COMPETING LABELS: A HEURISTIC APPROACH TO PSEUDO-LABELING IN DEEP SEMI-SUPERVISED LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HAMDİ BURAK BAYRAK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
SCIENTIFIC COMPUTING

JANUARY 2022

Approval of the thesis:

COMPETING LABELS: A HEURISTIC APPROACH TO PSEUDO-LABELING IN DEEP SEMI-SUPERVISED LEARNING

submitted by **HAMDİ BURAK BAYRAK** in partial fulfillment of the requirements for the degree of **Master of Science in Scientific Computing Department, Middle East Technical University** by,

Prof. Dr. A. Sevtap Kestel _____
Dean, Graduate School of **Applied Mathematics**

Prof. Dr. A. Sevtap Kestel _____
Head of Department, **Scientific Computing**

Assoc. Prof. Dr. Şeyda Ertekin _____
Supervisor, **Computer Engineering Dept., METU**

Assoc. Prof. Dr. Hamdullah Yücel _____
Co-supervisor, **Scientific Computing, IAM, METU**

Examining Committee Members:

Prof. Dr. Ömür Uğur _____
Scientific Computing, IAM, METU

Assoc. Prof. Dr. Şeyda Ertekin _____
Computer Engineering Department, METU

Assoc. Prof. Dr. Ahmet Murat Özbayoğlu _____
Artificial Intelligence Engineering Department, TOBB ETU

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: HAMDİ BURAK BAYRAK

Signature :

ABSTRACT

COMPETING LABELS: A HEURISTIC APPROACH TO PSEUDO-LABELING IN DEEP SEMI-SUPERVISED LEARNING

Bayrak, Hamdi Burak

M.S., Department of Scientific Computing

Supervisor : Assoc. Prof. Dr. Şeyda Ertekin

Co-Supervisor : Assoc. Prof. Dr. Hamdullah Yücel

January 2022, 67 pages

Semi-supervised learning is one of the dominantly utilized approaches to reduce the reliance of deep learning models on large-scale labeled data. One mostly used method of this approach is pseudo-labeling. However, pseudo-labeling, especially its originally proposed form tends to remarkably suffer from noisy training when the assigned labels are false. In order to mitigate this problem, in our work, we investigate the gradient sent to the neural network and propose a heuristic method, called competing labels. In this method, we arrange the loss function and choose the pseudo-labels in a way that the gradient the model receives contains more than one negative element. We test our method on MNIST, Fashion-MNIST, and KMNIST datasets and show that our method has a better generalization performance compared to the originally proposed pseudo-labeling method.

Keywords: Semi-supervised learning, Deep learning, Pseudo-labeling, Machine learning

ÖZ

YARIŞAN ETİKETLER: YARI DENETİMLİ DERİN ÖĞRENMEDEKİ SÖZDE ETİKETLEMEYE BULUŞSAL BİR YAKLAŞIM

Bayrak, Hamdi Burak
Yüksek Lisans, Bilimsel Hesaplama Bölümü
Tez Yöneticisi : Doç. Dr. Şeyda Ertekin
Ortak Tez Yöneticisi : Doç. Dr. Hamdullah Yücel

Ocak 2022, 67 sayfa

Yarı denetimli öğrenme, derin öğrenme modellerinin büyük miktarda etiketli veriye olan bağımlılığını azaltmada yoğun şekilde faydalanılan yaklaşımlardan biridir. Bu yaklaşımda çoğunlukla kullanılan metot sözde etiketlemedir. Ancak, sözde etiketleme, özellikle orjinal formu, atanan etiketler hatalı olduğu zaman, gürültülü eğitimden önemli bir ölçüde olumsuz etkilenmektedir. Bu problemin etkisini azaltmak için çalışmamızda, sinir ağına gönderilen gradyanı inceliyor ve yarışan etiketler metodunu öne sürüyoruz. Bu metotta kayıp fonksiyonunu ve sözde etiketleri, modelin aldığı gradyanın birden fazla eksi eleman içermesini sağlayacak şekilde seçiyoruz. Metodumuzu MNIST, Fashion-MNIST, ve KMNIST veri kümelerinde test ediyor ve sunduğumuz metodun orjinal çalışmadaki metoda kıyasla daha iyi bir genelleştirme performansına sahip olduğunu gösteriyoruz.

Anahtar Kelimeler: Yarı-denetimli öğrenme, Derin öğrenme, Sözde etiketleme, Makine öğrenmesi

ACKNOWLEDGMENTS

I would like to give a special thanks to my thesis advisor Assoc. Prof. Şeyda Ertekin and co-advisor Assoc. Prof. Hamdullah Yücel for reviewing and sharing their valuable opinions about my thesis.

This work was partially supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) with the project "Development of Calorimeter Simulations in High Energy Particle Physics by Generative Adversarial Networks" (project no: 119F084).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Recent deep learning methods surpass the other machine learning algorithms in supervised learning tasks and such an extraordinary achievement is mostly explained by the fact that there are enormous amounts of labeled data which can be used in the training [15]. However, to collect or construct such a vast number of labeled data is not a straightforward task and is often costly because such a task requires time and expert knowledge.

In order to reduce the reliance on labeled data, many approaches have been invented. Some important ones among them are self-supervised learning [7], few-shot learning [8], and semi-supervised learning [4]. And semi-supervised learning is one of widely used approaches to handle datasets where the number of labeled data is very limited.

The objective in semi-supervised learning is to take advantage of samples whose labels are unavailable together with the labeled samples in order to improve the performance in a supervised learning task [3]. In semi-supervised learning one mostly accepted assumption is that the decision boundaries should not lie in high density regions but be in low density regions [4]. There are two dominant methods in semi-supervised learning that attempt to achieve this condition, namely consistency regularization and pseudo-labeling.

The main purpose of consistency regularization methods is to ask a model to produce "similar" outputs when an input and its perturbed form are given to the model [21]. In a more formal expression, when the input of a model is perturbed, the output should be invariant. For instance, in temporal ensembling method, the outputs for a sample

are accumulated for multiple epochs and the distribution of them is encouraged to be invariant [13].

Pseudo-labeling methods, on the other hand, follow entirely different strategy to overcome the limited label problem. In this approach, to be able to incorporate unlabeled samples into the training, some artificial labels, so called pseudo-labels, are assigned to each unlabeled samples, and the model is trained using both types of samples i.e., labeled ones (which have their true labels) and unlabeled ones (that have their pseudo-labels) [16].

In the original paper of pseudo-labeling, the author's pseudo-label selection is to assign only one label to a sample based on the sample's model output [16]. However, such a pseudo-label selection strategy comes with its own problem: When the assigned label is false, the model is trained with noisy data. As a consequence of this, the performance of the model (e.g., classification score of new samples) is degraded [1, 19].

To mitigate the negative effect (to the training) of originally proposed pseudo-label assignment strategy, there are many attempts in the literature and some important of them as follows:

In the work of Iscen et al. [11], the authors construct a graph in their training method, and based on neighborhood associated with this graph, they assign labels to the unlabeled samples.

In the work of Arazo et al. [1], a soft version of pseudo-labels (i.e., labels can have a value between 0 and 1, they are not restricted to binary choice) are taken into consideration and an intuitive loss function (i.e., which contains a lot of regularization terms) is attempted to minimize. This work is mainly based on the idea in the paper [20] where the authors construct a joint optimization problem in order to deal with noisy data. Furthermore, in order to increase the model's generalization ability, they utilize a popular regularization method, namely mixup [24].

Another solution to weaken the negative effect of the labeling strategy which is proposed in the original work is to select labels with "care". The authors in the paper [19] give labels to the unlabeled data if such samples' model outputs satisfy a determined

level of certainty condition. That is, the authors pick and label only certain amount of the samples that satisfy a pre-determined condition. Since at the first time, it is not possible to label all the unlabeled samples, the authors attempt to re-train the model by using newly pseudo-labeled samples. While training iteratively, the authors adopt a similar method expressed in the paper [9].

In this thesis, we study the single label image classification problem in the semi-supervised learning setting from the perspective of pseudo-labeling. However, unlike the previous works, in order to overcome the problem (i.e., selecting single label for an unlabeled sample causes noisy training, especially when the selection is false) associated with the originally proposed pseudo-labeling technique [16], we focus on the gradient that our model receives, and propose a heuristic method, called competing labels, where we design our loss function and determine the pseudo-labels in such a way that the gradient sent to the parametrized function contains more than one negative element—therefore, the model has a tendency to increase more than one of its (softmax) outputs. In our work, we take into consideration a class-balanced scenario and restrict ourselves to only a single model, namely a convolutional neural network with two hidden layers.

For this objective, we arrange the rest of the thesis as follows: in the next section, we propose some definitions and notations that are used in the coming chapters. In Chapter 2, we derive an expression for the gradient (or signal vector) that is sent to the model, and discuss what type of a loss function we should use in order to make the signal vector to be able to contain more than one negative signal. In Chapter 3, we focus on the selection of pseudo-labels, and propose a strategy for pseudo-label selection that allows both of these: the gradient has multiple negative elements and the magnitudes of these elements are greater than certain value. In Chapter 4, we test our approach (i.e., competing labels) on three different datasets, and show the better performance of our method against the "naive" pseudo-labeling method. Furthermore, in this chapter, we compare our outcomes with the results obtained in the originally proposed work. In the last chapter, we briefly give a summary and share important conclusions of our work.

1.1 Preliminaries: Notations and Definitions

In a typical semi-supervised learning task, we have two datasets: labeled and unlabeled. The labeled dataset D_L has the form

$$D_L = \left\{ (\boldsymbol{s}^{(i)}, \boldsymbol{y}^{(i)}) \right\}_{i=1}^{N_L},$$

where $\boldsymbol{s}^{(i)}$ is the sample (or input of the model), $\boldsymbol{y}^{(i)}$ is the (true) label vector, and N_L is the number of labeled samples. Since in our work, our interest is a K class image classification problem, the inputs, i.e., $\boldsymbol{s}^{(i)}$, turn out as images.

Moreover, because our problem is restricted to the single-label (or also known multi-class) classification, that is, for $(\boldsymbol{s}, \boldsymbol{y}) \in D_L$, the input $\boldsymbol{s}$ can belong to only one of $K > 2$ classes, an element y_j of the label vector $\boldsymbol{y}$ has the value $y_j = 1$ if $\boldsymbol{s}$ belongs to jth class and $y_j = 0$ otherwise, where $j = 1, 2, ..., K$. Note that, because of being single-label problem, only one element of the label vector $\boldsymbol{y}$ is 1, and the rest are 0.

The unlabeled dataset D_U, on the other hand, has an appearance:

$$D_U = \left\{ \boldsymbol{s}^{(i)} \right\}_{i=1}^{N_U},$$

where $\boldsymbol{s}^{(i)}$ and N_U are the ith sample and the number of samples of the unlabeled dataset D_U, respectively. One important distinction between D_L and D_U is that the elements of D_U do not include any label vector.

Because we employ the pseudo-labeling approach, the new (i.e., pseudo-labeled) unlabeled dataset $\widetilde{D}_U$ becomes

$$\widetilde{D}_U = \left\{ (\boldsymbol{s}^{(i)}, \tilde{\boldsymbol{y}}^{(i)}) \right\}_{i=1}^{N_U},$$

where the vector $\tilde{\boldsymbol{y}}^{(i)}$ is the pseudo-label vector of the ith sample. Note that in our problem we define the pseudo-label vector $\tilde{\boldsymbol{y}}$ in hard version; that is,

$$\tilde{\boldsymbol{y}} = \big[y_1, y_2, ..., y_K\big] \subseteq \{0, 1\}^K.$$

One remark is that, as opposed to the case in the labeled dataset, more than one elements of the pseudo-label vector can be 1 (remember it is just one in the label vector of the labeled dataset), and we use this property in the coming chapters.

Also, to represent the true class of a sample $\boldsymbol{s}$, we use the number $c \in \{1, 2, ..., K\}$. So, the cth element of the corresponding true label vector $\boldsymbol{y}$ is $y_c = 1$. Note that here $\boldsymbol{s}$ can belong to D_L or D_U.

In our work, we define our model or parametrized function $F_{\boldsymbol{\theta}} : \mathbb{R}^I \longrightarrow \mathbb{R}^K$ as

$$\boldsymbol{x} = F_{\boldsymbol{\theta}}(\boldsymbol{s}), \tag{1.1}$$

where, $\boldsymbol{\theta}$, I, and K are the parameter vector of the function, the input dimension, and the number of classes (defined on a classification problem), respectively. As assumed in almost all deep learning problems, $F_{\boldsymbol{\theta}}$ is differentiable with respect to the parameter vector $\boldsymbol{\theta}$.

Based on the work [2], we define the softmax function $S : \mathbb{R}^K \longrightarrow \mathbb{R}^K$ as $\boldsymbol{z} = S(\boldsymbol{x})$, where for $j = 1, 2, ..., K$,

$$z_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}. \tag{1.2}$$

Now, we list two important properties of this function that can be seen effortlessly by equation (1.2) and which we do not attempt to prove:

$$0 < z_i < 1, \text{ for every } i = 1, 2, ..., K. \tag{1.3a}$$

$$z_1 + z_2 + \cdots + z_K = 1. \tag{1.3b}$$

In our work, we define the loss function as $C : \mathbb{R}^{2K} \longrightarrow \mathbb{R}$. Note that in our definition the function C takes two arguments each of which is a K dimensional vector (this is why its domain dimension is $2K$) and maps to a scalar. More specifically, the function C takes the softmax output vector $\boldsymbol{z}$ and a label vector $\boldsymbol{y}$ (regardless of if the label

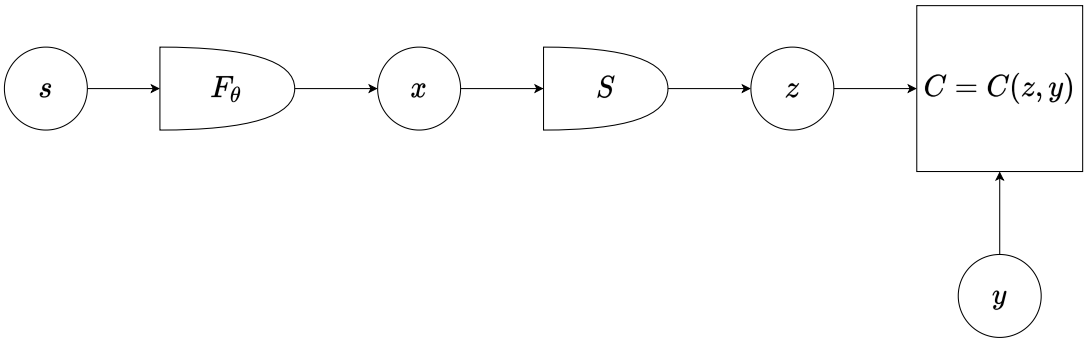


Figure 1.1: Forward propagation diagram

vector is a pseudo-label vector) and returns a value $C(\boldsymbol{z}, \boldsymbol{y})$. Once more, as usual, we assume the loss function C has a first order derivative with respect to the parameter vector $\boldsymbol{\theta}$.

Using the definitions above, we define the learning via pseudo-labeling in semi-supervised setting as to solve the following optimization problem:

$$\arg\min_{\boldsymbol{\theta}} \frac{1}{N_L + N_U} \sum_{i=1}^{N_L+N_U} C(\boldsymbol{z^{(i)}}, \tilde{\boldsymbol{y}}^{(\boldsymbol{i})}), \tag{1.4}$$

where in order to simplify the notation, we use $\tilde{\boldsymbol{y}}^{(\boldsymbol{i})} = \boldsymbol{y}^{(\boldsymbol{i})}$ if $\boldsymbol{s}^{(\boldsymbol{i})}$ is a labeled sample (otherwise we do not make any change, it is just a pseudo-label vector), and $\boldsymbol{z}^{(\boldsymbol{i})}$ is the softmax output vector for the sample $\boldsymbol{s}^{(\boldsymbol{i})}$; that is, $\boldsymbol{z}^{(\boldsymbol{i})} = (S \circ F_{\boldsymbol{\theta}})(\boldsymbol{s}^{(\boldsymbol{i})})$.

Bringing the all the functions together, for a sample $\boldsymbol{s}$, a typical forward propagation can be expressed as in Figure 1.1, where the capital letters are functions (i.e., $F_{\boldsymbol{\theta}}$, S, and C are the parametrized, softmax and loss functions, respectively) and the lower-case letters are the outputs of the functions except for $\boldsymbol{s}$ and $\boldsymbol{y}$. Note that at the end of this propagation in Figure 1.1, the loss function C takes not only the softmax output vector $\boldsymbol{z}$ but also it takes the corresponding label vector based on our definition of the loss function.

Since we optimize our loss function iteratively, we use square brackets to represent the jth iteration or step. For instance, to indicate the softmax output vector of ith sample at step j, we write $\boldsymbol{z}^{(i)[j]}$. In order to refer to the kth element of the same vector, we would write $z_k^{(i)[j]}$.

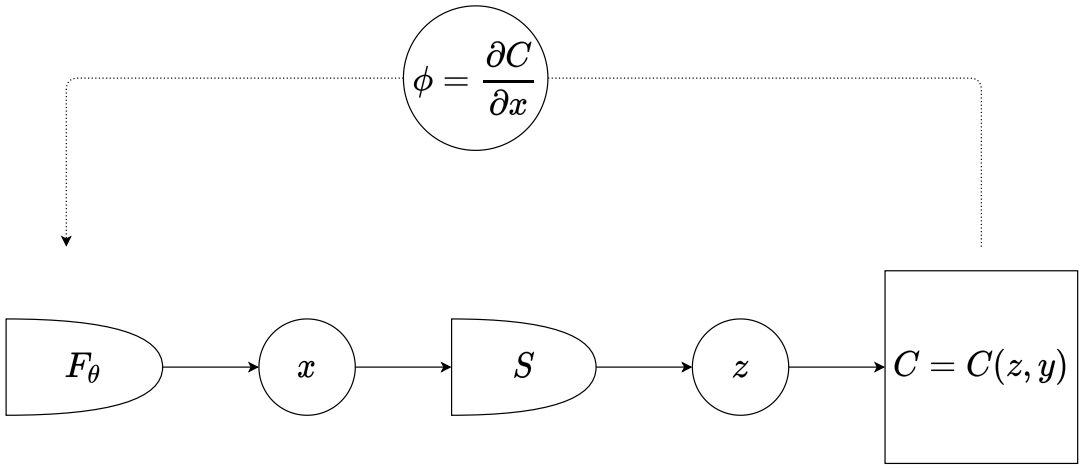


Figure 1.2: Signal vector propagation diagram

In our work, we define the concept noisy training as

$$z_c^{(i)[j]} > z_c^{(i)[j+1]}. \tag{1.5}$$

That is, the model encounters noisy training when the model produces a lower true softmax output z_c at the next iteration $j + 1$ compared to the iteration j. In order to measure the level of noisy training we consider how many samples N satisfy (1.5). If N is some large number, we would call it highly noisy training. Whereas if it is small, we say it is less noisy training.

Now, we define probably the most important concept of this thesis, namely signal vector. By the expression, the gradient $\boldsymbol{\phi}$ that the parametrized function $F_{\boldsymbol{\theta}}$ receives, or shortly the signal vector, we refer to the partial derivative of the loss function C with respect to the output vector $\boldsymbol{x}$ of the parametrized function $F_{\boldsymbol{\theta}}$ for a given sample $\boldsymbol{s}$. In a more symbolic form we express the signal vector $\boldsymbol{\phi}$ as

$$\boldsymbol{\phi} = \frac{\partial C}{\partial \boldsymbol{x}}. \tag{1.6}$$

An illustration of the signal vector $\boldsymbol{\phi}$ can be seen in Figure 1.2. In order to obtain this vector mathematically, when we write the partial derivative of the loss function with respect to the parameter vector (since we use gradient based optimization methods, we have to calculate this quantity), we have by the chain rule:

$$\frac{\partial}{\partial \boldsymbol{\theta}} C(\boldsymbol{z}, \boldsymbol{y}) = \frac{\partial C}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\theta}}, \tag{1.7}$$

where we use the fact that the label vector $\boldsymbol{y}$ is independent of the parameter vector $\boldsymbol{\theta}$. As clearly seen, the first factor on the right hand side of (1.7), is just the signal vector defined in (1.6), and the second factor is the partial derivative of the parametrized function with respect to the model parameter vector. We will give a more detailed analysis of the signal vector $\boldsymbol{\phi}$ and discuss how to make it contain many negative elements in the next chapter.

One terminology at that point is, as mentioned before, when we say "signal vector", we refer to the vector $\boldsymbol{\phi}$. Also, we use the term "signal" to mean an element of the signal vector. Similarly, when we use the plural form of it, "signals", we indicate more than one elements in the vector $\boldsymbol{\phi}$.

Another important concept we use throughout the thesis quite frequently is naive pseudo-labeling. Let $\boldsymbol{s}$ be a sample whose label is unavailable. Also let $\boldsymbol{z}$ be the softmax output vector, i.e., $\boldsymbol{z} = (S \circ F_{\boldsymbol{\theta}})(\boldsymbol{s})$. Then, by naive pseudo-labeling, we mean to assign a pseudo-label vector $\tilde{\boldsymbol{y}}$ to the unlabeled sample $\boldsymbol{s}$ according to the definition given in the original pseudo-label paper [16]; that is,

$$\tilde{y}_i = \begin{cases} 1, & \text{if } i = \arg\max_j z_j, \\ 0, & \text{otherwise.} \end{cases} \tag{1.8}$$

As seen in equation (1.8), the naive pseudo-labeling uses the softmax output vector $\boldsymbol{z}$ while constructing the pseudo-label vector $\tilde{\boldsymbol{y}}$. More specifically, such a labeling strategy returns a pseudo-label vector where all the elements are zero except for the element whose index is equal to the result of $\arg\max \boldsymbol{z}$. Again in the next section, we will see that such a labeling method leads to a signal vector that contains only one negative signal.

We now give accuracy for a set D. Let $D = \left\{(\boldsymbol{s^{(i)}}, \boldsymbol{y^{(i)}})\right\}_{i=1}^{N}$ be an arbitrary set containing sample and label vector pairs. Then, by the accuracy of the set D we mean the quantity acc given as

$$acc = \frac{1}{N}\sum_{i=1}^{N} Q\Big(\arg\max \boldsymbol{y^{(i)}} = \arg\max(S \circ F_{\boldsymbol{\theta}})(\boldsymbol{s^{(i)}})\Big),$$

where $Q(P)$ is a function and returns 1 if P is true and returns 0 otherwise.

Finally, we describe the mixup regularization method proposed in the work [24].

Let $A = [(\boldsymbol{s}^{(1)}, \boldsymbol{y}^{(1)}), ..., (\boldsymbol{s}^{(N)}, \boldsymbol{y}^{(N)})]$ be an N sample-label vector sequence. Also, let B be the shuffled version of A (i.e., the elements of A are randomly ordered). Then, to apply mixup regularization to the sequence A we mean to have the sample-vector sequence G which is a sequence obtained by the combination of elements of A and B where ith element $(g_1^{(i)}, g_2^{(i)})$ of G is given by

$$g_1^{(i)} = \gamma_i a_1^{(i)} + (1-\gamma_i) b_1^{(i)}, \tag{1.9a}$$

$$g_2^{(i)} = \gamma_i a_2^{(i)} + (1-\gamma_i) b_2^{(i)}, \tag{1.9b}$$

where $(a_1^{(i)}, a_2^{(i)})$ and $(b_1^{(i)}, b_2^{(i)})$ are ith elements of A and B, respectively, and γ_i is a number from the Beta distribution, i.e., $\gamma_i \sim Beta(a, b)$. Note that since in our problem, we restrict ourselves to the hard labels, we rearrange the the vector g_2 such that the greatest element of g_2 is 1 and the rest are zero.

CHAPTER 2

SIGNAL VECTOR

In this chapter, we analyze the signal vector defined in Preliminaries Section 1.1. Before starting we restate the functions and variables we use throughout this chapter.

For a sample, label vector pair $(\boldsymbol{s}, \tilde{\boldsymbol{y}}) \in D_L \cup \widetilde{D}_U$, the output vector $\boldsymbol{x}$ of the model $F_{\boldsymbol{\theta}}$ is defined as:

$$\boldsymbol{x} = F_{\boldsymbol{\theta}}(\boldsymbol{s}). \tag{2.1}$$

Also, the softmax output vector $\boldsymbol{z}$ is given as

$$\boldsymbol{z} = (S \circ F_{\boldsymbol{\theta}})(\boldsymbol{s}). \tag{2.2}$$

Note that we consider the parameter vector $\boldsymbol{\theta}$ and all the K dimensional vectors $\boldsymbol{x}$, $\boldsymbol{z}$, and $\tilde{\boldsymbol{y}}$ as column vectors.

Furthermore, by a loss function we mean a function $C : \mathbb{R}^{2K} \longrightarrow \mathbb{R}$ which has all properties described in Preliminary Section 1.1 defined as

$$C = C(\boldsymbol{z}, \tilde{\boldsymbol{y}}). \tag{2.3}$$

Here we note that the (true and pseudo) label vector is independent of the parameter vector $\boldsymbol{\theta}$; that is, $\frac{\partial \tilde{\boldsymbol{y}}}{\partial \boldsymbol{\theta}} = 0$.

To optimize the learning problem given in equation (1.4) (with a first order optimization method), one has to calculate the partial derivative of the loss function C with respect to the parameter vector $\boldsymbol{\theta}$, i.e., $\frac{\partial C(\boldsymbol{z}, \tilde{\boldsymbol{y}})}{\partial \boldsymbol{\theta}}$. By using the chain rule, one can ex-

press this derivative as:

$$\begin{aligned}\frac{\partial C(\boldsymbol{z},\tilde{\boldsymbol{y}})}{\partial \boldsymbol{\theta}} &= \frac{\partial C}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}} + \frac{\partial C}{\partial \tilde{\boldsymbol{y}}}\frac{\partial \tilde{\boldsymbol{y}}}{\partial \boldsymbol{\theta}} \\ &= \frac{\partial C}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\theta}},\end{aligned} \tag{2.4}$$

where we use the fact that the label vector is independent of the parameter vector. In fact, in order to obtain an expression in terms of the signal vector, one can further apply the chain rule to (2.4) by benefiting from equation (2.1):

$$\begin{aligned}\frac{\partial C(\boldsymbol{z},\tilde{\boldsymbol{y}})}{\partial \boldsymbol{\theta}} &= \frac{\partial C}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\theta}} \\ &= \frac{\partial C}{\partial \boldsymbol{x}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\theta}} \\ &= \boldsymbol{\phi}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\theta}},\end{aligned} \tag{2.5}$$

where $\boldsymbol{\phi} = \frac{\partial C}{\partial \boldsymbol{x}}$ is the signal vector as described in 1.1 (i.e., it contains 1 row and K columns), and the second factor in the right hand side of the equation above (i.e., $\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\theta}}$) is the Jacobian associated with the model (i.e., it contains K rows and the number of model parameters columns).

Even though we attempt to obtain an expression (of derivative of the loss function with respect to the model parameters) in terms of the signal vector, we have not explained why we even try to get such a formula. Now, in the next section, we give our motivation of doing this.

2.1 Motivation Behind Signal Vector

To express our motivation, we claim that the following hypothesis is true.

Let $\boldsymbol{s}$ be a sample and $\boldsymbol{z}^{[j]}$ be the corresponding softmax output vector at the jth iteration. Also let $\boldsymbol{\phi}^{[j]}$ be the corresponding signal vector at iteration j. Then, we claim that if the ith element of the signal vector at the iteration j is negative; that is $\phi_i^{[j]} < 0$, then the model tends to produce a greater ith softmax output in the next iteration; that is, $z_i^{[j]} < z_i^{[j+1]}$.

In fact, a similar hypothesis can be proposed for positive signals; that is, if the ith signal is positive at jth iteration, then the model tends to lower the ith softmax output

in the next iteration. However, for now, we restrict our attention to the negative signal case.

To support our claim, we restrict ourselves to the simplest and well-known first order optimization method, namely gradient descent. The optimization algorithm tries to minimize the loss $C(\boldsymbol{z}, \boldsymbol{y})$, and to do this, at each iteration, it makes the model parameters move in the direction where the maximum decrease in the loss occurs. However, how exactly does the optimizer "know" this maximum decrease information?

In one perspective, we can say that this information is stored in the signal vector $\boldsymbol{\phi}$. The reason results from the definition of this vector. By the definition of $\boldsymbol{\phi} = \frac{\partial C}{\partial \boldsymbol{x}}$, its each element ϕ_i represents how much the loss function C changes when a "unit" change occurs in the model output x_i. So, for instance, if the first element of the signal vector, i.e., ϕ_1 is negative and the rest of the elements are positive, the optimizer has a tendency to change the model parameters in a way that it increases x_1 in the next iteration (because the signal vector indicates that the loss reduces when x_1 increases) while it decreases the other $x_i, i \neq 1$ (because the positive signals are associated with the increase in the loss). As a result of this, we can expect that the corresponding softmax output, i.e., z_1 increases (since the softmax function in (1.2) is just a normalization function) because of the increase in x_1, and the others (i.e., $z_2, ..., z_K$) decrease.

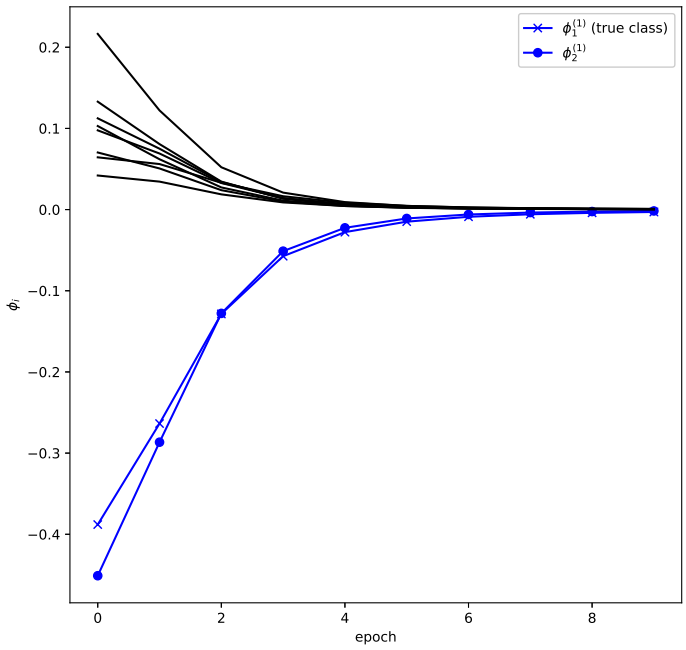
Similar intuitive reasoning can be applied for the case where the signal vector $\boldsymbol{\phi}$ comprises of more than one negative element, such as two with $\phi_1, \phi_2 < 0$ (and the rest are positive). In this case the gradient or the signal vector that the model receives contains two negative signals and this situation stimulates the optimizer to increase the corresponding model outputs (i.e., x_1, x_2). As a consequence of this, it is likely (not definitely) that the corresponding softmax outputs z_1, z_2 would increase (see figure 2.1).

Based on this hypothesis, we see that when we somehow manage to send a signal vector $\boldsymbol{\phi}$ whose true index is negative, i.e., $\phi_c < 0$, we can avoid the noisy training described in (1.5). In other expression, to avoid the noisy training, we think that the model should receive a signal vector where the true element of it is negative.

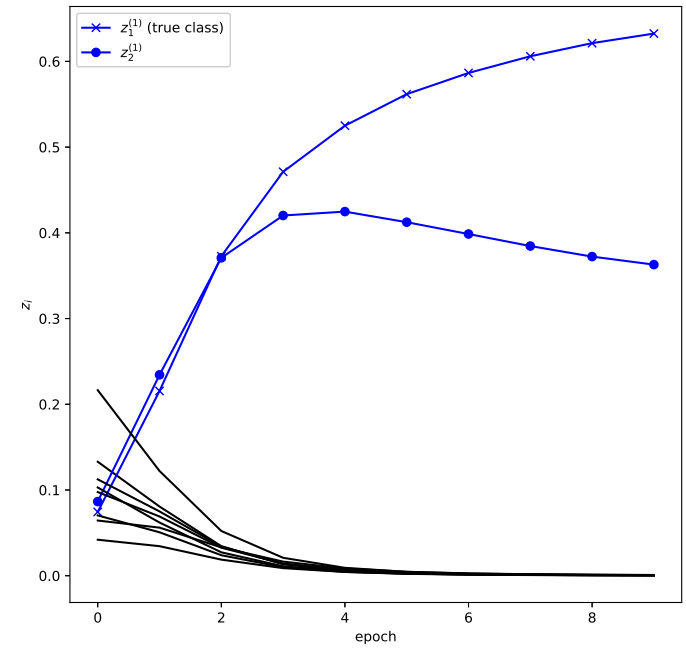
From the perspective of an unlabeled sample where we do not know what type of a signal vector we should construct, if we allow the corresponding signal vector to have only one negative signal, it is less likely to avoid noisy training since, for such avoidance, the index of negative signal and the true class are needed to be the same. In fact, this is the case for naive pseudo-labeling; with this labeling—and the cross entropy loss— the signal vector can only include one negative signal which we discover in Section 2.3.

On the other hand, if we allow the signal vector to have more negative signals, we would increase the chance that one of the negative signals that is associated with the true signal ϕ_c is negative. Hence, based on our hypothesis above, we would increase the chance of avoiding noisy training. In fact, this is the idea on which our heuristic method, i.e., competing labels, is based (to obtain such a signal vector, i.e., containing more negative signals, we propose to use a different loss function and labeling technique compared to the naive one about which we give more details in the coming sections of this chapter).

After expressing our motivation, we move to the next section in order to derive some important properties of the signal vector. We test our hypothesis with some experiments in the last section of this chapter.



(a) Signals vs epoch



(b) Softmax outputs vs epoch

Figure 2.1: Change of signals and softmax outputs during epochs for a single sample. Black and colored lines are associated with the positive and negative signals, respectively.

2.2 Signal Vector Analysis

The signal vector $\boldsymbol{\phi}$ can be expressed as indicated in (2.5):

$$\boldsymbol{\phi} = \boldsymbol{\alpha}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}, \tag{2.6}$$

where we define $\boldsymbol{\alpha} = \frac{\partial C}{\partial \boldsymbol{z}}$. In this equation, the vector $\boldsymbol{\alpha}$ is a $1 \times K$ dimensional vector and the second factor on the right hand side is the Jacobian associated with the softmax function. In order to calculate the signal vector, we now first focus on the Jacobian and state the following proposition:

Proposition 1 (Derivative of softmax function)**.** *Let* $\mathbf{x}$ *be in* $\mathbb{R}^K$*. And let* $\mathbf{z}$ *be the output vector of the softmax function* S*, i.e.,* $\mathbf{z} = S(\mathbf{x})$*. Then, the Jacobian* $\frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ *is*

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \begin{bmatrix} (1-z_1)z_1 & -z_1z_2 & \dots & -z_1z_K \\ -z_2z_1 & (1-z_2)z_2 & \dots & -z_2z_K \\ \vdots & \vdots & \vdots & \vdots \\ -z_Kz_1 & -z_Kz_2 & \dots & (1-z_K)z_K \end{bmatrix}, \tag{2.7}$$

where $\frac{\partial z_i}{\partial x_j} = (1-z_i)z_i$ *for* $i = j$*, and* $\frac{\partial z_i}{\partial x_j} = -z_iz_j$ *for* $i \neq j$*.*

Proof. For $i = j$, we have

$$\begin{aligned} \frac{\partial z_i}{\partial x_i} &= \frac{\partial}{\partial x_i}\frac{e^{x_i}}{\sum_k e^{x_k}} \\ &= e^{x_i}/\sum_k e^{x_k} - e^{x_i}e^{x_i}/\big(\sum_k e^{x_k}\big)^2 \\ &= z_i - z_i^2 \\ &= (1-z_i)z_i. \end{aligned}$$

For $i \neq j$, on the other hand,

$$\begin{aligned} \frac{\partial z_i}{\partial x_j} &= \frac{\partial}{\partial x_j}\frac{e^{x_i}}{\sum_k e^{x_k}} \\ &= 0 - e^{x_i}e^{x_j}/\big(\sum_k e^{x_k}\big)^2 \\ &= -z_iz_j. \end{aligned}$$

□

Now, we give an expression of a signal ϕ_i in terms of $\boldsymbol{\alpha}$ and $\boldsymbol{z}$.

Proposition 2 (Value of signal)**.** *Let the signal vector $\boldsymbol{\phi}$ be in* (2.6)*. Then, a signal ϕ_i in the signal vector $\boldsymbol{\phi}$ can be expressed as*

$$\phi_i = z_i(\alpha_i - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T), \tag{2.8}$$

where the operation $\cdot$ is the dot product.

Proof. To find the signal ϕ_i we multiply the vector $\boldsymbol{\alpha}$ with the ith column of the Jacobian given in (2.7); that is,

$$\begin{aligned}
\phi_i &= \boldsymbol{\alpha} \cdot [-z_1 z_i, ..., -z_{i-1} z_i, (1 - z_i) z_i, -z_{i+1} z_i, ..., -z_K z_i] \\
&= z_i \boldsymbol{\alpha} \cdot [-z_1, ..., -z_{i-1}, (1 - z_i), -z_{i+1}, ..., -z_K] \\
&= z_i \Big(- \sum_{i=1}^{K} \alpha_i z_i + \alpha_i \Big) \\
&= z_i(\alpha_i - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T).
\end{aligned}$$

□

Next, we state the signal conservation property.

Theorem 1 (Signal conservation)**.** *Let the signal vector $\boldsymbol{\phi}$ be defined as in* (2.6)*. Then, the quantity $\phi_1 + \phi_2 + \cdots + \phi_K$ is conserved and equal to zero; that is,*

$$\sum_{i=1}^{K} \phi_i = 0. \tag{2.9}$$

Proof. To prove this statement we benefit from value of signal Proposition 2.

$$\begin{aligned}
\phi_1 + \phi_2 + \cdots + \phi_K &= z_1(\alpha_1 - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T) + z_2(\alpha_2 - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T) + \cdots + z_K(\alpha_K - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T) \\
&= \sum_{i=1}^{K} \alpha_i z_i - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T \sum_{i=1}^{K} z_i \\
&= \boldsymbol{\alpha} \cdot \boldsymbol{z}^T - \boldsymbol{\alpha} \cdot \boldsymbol{z}^T \sum_{i=1}^{K} z_i \\
&= 0,
\end{aligned} \tag{2.10}$$

where at the last step we use the property of the softmax function in (1.3). □

At that point, we make some comments about the signal conservation Theorem 1. This theorem says that we cannot have arbitrary signals in ϕ. For instance, we do not encounter a case where all the signals are negative. Similarly, we cannot construct a signal vector where all elements are positive. From the perspective expressed in Section 2.1, a parametrized function cannot receive a gradient that make the model have a tendency to increase (or decrease if the elements are positive) all the softmax outputs in the next iteration.

Another important point we underline here is that Theorem 1 is independent of the loss function; that is, after the softmax layer we can use any loss function whose first derivative exists and it must obey this rule.

Now we move on the next section to figure out what type of a signal vector the naive pseudo-labeling leads to.

2.3 Signal Vector of Naive Pseudo-Labeling

In our discussion, we restrict ourselves to a loss function which is the mostly used in classification task, namely cross-entropy loss, as done in originally proposed pseudo-labeling paper [16]. Cross-entropy loss (of categorical version) for a sample label vector pair $(\boldsymbol{s}, \tilde{\boldsymbol{y}})$ is defined as [1]:

$$C = -\sum_{i=1}^{K} \tilde{y}_i \log z_i, \tag{2.11}$$

where $\boldsymbol{z}$ is the softmax output vector (of the input $\boldsymbol{s}$). In order to calculate the signal vector of the naive pseudo-labeling, we apply the definition given in (1.8) where only one label is 1 and the others are zero in the vector $\tilde{\boldsymbol{y}}$. Now, without loss of generality, we assume that the first element of the label vector is 1, i.e., $\tilde{y}_1 = 1$ and the rest are zero, i.e., $\tilde{y}_i = 0$, for $i = 2, ..., K$. Then, the cross entropy loss in equation (2.11) becomes:

$$C = -\log z_1. \tag{2.12}$$

When we calculate the derivative of the loss in equation (2.12) with respect to the softmax output vector $\boldsymbol{z}$, the result turns out $\alpha_1 = -1/z_1$ and $\alpha_i = 0$ for $i = 2, ..., K$. Furthermore, the product $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T$ becomes $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T = -1$. Therefore, by employing Proposition 2, we have the signal vector for naive pseudo-labeling as

$$\boldsymbol{\phi} = \begin{bmatrix} -(1 - z_1) & z_2 & \cdots & z_K \end{bmatrix}. \tag{2.13}$$

As seen from equation (2.13), the signal vector $\boldsymbol{\phi}$ contains only one negative signal, i.e., ϕ_1 and the all other elements are positive. Moreover, this negative element occurs at the index j in the signal vector for which $\tilde{y}_j = 1$. Furthermore, one can also test the signal conservation rule in this setting:

$$\begin{aligned} \phi_1 + \phi_2 + \cdots + \phi_K &= z_1 + z_2 + \cdots + z_K - 1 \\ &= 0, \end{aligned}$$

where we use the property of softmax function given in equation (1.3). As discussed in Section 2.1, it is unlikely that giving only one negative signal to the model allows to avoid noisy learning, especially, when the given signal is false. Now, to mitigate this issue, we attempt to obtain a scenario where not one but many elements of signal vector are negative.

2.4 Obtaining More Negative Signals

In this section, we discuss how we can construct a signal vector which includes many negative elements.

One possible attempt to achieve this purpose is to set two elements of the pseudo-label vector as 1, e.g., $\tilde{y}_1, \tilde{y}_2 = 1$. Under this condition the cross entropy loss function defined in (2.11) becomes:

$$C = -\log z_1 - \log z_2. \tag{2.14}$$

We have $\boldsymbol{\alpha} = [-1/z_1, -1/z_2, 0, ..., 0]$ and $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T = -2$ when calculating the vector

$\boldsymbol{\alpha}$ and the product $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T$. Using Proposition 2, we obtain the signals as $\phi_1 = z_1\Big(- 1/z_1 + 2\Big)$ and $\phi_2 = z_2\Big(- 1/z_2 + 2\Big)$. However, such a strategy in order to obtain negative signals would not allow us to necessarily obtain negative elements. For instance, for $z_1 > 0.5$, the signal ϕ_1 would be positive because $z_1 > 0$ (by the property of the softmax function (1.3)) and $\Big(- 1/z_1 + 2\Big) > 0$ (hence $\phi_1 > 0$).

To be able to have negative elements in the signal vector, we keep utilizing the strategy of setting many elements of the pseudo-label one. However, at this point we use the following type of loss function to reach our goal:

$$C = -\log\big(\boldsymbol{z} \cdot \tilde{\boldsymbol{y}}\big). \tag{2.15}$$

Now, we calculate this loss function for the same pseudo-label vector, i.e., $\tilde{\boldsymbol{y}} = [1, 1, 0, ..., 0]^T$. After inserting such a label vector into equation (2.15), we have

$$C = -\log(z_1\tilde{y}_1 + z_2\tilde{y}_2). \tag{2.16}$$

To be able to find the signal vector resulting from the loss (2.16), we calculate an element α_i in the vector $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T$ and they turn out as $\alpha_i = -1/\Big(z_1 + z_2\Big)$ for $i = 1, 2$ and $\alpha_i = 0$ for $i \neq 1, 2$, and $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T = -1$. Using once again Proposition 2, we have $\phi_i = z_i\Big(- 1/\big(z_1 + z_2\big) + 1\Big)$ for $i = 1, 2$ and $\phi_i = z_i$ for $i \neq 1, 2$.

Notice that in this setting we no longer suffer from positive signal issue that previously occurred for the cross-entropy loss function. The reason is that $z_1 + z_2 < 1$ (see (1.3)), hence $-1/\big(z_1 + z_2\big) + 1 < 0$. Furthermore, for those ϕ_i where $\tilde{y}_i = 0$, the signal becomes positive (i.e., $\phi_i = z_i$).

Now, we attempt to show that not only two but we can obtain $\phi_i < 0$ for all those elements of the pseudo-label vector $\tilde{\boldsymbol{y}}$ with $\tilde{y}_i = 1$ when we use the loss function provided in equation (2.15).

Let I be a nonempty proper subset of the full index set $\{1, 2, ..., K\}$, i.e., $I \subset \{1, 2, ..., K\}$. Let $\tilde{y}_i = 1$ for every $i \in I$ and $\tilde{y}_i = 0$ otherwise. Then, the loss function becomes

$$C = -\log\Big(\sum_{i\in I} z_i\Big). \tag{2.17}$$

Then, the element α_i of $\boldsymbol{\alpha}$ is found as:

$$\alpha_i = \begin{cases} -1/\Big(\sum_{i\in I} z_i\Big), & \text{if } i \in I \\ 0, & \text{otherwise} \end{cases}. \tag{2.18}$$

Also, using (2.18) the product $\boldsymbol{\alpha} \cdot \boldsymbol{z}^T$ becomes

$$\begin{aligned} \boldsymbol{\alpha} \cdot \boldsymbol{z}^T &= \alpha_1 z_1 + \alpha_2 z_2 + ... + \alpha_K z_K \\ &= -\sum_{i\in I} z_i / \sum_{i\in I} z_i \\ &= -1. \end{aligned} \tag{2.19}$$

Therefore, applying Proposition 2, we have an expression for an arbitrary signal ϕ_i as:

$$\phi_i = \begin{cases} z_i\Big(-1/\big(\sum_{i\in I} z_i\big) + 1\Big), & \text{if } i \in I \\ z_i, & \text{otherwise} \end{cases}. \tag{2.20}$$

As clearly seen from equation (2.20), for those indices i for which $\tilde{y}_i = 1$, $\phi_i < 0$ and for those i with $\tilde{y}_i = 0$, the signals are positive.

Moreover, from the same equation, we observe that the amplitude of a signal, i.e., $|\phi_i|$ is directly proportional to z_i. For instance, among negative signals (i.e., $i \in I$), whenever $z_i > z_j$, we have $|\phi_i| > |\phi_j|$. Since ith signal has a greater amplitude, we may expect that the optimizer would favor the increase of z_i more compared to the element z_j in the next iteration in order to reduce the loss. A similar reasoning may also be valid for positive signals. For $z_i > z_j$, the optimizer tries to reduce z_i more in the next iteration in order to reach a minimum point of the loss.

Notice that for a label vector whose only one element is 1, the loss functions described in (2.11) and (2.15) produce the same result. For example, for $y_1 = 1$ and $y_i = 0$ for

$i = 2, ..., K$, the cross-entropy loss in (2.11) produces $C = -\log z_1$. On the other hand, since $\boldsymbol{z} \cdot \boldsymbol{y} = z_1$, the loss in (2.15) becomes $C = -\log z_1$. Based on this observation, we can say that the loss functions in (2.11) and (2.15) produce the same scalar for the true label vector and the pseudo-label vector (see (1.8)) because only single element in each of these vectors is 1.

In the coming section, we conduct some experiments on the loss function provided in equation (2.15) to understand its behaviour better and to see the relationship between the negative signals and the softmax outputs.

2.5 Negative Signals Experiments

In this section, we investigate the relationship between the sent signals to the model and the change in the softmax outputs corresponding to these signals. During our experimentation session, we only focus on one and four samples cases and these samples are given in Figures 2.2 and 2.3. These images belong to one of $K = 10$ class categories. Furthermore, we use Adam optimization method with a learning rate 2×10^{-4} and train our model with a fixed number of epoch which is 10.

Also, in our experiments, we take into consideration three different number of negative signals, namely one, two, and three. To construct these signals, we employ the loss function we have proposed in equation (2.15) together with the appropriate pseudo-label vectors. In each case, we use one and four samples described above to investigate signal vector and softmax output vector relationship.

2.5.1 One Negative Signal

2.5.1.1 One Sample

In this one negative signal, one sample case, we use the image given in Figure 2.2, as a sample and set its pseudo-label vector as $\tilde{y}_1 = 1$ and $\tilde{y}_i = 0$ for $i \neq 1$ as also shown in Table 2.1.

Utilizing such a label vector, we obtain signals and softmax outputs for the sample for

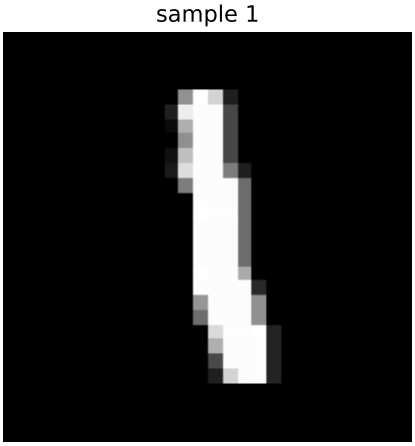


Figure 2.2: One sample for experimentation. The image is taken from the MNIST dataset [6].

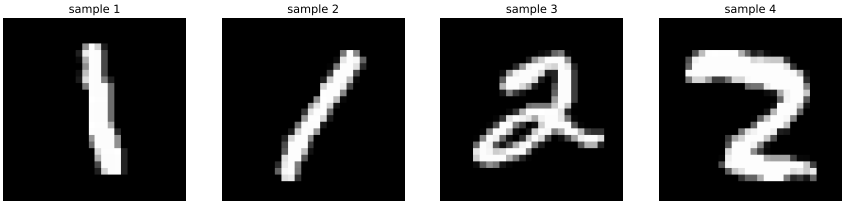


Figure 2.3: Four samples for experimentation. The images are taken from the MNIST dataset [6].

Table 2.1: One negative signal, one sample pseudo-labels

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
$\boldsymbol{s}^{(1)}$	1	0	0	0	0	0	0	0	0	0

each epoch as given in Figure 2.4. The graph on the left in Figure 2.4 shows softmax outputs values during each epoch while the graph on the right displays the signal values. The values at epoch zero indicates the initial values. That is, for softmax outputs, it is the value when the sample is calculated with the initial model parameters. In terms of signal, epoch zero represents the gradient or ϕ that is sent to the model to change the parameters in the initial epoch. After parameters are updated in the initial epoch (i.e., epoch zero), the values of softmax outputs are expressed in epoch 1 as seen in the same Figure.

Our first observation in Figure 2.4 is that there is only one negative signal (represented with a blue color) and the rest are positive as predicted analytically in the previous

section. We also observe that giving negative signal to the model encourages it to produce higher values for the corresponding softmax output (i.e., z_1). Whereas the positive ones cause to change in the model parameters such that the corresponding outputs decrease. Furthermore, we see that the amplitude of the negative signal is relatively large compared to the positive ones. This is because the signal value of ϕ_1 is equal to $-(1 - z_1)$ while the others are z_i.

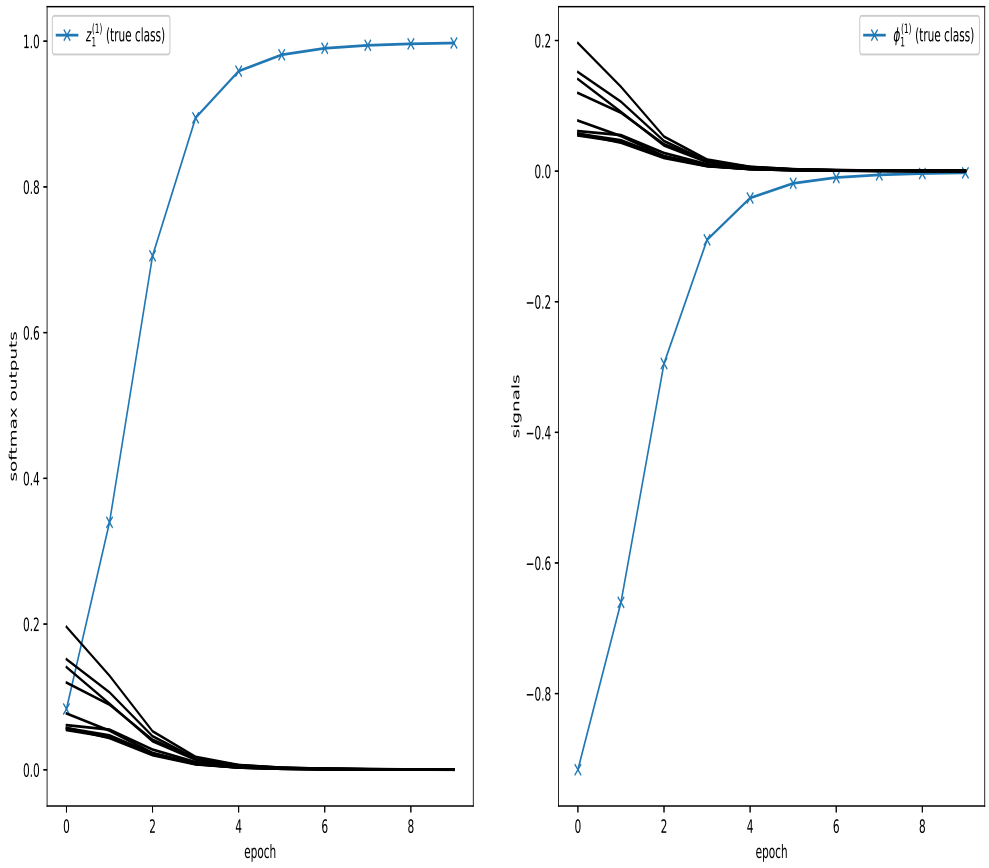


Figure 2.4: Behaviour of softmax outputs and signals during the training. Black, and colored lines associated with the values 0 and 1 in the label vector, respectively.

2.5.1.2 Four Samples

In this case, we use the four images given in Figure 2.3 as inputs. Moreover, for these inputs we use the label vectors which are given in Table 2.2. When we train our model

with these labels and images, we obtain the relationship between softmax outputs and signals in Figure 2.5.

Table 2.2: One negative signal, four samples, pseudo-labels

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
$\boldsymbol{s}^{(1)}$	1	0	0	0	0	0	0	0	0	0
$\boldsymbol{s}^{(2)}$	1	0	0	0	0	0	0	0	0	0
$\boldsymbol{s}^{(3)}$	0	1	0	0	0	0	0	0	0	0
$\boldsymbol{s}^{(4)}$	0	1	0	0	0	0	0	0	0	0

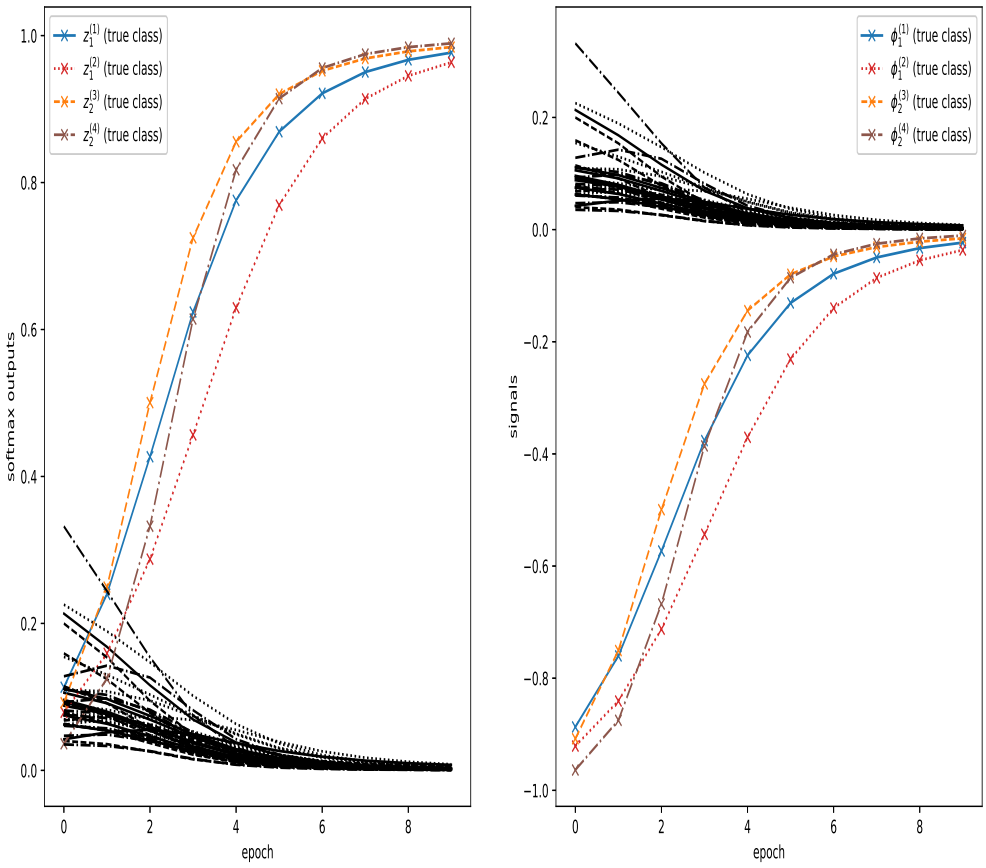


Figure 2.5: Four samples with one negative signals. Black, and colored lines associated with the values 0 and 1 in the label vector, respectively.

Figure 2.5 shows how the softmax outputs and signals evolve during the epochs and each color represents a different input. As seen in the same plot, each sample's signal vector contains only one negative element. As we have already stated those elements are determined by the pseudo-label vector; that is, if a element $\tilde{y}_i = 1$, then the cor-

responding signal value is negative. And those elements increase the corresponding softmax outputs. We also observe that the same magnitude situation (i.e., the absolute value of negative signal is remarkably greater than the positive one) is valid for four samples case regardless of the sample.

2.5.2 Two Negative Signals

2.5.2.1 One Sample

In this case, we set the pseudo-label vector in a way that we obtain two negative signals. We perform this operation by setting the vector as in Table 2.3. When we train our model with one input and the determined pseudo-label vector, we obtain a relationship between softmax outputs and the signals for each epoch as given in Figure 2.6.

Table 2.3: Pseudo-labels for two negative signals

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
$\boldsymbol{s}^{(1)}$	1	1	0	0	0	0	0	0	0	0

As seen in Figure 2.6, there are two negative signals, namely ϕ_1 and ϕ_2, which result from the labeling. Also, we observe that these two negative signals increase the corresponding softmax outputs (i.e., z_1 and z_2) while the epochs increase. However, after a certain epoch, the increase no longer occur since the corresponding signals die (i.e., the blue curves in the figure). Moreover, we observe for the softmax outputs with positive signals have a tendency to decrease (i.e., the black curves in the same figure). Furthermore, we observe that with the two negative signals case, the magnitudes of the negative signals decrease compared to the case with one negative signal. The reason is that according to the signal conservation theorem, the sum of magnitudes of negative signals (i.e., $|\phi_1| + |\phi_2|$) is restricted by the positive signals magnitudes' sum (i.e., $|\phi_3|+\cdots+|\phi_K|$). So, the negative signals are shared, hence the magnitude decreases. Finally, when the magnitude of the negative singal is greater (e.g., $|\phi_1| > |\phi_2|$), the optimizer has a tendency to increase the corresponding softmax output more in the next epochs (e.g., $z_1 > z_2$ during the epochs).

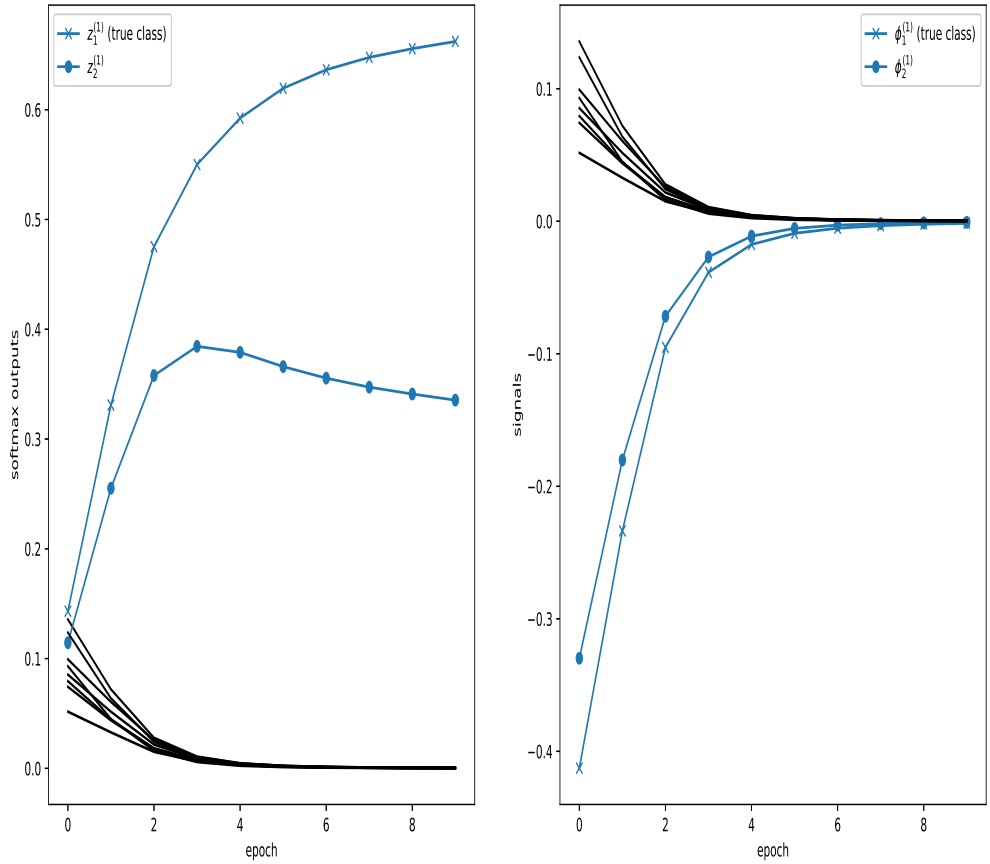


Figure 2.6: One sample with two negative signals. Black, and colored lines associated with the values 0 and 1 in the label vector, respectively.

2.5.2.2 Four Samples

We now consider the four samples case. To obtain two negative signals for each of the samples, we set the label vectors as given in Table 2.4. After training in this setting we obtain Figure 2.7.

Table 2.4: Pseudo-labels for two negative signals and four samples

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
$\boldsymbol{s}^{(1)}$	1	0	1	0	0	0	0	0	0	0
$\boldsymbol{s}^{(2)}$	1	0	0	1	0	0	0	0	0	0
$\boldsymbol{s}^{(3)}$	0	1	0	0	1	0	0	0	0	0
$\boldsymbol{s}^{(4)}$	0	1	0	0	0	1	0	0	0	0

As seen in Figure 2.7, based on the label vector, those signals turn out as negative (i.e., if the element is 1 in the label vector, then the corresponding signal is negative). Moreover, when the signal is negative, the corresponding softmax output tends to increase as observed in the previous experiments. Furthermore, when the magnitude of the negative signal is greater (e.g., $|\phi_4^{(2)}| > |\phi_1^{(2)}|$), the optimizer favors the increase of the corresponding softmax output (e.g., $z_4^{(2)} > z_1^{(2)}$) in the next iterations.

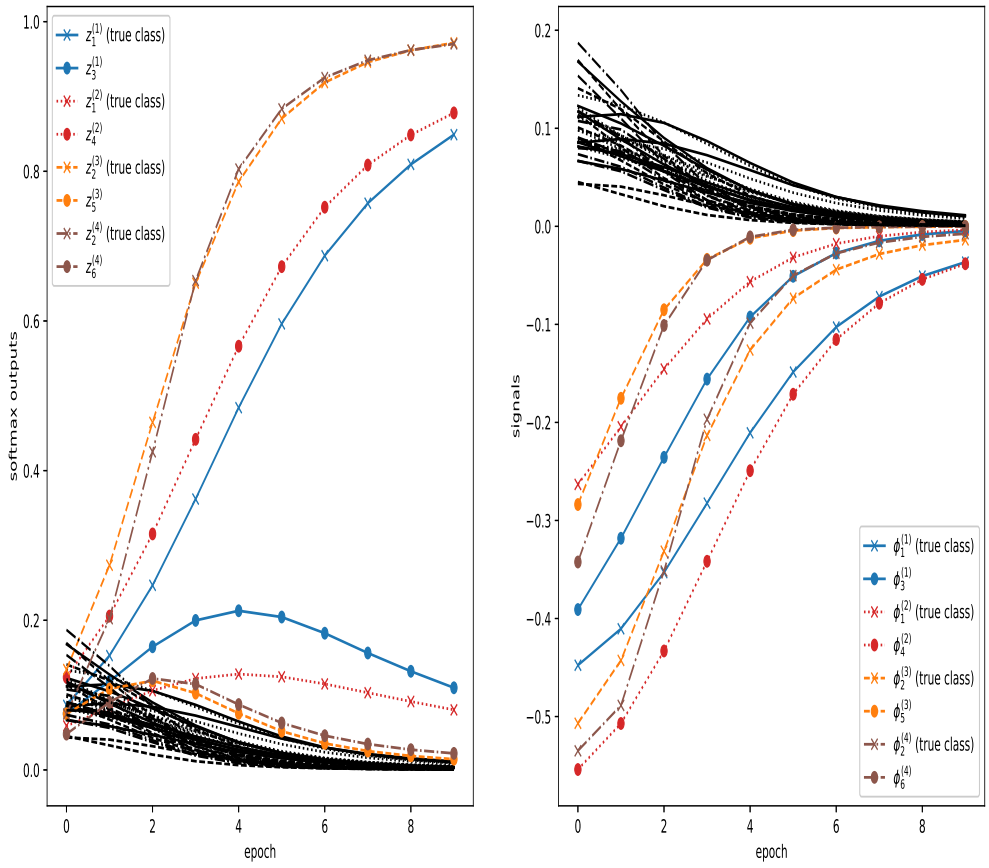


Figure 2.7: Two negative signals with four samples. Black, and colored lines associated with the values 0 and 1 in the label vector, respectively.

2.5.3 Three Negative Signals

2.5.3.1 One Sample

In our final negative signal case, we first focus on one sample situation with the label vector given in Table 2.5.

Table 2.5: Pseudo-labels for three negative signals and one sample

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
$\boldsymbol{s}^{(1)}$	1	1	1	0	0	0	0	0	0	0

After training in this setting, we obtain the results displayed in Figure 2.8. As seen in the Figure, the negative signals increase the corresponding softmax outputs. Also, we observe the same behaviour in the absolute value of the negative signal: when the number of negative signal increases, the each negative signal magnitude reduces. This is actually expected because for a sum of positive signals, there must be in the same amount but in negative signals by the signal conservation rule. So, the signal is simply shared among the negative ones, hence the magnitude decreases. Moreover, the optimizer favors the softmax output with the greater magnitude of negative signal (e.g., $|\phi_1| < |\phi_2|$ results in $z_1 < z_2$ in the successive epochs).

2.5.3.2 Four Samples

Finally, we investigate three negative signals situation with four samples. We use the pseudo-labels given in Table 2.6. After the training, we have Figure 2.9. Again, we observe that depending on the labels of the inputs, the corresponding signals become negative; those signals tend to stimulate the optimizer to increase the corresponding softmax outputs.

Table 2.6: Pseudo-labels for three negative signals and four samples

	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}
$\boldsymbol{s}^{(1)}$	1	1	1	0	0	0	0	0	0	0
$\boldsymbol{s}^{(2)}$	1	0	0	1	1	0	0	0	0	0
$\boldsymbol{s}^{(3)}$	0	1	1	0	0	0	1	0	0	0
$\boldsymbol{s}^{(4)}$	0	1	0	1	0	0	0	0	1	0

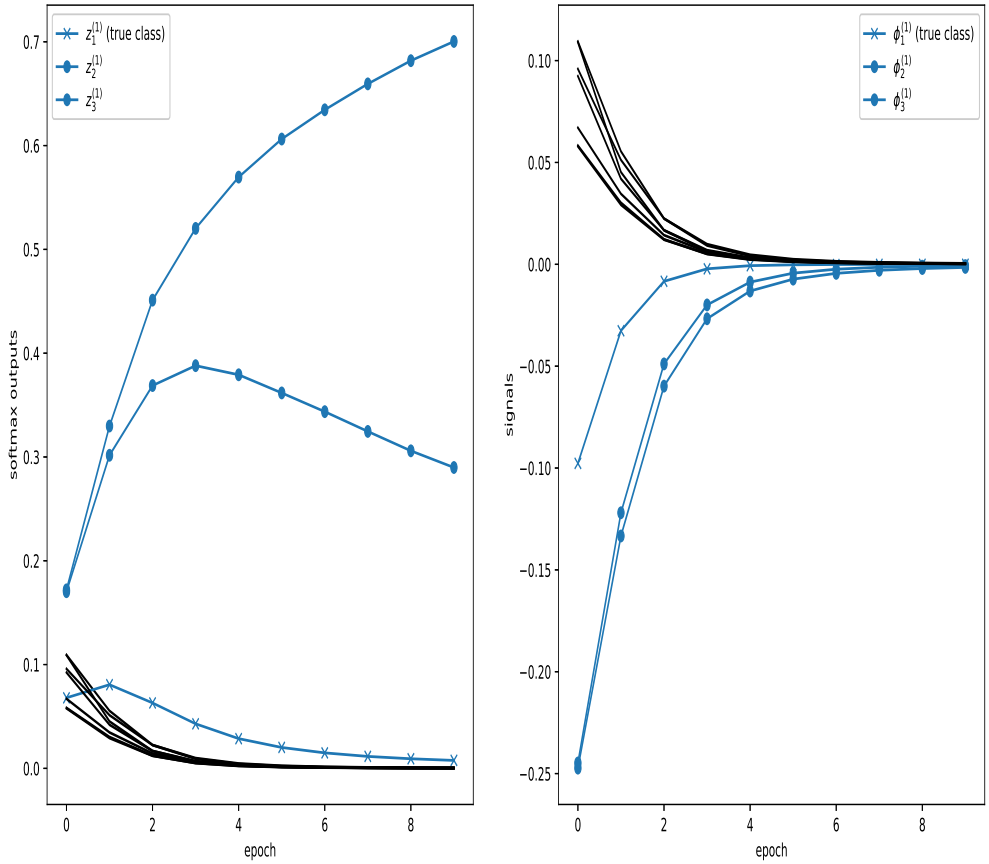


Figure 2.8: Softmax outputs and signals for three negative signals and one input. Black, and colored lines associated with the values 0 and 1 in the label vector, respectively.

So far, we have discussed how to construct a signal vector with many negative signals, that is, using a loss in type $-\log(\boldsymbol{z}\cdot\tilde{\boldsymbol{y}})$ and a pseudo-label vectors with many 1 entries. However, we haven't indicated how to choose those pseudo-labels (i.e., which indices we should determine as 1 in the vector $\tilde{\boldsymbol{y}}$). In the next chapter we investigate this problem.

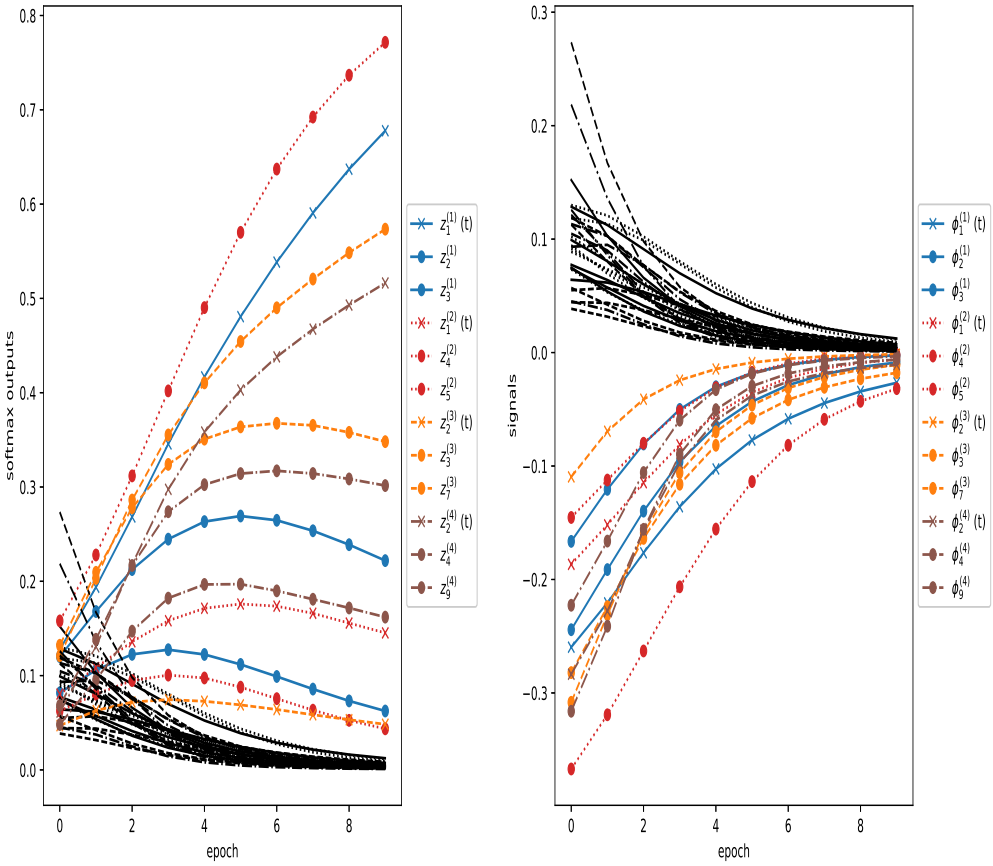


Figure 2.9: Softmax outputs and signals for four samples with three negative signals. Black, and colored lines associated with the values 0 and 1 in the label vector, respectively.

CHAPTER 3

PSEUDO-LABEL SELECTION

In the previous chapter (i.e., Chapter 2), we find out a way to construct many negative signals: we manage this by using the loss function having the form $C = -\log(\boldsymbol{z} \cdot \tilde{\boldsymbol{y}})$ and allowing the vector $\tilde{\boldsymbol{y}}$ to have many 1 for its elements $\tilde{y}_i$. However, we have not determined how to fill this pseudo-label vector; i.e., which indices should be 1.

To determine the pseudo-labels by the naive pseudo-labeling method, for instance, would lead to highly noisy training, especially at the beginning of the training. The reason is that the model at the beginning produces softmax outputs z_i randomly, so it would be very likely to lead to a false choice of label.

To illustrate this random distribution at the initial state, we utilize 100 (MNIST [6], class-balanced) samples, and plot the histogram of each sample's softmax output (i.e., histogram of the set $\{z_k^{(i)}\}, i = 1, ...100, k = 1, ..., 10$). Figure 3.1 shows that the softmax outputs are located around $1/K = 0.1$ and it is very unlikely that the index of the greatest z_i for a sample $\boldsymbol{s}$ represents the true class of that sample.

To overcome this problem, there is a popular technique [1, 11], called warm-up, and now we introduce this.

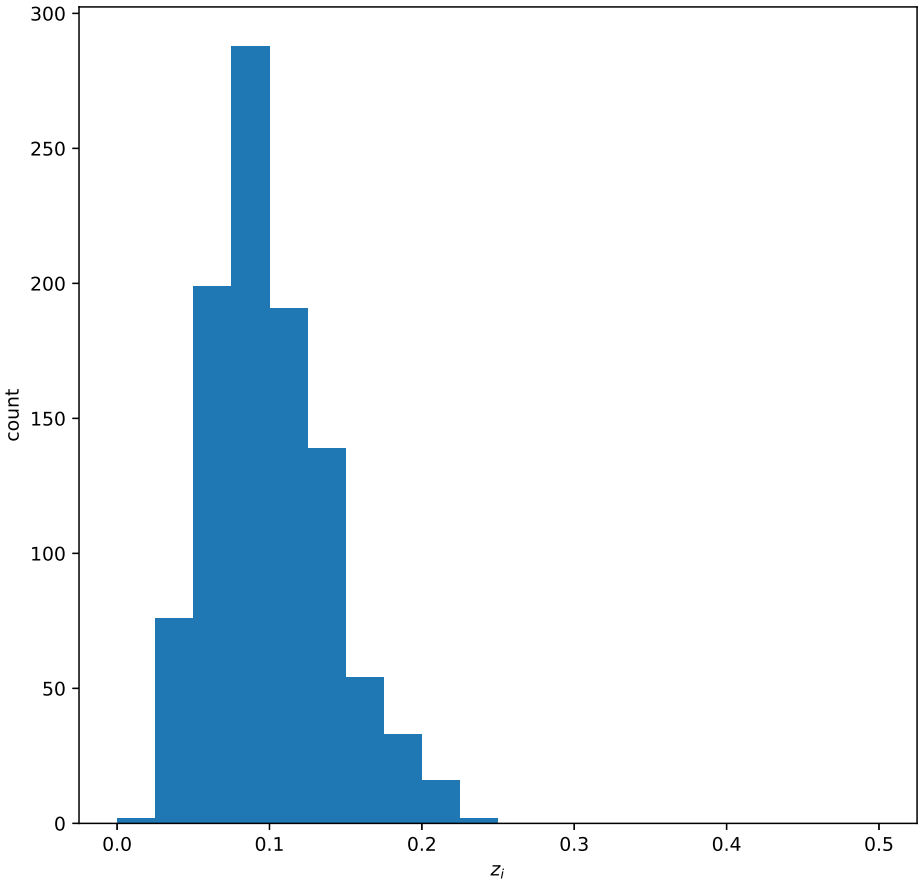


Figure 3.1: Initial distribution of the elements of 100 softmax output vectors.

3.1 Warm-up Phase

Warm-up phase is described as to train a model a certain number of epochs by using only labeled samples [1]. In that way, the model gains a sort of "intuition" about the classes and therefore, not produce random softmax outputs z_i. In order to illustrate this, we split the 100 samples into 10 labeled and 90 unlabeled (class-balanced) datasets and train the model only on the labeled samples (for 20 epochs and using the loss $-\log(\boldsymbol{z} \cdot \tilde{\boldsymbol{y}})$).

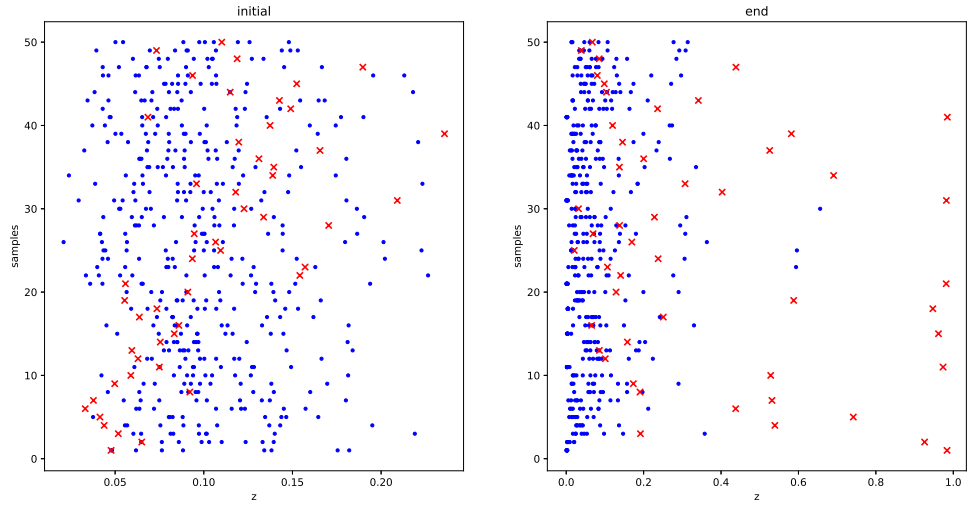
Figure 3.2 shows that how the softmax outputs change at the end of warm-up phase. As seen in the same figure at the initial state (on the left) all outputs are very close to each other. Whereas at the end, the model favors the true softmax outputs so that these outputs (i.e., red crossed points in the figure) are mostly beyond the false ones (i.e., the blue points). Note that our labeled samples are $1, 11, ...,$ and 91st samples. So, the relevant softmax outputs of those samples reach to 1 at the end of the phase.

One possible version of the loss used in the warm-up phase could be

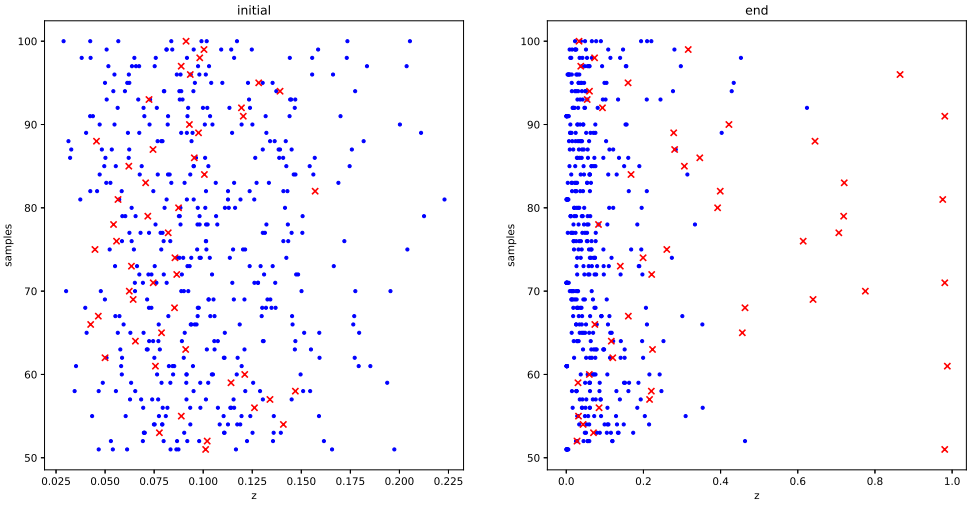
$$C = -\log(\boldsymbol{z} \cdot \tilde{\boldsymbol{y}}) - \lambda \log(1 - \boldsymbol{z} \cdot \tilde{\boldsymbol{y}}), \tag{3.1}$$

where the second term is the delay term and the constant λ is the delay coefficient. We introduce such a loss for the warm-up phase because in the previous one, true softmax outputs of the labeled ones converge to 1. It prevents those samples from contributing to the training in terms of gradient (i.e., signals) (see equation 2.20) in the "after-warm-up" phase where we train our model by using both types of datasets (i.e., labeled and unlabeled).

Figure 3.3 shows that when the delay coefficient λ is 1, how the softmax outputs change. We see from the same figure that when the warm-up phase is completed, the true softmax outputs of the labeled samples are located around 0.5 (because equation (3.1) becomes $-\log z_c - \log(1 - z_c)$ having a minimum at $z_c = 0.5$), and still the model achieves to produce the outputs of the unlabeled samples where the true ones are beyond the false ones. We treat this parameter as a hyperparameter while obtaining our results in the next chapter.

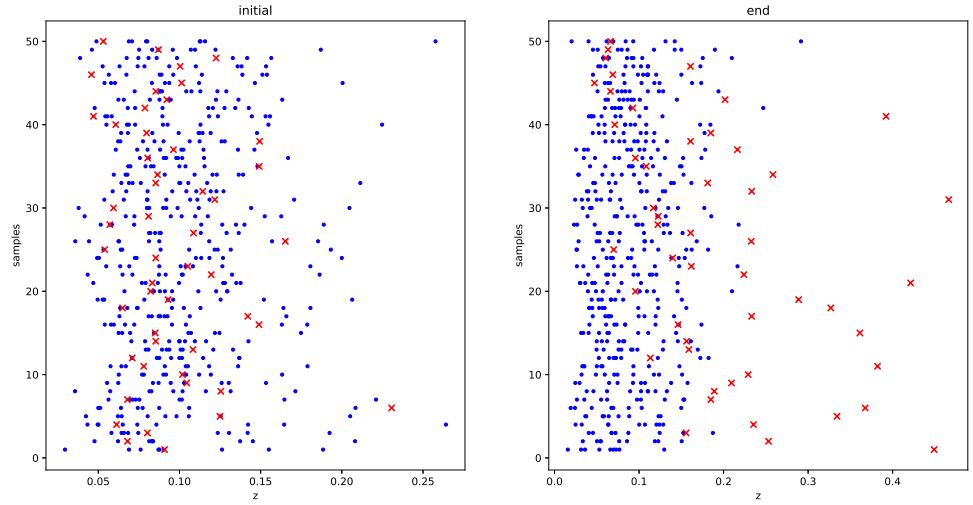


(a) Softmax output values (z_i) for first 50 samples.

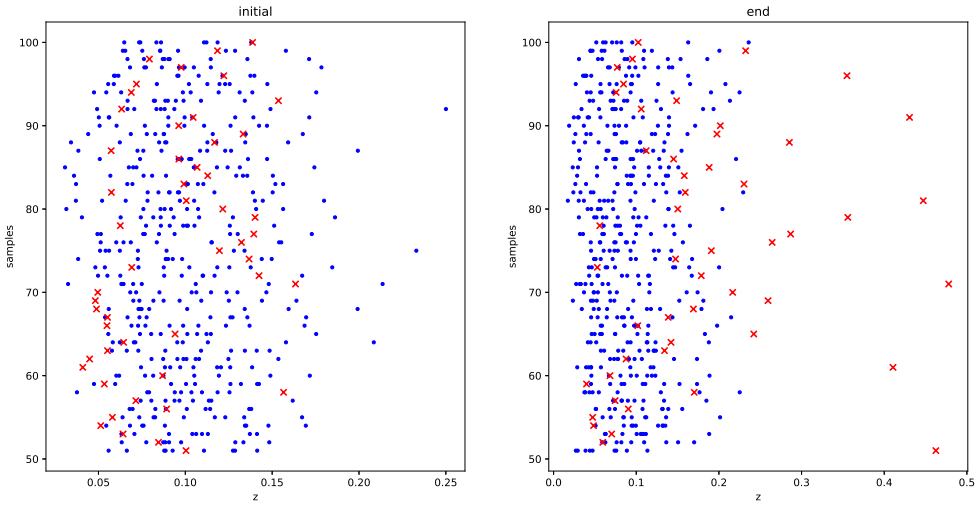


(b) Softmax output values (z_i) for last 50 samples

Figure 3.2: Softmax output values of the samples at the beginning (left) and the end (right) of the warm-up phase. Red crossed points represent the true classes.



(a) Softmax output values (z_i) for first 50 samples.



(b) Softmax output values (z_i) for last 50 samples

Figure 3.3: Softmax output values of the samples at the beginning (left) and the end (right) of the warm-up phase. Red crossed points represent the true classes. The delay coefficient λ is 1.

In our work, we also take advantage of this technique (i.e., warm-up phase) in the training. Now, we introduce two pseudo-label selection strategies, namely pseudo-labeling with fixed n and pseudo-labeling with threshold τ, in order to obtain more than one negative signal.

3.2 Pseudo-label Selection with Fixed n

This strategy actually is a quite natural way to select pseudo-labels; that is, we choose first n greatest z_i and determine the pseudo-labels according to indices of these elements. Now, we express it in a more formal way.

Let $\boldsymbol{z} = [z_1, ..., z_K]^T$ and $\tilde{\boldsymbol{y}} = [\tilde{y}_1, ..., \tilde{y}_K]^T$ be the softmax output vector and the label vector of a sample $\boldsymbol{s}$, respectively. Let $\boldsymbol{z}_{sorted}$ be the sorted form of $\boldsymbol{z}$ in ascending order (i.e., $z_{sorted_i} < z_{sorted_{i+1}}$). Also, let the corresponding sorted label vector be $\tilde{\boldsymbol{y}}_{sorted}$ that is described as if $z_{sorted_i} = z_j$, then $\tilde{y}_{sorted_i} = \tilde{y}_j$, for $i = 1, ..., K$ (e.g., if $\boldsymbol{z} = [z_1, z_2, z_3, z_4]^T$ and $\boldsymbol{z}_{sorted} = [z_1, z_4, z_3, z_2]^T$, then $\tilde{\boldsymbol{y}}_{sorted} = [\tilde{y}_1, \tilde{y}_4, \tilde{y}_3, \tilde{y}_2]^T$). Then, by the pseudo-label selection with fixed n, we mean to construct $\tilde{\boldsymbol{y}}_{sorted}$ as

$$\tilde{y}_{sorted_i} = \begin{cases} 1, & \text{for } i \in \{K, K-1, ..., K-n+1\} \\ 0, & \text{otherwise} \end{cases}, \tag{3.2}$$

where K is the number of classes (as usual).

For instance, assume $\boldsymbol{z}_{sorted} = [z_1, z_4, z_3, z_2]^T$. Then, pseudo-label selection with fixed $n = 2$ results in $\tilde{\boldsymbol{y}}_{sorted} = [0, 0, 1, 1]^T$. Since $\tilde{\boldsymbol{y}}_{sorted} = [\tilde{y}_1, \tilde{y}_4, \tilde{y}_3, \tilde{y}_2]^T$ by definition, we have $\tilde{\boldsymbol{y}} = [0, 1, 1, 0]^T$.

One remark here is that this labeling strategy is equivalent to the naive pseudo-labeling given in (1.8) for $n = 1$ since both take the index of the maximum element in $\boldsymbol{z}$ into consideration.

Although this pseudo-label selection allows us to obtain a signal vector with many signals, we may encounter some issues in terms of the magnitudes of those negative ones. The reason is that because of the signal conservation Theorem 1, the sum negative signals must be equal to the negative of the sum of positive signals. So, each

negative signal's magnitude tends to decrease when n increases (and it is zero for $n = K$).

So, we also introduce the next pseudo-label selection way in order to combat such a possible problem.

3.3 Pseudo-label Selection with Threshold τ

We directly begin with the formal definition.

Let $\boldsymbol{z} = [z_1, ..., z_K]^T$ and $\tilde{\boldsymbol{y}} = [\tilde{y}_1, ..., \tilde{y}_K]^T$ be the softmax output vector and the label vector of a sample $\boldsymbol{s}$, respectively. Let $\boldsymbol{z}_{sorted}$ be the sorted form of $\boldsymbol{z}$ in ascending order. Also, let $\tilde{\boldsymbol{y}}_{sorted}$ the corresponding sorted label vector which is described as if $z_{sorted_i} = z_j$, then $\tilde{y}_{sorted_i} = \tilde{y}_j$, for $i = 1, ..., K$ (again, e.g., if $\boldsymbol{z}_{sorted} = [z_1, z_4, z_3, z_2]^T$, then $\tilde{\boldsymbol{y}}_{sorted} = [\tilde{y}_1, \tilde{y}_4, \tilde{y}_3, \tilde{y}_2]^T$). Then, by the pseudo-label selection with threshold $\tau \in (0, 1)$, we mean to construct $\tilde{\boldsymbol{y}}_{sorted}$ as

$$\tilde{y}_{sorted_i} = \begin{cases} 0, & \text{for } i \in \{1, ..., M\} \setminus \{K\} \\ 1, & \text{for } i \in \{M+1, ..., K\} \setminus \{K+1\} \end{cases}, \tag{3.3}$$

where $M \in \{1, ..., K\}$ is the smallest integer that satisfies

$$\sum_{i \leq M} z_{sorted_i} > \tau. \tag{3.4}$$

For example, let $\boldsymbol{z} = [0.3, 0.2, 0.1, 0.4]^T$. Then, $\boldsymbol{z}_{sorted} = [0.1, 0.2, 0.3, 0.4]^T = [z_3, z_2, z_1, z_4]^T$. For $\tau = 0.25$, M would be 2 since $z_{sorted_1} < \tau$ and $z_{sorted_1} + z_{sorted_2} > \tau$. Therefore, we have $\tilde{\boldsymbol{y}}_{sorted} = [0, 0, 1, 1]^T$ according to equation (3.3); hence $\tilde{\boldsymbol{y}} = [1, 0, 0, 1]^T$.

Let us increase the threshold to $\tau = 0.9$. In this case, M would be 4. Thus, since $i = 4 \in \{5, ..., 4\} \setminus \{5\}$, $\tilde{y}_{sorted_4} = 1$. Also, $\tilde{y}_{sorted_i} = 0$ for $i = 1, 2, 3$ because $i \in \{1, ..., 4\} \setminus \{4\}$. Hence, $\tilde{\boldsymbol{y}} = [0, 0, 0, 1]^T$.

One observation in this definition is that since, during the training, the z_i with pos-

itive signals decrease, it automatically allows to increase M to satisfy the condition expressed in (3.4). So, as opposed to the previous selection strategy, the number of 1 in the label vector is not constant during the training.

Another observation about this definition is that the negative signal magnitude for z_{sorted_i} can be controlled by the value of threshold τ. To show this, we suppose $\boldsymbol{z}_{sorted}$ is the sorted version of $\boldsymbol{z}$ and $\boldsymbol{\phi}_{sorted}$ is the corresponding signal vector of $\boldsymbol{z}_{sorted}$ (e.g., if $\boldsymbol{z}_{sorted} = [z_1, z_4, z_3, z_2]^T$, then $\boldsymbol{\phi}_{sorted} = [\phi_1, \phi_4, \phi_3, \phi_2]^T$). For $M < K - 1$, we can express the negative signal ϕ_{sorted_i} by using Proposition 2 as

$$\begin{aligned} \phi_{sorted_i} &= z_{sorted_i}\Big(\frac{-1}{\sum_{i\in\{M+1,\dots,K\}} z_{sorted_i}} + 1\Big) \\ &= z_{sorted_i}\Big(\frac{-\sum_{i\in\{1,\dots,M\}} z_{sorted_i}}{1 - \sum_{i\in\{1,\dots,M\}} z_{sorted_i}}\Big), \end{aligned} \tag{3.5}$$

where we use the softmax function property (1.3).

According to equation (3.5), for $\tau = 0.5$, the magnitude $|\phi_{sorted_i}|$ would be greater than z_{sorted_i} because M satisfies the condition (3.4) (i.e., $\sum_{i\in\{1,\dots,M\}} z_{sorted_i} > 0.5$). Furthermore, for $\tau = 0.2$, $|\phi_{sorted_i}| > z_{sorted_i}/4$ since $\sum_{i\in\{1,\dots,M\}} z_{sorted_i} > 0.2$ by the condition (3.4).

For $M = K - 1, K$, on the other hand, we would have a signal vector $\boldsymbol{\phi}$ where only one signal ϕ_i is negative and is equal to $-(1 - z_i)$ (by equation 3.3 and Proposition 2).

Since we do not know which labeling strategy (and which parameter) would produce better results in our training, we treat both pseudo-labeling strategies as hyperparameters of our model.

Now, we are ready to train our model (i.e., we have determined our loss function and come up with two strategies to construct pseudo-labels), and Algorithm 1 is the algorithm of how we train our model.

Algorithm 1 Training with different pseudo-labeling methods.

Determine the labeled and unlabeled datasets D_L and D_U.

Warm-up phase:

Determine number of iterations N and minibatch size m.

Use the loss C given in (3.1).

Determine the delay coefficient λ.

Set $k = |D_L|/m$.

for number of training iterations N **do**

  for k steps **do**

    Sample minibatch of m labeled sample, vector pairs $\{(\boldsymbol{s^{(1)}}, \boldsymbol{y^{(1)}}), ..., (\boldsymbol{s^{(m)}}, \boldsymbol{y^{(m)}})\}$.

    Compute the corresponding softmax output vectors $\{\boldsymbol{z^{(1)}}, ..., \boldsymbol{z^{(m)}}\}$.

    Update the model by its stochastic gradient:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} C(\boldsymbol{z}^{(i)}, \boldsymbol{y}^{(i)}).$$

  end for

end for

return $\boldsymbol{\theta}_{warm-up} = \boldsymbol{\theta}$

After-warm-up phase:

Determine the number of iterations $N_{after-warm-up}$ and minibatch sizes m_L and m_U.

Use the loss C given in (2.15).

Determine pseudo-label selection strategy.

Set $k = |D_U|/m_U$.

Initialize the model: $\boldsymbol{\theta} = \boldsymbol{\theta}_{warm-up}$

for number of training iterations $N_{after-warm-up}$ **do**

  for k steps **do**

    Sample minibatch of m_L labeled sample, vector pairs $\{(\boldsymbol{s^{(1)}}, \boldsymbol{y^{(1)}}), ..., (\boldsymbol{s^{(m_L)}}, \boldsymbol{y^{(m_L)}})\}$.

    Sample minibatch of m_U unlabeled samples $\{\boldsymbol{s^{(1)}}, ..., \boldsymbol{s^{(m_U)}}\}$.

    Compute the corresponding softmax output vectors $\{\boldsymbol{z^{(1)}}, ..., \boldsymbol{z^{(m_L)}}\}$, $\{\boldsymbol{z^{(1)}}, ..., \boldsymbol{z^{(m_U)}}\}$.

    Compute the pseudo-label vectors $\{\tilde{\boldsymbol{y}}^{(1)}, ..., \tilde{\boldsymbol{y}}^{(m_U)}\}$ based on the selection strategy.

    Update the model by its stochastic gradient:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \frac{1}{m_L + m_U} \Big(\sum_{i=1}^{m_L} C(\boldsymbol{z}^{(i)}, \boldsymbol{y}^{(i)}) + \sum_{i=1}^{m_U} C(\boldsymbol{z}^{(i)}, \tilde{\boldsymbol{y}}^{(i)}) \Big).$$

  end for

end for

CHAPTER 4

NUMERICAL RESULTS AND DISCUSSION

In this chapter, we test our method (competing labels, i.e., sending many negative signals to the model) against the naive pseudo-label method on three popular gray-scale image datasets, namely, MNIST, Fashion-MNIST, and KMNIST. In our all trainings we use PyTorch deep learning framework [18] and use the machine Intel Xeon i5 with a Nvidia K40 GPU in our computation.

4.1 MNIST

4.1.1 Dataset

MNIST dataset is a $K = 10$ class gray-scale image dataset [6] (see Figure 4.1). Each image has 28×28 dimensions (height and width). This dataset contains 70000 images in total where 60000 images of them are allocated for the training set, while 10000 of them are for the test set.

4.1.2 Model

During the training on MNIST and all other datasets, we use a single model, namely a convolutional neural network [14]. Our model contains two hidden layers where the first hidden layer consists of 64 channels, 4×4 kernel size, and 2 stride. The second one, on the other hand, contains 128 channels, 4×4 kernel size, and 2 stride. Also, each hidden layer is followed by batch normalization [10] and leaky ReLU activation

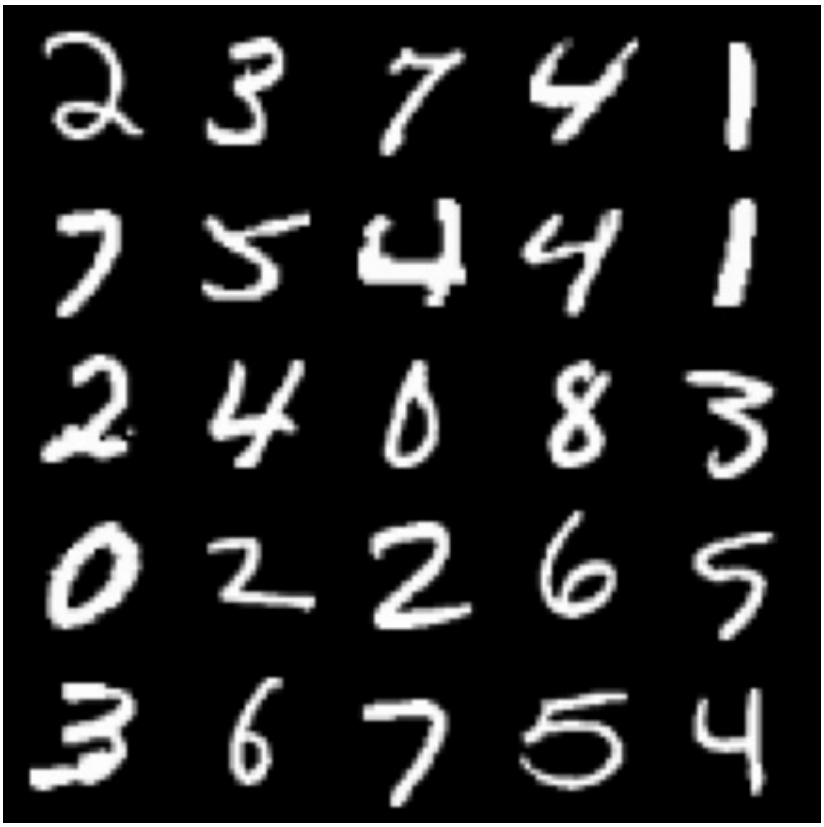


Figure 4.1: 25 MNIST images [6]

function with the negative slope 0.2 [23]. And the output layer maps to a $K = 10$ dimensional vector.

4.1.3 Training

In the training, as an optimization method, we use a stochastic gradient descent with Adam [12] having a learning rate 2×10^{-4} for all datasets (for the other hyperparameters of the optimizer, we use the recommended values in [12]). Also, we scale our dataset to have pixel values in a unit range $[0, 1]$, then standardize with the mean and standard deviation, 0.5 and 0.5, respectively.

While constructing our labeled and unlabeled datasets, we utilize the following procedure: First we pick N_L number of labeled samples from the training dataset. Then, from the remaining, we randomly choose N_V samples for the validation purpose, and we remove the labels of the rest of the samples and devote for the unlabeled samples. So, in the final situation, we have three datasets, namely the labeled dataset D_L, the unlabeled dataset D_U, and the validation dataset D_V where only D_L and D_V contain labels. In our all experiments, we deterimine the size of the validation set D_V as $N_V = 1000$.

In the training, there are two phases, warm-up and after-warm-up phase. In the warm-up phase, we train our model for 20 epochs. Note that in this phase, we only use the labeled dataset D_L, and the mini batch-size is 32.

As opposed to the warm-up phase, in the after-warm-up phase we utilize both datasets the labeled one D_L and unlabeled one D_U. While constructing the mini batch in this phase, we use 32 images from the labeled dataset D_L (as done in [16]) and 64 from the unlabeled dataset D_U; so the entire mini batch size is 96. We train our model for 20 epochs. Note that we define the number of epoch in terms how many times the model "sees" the unlabeled set as performed in [11]. So, when the model uses the entire unlabeled data only once, it actually utilizes the labeled data more than one to update the parameters.

4.1.4 Hyperparameter Selection

To be able to train our model we have to determine the other hyperparameters such as the delay coefficient λ and the pseudo-label selection strategy described in Chapter 3. To do this, we take into consideration the dataset D_L with $N_L = 600$ (and the other datasets D_U and D_V are determined based on the procedure mentioned at the beginning of the previous section).

We train our model for three times and obtain the (best) validation accuracy for different hyperparameters as in Table 4.1. As seen in the table, the naive pseudo-labeling i.e., $n = 1$ has its greatest value for the delay coefficient $\lambda = 1$. Also, the values for the fixed pseudo-label selection with $n \neq 1$ are lower compared to the pseudo-label

Table 4.1: Average validation accuracy (of three trials) for different hyperparameters and 600 labeled samples

	$\lambda = 0$	$\lambda = 1$
$\tau = 0.01$	95.9	95.3
$\tau = 0.05$	95.6	95.8
$\tau = 0.1$	95.7	95.4
$\tau = 0.2$	95.3	95.6
$\tau = 0.3$	95.4	95.4
$n = 1$	95.4	95.5
$n = 2$	93.5	93.4
$n = 3$	93.0	92.4
$n = 4$	92.5	92.3
$n = 5$	92.3	92.1

selection via threshold. Even though for the threshold $\tau = 0.01$ and the delay coefficient $\lambda = 0$, we have the greatest value (95.9) in the threshold selection strategy, for the seek of comparison (i.e., to be able to compare with the naive pseudo-labeling), we choose the second highest one with $\tau = 0.05$ and $\lambda = 1$.

At that point, one interesting question would be how the different pseudo-label selection strategies behave while training. To illustrate this, we consider the delay coefficient $\lambda = 1$ case and $N_L = 600$.

Figure 4.2 shows the average number of negative signals in the unlabeled data (note that for the illustration purpose we define the epoch in the horizontal axis in Figures 4.2, 4.3, and 4.4 in terms of labeled data; that is, how many times the model sees the entire labeled dataset. So, for $N_L = 600$, after approximately 50 epochs past, the model sees the entire unlabeled data). We observe that as expected for the fixed pseudo-label selection, i.e., $n = 1, ..., 5$, the average number of the negative signals is constant.

However, it is not the case for the selection via threshold: we have at the early epochs some high number of signals in average. And this number gradually dies, while epochs increase and converges to one. In other words, the pseudo-label selection via threshold turns into naive-pseudo-labeling after a certain value of epochs.

From Figure 4.2, we also see that when the threshold decreases the convergence of the signals to one slows down as expected since the condition in 3.4 with a smaller

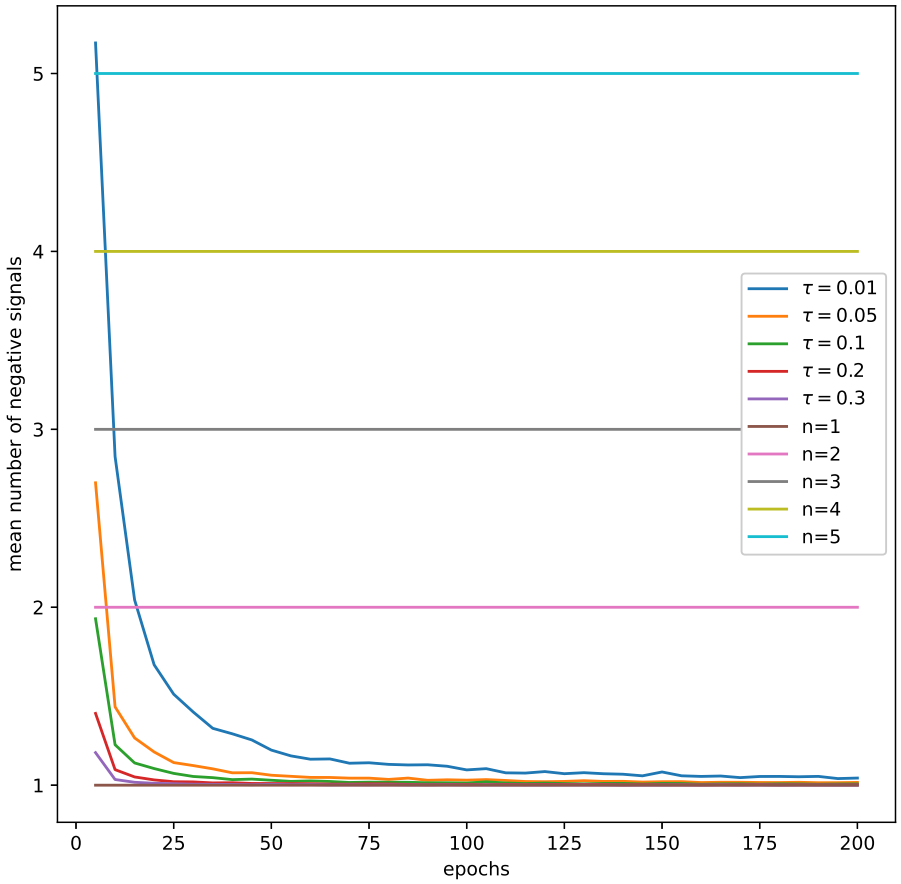


Figure 4.2: Average number of negative signals vs epochs

threshold τ can be satisfied by a smaller integer M in 3.3.

Another statistics in the unlabeled dataset that we investigate is the average of the greatest softmax outputs (i.e., $\frac{1}{N_U}\sum_{i=1}^{N_U}\max_j z_j^{(i)}$). Figure 4.3 shows how average z_{max} of unlabeled samples change while epoch increases. We observe from the same figure that while epoch changes, the naive pseudo-labeling and the threshold strategies behave similarly, compared to the fixed selection with $n = 2, ..., 5$. Also we see that the label selection with $n > 1$ has difficulty to converge to 1.

To find out the reason for this, we plot the average loss of unlabeled samples, i.e., $-\frac{1}{N_U}\sum_{i=1}^{N_U}\log(\boldsymbol{z}^{(i)} \cdot \tilde{\boldsymbol{y}}^{(i)})$. Figure 4.4 shows that how the unlabeled samples loss changes when the epoch increases. We see that for the selection with $n > 1$, the

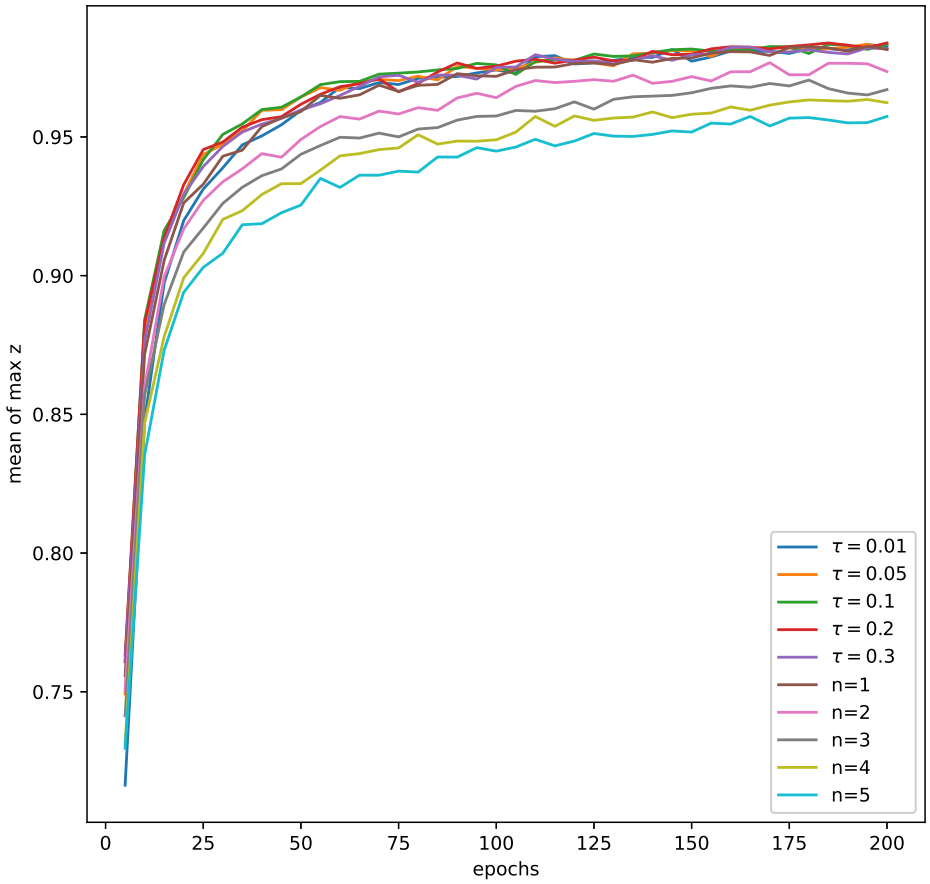


Figure 4.3: Average $\boldsymbol{z}_{max}$ vs epochs

loss converges very quickly zero, whereas for the others (the selection with $n = 1$ and τ), the loss does not reach to zero at the end of the training. One reason for this discrepancy could be that fact that to make the sum of more than one softmax outputs easier than making only one softmax output (i.e., for $n > 1$ there are more than one z_i in the product $\boldsymbol{z} \cdot \tilde{\boldsymbol{y}}$, while for $n = 1$ and τ selections there is only one after a certain value of epoch); hence $\log(\boldsymbol{z} \cdot \tilde{\boldsymbol{y}}) \approx 0$ for the label selection $n > 1$.

Now we move on next section and we employ the hyperparameters that we determine in this section (i.e., $\lambda = 1$, $n = 1$, and, $\tau = 0.05$) to discover which pseudo-labeling method (naive labeling or competing labels) performs better in different number of labeled samples situations.

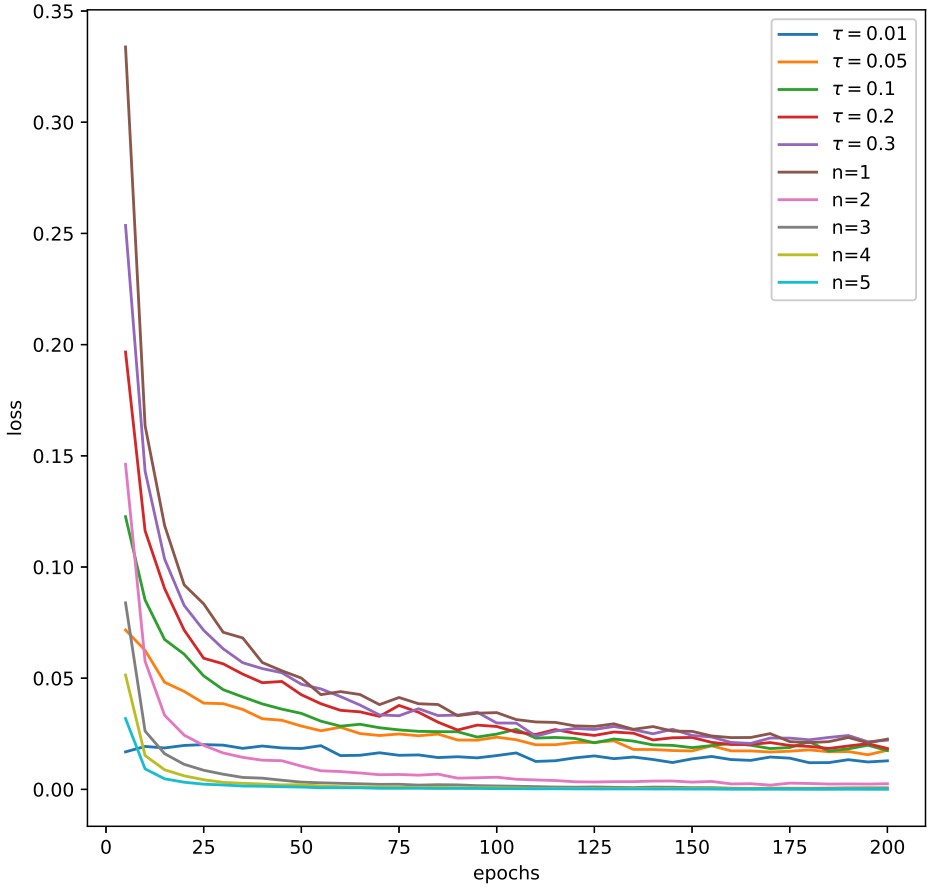


Figure 4.4: Unlabeled loss vs epochs

4.1.5 Results

In this section, we test the performances of the two pseudo-labeling methods, namely, the naive pseudo-labeling and competing labels under various labeled data scenarios. Specifically, we consider four situations, namely, $N_L = 100, 600, 1000, 3000$.

During our training on this four situations, we employ the same hyperparameters (we find out in the previous section, i.e., delay coefficient $\lambda = 1$, and pseudo-labeling via $\tau = 0.05$) in all cases following the convention in [1, 19]. That is, we do not search the optimum hyperparameters (i.e., λ, τ, and n) for each different labeled data case.

We train our model for 10 experiments for each N_L by initializing the model with the

same weights (as done in [16]). In each experiment, we re-split the dataset into three different datasets (N_L, N_U, and N_V), and the two methods (i.e., naive pseudo-labeling and competing labels) are used in exactly the same split. Therefore, we can measure the performance of each method for the same datasets and the model initialized with the same weights. For an experiment, we report the test accuracy of the model at the epoch where the model has the greatest validation accuracy as performed in [20].

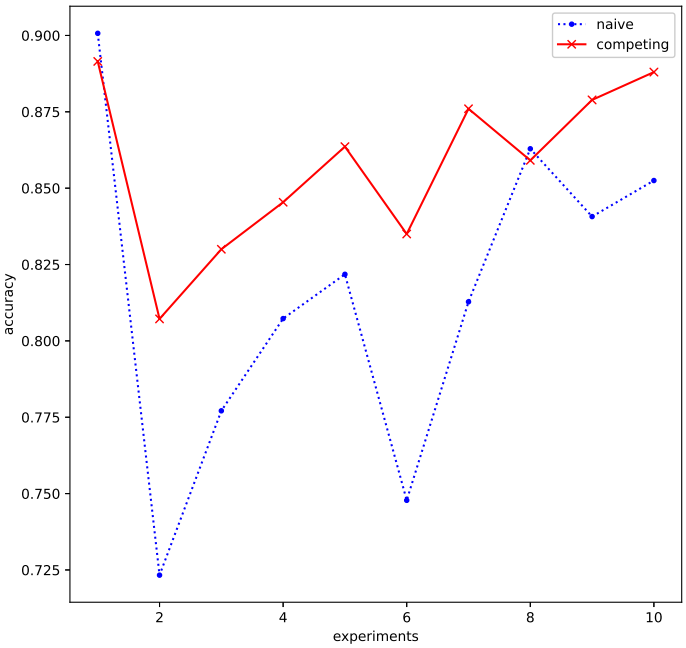
Table 4.2: MNIST. Average test accuracy and standard deviation of 10 experiments. Bold number indicates the highest accuracy.

Labeled samples:	100	600	1000	3000
CNN	73.8 ± 1.6	91.9 ± 0.4	93.9 ± 0.1	96.3 ± 0.2
CNN + Naive	81.5 ± 5.4	94.7 ± 0.2	95.7 ± 0.3	96.8 ± 0.2
CNN + Competing	$\mathbf{85.6 \pm 2.8}$	$\mathbf{95.1 \pm 0.4}$	$\mathbf{95.9 \pm 0.2}$	$\mathbf{96.9 \pm 0.2}$

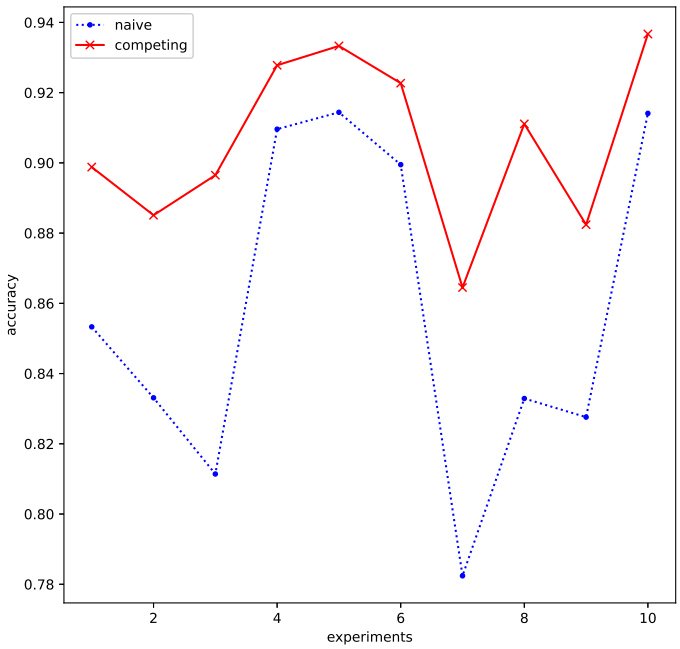
Table 4.2 shows the test accuracy of different methods for various number of labeled samples. In the table, CNN row indicates that the model is trained using only labeled samples; that is, there is no unlabeled data involved in the training. CNN + Naive and CNN + Competing rows, on the other hand, represent that in the training the naive and competing methods are used, respectively. When we use the pseudo-labeling (regardless of which method we use) the test accuracy increases. However, when comparing two methods, we see that in each case, the competing labels method is superior over the naive method; that is the competing labels method shows better generalization performance. Also from the same table, we observe the discrepancy between the accuracy of two methods enlarges when the supervision weakens, i.e., the number of labeled samples decreases.

Since we measure the performances of two methods under the same condition (i.e., the same data and model parameters), we illustrate (for $N_L = 100$) how they perform in each experiment. As seen from Figure 4.5a, depending on the data split (i.e., D_L, D_U, D_V), in each experiment, classification accuracy of both methods changes similarly. However, the competing labels method shows better performance in almost all the experiments compared to the naive one.

Next we adapt the mixup regularization method (that is expressed in (1.9) with the constants $a = b = 1$ of the Beta distribution as used in [1]) to see if we can get any improvement in generalization. In this mixup case, we apply mixup only to the



(a) Without Mixup



(b) With Mixup

Figure 4.5: Accuracy vs experiments for MNIST 100 labeled samples

labeled dataset in both phases in the mini batch level, and train in the after-warm-up phase for 50 epochs.

Table 4.3 shows that when the mixup regularization method is added, test accuracy scores of the both methods increase. Moreover, for the cases $N_L = 1000, 3000$ where the score difference is small, both methods reach the same score. However, when the supervision is significantly small, the competing method with mixup performs better in terms of generalization.

We can also illustrate how each method behaves on the same data split. To show it, we focus on the labeled case $N_L = 100$ where the standard deviation is the highest. Figure 4.5b indicates that both methods' classification accuracy changes similarly on the same data split. However, as in the "without mixup" case, when the mixup regularization is applied, in all the experiments, competing method performs better.

Table 4.3: MNIST (including mixup). Average test accuracy and standard deviation of 10 experiments. Bold number indicates the highest accuracy.

Labeled samples:	100	600	1000	3000
CNN	73.8 ± 1.6	91.9 ± 0.4	93.9 ± 0.1	96.3 ± 0.2
CNN + Naive	81.5 ± 5.4	94.7 ± 0.2	95.7 ± 0.3	96.8 ± 0.2
CNN + Competing	85.6 ± 2.8	95.1 ± 0.4	95.9 ± 0.2	96.9 ± 0.2
CNN + Naive + Mixup	85.8 ± 4.8	96.1 ± 0.3	$\mathbf{96.6 \pm 0.2}$	$\mathbf{97.3 \pm 0.1}$
CNN + Competing + Mixup	$\mathbf{90.6 \pm 2.4}$	$\mathbf{96.2 \pm 0.1}$	$\mathbf{96.6 \pm 0.3}$	$\mathbf{97.3 \pm 0.2}$

Even though our training setup and the setup in [16] is quite different, we compare our best results with the results provided in the same work. We observe from Table 4.4 that our model CNN and the author's model DropNN show different performances under difference labeled scenarios. Secondly, in all cases except for $N_L = 3000$ competing labels method (i.e., CNN + Competing) performs better than the naive method (i.e., DropNN + Naive). Finally, when we compare the best results of us (i.e., Competing + Mixup) with the author's (Naive + DAE), we observe that in the first three cases, $N_L = 100, 600, 1000$ our method has a greater generalization performance. In the last case ($N_L = 3000$), they both perform equally well.

Now, we move to the next section to measure the performances of two methods (naive and competing labels) on a different dataset, namely Fashion-MNIST.

Table 4.4: Competing labels vs Naive labeling. Last three rows are adapted from the work [16]. Bold number indicates the highest test accuracy.

Labeled samples:	100	600	1000	3000
CNN	73.8	91.9	93.9	96.3
CNN + Competing	85.6	95.1	95.9	96.9
CNN + Competing + Mixup	**90.6**	**96.2**	**96.6**	**97.3**
DropNN	78.1	91.4	93.4	96.3
DropNN + Naive	83.9	95.0	95.7	97.2
DropNN + Naive + DAE	89.5	96.0	96.5	**97.3**

4.2 Fashion-MNIST

4.2.1 Dataset

Likewise MNIST dataset, Fashion-MNIST dataset is a $K = 10$ class gray-scale image dataset [22] (see Figure 4.6). It contains 60000 images for training and 10000 images for testing purpose and each image has 28×28 dimensions.

4.2.2 Results

In the choice of hyperparameters in Fashion-MNIST dataset, by inspiring from the work [17], we keep the hyperparameters the same (i.e., used in the previous case, MNIST experiment), and not "over-tweaking" those. Also, we imitate the same training procedure expressed in the previous section. However, since the competing label and naive pseudo-labeling do not show a distinct behavior for $N_L = 3000$ case, we omit this case and only focus on the labeled datasets with $N_L = 100, 600, 1000$.

Table 4.5: Average test accuracy (for 5 experiments) of the model on the completely labeled datasets.

	MNIST	Fashion-MNIST	KMNIST
accuracy	98.6 ± 0.1	89.1 ± 0.2	90.3 ± 0.3

After training (i.e., 10 experiments) we have the results in Table 4.6. Our first observation from this table is that when our model is trained on only labeled data, the test accuracy is lower compared to the MNIST case. One possible explanation for this is that our model is not well suited for the classification of Fashion-MNIST. Actually, we can support this view by Table 4.5. It indicates that when our model is trained by

Figure 4.6: 25 Fashion-MNIST images [22]

treating all the unlabeled samples as labeled ones, it shows a lower classification performance for Fashion-MNIST compared to MNIST. Also, Figure 4.7 shows that even though the accuracy on the training set ($N_L = 59000$) increases for Fashion-MNIST dataset as epoch increases (the line with the label "fashion-mnist train"), the accuracy on the validation set ($N_V = 1000$) reaches its maximum around 20 epochs (the line with the label "fashion-mnist val"). This can be considered as the evidence for the fact that our model is less capable of generalizing the Fashion-MNIST data compared to the MNIST.

Secondly, our method (competing labels) is, as similar to the previous one, is superior to the naive one for each cases, i.e., $N_L = 100, 600, 1000$; that is, it produces greater test accuracy with the competing labels method (see Table 4.6). Also, we see that the

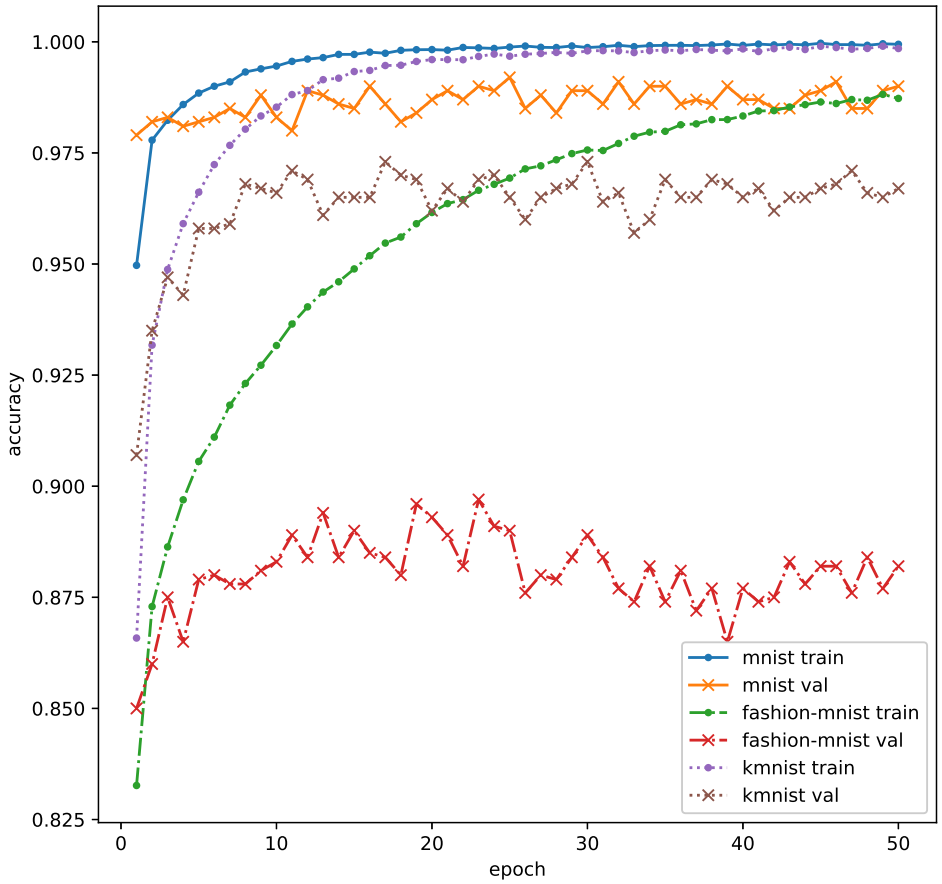


Figure 4.7: Fully labeled training for MNIST, Fashion-MNIST, and KMNIST datasets

naive one performs worse than the only labeled case for $N_L = 100$.

Table 4.6: Fashion-MNIST. Average test accuracy and standard deviation of 10 experiments. Bold number indicates the highest accuracy.

Labeled samples:	100	600	1000
CNN	66.5 ± 1.5	76.5 ± 0.8	78.2 ± 0.5
CNN + Naive	66.3 ± 4.4	77.1 ± 0.4	78.6 ± 0.6
CNN + Competing	$\mathbf{68.8 \pm 3.3}$	$\mathbf{77.2 \pm 0.8}$	$\mathbf{78.7 \pm 0.7}$

Once again, benefiting from the same data and model parameters, we compare two methods experiment-wise. Figure 4.8a shows that how accuracy resulting from two different methods changes for different experiments. We see from Figure 4.8a that during the experiments their test accuracy fluctuations are similar. However, as similar to MNIST dataset, in almost all the experiments the competing method has a higher accuracy score than the naive one.

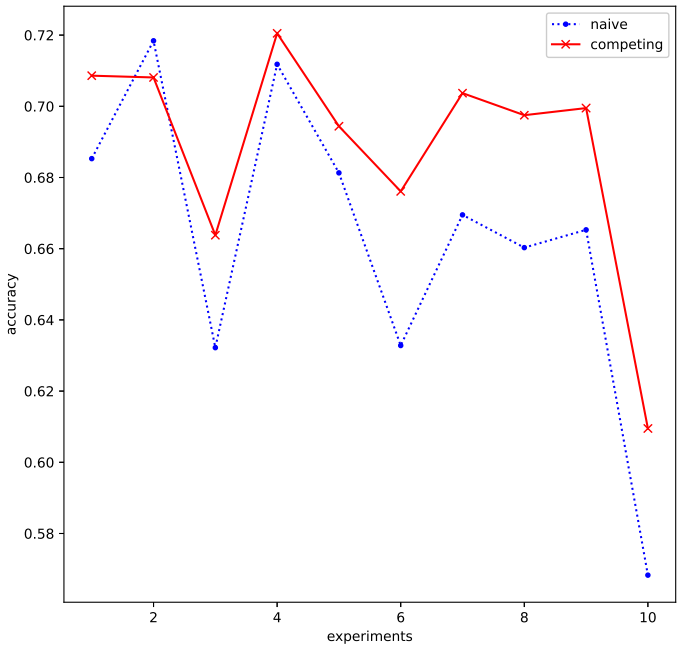
Next, we apply the mixup regularization method to our training. After applied and trained, we obtain the test results in Table 4.7. The table shows us that when we add the mixup factor, the accuracy for both method increases. However, for the method competing labels, the test accuracy is greater than the naive one. And the greatest difference occurs for the weakest supervision, i.e., $N_L = 100$. We can also check

Table 4.7: Fashion-MNIST (including mixup). Average test accuracy and standard deviation of 10 experiments. Bold number indicates the highest accuracy.

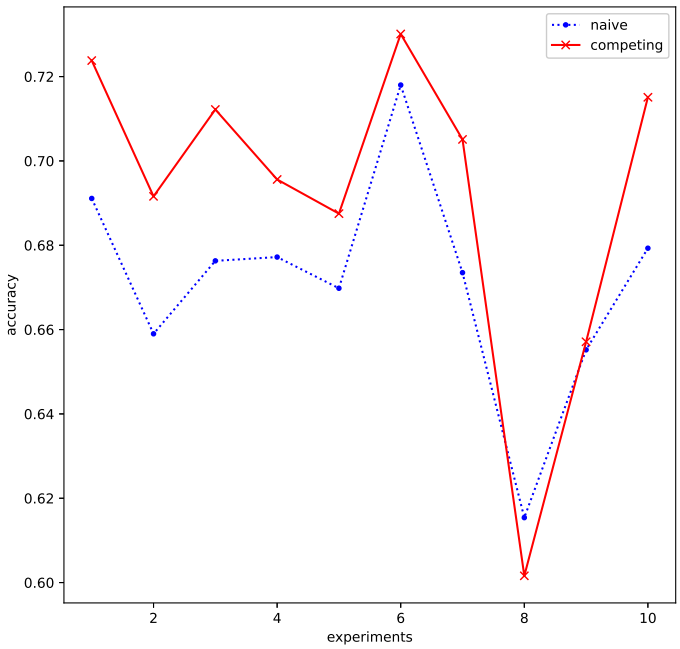
Labeled samples:	100	600	1000
CNN	66.5 ± 1.5	76.5 ± 0.8	78.2 ± 0.5
CNN + Naive	66.3 ± 4.4	77.1 ± 0.4	78.6 ± 0.6
CNN + Competing	68.8 ± 3.3	77.2 ± 0.8	78.7 ± 0.7
CNN + Naive + Mixup	67.2 ± 2.7	77.3 ± 0.9	78.6 ± 0.9
CNN + Competing + Mixup	$\mathbf{69.2 \pm 3.8}$	$\mathbf{77.4 \pm 0.7}$	$\mathbf{78.7 \pm 0.6}$

the behaviour of two methods during the experiments when the mixup regularization is present. We investigate the case $N_L = 100$, where the standard deviation is the highest. Figure 4.8b shows that under the same data splits, both methods behave similarly: the increase and decrease in their accuracy during the experiments are similar. However, the competing method leads to higher generalization performance in almost each experiment.

In the next and final section, we investigate the two methods' performance on our



(a) Without Mixup



(b) With Mixup

Figure 4.8: Accuracy vs experiments for Fashion-MNIST 100 labeled samples

third dataset, namely KMNIST.

4.3 KMNIST

Figure 4.9: 25 KMNIST images [5]

4.3.1 Dataset

In our final dataset, we take into consideration KMNIST image dataset. Likewise the previous two datasets, each image in KMNIST is $K = 10$ class gray-scale image, and has 28×28 dimensions [5] (see Figure 4.9). Also, the training and the test dataset sizes of KMNIST are the same with the sizes in the MNIST dataset.

4.3.2 Results

Just like in the Fashion-MNIST case, we use the same parameters and training procedures in MNIST. Also, we restrict our attention to the labeled datasets with $N_L = 100, 600, 1000$.

After training for 10 times (i.e., experiments), we have the results in Table 4.8. Our first observation from the table is that the scores for the case where only labeled samples are used are lower than MNIST experiments. The reason might be similar to the one in Fashion-MNIST, our model is not suitable for classification of this dataset.

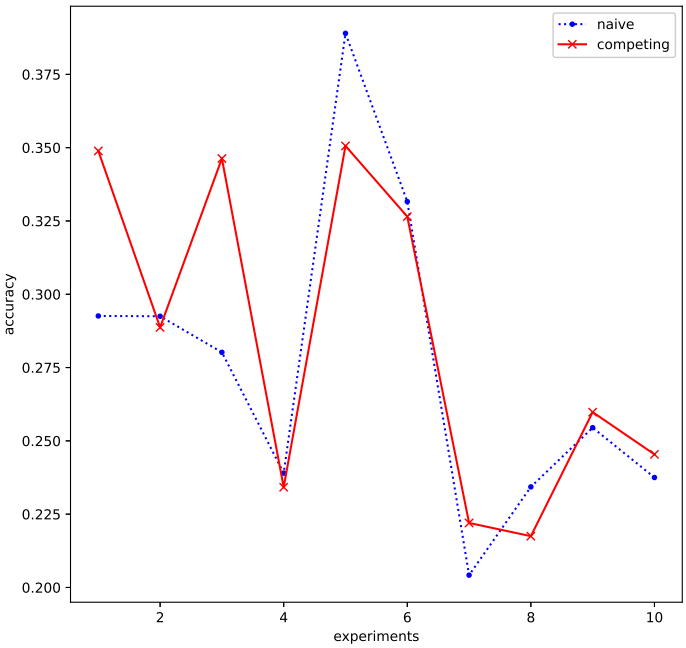
Secondly, we see that for $N_L = 100$, the test accuracy produced by the pseudo-labeling (including both methods, naive and competing) is lower than the case where the training includes only labeled samples. One possible explanation for this might be the following: Since there is a weak supervision, when the warm-up phase finishes, neither of the methods can achieve to involve the true labels in their pseudo-labeling procedure sufficiently enough. Thus, the model encounters highly noisy training and the performance on the test set degrades. Finally, once again for all N_L, the accuracy obtained via the competing method is greater than the naive method.

Table 4.8: KMNIST. Average test accuracy and standard deviation of 10 experiments. Bold number indicates the highest accuracy.

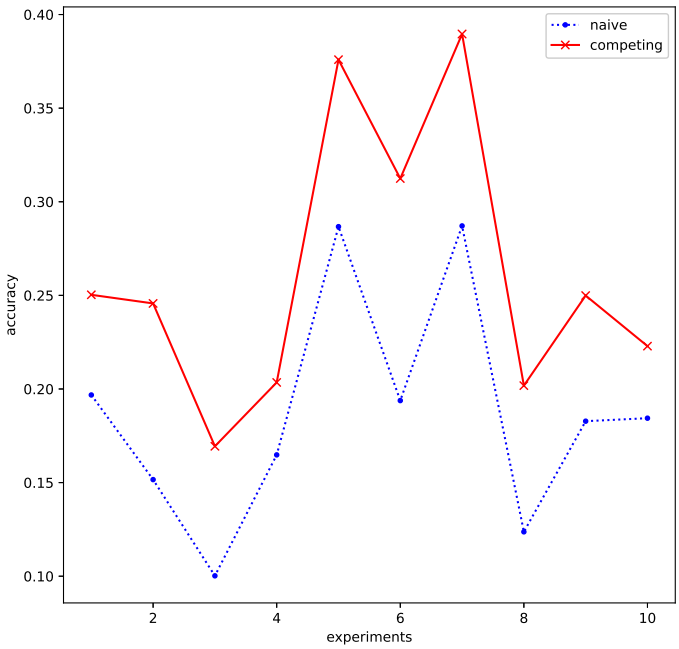
Labeled samples:	100	600	1000
CNN	$\mathbf{40.1 \pm 1.5}$	60.7 ± 1.0	66.0 ± 0.7
CNN + Naive	27.6 ± 5.4	62.5 ± 1.7	67.5 ± 1.9
CNN + Competing	28.4 ± 5.5	$\mathbf{63.8 \pm 1.2}$	$\mathbf{68.5 \pm 1.2}$

When we investigate the behavior of these two methods during the experiments, we obtain the results exhibited in Figure 4.10a. This figure indicates that in most of the experiments, the competing method produces greater test accuracy score; in some of them both methods produce quite similar results. And in average (as seen in Table 4.8), the competing one produces a greater test score compared to the naive one.

Lastly, we apply the mixup regularization method and the results turn out as in Table 4.9. We observe that when this regularization is adopted, the competing method is still superior in terms of generalization of the new data compared to the naive one for all number of labeled samples. Also, we see that after adding the regularization to the



(a) Without Mixup



(b) With Mixup

Figure 4.10: Accuracy vs experiments for KMNIST 100 labeled samples

training, the results do not get improved for $N_L = 100, 600$.

Now we analyze both methods' test accuracy scores during the experiments when the mixup is provided for the labeled case with the highest standard deviation (i.e., $N_L = 100$). Figure 4.10b shows that, similar to the previous cases, the accuracy obtained by the both methods varies in a similar manner. However, in all of the experiments, the scores produced by the competing method are greater than the naive one's.

Table 4.9: KMNIST (including mixup). Average test accuracy and standard deviation of 10 experiments. Bold number indicates the highest accuracy.

Labeled samples:	100	600	1000
CNN	$\mathbf{40.1 \pm 1.5}$	60.7 ± 1.0	66.0 ± 0.7
CNN + Naive	27.6 ± 5.4	62.5 ± 1.7	67.5 ± 1.9
CNN + Competing	28.4 ± 5.5	$\mathbf{63.8 \pm 1.2}$	68.5 ± 1.2
CNN + Naive + Mixup	18.7 ± 6.1	61.5 ± 3.2	67.8 ± 0.9
CNN + Competing + Mixup	26.2 ± 7.4	63.2 ± 1.8	$\mathbf{68.7 \pm 1.6}$

CHAPTER 5

SUMMARY AND CONCLUSION

In this thesis, we have investigated multi-class image classification problem in semi-supervised learning setting from the pseudo-label point of view. Specifically, we have taken into consideration originally proposed pseudo-labeling method in [16] and attempted to solve the problem of noisy training that results from the naive labeling technique via proposing the competing labels approach.

In our approach, we have first investigated the gradient or signal vector that the model receives and introduced a new loss function that allows us to obtain many negative signals (see Chapter 2). Later, we have investigated how to choose pseudo-labels and introduced two strategies namely, pseudo-labeling by fixed n and pseudo-labeling by threshold τ (see Chapter 3).

In the fourth chapter (Chapter 4), we have tested our method and compared it with the naive pseudo-labeling. Firstly, we have tested two methods on MNIST dataset and have shown that the competing method, regardless of the number of labeled samples, produces a greater classification accuracy on test set compared to the naive one. Moreover, we have observed that the discrepancy between the performances of two methods increases in the weak supervision cases (i.e., the number of labeled samples are 100 and 600). Furthermore, we have compared our highest scores with the best results of the originally proposed work [16], and have shown that our method shows a better performance for the cases of $100, 600, 1000$ labeled samples.

We have also tested two methods on the datasets Fashion-MNIST and KMNIST. Although our model is not well suited to classify these datasets, we have observed that

under the same training conditions, the competing method produces better generalization performance compared to the naive one for each of $100, 600$, and 1000 number of labeled samples. We have also noted that the discrepancy in terms of generalization performance reaches to the highest value (where the competing one's test score is greater than the naive one's) when the lowest supervision is present.

As a future work, to understand the efficiency of our approach better, especially its generalization performance for the case where the labeled samples are very limited, we plan to expand our work from the gray-scale images and to test our the method on other datasets such as images containing three channels. Also, to test the effectiveness of our method in terms of the number of classes of a dataset, we will consider the datasets with a various number of class categories that is ten in the current work.

As a final comment, we think that in pseudo-labeling based approaches, the gradient sent to the parametrized function is crucial to the performance of a model. So, it may not be quite helpful focusing only on the labeling strategy or merely taking into consideration the loss function because the signal vector is an outcome of the choice of both the labeling strategy and the loss function.

REFERENCES

[1] E. Arazo, D. Ortego, P. Albert, N. E. O'Connor, and K. McGuinness, Pseudo-labeling and confirmation bias in deep semi-supervised learning, in *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2020.

[2] J. S. Bridle, Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition, in *Neurocomputing*, pp. 227–236, Springer, 1990.

[3] O. Chapelle, B. Scholkopf, and A. Zien, Semi-supervised learning, IEEE Transactions on Neural Networks, 20(3), pp. 542–542, 2009.

[4] O. Chapelle and A. Zien, Semi-supervised classification by low density separation, in *International workshop on artificial intelligence and statistics*, pp. 57–64, PMLR, 2005.

[5] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, Deep learning for classical japanese literature, arXiv preprint arXiv:1812.01718, 2018.

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

[7] C. Doersch, A. Gupta, and A. A. Efros, Unsupervised visual representation learning by context prediction, in *Proceedings of the IEEE international conference on computer vision*, pp. 1422–1430, 2015.

[8] C. Finn, P. Abbeel, and S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

[9] C. Haase-Schütz, R. Stal, H. Hertlein, and B. Sick, Iterative label improvement: Robust training by confidence based filtering and dataset partitioning, in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 9483–9490, IEEE, 2021.

[10] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

[11] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, Label propagation for deep semi-supervised learning, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5070–5079, 2019.

[12] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.

[13] S. Laine and T. Aila, Temporal ensembling for semi-supervised learning, in *ICLR (Poster)*, OpenReview.net, 2017.

[14] Y. LeCun and Y. Bengio, *Convolutional Networks for Images, Speech, and Time Series*, p. 255–258, MIT Press, Cambridge, MA, USA, 1998, ISBN 0262511029.

[15] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, Nature, 521(7553), pp. 436–444, 2015.

[16] D. H. Lee, Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks, in *Workshop on challenges in representation learning, ICML*, volume 3, p. 896, 2013.

[17] A. Oliver, A. Odena, C. Raffel, E. D. Cubuk, and I. J. Goodfellow, Realistic evaluation of deep semi-supervised learning algorithms, in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, p. 3239–3250, Curran Associates Inc., Red Hook, NY, USA, 2018.

[18] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, Automatic differentiation in pytorch, in *NIPS-W*, 2017.

[19] M. N. Rizve, K. Duarte, Y. S. Rawat, and M. Shah, In defense of pseudo-labeling: An uncertainty-aware pseudo-label selection framework for semi-supervised learning, in *International Conference on Learning Representations*, 2021.

[20] D. Tanaka, D. Ikami, T. Yamasaki, and K. Aizawa, Joint optimization framework for learning with noisy labels, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5552–5560, 2018.

[21] V. Verma, K. Kawaguchi, A. Lamb, J. Kannala, A. Solin, Y. Bengio, and D. Lopez-Paz, Interpolation consistency training for semi-supervised learning, Neural Networks, 145, pp. 90–106, 2022.

[22] H. Xiao, K. Rasul, and R. Vollgraf, Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms, arXiv, abs/1708.07747, 2017.

[23] B. Xu, N. Wang, T. Chen, and M. Li, Empirical evaluation of rectified activations in convolutional network, arXiv preprint arXiv:1505.00853, 2015.

[24] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, mixup: Beyond empirical risk minimization, in *International Conference on Learning Representations*, 2018.