbf{x}}$ is gradually moved away from the real examples $\mathbf{x}$ or $\mathbf{x}'$ respectively. As a result, the indeterminacy of manifold intrusion's occurrence increases.

By combining the two factors stated above, we can formulate our implementation of pseudo relabelling as follows. Consider a classifier parameterized by θ. For any synthetic input $\tilde{\mathbf{x}} = \lambda\mathbf{x} + (1-\lambda)\mathbf{x}'$, we still denote its initial synthetic label by $\tilde{\mathbf{y}} = \lambda\mathbf{y} + (1-\lambda)\mathbf{y}'$. In addition, we denote its one-hot pseudo label by $\tilde{\mathbf{y}}^{\text{pl}}$, which is transformed from the model's output of $\tilde{\mathbf{x}}$.

We also need to set up a threshold on the winning scores as mentioned above. It is the criterion that determines whether the synthetic data needs to be relabelled or not. Given $\tilde{\mathbf{x}}$, let's denote the output of the model by $f_\theta(\tilde{\mathbf{x}})$ and denote the final training target of $\tilde{\mathbf{x}}$ by $\tilde{\mathbf{y}}^{\text{pr}}$. Suppose we define $\gamma \in [0, 1]$ as the label combination coefficient, then $\tilde{\mathbf{y}}^{\text{pr}}$ can be obtained as follows:

$$\tilde{\mathbf{y}}^{\text{pr}} := \begin{cases} \tilde{\mathbf{y}} & \max_i f_\theta(\tilde{\mathbf{x}})_i \leq \text{threshold} \\ \gamma \cdot \tilde{\mathbf{y}} + (1-\gamma) \cdot \tilde{\mathbf{y}}^{\text{pl}} & \text{otherwise} \end{cases} \tag{4.1}$$

4.3.2 Experiments Setup

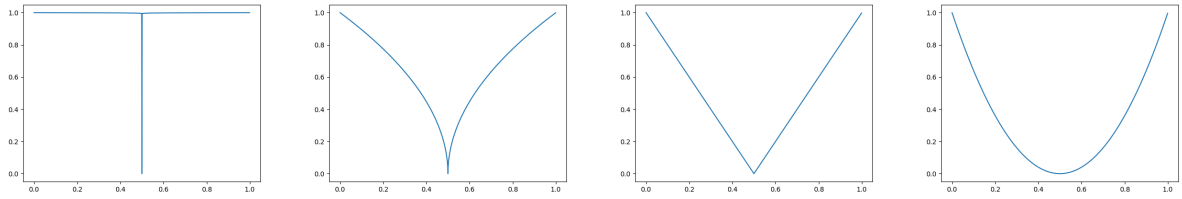
In Equation 4.1, γ can be seen as the reflection of the likelihood of manifold intrusion's occurrence. Based on our belief that this likelihood is correlated with the value of λ, γ needs to be adaptively adjusted depending on λ. To do so, we can define a function $g : [0, 1] \to [0, 1]$ mapping each λ to a real number in the interval $[0, 1]$, *i.e.* $\gamma := g(\lambda)$.

For the synthetic input $\tilde{\mathbf{x}} = \lambda\mathbf{x} + (1 - \lambda)\mathbf{x}'$, we assume that $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}', \mathbf{y}')$ are both identically and independently drawn from the training set. In other words, shifting λ either from 1 to 0.5 or from 0 to 0.5 are overall equivalent. Thus, $g(\lambda)$ can be defined as a symmetric function with respect to $\lambda = 0.5$. Also, notice that $\gamma := g(\lambda)$ ought to be nonincreasing if $\lambda \in [0, 0.5)$. Correspondingly, it is nondecreasing if $\lambda \in (0.5, 1]$.

In this thesis, we propose a form of defining the function $g(\lambda)$ shown as follows:

$$g(\lambda) = \frac{1-\delta}{0.5^\beta}|\lambda - 0.5|^\beta + \delta \tag{4.2}$$

where $\beta \geq 0$ and controls the curvature of the function, and $\delta \in [0, 1]$ and is the function's output at $\lambda = 0.5$. Notice that δ is also the minimal value of $g(\lambda)$. Figure 4.2 shows some examples of $g(\lambda)$ defined by Equation 4.2 with different δ's. Thus, in pseudo relabeling we can treat β, δ and the winning score threshold as the hyperparameters of the algorithm.



(a) $\beta = 0.001$ (b) $\beta = 0.5$ (c) $\beta = 1$ (d) $\beta = 2$

Figure 4.2: Examples of $\gamma = g(\lambda)$ ($\delta = 0$)

4.3.3 Experiments and Results

We first carry out the experiments on the Spiral3-90 dataset shown in Figure 2.1(a). The model we use is a 2-layer MLP in which the width of the hidden layer is 500. To set up the baselines, we first train the model both in the ERM scheme and in the Standard Mixup scheme 3.1 with $\alpha = 0.1$ and 1. The results of the testing accuracies are shown in Table 4.1.

Method	Acc_{test} (%)	p Value (%)
ERM/baseline	83.71±1.29	
Mixup $\alpha = 0.1$	**87.19±0.85**	0.01 ✓
Mixup $\alpha = 1$	69.04±1.29	

Table 4.1: Baselines on Spiral3-90

Method		Acc_{test} (%)	p Value (%)
Mixup $\alpha = 0.1$/baseline		87.19±0.85	
Pseudo Relabelling ($\delta = 0$)			
β	threshold		
2	0.9	86.70±0.85	
2	0.7	84.84±0.68	
2	0.5	85.07±0.71	
1	0.9	86.94±0.83	
1	0.7	85.38±0.55	
1	0.5	85.29±0.65	
0.5	0.9	87.18±0.86	
0.5	0.7	85.88±0.64	
0.5	0.5	85.77±0.62	
0.001	**0.9**	**87.43±0.82**	52.86 ✗
0.001	0.7	87.18±0.89	
0.001	**0.5**	**87.30±0.89**	78.07 ✗

Table 4.2: Pseudo Relabelling on Spiral3-90 ($\alpha = 0.1$)

In Table 4.1, the p value in the 3^{rd} column refers to the Student's t-test result between the testing accuracy in the corresponding row and that of the baseline. The significance levels of the t-tests are set as 0.05 as the conventional way. The results show that Mixup training with $\alpha = 0.1$ provides the best generalization of the trained model, while Mixup with $\alpha = 1$ leads to a even worse performance compared to ERM. This does make sense since Figures 2.9(a) and 2.9(c) show that λ's generated from Beta$(0.1, 0.1)$ tend to be

Method		Acc_{test} (%)	p Value (%)
Mixup $\alpha = 1$/baseline		69.04±1.29	
Pseudo Relabelling ($\delta = 0$)			
β	threshold		
2	**0.9**	**69.19±1.01**	77.55 ✗
2	0.8	68.57±1.49	
2	0.7	67.20±1.38	
1	**0.9**	**69.08±1.22**	94.40 ✗
1	0.7	67.76±1.25	
1	0.5	68.29±0.95	
0.5	**0.9**	**69.19±1.19**	79.00 ✗
0.5	0.7	68.50±1.28	
0.5	**0.5**	**69.94±0.85**	8.20 ✗
0.001	0.9	68.82±1.53	
0.001	0.7	68.82±1.28	
0.001	0.5	68.76±1.11	

Table 4.3: Pseudo Relabelling on Spiral3-90 ($\alpha = 1$)

closer to 0 or 1 compared to those generated from Beta$(1, 1)$, making manifold intrusion less likely to occur.

Assuming that pseudo relabelling does relabel the synthetic data properly, we may expect that it can not only improve the model's generalization under the same setting of α, but can also further improve the best achieved performance of Mixup. In the experiments, we apply pseudo relabelling in Mixup both with $\alpha = 0.1$ and 1, and the results are listed in Tables 4.2 and 4.3. The results show that not only pseudo relabelling fails to further improve the generalization of the Mixup-trained networks, but also the overall performance of Mixup with $\alpha = 1$ is still significantly lower than that of Mixup with $\alpha = 0.1$.

We then perform the experiments on 30% of the CIFAR10 training set without data augmentation. ResNet18 is trained using both ERM and Mixup. In Mixup training we still respectively set $\alpha = 0.1$ and $\alpha = 1$. We first present the results of the training baselines in Table 4.4. Then we apply pseudo relabelling in Mixup training with both $\alpha = 0.1$ and $\alpha = 1$. The results are given in Tables 4.5 and 4.6.

The results in table 4.5 show that when $\alpha = 0.1$, pseudo relabelling is still unable to further improve the performance of Mixup. Differently, Table 4.6 shows that when

$\alpha = 1$, by fine-tuning the algorithm's hyperparameters pseudo relabelling can improve the generalization of the Mixup-trained network. We then conduct the t-test between the best achieved result of pseudo relabelling ($79.87 \pm 0.92\%$) and the result of the Mixup baseline with $\alpha = 0.1$ ($79.08 \pm 1.17\%$). However, the p value of the t-test is 11.05%, indicating that pseudo relabelling still has not significantly improve the overall performance of Mixup.

Method	Acc_{test} (%)	p Value (%)
ERM/baseline	76.10±2.26	
Mixup $\alpha = 0.1$	**79.08±1.17**	0.16 ✓
Mixup $\alpha = 1$	77.25±0.53	

Table 4.4: Baselines on 30% CIFAR10 (Without Data Augmentation)

Method	Acc_{test} (%)	p Value (%)
Mixup $\alpha = 0.1$/baseline	79.08±1.17	
Pseudo Relabelling $\beta = 2, \delta = 0$ threshold		
0.9	78.65±2.29	
0.7	79.06±1.67	
0.5	78.54±2.04	

Table 4.5: Pseudo Relabelling on 30% CIFAR10 (Without Data Augmentation, $\alpha = 0.1$)

Method	Acc_{test} (%)	p Value (%)
Mixup $\alpha = 1$/baseline	77.25±0.53	
Pseudo Relabelling $\beta = 2, \delta = 0$ threshold		
0.9	**77.36±0.55**	65.43 ✗
0.7	**79.79±0.50**	0.01 ✓
0.5	**79.87±0.92**	0.01 ✓

Table 4.6: Pseudo Relabelling on 30% CIFAR10 (Without Data Augmentation, $\alpha = 1$)

4.4 Reference Relabelling

4.4.1 Relabelling via Reference Examples

In ERM training, the generalization gap is generally inevitable due to the finiteness of the training data. Thus, many regularization techniques like data augmentation basically aim to expand the sizes of the training sets.

In Mixup, each synthetic example can be seen as being drawn from the line segment between a pair of the real examples. Then regardless of the spatial structures of the data manifolds, a synthetic example only carries the information of a 1D manifold in the data space, resulting in the lack of the global information. Thus, similar to some of the regularization techniques in ERM, in Mixup we may correct the labels of the synthetic examples by inducing some additional training examples as the reference. In fact, in several works the authors have implied that we can solve manifold intrusion by breaking the locally linear constraints of Mixup. For example, in [15] the authors propose *Nonlinear Mixup*. It indicates that instead of formulating the synthetic examples via linear interpolations as defined in Equation 3.1, we can expand the formulating space to 2D rectangular plates. In fact, *GenLabel* [36] can also be seen as a relabelling algorithm that takes all the training examples as the reference examples to relabel each synthetic example.

In this section, we propose the **reference relabelling** scheme. Consider a synthetic example $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ formulated by $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}', \mathbf{y}')$, reference relabelling additionally draws a finite number of the real examples $(\mathbf{x}^{*(1)}, \mathbf{y}^{*(1)}), (\mathbf{x}^{*(2)}, \mathbf{y}^{*(2)}), \cdots, (\mathbf{x}^{*(k)}, \mathbf{y}^{*(k)})$ from the training set. The statistical information of all these real examples including $(\mathbf{x}, \mathbf{y})$ and $(\mathbf{x}', \mathbf{y}')$ are used to relabel $\tilde{\mathbf{x}}$.

4.4.2 Instance-Wise Hypothesis of Gaussian Mixture Models

With the existence of the additional examples, the linear labelling policy of Mixup is apparently no longer available. Consider a C-class classification task with a training set $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$ and a model whose output is denoted by $f_\theta(\cdot)$. Different from *GenLabel* [36], in reference relabelling we alternatively hypothesize a instance-wise Gaussian generative model with respect to each of the observed real examples (the reference examples and the two interpolating ones).

Let's denote the data space by $\mathcal{X} = \mathbb{R}^{d_0}$, where $d_0 \in \mathbb{Z}^+$ is the dimension of $\mathcal{X}$. Note that although the input features may be in the tensors forms, they can nevertheless

be reshaped and treated as vectors. We hypothesize that each example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ in S produces a class-dependent generative model in the Gaussian form. In other words, if we consider $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ as the only training example observed, then the distribution of the feature variable X in the data space $\mathcal{X}$ can be defined by a d_0-variate Gaussian distribution:

$$X|\mathbf{x}^{(i)} \sim \mathcal{N}(\mathbf{x}^{(i)}, \mathbf{\Sigma}) \tag{4.3}$$

where $\mathbf{\Sigma}$ is the covariance matrix whose shape is $d_0 \times d_0$.

Conventionally in unsupervised learning, the Gaussian generative models need to be trained, where the mean vectors and the covariance matrices are seen as the models' parameters. In our case however, the Gaussian generative model with respect to each training example simply takes the example's input feature as its mean. Additionally, the covariance matrices are also prefixed. Consider $\mathcal{N}(\mathbf{x}^{(i)}, \mathbf{\Sigma})$, we assume that the variables on all its axes have the same variances and are irrelevant with each others. This is because the model $\mathcal{N}(\mathbf{x}^{(i)}, \mathbf{\Sigma})$ is produced without observing any other examples besides $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$. Thus, the covariance matrix $\mathbf{\Sigma}$ is a scalar matrix, *i.e.* a diagonal matrix with the equal diagonal entries:

$$\begin{aligned} \mathbf{\Sigma} &:= \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_{d_0}^2 \end{bmatrix} \\ &:= \mathbf{diag}(\sigma_1^2, \sigma_2^2, \cdots, \sigma_{d_0}^2) \end{aligned} \tag{4.4}$$

where $\sigma_1^2 = \sigma_2^2 = \cdots = \sigma_{d_0}^2 = \sigma^2$. Additionally, we assume that σ remains a prefixed value in the instance-wise Gaussian generative model $\mathcal{N}(\mathbf{x}^{(i)}, \mathbf{\Sigma})$ with respect to each $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in S$.

For simplification, we may denote $\mathcal{N}(\mathbf{x}^{(i)}, \mathbf{\Sigma})$ by $\mathcal{N}^{(i)}$ for $i = 1, 2, \cdots, n$. Now consider a synthetic input feature $\tilde{\mathbf{x}} = \lambda \mathbf{x}^{(i_1)} + (1 - \lambda)\mathbf{x}^{(i_2)}$ where $(\mathbf{x}^{(i_1)}, \mathbf{y}^{(i_1)}), (\mathbf{x}^{(i_2)}, \mathbf{y}^{(i_2)}) \in S$. For any $m \geq 2$, we draw $m-2$ additional examples from S as the reference examples, which we denote by: $\mathbf{x}^{(i_3)}, \mathbf{x}^{(i_4)}, \cdots, \mathbf{x}^{(i_m)}$. Each of the m real examples is related to a instance-wise Gaussian generative model, *i.e.* $X|\mathbf{x}^{(i_j)} \sim \mathcal{N}^{(i_j)}$ for $j = 1, 2, \cdots, m$. Since $\tilde{\mathbf{x}} \in \mathcal{X}$, we have:

$$P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)}) = f_{\mathcal{N}}^{(i_j)}(\tilde{\mathbf{x}}) \tag{4.5}$$

where $f_{\mathcal{N}}^{(i_j)}(\cdot)$ denotes the PDF of $\mathcal{N}^{(i_j)}$. Equation 4.5 represents the likelihood of $\tilde{\mathbf{x}}$ conditioning on it being generated from $\mathcal{N}^{(i_j)}$.

Assuming that the distributions $\mathcal{N}^{(i_j)}$ for $j = 1, 2, \cdots, m$ have equal prior probabilities:

$$P(X \sim \mathcal{N}^{(i_1)}) = P(X \sim \mathcal{N}^{(i_2)}) = \cdots = P(X \sim \mathcal{N}^{(i_m)}) = \frac{1}{m} \tag{4.6}$$

then by Bayesian Theorem, we have:

$$\begin{aligned} P_{\mathcal{N}|X}(X \sim \mathcal{N}^{(i_j)}|\tilde{\mathbf{x}}) &= \frac{P_{X,\mathcal{N}}(\tilde{\mathbf{x}}, X \sim \mathcal{N}^{(i_j)})}{P_X(\tilde{\mathbf{x}})} \\ &= \frac{P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)}) P_{\mathcal{N}}(X \sim \mathcal{N}^{(i_j)})}{P_X(\tilde{\mathbf{x}})} \\ &= \frac{P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)})}{P_X(\tilde{\mathbf{x}})} \cdot \frac{1}{m} \end{aligned} \tag{4.7}$$

for $j = 1, 2, \cdots, m$, which defines the posterior probability of X following the distribution $\mathcal{N}^{(i_j)}$.

We can combine $\mathcal{N}^{(i_1)}, \mathcal{N}^{(i_2)}, \cdots, \mathcal{N}^{(i_m)}$ into a Gaussian mixture model to help to relabel $\tilde{\mathbf{x}}$. In this case, we have:

$$\sum_{j=1}^{m} P_{\mathcal{N}|X}(X \sim \mathcal{N}^{(i_j)}|\tilde{\mathbf{x}}) = 1 \tag{4.8}$$

which indicates:

$$\begin{aligned} &\sum_{j=1}^{m} \frac{P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)})}{P_X(\tilde{\mathbf{x}})} \cdot \frac{1}{m} \\ =&\frac{1}{m} \sum_{j=1}^{m} \frac{P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)})}{P_X(\tilde{\mathbf{x}})} \\ =&1 \end{aligned} \tag{4.9}$$

Then we have:

$$P_X(\tilde{\mathbf{x}}) = \frac{1}{m} \sum_{j=1}^{m} P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)}) \tag{4.10}$$

Let's denote $P_{\mathcal{N}|X}(X \sim \mathcal{N}^{(i_j)}|\tilde{\mathbf{x}})$ by $\rho^{(i_j)}$. From Equations 4.5, 4.7 and 4.10, we have:

$$\begin{aligned}\rho^{(i_j)} :&= P_{\mathcal{N}|X}(X \sim \mathcal{N}^{(i_j)}|\tilde{\mathbf{x}}) \\ &= \frac{P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_j)})}{\sum_{k=1}^{m} P_{X|\mathcal{N}}(\tilde{\mathbf{x}}|X \sim \mathcal{N}^{(i_k)})} \\ &= \frac{f_{\mathcal{N}}^{(i_j)}(\tilde{\mathbf{x}})}{\sum_{k=1}^{m} f_{\mathcal{N}}^{(i_k)}(\tilde{\mathbf{x}})}\end{aligned} \tag{4.11}$$

for $j = 1, \cdots, m$.

Notice that $\tilde{\mathbf{x}}$ being generated from $\mathcal{N}^{(i_j)}$ is the sufficient condition of it being in the same class with $\mathbf{x}^{(i_j)}$. Let's denote the training targets of $\tilde{\mathbf{x}}$ reassigned by reference relabelling by $\tilde{\mathbf{y}}^{\text{ref}}$. Therefore, $\tilde{\mathbf{y}}^{\text{ref}}$ can be defined as the average of $\mathbf{y}^{(i_1)}, \mathbf{y}^{(i_2)}, \cdots, \mathbf{y}^{(i_m)}$ weighted by $\rho^{(i_1)}, \rho^{(i_2)}, \cdots, \rho^{(i_m)}$:

$$\tilde{\mathbf{y}}^{\text{ref}} = \sum_{j=1}^{m} \rho^{(i_j)} \mathbf{y}^{(i_j)} \tag{4.12}$$

Note that $\tilde{\mathbf{y}}^{\text{ref}} \in \mathbb{R}^C$ and is capable of serving as a training target in the classification task, since it can validly represent a discrete probability distribution:

$$\begin{aligned}\sum_{k=1}^{C} \tilde{y}_k^{\text{ref}} &= \sum_{j=1}^{m} \left(\rho^{(i_j)} \sum_{k=1}^{C} y_k^{(i_j)} \right) \\ &= \sum_{j=1}^{m} \rho^{(i_j)} \\ &= \sum_{j=1}^{m} P_{\mathcal{N}|X}(X \sim \mathcal{N}^{(i_j)}|\tilde{\mathbf{x}}) \\ &= 1\end{aligned} \tag{4.13}$$

since $\mathbf{y}^{(i_1)}, \mathbf{y}^{(i_2)}, \cdots, \mathbf{y}^{(i_m)}$ are all one-hot labels and Equation 4.8 holds true.

In addition, recall the covariance matrix defined in Equation 4.4. If $\sigma > 0$, then the inverse

of the matrix is simply:

$$\begin{aligned}\boldsymbol{\Sigma}^{-1} &= \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_{d_0}^2} \end{bmatrix} \\ &= \mathbf{diag}\left(\frac{1}{\sigma_1^2}, \frac{1}{\sigma_2^2}, \cdots, \frac{1}{\sigma_{d_0}^2}\right)\end{aligned} \tag{4.14}$$

where $\frac{1}{\sigma_1^2} = \frac{1}{\sigma_2^2} = \cdots = \frac{1}{\sigma_{d_0}^2} = \frac{1}{\sigma^2}$. Thus, using Equation 2.29, the PDF of $\mathcal{N}^{(i_j)}$ can be computed as follows:

$$\begin{aligned} f_{\mathcal{N}}^{(i_j)}(\tilde{\mathbf{x}}) &= \frac{1}{\sqrt{(2\pi)^{d_0}|\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)})^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)})\right) \\ &= \frac{1}{\sqrt{(2\pi)^{d_0}\sigma^{2d_0}}} \exp\left(-\frac{1}{2\sigma^2}(\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)})^{\mathrm{T}}(\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)})\right) \\ &= \frac{1}{\sqrt{(2\pi\sigma^2)^{d_0}}} \exp\left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)}\|_2^2}{2\sigma^2}\right) \end{aligned} \tag{4.15}$$

Therefore, from equation 4.11 we have:

$$\begin{aligned} \rho^{(i_j)} &= \frac{f_{\mathcal{N}}^{(i_j)}(\tilde{\mathbf{x}})}{\sum_{k=1}^{m} f_{\mathcal{N}}^{(i_k)}(\tilde{\mathbf{x}})} \\ &= \frac{\frac{1}{\sqrt{(2\pi\sigma^2)^{d_0}}} \exp\left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)}\|_2^2}{2\sigma^2}\right)}{\sum_{k=1}^{m} \frac{1}{\sqrt{(2\pi\sigma^2)^{d_0}}} \exp\left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_k)}\|_2^2}{2\sigma^2}\right)} \\ &= \frac{\exp\left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)}\|_2^2}{2\sigma^2}\right)}{\sum_{k=1}^{m} \exp\left(-\frac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_k)}\|_2^2}{2\sigma^2}\right)} \end{aligned} \tag{4.16}$$

Which completes the computation of $\tilde{\mathbf{y}}^{\mathrm{ref}}$ defined in Equation 4.12.

4.4.3 Experiments and Results

In reference relabelling, we denote the number of the reference examples with respect to each synthetic example by "#". It is also one of the hyperparameters of the algorithm. We first perform the experiments on both the original and the 30% of the CIFAR10 training set. ResNet18 is trained, and both data augmentation and weight decay are used. The results are shown in Table 4.7 and Table 4.8 respectively. The results show that in neither case has reference relabelling further improved the generalization performance of the Mixup-trained model.

Method		Acc_{test} (%)	p Value (%)
Mixup $\alpha = 1$/baseline		91.60±0.30	
Reference Relabelling			
#	σ		
1	10	90.43±0.23	
1	60	91.47±0.14	
1	100	90.66±0.16	
2	10	90.41±0.23	
2	60	91.21±0.17	
2	100	90.46±0.11	
3	10	90.31±0.41	
3	60	91.11±0.30	
3	100	90.16±0.31	

Table 4.7: Reference Relabelling on 30% CIFAR10 (With Data Augmentation, $\alpha = 1$)

Note that by increasing #, the space complexity of the algorithm's implementation is multiplied. Thus, in our experiments mentioned above, we assign at most three reference examples for each synthetic example. Considering the size of the training set and how complex the data statistical structure is in it, such a relatively small amount of the reference examples may cause a large variance within different sampling trails. That is to say, one single prefixed σ may not be suitable globally. Thus, we define another hyperparameter β in reference relabelling. Consider still a synthetic input feature $\tilde{\mathbf{x}} = \lambda \mathbf{x}^{(i_1)} + (1-\lambda)\mathbf{x}^{(i_2)}$ and $m-2$ reference examples $\mathbf{x}^{(i_3)}, \mathbf{x}^{(i_4)}, \cdots, \mathbf{x}^{(i_m)}$. σ is defined as follows:

$$\sigma := \frac{\|\mathbf{x}^{(i_1)} - \mathbf{x}^{(i_2)}\|_2}{\beta} \tag{4.17}$$

Method		Acc_{test} (%)	p Value (%)
Mixup $\alpha = 1$/baseline		95.95±0.15	
Reference Relabelling			
#	σ		
1	10	95.34±0.08	
1	60	95.76±0.03	
1	100	95.23±0.09	
2	10	95.28±0.07	
2	60	95.74±0.16	
2	100	95.15±0.11	
3	10	95.29±0.15	
3	60	95.64±0.18	
3	100	95.03±0.16	

Table 4.8: Reference Relabelling on 100% CIFAR10 (With Data Augmentation, $\alpha = 1$)

In this case, β is prefixed so that the ratio of the euclidean distance between the pair of the formulating examples to σ is a constant value globally:

$$\frac{\|\mathbf{x}^{(i_1)} - \mathbf{x}^{(i_2)}\|_2}{\sigma} := \beta \tag{4.18}$$

Thus, $\rho^{(i_j)}$ is alternatively defined as follows:

$$\begin{aligned} \rho^{(i_j)} &= \frac{\exp\left(-\dfrac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)}\|_2^2}{2\sigma^2}\right)}{\sum_{k=1}^{m} \exp\left(-\dfrac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_k)}\|_2^2}{2\sigma^2}\right)} \\ &= \frac{\exp\left(-\dfrac{\beta^2}{2} \cdot \dfrac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_j)}\|_2^2}{\|\mathbf{x}^{(i_1)} - \mathbf{x}^{(i_2)}\|_2^2}\right)}{\sum_{k=1}^{m} \exp\left(-\dfrac{\beta^2}{2} \cdot \dfrac{\|\tilde{\mathbf{x}} - \mathbf{x}^{(i_k)}\|_2^2}{\|\mathbf{x}^{(i_1)} - \mathbf{x}^{(i_2)}\|_2^2}\right)} \end{aligned} \tag{4.19}$$

For any $\tilde{\mathbf{x}} = \lambda\mathbf{x} + (1-\lambda)\mathbf{x}'$ with λ fixed, we believe that the farther the distance between $\mathbf{x}$ and $\mathbf{x}'$ is, the more likely manifold intrusion is to occur. This is due to the fact that $\tilde{\mathbf{x}}$ is located farther away from the data manifolds where the two real examples belong to.

Thus, by adopting Equation 4.19 with β prefixed, we see that the farther apart $\mathbf{x}^{i_1}$ and $\mathbf{x}^{i_2}$ are, the higher their labels' corresponding weights in $\tilde{\mathbf{y}}^{\text{ref}}$ are. This is in accordance with the belief mentioned above, and it is reasonable that relabelling is more needed by the synthetic examples with respect to which manifold intrusion is more likely to occur. For the β-fixed reference relabelling algorithm, we perform the experiments on Spiral3-90 dataset. The Mixup training baselines are the same as those listed in Table 4.1. The results with $\alpha = 0.1$ and $\alpha = 1$ are given in Table 4.9 and Table 4.10 respectively. Table 4.10 shows that under the setting $\alpha = 0.1$ of Mixup, reference relabelling overwhelms the corresponding baseline when $\# = 1$ and $\beta = 3$. We then conduct the t-test between its result ($71.13 \pm 0.81\%$) and the best achieved result of Mixup ($87.19 \pm 0.85\%$). However, the p value is 0.01, indicating that the result of reference relabelling is still significantly lower than the best result of standard Mixup.

Method	Acc_{test} (%)	p Value (%)
Mixup $\alpha = 0.1$/baseline	87.19±0.85	
Reference Relabelling (#=1)		
β		
1	84.00±0.82	
2	87.13±0.69	
3	87.07±1.06	
10	86.80±0.81	

Table 4.9: Reference Relabelling on Spiral3-90 ($\# = 1$, $\alpha = 0.1$)

Method	Acc_{test} (%)	p Value (%)
Mixup $\alpha = 1$/baseline	69.04±1.29	
Reference Relabelling		
β		
1	60.53±1.31	
2	68.07±0.83	
3	**71.13±0.81**	0.04 ✓
10	**69.80±0.98**	15.52 ✗

Table 4.10: Reference Relabelling on Spiral3-90 ($\# = 1$, $\alpha = 1$)

4.5 Cautious Mixing Strategy

To verify whether the two of our proposed relabelling algorithms can indeed provide better training targets for the synthetic data, we have also conducted extensive experiments on Spiral dataset. We first well train a 3-layer MLP on Spiral3-6000 with ERM and use it to predict the ground-truth for any input. Then for some synthetic examples, we generate three types of the training targets for them based on the linear interpolation policy, pseudo relabelling and reference relabelling respectively. Finally, we compute the differences between the ground-truth of these synthetic examples and the three types of their training targets using the definition of MSE. The results are given in Table 4.11.

Labelling Method	MSE w.r.t. Ground-Truth ($\times 10^{-2}$)
Standard Mixup/baseline	23.3413
Pseudo Relabelling	23.3018
Reference Relabelling	23.0705

Table 4.11: Evaluating the Relabelling Algorithms on Spiral3-90

We can see that the two relabelling algorithms can both slightly make the training targets of the synthetic data closer to its ground-truth, which indicates that they are potentially effective. However, from the experiments results provided in Section 4.3.3 and Section 4.4.3, we have seen that so far the two of our proposed relabelling algorithms don't seem to be promising in either actually solving manifold intrusion or further improving the generalization performance on the basis of Mixup. Moreover, from our experience of tuning the hyperparameters of the algorithms, it stands to reason that even if the algorithms can perform as expected, the processes of implementing them can be potentially complicated. We have also noticed this issue in the works of some other regarding the relabelling algorithms. For instance, although *GenLabel* [36] has been shown to be one of the most successful relabelling algorithms to date, it nevertheless requires training an entire additional generative network, the workload of which may be no less than training the classifier itself.

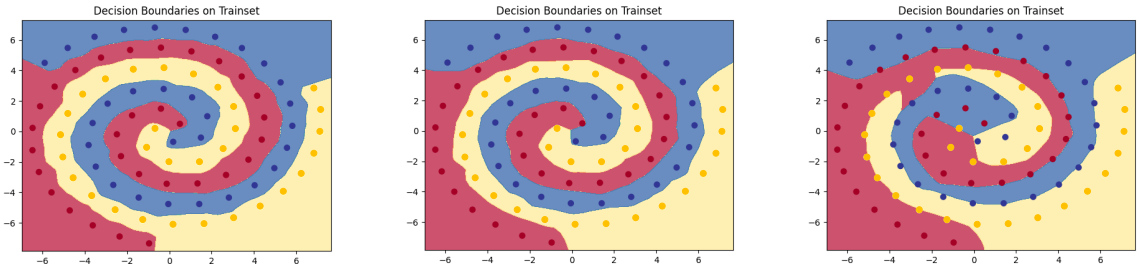
Besides the relabelling strategy, we summarize some of the other relabelling schemes as the **cautious mixing** strategy. The objective of this strategy is to avoid formulating the synthetic examples that are highly likely for manifold intrusion to occur by cautiously designing the mixing policy. To do so, one of the most straightforward ways is to tune the values of λ. As mentioned in Section 4.3.1, the synthetic data formulated via smaller λ is located closer to the manifolds of the real data, thus manifold intrusion is less likely to

occur. To perform cautious mixing in practice from this point of view, we can alternatively cautiously control the value of α so that the λ's generated by Beta(α, α) can be overall close to 0 or 1. In fact, is has been indicated in various works that Mixup training with a smaller choice of α tend to provide better generalization performance [7, 8, 14, 38, 44].

Derived from this belief, several works regarding the cautious mixing strategy has proposed the algorithms to adjust the value of α properly. For example, in [16] the authors propose the *AdaMixUp* scheme, which trains an additional network that learns the optimal choices of α for the training set. The authors also empirically show their improvement on the generalization performance of the Mixup-trained networks using their proposed algorithm.

The success and the implications in the aforementioned articles have conveyed a message: although the degree of freedom to formulate the synthetic data is reduced, avoiding the emergence of the intrusive synthetic examples by fine-tuning α is still promising to prevent manifold intrusion. In fact, the results of some of our experiments have also shown the similar patterns. For example, Table 4.4 shows us that when training the networks on the 30% CIFAR10 without data augmentation, a smaller choice of α provides the Mixup-trained model with better generalization, which may even surpass the performance of the ERM baseline. Additionally, on the Spiral3-90 dataset this tendency also holds true, which can be seen from Table 4.1.

To verify the message mentione above, we have also set up another series of the training experiments on the uniform Spiral3-90 dataset as shown in Figure 2.1(b). We plot the learned decision boundaries of the models on the training set, visualizing the classification functions fitted by the trained models. The results are illustrated in Figure 4.3.



(a) ERM (b) Mixup $\alpha = 0.1$ (c) Mixup $\alpha = 1$

Figure 4.3: Decision Boundaries on Uniform Spiral3-90

First let's compare Figure 4.3(a) and Figure 4.3(c), we see that in the latter case Mixup fails to generalize the model as expected, and instead it has caused the underfitting issue.

This is in accordance with our aforementioned understanding about manifold intrusion that larger choices of α are more likely to create the intrusive synthetic examples.

From Figures 4.3(a) and 4.3(b), we can see that in the area sandwiched by the adjunct spiral curves of different classes, both the ERM-trained model and the Mixup-trained model can fit the decision boundaries around the median curves between the real data manifolds. However, Mixup with $\alpha = 0.1$ can lead to smoother decision boundaries than the ERM baseline. Also, in the areas around the outer ends of the spiral curves, the decision boundaries of the ERM-trained model is fitted closely to the "tail" data points, while those of the Mixup-trained model with $\alpha = 0.1$ are looser. These two behaviors indicate that on the uniform Spiral3-90 dataset, Mixup with $\alpha = 0.1$ can exempt the trained model's performance from being hurt by manifold intrusion to some extent. It can also lead the model to not fit too close to the training data.

In addition, we have designed another toy dataset named the "radiate" dataset, which is shown in Figure 4.4. It is a 3-class dataset of 2-dimensional data points. We train a 2-layer MLP on the dataset both using ERM and Mixup. The decision boundaries fitted by the trained models are presented in Figure 4.5. From Figures 4.5(a), we see that the ERM-trained model tends to fit the decision boundaries closely to the training data. In Figure 4.5(b), the Mixup-trained model with $\alpha = 100$ ends up with the undeerfitting issue. Notice that via intuitive observation of the radiate dataset, we believe that manifold intrusion is less likely to occur compared to the Spiral datasets. Thus, the radiate dataset may allow a larger degree of freedom to set α. In fact, Figures 4.5(d) and 4.5(c) show us that both the settings $\alpha = 0.1$ and $\alpha = 1$ enable the Mixup-trained models to fit the smooth and loose decision boundaries.

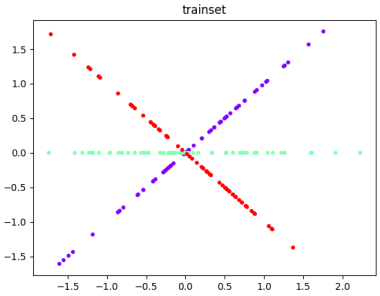
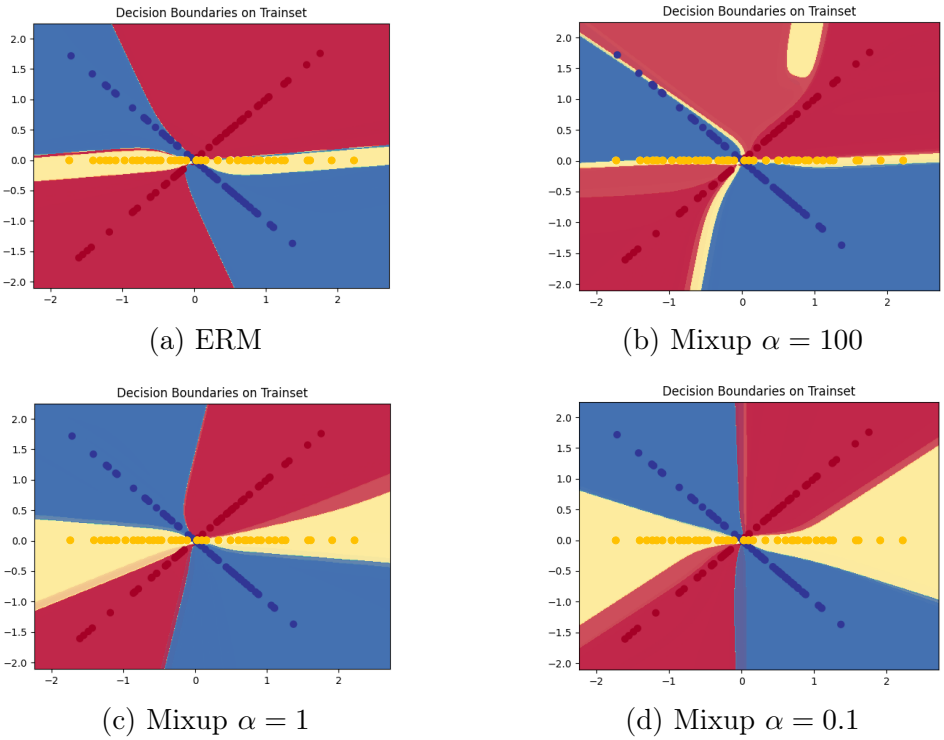


Figure 4.4: Radiate Dataset



(a) ERM

(b) Mixup $\alpha = 100$

(c) Mixup $\alpha = 1$

(d) Mixup $\alpha = 0.1$

Figure 4.5: Decision Boundaries on Radiate Dataset

4.6 Concluding Remarks

We have first discussed a primary understanding of why manifold intrusion hurts generalization: manifold intrusion brings about the conflicts between the synthetic labels and the ground-truth labels of the synthetic data. We then summarize the majority of the proposed arts about preventing manifold intrusion into two categories: the relabelling strategy and the cautious mixing strategy. We also show and explain in details two of our proposed relabelling algorithms. Eventually, we compare the arts of the two strategies from the perspectives of both their effectiveness and their efficiency. We believe that compared to the relabelling strategy, the cautious mixing strategy is overall more promising.

In conclusion, although the relabelling strategy may still have potential prospects, we nevertheless suggest that the researchers focus more on the cautious mixing strategy in the works about solving manifold intrusion.

Chapter 5

Over-Training

In this chapter, we present our observation of a phenomenon in Mixup training: the best achieved generalization performance of a Mixup-trained model may decrease as the training epochs are increased. We have verified this behavior on several models and benchmark datasets. We then provide our explanation of this issue. First, we show that in Mixup training the synthetic data may contain label noise induced by the random interpolations of the training data. Then we demonstrate that over-training the model with Mixup may result in its overfitting the noise synthetic data, which we believe is the cause of the U-shaped generalization curve as shown in Figure 1.1. In addition, we have also performed extensive experiments to verify our proposed explanation.

5.1 Related Works

Since the thought-provoking work of [43], in which they show that neural networks are able to fit the data with random labels, the generalization behavior on the corrupted label datasets has been widely investigated by many researchers [5, 12, 29, 30, 40]. Specifically, [5] observes that the neural networks learns the clean patterns first before fitting to the data with random labels. This is further explained by [3] in which the authors demonstrate that under the overparameterization regime, the convergence of the loss depends on the projections of the data labels on the eigenvectors of some Gram matrices, and these projections are different between the true labels and the random labels. As a parallel research line, during training a deep neural network, an epoch-wise double descent behavior of the testing loss is first observed in [32]. The theoretical understanding of this phenomena is still limited, and only a few works try to explain it such as [19, 34, 37]. Some parts of

the analyses in these works are similar to an earlier work of [1], which also inspires our theoretical explanation of the U-sharped curve of Mixup training. Robust overfitting is also another related research line, which is found by [35]. In particular, robust overfitting is a phenomena in adversarial training that the robust accuracy may first increase then decrease after a longer training time. [9] shows that robust overfitting is deemed to the early part of epoch-wise double descent due to the *implicit label noise* induced by adversarial training. Since Mixup training has been connected to adversarial training or adversarial robustness in the previous works [2, 46], the work of [9] indeed motivates us to study the label noise induced by Mixup training.

5.2 Lower Bound of Mixup Loss

We first provide the lower bound of the ERM empirical loss. Consider a C-class classification task with a training set $S = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{n}$. Let $f_\theta(\cdot)$ denote the model's output and let $\ell(\cdot)$ refer to the cross-entropy loss as defined in Equation 2.11. For any example $(\mathbf{x}, \mathbf{y}) \in S$, the lower bound of its corresponding loss is given as follows:

$$\begin{aligned}
\ell(\theta, \mathbf{x}, \mathbf{y}) &= -\sum_{i=1}^{C} y_i \log \big(f_\theta(\mathbf{x})_i\big) \\
&= -\sum_{i=1}^{C} y_i \log \left(\frac{f_\theta(\mathbf{x})_i}{y_i} y_i\right) \\
&= -\sum_{i=1}^{C} y_i \log \frac{f_\theta(\mathbf{x})_i}{y_i} - \sum_{i=1}^{C} y_i \log y_i \\
&= D_{\mathrm{KL}}\big(\mathbf{y} \| f_\theta(\mathbf{x})\big) + \mathcal{H}(\mathbf{y}) \\
&\geq \mathcal{H}(\mathbf{y})
\end{aligned} \tag{5.1}$$

where the equality holds if and only if $f_\theta(\mathbf{x}) = \mathbf{y}$. Since in S each label $\mathbf{y}^{(i)}$ is a one-hot label, according to the definition of entropy given by Equation 2.31, we have:

$$\begin{aligned}
\mathcal{H}(\mathbf{y}) &= -\sum_{i=1}^{C} y_i \log y_i \\
&= 0
\end{aligned} \tag{5.2}$$

Thus, the lower bound of the ERM empirical loss is given as follows:

$$\begin{aligned}\hat{R}_S(\theta) &= \frac{1}{|S|}\sum_{(\mathbf{x},\mathbf{y})\in S}\ell(\theta,\mathbf{x},\mathbf{y}) \\ &\geq 0\end{aligned} \tag{5.3}$$

The equality holds if $f_\theta(\mathbf{x}^{(i)}) = \mathbf{y}^{(i)}$ holds true constantly for $i = 1, 2, \cdots, n$.

Then we provide the lower bound of the Mixup target loss defined by Equation 3.4. For a given λ and a pair of the examples $(\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}') \in S$, suppose $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = (\lambda\mathbf{x}+(1-\lambda)\mathbf{x}', \lambda\mathbf{y}+(1-\lambda)\mathbf{y}')$ is a cross-class-mixed synthetic example, *i.e.* $\mathbf{y} \neq \mathbf{y}'$. Then via Equation 5.1, the lower bound of the cross-entropy loss of $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ is given as follows:

$$\begin{aligned}\ell(\theta,\tilde{\mathbf{x}},\tilde{\mathbf{y}}) &\geq \mathcal{H}(\tilde{\mathbf{y}}) \\ &= -\sum_{i=1}^{C}\tilde{y}_i\log\tilde{y}_i \\ &= -\big(\lambda\log\lambda + (1-\lambda)\log(1-\lambda)\big)\end{aligned} \tag{5.4}$$

Recall the definition of the Mixup target loss, we can exchange the operations of computing the expectation and computing the empirical average in Equation 3.4 so that the definition of $\hat{R}_{\widetilde{S}_\lambda}(\theta, \alpha)$ can be rewritten as follows:

$$\hat{R}_{\widetilde{S}_\lambda}(\theta,\alpha) := \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbb{E}_\lambda\ell(\theta,\lambda\mathbf{x}^{(i)}+(1-\lambda)\mathbf{x}^{(j)},\lambda\mathbf{y}^{(i)}+(1-\lambda)\mathbf{y}^{(j)}) \tag{5.5}$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$. In practice, a common choice of α is 1. Note that in this case, $\text{Beta}(1, 1)$ is simply the uniform distribution defined on $[0, 1]$, which we denote by $U(0, 1)$.

Then the lower bound of $\mathbb{E}_\lambda \ell(\theta, \tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ is given as follows:

$$\begin{aligned}
\mathbb{E}_\lambda \ell(\theta, \tilde{\mathbf{x}}, \tilde{\mathbf{y}}) &\geq - \mathop{\mathbb{E}}_{\lambda \sim U(0,1)} \big(\lambda \log \lambda + (1-\lambda)\log(1-\lambda)\big) \\
&= -\int_0^1 \lambda \log \lambda + (1-\lambda)\log(1-\lambda)\ d\lambda \\
&= -2\int_0^1 \lambda \log \lambda\ d\lambda \\
&= -2\left(\log \lambda \int_0^1 \lambda\ d\lambda - \int_0^1 \frac{1}{\lambda}\left(\int_0^1 \lambda\ d\lambda\right) d\lambda\right) \\
&= -2\left(\frac{\lambda^2 \log \lambda}{2} - \frac{\lambda^2}{4}\right)\Bigg|_0^1 \\
&= 0.5
\end{aligned} \tag{5.6}$$

Note that if $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ is formulated via in-class mixing, *i.e.* $\mathbf{y} = \mathbf{y}'$, then the synthetic label $\tilde{\mathbf{y}}$ is still a one-hot label. In this case the lower bound of the synthetic example's loss is 0 regardless of λ according to Equation 5.2. Suppose S is balanced, namely the numbers of the training examples in all the C classes are identical, then the in-class mixing occurs with probability $\frac{1}{C}$. Therefore, for any given training set S that is balanced, the overall lower bound of the Mixup target loss is given as follows:

$$\begin{aligned}
\hat{R}_{\widetilde{S}_\lambda}(\theta, \alpha) &\geq \frac{1}{C} \times 0 + \frac{C-1}{C} \times 0.5 \\
&= \frac{C-1}{2C}
\end{aligned} \tag{5.7}$$

The equality holds if $f_\theta(\tilde{\mathbf{x}}) = \tilde{\mathbf{y}}$ holds true for each $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) \in \widetilde{S}$.

The lower bound $\frac{C-1}{2C}$ provided in Equation 5.7 allows us to make sense of the Mixup loss during training. For example, for a 10-class classification task, the bound has value 0.45. Then only when the Mixup loss approaches this value may we conclude that the model parameter is near an optimum (assuming the model has sufficient capacity).

5.3 Empirical Observations

We here empirically show that over-training with Mixup may hurt the networks' generalization.

We conduct experiments using CIFAR10, CIFAR100 and SVHN. For CIFAR10 and SVHN, we adopt both the original training set and a balanced subset of it containing 30% of the training data. For CIFAR100, we only use the original training set, since downsampling CIFAR100 appears to result in the high variances in the testing performance baselines. We train the ResNet networks on the three datasets using both ERM and Mixup while adopting SGD with weight decay. No data augmentation is used.

In each training trial, we train the network for in total a fixed number of epochs. We record the minimal training loss achieved by the network during the training process, and we also record the network's testing accuracy of the epoch at which the minimal training loss is achieved. Local loss landscape (where "loss" refers to the empirical risk defined using the real data) around the found solution at the aforementioned epoch is visualized in 2D following [27]. We gradually increase the total number of the training epochs in different trials of training so as to gradually over-train the network.

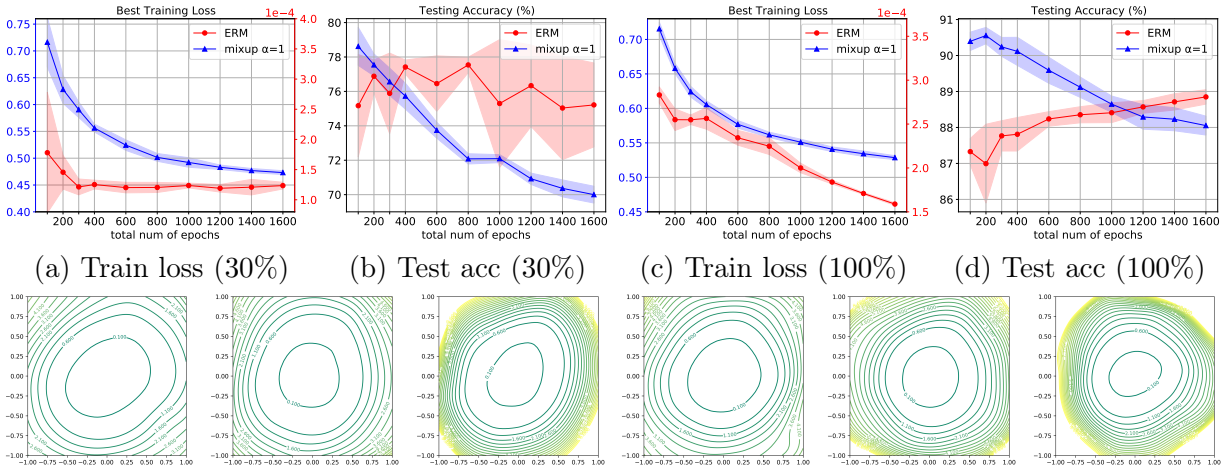
We repeat each training trial for 10 times (using 10 different random seeds) and we average the recorded training losses and testing accuracies. For example, Figure 1.1(a) illustrates the results of training ResNet18 on 100% CIFAR10 data. Each point represents the average of the recorded training losses in a training trial, and the corresponding label on the horizontal axis represents the total number of epochs of that trial. The width of the shade beside each point reflects the deviation of the recorded results in the corresponding trial.

5.3.1 Results

ResNet18 is used for the CIFAR10 and SVHN datasets.

Training is performed for up to 1600 epochs for CIFAR10, and the results for CIFAR10 are shown in Figure 5.1. For both the 30% dataset and the full dataset, we see clearly that after some number of epochs (*e.g.* epoch 200 for the full dataset), the testing accuracy of the Mixup-trained network starts decreasing, and this trend continues. This confirms that over-training with Mixup hurts the network's generalization. One would observe a U-shaped curve, as shown in Figure 1.1 (right), if we were to plot the testing errors and include the results from earlier epochs. Notably, this phenomenon is not observed in ERM. We also found that over-training with Mixup tends to force the network to learn a solution located at a sharper local minima on the loss landscape, a phenomenon correlated with degraded generalization performance [20, 21].

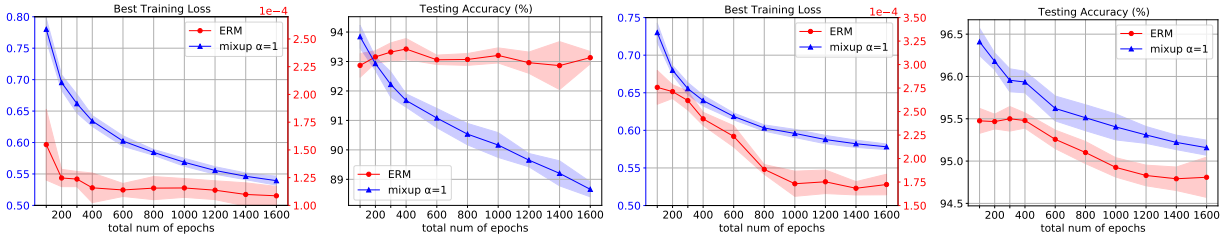
Training is performed for up to 1000 epochs for SVHN, and the results are presented in Figure 5.2. Mixup exhibits a similar phenomenon as it does for CIFAR10. What differs



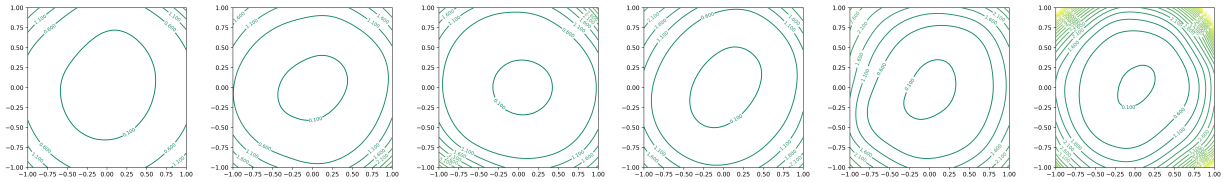
(a) Train loss (30%) (b) Test acc (30%) (c) Train loss (100%) (d) Test acc (100%)

(e) 200 epochs (f) 400 epochs (g) 800 epochs (h) 200 epochs (i) 400 epochs (j) 800 epochs

Figure 5.1: Results of training ResNet18 on CIFAR10 training set (100% data and 30% data) without data augmentation. Top row: training loss and test accuracy for ERM (red) and Mixup (blue). Bottom row: loss landscapes of the Mixup-trained ResNet18 at various training epochs; (e),(f),(g): 30% CIFAR10 dataset; (h),(i),(j): 100% CIFAR10 dataset.



(a) Train loss (30%) (b) Test acc (30%) (c) Train loss (100%) (d) Test acc (100%)

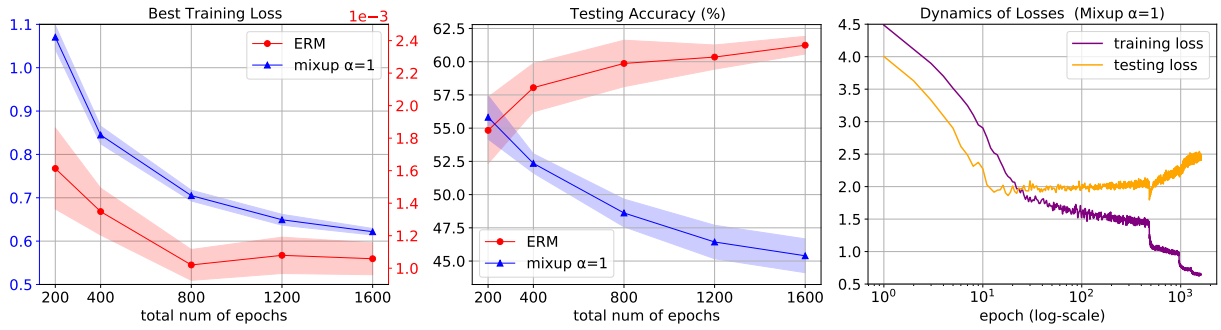


(e) 200 epochs (f) 400 epochs (g) 800 epochs (h) 200 epochs (i) 400 epochs (j) 800 epochs

Figure 5.2: Results of training ResNet18 on SVHN training set (100% data and 30% data) without data augmentation. Top raw: training loss and testing accuracy for ERM (red) and Mixup (blue). Bottom raw: loss landscape of the Mixup-trained ResNet18 at various training epochs: the left three figures are for the 30% SVHN dataset, and the right three are for the full SVHN dataset.

notably is that over-training with ERM on the original SVHN training set appears to also lead to the worsened testing accuracy. However, this does not occur on the 30% SVHN training set. This might be related to the epoch-wise double descent behavior of ERM training. That is, when over-training ResNet18 on the whole training set with a total of 1000 epochs, the network is still in the first stage of over-fitting the training data, while when over-training the network on 30% of the training set, the network learns faster on the training data due to the smaller sample size, thus it passes the turning point of the double descent curve earlier.

ResNet34 is used for the more challenging task on CIFAR100. This choice allows Mixup training to drive its loss to lower values, closer to the lower bound given in Equation 5.7. Training is performed for up to 1600 epochs. The results are plotted in Figure 5.3. The results again confirm that over-training with Mixup hurts the generalization capability of the learned models. A U-shaped testing loss curve (obtained from a single trial) is also observed in Figure 5.3(c).



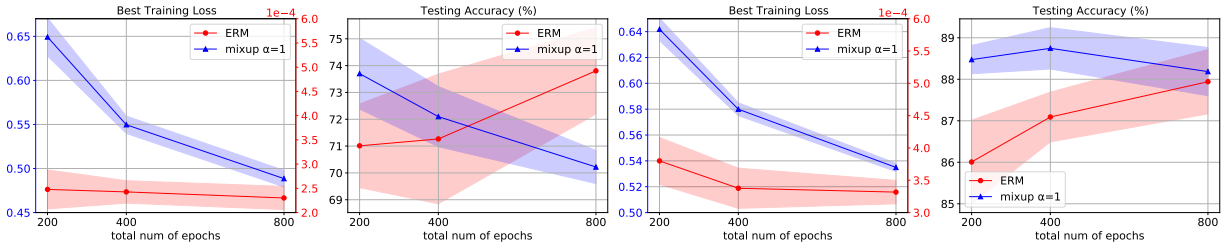
(a) Train loss (100%) (b) Test acc (100%) (c) U-shaped curve

Figure 5.3: Observations of over-training ResNet34 on CIFAR100 without data augmentation. Figure 5.3(c) show how the training loss and the testing loss change during a single trial of over-training ResNet34 on CIFAR100 with Mixup. A remarkable U-shaped curve of the testing loss is observed.

Besides CIFAR100, ResNet34 is also used for the CIFAR10 and the SVHN datasets.

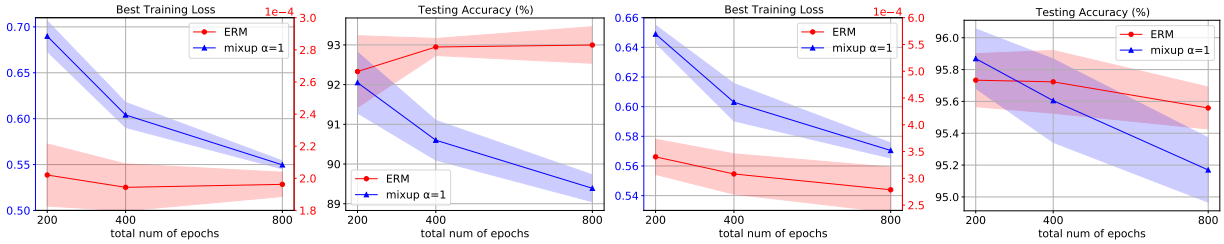
Training is performed on both CIFAR10 and SVHN for in total 200, 400 and 800 epochs respectively. The results for CIFAR10 are shown in Figure 5.4. For both the 30% dataset and the original dataset, Mixup exhibits a similar phenomenon as it does in training ResNet18 on CIFAR10. The difference is that over-training ResNet34 with ERM let the testing accuracy gradually increase on both the 30% dataset and the original dataset.

The results for SVHN are shown in Figure 5.5. These results are also in accordance with those of training ResNet18 on both 30% and 100% of the SVHN dataset.



(a) Train loss (30%) (b) Test acc (30%) (c) Train loss (100%) (d) Test acc (100%)

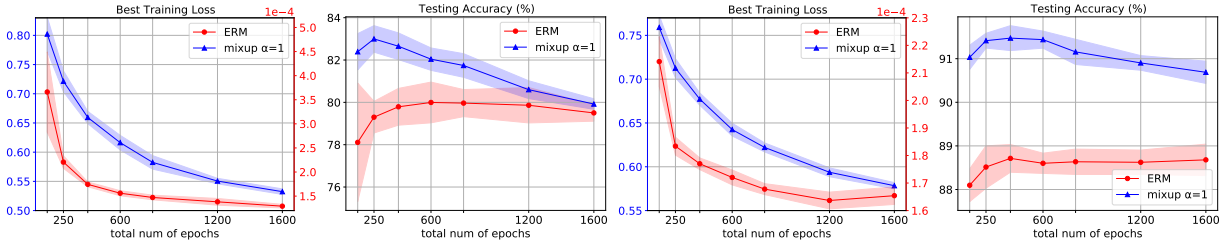
Figure 5.4: Results of the recorded training losses and testing accuracies of training ResNet34 on CIFAR10 training set (100% data and 30% data) without data augmentation.



(a) Train loss (30%) (b) Test acc (30%) (c) Train loss (100%) (d) Test acc (100%)

Figure 5.5: Results of the recorded training losses and testing accuracies of training ResNet34 on SVHN training set (100% data and 30% data) without data augmentation.

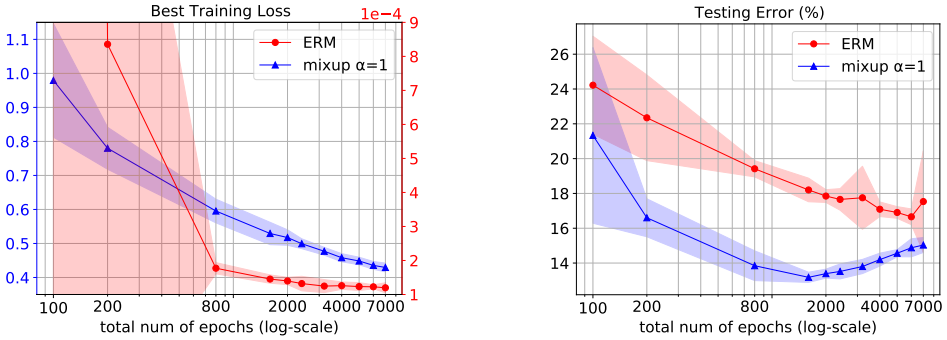
In addition, we have trained VGG16 on the CIFAR10 training set (100% data and 30% data) for up to in total 1600 epochs without data augmentation. The results are provided in Figure 5.6. In both cases, over-training VGG16 with either ERM or Mixup can gradually reduce the best achieved training loss. However, the testing accuracy of the Mixup-trained network also decreases, while that of the ERM-trained network has no significant change.



(a) Train loss (30%) (b) Test acc (30%) (c) Train loss (100%) (d) Test acc (100%)

Figure 5.6: Results of the recorded training losses and testing accuracies of training VGG16 on CIFAR10 training set (30% data and 100% data) without data augmentation.

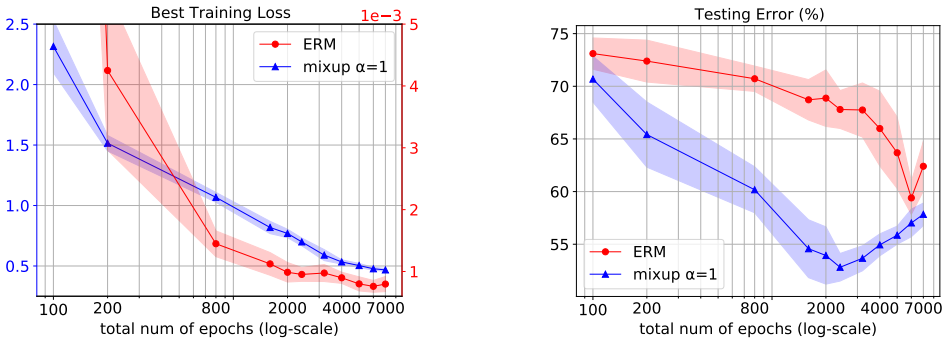
We have also carried out the over-training experiments with data augmentation applied including "random crop" and "horizontal flip". We train ResNet18 on 10% of the CIFAR10 training set with data augmentation for up to in total 7000 epochs. The results are given in Figure 5.7. In this case, the Mixup-trained models also produce a U-shaped generalization curve along with the total numbers of epochs. However, the inflection point of the U-shaped curve comes much later compared to that in any of the previous experiments where data augmentation is not applied.



(a) Training Loss (10% data) (b) Testing Error (10% data)

Figure 5.7: Results of the recorded training losses and testing errors of over-training ResNet18 on 10% of the CIFAR10 training set with data augmentation.

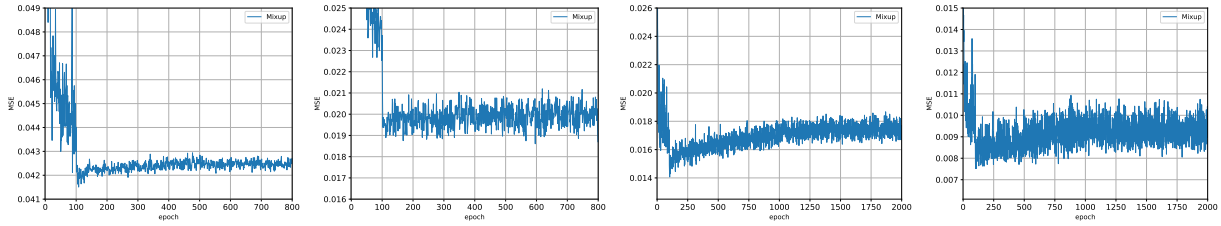
Also, the results of over-training ResNet34 on 10% of the CIFAR100 training set for up to in total 7000 epochs are given in Figure 5.8. In this case, the similar phenomenons as in the previous case has also been observed.



(a) Training Loss (10% data) (b) Testing Error (10% data)

Figure 5.8: Results of the recorded training losses and testing errors of over-training ResNet34 on 10% of the CIFAR100 training set with data augmentation.

We also conduct Mixup training experiments by using the MSE loss on CIFAR10 and SVHN. Figure 5.9 shows that the U-shaped behavior also holds for the MSE loss. The learning rate is divided by 10 at epoch 100 and 150 successively.



(a) CIFAR10 (30%) (b) CIFAR10 (100%) (c) SVHN (30%) (d) SVHN (100%)

Figure 5.9: Dynamics of MSE during Mixup training.

5.4 Theoretical Explanation

In this section, we first demonstrate that Mixup training may induce undesired label noises. We then analyze a special case to illustrate why label noise will render the U-shaped behaviour in its generalization dynamics.

5.4.1 Mixup Induces Label Noise

We will use the capital letters X and Y to denote the random variables corresponding to the input feature and output label, while reserving the notations $\mathbf{x}$ and $\mathbf{y}$ to denote their realizations respectively. In particular, we consider that each true label $\mathbf{y}$ is an element, *i.e.* a token, in $\mathcal{Y}$, not a one-hot vector in $\mathcal{P}(\mathcal{Y})$. Let $P(Y|X)$ be the ground-truth conditional distribution of the label Y given the input feature X. For the simplicity of the notation, we will also express $P(Y|X)$ using a vector-valued function $f : \mathcal{X} \to \mathbb{R}^C$, where $f_j(\mathbf{x}) := P(Y = j|X = \mathbf{x})$ for each dimension $j \in \mathcal{Y}$ and input $\mathbf{x}$. Under this ground truth, the correct hard-assignment of the label for $\mathbf{x}$ is $\arg\max_{j\in\mathcal{Y}} f_j(\mathbf{x})$.

For simplification, we consider Mixup with a fixed $\lambda \in [0, 1]$; the extension to a random λ is straight-forward. Let $\widetilde{X}$ and $\widetilde{Y}$ be the random variables corresponding to the synthetic feature and synthetic label respectively. Then $\widetilde{X} := \lambda X + (1 - \lambda)X'$. Let $P(\widetilde{Y}|\widetilde{X})$ be the conditional distribution of the synthetic label conditioned on the synthetic feature induced by Mixup, *i.e.* $P(\widetilde{Y} = j|\widetilde{X}) = \lambda f_j(X) + (1 - \lambda)f_j(X')$ for each j. Then for a synthetic

feature $\widetilde{X}$, there are two ways to assign to it a hard label. The first is based on the ground truth, assigning $\widetilde{Y}_{\mathrm{h}}^* := \arg\max_{j\in\mathcal{Y}} f_j(\widetilde{X})$. The second is based on the Mixup-induced conditional $P(\widetilde{Y}|\widetilde{X})$, assigning $\widetilde{Y}_{\mathrm{h}} := \arg\max_{j\in\mathcal{Y}} P(\widetilde{Y}=j|\widetilde{X})$. When the two assignments disagree, or $\widetilde{Y}_{\mathrm{h}} \neq \widetilde{Y}_{\mathrm{h}}^*$, we say that the Mixup-assigned label $\widetilde{Y}_{\mathrm{h}}$ is noisy.

Theorem 5.4.1. *For any fixed X, X' and $\widetilde{X}$ related by $\widetilde{X} = \lambda X + (1-\lambda)X'$ for a fixed $\lambda \in [0,1]$. The probability of assigning a noisy label is lower bounded by*

$$P(\widetilde{Y}_{\mathrm{h}} \neq \widetilde{Y}_{\mathrm{h}}^*|\widetilde{X}) \geq \mathrm{TV}(P(\widetilde{Y}|\widetilde{X}), P(Y|X)) \geq \frac{1}{2}\sup_{j\in\mathcal{Y}}\left|f_j(\widetilde{X}) - [(1-\lambda)f_j(X) + \lambda f_j(X')]\right|,$$

where $\mathrm{TV}(\cdot,\cdot)$ is total variation distance (see Section 2.15).

Remark 5.4.1. *This lower bound hints that label noises induced by Mixup training depends on the distribution of the original data: P_X, the convexity of $f(X)$ and the value of λ. Clearly, Mixup will not create any noisy label almost surely when f_j is linear for each j.*

Remark 5.4.2. *In practice, we often use the one-hot vector to denote the real data label, that is to say, we let $\max_{j\in\mathcal{Y}} f_j(X) = 1$ and $\sum_{j=1}^{C} f_j(X) = 1$. Thus, the probability of assigning noisy label to a given synthetic data can be discussed in three situations: i) if $\widetilde{Y}_{\mathrm{h}}^* \notin \{Y, Y'\}$, where Y could be the same with Y', then $\widetilde{Y}$ is a noisy label with probability one; ii) if $\widetilde{Y}_{\mathrm{h}}^* \in \{Y, Y'\}$ where $Y \neq Y'$, then a noisy label is assigned with probability at least λ or $1-\lambda$; iii) if $\widetilde{Y}_{\mathrm{h}}^* = Y = Y'$, then $\widetilde{Y}_{\mathrm{h}}^* = \widetilde{Y}$.*

As shown in some previous works [3, 5], when the neural networks are trained with a fraction of the random labels, they will first learn the clean data and then overfit to the data with the noisy labels. In the Mixup training case, we indeed create much more data than traditional ERM training (e.g., n^2 for a fixed λ). Thus, Mixup training will give higher testing performance in the first stage of learning (where the neural networks learn the true pattern of the data [5]), and then the performance will be impaired due to overfitting to the noisy data. In some cases, if $\widetilde{Y}_{\mathrm{h}}^* \notin \{Y, Y'\}$ happens with a high chance, which is exactly the manifold intrusion problem, then the synthetic dataset contains too many noisy labels, thus Mixup training may be unable to give better performance than ERM training.

To this end, Theorem 5.1 suggests that for any classification problem, Mixup training induces label noise. Next, we will provide a theoretical analysis using a regression setup to explain that such label noise may result in the U-shape learning curve. The choice of a regression setup in this analysis is due to the difficulty in directly analyzing classification problems (under the cross-entropy loss). We note that the regression setting

may not perfectly explain the U-shaped curve in classification datasets, we however believe that they give adequate insight illuminating such scenarios as well. Indeed, due to the difficulty of analyzing the cross-entropy loss, most of the analytic works for deep learning study regression problems under the square-error loss and use insights obtain this way to explain the behaviour of deep neural net in classification settings. For example, [4] uses a regression setup to analyze the optimization and generalization property of overparameterized neural networks. [41] theoretically analyze the bias-variance trade-off in deep network generalization using a regression problem.

5.4.2 Regression Setting With Random Feature Models

To further illustrate the training dynamics on dataset with noisy labels, we now consider a simple least squares regression problem. Let $\mathcal{Y} = \mathbb{R}$ and let $f : \mathcal{X} \to \mathcal{Y}$ be the ground-truth labelling function. Let $(\widetilde{X}, \widetilde{Y})$ be a synthetic pair obtained by mixing (X, Y) and (X', Y'). Let $\widetilde{Y}^* = f(\widetilde{X})$ and $Z := \widetilde{Y} - \widetilde{Y}^*$. Then Z can be regarded as noise introduced by Mixup, which may be data-dependent. For example, if f is strongly convex with some parameter $\rho > 0$, then $Z \geq \frac{\rho}{2}\lambda(1-\lambda)||X - X'||_2^2$.

Given a synthetic training dataset $\widetilde{S} = \{(\widetilde{X}_i, \widetilde{Y}_i)\}_{i=1}^m$, consider a random feature model, $\theta^T \phi(X)$, where $\phi : \mathcal{X} \to \mathbb{R}^d$ and $\theta \in \mathbb{R}^d$. We will train the model using gradient descent on the MSE loss

$$\hat{R}_{\widetilde{S}}(\theta) := \frac{1}{2m} \left|\left| \theta^T \widetilde{\Phi} - \widetilde{\mathbf{Y}}^T \right|\right|_2^2,$$

where $\widetilde{\Phi} = [\phi(\widetilde{X}_1), \phi(\widetilde{X}_2), \ldots, \phi(\widetilde{X}_m)] \in \mathbb{R}^{d \times m}$ and $\widetilde{\mathbf{Y}} = [\widetilde{Y}_1, \widetilde{Y}_2, \ldots, \widetilde{Y}_m] \in \mathbb{R}^m$. Notably, we will consider ϕ as fixed and only update θ.

For a fixed λ, Mixup can create $m = n^2$ synthetic examples. Thus it is reasonable to assume $m > d$ (e.g., under-parameterized regime) in Mixup training. For example, ResNet50 has less than 30 million parameters while the square of the CIFAR10 training set's size is larger than 200 million without using other data augmentation techniques. Then the gradient flow is

$$\dot{\theta} = -\eta \nabla \hat{R}_{\widetilde{S}}(\theta_t) = \frac{\eta}{m} \widetilde{\Phi} \widetilde{\Phi}^T \left(\widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}} - \theta_t \right), \tag{5.8}$$

where η is the learning rate and $\widetilde{\Phi}^\dagger = (\widetilde{\Phi}\widetilde{\Phi}^T)^{-1}\widetilde{\Phi}$ is the Moore–Penrose inverse of $\widetilde{\Phi}^T$ (only possible when $m > d$).

Thus, we have the following important lemma.

Lemma 5.4.1. *Let* $\theta^* = \widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}}^*$ *and* $\theta^{\text{noise}} = \widetilde{\Phi}^\dagger \mathbf{Z}$ *wherein* $\mathbf{Z} = [Z_1, Z_2, \ldots, Z_m] \in \mathbb{R}^m$*, the ODE of Equation (5.8) has the following closed form solution*

$$\theta_t - \theta^* = (\theta_0 - \theta^*) e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t} + (\mathbf{I}_d - e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t})\theta^{\text{noise}}. \tag{5.9}$$

Remark 5.4.3. *Lemma 5.4.1 indicates that the dynamics of* θ *gives a U-shaped curve in each dimension, and the increasing behavior results from the second term that contains the noise* $\mathbf{Z}$*. More precisely, the first term in Equation (5.9) is monotonically decreasing and it dominates the dynamics of* θ_t *in the early phase of learning. Remarkably* $\theta^* = \widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}}^*$ *may be understood as the "clean pattern" of the training data. Then we see that the model, in the early phase, is learning the "clean pattern", which generalizes to the unseen data (i.e.* (X, Y)*). In the later training phase, the second term in Equation (5.9) gradually dominates the trajectory of* θ_t*, and the model learns the"noisy pattern", namely* $\theta^{\text{noise}} = \widetilde{\Phi}^\dagger \mathbf{Z}$*. This then hurts generalization. It is noteworthy that* $\theta^* + \theta^{\text{noise}}$ *is also the closed-form solution for the regression problem (under Mixup labels). This suggests that the optimization problem associated with the Mixup loss has a "wrong" solution, but it is possible to benefit from only solving this problem partially, using gradient descent without over-training.*

For a given synthetic dataset $\widetilde{S}$, the expected population risk as a function of time step t is

$$R_t := \mathbb{E}_{\theta_t, X, Y} \left|\left| \theta_t^T \phi(X) - Y \right|\right|_2^2.$$

The following theorem shows the dynamics of the population risk under mild assumptions.

Theorem 5.4.2 (Dynamic of Population Risk)**.** *Given a synthetic dataset* $\widetilde{S}$*, assume* $\theta_0 \sim \mathcal{N}(0, \xi^2 \mathrm{I}_d)$*,* $||\phi(X)||^2 \leq C_1/2$ *for some constant* $C_1 > 0$ *and* $|Z| \leq \sqrt{C_2}$ *for some constant* $C_2 > 0$*, then we have the following upper bound*

$$R_t - R^* \leq C_1 \sum_{k=1}^{d} \left[\left(\xi_k^2 + \theta_k^{*2}\right) e^{-2\eta\mu_k t} + \frac{C_2}{\mu_k} \left(1 - e^{-\eta\mu_k t}\right)^2 \right] + 2\sqrt{C_1 R^* \zeta},$$

where $R^* = \mathbb{E}_{X,Y} \left|\left| Y - \theta^{*T} \phi(X) \right|\right|_2^2$*,* $\zeta = \sum_{k=1}^{d} \max\{\xi_k^2 + \theta_k^{*2}, \frac{C_2}{\mu_k}\}$ *and* μ_k *is the* k^{th} *eigenvalue of the matrix* $\frac{1}{m}\widetilde{\Phi}\widetilde{\Phi}^T$*.*

Remark 5.4.4. *The additive noise* Z *is usually assumed as a zero mean Gaussian in the literature of generalization dynamics analysis [1, 19, 34]. We note that the boundness assumption of the data-dependent noise in the theorem is easily satisfied as long as the output of* f *is bounded, while there is no clue to assume* Z *is Gaussian.*

Remark 5.4.5. *If we further let $\xi = 0$ (i.e. using zero initialization) and assume that the eigenvalues of the matrix $\frac{1}{m}\widetilde{\Phi}\widetilde{\Phi}^T$ are all equal to μ, then the summation part in the bound above can be re-written as $C_1 ||\theta^*||^2 e^{-2\eta\mu t} + C_2/\mu (1 - e^{-\eta\mu t})^2$, then it is clear that the magnitude of the curve is controlled by the norm of θ^*, the norm of the representation, the noise level and μ.*

Theorem 5.4.2 indicates that the population risk in each dimension will first convexly decrease due to the first term (*i.e.* $(\xi_k^2 + \theta_k^{*2}) e^{-2\eta\widetilde{\mu}_k t}$, then it will concavely grow due to the existence of the label noises (*i.e.* $\frac{C_2}{\mu_k}(1 - e^{-\eta\mu_k t})^2)$). Overall, the population risk will be endowed with the U-shaped behavior. Notice that the quantity $\eta\mu_k$ plays a key role in the upper bound, the larger $\eta\mu_k$ is, the earlier inflection point of the U-shaped curve comes. This may have an interesting application, justifying a multi-stage training strategy where the learning rate is reduced at each new stage. Suppose that with the initial learning rate, at epoch T, the test error has dropped to the bottom of the U-curve corresponding to this learning rate. If the learning rate is decreased at this point, then the U-shape curve corresponding to the new learning rate may have a lower minimum error and its bottom shifted to the right. In this case, the new learning rate allows the testing error to move to the new U-shaped curve and further decay.

5.5 Empirical Verification

In this section, we present empirical evidences to validate our theoretical results in Section 5.4.

5.5.1 A Teacher-Student Toy Setup

To empirically verify our theoretical results discussed in Section 5.4.2, we invoke a simple teacher-student setting. Consider the original data $\{X_i\}_{i=1}^n$ are drawn i.i.d. from a standard Gaussian $\mathcal{N}(0, \mathrm{I}_{d_0})$, and the teacher network is a two-layer neural networks with $\tanh(\cdot)$ as the activation function. The student network is also a two-layer neural network with $\tanh(\cdot)$, where we fix the parameters in the first layer and only train the second layer. The output dimension of the first layer is 100 (i.e. $d = 100$) and we use full-batch gradient descent to train the student network using MSE as the loss function. For the value of λ, we consider two cases: a fixed value with $\lambda = 0.5$ and a random value drawn from $\mathrm{Beta}(1, 1)$ at each epoch. As a comparison, we also present the result of ERM training in

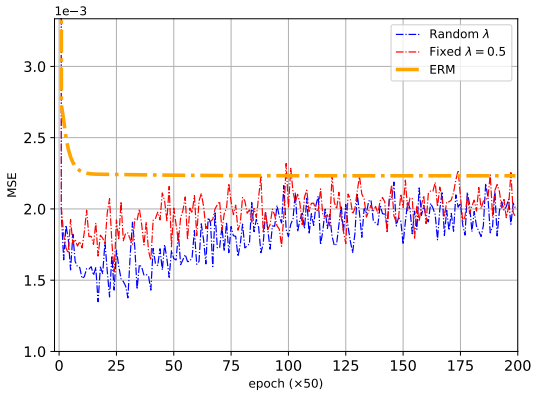


Figure 5.10: Dynamics of Testing Loss in the Teacher-Student setting.

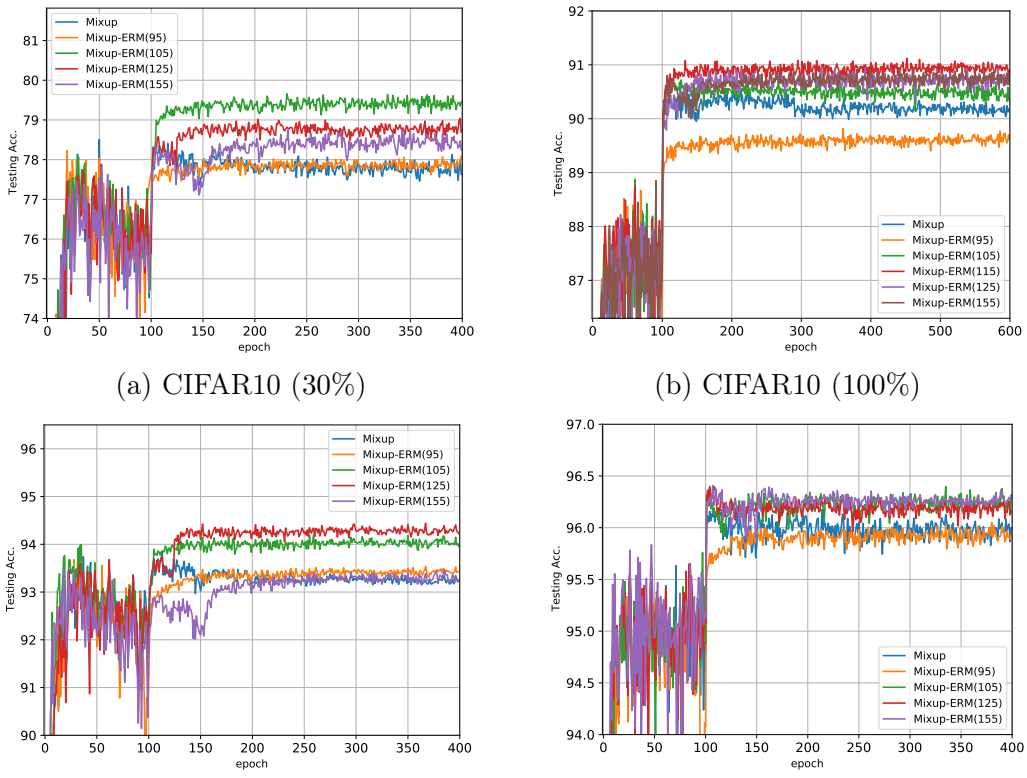
an overparameterized regime (i.e., $n < d$). All the testing loss dynamics are presented in Figure 5.10.

From Figure 5.10, we first note that Mixup still outperforms ERM in this regression problem, but clearly, only Mixup training has a U-shaped curve while the testing loss of ERM training converges to a constant value. Furthermore, the testing loss of Mixup training is endowed with a U-shaped behavior for both of the λ scenarios This indeed justifies that our analysis does not depend on any specific λ distribution. Moreover, Figure 5.10 indicates that when λ is fixed as 0.5, the increasing stage of the U-shaped curve comes earlier than that of $\lambda \sim \text{Beta}(1, 1)$. This is again consistent with our theoretical results in Section 5.4.2. That is, owning to the fact that λ with a constant value of 0.5 provides larger noise level for Mixup, the noise domination effect for Mixup training comes earlier.

5.5.2 Using Mixup Only in the Early Stage of Training

We here aim to empirically verify that Mixup can induce label noises as discussed in Section 5.4.1.

If Mixup training learns the "clean patterns" in the early stage of the training process and then overfits the "noisy patterns" in the latter stage, then we can stop applying Mixup in training the model after a certain number of epochs. Doing so, we can prevent the model from overfitting the noises induced by Mixup. We present the results of utilizing such training schema for both CIFAR10 and SVHN in Figure 5.11.



(a) CIFAR10 (30%)

(b) CIFAR10 (100%)

(c) SVHN (30%)

(d) SVHN (100%)

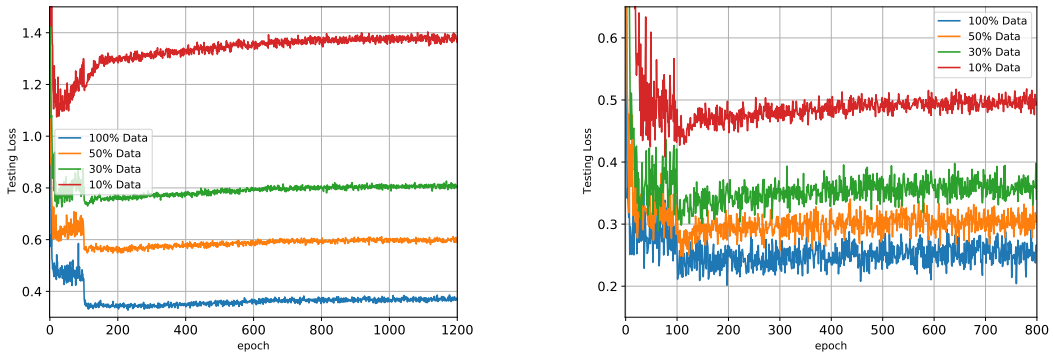
Figure 5.11: Switching from Mixup training to ERM training. The number in the bracket is the epoch number where we let $\alpha = 0$ (i.e. Mixup training becomes ERM training).

Results in Figure 5.11 clearly indicate that switching from Mixup to ERM at a propertied time point during training will successfully avoid the generalization degradation of Mixup training. Figure 5.11 also suggests that it may not boost the model's performance if we change Mixup to ERM before the clean examples created by Mixup have large effect. In addition, if we change Mixup to ERM too late, then the memorization of the noisy data may already has negative effect on the generalization performance. We note that our results here can be regarded as a complement to the work of [13], where the authors show that a regularization technique only matters during the early phase of learning.

5.6 Further Investigation

Impact of Data Size on U-shaped Curve

Although the U-shaped behavior occurs for using both 100% and 30% of the original data of CIFAR10 and SVHN, we notice that the smaller-size datasets facilitate the U-shaped behavior to present. We present such experimental results in Figure 5.12.



(a) CIFAR10 (b) SVHN

Figure 5.12: Over-training on Different Number of Samples.

It may be tempting to think that we can apply the usual analysis of the generalization dynamics in the existing literature [1, 28, 37], where they utilize some tools from random matrix theory. For example one can analyze the distribution of the eigenvalues in Theorem 5.4.2. Specifically, if entries in Φ are independently identically distributed with zero mean, then the eigenvalues $\{\mu_k\}_{k=1}^d$ follow the Marchenko-Pasteur (MP) distribution [31] in the limit $d, m \to \infty$ with $d/m = \gamma \in (0, +\infty)$, which is defined as:

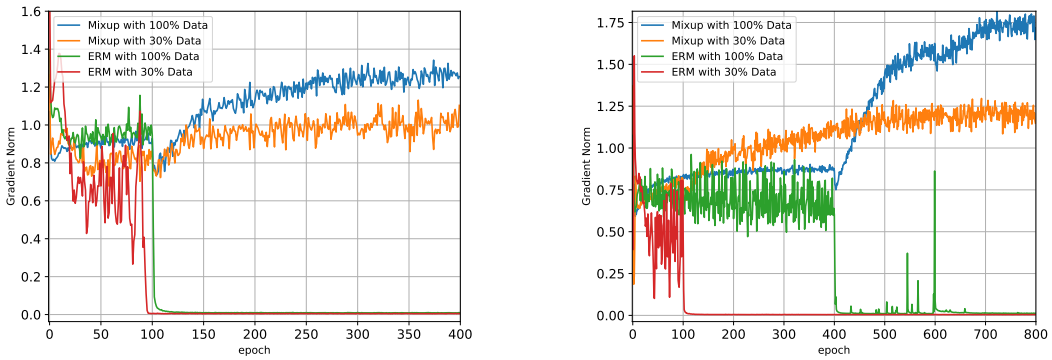
$$P^{MP}(\mu|\gamma) := \frac{1}{2\pi} \frac{\sqrt{(\gamma_+ - \mu)(\mu - \gamma_-)}}{\mu\gamma} \mathbb{1}_{\mu \in [\gamma_-, \gamma_+]},$$

where $\gamma_\pm = (1 \pm \gamma)^2$. Note that P^{MP} are only non-zero when $\mu = 0$ or $\mu \in [\gamma_-, \gamma_+]$. When γ is close to one, the probability of the extremely small eigenvalues is immensely increased. From Theorem 5.4.2, when μ_k is small, the second term that contains noise will badly dominate the behavior of population risk and converge to a larger value. Thus, letting $d \ll m$ will alleviate the domination of the noise term in Theorem 5.4.2. However, it is important to note that such analysis is not rigorous enough since the columns in Φ are not independent (each two columns might come from the linear combination of the same two original instances). To apply the similar analysis here, one may need to remove or relax the

independence conditions for the MP distribution to hold, for example, by invoking some techniques developed in [6]. This is beyond the scope of this thesis, and we would like to leave it for the future study.

Gradient Norm in Mixup Training Does Not Vanish

Normally, ERM training will obtain zero gradient norm at the end of training, which indicates that a local minimum is found by SGD. However, we observe that the gradient norm of Mixup training does not converge to zero, as shown in Figure 5.13.



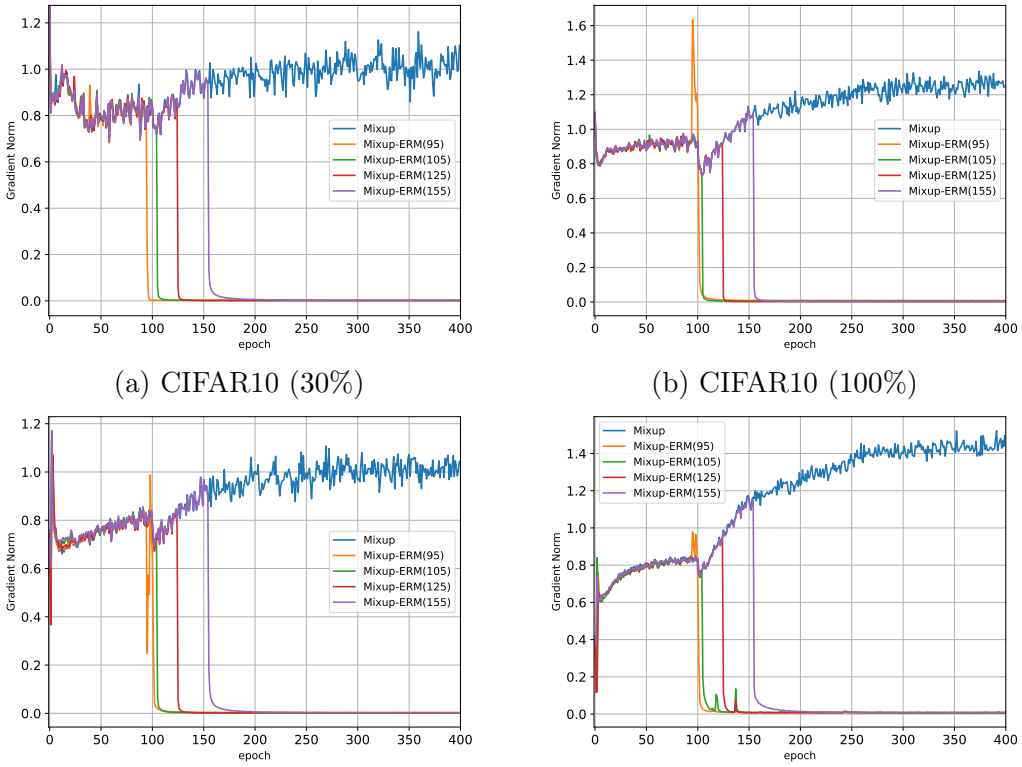
(a) Gradient Norm on CIFAR10 (b) Gradient Norm on SVHN

Figure 5.13: Dynamics of Gradient Norm.

Indeed, the gradient norm in Mixup training even increases until converging to a maximum value instead of converging to zero. When the models are trained on random labels, this increasing trend of the gradient norm is also observed in some previous works [12, 40]. Specifically, in [40], such an increasing behavior is interpreted as a sign that the training of SGD enters a "memorization regime", and after the overparameterized neural networks memorize all the noisy labels, the gradient norm (or gradient dispersion in [40]) will decrease again until it converges to zero. In Mixup training, since the size of the synthetic dataset is usually larger than the number of the parameters (i.e., $m > d$), the neural networks may not be able to memorize all the noisy labels in this case. Notice that m is larger than n^2 in practice since λ is not fixed for all the pairs of original data.

Notably, although ERM training is able to find a local minimum in the first 130 epochs on CIFAR10, Figure 1.1 indicates that Mixup training nevertheless outperforms ERM in the first 400 epochs. Similar observation also holds for SVHN. This result indeed suggests that Mixup can generalize well without converging to any stationary point. Notice that there is a close observation in the recent work of [45], where they show that the large-scale neural networks generalize well without the vanishing of the gradient norm during training.

Additionally, by switching Mixup training to ERM training, as what we did in Figure 5.11, the gradient norm will instantly become zero as shown in Figure 5.14. This further justifies that the "clean patterns" are already learned by the Mixup-trained neural network at the early stage of training, and the original data may not provide any useful gradient signal.



(a) CIFAR10 (30%)

(b) CIFAR10 (100%)

(c) SVHN (30%)

(d) SVHN (100%)

Figure 5.14: Dynamics of Gradient Norm when changing Mixup training to ERM training.

5.7 Concluding Remarks

We empirically discovered a previously unobserved phenomenon in Mixup training: over-training with Mixup may give rise to a U-shaped generalization curve. We further theoretically showed that Mixup training may introduce undesired data-dependent label noises to the synthetic data, and such noises facilitate the U-shaped generalization curve to occur. That is, Mixup improves a model's generalization through fitting the clean patterns

at the early training stage, but as training progresses, Mixup becomes overfitting to the introduced noise. We also provided the experimental indications that unlike ERM training, Mixup can generalize well without converging to any stationary point.

Our research here uncovers a unique generalization behavior of the effective Mixup regularizer, which paves the ways for several promising directions that are worth being further studied. First, leveraging the U-shaped generalization behavior identified to devise a training paradigm for Mixup to automatically optimize its regularization effect would be beneficial. Second, unifying Mixup's generalization behavior here with those being pointed out by the previous works would be useful. Finally, theoretically verifying that Mixup generalizes well without converging to any stationary point would help improve our understanding on Mixup's generalization capabilities.

Chapter 6

Conclusions and Future Works

In the first part of this thesis, we focused on the techniques about solving manifold intrusion [16]. We categorised the majority of the corresponding arts into two categories: the relabelling strategy and the cautious mixing strategy. The relabelling strategy refers to the series of techniques that attempt to solve manifold intrusion by relabelling the synthetic training data, while under the cautious mixing regime we aim to solve this issue by carefully adjusting the interpolation policy of Mixup.

We then proposed two new relabelling algorithms: pseudo relabelling and reference relabelling, and we carried out the experiments of these two algorithms. Comparing the experiments results with the standard Mixup baselines and the ERM baselines, we showed that the two of our proposed algorithms had both failed to properly prevent manifold intrusion.

For the cautious mixing strategy, we empirically showed by cautiously selecting the interpolation coefficient α, the generalization performance of the Mixup-trained model can be further improved on the basis of the ERM baseline. Particularly, we showed that on the Spiral3-90 dataset and the radiate dataset, smaller choices of α can lead the models to learn smoother and fairer decision boundaries. At the end of this part, we compared the arts of these two strategies in the senses of both their efficiency and their effectiveness. We indicated that in general the cautious mixing strategy outperforms the relabelling strategy, making the former more promising.

As a conclusion, we also proposed a suggestion for the future works about solving manifold intrusion: although the relabelling strategy may still have potential prospects, we nevertheless suggest that the researchers focus more on the cautious mixing strategy. More specifically, since a variety of works have already made their explorations from the

perspective of adjusting the interpolation coefficient, we believe that making attempt to optimize the examples pairing policy would be another challenging yet promising path.

In the second part, we report a previously unobserved phenomenon that we empirically discovered in Mixup training: over-training with Mixup may give rise to a U-shaped generalization curve. In other words, we had observed that if we gradually increase the training epochs of a network on a variety of the benchmark datasets, while the best achieved training loss can be further reduced, the corresponding testing performance also drops significantly, leading to the overfitting issue.

We then theoretically explained this behavior: Mixup training may introduce undesired data-dependent label noises to the synthetic data, and such noises facilitate the U-shaped generalization curve to occur. That is, Mixup improves a model's generalization through fitting the clean patterns at the early training stage, but as training progresses, the Mixup-trained model becomes overfitting to the introduced noise. We also provided the experimental indications that unlike ERM training, Mixup can generalize well without converging to any stationary point.

Our research here uncovers a unique generalization behavior of the effective Mixup regularizer, which paves the ways for several promising directions that are worth being further studied.

- First, leveraging the U-shaped generalization behavior identified to devise a training paradigm for Mixup to automatically optimize its regularization effect would be beneficial.
- Secondly, we can integrate the generalization behavior of Mixup with those of other scenarios in deep learning such as [5, 12, 29, 30, 40] to gain a deeper understanding of this field. For example, in [12], the authors indicate that the regularization in the early stage of training plays the most essential role in improving the generalization performance.
- Finally, theoretically verifying that Mixup generalizes well without converging to any stationary point would help improve our understanding on Mixup's generalization capabilities.

References

[1] Madhu S Advani, Andrew M Saxe, and Haim Sompolinsky. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 132:428–446, 2020.

[2] Guillaume P Archambault, Yongyi Mao, Hongyu Guo, and Richong Zhang. Mixup as directional adversarial training. *arXiv preprint arXiv:1906.06875*, 2019.

[3] Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning*, pages 322–332. PMLR, 2019.

[4] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *ICML*, volume 97, pages 322–332. PMLR, 2019.

[5] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International conference on machine learning*, pages 233–242. PMLR, 2017.

[6] Jennifer Bryson, Roman Vershynin, and Hongkai Zhao. Marchenko–pastur law with relaxed independence conditions. *Random Matrices: Theory and Applications*, 10(04):2150040, 2021.

[7] Luigi Carratino, Moustapha Cissé, Rodolphe Jenatton, and Jean-Philippe Vert. On mixup regularization. *arXiv preprint arXiv:2006.06049*, 2020.

[8] Jiaao Chen, Zichao Yang, and Diyi Yang. Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification. *CoRR*, abs/2004.12239, 2020.

[9] Tianlong Chen, Zhenyu Zhang, Sijia Liu, Shiyu Chang, and Zhangyang Wang. Robust overfitting may be mitigated by properly learned smoothening. In *International Conference on Learning Representations*, 2020.

[10] Casio Computer Co. Beta distribution (chart) calculator. https://keisan.casio.com/exec/system/1180573226.

[11] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[12] Yu Feng and Yuhai Tu. Phases of learning dynamics in artificial neural networks in the absence or presence of mislabeled data. *Machine Learning: Science and Technology*, 2(4):043001, 2021.

[13] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. *Advances in Neural Information Processing Systems*, 32, 2019.

[14] Kristjan Greenewald, Anming Gu, Mikhail Yurochkin, Justin Solomon, and Edward Chien. k-mixup regularization for deep learning via optimal transport, 2021.

[15] Hongyu Guo. Nonlinear mixup: Out-of-manifold data augmentation for text classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4044–4051, Apr. 2020.

[16] Hongyu Guo, Yongyi Mao, and Richong Zhang. Mixup as locally linear out-of-manifold regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3714–3722, 2019.

[17] Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. *arXiv preprint arXiv:2202.07179*, 2022.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[19] Reinhard Heckel and Fatih Furkan Yilmaz. Early stopping in deep networks: Double descent and how to eliminate it. In *International Conference on Learning Representations*, 2021.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural computation*, 9(1):1–42, 1997.

[21] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[22] Jang-Hyun Kim, Wonho Choo, and Hyun Oh Song. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *International Conference on Machine Learning (ICML)*, 2020.

[23] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2009.

[24] Alexander Lenail. Publication-ready cnn-architecture schematics. http://alexlenail.me/NN-SVG/LeNet.html.

[25] Alexander Lenail. Publication-ready mlp-architecture schematics. http://alexlenail.me/NN-SVG/index.html.

[26] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[27] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31, 2018.

[28] Zhenyu Liao and Romain Couillet. The dynamics of learning: A random matrix approach. In *International Conference on Machine Learning*, pages 3072–3081. PMLR, 2018.

[29] Sheng Liu, Jonathan Niles-Weed, Narges Razavian, and Carlos Fernandez-Granda. Early-learning regularization prevents memorization of noisy labels. *Advances in neural information processing systems*, 33:20331–20342, 2020.

[30] Sheng Liu, Zhihui Zhu, Qing Qu, and Chong You. Robust training under label noise by over-parameterization. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba

Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 14153–14172. PMLR, 17–23 Jul 2022.

[31] Vladimir A Marčenko and Leonid Andreevich Pastur. Distribution of eigenvalues for some sets of random matrices. *Mathematics of the USSR-Sbornik*, 1(4):457, 1967.

[32] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *International Conference on Learning Representations*, 2020.

[33] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[34] Mohammad Pezeshki, Amartya Mitra, Yoshua Bengio, and Guillaume Lajoie. Multi-scale feature learning dynamics: Insights for double descent. In *International Conference on Machine Learning*, pages 17669–17690. PMLR, 2022.

[35] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.

[36] Jy-yong Sohn, Liang Shang, Hongxu Chen, Jaekyun Moon, Dimitris Papailiopoulos, and Kangwook Lee. Genlabel: Mixup relabeling using generative models. *arXiv preprint arXiv:2201.02354*, 2022.

[37] Cory Stephenson and Tyler Lee. When and how epochwise double descent happens. *arXiv preprint arXiv:2108.12006*, 2021.

[38] Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[39] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6438–6447. PMLR, 09–15 Jun 2019.

[40] Ziqiao Wang and Yongyi Mao. On the generalization of models trained with SGD: Information-theoretic bounds and implications. In *International Conference on Learning Representations*, 2022.

[41] Zitong Yang, Yaodong Yu, Chong You, Jacob Steinhardt, and Yi Ma. Rethinking bias-variance trade-off for generalization of neural networks. In *ICML*, 2020.

[42] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[43] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[44] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[45] Jingzhao Zhang, Haochuan Li, Suvrit Sra, and Ali Jadbabaie. Neural network weights do not converge to stationary points: An invariant measure perspective. In *International Conference on Machine Learning*, pages 26330–26346. PMLR, 2022.

[46] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Zou. How does mixup help with robustness and generalization? In *International Conference on Learning Representations*, 2021.

[47] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, and James Zou. When and how mixup improves calibration. In *International Conference on Machine Learning*, pages 26135–26160. PMLR, 2022.

APPENDICES

A Proof of Theorem 5.4.1

Proof. By the coupling inequality i.e. Lemma 2.15.2, we have

$$\mathrm{TV}(P(\widetilde{Y}_{\mathrm{h}}|\widetilde{X}), P(\widetilde{Y}_{\mathrm{h}}^*|\widetilde{X})) \leq P(\widetilde{Y}_{\mathrm{h}} \neq \widetilde{Y}_{\mathrm{h}}^*|\widetilde{X}),$$

Since $\mathrm{TV}(P(\widetilde{Y}_{\mathrm{h}}|\widetilde{X}), P(Y|X)) = \mathrm{TV}(P(\widetilde{Y}_{\mathrm{h}}|\widetilde{X}), P(\widetilde{Y}_{\mathrm{h}}^*|\widetilde{X}))$, then the first inequality is straightforward.

For the second inequality, by Lemma 2.15.1, we have

$$\begin{aligned}
\mathrm{TV}(P(\widetilde{Y}_{\mathrm{h}}|\widetilde{X}), P(\widetilde{Y}_{\mathrm{h}}^*|\widetilde{X})) =& \frac{1}{2}\sum_{j=1}^{C}\left|P(\widetilde{Y}^* = j|\widetilde{X}) - P(\widetilde{Y} = j|\widetilde{X})\right| \\
=& \frac{1}{2}\sum_{j=1}^{C}\left|f_j(\widetilde{X}) - ((1-\lambda)f_j(X) + \lambda f_j(X'))\right| \\
\geq& \sup_j \frac{1}{2}\left|f_j(\widetilde{X}) - ((1-\lambda)f_j(X) + \lambda f_j(X'))\right|.
\end{aligned}$$

This completes the proof. □

B Proof of Lemma 5.4.1

Proof. The proof here is trivial. The ordinary differential equation of Equation (5.8) (Newton's law of cooling) has the closed form solution

$$\theta_t = \widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}} + (\theta_0 - \widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}}) e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t}. \tag{1}$$

Recall that $\widetilde{\mathbf{Y}} = \widetilde{\mathbf{Y}}^* + \mathbf{Z}$,

$$\begin{aligned}
\theta_t =& \widetilde{\Phi}^\dagger \left(\widetilde{\mathbf{Y}}^* + \mathbf{Z}\right) + (\theta_0 - \widetilde{\Phi}^\dagger \left(\widetilde{\mathbf{Y}}^* + \mathbf{Z}\right)) e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t} \\
=& \widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}}^* + \widetilde{\Phi}^\dagger \mathbf{Z} + \left(\theta_0 - \widetilde{\Phi}^\dagger \widetilde{\mathbf{Y}}^*\right) e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t} - \widetilde{\Phi}^\dagger \mathbf{Z} e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t} \\
=& \theta^* + (\theta_0 - \theta^*) e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t} + (\mathbf{I}_d - e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t})\theta^{\text{noise}},
\end{aligned}$$

which concludes the proof. □

C Proof of Theorem 5.4.2

Proof. We first notice that

$$\begin{aligned}
R_t =& \mathbb{E}_{\theta_t, X, Y} \left|\left|\theta_t^T \phi(X) - Y\right|\right|_2^2 \\
=& \mathbb{E}_{\theta_t, X, Y} \left|\left|\theta_t^T \phi(X) - \theta^{*T}\phi(X) + \theta^{*T}\phi(X) - Y\right|\right|_2^2 \\
=& \mathbb{E}_{\theta_t, X} \left|\left|\theta_t^T \phi(X) - \theta^{*T}\phi(X)\right|\right|_2^2 + \mathbb{E}_{X,Y} \left|\left|\theta^{*T}\phi(X) - Y\right|\right|_2^2 + 2\mathbb{E}_{\theta_t, X, Y} \langle \theta_t^T \phi(X) - \theta^{*T}\phi(X), \theta^{*T}\phi(X) - Y \rangle \\
\leq& \mathbb{E}_X \left|\left|\phi(X)\right|\right|_2^2 \mathbb{E}_{\theta_t} \left|\left|\theta_t^T - \theta^{*T}\right|\right|_2^2 + R^* + 2\sqrt{\mathbb{E}_{\theta_t, X} \left|\left|\theta_t^T \phi(X) - \theta^{*T}\phi(X)\right|\right|_2^2 \mathbb{E}_{X,Y} \left|\left|\theta^{*T}\phi(X) - Y\right|\right|_2^2} \\
\leq& \frac{C_1}{2} \mathbb{E}_{\theta_t} \left|\left|\theta_t^T - \theta^{*T}\right|\right|_2^2 + R^* + 2\sqrt{\frac{C_1 R^*}{2} \mathbb{E}_{\theta_t} \left|\left|\theta_t^T - \theta^{*T}\right|\right|_2^2},
\end{aligned} \tag{2}$$

where the first inequality is by the Cauchy–Schwarz inequality and the second inequality is by the assumption.

Recall Equation (5.9),

$$\theta_t - \theta^* = (\theta_0 - \theta^*) e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t} + (\mathbf{I}_d - e^{-\frac{\eta}{m}\widetilde{\Phi}\widetilde{\Phi}^T t}) \widetilde{\Phi}^\dagger \mathbf{Z}.$$

By eigen-decomposition we have

$$\frac{1}{m}\widetilde{\Phi}\widetilde{\Phi}^T = V\Lambda V^T = \sum_{k=1}^{d} \mu_k v_k v_k^T,$$

where $\{v_k\}_{k=1}^d$ are orthonormal eigenvectors and $\{\mu_k\}_{k=1}^d$ are corresponding eigenvectors. Then, for each dimension k,

$$(\theta_{t,k} - \theta_k^*)^2 \leq 2(\theta_{0,k} - \theta_k^*)^2 e^{-2\eta\mu_k t} + 2(1 - e^{-\eta\mu_k t})^2 \frac{mC_2}{m\mu_k},$$

Taking expectation over θ_0 for both side, we have

$$\mathbb{E}_{\theta_0}\left(\theta_{t,k} - \theta_k^*\right)^2 \leq 2(\xi_k^2 + \theta_k^{*2}) e^{-2\eta\mu_k t} + 2(1 - e^{-\eta\mu_k t})^2 \frac{C_2}{\mu_k}. \tag{3}$$

Notich that the RHS in Equation 3 first monotonically decreases and then monotonically increases, so the maximum value of RHS is achieved either at $t = 0$ or $t \to \infty$. That is,

$$\mathbb{E}_{\theta_0}\left|\left|\theta_t^T - \theta^{*T}\right|\right|_2^2 \leq \sum_{k=1}^{d} 2\max\{\xi_k^2 + \theta_k^{*2}, \frac{C_2}{\mu_k}\}. \tag{4}$$

Plugging Equation 3 and Equation 2, we have

$$\begin{aligned} R_t \leq & \frac{C_1}{2}\mathbb{E}_{\theta_t}\left|\left|\theta_t^T - \theta^{*T}\right|\right|_2^2 + R^* + 2\sqrt{\frac{C_1 R^*}{2}\mathbb{E}_{\theta_t}\left|\left|\theta_t^T - \theta^{*T}\right|\right|_2^2} \\ \leq & R^* + C_1 \sum_{k=1}^{d} (\xi_k^2 + \theta_k^{*2}) e^{-2\eta\mu_k t} + (1 - e^{-\eta\mu_k t})^2 \frac{C_2}{\mu_k} + 2\sqrt{C_1 R^* \zeta}, \end{aligned}$$

where $\zeta = \sum_{k=1}^{d} \max\{\xi_k^2 + \theta_k^{*2}, \frac{C_2}{\mu_k}\}$. This concludes the proof. □