DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2017

A New Scheme for Training ReLU-Based Multi-Layer Feedforward Neural Networks

HAO WANG

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

A New Scheme for Training ReLU-Based Multi-Layer Feedforward Neural Networks

HAO WANG

Master in Computer Science, 120 credits
E-mail: hawan@kth.se
Date: November 10, 2017
Supervisor: Saikat Chatterjee
Examiner: Mario Romero Vega
Swedish title: Ett nytt system för att träna ReLU-baserade och framkopplade neurala nätverk med flera lager
School of Computer Science and Communication

Abstract

A new scheme for training Rectified Linear Unit (ReLU) based feedforward neural networks is examined in this thesis. The project starts with the row-by-row updating strategy designed for Single-hidden Layer Feedforward neural Networks (SLFNs). This strategy exploits the properties held by ReLUs and optimizes each row in the input weight matrix individually, under the common optimization scheme. Then the Direct Updating Strategy (DUS), which has two different versions: Vector-Based Method (VBM) and Matrix-Based Method (MBM), is proposed to optimize the input weight matrix as a whole. Finally DUS is extended to Multi-hidden Layer Feedforward neural Networks (MLFNs). Since the extension, for general ReLU-based MLFNs, faces an initialization dilemma, a special structure MLFN is presented.

Verification experiments are conducted on six benchmark multi-class classification datasets. The results confirm that MBM algorithm for SLFNs improves the performance of neural networks, compared to its competitor, regularized extreme learning machine. For most datasets involved, MLFNs with the proposed special structure perform better when adding extra hidden layers.

Sammanfattning

Ett nytt schema för träning av rektifierad linjär enhet (ReLU)-baserade och framkopplade neurala nätverk undersöks i denna avhandling. Projektet börjar med en rad-för-rad-uppdateringsstrategi designad för framkopplade neurala nätverk med ett dolt lager (SLFNs). Denna strategi utnyttjar egenskaper i ReLUs och optimerar varje rad i inmatningsviktmatrisen individuellt, enligt en gemensam optimeringsmetod. Därefter föreslås den direkta uppdateringsstrategin (DUS), som har två olika versioner: vektorbaserad metod (VBM) respektive matrisbaserad metod (MBM), för att optimera ingångsviktmatrisen som helhet. Slutli- gen utvidgas DUS till framkopplade neurala nätverk med flera lager (MLFN). Eftersom utvidgningen för generella ReLU-baserade MLFN står inför ett initieringsdilemma presenteras därför en MLFN med en speciell struktur.

Verifieringsexperiment utförs på sex datamängder för klassificering av flera klasser. Resultaten bekräftar att MBM-algoritmen för SLFN förbättrar prestanda hos neurala nätverk, jämfört med konkurrenten, den regulariserade extrema inlärningsmaskinen. För de flesta använda dataset, fungerar MLFNs med den föreslagna speciella strukturen bättre när man lägger till extra dolda lager.

Acknowledgments

This thesis project for the master's degree in Computer Science was conducted in Department of Information Science and Technology, at KTH Royal Institute of Technology. I am thankful to Saikat Chatterjee, Mostafa Sadeghi, Alireza Mahdavi Javid and Mario Romero Vega who have helped me in the research and examined my work. I am also grateful to my parents and all my beloved ones that have supported me during the master's programme financially and spiritually. Lastly, a special acknowledgment to Xiangyu Lei who has reviewed the first draft of this thesis report.

致谢

首先，感谢瑞典皇家理工学院(Kungliga Tekniska högskolan)信息科学与技术部为我提供完成硕士毕业设计的场所以及其他物质资助。其次，诚恳地感谢我的导师Saikat Chatterjee，同组博士生Mostafa Sadeghi和 Alireza Mahdavi Javid，以及考核官Mario Romero Vega对我研究的帮助与审核。再者，真挚地感谢我的父母以及所有我所挚爱的人们在整个硕士学习期间对我的支持。最后，感谢我的舍友雷翔宇同学帮助我修改论文初稿中的拼写等错误。

Contents

Chapter 1

Introduction

Due to an increasingly large amount of data and much more computing power available, Machine Learning (ML) [Bis06] has become more and more popular among data analysis and many other applications, including auto-driving system, face recognition, robots, etc. Machine learning is typically used when there is a large amount of data, within which some patterns exist, available for use, but it is extremely difficult or infeasible to define rules mathematically. This kind of problems often depends on several subtle factors, with the relations being extremely complicated, and people is even not sure about whether the problem depends on a certain factor or not [SV08]. For example, recognizing written digits from images can be very easy for a human to perform, but it is almost impossible to derive analytical rules and write a program to recognize a digit from a picture. Hopefully, machine learning algorithms can be applied to solve similar problems.

1.1 Background

Machine learning is a large topic and there are dozens of machine learning algorithms, such as Support Vector Machine (SVM) [CY11], Random Forest (RF) [ZM12] and Artificial Neural Network (ANN) [Roj96] available both in research and industry. Among all kinds of learning methods, ANN is a family of very poplar ML algorithms, in which Deep Learning is widely known by the public because of Google's AlphaGo [MWK16].

There exist various types of ANNs, but this project focuses on Feedforward Neural Networks (FNNs), where feedforward means that con-

nections between units do not form a cycle [Zel+94]. Typical ANNs are commonly made up of basic units called perceptrons, and an illustration of a perceptron can be found in Fig. 1.1. In the figure, values $x_1, \ldots, x_n$ denote the inputs, with $w_1, \ldots, w_n$ representing the corresponding weights. It is notable that there is a constant input 1 to the perceptron, and together with its weight factor b, they provide a bias. The components marked with $\mathbf{\Sigma}$ and $\mathbf{g}$ are a summation and an activation unit. The predicted output is then denoted as $\hat{t}$.

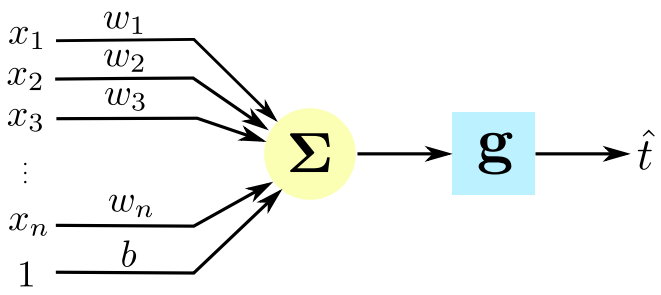


Figure 1.1: An illustration of a perceptron [Roj96].

In output and input layers, perceptrons often have linear activation functions, specifically $g(x) = x$. On the contrary, hidden layer units are usually combined with non-linear activation functions, such as Rectified Linear Unit (ReLU), sigma function, etc. A detailed list of popular non-linear activation functions can be found in Appendix A. And ReLU will be covered in detail in Chapter 2.

Fig. 1.2 shows the architecture of a neural network with only one hidden layer. In the figure, column vectors $\mathbf{x} \in \mathbb{R}^P$ and $\hat{\mathbf{t}} \in \mathbb{R}^Q$ are an input and output of the network, where P is the input dimension and Q is that of the output. For the hidden layer, input and output are denoted as $\mathbf{z} \in \mathbb{R}^H$ and $\mathbf{y} \in \mathbb{R}^H$ respectively, with H being the hidden dimension. $\mathbf{W}$ and $\mathbf{O}$ are the input and output weight matrix, while $\mathbf{X}$ and $\mathbf{T}$ denote the feature and label matrix, respectively, of a dataset. According to these notations, the cost function can be defined as the summation of differences, measured by l_p norms, between predictions and real labels over the whole dataset, i.e.

$$C(\mathbf{O}, \mathbf{W}) = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \hat{\mathbf{t}}\|_p^p = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \mathbf{O} \cdot g(\mathbf{W}\mathbf{x})\|_p^p. \quad (1.1)$$

This cost is the target function that we want to minimize when train-

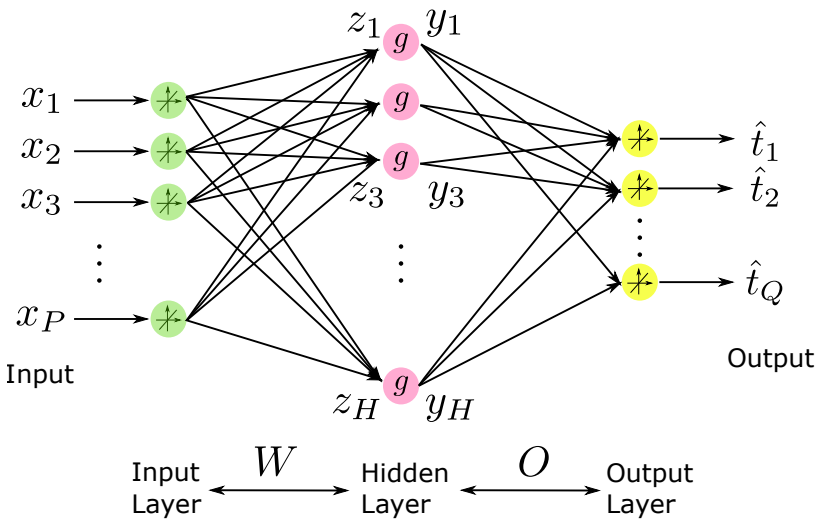


Figure 1.2: An illustration of a single hidden layer FNN.

ing the neural network. Unfortunately, due to the existence of the non-linear activation function $g(.)$, the minimization problem becomes non-convex, which means that the optimal solution cannot be found efficiently. Therefore, the famous gradient-decent algorithm, Back-Propagation (BP), was proposed for training FNNs. BP algorithm can date back to 1970s and its origin can be considered as automatic differentiation, fist published by Seppo L. [Lin70]. However, BP algorithm usually takes a long time and can easily stuck at different local optima. In order to achieve a good performance, this method can take a large number of iterations, especially when dealing with a complicated problem and a large network being applied. Some fundamental introductions of BP algorithm and its variants can be found in [Pat+14; Dre90; Li+09]. Also, refer to [Kri07; Roj96; MOM12] for more general information on ANNs and practical implementation tricks.

1.2 Related Work

Since back-propagation is the dominating training algorithm for FNNs, it is natural to ask can we design a new algorithm to train artificial neural networks? The answer is positive and one of the most successful and famous attempts is Extreme Learning Machine (ELM). To speed up the training process and improve the generalization performance,

Guang-Bin Huang et al. [HZS04; HZS06] proposed ELMs for training Single-hidden Layer Feedforward neural Networks (SLFNs). Their method is based on the common optimization scheme and proved to not only achieve minimal errors but also lead to solutions with the smallest norms. The main advantages of ELMs are: fast learning speed, simple implementation and good generalization performance. Huang et al. also showed that a SLFN with random hidden nodes has universal approximation ability [HCS06]. Based on their simulation results, ELM is qualified with very fast learning speed and good generalization performance in both regression and multi-class classification applications [Hua+12]. A good explanation of differences and relations between ELM and other popular machine learning algorithms, such as SVM and its variants, can be found in [Hua15].

There also exist a large number of studies focusing on other applications of ELMs. Kasun [Kas+13] and his colleges applied ELMs on unsupervised learning, reported the results of using ELM as an auto-encoder for feature representation learning and compared the results with other popular deep neural networks. To handle large-scale data, Lin et al. [Lin+13] proposed an mechanism to outsource ELMs in cloud computing. They showed that their mechanism significantly improved the learning speed of original ELM algorithm by conducting extensive experiments and claimed that they were the first one who had managed to outsource ELMs. The use of ELM in visualization was explored by Akusok et al. [Alu+13] They presented a nonlinear visualization technique, where ELM algorithm was used to estimate the reconstruction errors and achieved extremely fast error estimation speeds.

1.3 Thesis Project

Inspired by the strict mathematical justification of ELM, the thesis focuses on developing new training algorithms for ReLU-based FNNs, under the common optimization scheme. The verification simulations are carried out in Department of Information Science and Engineering, using computing resources available in School of Electrical Engineering, KTH. Due to the limited time and computational power, the experiments focus on some chosen benchmark multi-class classification datasets.

The remainder of t his report is organized as follows. Theory, methods, and mathematical derivations are included in Chapter 2. Specifically, in Section 2.1, the row-by-row strategy is proposed as the starting point of this thesis research. Section 2.2 describes the fundamental direct updating strategy for SLFNs. Section 2.3 and Section 2.4 present the extension to double-hidden layer and multi-hidden layer FNNs, respectively. All simulation results are summarized and illustrated in Chapter 3, while conclusions, future work and other topics are discussed in Chapter 4.

Chapter 2

Theory and Methods

In this chapter, the row-by-row updating strategy for SLFNs is firstly introduced as the origin of this thesis research. Then the direct updating strategy for SLFNs is proposed as the main method and outcome. Finally, attempts of extending the direct updating strategy to networks with more than one hidden layer are included at the end of this chapter.

2.1 Row-By-Row Updating Strategy

The row-by-row updating strategy is the starting point of this research and perfectly reflects the main idea that how ReLU properties can be exploited to update weight matrices.

2.1.1 ReLU-Based SLFN

A rectified linear unit, also called rectifier, is an activation function defined as follows:

$$g(x) = \max(0, x), \tag{2.1}$$

with x denoting the input to the neuron and $g(x)$ being the output. This activation function is also called positive linear transfer function, which maps positive inputs to themselves as outputs and maps negative inputs to zeros. ReLUs are qualified with strong mathematical motivations and biological justifications [HSS03], since they only response to positive inputs and stay in inactive states when negative values are fed, therefore, given random inputs to a network based on

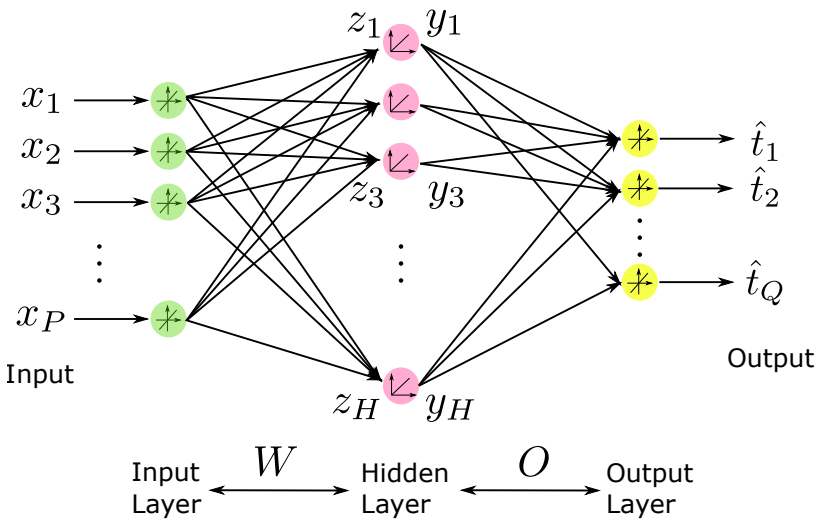


Figure 2.1: An illustration of a ReLU-based SLFN.

ReLUs, there will be on average half of the neurons stay in inactive modes.

Now construct a single hidden layer feedforward neural network, let column vectors $\mathbf{x} \in \mathbb{R}^P$ and $\hat{\mathbf{t}} \in \mathbb{R}^Q$ be an input and output of the network, where P is the input dimension and Q is the output dimension. $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}]$ and $\mathbf{T} = [\mathbf{t}^{(1)}, \mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(N)}]$ are the feature matrix and label (target) matrix in a dataset, with N denoting the number of training samples. The input weight matrix $\mathbf{W}$, also known as the input matrix, connects the input and hidden layer. $\mathbf{W}$ is a $P \times H$ matrix, where H represents the number of nodes in the hidden layer. Let column vectors $\mathbf{z} = [z_1, z_2, \ldots, z_H]^T$ and $\mathbf{y} = [y_1, y_2, \ldots, y_H]^T$ be an input and output of the hidden layer respectively,

$$\begin{aligned} \mathbf{z} &= \mathbf{W} \cdot \mathbf{x}, \\ \mathbf{y} &= g(\mathbf{z}) = g(\mathbf{W} \cdot \mathbf{x}), \end{aligned} \tag{2.2}$$

where $g(.)$ is a rectified linear unit.

The output weight matrix, also called the output matrix, $\mathbf{O} \in \mathbb{R}^{Q \times H}$ connects the hidden layer and output layer. Define a prediction from the network as

$$\hat{\mathbf{t}} = \mathbf{O} \cdot g(\mathbf{W} \cdot \mathbf{x}). \tag{2.3}$$

Fig. 2.1 illustrates the structure and configurations of the ReLU-based

SLFN specified above.

2.1.2 Optimization

In order to optimize the weights in a network, an appropriate cost function should be introduced first. Here the cost function $C(\mathbf{O}, \mathbf{W})$ is defined as the summation of differences, which are measured by l_p norms, between predictions and targets over the whole dataset, i.e.

$$C(\mathbf{O}, \mathbf{W}) = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \hat{\mathbf{t}}\|_p^p = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \mathbf{O}\cdot g(\mathbf{W}\mathbf{x})\|_p^p. \quad (2.4)$$

If l_2 norms are used, the cost function is equivalent to

$$C(\mathbf{O}, \mathbf{W}) = \|\mathbf{T} - \hat{\mathbf{T}}\|_F^2 = \|\mathbf{T} - \mathbf{O}\cdot g(\mathbf{W}\mathbf{X})\|_F^2. \quad (2.5)$$

where $\|.\|_F^2$ denotes the Frobenius norm. From the definition, it is clear that the cost depends on both $\mathbf{W}$ and $\mathbf{O}$. In the row-by-row strategy, weight matrices are updated alternatively. And the output matrix $\mathbf{O}$, which is optimized by regularized ELM algorithm, will be examined first.

Initially, the weight matrix $\mathbf{W}$ is randomly generated from a standard Gaussian or a uniform distribution (typically between -1 and 1). Then the output weights can be learned by applying ELM algorithm, i.e. solving a least-square minimization problem,

$$\arg\min_{\mathbf{O}} \|\mathbf{T} - \mathbf{O}\cdot\mathbf{Y}\|_F^2. \quad (2.6)$$

To handle over-fitting problems, regularized ELM is proposed by Deng et al. [DZC09]. This method solves the minimization problem in Eq.(2.6) with an additional constraint on the Frobenius norm of $\mathbf{O}$,

$$\arg\min_{\mathbf{O}} \|\mathbf{T} - \mathbf{O}\cdot\mathbf{Y}\|_F^2 \text{ s.t. } \|\mathbf{O}\|_F^2 \leq \epsilon_O. \quad (2.7)$$

The other more convenient way is to add a regularization term in the minimization problem,

$$\arg\min_{\mathbf{O}} \|\mathbf{T} - \mathbf{O}\cdot\mathbf{Y}\|_F^2 + \lambda_O\|\mathbf{O}\|_F^2, \quad (2.8)$$

in which λ_O is some regularization factor and needs to be tuned. In

this optimization problem, $\mathbf{O}$ has a close form solution,

$$\mathbf{O}^* = \mathbf{T}\mathbf{Y}^T(\mathbf{Y}\mathbf{Y}^T + \lambda_O\mathbf{I})^{-1}, \tag{2.9}$$

with $\mathbf{O}^*$ being the optimal solution and $\mathbf{I}$ representing an identity matrix with proper dimensions.

After learning the output weights, in the row-by-row strategy, each row in $\mathbf{W}$ will be updated individually. With respect to its ith row $\mathbf{w}_i$, $\mathbf{W}$ can be written in a row vector form,

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_i \\ \vdots \\ \mathbf{w}_H \end{bmatrix} = \begin{bmatrix} \mathbf{W}_{i^-} \\ \mathbf{w}_i \\ \mathbf{W}_{i^+} \end{bmatrix}. \tag{2.10}$$

Now define a set $\tau_i \subset (\mathbf{X}, \mathbf{T})$ containing the data samples for which the inputs to the ith hidden node are positive, i.e.

$$\tau_i = \{(\mathbf{x}, \mathbf{t}) | z_i = \mathbf{w}_i\mathbf{x} \geq 0\}. \tag{2.11}$$

And its complement set, with respect to the full set $(\mathbf{X}, \mathbf{T})$, is denoted as τ_{i^c}, which means that $\tau_i \cup \tau_{i^c} = (\mathbf{X}, \mathbf{T})$ and $\tau_i \cap \tau_{i^c} = \emptyset$. The representations of $\mathbf{z}$ and $\mathbf{y}$ in vector forms are defined as

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_{i^-} \\ z_i \\ \mathbf{z}_{i^+} \end{bmatrix} \text{ and } \mathbf{y} = \begin{bmatrix} \mathbf{y}_{i^-} \\ y_i \\ \mathbf{y}_{i^+} \end{bmatrix} = \begin{bmatrix} g(\mathbf{z}_{i^-}) \\ g(z_i) \\ g(\mathbf{z}_{i^+}) \end{bmatrix} = \begin{bmatrix} g(\mathbf{W}_{i^-}\mathbf{x}) \\ g(\mathbf{w}_i\mathbf{x}) \\ g(\mathbf{W}_{i^+}\mathbf{x}) \end{bmatrix}. \tag{2.12}$$

Note that, for $(\mathbf{x}, \mathbf{t}) \in \tau_i$, we have

$$\mathbf{y} = \begin{bmatrix} g(\mathbf{W}_{i^-}\mathbf{x}) \\ g(\mathbf{w}_i\mathbf{x}) \\ g(\mathbf{W}_{i^+}\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{i^-} \\ \mathbf{w}_i\mathbf{x} \\ \mathbf{y}_{i^+} \end{bmatrix}, \tag{2.13}$$

otherwise, if $(\mathbf{x}, \mathbf{t}) \in \tau_{i^c}$,

$$\mathbf{y} = \begin{bmatrix} g(\mathbf{W}_{i^-}\mathbf{x}) \\ g(\mathbf{w}_i\mathbf{x}) \\ g(\mathbf{W}_{i^+}\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \mathbf{y}_{i^-} \\ 0 \\ \mathbf{y}_{i^+} \end{bmatrix}. \tag{2.14}$$

The similar column vector notation of $\mathbf{O}$, with respect to its ith column $\mathbf{o}_i$, is

$$\mathbf{O} = [\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_i, \ldots, \mathbf{o}_Q] = [\mathbf{O}_{i^-}, \mathbf{o}_i, \mathbf{O}_{i^+}]. \tag{2.15}$$

Based on all these vector notations, the cost function Eq.(2.4) can be written as

$$C(\mathbf{W}) = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t}-\mathbf{O}\cdot\mathbf{y}\|_p^p = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t}-(\mathbf{O}_{i^-}\mathbf{y}_{i^-}+\mathbf{o}_i y_i+\mathbf{O}_{i^+}\mathbf{y}_{i^+})\|_p^p. \tag{2.16}$$

Considering the properties presented in Eq.(2.13) and Eq.(2.14), the cost can be split into two parts: one for $(\mathbf{x}, \mathbf{t}) \in \tau_i$ and the other one for $(\mathbf{x}, \mathbf{t}) \in \tau_{i^c}$,

$$C(\mathbf{W}) = \sum_{(\mathbf{x},\mathbf{t})\in\tau_i} \|(\mathbf{t}-\mathbf{O}_{i^-}\mathbf{y}_{i^-}-\mathbf{O}_{i^+}\mathbf{y}_{i^+})-\mathbf{o}_i y_i\|_p^p + \sum_{(\mathbf{x},\mathbf{t})\in\tau_{i^c}} \|\mathbf{t}-\mathbf{O}_{i^-}\mathbf{y}_{i^-}-\mathbf{O}_{i^+}\mathbf{y}_{i^+}\|_p^p. \tag{2.17}$$

In the row-by-row strategy, each row is updated individually, therefore, all terms that do not depend on $\mathbf{w}_i$ can be considered as constants. For simplicity, let us replace the second summation in Eq.(2.17) by C_Σ and define $\tilde{\mathbf{t}} = \mathbf{t} - \mathbf{O}_{i^-}\mathbf{y}_{i^-} - \mathbf{O}_{i^+}\mathbf{y}_{i^+}$. Then the cost is further simplified as follows:

$$C(\mathbf{w}_i) = \sum_{(\mathbf{x},\mathbf{t})\in\tau_i} \|\tilde{\mathbf{t}} - \mathbf{o}_i y_i\|_p^p + C_\Sigma = \sum_{(\mathbf{x},\mathbf{t})\in\tau_i} \|\tilde{\mathbf{t}} - \mathbf{o}_i\mathbf{w}_i\mathbf{x}\|_p^p + C_\Sigma. \tag{2.18}$$

Here, we consider $\mathbf{O}$ is fixed and the cost in Eq.(2.18) only depends on $\mathbf{w}_i$, therefore, it is possible for us to formulate an optimization problem as

$$\arg\min_{\mathbf{w}_i} \; C(\mathbf{w}_i) \text{ s.t. } \begin{cases} \|\mathbf{w}_i\|_p^p \le \epsilon_w \\ \forall(\mathbf{x},\mathbf{t}) \in \tau_i, \mathbf{w}_i\mathbf{x} \ge 0 \end{cases}. \tag{2.19}$$

This minimization problem is convex and can be solved efficiently. The row-by-row updating strategy optimizes $\mathbf{W}$ and $\mathbf{O}$ alternatively and the algorithm is summarized as follows:

Algorithm RBR: Given a training set $(\mathbf{X}, \mathbf{T})$, and a SLFN with a ReLU-based hidden layer,

Step 1: randomly assign the input weight matrix $\mathbf{W}$. Typically, entries in $\mathbf{W}$ are generated from a standard Gaussian distribution or a uniform distribution between -1 and 1.

Step 2: optimize the output weight matrix $\mathbf{O}$ according to Eq.(2.9), i.e. regularized ELM algorithm.

Step 3: update each row in $\mathbf{W}$ by solving the convex optimization problem described in Eq.(2.19).

Step 4: repeat step 2 and step 3 until the training error converges or reach the predefined maximum iteration number.

2.2 Direct Updating Strategy

In Section 2.1, the input weights are optimized by searching solutions to a series of convex optimization problems and learn each row individually. However, is it possible to update $\mathbf{W}$ as a whole in one optimization problem instead of several ones? The answer is yes. This section describes the Direct Updating Strategy (DUS) in two versions: Vector-Based Method (VBM) and Matrix-Based Method (MBM). VBM is more straightforward, in terms of mathematical derivation, and more memory-friendly, while MBM is designed for its efficient implementation.

2.2.1 Vector-Based Method for SLFNs

Let us examine the characteristics of ReLU output first. Column vector $\mathbf{y}$, an output of the ReLU-based hidden layer, contains rather positive numbers or zeros, specifically,

$$y_i = \begin{cases} z_i = \mathbf{w}_i\mathbf{x} & \text{if } z_i \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.20}$$

We define a new input weight matrix $\tilde{\mathbf{W}}_\mathbf{x}$ for each pair of $(\mathbf{x}, \mathbf{t}) \in (\mathbf{X}, \mathbf{T})$ as

$$\tilde{\mathbf{W}}_\mathbf{x} = \begin{bmatrix} \tilde{\mathbf{w}}_\mathbf{x}^1 \\ \tilde{\mathbf{w}}_\mathbf{x}^2 \\ \vdots \\ \tilde{\mathbf{w}}_\mathbf{x}^H \end{bmatrix} \text{ and } \tilde{\mathbf{w}}_\mathbf{x}^i = \begin{cases} \mathbf{w}_i & \text{if } \mathbf{w}_i\mathbf{x} \geq 0 \\ \mathbf{0} & \text{otherwise} \end{cases}. \tag{2.21}$$

The new weight matrix $\tilde{\mathbf{W}}_\mathbf{x}$ depends on $\mathbf{x}$, and it just sets rows, such that $\mathbf{w}_i\mathbf{x} \leq 0$, in $\mathbf{W}$ to zero vectors. This manipulation can be achieved by multiplying a diagonal matrix $\mathbf{\Omega}_\mathbf{x}$, with only ones and zeros on the

diagonal, to the left of $\mathbf{W}$, i.e.

$$\tilde{\mathbf{W}}_{\mathbf{x}} = \mathbf{\Omega}_{\mathbf{x}}\mathbf{W}\ ,\ \mathbf{\Omega}_{\mathbf{x}} = \begin{bmatrix} \omega_{\mathbf{x}}^1 & 0 & \dots & 0 \\ 0 & \omega_{\mathbf{x}}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \omega_{\mathbf{x}}^H \end{bmatrix} \text{ and } \omega_{\mathbf{x}}^i = \begin{cases} 1 & \text{if } \mathbf{w}_i\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.22}$$

According to the notations in Eq.(2.22), a new cost function is defined as follows:

$$C(\mathbf{W}) = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t}-\mathbf{O}\tilde{\mathbf{W}}_{\mathbf{x}}\mathbf{x}\|_p^p = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t}-\mathbf{O}\cdot\mathbf{\Omega}_{\mathbf{x}}\mathbf{W}\cdot\mathbf{x}\|_p^p\ . \tag{2.23}$$

Now, let $\tilde{\mathbf{O}}_{\mathbf{x}} = \mathbf{O}\mathbf{\Omega}_{\mathbf{x}}$, and formulate a minimization problem as

$$\begin{aligned} \arg\min_{\mathbf{W}}\ & \lambda_W\|\mathbf{W}\|_F^2 + \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \tilde{\mathbf{O}}_{\mathbf{x}}\mathbf{W}\cdot\mathbf{x}\|_p^p \\ & \text{s.t. } \forall(\mathbf{x},\mathbf{t}) \in (\mathbf{X},\mathbf{T}), \mathbf{\Omega}_{\mathbf{x}}\mathbf{W}\mathbf{x} \geq \mathbf{0}, \end{aligned} \tag{2.24}$$

in which λ_W is a regularization factor for $\mathbf{W}$. The optimization problem presented in Eq.(2.24) allows us to update the input weight matrix by solving only one minimization problem. Unfortunately, this problem involves a large number of matrix multiplications and summations, which makes it difficult to be implemented efficiently. To speed up the implementation at the expense of using more memory, the optimization problem can be formulated in a matrix form.

2.2.2 Matrix-Based Method for SLFNs

Matrices $\mathbf{Z} = [\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(N)}]$ and $\mathbf{Y} = [\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(N)}]$ contain inputs and outputs of the hidden layer. According to Section 2.1.2, the cost function in a matrix form is

$$C(\mathbf{W}, \mathbf{O}) = \|\mathbf{T} - \hat{\mathbf{T}}\|_F^2 = \|\mathbf{T} - \mathbf{O}\cdot g(\mathbf{W}\mathbf{X})\|_F^2. \tag{2.25}$$

In VBM, ReLUs are replaced by several matrices that depend on each input $\mathbf{x} \in \mathbf{X}$. A similar trick can also be applied in MBM. Let us define a $H \times N$ matrix $\mathbf{\Phi}$, whose entries are only ones and zeros, indicating the signs of entries in $\mathbf{Z}$. Specifically, 1 means the corresponding element

in $\mathbf{Z}$ is larger than or equal to 0, 0 means otherwise,

$$\mathbf{\Phi} = [\phi_{ij}] \text{ , } \mathbf{Z} = [z_{ij}] \text{ , } i = 1, \ldots, H \text{ , } j = 1, \ldots, N \text{ and } \phi_{ij} = \begin{cases} 1 & \text{if } z_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases} . \tag{2.26}$$

According to the definition, it is obvious that the matrix $\mathbf{\Phi}$ depends on $\mathbf{W}$, since the feature matrix $\mathbf{X}$ is fixed. With the help of $\mathbf{\Phi}$ and the Hadamard product, also known as the entry-wise product and denoted by $\odot$, the non-linear transfer function $g(.)$ can be replaced. Then the cost function becomes

$$C(\mathbf{W}, \mathbf{O}) = \|\mathbf{T} - \mathbf{O} \cdot (\mathbf{\Phi} \odot (\mathbf{W} \cdot \mathbf{X}))\|_F^2 . \tag{2.27}$$

Based on this cost, the regularized minimization problem is formulated as

$$\begin{aligned} \arg\min_{\mathbf{W}} \|\mathbf{T} - \mathbf{O} \cdot \mathbf{\Phi} \odot (\mathbf{W}\mathbf{X})\|_F^2 + \lambda_W \|\mathbf{W}\|_F^2 \\ \text{s.t. } \mathbf{\Phi} \odot (\mathbf{W}\mathbf{X}) \geq \mathbf{0}. \end{aligned} \tag{2.28}$$

The optimization problem above can be solved by Alternating Direction Method of Multipliers (ADMM) [Fit+14; GM76], which is a gradient-based iterative algorithm for handling convex optimization problems. Let us define two dual variables: $\mathbf{Z_X} = \mathbf{\Phi} \odot \mathbf{Z_W}$ and $\mathbf{Z_W} = \mathbf{WX}$, then a Lagrangian function L is formulated as

$$L = \frac{1}{2}\|\mathbf{T} - \mathbf{O} \cdot \mathbf{Z_X}\|_F^2 + \frac{1}{2\mu}(\|\mathbf{WX} - \mathbf{Z_W}\|_F^2 + \|\mathbf{\Phi} \odot \mathbf{Z_W} - \mathbf{Z_X}\|_F^2 + \lambda_W \|\mathbf{W}\|_F^2), \tag{2.29}$$

where μ is the penalty factor in ADMM algorithm and λ_W is a factor that imposes regularization on $\mathbf{W}$. The dual problem of Eq.(2.28) is written as

$$\arg\min_{\mathbf{W}} \ L \text{ s.t. } \mathbf{Z_X} \geq \mathbf{0}. \tag{2.30}$$

In oder to update $\mathbf{Z_X}$, take the derivative of L with respect to $\mathbf{Z_X}$, and set the derivative to $\mathbf{0}$, then we can obtain a close form updating equation for $\mathbf{Z_X}$ as

$$\mathbf{Z_X^*} = \mathbf{A}^{-1}\mathbf{B} \text{ and } \begin{cases} \mathbf{A} = \mathbf{O}^T\mathbf{O} + \frac{1}{\mu}\mathbf{I} \\ \mathbf{B} = \mathbf{O}^T\mathbf{T} + \frac{1}{\mu}(\mathbf{\Phi} \odot \mathbf{Z_W}) \end{cases} , \tag{2.31}$$

in which $\mathbf{I}$ denotes an identity matrix of proper dimensions. Due to the constraint $\mathbf{Z_X} \geq \mathbf{0}$ in minimization problem (2.30), simply set all negative entries in $\mathbf{Z_X^*}$ to zeros, after updating $\mathbf{Z_X}$ according to Eq.(2.31).

Similarly, we can obtain updating equations for $\mathbf{Z_W}$ and $\mathbf{W}$ as (for detailed mathematical derivations, please refer to Appendix B.)

$$\begin{aligned} \mathbf{Z_W^*} &= \mathbf{D} \oslash \mathbf{C} \text{ and } \begin{cases} \mathbf{C} = \mathbf{\Phi} + 1 \\ \mathbf{D} = \mathbf{\Phi} \odot \mathbf{Z_X} + \mathbf{WX} \end{cases}, \\ \mathbf{W}^* &= \mathbf{EF}^{-1} \text{ and } \begin{cases} \mathbf{E} = \mathbf{Z_W} \cdot \mathbf{X}^T \\ \mathbf{F} = \mathbf{XX}^T + \lambda_W \cdot \mathbf{I} \end{cases}, \end{aligned} \tag{2.32}$$

with notation $\oslash$ denoting the entry-wise division. It is notable that adding a regularization term not only imposes a constraint on $\|\mathbf{W}\|_F^2$, but also increase the non-singularity of the square matrix $\mathbf{F}$, which makes it possible to obtain a close form solution. It is encouraging to see that all solutions are in close forms, otherwise random noises or a gradient-descent method is needed.

The whole matrix-based algorithm can be summarized as follows:
Algorithm MBM: Given a training set $(\mathbf{X}, \mathbf{T})$, and a SLFN with a ReLU-based hidden layer,

Step 1: randomly assign the input weight matrix $\mathbf{W}$ (from a standard Gaussian or a uniform distribution between -1 and 1).

Step 2: learn the output weight matrix $\mathbf{O}$ according to Eq.(2.9).

Step 3: update variables $\mathbf{Z_X}$, $\mathbf{Z_W}$ and $\mathbf{W}$ *sequentially* according to Eq.(2.31,2.32), then compute a new $\mathbf{\Phi}$ corresponds to $\mathbf{W}^*$.

Step 4: repeat step 2 and step 3 until the training error converges or reach the maximum iteration number.

2.3 Training Double-Hidden Layer FNNs

Before addressing general multi-hidden layer networks, it is better to consider FNNs with only one more hidden layer first. In this section, the direct updating strategy, described in Section 2.2, is extended to Double-hidden Layer Feedforward neural Networks (DLFNs) with only ReLU-based hidden layers. The extension of the vector-based method is straightforward, while the matrix-based algorithm needs more efforts in derivations.

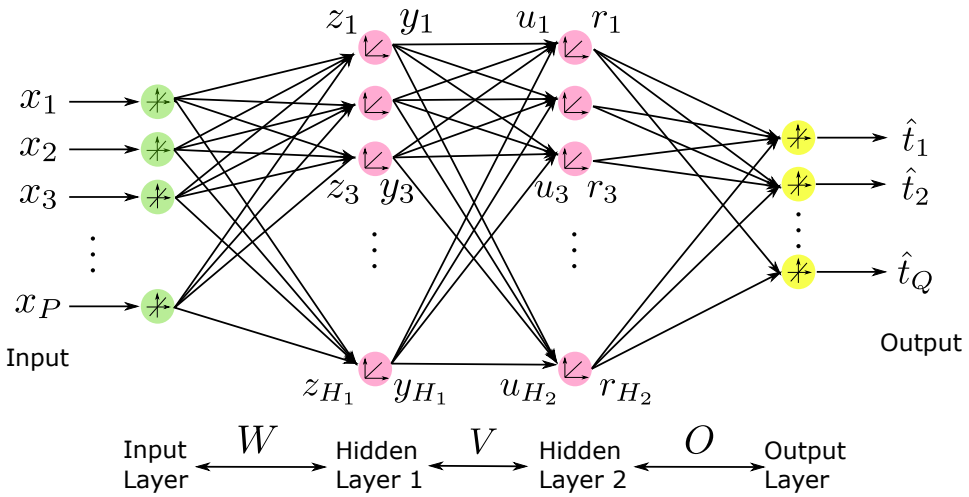


Figure 2.2: An illustration of a ReLU-based DLFN.

2.3.1 Network Setups

Given a DLFN, P and Q still represent the input and output dimension. Matrices $\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}]$ and $\hat{\mathbf{T}} = [\hat{\mathbf{t}}^{(1)}, \hat{\mathbf{t}}^{(2)}, \ldots, \hat{\mathbf{t}}^{(N)}]$ denote the input and output of the DLFN, respectively. Let H_1 and H_2 be the number of hidden nodes in the first and second hidden layer, respectively. Column vectors $\mathbf{z} = [z_1, z_2, \ldots, z_{H_1}]^T$ and $\mathbf{y} = [y_1, y_2, \ldots, y_{H_1}]^T$ are an input and output of the first hidden layer, and similarly, vectors $\mathbf{u} = [u_1, u_2, \ldots, u_{H_2}]^T$ and $\mathbf{r} = [r_1, r_2, \ldots, r_{H_2}]^T$ denote that of the second hidden layer. Matrices $\mathbf{W}$ and $\mathbf{O}$ still are the input and output weight matrix, while the $H_1 \times H_2$ weight matrix connecting two hidden layers is denoted by $\mathbf{V}$. According to these setups, we have following relations:

$$\begin{cases} \mathbf{z} = \mathbf{W}\mathbf{x} \\ \mathbf{y} = g(\mathbf{z}) = g(\mathbf{W}\mathbf{x}) \\ \mathbf{u} = \mathbf{V}\mathbf{y} = \mathbf{V} \cdot g(\mathbf{W}\mathbf{x}) \\ \mathbf{r} = g(\mathbf{u}) = g(\mathbf{V} \cdot g(\mathbf{W}\mathbf{X})) \\ \hat{\mathbf{t}} = \mathbf{O}\mathbf{r} = \mathbf{O} \cdot g(\mathbf{V} \cdot g(\mathbf{W}\mathbf{x})) \end{cases} . \tag{2.33}$$

Note that all non-linear activation functions $g(.)$ in Eq.(2.33) are ReLUs. The illustration in Fig. 2.2 shows how the network is structured and configured.

2.3.2 Vector-Based Method for DLFNs

In Section 2.2.1, the matrix $\mathbf{\Omega_x}$ is defined to select several rows in $\mathbf{W}$. By the same approach, another diagonal matrix $\mathbf{\Theta_y}$ is defined to conduct similar row manipulations on $\mathbf{V}$,

$$\tilde{\mathbf{V}}_\mathbf{y} = \begin{bmatrix} \tilde{\mathbf{v}}_\mathbf{y}^1 \\ \tilde{\mathbf{v}}_\mathbf{y}^2 \\ \vdots \\ \tilde{\mathbf{v}}_\mathbf{y}^{H_2} \end{bmatrix} \text{ and } \tilde{\mathbf{v}}_\mathbf{y}^i = \begin{cases} \mathbf{v}_i & \text{if } \mathbf{v}_i\mathbf{y} \geq 0 \\ \mathbf{0} & \text{otherwise} \end{cases},$$
$$\tilde{\mathbf{V}}_\mathbf{y} = \mathbf{\Theta_y V}\ ,\ \mathbf{\Theta_y} = \begin{bmatrix} \theta_\mathbf{y}^1 & 0 & \dots & 0 \\ 0 & \theta_\mathbf{y}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \theta_\mathbf{y}^{H_2} \end{bmatrix} \text{ and } \theta_\mathbf{y}^i = \begin{cases} 1 & \text{if } \mathbf{v}_i\mathbf{y} \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.34}$$

Based on the relations and notations presented in Eq.(2.33,2.34), the cost function for a DLFN is obtained as

$$C(\mathbf{W}, \mathbf{V}, \mathbf{O}) = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \mathbf{O\Theta_y V\Omega_x W x}\|_p^p. \tag{2.35}$$

The output weight matrix $\mathbf{O}$ is updated by solving following least-square minimization problem with a regularization,

$$\arg\min_{\mathbf{O}}\ \lambda_O\|\mathbf{O}\|_F^2 + \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \mathbf{Or}\|_p^p\ . \tag{2.36}$$

Let us define $\tilde{\mathbf{O}}_\mathbf{y} = \mathbf{O\Theta_y}$, $\tilde{\mathbf{O}}_x = \mathbf{O\Theta_y V\Omega_x}$ and formulate the other two optimization problems to update $\mathbf{W}$ and $\mathbf{V}$ as

$$\begin{aligned} &\arg\min_{\mathbf{W}}\ C(\mathbf{W}, \mathbf{V}, \mathbf{O}) = \lambda_W\|\mathbf{W}\|_F^2 + \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \tilde{\mathbf{O}}_\mathbf{x}\mathbf{W}\mathbf{x}\|_p^p \\ &\text{s.t. } \forall(\mathbf{x},\mathbf{t}) \in (\mathbf{X},\mathbf{T}), \mathbf{\Omega_x W x} \geq \mathbf{0} \text{ and } \mathbf{\Theta_y V\Omega_x W x} \geq \mathbf{0}, \\ &\arg\min_{\mathbf{V}}\ C(\mathbf{W}, \mathbf{V}, \mathbf{O}) = \lambda_V\|\mathbf{V}\|_F^2 + \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \tilde{\mathbf{O}}_\mathbf{y}\mathbf{V}\mathbf{y}\|_p^p \\ &\text{s.t. } \forall(\mathbf{x},\mathbf{t}) \in (\mathbf{X},\mathbf{T}), \mathbf{\Theta_y V y} \geq \mathbf{0}. \end{aligned} \tag{2.37}$$

The proposed training algorithm for DLFNs alternatively updates $\mathbf{O}$, $\mathbf{W}$, $\mathbf{V}$... until the training error converges or the predefined maximum

iteration number is reached. The algorithm is summarized as follows:
Algorithm VBM2: Given a training set $(\mathbf{X}, \mathbf{T})$, and a DLFN with only ReLU-based hidden layers,

Step 1: properly initialize weight matrices $\mathbf{W}$ and $\mathbf{V}$.

Step 2: compute the output weight matrix $\mathbf{O}$ by solving the regularized least-square minimization problem formulated in Eq.(2.36).

Step 3: learn $\mathbf{W}$ and $\mathbf{V}$ *sequentially* according to optimization problems presented in Eq.(2.37).

Step 4: repeat step 2 and step 3 until the training error converges or the maximum iteration number is reached.

2.3.3 Matrix-Based Method for DLFNs

Like always, the weight matrix $\mathbf{O}$ is the easiest target to optimize. Even in DLFN cases, output weights still can be learned by searching the solution to a simple least-square minimization problem,

$$\arg\min_{\mathbf{O}} \|\mathbf{T} - \mathbf{O} \cdot \mathbf{R}\|_F^2 + \lambda_O \|\mathbf{O}\|_F^2. \tag{2.38}$$

The emphasis should be attached on $\mathbf{W}$, the input weight matrix. In a SLFN case, we have seen that the ReLU activation function can be replaced by a Hadamard product. Let us define two 0-1 matrices $\mathbf{\Phi}$ and $\mathbf{\Psi}$ for the hidden layers. Note that the matrix $\mathbf{\Phi}$ is the same as in a SLFN case except for its size change, while $\mathbf{\Psi}$ is defined as

$$\mathbf{\Psi} = [\psi_{ij}]\ ,\ \mathbf{U} = [u_{ij}]\ ,\ i = 1, \ldots, H_2\ ,\ j = 1, \ldots, N \text{ and } \psi_{ij} = \begin{cases} 1 & \text{if } u_{ij} \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.39}$$

By introducing $\mathbf{\Phi}$ and $\mathbf{\Psi}$, the cost function is written as

$$C(\mathbf{W}, \mathbf{V}, \mathbf{O}) = \|\mathbf{T} - \mathbf{O} \cdot (\mathbf{\Psi} \odot (\mathbf{V} \cdot (\mathbf{\Phi} \odot (\mathbf{W} \cdot \mathbf{X}))))\|_F^2. \tag{2.40}$$

Define following 4 dual variables for ADMM: $\mathbf{Z_W} = \mathbf{WX}$, $\mathbf{Z_X} = \mathbf{\Phi} \odot \mathbf{Z_W}$, $\mathbf{Z_V} = \mathbf{VZ_X}$ and $\mathbf{Z_U} = \mathbf{\Psi} \odot \mathbf{Z_V}$. With those notations, the Lagrangian function is formulated as

$$\begin{aligned} L(\mathbf{W}) =& \frac{1}{2}\|\mathbf{T} - \mathbf{O} \cdot \mathbf{Z_U}\|_F^2 + \frac{1}{2\mu}(\|\mathbf{WX} - \mathbf{Z_W}\|_F^2 + \|\mathbf{\Phi} \odot \mathbf{Z_W} - \mathbf{Z_X}\|_F^2 \\ &+ \|\mathbf{VZ_X} - \mathbf{Z_V}\|_F^2 + \|\mathbf{\Psi} \odot \mathbf{Z_V} - \mathbf{Z_U}\|_F^2 + \lambda_W \|\mathbf{W}\|_F^2) \end{aligned} \tag{2.41}$$

and the dual optimization problem for optimizing the input weight matrix $\mathbf{W}$ can be written as

$$\arg\min_{\mathbf{W}} \; L(\mathbf{W}) \text{ s.t. } \begin{cases} \mathbf{Z_X} \geq \mathbf{0} \\ \mathbf{Z_U} \geq \mathbf{0} \end{cases} . \tag{2.42}$$

By computing derivatives with respect to different variables, we will obtain updating equations as (refer to Appendix B for detail)

$$\begin{cases} \mathbf{Z}_\mathbf{U}^* = \mathbf{A}^{-1} \cdot \mathbf{B} \\ \mathbf{Z}_\mathbf{V}^* = \mathbf{D} \oslash \mathbf{C} \\ \mathbf{Z}_\mathbf{X}^* = \mathbf{E}^{-1} \cdot \mathbf{F} \\ \mathbf{Z}_\mathbf{W}^* = \mathbf{H} \oslash \mathbf{G} \\ \mathbf{W}^* = \mathbf{K} \cdot \mathbf{J}^{-1} \end{cases} \text{ and } \begin{cases} \mathbf{A} = \mathbf{O}^T\mathbf{O} + \frac{1}{\mu}\mathbf{I} \\ \mathbf{B} = \mathbf{O}\mathbf{T} + \frac{1}{\mu}(\mathbf{\Psi} \odot \mathbf{Z_V}) \\ \mathbf{C} = \mathbf{\Psi} + 1 \\ \mathbf{D} = \mathbf{\Psi} \odot \mathbf{Z_U} + \mathbf{V}\mathbf{Z_X} \\ \mathbf{E} = \mathbf{V}^T\mathbf{V} + \mathbf{I} \\ \mathbf{F} = \mathbf{\Phi} \odot \mathbf{Z_W} + \mathbf{V}^T\mathbf{Z_V} \\ \mathbf{G} = \mathbf{\Phi} + 1 \\ \mathbf{H} = \mathbf{W}\mathbf{X} + \mathbf{\Phi} \odot \mathbf{Z_X} \\ \mathbf{J} = \mathbf{X}^T\mathbf{X} + \lambda_W\mathbf{I} \\ \mathbf{K} = \mathbf{Z_W}\mathbf{X}^T \end{cases} . \tag{2.43}$$

Unlike the input weight matrix, the weights connecting two hidden layers can be easily trained by the algorithm proposed for SLFNs. Once $\mathbf{W}$ has been updated, calculate the output $\mathbf{Y}$ of the first hidden layer and consider the second hidden layer as a SLFN with $\mathbf{Y}$ being the input. Now we propose the summarized matrix-based algorithm for DLFNs:

Algorithm MBM2: Given a training set $(\mathbf{X}, \mathbf{T})$, and a ReLU-based DLFN,

Step 1: properly initialize weight matrices $\mathbf{W}$ and $\mathbf{V}$.

Step 2: update the output weight matrix $\mathbf{O}$ according to the optimization problem formulated in Eq.(2.38).

Step 3: learn each variables listed in Eq.(2.43) *sequentially* with presented equations. And compute new $\mathbf{\Phi}$ and $\mathbf{\Psi}$ correspond to $\mathbf{W}^*$.

Step 4: train the hidden weight matrix $\mathbf{V}$ by applying a SLFN input weight matrix learning algorithm, then update $\mathbf{\Psi}$.

Step 5: repeat step 2 – step 4 until the training error converges or the maximum iteration number is reached.

2.4 Extension to Multi-Hidden Layer FNNs

The final target of the thesis is to design an algorithm that is suitable for training general ReLU-based Multi-hidden Layer Feedforward neural Networks (MLFNs). This section covers how this goal is achieved in three steps: how DUS is extended to general ReLU-based MLFNs; proposal of MLFNs with a special structure; what the value explosion is and how to handle it.

2.4.1 Extension of Direct Updating Strategy

From the results presented in Section 2.3, it is obvious that the Direct Updating Strategy can be easily extended to MLFNs.

Given a neural network with L ReLU-based hidden layers, with the input and hidden weight matrices denoted by $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L$, under the vector-based framework, the cost function is defined as

$$C(\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{O}) = \sum_{(\mathbf{x},\mathbf{t})\in(\mathbf{X},\mathbf{T})} \|\mathbf{t} - \mathbf{O}\mathbf{\Omega}_{\mathbf{x}}^{L}\mathbf{W}_L \ldots \mathbf{\Omega}_{\mathbf{x}}^{2}\mathbf{W}_2\mathbf{\Omega}_{\mathbf{x}}^{1}\mathbf{W}_1\mathbf{x}\|_p^p. \tag{2.44}$$

The updating algorithms for each individual $\mathbf{W}_i$ and $\mathbf{O}$ follows the scheme described in Section 2.3.2.

To replace all non-liner functions in the cost, let us define L 0-1 matrices $\mathbf{\Phi}_1, \ldots, \mathbf{\Phi}_L$. And the corresponding matrix-based cost function is written as

$$C(\mathbf{W}_1, \ldots, \mathbf{W}_L, \mathbf{O}) = \|\mathbf{T} - \mathbf{O}(\mathbf{\Phi}_L \odot (\mathbf{W}_L \ldots \mathbf{\Phi}_1 \odot (\mathbf{W}_1\mathbf{X})))\|_F^2 . \tag{2.45}$$

This cost can be minimized by applying ADMM with $2L$ dual variables.

Unfortunately, there is a problem with these two MLFN training algorithms. To be specific, it is difficult to obtain a good initialization of weight matrices. In order to handle this initialization dilemma, the special structure MLFNs will be presented.

2.4.2 ReLU-Based MLFNs with Same Hidden Dimensions

Consider a ReLU-based multi-hidden layer FNN, whose hidden dimensions are the same and denoted as H. The input matrix and matrices connecting hidden layers are $\mathbf{W}_1, \mathbf{W}_2, \ldots, \mathbf{W}_L$, and the output

matrix is represented by $\mathbf{O}$. $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_L$ and $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_L$ denote inputs and outputs of each hidden layer respectively.

Now start with a SLFN, defined in Section 2.1.1, we can get an updated input matrix $\mathbf{W}_1^*$ and output matrix $\mathbf{O}_1^*$ by applying MBM algorithm summarized in Section 2.2.2. Then add the second hidden layer after the first one, and initialize $\mathbf{W}_2$ as a *positive random square* matrix. Then we have following relations,

$$\begin{cases} \mathbf{Z}_1 = \mathbf{W}_1^* \mathbf{X} \\ \mathbf{Y}_1 = g(\mathbf{Z}_1) = g(\mathbf{W}_1^* \mathbf{X}) \\ \hat{\mathbf{T}}_1^* = \mathbf{O}_1^* \cdot g(\mathbf{W}_1^* \mathbf{X}) \end{cases} \quad \text{and} \quad \begin{cases} \mathbf{Z}_2 = \mathbf{W}_2 \mathbf{Y}_1 \\ \mathbf{Y}_2 = g(\mathbf{Z}_2) = g(\mathbf{W}_2 \mathbf{Y}_1) \\ \hat{\mathbf{T}}_2 = \mathbf{O}_2 \cdot g(\mathbf{W}_2 \mathbf{Y}_1) \end{cases}, \tag{2.46}$$

where,$\hat{\mathbf{T}}_1^*$ and $\hat{\mathbf{T}}_2$ denote the predictions from the optimized SLFN and initial DLFN respectively. Refer to Fig. 2.3 for an illustration of the structure and relations specified in Eq.(2.46).

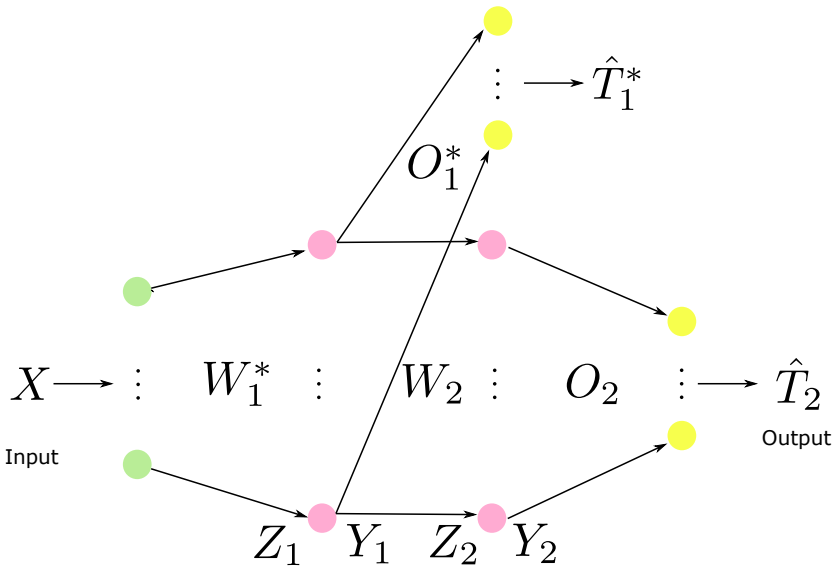


Figure 2.3: An illustration of adding one more hidden layer to a well-trained SLFN.

Since $\mathbf{W}_2$ is a random square matrix, then there is a large possibility that it is invertible. If we initialize the output matrix as $\mathbf{O}_2 = \mathbf{O}_1^* \mathbf{W}_2^{-1}$, then the prediction of the DLFN is $\hat{\mathbf{T}}_2 = \mathbf{O}_2 \mathbf{Y}_2 = \mathbf{O}_1^* \mathbf{W}_2^{-1} \cdot g(\mathbf{W}_2 \mathbf{Y}_1)$. Thanks to ReLUs, the output of each hidden layer is non-negative, specifically $\mathbf{Y}_i \geq \mathbf{0}$. Since $\mathbf{W}_2$ is generated as a positive random ma-

trix, therefore, the term $\mathbf{W}_2\mathbf{Y}_1$ is non-negative. This implies that $\hat{\mathbf{T}}_2 = \mathbf{O}_1^*\mathbf{W}_2^{-1}\mathbf{W}_2\mathbf{Y}_1 = \mathbf{O}_1^*\mathbf{Y}_1 = \hat{\mathbf{T}}_1^*$, consequently, the initial double-hidden layer network will have the same output as the updated single-hidden layer network.

Then construct a new SLFN with $\mathbf{Y}_1$ and $\mathbf{T}$ denoting the input and target matrix respectively, while the input and output matrix are initialized as $\mathbf{W}_2$ and $\mathbf{O}_2$ specified above. With the help of SLFN training algorithm, we can again update the new single-hidden layer network and possibly further decrease the errors. This strategy can be repeated to add more hidden layers.

Initially, this multi-hidden layer strategy starts with a ReLU-based SLFN and updates the weights by the proposed MBM algorithm. Then add another hidden layer with the same dimension and initializes the weights to guarantee that the new network, with one more layer, will have the same performance as the updated previous network. In Section 2.2, it is shown that the proposed updating algorithm will decrease, or at least not increase, the training error. Now define the error of the starting SLFN as e_1, the error of the DLFN as e_2, etc. Then following inequalities

$$e_1 \leq e_2 \leq \cdots \leq e_L \tag{2.47}$$

will hold. Obviously, this multi-layer strategy further decreases, at least not increases, training errors, thus probably improving the performance.

2.4.3 Value Explosion and Normalization

In the previous section, each extra hidden weight matrix is initialized as a random positive square matrix. Therefore, after passing through the newly added layer, the input values can get amplified and finally become infinities. This is called the value explosion problem. To handle this, the updated weight matrices $\mathbf{W}_i^*$ and $\mathbf{O}_i^*$ are normalized, after learning the ith newly added hidden layer, as follows:

$$\tilde{\mathbf{W}}_i^* = \mathbf{W}_i^* \cdot C_{Norm}\,,\ \tilde{\mathbf{O}}_i^* = \frac{\mathbf{O}_i^*}{C_{Norm}} \text{ and } C_{Norm} = \frac{\|\mathbf{Y}_{i-1}\|_F^2}{\|g(\mathbf{W}_i^*\mathbf{Y}_{i-1})\|_F^2}, \tag{2.48}$$

in which $\tilde{\mathbf{W}}_i^*$ and $\tilde{\mathbf{O}}_i^*$ are the normalized weight matrices, and $\mathbf{Y}_{i-1}$ is the output of the previous hidden layer, according to the definitions in Section 2.4.2.

Now, for the ith hidden layer, let us compute the Frobenius norm of its output,

$$\|\mathbf{Y}_i\|_F^2 = \|g(\tilde{\mathbf{W}}_i^* \mathbf{Y}_{i-1})\|_F^2 = \|g(\mathbf{W}_i^* \mathbf{Y}_{i-1})\|_F^2 \cdot \frac{\|\mathbf{Y}_{i-1}\|_F^2}{\|g(\mathbf{W}_i^* \mathbf{Y}_{i-1})\|_F^2} = \|\mathbf{Y}_{i-1}\|_F^2. \tag{2.49}$$

Hopefully, each hidden layer has outputs with an equal Frobenius norm, and thus fixing the value explosion problem. Further more, the final prediction of the output layer will still remain the same,

$$\hat{\mathbf{T}}_i = \tilde{\mathbf{O}}_i^* \cdot g(\tilde{\mathbf{W}}_i^* \mathbf{Y}_{i-1}) = \frac{\mathbf{O}_i^*}{C_{Norm}} \cdot C_{Norm} \cdot g(\mathbf{W}_i^* \mathbf{Y}_{i-1}) = \mathbf{O}_i^* \cdot g(\mathbf{W}_i^* \mathbf{Y}_{i-1}). \tag{2.50}$$

We can see that the normalization only changes the scale of the hidden layer outputs, and the final prediction still remain unchanged.

Chapter 3

Verification Experiments

In Chapter 2, several different algorithms are proposed, but the importance should also be attached to how they perform on real datasets. Therefore, this chapter focuses on experiments carried out in the verification phase of this thesis. The benchmark datasets and experiment setups are described and simulation results are summarized and illustrated in tables and charts.

3.1 Benchmark Datasets and Setups

Due to limitations of computation power and time, the verification simulations focus on 6 multi-class classification datasets with various feature dimensions and sizes. The chosen benchmark datasets are:

(1) a dataset with a low dimension and small size, i.e. *Vowel* [Lic13];

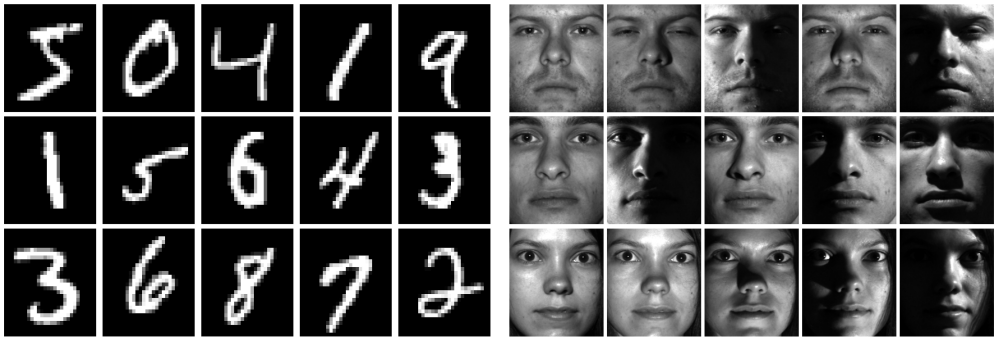
(2) a dataset that is low in dimension but large in size, that is *Letter* [Lic13];

(3) a dataset with a medium feature number and size, i.e. *Satimage* [Lic13];

(4) two datasets with high dimensions but medium sizes, i.e. *AR* and *ExtendedYaleB* [JLD11];

(5) a dataset that is large in both dimension and size, that is *MNIST* dataset [JLD11].

Among them, *Vowel* dataset is aimed to test speech recognition performance. And all other five datasets belong to image recognition applications. Take *MNIST* dataset for example, it contains different hand-written digits and the task is to recognize digits from images of size 28×28. Fig. 3.1(a) shows several sample images in *MNIST* dataset.



(a) (b)

Figure 3.1: Sample images from different image classification datasets. (a) Samples from *MNIST* dataset. (b) Samples from *ExtendedYaleB* dataset [GBK01].

Fig. 3.1(b) contains samples taken from *ExtendedYaleB*, a face recognition dataset. The detailed specifications of datasets are summarized in Table 3.1, where the symbol "#" is the number sign. Also note that the last column "Random Partition" indicates whether the training and testing samples are shuffled at each trail.

In order to measure errors more properly, Normalized Mean Error (NME) is used in verification experiments. By definition, this error is calculated in *dB* scale, specifically, if define a target as $\mathbf{T}$ and prediction as $\hat{\mathbf{T}}$, then the NME is computed as

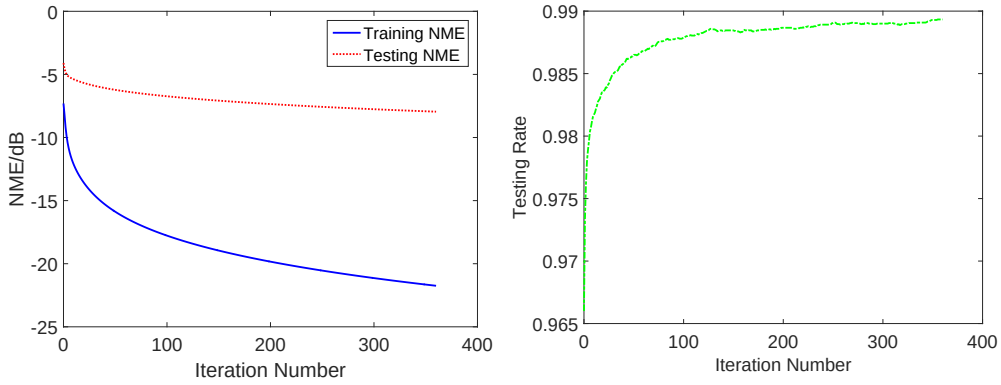
$$\text{NME} = 10 \log_{10} \frac{\mathbb{E}\{\|\mathbf{T} - \hat{\mathbf{T}}\|_F^2\}}{\mathbb{E}\{\|\mathbf{T}\|_F^2\}} , \tag{3.1}$$

where $\mathbb{E}$ is the expectation notation.

Before applying training algorithms on datasets, standard score, also known as z-score, can be used for normalization. By computing

Table 3.1: Specifications of Benchmark Datasets

Dataset Name	Feature #	Label #	Training #	Testing #	Random Partition
Vowel	10	11	528	462	No
Letter	16	26	13333	6667	Yes
Satimage	36	7	4435	2000	No
AR	540	100	1800	800	Yes
ExtendedYaleB	504	38	1600	800	Yes
MNIST	784	10	60000	10000	No



(a) (b)

Figure 3.2: Iterative performance improvements of MBM algorithm, for a SLFN of 1008 hidden nodes, on *ExtendedYaleB* dataset (with z-score). (a) Training and testing NME. (b) Testing rate.

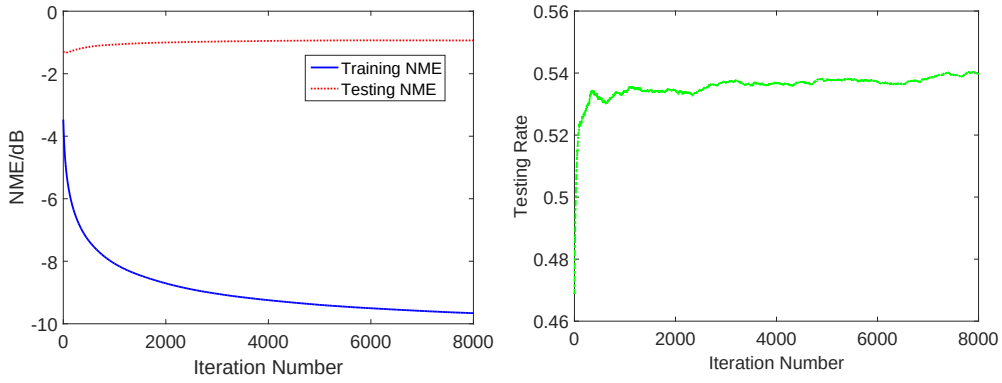
z-scores, data points will be normalized to zero mean and standard deviation 1. Specifically, given a data point x in a dataset of mean $\overline{x}$ and standard deviation σ, its z-score is calculated by [Kre10]

$$z(x) = \frac{x - \overline{x}}{\sigma} \,. \tag{3.2}$$

The simulations are conducted under two conditions: with or without z-score being applied on datasets.

3.2 Simulation Results

In the first stage, we carry out simulations on MBM algorithm for SLFNs and compare the performance with regularized ELM of the same hidden dimension. In Section 2.2.2, it is claimed that the matrix-based method has at least as good performance as regularized ELM, since the method is designed to decrease errors on top of regularized ELM. An example of the behavior, in terms of training error, testing error and testing accuracy, of MBM algorithm on *ExtendedYaleB* dataset is shown in Fig. 3.2. And the simulation is done under the condition that z-score is applied for normalization. Note that the starting point of each curve is the corresponding performance measure of regularized ELM. It is clear that MBM algorithm, with proper regularization



(a) (b)

Figure 3.3: Improvements on testing rate with testing error marginally increasing: an example of a SLFN, with 100 hidden nodes, on *Vowel* dataset (without z-score). (a) Training and testing NME. (b) Testing rate.

factors, decreases both the training and testing error while improves the testing rate steadily. However, the results on *Vowel* dataset show that, for a single-hidden layer network, the testing accuracy increases but the corresponding error also experience a slight increment, which can be seen from Fig. 3.3(a). Fortunately, this phenomenon is only seen on this dataset, but the reasons still remain unknown.

The factor λ_{ELM}, imposing regularization, in ELM is well tuned and the optimal parameter is chosen. In MBM, theoretically there are two regularization factors λ_W and λ_O, one for the input matrix $\mathbf{W}$ and the other for the output matrix $\mathbf{O}$. But in practice, if set $\lambda_O = \lambda_{ELM}$, acceptably good performance will still be obtained. The penalty factor μ in ADMM is chosen from $10^3 - 10^{10}$, and fortunately the performance does not depend heavily on this parameter. So, for most datasets it is set to the same value (5×10^8). The iteration number N_{itr} should vary from one dataset to another, and typically stop iterations when the improvement is marginal. The results, averaged over 30 random trails, for all six datasets are summarized in Table 3.2 and Table 3.4, while the corresponding parameter assignments can be found in Table 3.3 and Table 3.5. Note that Table 3.2 and 3.3 correspond to the simulations results with z-score and the other two tables (3.4 and 3.5) contain the results without z-score. The symbol "—" in the tables means that the corresponding figure is not available. And the column "Hidden #"

Table 3.2: Classification Performance Comparison (with Z-Score)

Dataset Name	Hidden #	Regularized ELM			MBM SLFN			MBM Multi-layer		
		Training NME	Testing NME	Testing Rate(%)	Training NME	Testing NME	Testing Rate(%)	Training NME	Testing NME	Testing Rate(%)
Vowel	100	-3.325	**-0.816**	37.11	-6.278	-0.612	41.04	**-21.106**	-0.125	**41.30**
Vowel	1000	-4.759	**-1.070**	40.62	**-7.590**	-0.791	**42.45**	—	—	—
Letter	160	-2.567	-2.454	76.53	-5.263	-4.878	88.27	**-35.358**	**-10.536**	**94.72**
Satimage	180	-5.559	-4.966	79.84	-7.580	-6.046	83.32	**-8.493**	**-6.158**	**83.57**
AR	270	-1.706	-0.905	85.42	**-12.499**	**-6.149**	**97.49**	—	—	—
AR	1080	-4.763	-1.668	95.90	-14.734	-6.468	98.01	**-26.712**	**-8.115**	**98.18**
ExtendedYaleB	252	-3.145	-1.834	88.18	**-18.832**	**-7.541**	**98.33**	—	—	—
ExtendedYaleB	1008	-7.304	-4.099	96.60	-21.747	-7.952	**98.93**	**-42.879**	**-10.923**	**98.91**
MNIST	392	-5.394	-5.449	91.40	-10.573	-9.895	97.16	**-30.012**	**-14.340**	**97.84**

Table 3.3: Parameter Setups of Different Methods (With Z-Score)

Dataset Name	Hidden #	Regularized ELM	MBM SLFN				MBM Multi-layer				
		λ_{ELM}	λ_O	λ_W	μ	N_{itr}	$\lambda_{O,n}$	$\lambda_{W,n}$	μ_n	$N_{itr,1}$	$N_{itr,n}$
Vowel	100	10	10	0.1	5×10^8	8000	1×10^4	0.1	5×10^8	8000	8000
Vowel	1000	100	100	0.1	5×10^8	8000	—	—	—	—	—
Letter	160	1×10^{-6}	1×10^{-6}	1×10^{-5}	5×10^{10}	12000	1×10^7	1×10^{-5}	5×10^{10}	2000	2000
Satimage	180	50	50	1	5×10^8	8000	1×10^7	1	5×10^8	400	400
AR	270	3000	3000	2.5	5×10^8	800	—	—	—	—	—
AR	1080	3000	3000	1	5×10^8	800	1×10^9	1	5×10^{10}	200	300
ExtendedYaleB	252	4000	4000	1	5×10^8	800	—	—	—	—	—
ExtendedYaleB	1008	1×10^4	1×10^4	0.5	5×10^8	800	1×10^8	0.5	5×10^{10}	400	300
MNIST	392	1×10^{-5}	1×10^{-5}	100	5×10^8	1000	1×10^9	10	5×10^8	200	400

Table 3.4: Classification Performance Comparison (Without Z-Score)

Dataset Name	Hidden #	Regularized ELM			MBM SLFN			MBM Multi-layer		
		Training NME	Testing NME	Testing Rate(%)	Training NME	Testing NME	Testing Rate(%)	Training NME	Testing NME	Testing Rate(%)
Vowel	100	-3.466	-1.285	46.92	-9.661	-0.937	53.95	**-17.468**	**-2.410**	**62.08**
Vowel	1000	-1.717	**-1.177**	52.32	**-2.432**	**-1.176**	**56.31**	—	—	—
Letter	160	-2.364	-2.273	72.12	-5.041	-4.758	85.66	**-32.926**	**-11.885**	**96.08**
Satimage	180	-1.407	-1.355	82.04	-1.737	-1.551	88.28	**-1.953**	**-1.641**	**89.18**
AR	270	-1.711	-0.927	85.11	**-16.677**	**-6.363**	**97.53**	—	—	—
AR	1080	-4.502	-1.826	96.00	**-19.055**	**-6.858**	**97.90**	—	—	—
ExtendedYaleB	252	-3.628	-2.392	87.35	**-14.667**	**-6.713**	**97.60**	—	—	—
ExtendedYaleB	1008	-9.451	-4.752	96.27	**-17.105**	**-7.529**	**97.75**	-11.278	-6.492	**97.73**
MNIST	392	-5.221	-5.286	91.08	-10.923	-9.533	**96.90**	**-27.860**	**-12.843**	**96.98**

Table 3.5: Parameter Setups of Different Methods (Without Z-Score)

Dataset Name	Hidden #	Regularized ELM	MBM SLFN				MBM Multi-layer				
		λ_{ELM}	λ_O	λ_W	μ	N_{itr}	$\lambda_{O,n}$	$\lambda_{W,n}$	μ_n	$N_{itr,1}$	$N_{itr,n}$
Vowel	100	40	40	0.05	5×10^3	8000	1000	2	5×10^3	8000	8000
Vowel	1000	100	100	0.01	5×10^5	2000	—	—	—	—	—
Letter	160	1×10^{-6}	1×10^{-6}	1×10^{-5}	5×10^{10}	12000	1×10^7	1	5×10^{10}	2000	2000
Satimage	180	1×10^{-3}	1×10^7	1×10^{-6}	5×10^8	8000	2.5×10^{11}	100	5×10^5	400	800
AR	270	5×10^7	5×10^7	1×10^4	5×10^8	800	—	—	—	—	—
AR	1080	5×10^7	5×10^7	1×10^4	5×10^8	800	—	—	—	—	—
ExtendedYaleB	252	1×10^7	1×10^7	5×10^3	5×10^8	800	—	—	—	—	—
ExtendedYaleB	1008	1×10^7	1×10^7	5×10^3	5×10^8	800	1×10^{10}	5×10^3	5×10^8	400	400
MNIST	392	1×10^{-5}	1×10^{-5}	1	5×10^8	1000	1×10^{10}	1	5×10^8	200	400

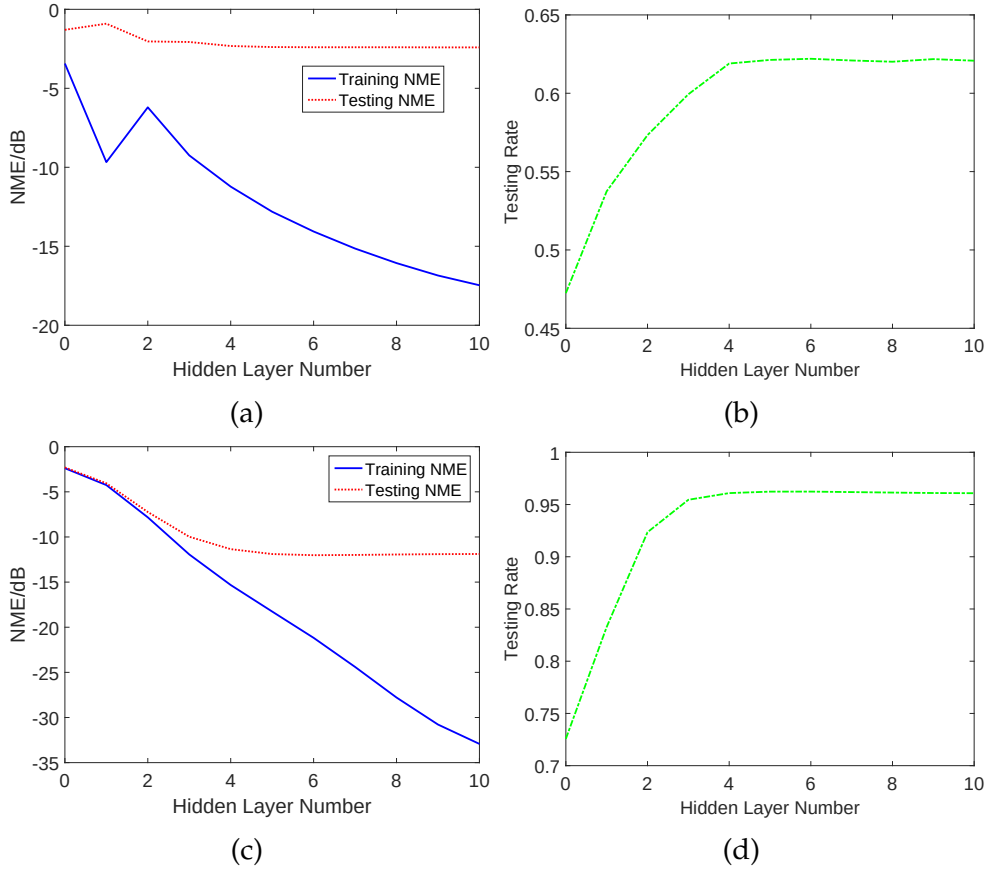


Figure 3.4: Step improvements in performance measures of same hidden dimension MLFNs: examples on *Vowel* and *Letter* (both without z-score). (a) Training and testing NME of *Vowel*. (b) Testing rate of *Vowel*. (c) Training and testing NME of *Letter*. (d) Testing rate of *Letter*.

indicates the hidden dimensions.

Given a SLFN, practically, the input weights are randomly generated from a standard Gaussian distribution and the output matrix is updated by applying regularized ELM algorithm with an optimal regularization factor λ_{ELM}. Afterwards, $\mathbf{W}$ and $\mathbf{O}$ are optimized alternatively, using new factors λ_W and λ_O, for a certain number of iterations.

From the results on *AR*, *ExtendedYaleB*, and *MNIST* dataset, it is possible to claim that even when the hidden dimensions are halves of the feature dimensions, proposed MBM learning algorithm still can achieve reasonably good performance for some datasets. Especially for *AR* and *ExtendedYaleB*, the performance of networks with smaller number of hidden nodes are only slightly inferior to that of higher

dimension networks. This obviously can be regarded as a merit of MBM algorithm.

Verification experiments are also conducted on the special structure MLFNs, described in Section 2.4.2. And the hidden layer number L is fixed to 10 in all simulation trials. In practice, a MLFN is built on top of a well-trained SLFN. But the iteration number should be $N_{itr,1}$ instead of N_{itr}. Each newly-added hidden layer is then trained with new parameters $\lambda_{W,n}$, $\lambda_{O,n}$ and $N_{itr,n}$, while the penalty factor μ_n in ADMM can either remain the same or change to another value. By adding more hidden layers, we expect step improvements in the performance measures, according to Section 2.4.2. Fig. 3.4 shows the improvements on *Vowel* and *Letter* dataset (both without z-score), with hidden dimensions being 100 and 160 respectively. The simulation results are plotted against the hidden layer number, and the starting points (layer number equals to 0) correspond to the values of original regularized ELM. In Fig. 3.4(a), it is noticeable that there is a sharp rise in the training error, when the second hidden layer is added. This is possibly due to the sudden change of regularization factors. Fortunately, after the second hidden layer, the training error drops consistently. This sharp rise is only seen in *Vowel* dataset, and the curves for other datasets are similar to what is shown in Fig.3.4(c) and 3.4(d). However, the performance of MLFNs, compared to SLFNs, remain almost constant on *AR* and *ExtendedYaleB*.

Chapter 4

Conclusions and Discussions

This chapter covers conclusions drawn from the results, future work and discussions on non-technical topics, such as society, sustainability and economics.

4.1 Conclusions

The goal of the thesis project is fulfilled since the algorithms proposed in Chapter 2 have rigorous mathematical motivations, which has been shown. The final version algorithm MBM is developed step by step from the starting point, the row-by-row updating strategy. Each algorithm presented is derived under the strict optimization scheme. And from the performance comparisons presented in Chapter 3, we can conclude that proposed MBM algorithm improves the classification performance of original regularized ELM in a noticeable manner, on most datasets. Though, the results on *Vowel* dataset show a slight rise in testing error, which is not expected, but the testing rate still improves anyway.

Efforts are also given to extending new algorithms to neural networks with more than one hidden layer. Although, the proposed algorithms for DLFNs are not tested in experiments, because of lacks of enough amount of time and computing power, the algorithms are still qualified with enough theoretical justifications. A special structure MLFN is presented and also included in the verification phase. The experiment outputs convince us that MLFNs with such a structure improve the classification abilities of networks in most cases. It is also noticeable that testing accuracies remain almost constant on *AR*

and *ExtendedYaleB* dataset, when adding extra hidden layers. This phenomenon is probably caused by the fact that the testing accuracies for both datasets are already very high, around 98%, using only single-hidden layer networks. Thus, it becomes more difficult to achieve further improvements.

4.2 Future Work and Discussions

Due to limitations of time and computational recurses, the initialization problem in DLFNs is not examined and the related training methods are not tested on standard datasets. Therefore, how to initialize weight matrices properly in DLFNs needs to be investigated and experiments should be conducted on the proposed algorithms in DLFN cases. Since the verification experiments only focus on classification performance of the proposed algorithms, in order to achieve a more full-scale evaluation of the new algorithms, regression datasets need to be involved. Besides, it is noticeable that the hidden dimensions of neural networks implemented in the verification section are relatively low. It is better to test the methods under higher dimension conditions, if enough computing power is available. Finally, how to extend the direct updating strategy to neural networks based on other piecewise linear activation functions is also an interesting research topic.

The original goal of the thesis is to develop new algorithms and apply them on a special clinic dataset to perform disease predictions for new-born babies. The algorithms are expected to have better classification performances, specifically, low false positive and negative rates. And this is aimed to help doctors identify fatal diseases on new-born babies as early as possible in order to save their lives. Though, this aim has not been fulfilled because the access to the clinic data was not available during the research. This project still has its own social contributions, since they can be applied as soon as the data are available and possibly improve the accuracies. If this is the case, more infants will be saved from deaths and more pressure will be released from doctors.

The proposed algorithms are designed for general propose, therefore, they can be applied in many real life applications. For example, the algorithms can be implemented to estimate water quality (refer to [WDL14] for a similar study). This application will help people deal

with the relationship between water pollution and industrial development, which promotes the economical sustainability. Other industrial applications, such as auto-driving system and intelligent robotics, not only improve the security of human beings but also reduce energy consumptions.

Bibliography

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 0-387-31073-8.

[SV08] Alex Smola and S.V.N. Vishwanathan. *Introduction to Machine Learning*. Cambridge University Press, 2008.

[CY11] Colin Campbell and Yiming Ying. *Learning with Support Vector Machines*. Morgan & Claypool Publishers, 2011. ISBN: 1608456161, 9781608456161.

[ZM12] Cha Zhang and Yunqian Ma. *Ensemble Machine Learning: Methods and Applications*. Springer Publishing Company, Incorporated, 2012. ISBN: 1441993258, 9781441993250.

[Roj96] R. Rojas. *Neural Networks: A Systematic Introduction*. New York, NY, USA: Springer, 1996. ISBN: 978-3-540-60505-8,978-3-642-61068-4.

[MWK16] Adam Henry Marblestone, Greg Wayne, and Konrad P Kording. "Towards an integration of deep learning and neuroscience". In: *bioRxiv* (2016). DOI: 10.1101/058545. URL: https://www.biorxiv.org/content/early/2016/06/13/058545.

[Zel+94] Andreas Zell et al. "SNNS (Stuttgart Neural Network Simulator)". In: *Neural Network Simulation Environments*. Ed. by Josef Skrzypek. Boston, MA: Springer US, 1994. ISBN: 978-1-4615-2736-7. DOI: 10.1007/978-1-4615-2736-7_9. URL: https://doi.org/10.1007/978-1-4615-2736-7_9.

[Lin70] Seppo Linnainmaa. "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". MA thesis. University Helsinki, 1970.

[Pat+14] Medha J. Patel et al. "An Introduction to Back Propagation Learning and its Application in Classification of Genome Data Sequence". In: *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012*. New Delhi: Springer India, 2014, pp. 609–615. ISBN: 978-81-322-1602-5. DOI: 10.1007/978-81-322-1602-5_65. URL: https://doi.org/10.1007/978-81-322-1602-5_65.

[Dre90] Stuart E. Dreyfus. "Artificial neural networks, back propagation, and the Kelley-Bryson gradient procedure". In: *Journal of Guidance, Control, and Dynamics* 13.5 (1990), pp. 926–928. DOI: 10.2514/3.25422.

[Li+09] Y. Li et al. "The Improved Training Algorithm of Back Propagation Neural Network with Self-adaptive Learning Rate". In: *International Conference on Computational Intelligence and Natural Computing*. Vol. 1. 2009, pp. 73–76.

[Kri07] D. Kriesel. *A Brief Introduction to Neural Networks*. 2007. URL: availableathttp://www.dkriesel.com.

[MOM12] Grgoire Montavon, Genevive Orr, and Klaus-Robert Mller. *Neural Networks: Tricks of the Trade*. 2nd. Springer, 2012. ISBN: 9783642352881.

[HZS04] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. "Extreme learning machine: a new learning scheme of feedforward neural networks". In: *Proceedings of IEEE International Joint Conference on Neural Networks*. Vol. 2. 2004, pp. 985–990.

[HZS06] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. "Extreme learning machine: Theory and application". In: *Neurocomputing* 70 (2006), pp. 489–501.

[HCS06] G.-B. Huang, L. Chen, and C.-K. Siew. "Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes". In: *IEEE Transactions on Neural Networks* 17.4 (2006), pp. 879–892.

[Hua+12] G.-B. Huang et al. "Extreme Learning Machine for Regression and Multiclass Classification". In: *IEEE Transactions on Systems, Man and Cybernetics-Part B: Cybernetics* 42.2 (2012), pp. 513–529.

[Hua15] G.-B. Huang. "What are Extreme Learning Machines? Filling the Gap Between Frank Rosenblatt's Dream and John von Neumann's Puzzle". In: *Cognitive Computation* 7.3 (2015), pp. 263–278.

[Kas+13] K.L.C. Kasun et al. "Representational Learning with ELMs for Big Data (Trends & Controversies / Extreme Learning Machines)". In: *IEEE Intelligent Systems* 28.6 (2013), pp. 31–34.

[Lin+13] J. Lin et al. "A secure and practical mechanism for outsourcing ELMs in cloud computing.(Trends & Controversies / Extreme Learning Machines)". In: *IEEE Intelligent System* 28.6 (2013), pp. 35–38.

[Alu+13] A. Alusok et al. "ELMVIS: a nonlinear visualization technique using random permutations and ELMs.(Trends & Controversies / Extreme Learning Machines)". In: *IEEE Intelligent System* 28.6 (2013), pp. 41–46.

[HSS03] R.H.R. Hahnloser, H.S. Seung, and J.J. Slotine. "Permitted and Forbidden Sets in Symmetric Threshold-Linear Networks". In: *Neural Computation* 15.3 (2003), pp. 621–638.

[DZC09] W. Deng, Q. Zheng, and L. Chen. *Regularized Extreme Learning Machine*. 2009. DOI: 10.1109/CIDM.2009.4938676.

[Fit+14] William Fitzgibbon et al., eds. *Modeling, Simulation and Optimization for Science and Technology*. Springer, 2014. ISBN: 9401790531, 9789401790536.

[GM76] Daniel Gabay and Bertrand Mercier. "A dual algorithm for the solution of nonlinear variational problems via finite element approximation". In: *Computers & Mathematics with Applications* 2.1 (1976), pp. 17–40.

[Lic13] M. Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

[JLD11] Z. Jiang, Z. Lin, and L. S. Davis. "Learning a discriminative dictionary for sparse coding via label consistent K-SVD". In: *CVPR 2011*. 2011, pp. 1697–1704. DOI: 10.1109/CVPR.2011.5995354.

[GBK01] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. “From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose”. In: *IEEE Trans. Pattern Anal. Mach. Intelligence* 23.6 (2001), pp. 643–660.

[Kre10] Eewin Kresyzig. *Advanced Engineering Mathematics*. 10th. John Wiley & Sons, 2010. ISBN: 0470458364.

[WDL14] Wei Wang, Changhui Deng, and Xiangjun Li. “Soft sensing of dissolved oxygen in fishpond via extreme learning machine”. In: *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*. IEEE. 2014, pp. 3393–3395.

Appendix A

Popular Activation Functions

Here are some non-linear activation functions [Bis06], which are widely used in neural networks, listed in Table A.1.

Table A.1: Popular Activation Functions

Function Name	Mathematical Form
Binary step function	$g(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$
Gaussian	$g(x) = \exp^{-x^2}$
Hyperbolic tangent	$g(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
Inverse tangent	$g(x) = \tan^{-1}(x)$
Leaky ReLU	$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.01 \cdot x & \text{otherwise} \end{cases}$
Logistic	$g(x) = \frac{1}{1+e^{-x}}$
ReLU	$g(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
Sinc function	$g(x) = \begin{cases} 1 & \text{if } x = 0 \\ \frac{\sin(x)}{x} & \text{otherwise} \end{cases}$
Sinusoid	$g(x) = \sin(x)$

Appendix B

Mathematical Derivations

B.1 MBM for SLFNs

From Section 2.2.2, the Lagrangian function L for optimizing the weight matrix $\mathbf{W}$ in a SLFN is

$$L = \frac{1}{2}\|\mathbf{T}-\mathbf{O}\cdot\mathbf{Z_X}\|_F^2+\frac{1}{2\mu}(\|\mathbf{WX}-\mathbf{Z_W}\|_F^2+\|\mathbf{\Phi}\odot\mathbf{Z_W}-\mathbf{Z_X}\|_F^2+\lambda_W\|\mathbf{W}\|_F^2). \tag{B.1}$$

To update $\mathbf{Z_X}$, we take the derivative with respect to $\mathbf{Z_X}$ and obtain

$$\frac{\partial L}{\partial \mathbf{Z_X}} = \mathbf{O}^T(\mathbf{OZ_X}-\mathbf{T}) + \frac{1}{\mu}\cdot(\mathbf{Z_X}-\mathbf{\Phi}\odot\mathbf{Z_W}). \tag{B.2}$$

Now set the derivative to zero, we have

$$(\mathbf{O}^T\mathbf{O}+\frac{1}{\mu}\cdot\mathbf{I})\cdot\mathbf{Z_X} = \mathbf{O}^T\mathbf{T}+\frac{1}{\mu}\cdot\mathbf{\Phi}\odot\mathbf{Z_W}. \tag{B.3}$$

Since the factor $\frac{1}{\mu}$ is positive, therefore the term $\mathbf{O}^T\mathbf{O}+\frac{1}{\mu}\cdot\mathbf{I}$ is invertible. Then $\mathbf{Z_X}$ can be optimized through following equation

$$\mathbf{Z_X^*} = (\mathbf{O}^T\mathbf{O}+\frac{1}{\mu}\cdot\mathbf{I})^{-1}\cdot(\mathbf{O}^T\mathbf{T}+\frac{1}{\mu}\cdot\mathbf{\Phi}\odot\mathbf{Z_W}). \tag{B.4}$$

Let us apply same tricks on $\mathbf{Z_W}$ and $\mathbf{W}$. Firstly, calculate the deriva-

tives of L in terms of $\mathbf{Z_W}$ and $\mathbf{W}$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{Z_W}} &= \frac{1}{\mu}\cdot(\mathbf{\Phi}\odot(\mathbf{\Phi}\odot\mathbf{Z_W}-\mathbf{Z_X})+\mathbf{Z_W}-\mathbf{WX}),\\ \frac{\partial L}{\partial \mathbf{W}} &= \frac{1}{\mu}\cdot((\mathbf{WX}-\mathbf{Z_W})\cdot\mathbf{X}^T+\lambda_{\mathbf{W}}\mathbf{W}).\end{aligned} \tag{B.5}$$

Then, set them to zeros and obtain updating equations as follows

$$\begin{aligned}\mathbf{Z}^*_{\mathbf{W}} &= (\mathbf{\Phi}\odot\mathbf{Z_X}+\mathbf{WX})\oslash(\mathbf{\Phi}+1),\\ \mathbf{W}^* &= \mathbf{Z_W}\mathbf{X}^T\cdot(\mathbf{X}\mathbf{X}^T+\lambda_{\mathbf{W}}\cdot\mathbf{I})^{-1}.\end{aligned} \tag{B.6}$$

B.2 MBM for DLFNs

The method to obtain close form solutions, for optimizing each variables in a DLFN, is as same as used in a SLFN case. Therefore, we only list all derivatives

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{Z_U}} &= \mathbf{O}^T\mathbf{O}\cdot\mathbf{Z_U}+\frac{1}{\mu}\cdot(\mathbf{Z_U}-\mathbf{\Psi}\odot\mathbf{Z_V}),\\ \frac{\partial L}{\partial \mathbf{Z_V}} &= \frac{1}{\mu}\cdot(\mathbf{\Psi}\odot(\mathbf{\Psi}\odot\mathbf{Z_V}-\mathbf{Z_U})+\mathbf{Z_V}-\mathbf{VZ_X}),\\ \frac{\partial L}{\partial \mathbf{Z_X}} &= \frac{1}{\mu}\cdot(\mathbf{Z_X}-\mathbf{\Phi}\odot\mathbf{Z_W}+\mathbf{V}^T(\mathbf{VZ_X}-\mathbf{Z_V})),\\ \frac{\partial L}{\partial \mathbf{Z_W}} &= \frac{1}{\mu}\cdot(\mathbf{\Phi}\odot(\mathbf{\Phi}\odot\mathbf{Z_W}-\mathbf{Z_X})+\mathbf{Z_W}-\mathbf{WX})),\\ \frac{\partial L}{\partial \mathbf{W}} &= \frac{1}{\mu}\cdot((\mathbf{WX}-\mathbf{Z_W})\cdot\mathbf{X}^T+\lambda_{\mathbf{W}}\mathbf{W}),\end{aligned} \tag{B.7}$$

and the corresponding final solutions

$$\begin{aligned}\mathbf{Z}^*_{\mathbf{U}} &= (\mathbf{O}^T\mathbf{O}+\frac{1}{\mu}\cdot\mathbf{I})^{-1}\cdot(\mathbf{O}^T\mathbf{T}+\frac{1}{\mu}\cdot\mathbf{\Psi}\odot\mathbf{Z_V}),\\ \mathbf{Z}^*_{\mathbf{V}} &= (\mathbf{\Psi}\odot\mathbf{Z_U}+\mathbf{VZ_X})\oslash(\mathbf{\Psi}+1),\\ \mathbf{Z}^*_{\mathbf{X}} &= (\mathbf{V}^T\mathbf{V}+\mathbf{I})^{-1}\cdot(\mathbf{V}^T\mathbf{Z_V}+\mathbf{\Phi}\odot\mathbf{Z_W}),\\ \mathbf{Z}^*_{\mathbf{W}} &= (\mathbf{\Phi}\odot\mathbf{Z_X}+\mathbf{WX})\oslash(\mathbf{\Phi}+1),\\ \mathbf{W}^* &= \mathbf{Z_W}\mathbf{X}^T\cdot(\mathbf{X}\mathbf{X}^T+\lambda_{\mathbf{W}}\cdot\mathbf{I})^{-1}.\end{aligned} \tag{B.8}$$