APPROVAL SHEET

Title of Dissertation: Convexification and Deconvexification for Training Artificial Neural Networks

Name of Candidate: Yichuan Gui
Degree of Philosophy, 2016

Dissertation and Abstract Approved: James T. Lo

Dr. James Ting-Ho Lo
Professor
Department of Mathematics
and Statistics

Date Approved: 04/22/2016

ABSTRACT

Title of Dissertation: Convexification and Deconvexification for Training Artificial Neural Networks

Yichuan Gui, Degree of Philosophy, 2016

Dissertation directed by: Dr. James Ting-Ho Lo, Professor
Department of Mathematics and Statistics

Dr. Konstantinos Kalpakis, Associate Professor
Department of Computer Science and
Electrical Engineering

The purpose of this dissertation research is to overcome a fundamental problem in the theory and application of artificial neural networks (ANNs). The problem, called the local minimum problem in training ANNs, has plagued the ANN community since the middle of 1980s.

ANNs trained with backpropagation are extensively utilized to solve various tasks in artificial intelligence fields for decades. The computing power of ANNs is derived through its particularly distributed structure together with the capability to learn and to generalize. However, application and further development of ANNs have been impeded by the local minimum problem and attracted much attention for a very long time.

A primary difficulty of solving the local minimum problem lies in the intrinsic non-convexity of training criteria of ANNs, which usually contain a large number of non-global local minima in their weight spaces. Although an enormous amount of solutions have been developed to optimize the free parameters of the objective function for consistently achieving a better optimum, these methods or algorithms are unable to solve the local minimum problem essentially with the intricate presence of the non-convex function.

To alleviate the fundamental difficulty of the local minimum problem in training ANNs, this dissertation proposes a series of methodologies by applying convexification and deconvexification to avoid non-global local minima and achieve global or near-global minima with satisfactory optimization and generalization performances. These methodologies are developed based on a normalized risk-averting error (NRAE) criterion. The use of this criterion removes the practical difficulty of computational overflow and ill-initialization existed in a risk-averting error criterion, which was the predecessor of the NRAE criterion and has benefits to effectively handle non-global local minima by convexifying the non-convex error space. With employing a proper convexification and deconvexification strategy, it is also uncovered that the NRAE criterion has the advantage in handling high-dimensional non-convex optimization of deep neural networks, which typically suffer from difficulties such like local minima, saddle points, large flat regions, etc. existed in the non-convex error spaces.

In this dissertation, the effectiveness of proposed methods based on the NRAE criterion is first evaluated in training multilayer perceptrons (MLPs) for function approximation tasks, demonstrating the optimization advantage in avoiding or alleviating the local minimum problem compared to the training with the standard mean squared error criterion. Moreover, the NRAE-based training methods that are applied to train convolutional neural networks and deep MLPs for recognizing handwritten digits in the MNIST dataset present better optimization and generalization results than many benchmark performances, which were achieved by integrating different non-convex training criteria and deep learning approaches. At last, to enhance the generalization of the ANN trained by the NRAE-based method, a statistical pruning method that prunes redundant connections of ANN is implemented and experimented for further improving the generalization ability of the ANN trained by the NRAE criterion.

Convexification and Deconvexification for Training Artificial Neural Networks

by

Yichuan Gui

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

To my wife Fei and my parents

ACKNOWLEDGMENTS

I would like to give my thanks to my advisers, Prof. James Lo and Prof. Yun Peng, for their significant support and guidance throughout this long march of pursing a Ph.D. degree. As my research adviser at Department of Mathematics and Statistics, Prof. James Lo provided me a great opportunity to make contributions in a research field of AI, which was I always dreamed about. He had very good insights to research problems and gave me constantly advices and inspirations. I would not even start my Ph.D. research without his encouragement. Prof. Yun Peng is my co-adviser at Department of Computer Science and Electrical Engineering, and he is a very kind professor with profound teaching and researching experiences in the research field of AI. He always thought ahead to problems where I encountered difficulties, and his ability to find solutions to complex problems with using small and simple experimental models was outstanding. The depth and long term perspective in his analyzing and solving approach has always been impressive. After his retirement, he still spent his personal time on reviewing and revising my dissertation, which I appreciated it very much.

I am also grateful to the rest members of my dissertation defense committee, Prof. Tim Oates, Prof. Charles Nicholas, Prof. Tim Finin, and Dr. Kostas Kalpakis for providing their valuable feedbacks and suggestions to my dissertation. Prof. Tim Oates is a pioneer in machine learning research and his machine learning class at UMBC is one of the best classes I ever took. I would like to thank him for being such an amazing teacher and researcher, while I could learn so much from him. I would also like to thank Prof. Charles Nicholas for being a great tutor in teaching many advanced research skills, which I have been benefited during the whole Ph.D. program. Prof. Tim Finin is an excellent professor who I was very lucky to work with for 6 months. His distinctive research viewpoint and

patient guidance helped me grow as a professional researcher and always inspired me in my research work. I would also like to thank Dr. Kostas Kalpakis, who accepted to be my co-adviser after Prof. Peng's retirement and provided so many help for finalizing my Ph.D. program.

Last but not least, I would like to express my deep gratitude to my lovely wife Fei Yu. Her love and support made my dream of being a Ph.D. come true. She is an endless source of happiness in this hardest journey to me, and her companion is the most precious treasure in my life. I would also like to thank my parents for their constantly support and confidence. Without their caring and encouragement, I would never make such a great achievement. Thank you so much for all your selfless dedication. I love you all!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BP	Backpropagation
CE	Cross-entropy
CNN	Convolutional Neural Network
DBM	Deep Boltzmann Machine
DBN	Deep Belief Network
DNN	Deep Neural Network
EGDC	Enhanced Gradual Deconvexification
XOR	Exclusive-OR
GD	Gradient Descent
GDC	Gradual Deconvexification
GPU	Graphics Processing Unit
GSP	Gradual Statistical Pruning
LMS	Least Mean Square
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NRAE	Normalized Risk-averting Error
PCA	Principle Component Analysis
RAE	Risk-averting Error
RBF	Radius Basis Function
RBM	Restricted Boltzmann Machine
SAE	Stacked Autoencoder
SGD	Stochastic Gradient Descent
SNR	Signal-to-noise Ratio
SP	Statistical Pruning
SVM	Support Vector Machine

Chapter 1

INTRODUCTION

1.1 Motivation

Artificial neural networks (ANNs) trained with backpropagation are commonly used to solve various tasks, such as signal processing, function approximation, classification, and pattern recognition, for decades. The computing power of ANN is derived through its highly distributed structure together with the capability to learn and thus to generalize. However, trying to improve and expand the development of ANNs has been limited fundamentally by the well-known local minimum problem, which has attracted much attention since its occurrence in training ANNs for a very long time.

The main difficulty of solving the local minimum problem arises from training criteria of ANNs, which are generally chosen as non-convex functions. These objective functions cause the presence of many distinct non-global local minima in the model parameter (e.g. weight) space, where an optimization method performs local search. If optimal regions of the model parameter space are unable to reach through employing a non-convex training criterion, performances of the ANN are restricted and expectations of the satisfactory performance cannot be consistently guaranteed, even with applying proper optimization methods or a large number of training trials.

As the development of deep neural networks (DNNs), many deep learning methods

illustrate significant advantages compared to its competitors of shallow learning through extracting better high-level features from data. Meanwhile, DNNs are able to represent high-level abstractions in data using both efficient learning algorithms and effective ANNs with deep network architectures, with achieving superior generalization capabilities in solving many challengeable AI-related tasks. Although deep learning employs complicated networks and intricate algorithms with heuristic techniques to minimize non-convex training criteria for impressive performances, the high-dimensional non-convex optimization has not been well treated by deep learning fundamentally. The training of DNNs is still affected by the use of high-dimensional non-convex training criteria, which hinders the proper learning of local representations of information with the distribution of critical points (i.e. maximum, minima, and saddle points). In fact, without transferring the non-convex optimization to a convex problem, many difficulties of training DNNs with the non-convex error criteria, such as local minima, saddle points, and large flat regions existed in the training error space, can hardly be solved fundamentally.

Although tremendous efforts have been made to overcome the local minimum problem in training ANNs, most methods, algorithms and techniques are generally developed to reduce the training error against a non-convex objective function by optimizing its free parameters. A new type of the convexification method based on the risk-averting error (RAE) criterion, which has the robustness to expand the error space of the objective function from non-convex to strictly convex thus avoiding the non-global local minimum, was well proven in theory and numerically experimented in training multilayer perceptrons. However, the RAE criterion suffers from the difficulty of its exponential magnitude and computational overflow in practical implementations, which restrict the development of using the RAE criterion to properly solve the local minimum problem in real applications.

This dissertation research is motivated by the theoretical essence of the RAE criterion and is dedicated to developing new methodologies for solving the difficulty of applying

the RAE criterion in practical perspectives, with alleviating the local minimum problem in training multilayer perceptrons (MLPs) and providing outstanding training performances. Inspired by the convexification purpose, it provides an effective approach to solve the high-dimensional non-convex optimization in training convolutional neural networks (CNNs) and deep MLPs for the satisfactory optimization and generalization performances.

1.2 Thesis Statement

This dissertation proposes a series of methodologies with applying convexification and deconvexification based on the normalized risk-averting error criterion to avoid or alleviate non-global local minima in training MLPs, while achieving the global or near-global minima with superior training errors. Moreover, this dissertation also provide an effective approach based on the convexification and deconvexification methodology to solve the high-dimensional non-convex optimization in training CNNs and deep MLPs, with minimizing training errors and maximizing generalization capabilities.

1.3 Contributions

Main contributions of this dissertation are describes as follows:

- Provided an approach based on the NRAE criterion with applying convexification and deconvexification to alleviate the fundamental difficulty of the local minimum problem in training MLPs.
- Eliminated many difficulties such as the computational overflow and the ill-initialization problem of the training method based on the risk-averting error criterion, which was the predecessor of the NRAE criterion.

- Developed a collection of NRAE-based methodologies to train MLPs in approximating functions with the superior optimization capabilities than those produced with the standard mean squared error (MSE) criterion.

- Developed an approach based on convexification and deconvexification to efficiently train neural networks with deep architectures, such as CNNs and deep MLPs, on recognizing handwritten digits of the MNIST dataset with better optimization and generalization performances than many benchmark results.

- Implemented a statistical neural network pruning method based on the hypothesis testing to improve the generalization of ANNs.

1.4 Dissertation Outline

The organization of this dissertation is described as follows. Chapter 2 introduces the background and related works of this dissertation research. Chapter 3 describes the theoretical foundation of the NRAE criterion and demonstrates conceptual experiments in applying the criterion to train MLPs. Chapter 4 presents the convexification methodology and evaluates two training methods in approximating functions and recognizing handwritten digits. In addition, a stagnant difficulty that limits the practical application of the convexification methodology is completely discussed. Chapter 5 demonstrates and evaluates the deconvexification methodology to resolve the stagnant problem in training MLPs for function approximation and data classification. Chapter 6 focuses on developing and evaluating an enhanced NRAE-based training method to train DNNs for achieving satisfactory performances on both optimization and generalization. Chapter 7 introduces a statistical neural network pruning approach to further improve the generalization of the NRAE-based methods in training ANNs. Chapter 8 summarizes the dissertation and outlines future works.

Chapter 2

BACKGROUND AND RELATED WORK

2.1 Local Minimum Problem in Training Artificial Neural Networks

A local minimum of a function is mathematically defined as the least value that the function takes at a point within a given neighborhood and it need not be (but may be) a global minimum of the function (e.g. Fig. 2.1). Although an optimization applied to a function commonly demands a global minimum, the greedy local search method often leads to a local minimum but not necessarily a global minimum in the solution space, which is referred as a local minimum problem.

The local minimum problem has impeded the development and the application of artificial neural networks (ANNs) trained with backpropagation (BP) [91] and has attracted much attention since its identification for decades [1, 7, 26, 37, 38, 80, 86, 87, 117]. It affects the performance of BP with the presence of local minima in addition to global minima for training ANNs. In general, since it is difficult to determine the numbers of local and global minima and BP is basically a hill-climbing technique [38], the training of ANNs has the risk of being trapped in a local minimum where every small change in trainable weights affects the objective function. It is obviously undesirable to complete the training process at a non-global local minimum, which is the local minimum with a non-zero and large value comparing to the global minimum with a zero value.

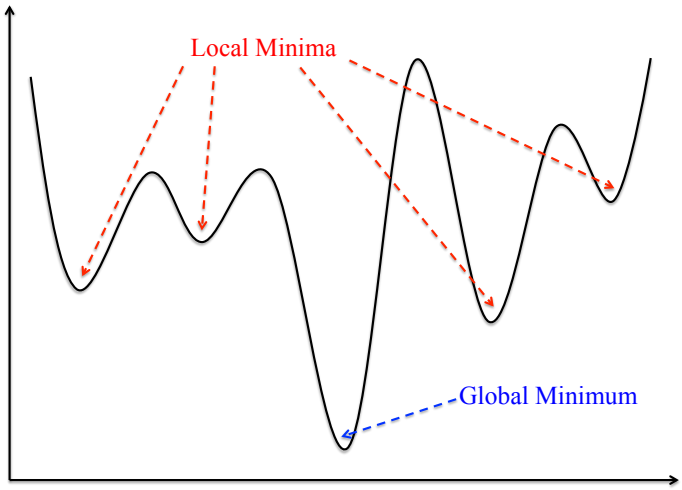


FIG. 2.1. The error space plot of a two-dimensional function with a global minimum and certain local minima.

To efficiently and effectively solve the local minimum problem or reduce the chance of getting trapped at the local minimum in training ANNs, many strategies have been explored in the literature and three classical categories of solutions are discussed in the following sections.

2.1.1 Heuristic Solution

As a heuristic solution to solve the local minimum problem, a simply and widely used technique is to train the ANN more than once with choosing initial weights from different regions of the weight space, and then find the best solution from those training sessions by trial and error [50]. However, a great number of trials need to be performed in this case, while the chance of finding the global minimum cannot be guaranteed.

In addition, more effective heuristics in training ANNs have been explored to avoid the local minimum problem and have been summarized in [64], such as shuffling training

examples in the stochastic learning, normalizing the inputs by applying the mean cancellation and the covariance equalization, selecting target outputs in the range of the activation function, using the symmetric activation function, initializing the weights for each training node based on the certain distribution with the mean zero and the standard deviation related to the number of connections feeding into the node, adaptively choosing learning rates with momentum. Most of these practical tricks can greatly improve the chances of escaping the local minimum and decreasing the convergence time in training ANNs. However, none of these heuristics guarantees that the training of ANNs consistently achieves a good solution that is located at the global or near-global minimum.

The heuristic strategy was an important type of solutions to avoid the local minimum problem back to the early research of ANNs, and it is still an effective approach to be applied for achieving good performances in the modern research of ANNs. By performing a great number of trials with using different heuristics to train ANNs, the success rate of finding an optimal result in statistical could significantly increases. However, heuristic solutions do not focus on handling the problem brought by the non-convex objective function in training ANNs, thus the local minimum problem is hardly to be solved fundamentally by the heuristic solutions.

2.1.2 Optimization Solution

Generally, gradient-based optimization methods, such as conjugate gradient [39], Broyden-Fletcher-Goldfarb-Shanno (BFGS) [13, 30, 33, 102], and Levenberg Marquardt [83], are commonly considered having advantages in avoiding local minima compared to gradient descent (GD) in training ANNs with BP. However, it has been pointed out that training even a 3-node network is NP-complete when all hidden nodes compute the discrete linear threshold function [10], and training a 3-node sigmoid network with zero threshold on the output node is NP-hard [104]. As a consequence, it implied that BP is generally

not an efficient algorithm unless P = NP at least. Therefore, existing algorithms, including both the first and second order optimization methods, cannot guarantee to achieve the optimal solution in polynomial time for training ANNs with the use of BP. More important, many gradient-based optimization methods are mainly designed for batch training, which is an inefficient training mode especially when the training dataset is large [115]. Most of these methods are only able to work on small architectures of ANNs [65].

Many non-gradient based optimization methods, such as genetic algorithm [80, 81], simulated annealing (SA) [15, 56], and ant colony optimization [21, 25], are focus on achieving good approximations to the global optimum of a given function in a large search space without being trapped into local minima. In these methods, SA has been generally applied to effectively find the optimal solution in training ANNs. The SA method simulates an annealing procedure in metallurgy through heating a metal above its melting point and then cooling slowly until it solidifies into a perfect crystalline structure, corresponding to a procedure of training the ANN for finding a global minimum of the objective function. The SA algorithm uses a probability P_α to denote a system state α of the energy function E_α (i.e. objective function) at a certain computational temperature t. Then, it leads the system to reach different states α guided by different probabilities P_α until the system reaches a state α^* with the lowest energy E_{α^*}, which is considered as the global minimum of E_α. When performing the SA, the global minimum is theoretically guaranteed to be reached with a high probability. However, the cooling schedule, which is an approach of decreasing the computational temperature t, is hard to control for achieving an efficient training. If t is reduced too fast, the SA approach may prematurely converge on a local minimum; if t is reduced too slow, the convergence time of the SA algorithm is dreadfully long. Although many improvement methods have been proposed to accelerate the SA method, such as Cauchy annealing [109], simulated re-annealing [49], generalized SA [111], and the SA with known global value [79], the general convergence speed of SA to achieve the global

minimum is still significantly slow in practical applications, even with performing reliable cooling schedules [26].

2.1.3 Convexification Solution

Although significant efforts of the heuristic and optimization solutions have been made to alleviate or avoid the local minimum in training ANNs, those methods or techniques are generally derived for reducing the training error against a non-convex objective by optimizing its free parameters (i.e. trainable weights in ANNs). If the non-convex objective function is not transformed to a convex function in a proper way, any region of the weight space that is unable to be visited by the training method could possibly contain the global or near-global minimum. More important, if the objective function is non-convex and the region of the weight space in the domain of this function is not visited by the training method, the optimization result obtained by such a training method cannot be claimed as a global or near-global minimum, because the objective function may have its global or near-global minimum only inside the region which is not visited by the training method. This is a fundamental difficulty for solving local minimum problem in training ANNs.

To overcome this fundamental difficulty, a convexification approach, which is applied to convexify the error space of the non-convex function for the global optimization, has been studied as a solution of the local minimum problem for decades. Two well-known convexification methods are the graduated nonconvexity method [8] and the Liu-Floudas convexification method [71, 116]. However, these methods are difficult to be applied for training ANNs with the capability to handle a large number of data and parameters in reasonable computation time. For example, the Liu-Floudas convexification method is able to convexify a twice continuously differentiable non-convex function with adding a quadratic term, but determining the weight α of the added quadratic term for the convexification involves massive computation when the number of parameters to be optimized is large.

A new type of the convexification method based on a risk-averting error (RAE) criterion has been proposed in [78] as

$$J_\lambda(w) := \sum_{k=1}^{K} e^{\left(\lambda\left\|y_k-\hat{f}(x_k,w)\right\|^2\right)} \tag{2.1}$$

Here, (x_k, y_k) is a set of input/output samples used in training the ANN for $k = 1, ..., K$, $\hat{f}(x_k, w)$ is the output of the ANN with a weight vector $w \in R^N$ according to input x_k, and λ is called the risk-sensitivity index. The use of the RAE criterion was motivated by emphasizing the large individual deviations $\left\|y_k - \hat{f}(x_k, w)\right\|^2$ of the mean squared error (MSE) criterion

$$Q(w) := \frac{1}{K}\sum_{k=1}^{K}\left\|y_k - \hat{f}(x_k, w)\right\|^2 \tag{2.2}$$

in an exponential manner with the adaptation of λ, thereby avoiding such large individual deviations and achieving robust performances.

A convexification theorem has been proven in [78], and it is restated as the following:

Theorem 1 (RAE Convexity)**.** *Given the risk-averting error criterion $J_\lambda(w)$, which is twice continuously differentiable, the Jacobian and Hessian matrix of $J_\lambda(w)$ can be defined as $D_K(w) := \left[\frac{\partial \hat{f}(x_k,w)}{\partial w_{ki}}\right]_{K\times N}$ and $H_J(w) := \left[\frac{\partial^2 J_\lambda(w)}{\partial w_i \partial w_j}\right]_{N\times N}$ respectively. A sequence of sets $P := \left\{w \in R^N \mid H_J(w) > 0\right\}$ is monotone increasing as λ increases, and a set $I := \left\{w \in R^N \mid rank(D_K(w)) < min(K, N)\right\}$ is the intersection of solution sets of $L(K, N)$ algebraic equations defined by setting the $L(K, N)$-th submatrices of $D_K(w)$ equal to zero. As λ increases to ∞, the convexity region of $J_\lambda(w)$ in the sequence of sets P expands monotonically to the entire weight or parameter space except a subregion in the complement set $(\cup_{\lambda>0}P)^c$, which is contained in the intersection I.*

Remark. As the number K of training samples increases, the number $L(K, N)$ of solution

sets increases rapidly, and the intersection I of these solution sets shrinks monotonically.

Theorem 1 demonstrates that the convexity region of $J_\lambda(w)$ is able to be expanded as λ increases, and the corresponding error space of $J_\lambda(w)$ can be well stretched to strictly convex with enough training samples when $\lambda \to \infty$. In another word, this theorem guarantees that choosing a larger λ and having more training samples can provide a wider convexity region of $J_\lambda(w)$ in respect to the entire error space. Furthermore, because $J_\lambda(w)$ is a monotonically increasing function of $Q(w)$, both criteria share the same local and global optima. As increasing of λ from zero to infinity, the convexity region of $J_\lambda(w)$ is able to be gradually expanded and its corresponding error space can be strictly convexified when λ approaches infinity, thereby raising a chance to avoid the local minima and thus achieve a global minimum. Therefore, the local minimum problem in training ANNs with the MSE criterion can be alleviated by the convexification of the error space of MSE on the RAE criterion. Experimental results in [72, 73] have demonstrated that the multilayer perceptrons (MLPs) trained with the RAE criterion consistently outperform the MLPs trained with the MSE criterion in approximating functions that have local minima in their error spaces, indicating that the RAE criterion provides a good approach to significantly avoid many local minima in training MLPs.

Nevertheless, choosing a proper λ to perform the training with the RAE criterion has been troubled in practice. It is known that the expansion magnitude of the convexity region of $J_\lambda(w)$ is depended on the value of λ. Although a larger initial λ can lead the error space of $J_\lambda(w)$ to be more convexified in achieving an optimal solution without non-global local minima, how large λ should be applied to initialize the training actually varies in different tasks. An adaptive method based on the RAE criterion in training MLPs proposed a solution to choose initial values of λ in different function approximation tasks [72]. However, it has limitations of choosing λ only at small values where $\lambda \leq 100$, because the computation of the exponential with any $\lambda > 100$ in Eq. (2.1) could easily incur a register overflow in

computer. A following work in [78] proposed a centering and bounding method to solve the computational issue. That method successfully extended the selecting range of λ into a scale larger than 100, but still smaller than some significantly large values, such as $\lambda = 10^{10}$. It is worth pointing out that, to handle complex training tasks, the convexity region of $J_\lambda(w)$ needs to be extremely expanded for achieving the strictly convex under a very large λ like 10^{10} without any register overflow in computer. Therefore, a principal work of this dissertation focuses on developing new methodologies derived from the theoretical essence of the RAE criterion with overcoming numerical difficulties in practice to alleviate the local minimum problem in training ANNs.

2.2 Learning with Deep Architectures

The influence of training deep architectures has become increasingly extensive and significant in machine learning since it appeared in 2006. In general terms, deep architectures are defined by the composition of multiple layers of nonlinear processing units where each unit contains trainable parameters. Machine learning with deep architectures, which is often referred to as deep learning, is based on the supervised or unsupervised learning of multiple levels of features or representations of the data in each layer, while higher level features are derived from lower level features to form a hierarchical representation. Particularly, features produced by lower layers represent the low-level informations that are combined to form high-level features in next layer, and this layer-wise approach repeats to produce higher level features until the final result from the last layer is generalized.

Deep learning is often contrasted with shallow learning by the number of parameterized transformations that a signal encounters as it propagates from the input layer to the output layer, where a parameterized transformation is a processing unit that has trainable parameters [100]. ANNs with multiple hidden layers of units between the input and output

layers are often called deep neural networks (DNNs). A credit assignment path (CAP), which is defined as a chain of transformations from input to output, has been applied to describe potentially causal connections between input and output and may vary in length [100], thereby to distinguish between deep learning and shallow learning. For example, a DNN has the depth of CAPs defined as the number of hidden layers plus one (the parameterized output layer), but a training model with the shallow architecture only consists of one layer of parameterized nonlinear units thus has CAP $= 1$. Although it is not generally agreed upon threshold of the depth of CAPs dividing shallow learning from deep learning, most researchers agree that deep learning has multiple nonlinear layers with CAP > 2 and consider CAP > 10 to be very deep learning [100].

Deep learning methods focus on learning feature hierarchies from various kinds of deep architectures, primary including DNNs [2, 3, 20, 60, 61, 88, 97, 112, 113] and other graphical models with many levels of hidden variables [40, 43]. Theoretical results in [2, 5] suggest that the deep architecture has a wide representation of functions in a more compact form than the shallow architecture, thus can learn the kind of complicated functions that represents high-level abstractions. Meanwhile, it implies that if a compact function represented by the deep architecture needs to be represented by the shallow architecture, a great number of representable components is required. This property alleviates the restriction on the representation capability of functions in learning machines with shallow architectures. Numerous experimental evidences illustrate significant advantages of deep learning compared to its competitors of shallow learning, while deep learning methods easily match or beat shallow learning methods and become the state-of-the-art in solving many AI-related tasks, such as computer vision [2, 64, 68, 82, 85, 88], natural language processing [20, 114], robotics [35], and information retrieval [96, 99].

Although successful demonstrations of the potential of deep learning methods were achieved in spite of the serious challenge of training models with many layers of adaptive

parameters, fundamental difficulties brought by the highly non-convex objective function with the potential of many distinct local minima in the model parameter space still exists in training DNNs. However, there were some arguments corresponding to the local minimum problem in training DNNs. It is pointed out in [27] that not all local minima provide equivalent generalization capabilities, and that for deep architectures, the standard training methods based on random initialization tend to guide the parameters into regions of the parameters space with the poor generalization. In this case, poor non-global local minima can bring worse performances on both the training and test dataset. On the other hand, a recent work in [22] argued that more profound difficulty of training DNNs is originated from the rapid diffusion of saddle points but not local minima, though such saddle points are located between high plateaus in the training error space and illusory express as the presence of local minima. Moreover, another recent work in [17] illustrated that most local minima are equivalent in providing similar performances on a test dataset, and the probability of finding a poor non-global local minimum is non-zero for small networks and reduces exponentially with the network size. This work also implied that the global minimum on the training dataset may not be worth exploring in practice because of overfitting. In these recent arguments, poor non-global local minima seems not to be a serious problem in training DNNs [61].

Nevertheless, we stress that, although many deep learning methods make great efforts to achieve the best performance by using a wide array of training techniques together with practical heuristics, training DNNs is essentially a high-dimensional non-convex optimization problem. Without an effective optimization solution, many difficulties of training non-convex error criteria, including local minima, saddle points, and large flat regions existed in the training error space, can hardly be solved fundamentally. On the contrary, with the proper convexification and deconvexification methodology, the non-convex optimization problem can be transformed to a convex optimization problem, while the difficulties

with the non-convex error criteria can be alleviated or solved by its convexification.

A significant intention of this dissertation is to provide a novel perspective based on the convexification solution of the non-convex optimization to resolve the difficulty of training DNNs associated with high-dimensional non-convex error criteria. In the following sections, we briefly review the background of some well-known deep learning models, such as convolutional neural networks (CNNs), stacked autoencoders (SAEs), deep belief networks (DBNs), and deep Boltzmann machines (DBMs).

2.2.1 Convolutional Neural Networks

From Hubel and Wiesel's classic work on the cat's primary visual cortex [47, 48], it was found that the visual cortex in living organisms contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a receptive field, while the sub-regions are spread to cover the entire visual field. Moreover, these cells act as local filters over the input space and are proper to extract the strong spatially local correlation presented in natural images. Based on that fact, two basic cell types have been identified: simple cells respond maximally to specific edge-like patterns within their receptive field, and complex cells have larger receptive fields and are locally invariant to the exact position of the pattern. Inspired by this significant breakthrough in biology, the first such model to be simulated on a computer was proposed in the early 1980's and called Neocognitron [31, 32], which employed a layer-wise, unsupervised competitive learning algorithm as the filter banks and a separately-trained supervised linear classifier as the output layer.

An outstanding innovation and improvement of Fukushima's work was the development of CNN [62–64] in the 1990's. CNN simplifies the architecture of Neocognitron and applies BP to train the entire network in a supervised manner. Particularly, CNNs are hierarchical neural networks that adopt convolutional and pooling layers to simulate functions

of the simple and complex cells in the multi-stage Hubel-Wiesel architecture, which extract local features at a high resolution and combine them into more complex features at lower resolutions. The loss of spatial information during the resolution conversion is compensated by an increasing number of feature maps in the higher layers. Therefore, a typical CNN is composed of several convolutional and pooling layers followed by a classification module, and this network design was widely considered as a successful DNN architecture in handling many computer vision tasks [64].

The success of CNN is benefited by the combination and alternation of its convolution and pooling layers, which capture the spatial topology and have rich feature representation of the input information with small number of trainable weights. It reduces the landscape complexity of the training criterion in weight space, and leads the optimization method to a good solution much easier. As the development of the deep learning methods, many variants of CNN were explored in the image classification literature. One important discover in [105] described a simplified CNN, which applied a data augmentation with elastic distortions and a combination of convolutional and pooling layers. This simplified CNN did not require complex training heuristics, such as momentum, weight decay, structure-dependent learning rates, and extra padding around the inputs and pooling layers, to finely tune the overall architecture. The result was a very simple yet general architecture, which can yield state-of-the-art performance for visual document analysis. However, without efficient computing resources and methods, if the number of layers in CNNs becomes larger and larger, it would be extremely expensive to train those intricate networks and make them perform well on complex image classification problems. With the rise of high-efficiency computing on graphics processing units (GPUs), training large CNNs has become possible. Several literatures in [3, 43, 52, 88] explored many practical approaches to effectively train DNNs. Soon after that, a series of GPU-based techniques were implemented and refined to enhance the training of large CNNs in handling complex image classification tasks

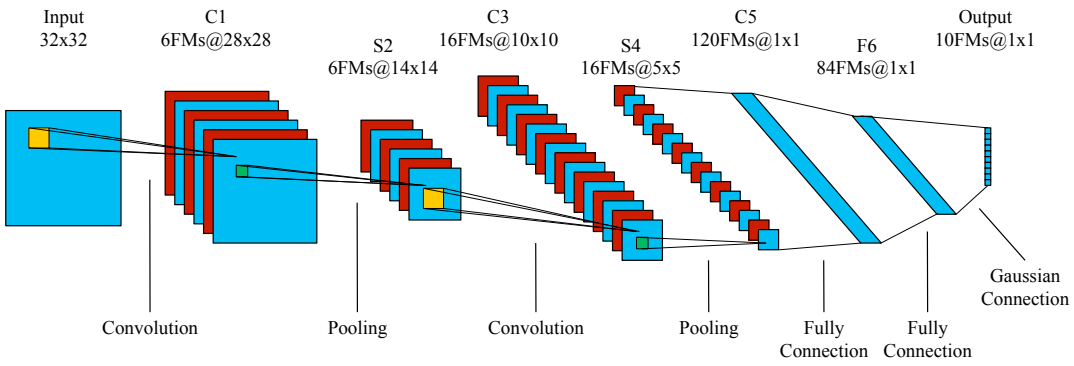


FIG. 2.2. Architecture of LeNet-5 for handwritten digits recognition on the MNIST dataset. LeNet-5 is composed of 8 layers, including 1 input layer as the data input, 3 convolutional layers (C1, C3, and C5), 2 pooling layers (S2 and S4), 1 fully connected layer (F6), and 1 output layer as the classifier. The dimension of the data input for LeNet-5 is 32×32, which is extended by adding zero background pixels to the original 28×28 MNIST dataset. Each layer after the input comprises several feature maps that are represented as "number of feature maps@size of each feature map" in the figure. For example, "6FMs@28x28" in the C1 layer means that this layer contains 6 feature maps and the size of each feature map is 28×28. In each convolutional and pooling layer, the size of the convolutional filter banks is fixed as 5×5 and the pooling region is always 2×2, respectively.

within days instead of months [18, 19]. Those techniques significantly improved upon the best performance in the literature for multiple image datasets, including the MNIST dataset [64], the CIFAR10 dataset [57], the NORB dataset [67], and the notable ImageNet challenge [24, 93].

In this dissertation research, a significant experiment of training the DNN focuses on a well-known CNN architecture called LeNet-5 [64] that is shown in Fig. 2.2, while we introduce some basic concepts of LeNet-5 as follows. LeNet-5 is composed of several layers where the input and output of each layer are sets of arrays called feature maps. The input feature map of each layer is an array that could have different dimensions based on different input types, e.g., each feature map could be a one-dimensional array for an

audio input, or a two-dimensional array for an image input, or a three-dimensional array for a video input. The output feature map of each layer contains several particular features extracted at all locations from the input. For a classification problem, layers of convolution and pooling are considered as a trainable feature extractor, and a trainable classifier in the form of fully connected layers will be added to the feature extractor to build an overall classifier.

Convolutional Layer A convolutional layer is employed to extract local features from feature maps of the previous layer and to construct new feature maps by performing convolution operations with filter banks, then putting through the activation function. The output feature map of each layer keeps several numbers of filter banks. The filter bank contains trainable weights and is generally represented by a 2-dimensional array. The number of filter banks for each feature map is chosen the same as the number of feature maps of the previous layer, thus one output feature map of each layer extracts one pattern of features from all input feature maps of each layer based on a set of particular filter banks. For each convolutional layer, if there are I_m input feature maps, then there will be J_m output feature maps and $I_m \times J_m$ filter banks. Generally, $J_m \geq I_m$, where $m = 1, ..., M$ and M is the number of convolutional layers. The computation of the convolutional layer is defined as

$$c_j^l = \varphi \left(\sum_{i=1}^{F_j} k_{ij}^l * c_i^{l-1} + b_j^l \right) \tag{2.3}$$

where $i = 1, ..., I_m$ and $j = 1, ..., J_m$. Here, l denotes the current layer with the input layer designated to be the first layer and the output layer designated to be the L-th layer in the network, and F_j represents a selection of input feature maps for the computation of the j-th output feature map. Eq. (2.3) performs a 2-dimensional discrete convolution operation (denoted by $*$ in the equation) between the trainable weights k_{ij}^l and the input feature map

c_i^{l-1}, then adds a trainable bias b_j^l to the result and passes it through an activation function $\varphi(\cdot)$ to get the output feature map c_j^l.

It is worth noticing that, if c_j^l and c_{j+1}^l are computed by using the same c_i^{l-1}, then the filter banks associated with the computation of c_j^l and c_{j+1}^l should be different, where the filter banks are k_{ij}^l and $k_{i(j+1)}^l$ respectively. It shows that each filter bank detects a particular feature at every locations on the input feature map, thus all trainable weights in the filter bank are shared to detect the same feature at all locations on the input feature map. This is an important property of the convolutional layer to reduce the number of free parameters, thereby reducing the capacity of the network together with the gap between the training error and the test error. Moreover, one output feature map is only associated with a certain number of particular filter banks, so different output feature maps are able to extract different types of local features from all input feature maps through different sets of filter banks. This is another significant feature of the convolutional layer, indicating that if the input feature map has any shifts or distortions, the output feature map will be changed by the same amount, otherwise it will be maintained unchanged.

Pooling Layer A pooling layer is generally applied after the convolutional layer to reduce the spatial resolution of the new generated feature maps from the previous layer by producing down-sampled version of input feature maps to the current layer. Each output feature map in the pooling layer does not associate with any filter bank, but has a trainable weight that treats each input feature map correspondingly. For each pooling layer, if there are J_n input feature maps, then there will be exactly J_n output feature maps and J_n trainable weights, where $n = 1, ..., N$ and N is the number of pooling layers. The computation of the pooling layer is defined as

$$p_j^l = \varphi\left(w_j^l \cdot d\left(p_j^{l-1}\right) + b_j^l\right) \tag{2.4}$$

where $j = 1, ..., J_n$ and $d(\cdot)$ represents a down-sampling function.

The pooling layer performs the down-sampling operation to the input feature map p_j^{l-1}, multiplies the result by a trainable weight w_j^l, then adds a trainable bias b_j^l to the result and passes it through an activation function $\varphi(\cdot)$ to get the output feature map p_j^l. The down-sampling operation considers a certain neighborhood (e.g., a 2×2 neighborhood) in each input feature map and calculate the average or the maximum value of the neighborhood (i.e., perform the average pooling or the max pooling to the neighborhood), then it strides a step larger than 1 but smaller than or equal to the neighborhood over the entire feature map. Therefore, it assures that each pooling layer produces a reduced-resolution output feature map, which is robust to small variations in the location of features of the input feature map. More recent CNNs do not apply any activation function after the max pooling, and they do not add a trainable weight and bias in the max pooling. Thus, unlike the average pooling, the max pooling function is commonly defined as

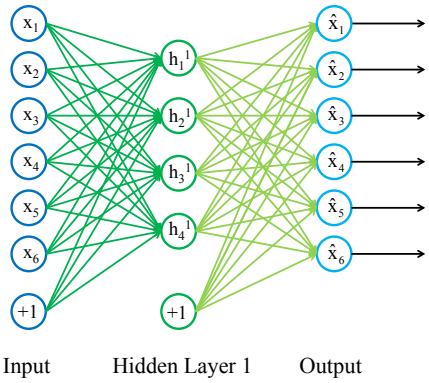
$$p_j^l = d_{\max}\left(p_j^{l-1}\right) . \tag{2.5}$$

Backpropagation on LeNet-5 All layers of LeNet-5 are trained with BP. For error propagation and weight adaptation in convolutional and fully-connected layers, the implementation of BP follows the standard procedure. However, errors after pooling layers have to be propagated through the certain pooling layer by calculating the error with respect to each unit incoming to that pooling layer. The particular implementation of the error propagation in the pooling layer depends on different choices of the pooling function. For example, if the average pooling is applied, the BP could uniformly distribute the error through a pooling unit to all of its input units in the previous layer. If the max pooling is selected, the BP would propagate the error to the input unit that is chosen as the maximum of all in the previous layer.

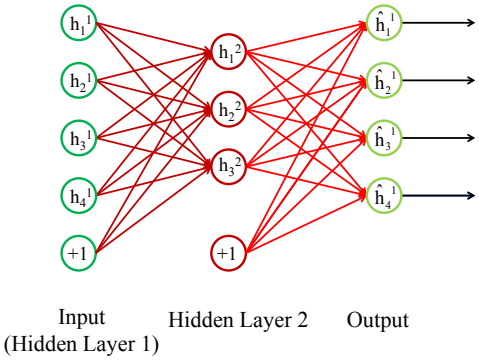
2.2.2 Stacked Autoencoders

An autoencoder, also called autoassociator, or Diabolo network [12, 42, 51, 91, 101], is a feedforward neural network that is very similar to the multilayer perceptron with an input layer, an output layer and one or more hidden layers connecting them. Unlike the multilayer perceptron, the idea of the autoencoder is to encode the input x into some representation $r(x)$ so that the input can be reconstructed from that representation, thus the autoencoder can be trained in an unsupervised manner to reconstruct its own inputs as outputs instead of being trained to predict some target values y given inputs x. The target output of the autoencoder is the autoencoder input itself, and the output layer of the autoencoder always has the same number of nodes as the input layer.

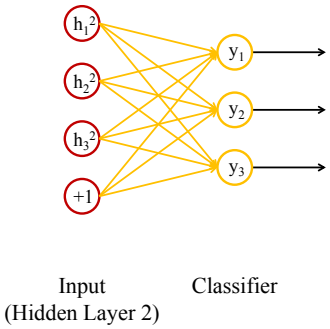
Autoencoders can be stacked to build a network with deep architectures, where each level of the network is associated with an autoencoder that can be trained separately. As an example, a typical stacked autoencoder for classification is trained as shown in Fig. 2.3. Experimental results in [3] illustrate that stacked nonlinear autoencoders with more hidden units than inputs trained by stochastic gradient descent are able to yield useful encoding representations, where the representations taken by a network as the input result in a low classification error. Furthermore, in order to force the hidden layer to discover more robust features and prevent it from simply learning the identity, the autoencoder can be trained to reconstruct the input from a corrupted version of it. In stacked denoising autoencoders [112, 113], the partially corrupted output is cleaned (i.e., denoised) with a specific approach to good representations, where a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input. Moreover, the denoising autoencoder is a stochastic version of the autoencoder, where the stochastic corruption process randomly sets some of the inputs (as many as half of them) to zero [112, 113]. Hence the denoising autoencoder is trying to predict the corrupted values



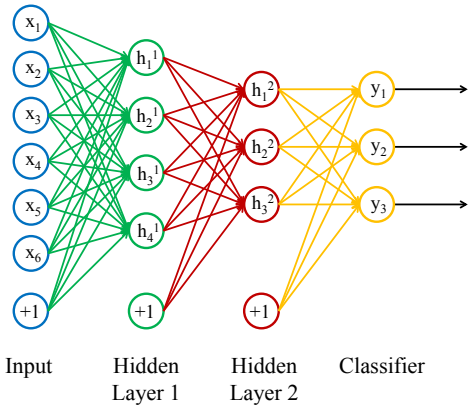
(a) Sparse Autoencoder I



(b) Sparse Autoencoder II



(c) Classifier



(d) Stacked Autoencoder

FIG. 2.3. A stacked autoencoder with the 6-4-3-3 architecture for classification. Fig. 2.3(a): the autoencoder is trained as a sparse autoencoder on the raw input x to learn the primary features h^1. Fig. 2.3(b): the autoencoder is trained as another sparse autoencoder on the primary features h^1 to learn the second features h^2. Fig. 2.3(c): the classifier is trained to map the second features h^2 to the classification labels y. Fig. 2.3(d): the stacked autoencoder is formed with two hidden layers and a final classifier to be fine-tuned as a supervised classification model.

from the uncorrupted values, for randomly selected subsets of missing patterns. Once stacked denoising autoencoders are trained, the output is used as the input to a learning model for the supervised fine-tuning.

2.2.3 Deep Belief Networks

DBNs are probabilistic generative models that are composed of multiple layers of hidden units or feature detectors. A typical DBN has two top layers with undirected and symmetric connections between them to form an associative memory, and has lower layers received top-down directed connections from the layer above, where the units in the bottom layer represent the data [43]. A DBN is often applied for pre-training a DNN, where the data for training such a network is limited and poorly initialized weights can make significant impacts on the performance of the final result. The pre-trained weights obtained from the DBN are employed as the initial weights of the DNN and are located in a region of the weight space that is closer to the optimal weights as compared to the random initialization. Then, these weights can be further fine-tuned by BP or other discriminative algorithms. This achieves for both improved modeling capability and faster convergence of the fine-tuning phase [59].

A DBN can be efficiently trained in an unsupervised, layer-by-layer manner where the layers are typically made of restricted Boltzmann machines (RBMs) [43]. An RBM is an undirected, generative energy-based model with an input layer and single hidden layer. Connections in the RBM only exist between the visible units of the input layer and the hidden units of the hidden layer, where visible-visible or hidden-hidden connections do not exist. The training method for RBMs was initially proposed in [44] and is known as contrastive divergence, which provides an approximation to the maximum likelihood method that would ideally be applied for learning the weights of the RBM [29, 45, 110]. Once an RBM is trained, another RBM can be stacked on the top of it to create a multilayer

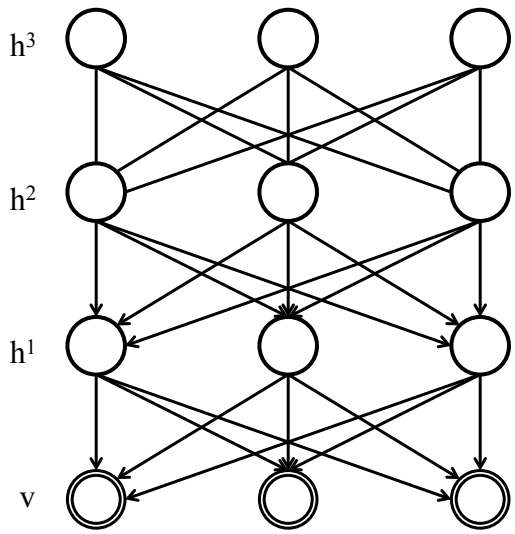


FIG. 2.4. A deep belief network with a set of visible units v and three hidden layers of units h^1, h^2, h^3. The top two layers form a restricted Boltzmann machine, and the remaining layers form a sigmoid belief network with directed and top-down connections.

network. For each stacking RBM, the input visible layer is initialized to a training vector and values of the input units in the previously trained RBM layers are assigned using the current weights and biases, while the final layer of the previously trained layers is used as input to the new RBM. The new RBM is then trained with the procedure above, and then this whole process can be repeated until some desired stopping criterion is met [5]. An example of DBN [95] is shown in Fig. 2.4.

DBNs are learned as one layer at a time by considering the values of the trainable weights that are inferred from the data in one layer as the data for training the next layer. This efficiently greedy layer-wise learning can be combined with other learning procedures that fine-tune all trainable weights to improve the generative or discriminative performance

of the entire network. Discriminative fine-tuning can be performed by adding a final layer of weights that represent the desired outputs and error derivatives of BP. It is mentioned in [40] that, when networks with many hidden layers are applied to highly structured input data such as images, BP works much better if the feature detectors in the hidden layers are initialized by learning a DBN that models the structure in the input data.

2.2.4 Deep Boltzmann Machines

A DBM is an undirected probabilistic graphical model (i.e., a type of Markov random field) with multiple layers of hidden random variables. It is a network of symmetrically coupled stochastic binary units, comprising a set of visible units and a series of layers of hidden units where all connections between layers are undirected. Like RBMs, there is no connection between the units of the same layer in DBMs. Because the DBM has undirected links, symmetric interactions are defined by certain model parameters to represent the visible-hidden and hidden-hidden connections, and these parameters can be easily set to convert a DBM to an RBM [41]. An example of DBM [98] is shown in Fig. 2.5.

DBMs have several advantages compared to other successful deep learning models [2, 95, 97]. Like DBNs, DBMs benefit from the ability of learning complex and abstract internal representations of the input based on the representations built by a large supply of unlabeled data and the overall fine-tuning with a limited number of labeled data. Unlike DBNs and CNNs, DBMs apply the training and inference procedure in both bottom-up and top-down fashions, providing effective approaches to distinguish the high-level representations of the ambiguous from complex input structures. More importantly, parameters of all layers in DBMs can be optimized jointly by following the approximate gradient of a variational lower-bound on the likelihood function, which greatly facilitates the learning of DBMs as better generative models. However, DBMs have a crucial drawback which restricts the performance with the use of the complicated architecture [98]. The approximate

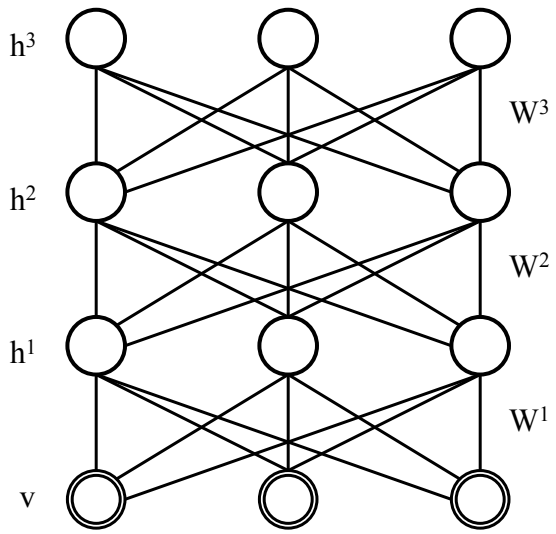


FIG. 2.5. A deep Boltzmann machine with a set of visible units $v \in \{0, 1\}$ and three hidden layers of binary units $h^1, h^2, h^3 \in \{0, 1\}$. All connections between layers are undirected but with no connections between the units of the same layer. Here, W^1, W^2, W^3 are model parameters that represent symmetric connections.

inference based on the mean-field approach in DBMs is around 25 to 50 times slower than a single bottom-up pass in DBNs, which makes the joint optimization of DBM parameters impractical for large datasets and limits the use of DBMs for extracting useful feature representations.

Chapter 3

NORMALIZED RISK-AVERTING ERROR CRITERION

In this chapter, we introduce the normalized risk-averting error (NRAE) as a novel training criterion, which was derived from the RAE criterion, to overcome the local minimum problem in training MLPs. We first present the theoretical foundation of the NRAE criterion, and then we provide the analytical results of feasibility of the NRAE criterion via presenting its bounded computation and robust convexity region. At last, we conceptually demonstrate the effectiveness of the NRAE criterion for avoiding non-global local minima in training MLPs with small data fitting examples.

3.1 Theoretical Foundation

The standard MSE criterion for training an MLP to fit a set of input/output samples (x_k, y_k) for $k = 1, ..., K$ is given in Eq. (2.2) as

$$Q(w) := \frac{1}{K} \sum_{k=1}^{K} \left\| y_k - \hat{f}\left(x_k, w\right) \right\|^2$$

The set of input/output samples (x_k, y_k) is assumed to satisfy the equation $y_k = f\left(x_k\right) + \xi_k$, where f is an unknown or known function, and ξ_k represents the random noise or zero. An optimal weight vector w^* for the best fit of the MLP to the set of input/output samples is

determined by minimizing $Q(w)$ with the variation of w, where $w \in R^N$. Determining w for the MLP to match the input/output samples as closely as possible is called training the MLP on the training data, i.e., the set of input/output samples.

The NRAE criterion [74] is defined as

$$C_\lambda(w) := \frac{1}{\lambda} \ln \left[\frac{1}{K} J_\lambda(w) \right] \tag{3.1}$$

where $J_\lambda(w)$ is the RAE criterion defined in Eq. (2.1) as

$$J_\lambda(w) := \sum_{k=1}^{K} e^{\left(\lambda \left\| y_k - \hat{f}(x_k, w) \right\|^2\right)}$$

as defined in Eq. (2.1). To demonstrate the practical feasibility of $C_\lambda(w)$ in applying as a training criterion, we discuss two principle properties in the following sections.

3.1.1 Bounded Computation

The convexification theorem in [78] states that the convexity region of $J_\lambda(w)$ can be expanded monotonically to the entire weight space as λ increases to infinity, while such expansion can help the RAE training transform the non-convex error space to convex for escaping from poor local minima. It has been confirmed in [72] that the poor local minima can be effectively avoided by selecting small λ values to expand the convexity region of $J_\lambda(w)$ in approximating small-sized functions. However, many real-world training tasks have very large datasets in general, while the training error space for the large-size dataset can be extremely complex. In this case, the computation of $J_\lambda(w)$ is not bounded, if a sufficiently large λ is required to expand the convexity region of $J_\lambda(w)$ to a tremendously great level. Because the evaluation of the exponential function $e^{\left(\lambda \left\| y_k - \hat{f}(x_k, w) \right\|^2\right)}$ in $J_\lambda(w)$ can easily cause register overflow in computer when λ is numerically chosen very large.

In order to employ $C_\lambda(w)$ as a training criterion, all computations in the NRAE training should be bounded without causing any arithmetic overflow. In this section, we demonstrate that $C_\lambda(w)$ together with its gradient

$$\nabla C_\lambda(w) := \left[\frac{\partial C_\lambda(w)}{\partial w_i}\right]_{1\times N} \tag{3.2}$$

and Hessian matrix (simply called as Hessian)

$$H_C(w) := \left[\frac{\partial^2 C_\lambda(w)}{\partial w_i \partial w_j}\right]_{N\times N} \tag{3.3}$$

can be safely calculated without any arithmetic overflow in evaluating the exponential function $e^{\left(\lambda\left\|y_k-\hat{f}(x_k,w)\right\|^2\right)}$, where $i,j=1,...,N$ and N is the total number of trainable weights in the network.

To present the bounded computation of $C_\lambda\,(w)$, let

$$\begin{aligned}\hat{y}_k\,(w) &:= \hat{f}\,(x_k,w)\\ \varepsilon_k\,(w) &:= y_k-\hat{y}_k\,(w)\end{aligned} \tag{3.4}$$

For a weight vector w, let a set

$$S\,(w) = \underset{k\in\{1,...,K\}}{\arg\max}\,\|\varepsilon_k\,(w)\|^2 \tag{3.5}$$

which may contain more than one elements if a tie exists, and

$$M = \min_m \{m\,|\,m\in S\,(w)\} \tag{3.6}$$

which is the smallest index among all values in the set $S\,(w)$. Note that the number $|S\,(w)|$

of elements in $S(w)$ may be larger than one. It follows that $\|\varepsilon_k(w)\|^2 \leq \|\varepsilon_M(w)\|^2$. Let

$$\eta_k(w) := e^{\lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)} \tag{3.7}$$

where $0 < \eta_k(w) \leq 1$ and $0 < \sum_{k=1}^{K} \eta_k(w) \leq K$. Then, the NRAE criterion $C_\lambda(w)$ in Eq. (3.1) can be rewritten as

$$\begin{aligned} C_\lambda(w) &= \frac{1}{\lambda} \ln\left(\frac{1}{K} J_\lambda(w)\right) \\ &= \frac{1}{\lambda} \ln\left(\frac{1}{K} \sum_{k=1}^{K} e^{\lambda\|\varepsilon_k(w)\|^2}\right) \\ &= \frac{1}{\lambda} \ln\left(\frac{1}{K} e^{\lambda\|\varepsilon_M(w)\|^2} \sum_{k=1}^{K} \eta_k(w)\right) \\ &= \frac{1}{\lambda} \ln\frac{1}{K} + \|\varepsilon_M(w)\|^2 + \frac{1}{\lambda} \ln\left(\sum_{k=1}^{K} \eta_k(w)\right) \end{aligned} \tag{3.8}$$

where

$$C_\lambda(w) \leq \frac{1}{\lambda} \ln\frac{1}{K} + \|\varepsilon_M(w)\|^2 + \frac{1}{\lambda} \ln K \leq \|\varepsilon_M(w)\|^2$$

It indicates that $C_\lambda(w)$ is bounded by the term $\|\varepsilon_M(w)\|^2$, which is independent of λ, and no computational overflow occurs when $\lambda >> 1$.

Consider the first-order derivative of $C_\lambda(w)$

$$\begin{aligned} \frac{\partial C_\lambda(w)}{\partial w_i} &= \frac{1}{\lambda J_\lambda(w)} \frac{\partial J_\lambda(w)}{\partial w_i} \\ &= \frac{1}{\lambda J_\lambda(w)} \left[-2\lambda \sum_{k=1}^{K} e^{\lambda\|\varepsilon_k(w)\|^2} \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i}\right] \\ &= \frac{-2\lambda e^{\lambda\|\varepsilon_M(w)\|^2} \sum_{k=1}^{K} \eta_k(w) \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i}}{\lambda e^{\lambda\|\varepsilon_M(w)\|^2} \sum_{k=1}^{K} \eta_k(w)} \\ &= \frac{-2 \sum_{k=1}^{K} \eta_k(w) \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i}}{\sum_{k=1}^{K} \eta_k(w)} \end{aligned} \tag{3.9}$$

where

$$\sum_{k=1}^{K} \eta_k(w) \leq K$$
$$\left| \sum_{k=1}^{K} \eta_k(w) \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i} \right| \leq \sum_{k=1}^{K} \left| \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i} \right|$$

The computation of $\frac{\partial \hat{y}_k(w)}{\partial w_i}$ is performed by BP. Hence, both the numerator and denominator of Eq. (3.9) are bounded without computational overflow when $\lambda >> 1$.

Consider the second order derivative of $C_\lambda(w)$

$$\frac{\partial^2 C_\lambda(w)}{\partial w_i \partial w_j} = \frac{1}{\lambda J_\lambda(w)} \frac{\partial^2 J_\lambda(w)}{\partial w_i \partial w_j} - \frac{1}{\lambda J_\lambda^2(w)} \frac{\partial J_\lambda(w)}{\partial w_i} \frac{\partial J_\lambda(w)}{\partial w_j} \tag{3.10}$$

It has been shown in [78] that

$$\frac{\partial^2 J_\lambda(w)}{\partial w_i \partial w_j} = 2\lambda \sum_{k=1}^{K} e^{\lambda \|\varepsilon_k(w)\|^2} \{2\lambda A_{kij}(w) + B_{kij}(w) - C_{kij}(w)\} \tag{3.11}$$

where

$$\begin{aligned} A_{kij}(w) &:= \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i} \frac{\partial \hat{y}_k^T(w)}{\partial w_j} \varepsilon_k(w) \\ B_{kij}(w) &:= \frac{\partial \hat{y}_k^T(w)}{\partial w_i} \frac{\partial \hat{y}_k(w)}{\partial w_j} \\ C_{kij}(w) &:= \varepsilon_k^T(w) \frac{\partial^2 \hat{y}_k(w)}{\partial w_i \partial w_j} \end{aligned} \tag{3.12}$$

are all $N \times N$ matrices. It follows that

$$\frac{1}{\lambda J_\lambda(w)} \frac{\partial^2 J_\lambda(w)}{\partial w_i \partial w_j} = \frac{2 \sum_{k=1}^{K} \eta_k(w) \{2\lambda A_{kij}(w) + B_{kij}(w) - C_{kij}(w)\}}{\sum_{k=1}^{K} \eta_k(w)} \tag{3.13}$$

Recalling in [78] that

$$\frac{\partial J_\lambda(w)}{\partial w_i} = -2\lambda \sum_{k=1}^{K} e^{\lambda \|\varepsilon_k(w)\|^2} \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i} \tag{3.14}$$

we obtain

$$\frac{1}{\lambda J_\lambda^2(w)} \frac{\partial J_\lambda(w)}{\partial w_i} \frac{\partial J_\lambda(w)}{\partial w_j} = \frac{4\lambda \left(\sum_{k=1}^{K} \eta_k(w) \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i} \right)}{\sum_{k=1}^{K} \eta_k(w)} \cdot \frac{\left(\sum_{k=1}^{K} \eta_k(w) \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_j} \right)}{\sum_{k=1}^{K} \eta_k(w)} \tag{3.15}$$

Notice that $0 < \eta_k(w) \leq 1$. Hence, Eq. (3.10) is bounded without any computational overflow when $\lambda >> 1$.

3.1.2 Convexification Property

Since the logarithm function is monotonically increasing, $C_\lambda(w)$ is a strictly monotone function of $J_\lambda(w)$. Accordingly, the convexity region of $C_\lambda(w)$ that contains the convexity region of $J_\lambda(w)$ expands in the same way as the convexity region of $J_\lambda(w)$ stretches as λ increases, if the convexity of $J_\lambda(w)$ is maintained as Theorem 1 states. It indicates that if the convexity region of $J_\lambda(w)$ does not have non-global local minima in nearly the entire weight space for a sufficiently large λ, the non-global local minima will not appear in the convexity region of $C_\lambda(w)$ that contains the said convexity region of $J_\lambda(w)$. Therefore, $C_\lambda(w)$ shares the same local and global optima with $J_\lambda(w)$. We summarize the above discussion as the following theorem:

Theorem 2 (NRAE Convexity)**.** *Given a weight or parameter space $w \in R^N$, if $D_K(w)$ is full rank, $\exists \Lambda \in R^+$ such that $H_J(w) > 0$ when $\lambda > \Lambda$ to maintain the convexity of the risk-averting error $J_\lambda(w)$, then the normalized risk-averting error $C_\lambda(w)$ shares the same*

local and global optima with $J_\lambda(w)$.

It is appropriate to note that, since $C_\lambda(w)$ is a concave function of $J_\lambda(w)$, the convexity region of $C_\lambda(w)$ does not expand as λ increases. It implies that $C_\lambda(w)$ is not able to be convexified like $J_\lambda(w)$ as λ increases. However, if a global or near-global minimum is achieved when the convexity region of $J_\lambda(w)$ is significantly expanded to the entire weight space as λ approaches infinity, the values of $\left\| y_k - \hat{f}(x_k, w) \right\|^2$ for $k = 1, ..., K$ in Eq. (2.2), Eq. (2.1), and Eq. (3.1) are all very small and close to zero. In this case, the training errors of $C_\lambda(w)$ and $Q(w)$ according to the global or near-global minimum are very close to zero, implying that $C_\lambda(w)$ shares the same global or near-global minimum with $Q(w)$.

If a global or near-global minimum is unavailable when the convexity region of $J_\lambda(w)$ is expanded to the entire weight space as λ increases, the NRAE criterion is still expected to perform as good as the MSE criterion when the convexity region of $C_\lambda(w)$ is shrunk as λ decreases to zero. It can be proven by rewriting (3.8) when λ approaches zero as

$$
\begin{aligned}
\lim_{\lambda \to 0} C_\lambda(w) &= \lim_{\lambda \to 0} \frac{1}{\lambda} \ln \left(\frac{1}{K} \sum_{k=1}^{K} e^{\lambda \|\varepsilon_k(w)\|^2} \right) \\
&= \lim_{\lambda \to 0} \frac{1}{\lambda} \ln \left(1 + \frac{1}{K} \sum_{k=1}^{K} \lambda \|\varepsilon_k(w)\|^2 + O(\lambda^2) \right) \\
&= \lim_{\lambda \to 0} \frac{1}{\lambda} \left(\frac{1}{K} \sum_{k=1}^{K} \lambda \|\varepsilon_k(w)\|^2 + O(\lambda^2) \right) \\
&= \frac{1}{K} \sum_{k=1}^{K} \|\varepsilon_k(w)\|^2 \\
&= Q(w)
\end{aligned}
\tag{3.16}
$$

where Taylor series expansions of exponential and logarithm functions

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots \tag{3.17}$$
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

are applied. Obviously, the NRAE criterion in (3.16) acts the same as the MSE criterion.

Overall, the computationally bounded NRAE criterion preserves the capability of avoiding the non-global local minimum when the convexity region of $J_\lambda(w)$ expands as λ increases to infinity, while achieving the same global or near-global minimum with the MSE criterion as λ decreases to zero. Therefore, a convexification or deconvexification procedure can be applied to alternate the convexity region of $C_\lambda(w)$ by increasing or decreasing λ as it ranges from 0 to ∞, thus leading to a global or near-global minimum of both the NRAE and the MSE criteria.

3.2 Evaluating the NRAE Criterion

For conceptually demonstrating the capability of NRAE as a training criterion, we employ two small data fitting examples to evaluate the NRAE criterion in training MLPs. Our primary goal here is to find out whether the NRAE training is able to avoid or alleviate non-global local minima through the convexification procedure with achieving a much better performance than the MSE training. More systematically experimental evaluation of the NRAE criterion will be given in next chapter.

3.2.1 Experimental Settings

To efficiently develop our experiments, the standard training of MLP is performed with the aid of several practical techniques, which have been suggested in [38, 65] and list as the following items. We stress that most techniques here are applied not only for the

evaluation of the NRAE criterion in this section, but also for all experiments performed in this dissertation unless particular experimental settings are stated.

1. The activation function in each training node of MLP is chosen as the hyperbolic tangent function $\varphi(v) = a\tanh(bv)$, where $a = 1.7159$ and $b = \frac{2}{3}$. With the setting, $\varphi(v)$ satisfies $\varphi(1) = 1$ and $\varphi(-1) = -1$. In addition, the second order derivative of $\varphi(v)$ can attain its maximum value at $v = 1$. The activation function can efficiently avoid the node saturation in training MLPs, and it has all benefits of the sigmoid function with satisfying the condition $\varphi(-v) = -\varphi(v)$.

2. All input and output defined in the training dataset are normalized into $[-1, 1]$ for keeping the mean value of training samples closing to zero. Such a normalization of the training dataset can efficiently avoid the "zigzagging" searching of the optimal weights on the training error space.

3. The weights of MLP are initialized by randomly selecting from a uniform distribution between $-2.4/F_i$ and $2.4/F_i$, where F_i is the fan-in (i.e., the number of inputs that can be connected to a node) of the i-th node. This initialization maintains the initial standard deviation of the weighted sums of each training node in the same range over all nodes, and keeps these nodes working on the normal operating region of the activation function. Otherwise, too large initial weights will drive hidden nodes into saturation, and too small initial weights will cause BP to work on a very flat area around the origin of the training error space.

4. The derivatives of MLP are computed by the standard BP. The weights of MLP are updated by BFGS in data fitting or function approximation tasks, and are updated by GD in classification tasks. The convergence of a training session is considered if the improvement of training errors according to the MSE values is less than 10^{-10} in

1,000 consecutive training epochs.

3.2.2 Learning XOR

The exclusive-OR (XOR) problem has been widely explored with the training difficulty of the presence of local minima and has been thoroughly discussed in many literatures [9, 11, 46, 70, 92, 106–108]. As a classic training problem, it has been commonly applied to demonstrate the capability of the MLP in handling the problem that is not linearly separable against the single-layer perceptron. In this section, we adopt the XOR problem to illustrate the effectiveness of the NRAE training compared to the MSE training in avoiding non-global local minima, and reveal how the convexification works on alternating the non-convex training error space.

The training dataset of the XOR problem often consists of four samples: (-1,-1,-1), (-1,1,1), (1,-1,1) and (1,1,-1). The first two values in each training sample compose a input vector x_k, and the last value in each training sample is the output value y_k, where $k = 1, 2, 3, 4$. The objective of the XOR task is to find the proper weights through training the MLP to minimize the error between the actual output $\hat{y}_k$ of the MLP and the target output y_k defined by the training dataset. As presented in Fig. 3.1, a 2-2-1 MLP, which contains two input nodes, one hidden layer with two hidden nodes, and one output node, is selected to perform the training of XOR with using the minimum number of trainable weights of the MLP.

In our experiments, we perform two different MLP training sessions using the same initial weights, where one session applies the MSE criterion and the other one applies the NRAE criterion with $\lambda = 0.5$. The initial weights of the MLP are randomly selected from $[-15, 15]$ rather than $[-2.4/F_i, 2.4/F_i]$. We employ this particular selection of initial weights because the XOR problem is easily to escape from local minima in the MSE training if the initial weights are selected from $[-2.4/F_i, 2.4/F_i]$ (i.e., close to the range of

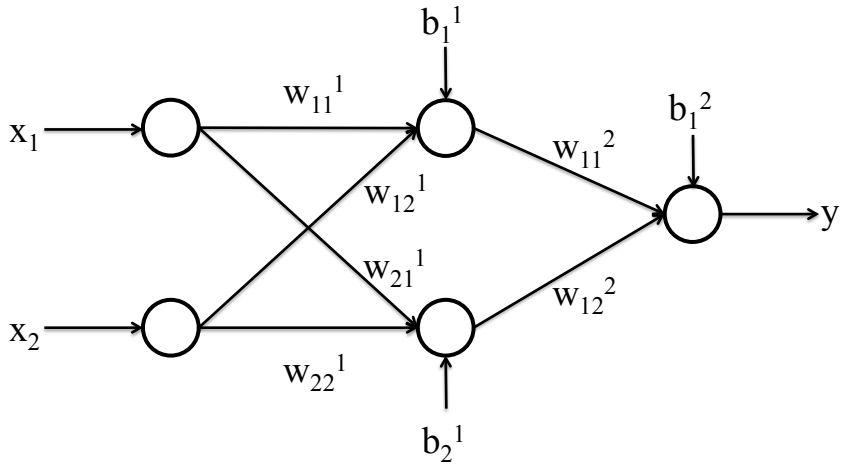


FIG. 3.1. A 2-2-1 MLP applied to learn the XOR function.

[-1,1]). In fact, it has been pointed out in [9, 46] that the XOR problem has the difficulty of avoiding local minima especially when the initial weights are very small (e.g., in the range of [-0.25,0.25]) or very large (e.g., in the range of [-5,5]). With selecting the initial weights from a larger range, we can test whether the NRAE training is able to avoid the non-global local minimum, which can be difficult for the MSE training to escape from.

From the two training sessions, we obtain two optimal weight vectors w^*_{MSE} and w^*_{NRAE} corresponding to the MSE training and the NRAE training until the convergence, respectively. After that, we randomly choose two weights w^1_{MSE} and w^2_{MSE} from w^*_{MSE} and set them as variables to change in the range of $[-15, 15]$. Then, we draw a contour map for the MSE criterion $Q(w)$ as the function of w^1_{MSE} and w^2_{MSE}. Because both the MSE training and the NRAE training are performed under the same MLP architecture with the same number of trainable weights, we employ w^1_{NRAE} and w^2_{NRAE} as variables to draw another contour map

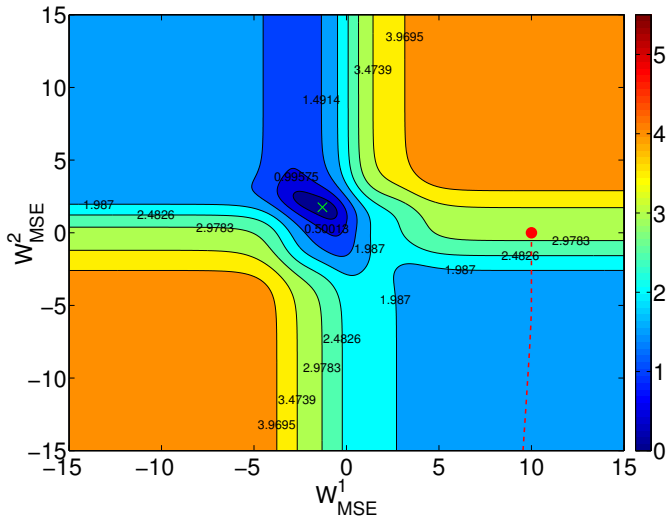
for the NRAE criterion $C_\lambda(w)$, where w^1_{NRAE} and w^2_{NRAE} are the corresponding weights chosen with the same indices of w^1_{MSE} and w^2_{MSE} as applied in the MSE training. These contour maps are shown in Fig. 3.2.

Fig. 3.2(a) demonstrates that the MSE training leads the searching path of the optimal weight vector to a poor non-global local minimum, which training error is plotted as the light blue color and located in the right bottom of the contour map. In addition, it is observed that the mentioned searching path in the MSE training is blocked by a continuous plateau plotted with the cyan color for reaching the global minimum. However, as illustrated in Fig. 3.2(b), both the light blue non-global local minimum and the cyan plateau located in the MSE training are expanded and merged together by the convexification in applying the NRAE criterion. An optimal searching path, which leads the NRAE training travel from a poor non-global local minimum to a global minimum, is created in the NRAE training.

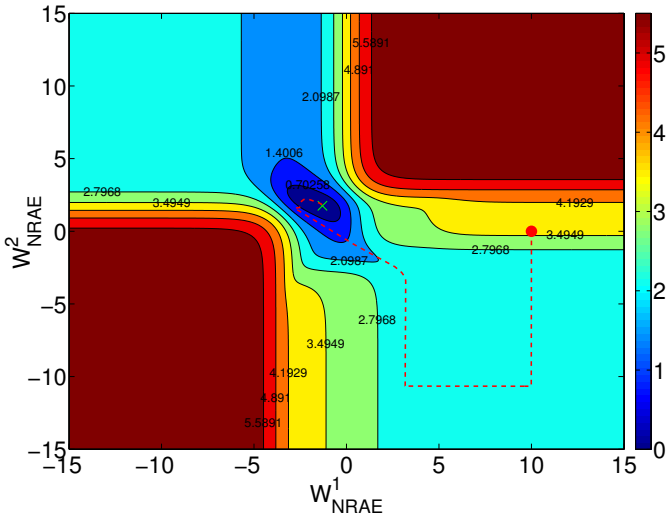
In order to evaluate the NRAE training in more test cases, we repeat the experiments of the MSE and the NRAE training in 100 times with randomly selecting 100 different sets of initial weights from $[-15, 15]$. Our experimental results indicate that only 27 out of 100 MSE training sessions can successfully escape from local minima and reach the global minimum, but all 100 NRAE training sessions consistently achieve global minima with the training errors closed to 10^{-10}. Therefore, the NRAE training confirms the effectiveness of the convexification methodology and demonstrates the tremendous superiority comparing to the MSE training in avoiding the local minimum problem.

3.2.3 Approximating One-notch Function

To further test the capability of the NRAE training in avoiding the non-global local minimum, we employ a one-notch function, which is designed to have the non-global local



(a) MSE Training



(b) NRAE Training ($\lambda = 0.5$)

FIG. 3.2. Contour maps for the MSE and the NRAE criteria as functions of two selected weights in training XOR. Different colors indicate different training errors shown in the color bar of each contour map. Exact number in each contour line denotes the MSE value of the training error. The red dot indicates the position of initial weight vector and the green cross indicates the location of the global minimum in the tested XOR problem. The red dash line shows the searching path of the optimal weight vector in the training.

minimum, to test the NRAE criterion. The one-notch function is defined as

$$y = f(x) = \begin{cases} 1 & \text{if } x \in [1, 2] \cup [2.1, 3.1] \\ 0 & \text{otherwise} \end{cases} \tag{3.18}$$

where $x \in X = [0, 4.1]$.

We evaluate three training models to approximate the one-notch function. They are separately defined as: an MLP trained by the MSE criterion, an MLP trained by the NRAE criterion, and a support vector machine (SVM) with the radial basis function (RBF) kernel. We evaluate whether the NRAE training is able to avoid the non-global local minimum existed in the one-notch function and approximate the function better than the MSE training with the use of the same MLP. Meanwhile, we test whether the NRAE training can achieve a better result than the SVM, which is commonly considered as one of the most famous and powerful learning models in solving regression and classification tasks [6]. At last, we evaluate if the NRAE training is able to tolerate data noises, and illustrate how well the noises can be generalized by the NRAE training compared to both the MSE and the SVM training.

To perform our experiments, we generate the training data of the one-notch function as follows. For the training data, input values x_k are selected by randomly sampling 2,000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (3.18). Then, a training dataset with 2,000 (x_k, y_k) samples is obtained. In addition, we need the validation data for inducing and testing generalization capabilities of well-trained models. For the validation data, input values x_k are selected by randomly sampling 1,000 different numbers from a uniform distribution on X, and corresponding output values y_k are also computed by Eq. (3.18). Then, a validation dataset with 1000 (x_k, y_k) samples is obtained. Here, the training and the validation datasets are independent and non-overlapping. To test the noise tolerance, we generate new noisy dataset

through adding noises to the target outputs y_k of the original 2,000 training samples and 1,000 validation samples. The level of added noises is specified by the signal-to-noise ratio (SNR), which is 10 times the natural logarithm of the ratio of the sum of squares of the target outputs O and the sum of squares of the noises E. Here, we use the Gaussian white noises with setting SNR as $10\log_{10}2^2 = 6$dB. The target one-notch functions with and without noises are presented in Figure 3.3(a) and Figure 3.3(b), respectively.

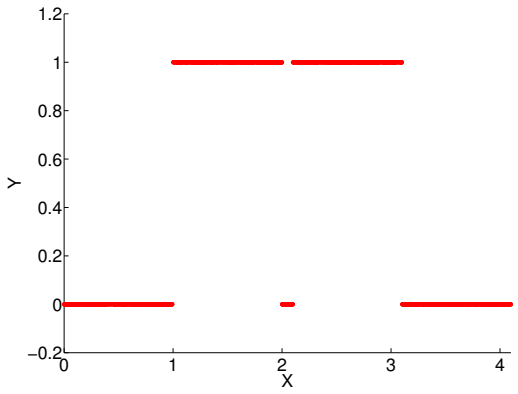
Two 1-4-1 MLPs started with the same initial weights are chosen to train the one-notch function separately by using the MSE and the NRAE criterion. For the noiseless data, it is noted in Fig. 3.3(c) and Fig. 3.3(e) that the approximated function trained by the NRAE criterion with $\lambda = 3.5$ successfully find the small notch defined in the target one-notch function. This approximation result is better than the function trained by the MSE criterion. Although the MSE training can be improved by carefully selecting proper initial weights at the beginning of the training, it costs an excessive amount of time to make these selections. Actually, based on our experiments, the MSE training sessions with adopting different initial weights never outperform a single NRAE training session at all. One reason is that the proper initial weights for the MSE training cannot fundamentally eliminate or avoid poor non-global local minima in the error space of the one-notch function, while these initial weights only provide proper starting points for the local-searching method to travel on the error space. If one or more non-global local minima appear on the searching path, the MSE training is stuck into these local minima. However, the NRAE training does not encounter this difficulty, because poor non-global local minima are able to be avoided or alleviated if a proper λ is chosen as well as the convexity region is sufficiently expanded. Therefore, not like the MSE training, there is no need to consider the initial weights selection in the NRAE training, because the multiple trials of different initial weights can be avoided by choosing the proper λ value in the NRAE training. For example, one of our tests shows that choosing different initial weights from the range of $[-100, 100]$ or the

range of $[-0.01, 0.01]$ can be completely handled by the NRAE training with reasonable λ values. All training sessions with those initial weights provide similar one-notch function approximation results, which virtually appear the same as Fig. 3.3(e).

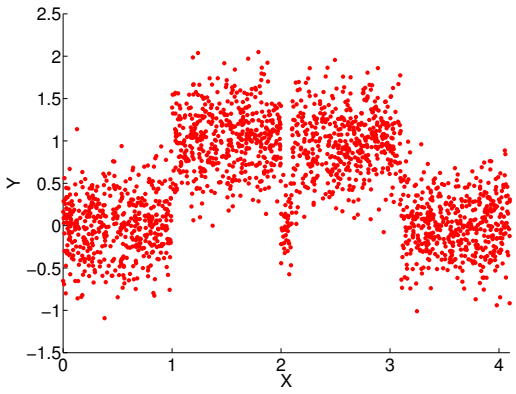
Another two 1-4-1 MLPs trained with the same initial weights are applied to approximate the one-notch function with measurement noises. Training results with approximated functions compared to the target functions are shown in Fig. 3.3(d) and Fig. 3.3(f). It is observed that the MSE training fits the most target data with a moderate noise level, but it merely fits the few target data of the pre-designed notch. On the other hand, the NRAE training with $\lambda = 6.5$ almost approximates all target data with a little fitting of noises under the same amount of measurement noises as the MSE training applied. Especially, the NRAE training fits the target data of the pre-designed notch well. This experiment illustrates that the NRAE training with a proper λ can provide a certain level of noise tolerance to approximate function with using the MLP.

The SVM is chosen to compare with the MLPs trained by the MSE or the NRAE criterion for the function approximation. We perform the SVM training with the regression learning option provided in SVM^{light} tool [53] to approximate the one-notch function with noiseless or noisy data. Our experimental results are shown in Fig. 3.3(g) and Fig. 3.3(h). The kernel function in the SVM training is RBF, which is denoted as $e^{(-\gamma\|a-b\|^2)}$ with $\gamma = 100$. It is noticed that such a large value of γ in the SVM training leads the approximated function to have a very large order, thus it is unable to fit the target one-notch function very well because of the severely overfitting. In fact, when we choose a smaller $\gamma = 10$, the RBF kernel significantly reduces the overfitting and makes the approximated function more smooth, but the pre-designed notch in the target function is also neglected by the SVM training. Moreover, when we select a very large $\gamma = 500$, the RBF kernel makes the approximating results even worse, because a much higher order of the approximated function greatly intensifies the overfitting in the SVM training. Based on our experiments,

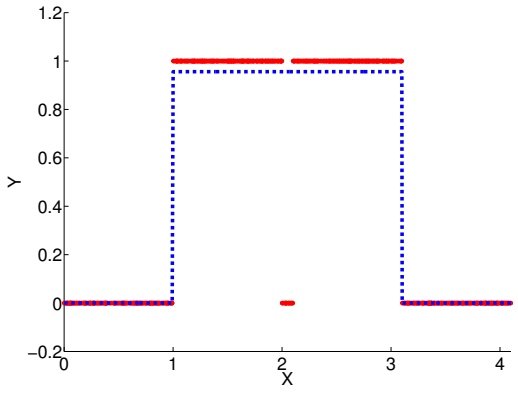
$\gamma = 100$ provides the best function approximation result over all of the SVM training trials. The one-notch function approximation results demonstrate that the SVM training is able to provide a better result than the MSE training with the 1-4-1 MLP, because the approximated function with a high order obtained by the SVM training has more expression capability to the one-notch function than the MLP trained by the MSE criterion. However, both the MSE and the SVM training do not have effective approaches to avoid or alleviate the non-global local minimum located in the one-notch function. Therefore, they can hardly provide a well-fitted one-notch function as what the NRAE training achieves.



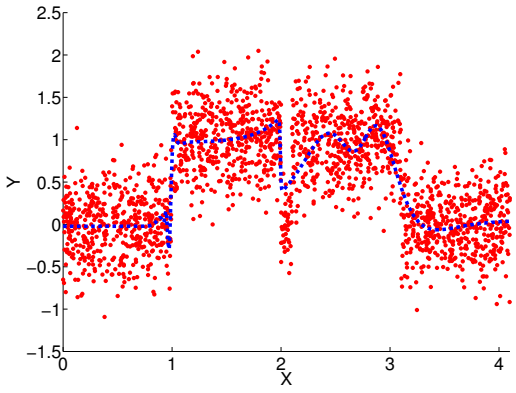
(a) One-notch Function



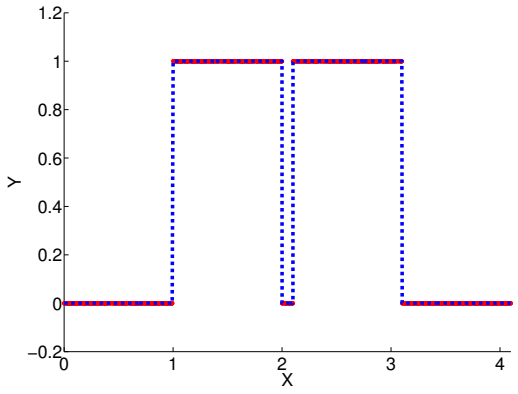
(b) One-notch Function with Noises



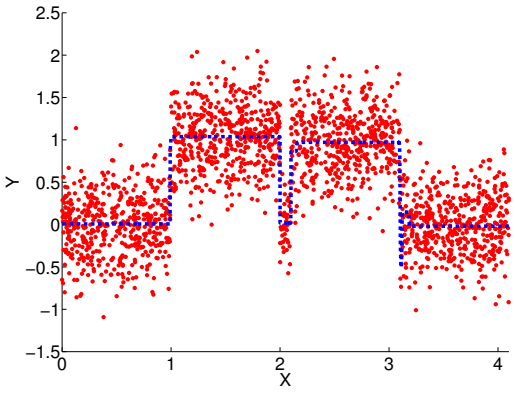
(c) MSE Training



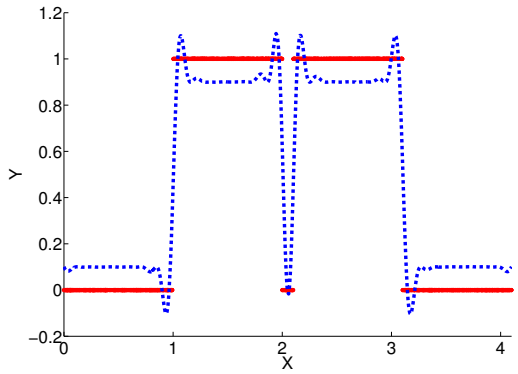
(d) MSE Training with Noises



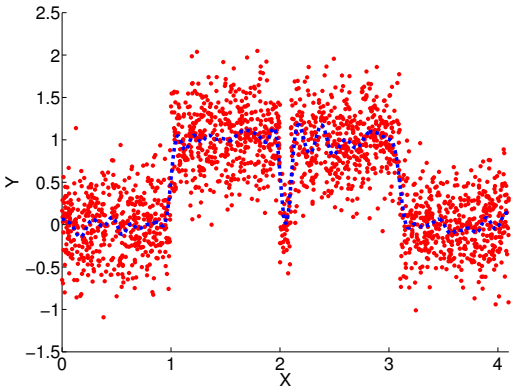
(e) NRAE Training ($\lambda = 3.5$)



(f) NRAE Training with Noises ($\lambda = 6.5$)



(g) SVM Training ($\gamma = 100$)



(h) SVM Training with Noises ($\gamma = 100$)

FIG. 3.3. Target functions and fitting plots for approximating the one-notch function. Numbers on horizontal and vertical axes in each subfigure represent the input and output of the function. Red dots denote target training data, and blue dash lines are approximated function plots. The figures on the left side describe the trained results with the noiseless data, and the figures on the right side present the results trained with the SNR=6dB noisy data.

3.3 Summary

The NRAE criterion, which is derived from the RAE criterion, is developed to overcome the local minimum problem in training ANNs. Since the convexity region of NRAE that contains the convexity region of RAE expands in the same way as the convexity region of RAE stretches as λ increases, and the NRAE does not grow exponentially like the way of RAE does as λ increases as well, the NRAE criterion can be applied under large values of λ to greatly expand the convexity region for avoiding the non-global local minimum without any computational overflow. Therefore, comparing to the MSE and RAE criteria, the NRAE criterion is better suited for overcoming the local minimum problem in training ANNs.

The effectiveness of the NRAE criterion in avoiding the non-global local minimum is conceptually evaluated by training MLPs on two small data fitting examples: the XOR problem and the one-notch function. Experimental results demonstrate significant advantages of the NRAE criterion compared to the standard MSE criterion in training MLPs to reach the global minimum, illustrating the feasibility of the NRAE criterion for solving small learning tasks with non-global local minima.

Chapter 4

CONVEXIFICATION METHODOLOGY

To systematically demonstrate the practical performance of the NRAE criterion in solving real-world tasks rather than small data fitting examples, this chapter presents the evaluation of the NRAE criterion using the NRAE training and the NRAE-MSE training methods, which are developed mainly according to the convexification procedure in NRAE. Both methods are tested on function approximation and handwritten digit recognition tasks, illustrating the effectiveness of the NRAE criterion in avoiding non-global local minima with applicable ranges of λ. In addition, we study a tricky but significant difficulty called the stagnant problem in training MLPs with the NRAE criterion, and we present concrete experimental evidences to confirm conjectures and provide solutions for that conundrum.

4.1 NRAE Training Method

As we discussed in Chapter 3, since the convexity region of $C_\lambda(w)$ that contains the convexity region of $J_\lambda(w)$ expands in the same way as the convexity region of $J_\lambda(w)$ stretches as λ increases, a straightforward convexification methodology based on the NRAE criterion is to apply $\lim_{\lambda\to\infty} C_\lambda(w)$ as an error criterion. However, $\lim_{\lambda\to\infty} C_\lambda(w)$ is discontinuous thus cannot effectively be used as the training criterion. This motivates the NRAE training method [74], which uses $C_\lambda(w)$ at a very large λ instead of the standard

MSE criterion $Q(w)$ throughout the training until a desired training error is achieved. The NRAE training method is described in Algorithm 1.

Algorithm 1 NRAE Training Method

Require: Initialize the weight vector w randomly, set a desired training error δ, and select $\lambda >> 1$;

1: **while** $C_\lambda(w) \geq \delta$ **do**
2: **for** $i = 1$ to N **do**
3: Evaluate the first-order derivative of $C_\lambda(w)$ with respect to w_i as $\frac{\partial C_\lambda(w)}{\partial w_i} \leftarrow \frac{-2\sum_{k=1}^{K} \eta_k(w)\varepsilon_k^T(w)\frac{\partial \hat{y}_k(w)}{\partial w_i}}{\sum_{k=1}^{K}\eta_k(w)}$;
4: Update w_i to w_i^* using $\frac{\partial C_\lambda(w)}{\partial w_i}$;
5: **end for**
6: Set $w \leftarrow w^*$;
7: **end while**
8: **return** The optimal weight vector w^*.

4.2 NRAE-MSE Training Method

Although the NRAE training method can be performed all the way until the best training error is reached, finding the optimal weight vector with the use of $C_\lambda(w)$ needs more computational efforts than using $Q(w)$. If the convexity region of $C_\lambda(w)$ can be greatly expanded at a very large λ, then it would be a great chance to avoid or alleviate the non-global local minimum existed in the error space of both $C_\lambda(w)$ and $Q(w)$, because they share the same local and global minima. In this case, the training criterion can be switched from $C_\lambda(w)$ to $Q(w)$ after the NRAE training is performed for a reasonable number of iterations. If a global or nearly global minimum is reached in the training excursion with the MSE criterion, the training is considered successful and stopped. Otherwise, the NRAE training resumes from the weight vector that the MSE excursion started with and continues for another reasonable number of iterations to be followed by another excursion with the

MSE criterion. In short, this approach comprises a sequence of cycles, where each cycle consists of an NRAE training followed by an MSE recursion. This new approach to use the NRAE criterion is called the NRAE-MSE training method [76].

The NRAE-MSE training method is described in Algorithm 2. Here, the weight vectors computed in the NRAE training and the MSE excursion are denoted as $w_{\mathrm{C}}(i)$ and $w_{\mathrm{Q}}(j)$, where $i = 1, ..., L_{\mathrm{NRAE}}$ and $j = 1, ..., L_{\mathrm{MSE}}$, respectively. The NRAE-MSE training method mainly repeats the following two steps:

1. The NRAE training: starting with a weight vector w, apply $C_\lambda(w)$ with the selected λ to perform the training for L_{NRAE} iterations. Each iteration starts with the current weight vector $w_{\mathrm{C}}(i)$ and produces a new weight vector, which is stored as the weight vector $w_{\mathrm{C}}(i+1)$ to replace the preceding one for the next iteration until the last iteration is ended with a weight vector $w_{\mathrm{C}}(L_{\mathrm{NRAE}})$.

2. The MSE excursion: starting with a weight vector $w_{\mathrm{C}}(L_{\mathrm{NRAE}})$, apply $Q(w)$ to perform the training for L_{MSE} iterations. In the process of the last iterations, if $Q(w)$ is less than or equal to a desired training error, or if a validation data shows that an overfitting of the training data occurs, stop the entire NRAE-MSE training. Otherwise, complete performing the last iterations, store the current weight vector as $w_{\mathrm{Q}}(L_{\mathrm{MSE}})$, and return to the step 1.

It is worth noticing that, although the NRAE-MSE training method switches between using NRAE and using MSE, the latter is only an excursion, but not a detour, from the former. Because the MSE excursion does not disturb the path of the NRAE training, the two steps in the NRAE-MSE training method are able to be implemented separately with reasonable number of interactions controlled by L_{NRAE} and L_{MSE}. Therefore, applying a parallel computing technique will significantly speedup the NRAE-MSE training method in practice.

Algorithm 2 NRAE-MSE Training Method

Require: Initialize the weight vector w randomly, choose L_{N} and L_{M}, set a desired training error δ, and select $\lambda >> 1$;

while $Q(w) > \delta$ or the overfitting is not detected **do**
  Set $w_{\mathrm{C}}(1) \leftarrow w$;
  for $i = 1$ to $L_{\mathrm{NRAE}} - 1$ **do** {Step 1: the NRAE training}
    Evaluate the first-order derivative of $C_\lambda(w_{\mathrm{C}}(i))$;
    Update $w_{\mathrm{C}}(i)$ to $w_{\mathrm{C}}(i+1)$;
  end for
  Set $w_{\mathrm{Q}}(1) \leftarrow w_{\mathrm{C}}(L_{\mathrm{NRAE}})$;
  for $j = 1$ to $L_{\mathrm{MSE}} - 1$ **do** {Step 2: the MSE excursion}
    Evaluate the first-order derivative of $Q(w_{\mathrm{Q}}(j))$;
    Update $w_{\mathrm{Q}}(j)$ to $w_{\mathrm{Q}}(j+1)$;
  end for
  Set $w^* \leftarrow w_{\mathrm{Q}}(L_{\mathrm{MSE}})$ and $w \leftarrow w^*$;
end while
return The optimal weight vector w^*.

4.3 Experimental Evaluation

In this section, the NRAE training and the NRAE-MSE training methods are evaluated by four function approximation examples and one handwritten digit recognition task, demonstrating the effectiveness of the proposed convexification methodology based on the NRAE criterion.

4.3.1 Function Approximation

Experimental Design Four target functions are designed for their MSEs to have non-global local minima in our experiments, which are intended to test the capability of the NRAE-based training methods in avoiding the non-global local minimum. To approximate a target function, 10 different sets of initial weights of MLP with a certain architecture are randomly chosen. Starting with such a set of initial weights, one standard MSE training, one NRAE training and one NRAE-MSE training session are performed. These three

training sessions for one and the same set of initial weights are called a training group. The corresponding two values of $Q(w)$ of the MLP resulting from the two NRAE-based training methods are recorded for comparing to the $Q(w)$ of the MLP resulting from the standard MSE training.

In order to test the capability of the NRAE-MSE training to tolerate noises in the training data, 10 additional training groups are composed for those four function approximation examples with adding noises to the target outputs in the training data. Here, we keep using Gaussian white noises with setting SNR as $10\log_{10} 2^2 = 6$dB. The same training strategies as applied for noiseless data are adopted for noisy data in each function approximation example. In addition, we use the validation data to test generalization capabilities in all the MSE and NRAE-MSE training sessions with noisy data. The size of the validation data is one half of the noisy training data, and the validation data is randomly selected from the target function with no overlapping to the training data.

In each of the sessions with the MSE or the NRAE training method, the maximum number of training epochs is set to 10^6. The NRAE-MSE training method is considered as comprising a sequence of cycles, where each cycle consists of an NRAE training followed by an MSE recursion. Hence, in the NRAE-MSE training method, we set $L_{\text{NRAE}} = 1 \times 10^4$ and $L_{\text{MSE}} = 1 \times 10^4$, while the maximum number of cycles is set to 50. As a result, the maximum number of training epochs including those in the NRAE training and the MSE recursion is 10^6, which is equal to the number of epochs in the MSE training as well as the NRAE training.

We present our experimental results for seven large λ values as 10^6, 10^7, 10^8, 10^9, 10^{10} and 10^{11}. For every function approximation example, each λ value is used to perform ten training groups with different sets of initial weights, where each training group contains one NRAE training session and one NRAE-MSE training session using the same set of initial weights for the given λ. Since we have 4 function approximation examples, 10 sets

of initial weights, 7 values of λ, and 2 datasets with or without noises, we totally perform $4 \times 10 \times 7 \times 1 = 280$ trials for the NRAE training method, $4 \times 10 \times 7 \times 2 = 560$ trials for the NRAE-MSE training method, and $4 \times 10 \times 2 = 80$ trials for the MSE training method.

Target functions used in function approximation examples are presented in Fig. 4.1. Definitions of target functions with the training and validation data, and MLP architectures for the experiments are described as the following. These settings of function approximation examples are always maintained the same as our descriptions in this section for all experiments once these functions are applied.

Three-notch A function with three notches is defined by

$$y = f(x) = \begin{cases} 0 & \text{if } x \in [0, 1.0] \cup [2.2, 2.3] \cup [3.5, 4.5] \\ 0.25 & \text{if } x \in [2.8, 3.0] \\ 0.5 & \text{if } x \in [1.5, 1.7] \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

where $x \in X = [0, 4.5]$. For the training data, input values x_k are selected by randomly sampling 2,000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (4.1). Then, a training dataset with 2,000 (x_k, y_k) samples is obtained. For the validation data, input values x_k are selected by randomly sampling 1,000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (4.1). Then, a validation dataset with 1,000 (x_k, y_k) samples is obtained. Here, the training dataset and the validation dataset are independently selected without any overlapping. MLPs with the 1-16-1 architecture are applied to all the training sessions for noiseless or noisy data.

Fine Features A smooth function with two fine features as spikes is defined by

$$y = f(x) = g\left(x, \frac{1}{6}, \frac{1}{2}, \frac{1}{6}\right) + g\left(x, \frac{1}{64}, \frac{1}{4}, \frac{1}{128}\right) + g\left(x, \frac{1}{64}, \frac{11}{20}, \frac{1}{128}\right) \tag{4.2}$$

where $x \in X = [0, 1]$ and g is defined as

$$g\left(x, \alpha, \mu, \sigma\right) = \frac{\alpha}{\sqrt{2\pi}\sigma} \cos\left(\frac{(x-\mu)\pi}{\sigma}\right) \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{4.3}$$

For the training data, input values x_k are selected by randomly sampling 2,000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (4.2). Then, a training dataset with 2,000 (x_k, y_k) samples is obtained. For the validation data, input values x_k are selected by randomly sampling 1,000 different numbers from a uniform distribution on X, and corresponding output values y_k are computed by Eq. (4.2). Then, a validation dataset with 1,000 (x_k, y_k) samples is obtained. Here, the training dataset and the validation dataset are independently selected without any overlapping. MLPs with the 1-15-1 architecture are applied to all the training sessions for noiseless or noisy data.

Under-sampled Segments A smooth function with two under-sampled segments is defined by

$$y = f(x) = g\left(x, \frac{1}{5}, \frac{1}{4}, \frac{1}{12}\right) + g\left(x, \frac{1}{5}, \frac{3}{4}, \frac{1}{12}\right) + g\left(x, \frac{1}{64}, \frac{5}{4}, \frac{1}{12}\right) \tag{4.4}$$

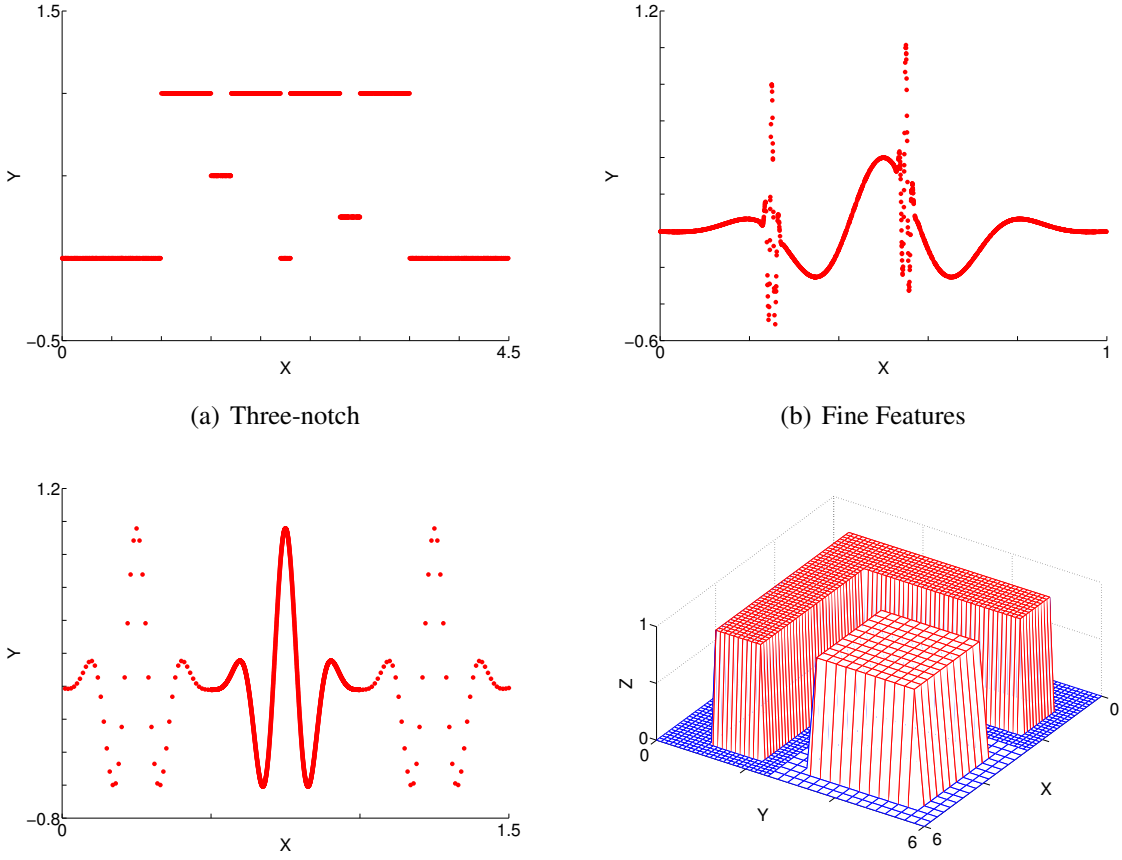
where $x \in X = [0, 1.5]$ and g is defined in Eq. (4.3). For the training data, input values x_k are collected by using 50 grid points from a uniform grid on $[0, 0.5]$, 50 grid points from a uniform grid on $[1.0, 1.5]$, and 2,000 grid points from a uniform grid on $(0.5, 1.0)$. Corresponding output values y_k are computed by Eq. (4.4). These values form a training dataset with 2,100 (x_k, y_k) input/output samples. For the validation data, input values x_k are collected by using 25 grid points from a uniform grid on $[0, 0.5]$, 25 grid points from a uniform grid on $[1.0, 1.5]$, and 1,000 grid points from a uniform grid on $(0.5, 1.0)$. Corresponding output values y_k are computed by Eq. (4.4).

These values form a validation dataset with 1,050 (x_k, y_k) input/output samples. Here, the training dataset and the validation dataset are independently selected without any overlapping. MLPs with the 1-12-1 architecture are applied to all the training sessions for noiseless or noisy data.

Under-sampled Square A function existed in the three-dimensional space, which has a letter 'L' shape and an under-sampled square raised from a plane, is defined on $[0, 6] \times [0, 6]$ by

$$z = f(x, y) = \begin{cases} 1 & \text{if } x \in [1.0, 5.5] \text{ and } y \in [1.0, 2.0] \\ 1 & \text{if } x \in [1.0, 2.0] \text{ and } y \in [2.0, 5.5] \\ 1 & \text{if } x \in [3.0, 5.5] \text{ and } y \in [3.0, 5.5] \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

In the training data, input values x_k and y_k are the 289 grid points from the uniform grid on $(2.5, 6] \times (2.5, 6]$ and 2,522 grid points from the uniform grid on $[0, 6] \times [0, 6] - (2.5, 6] \times (2.5, 6]$. Corresponding output values z_k are computed by Eq. (4.5). These values form a training dataset with 2,811 (x_k, y_k) samples. In the validation data, input values x_k and y_k are the 144 grid points from the uniform grid on $(2.5, 6] \times (2.5, 6]$ and 1,261 grid points from the uniform grid on $[0, 6] \times [0, 6] - (2.5, 6] \times (2.5, 6]$. Corresponding output values z_k are computed by Eq. (4.5). These values form a validation dataset with 1,405 (x_k, y_k) samples. Here, the training dataset and the validation dataset are independently selected without any overlapping. MLPs with the 2-9-3-1 architecture are applied to all the training sessions for noiseless or noisy data.



(a) Three-notch

(b) Fine Features

(c) Under-sampled Segments

(d) Under-sampled Square

FIG. 4.1. Target functions of four function approximation examples with designed non-global local minima. Numbers on horizontal and vertical axes in each subfigure represent the input and output of the function, respectively. From Fig. 4.1(a) to Fig. 4.1(c), red dots denote to the target training data. In Fig. 4.1(d), different colors (blue or red) are used to distinguish different output values (0 or 1) of the function on vertical axes.

Results and Discussion (Part I) In this section, we choose the three-notch function approximation with noiseless data as an example to demonstrate performances of the NRAE-based training methods over all selected sets of initial weights and λ values. Experimental results we obtained are presented as follows:

1. Learning curves of the NRAE training sessions as comparing to those of the MSE training sessions are presented in Fig. 4.2. It illustrates that the NRAE training method with a sufficiently large λ greatly outperforms the MSE training method and actually reaches a global or near global minimum, because the approximation error is nearly zero. Since the three-notch target function is designed to have non-global local minima, the observed learning curves in Fig. 4.2 indicate that training with the NRAE criterion under a sufficiently large λ has the capability to avoid the non-global local minimum, while training with the MSE criterion does not. Note that the MSE training quickly falls into a non-global local minimum with an MSE about 6×10^{-2} from the beginning to around 10,000 training epochs, while the NRAE training at each value of λ outperforms the MSE training after a certain number of training epochs, moving rather quickly toward a global or near global minimum where the MSE is in the range from 10^{-5} to 10^{-9}.

2. In our experimental trials, some training sessions with the NRAE criterion fails to reach a global or near global minimum. Table 4.1 presents all test results of the NRAE and NRAE-MSE training method with different values of λ under different sets of initial weights. It shows that in several training sessions when $\lambda = 10^6, 10^7, 10^{10}$ and 10^{11}, the NRAE training is unable to perform better than the MSE training. The reason can be concluded as the following: the smaller λ is, the more non-global local minima exist; the larger λ is, the greater convexity region with non-global local minima expands; when λ increases to an extremely large mag-

nitude, the error space of $C_\lambda(w)$ may become "flat", which slows down the NRAE training session thus affecting the performance in achieving the global or near-global minimum. More details related to this observation will be systematically discussed in the last section of this chapter.

3. We stress that the NRAE-MSE training consistently outperforms the MSE training in all trials with different sets of initial weights and with all tested values of λ. It indicates that the NRAE-MSE training method, which switches the training criterion between NRAE and MSE, is a significant improvement over the NRAE training method. In fact, the NRAE-MSE training method can be applied to replace the NRAE training method in many practical situations, with maintaining at least the same performance as the NRAE training method.

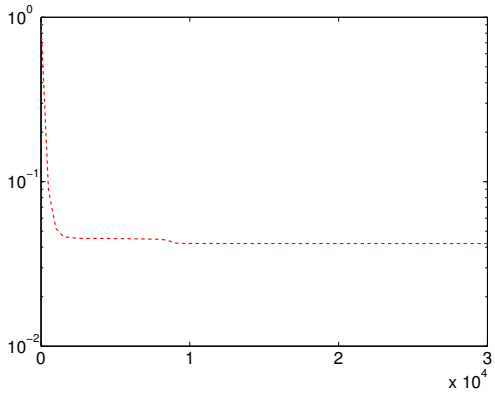
4. It is noticed that the training errors of NRAE-MSE for the same λ are distinct for different sets of initial weights as shown in Table 4.1. To explain that, three remarks are concluded as follows. First, there are many global minima located in "attraction basins" of different error spaces on $C_\lambda(w)$, while the NRAE-MSE training stops whenever a satisfactory MSE is reached or the MSE for the validation data starts to indicate overfitting. Therefore, the training sessions starting from different sets of initial weights travel through different paths toward different global minima and may be stopped at distinct points with different training errors. Second, it has been proven in Theorem 2 that the convexity region of $C_\lambda(w)$ in the error space with no non-global local minimum expands as λ increases, thereby avoiding the non-global local minimum. However, reaching the exactly global minimum is not theoretically guaranteed. At last, the observation might show that the NRAE-MSE training method is only able to reach a near global minimum. However, because the achieved training errors of NRAE-MSE for the same λ are all virtually zero, it is more likely that the

differences among them reflect numerical sensitivity of the algorithm.

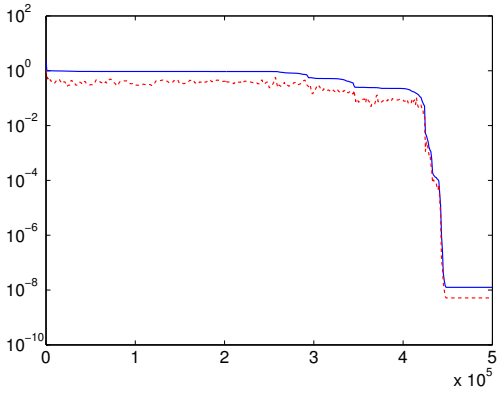
5. Without the MSE excursions involved, the NRAE-MSE training is simply an NRAE training, thus the performance of the NRAE-MSE training is at least as good as the NRAE training. As we discussed above, the training sessions starting from different sets of initial weights may be stopped at different points of global minima with different numbers of training epochs. In Table 4.1, some NRAE-MSE training errors are larger than the corresponding NRAE training errors, because the total number of epochs of the NRAE training in the NRAE-MSE training method is chosen as 10^4, which is smaller than the total number of training epochs 10^6 completed by the NRAE training method in our experiments. Nevertheless, with the same number of NRAE training epochs, the NRAE-MSE training errors are still lower than their corresponding MSE training errors for all randomly selected sets of initial weights and all tested values of λ.

Table 4.1. MSEs of the three-notch function approximation with noiseless data achieved by the MSE, the NRAE, and the NRAE-MSE training methods. The MSE training results are shown as a baseline. The best performances achieved by the NRAE training method are highlighted by underlines, while the best performances achieved by the NRAE-MSE training method are highlighted in bold.

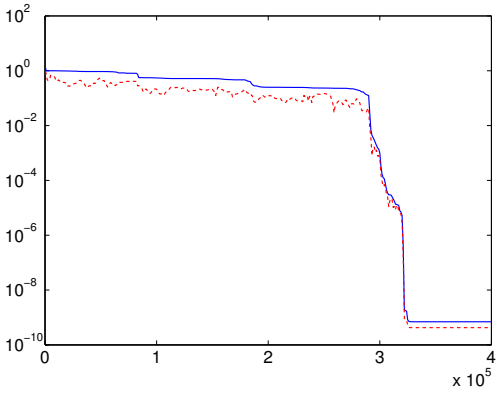
Three-notch Function Approximation		Set of Initial Weights									
		1	2	3	4	5	6	7	8	9	10
MSE		1.05×10^{-1}	1.94×10^{-2}	1.11×10^{-4}	1.35×10^{-2}	1.46×10^{-1}	1.01×10^{-1}	7.54×10^{-4}	5.42×10^{-3}	5.41×10^{-3}	8.97×10^{-2}
$\lambda = 10^6$	NRAE	7.14×10^{-8}	8.04×10^{-2}	4.55×10^{-2}	8.36×10^{-7}	9.25×10^{-1}	8.25×10^{-3}	9.36×10^{-8}	7.16×10^{-8}	1.43×10^{-2}	$\underline{5.23 \times 10^{-9}}$
	NRAE-MSE	1.47×10^{-5}	3.68×10^{-4}	1.99×10^{-5}	3.25×10^{-4}	2.54×10^{-4}	1.62×10^{-4}	1.24×10^{-4}	9.27×10^{-5}	6.83×10^{-4}	4.85×10^{-4}
$\lambda = 10^7$	NRAE	4.25×10^{-8}	2.86×10^{-2}	3.86×10^{-2}	9.14×10^{-4}	8.04×10^{-7}	4.27×10^{-2}	$\underline{8.01 \times 10^{-9}}$	9.02×10^{-8}	9.72×10^{-3}	7.08×10^{-7}
	NRAE-MSE	1.35×10^{-5}	1.57×10^{-5}	1.58×10^{-5}	2.48×10^{-5}	2.06×10^{-4}	1.54×10^{-5}	7.96×10^{-5}	5.05×10^{-5}	2.75×10^{-4}	3.99×10^{-4}
$\lambda = 10^8$	NRAE	$\underline{4.73 \times 10^{-9}}$	8.48×10^{-8}	$\underline{1.95 \times 10^{-8}}$	$\underline{1.53 \times 10^{-7}}$	9.54×10^{-7}	$\underline{7.64 \times 10^{-7}}$	5.26×10^{-7}	2.89×10^{-8}	1.74×10^{-7}	6.54×10^{-7}
	NRAE-MSE	4.75×10^{-6}	5.99×10^{-8}	9.00×10^{-9}	7.57×10^{-7}	5.56×10^{-7}	1.69×10^{-6}	7.29×10^{-8}	2.26×10^{-8}	6.74×10^{-8}	9.54×10^{-6}
$\lambda = 10^9$	NRAE	3.52×10^{-8}	$\underline{8.79 \times 10^{-9}}$	4.64×10^{-7}	7.23×10^{-7}	$\underline{5.54 \times 10^{-8}}$	8.66×10^{-7}	8.80×10^{-9}	$\underline{4.52 \times 10^{-9}}$	$\underline{7.37 \times 10^{-8}}$	9.36×10^{-7}
	NRAE-MSE	1.86×10^{-7}	6.66×10^{-8}	1.34×10^{-10}	4.83×10^{-8}	4.46×10^{-7}	4.63×10^{-7}	$\mathbf{1.63 \times 10^{-8}}$	9.35×10^{-9}	4.05×10^{-8}	1.35×10^{-7}
$\lambda = 10^{10}$	NRAE	8.43×10^{-8}	1.88×10^{-2}	8.78×10^{-3}	3.65×10^{-6}	1.17×10^{-6}	3.76×10^{-5}	2.01×10^{-3}	7.94×10^{-5}	5.68×10^{-7}	1.94×10^{-6}
	NRAE-MSE	7.33×10^{-8}	$\mathbf{1.96 \times 10^{-8}}$	9.34×10^{-11}	1.24×10^{-8}	7.53×10^{-8}	2.63×10^{-7}	1.74×10^{-8}	4.23×10^{-9}	1.72×10^{-8}	$\mathbf{3.78 \times 10^{-8}}$
$\lambda = 10^{11}$	NRAE	2.00×10^{-6}	6.07×10^{-6}	1.87×10^{-3}	2.16×10^{-4}	3.53×10^{-5}	9.85×10^{-2}	2.02×10^{-5}	4.39×10^{-5}	1.28×10^{-5}	1.27×10^{-7}
	NRAE-MSE	$\mathbf{2.57 \times 10^{-8}}$	2.16×10^{-8}	$\mathbf{5.50 \times 10^{-11}}$	$\mathbf{7.24 \times 10^{-9}}$	$\mathbf{2.54 \times 10^{-8}}$	$\mathbf{1.12 \times 10^{-7}}$	3.24×10^{-8}	$\mathbf{3.21 \times 10^{-9}}$	$\mathbf{9.63 \times 10^{-9}}$	7.78×10^{-8}



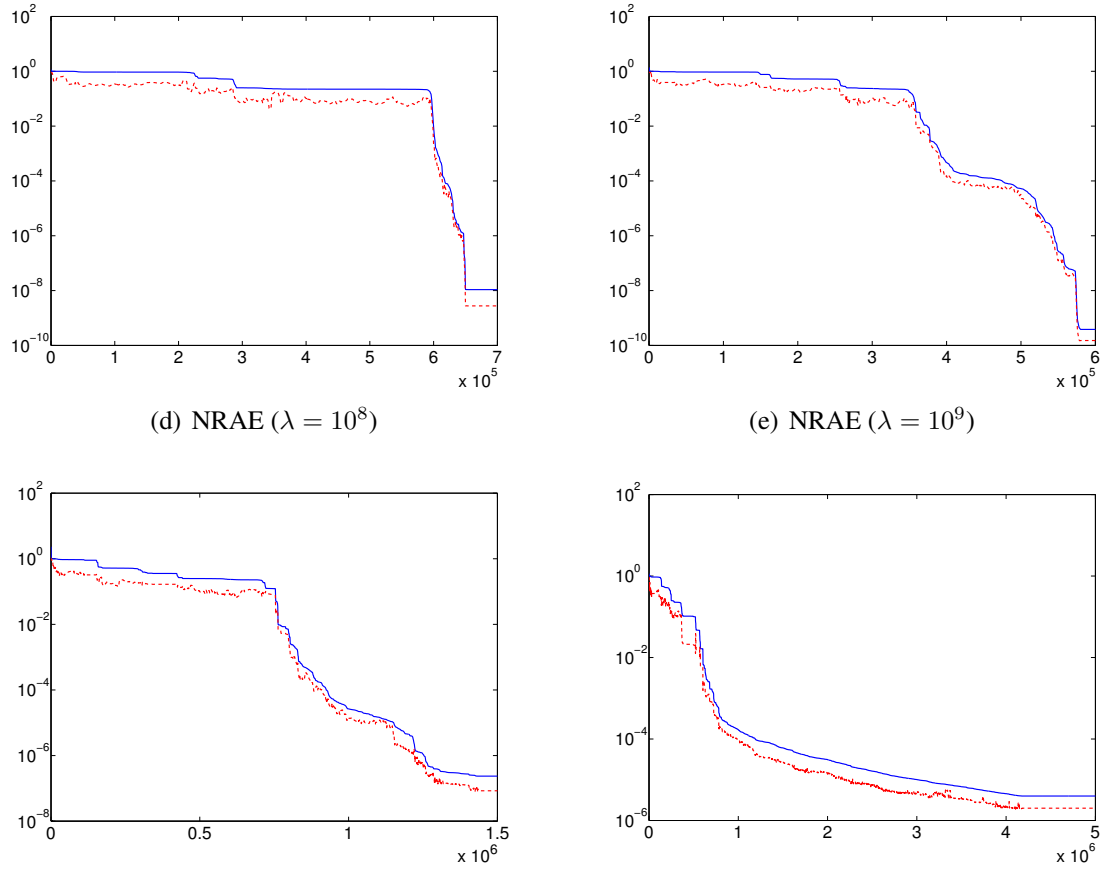
(a) MSE



(b) NRAE ($\lambda = 10^6$)



(c) NRAE ($\lambda = 10^7$)



(d) NRAE ($\lambda = 10^8$)

(e) NRAE ($\lambda = 10^9$)

(f) NRAE ($\lambda = 10^{10}$)

(g) NRAE ($\lambda = 10^{11}$)

FIG. 4.2. Learning curves of the three-notch function approximation achieved by the MSE and NRAE training. In Fig. 4.2(a), red dash lines represent the MSE training. In Fig. 4.2(b) - Fig. 4.2(g), blue solid lines represent the NRAE training and red dash lines are the corresponding curves respect to MSE values. Numbers on horizontal axes are numbers of training epochs. Numbers on vertical axes are values of training errors which are converted to the logarithmic numbers with respect to base 10. All training sessions are converged at the end of the shown curves.

Results and Discussion (Part II) In this section, we choose $\lambda = 10^6$ for all the function approximation examples to compare the NRAE-MSE training method to the standard MSE training method over all randomly selected sets of initial weights with noiseless or noisy data. Experimental results we obtained are presented as follows:

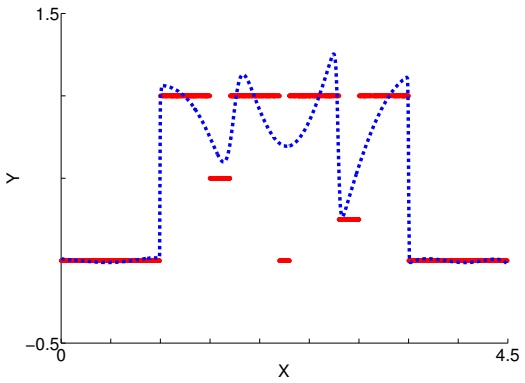
1. In Fig. 4.3, the NRAE-MSE training captures all the significant features and under-sampled segments of the target functions in our experiments with noiseless or noisy data, but the MSE training misses all the fine features and under-sampled segments of those target functions. Furthermore, the results obtained from the noisy data indicate that the NRAE-MSE training method is able to deal with a high level of noises in the training data. In spite of the noise, the MLP resulting from the NRAE-MSE training still captures the fine features and under-sampled segments of the target functions and maintains a high generalization level at the same time.

2. Fig. 4.4 demonstrates that the MLPs obtained by the NRAE-MSE training method outperform all their corresponding MLPs obtained by the MSE training method, without regarding to different selections of initial weights for both the noiseless and noisy data. It indicates that the performance of the NRAE-MSE training method is less sensitive to the selection of initial weights comparing to the MSE training method, while multiple trials for selecting the best resultant MLP is unnecessary in the NRAE-MSE training.

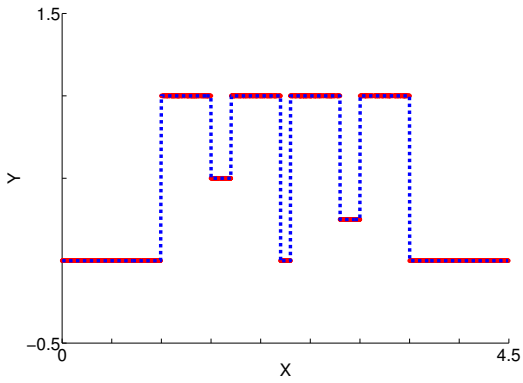
3. Some results shown in Fig. 4.4 imply that the noise added to the training data can help the MSE training method in approximating functions. We explain this observation over here using the function approximation of the under-sampled square as an example. In training with the noiseless data, the target value y_k on the under-sampled square is 1. If the platform on the under-sampled square is missing in the MLP under training, the sum squared error over the under-sampled square is

$\sum_{k=1}^{289} \left\| y_k - \hat{f}\left(x_k, w\right) \right\|^2 = \sum_{k=1}^{289} \|1-0\|^2 = 289$ in the noiseless training. Note that 289 is the number of grid points on the under-sampled square. On the other hand, in training with the noisy data, the new target value y_k on the under-sampled square is $1+\varepsilon_i$. Hence, the SSE over the under-sampled square is $\sum_{k=1}^{289} \left\| y_k - \hat{f}\left(x_k, w\right) \right\|^2$ $= \sum_{k=1}^{289} \|1+\varepsilon_i-0\|^2 \approx 289E\left[\|1+\varepsilon_i\|^2\right]$, where E denotes the expected value. It can be proven that if ε_i are drawn from independent identically distributed normal distributions $N\left(0,\sigma\right)$, the greater the standard deviation σ, the greater the difference $E\left[\|1+\varepsilon_i\|^2\right] - 1$. When σ is large enough, $289E\left[\|1+\varepsilon_i\|^2\right]$ is sufficiently large with a high probability to compensate for the lack of samples on the square and pull the MLP under training from 0 over the under-sampled square toward 1. The platform over the square is then "found" by and incorporated into the MLP, thus the performance of the MLP trained by the MSE method is improved.

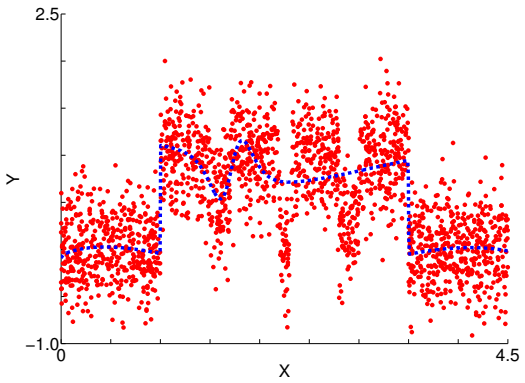
4. In Fig. 4.4(d), the MLP trained by the MSE method with the noisy data outperforms the MLP trained by the NRAE-MSE method with the noiseless data except the 5th and 6th set of initial weights. As we exemplified above, adding noises can help the MSE training method to find the platform over the under-sampled square. Therefore, in the MSE excursions of the NRAE-MSE training method, the MLP achieved on the noiseless data is inferior to the MLP produced by the MSE training method with the noisy data. This explains why the MLP trained by the MSE method with the noisy data can outperform the MLP trained by the NRAE-MSE method with the noiseless data. In addition, it also explains why the MLP trained by the NRAE-MSE method with the noisy data outperforms the MLP trained by the same method with the noiseless data.



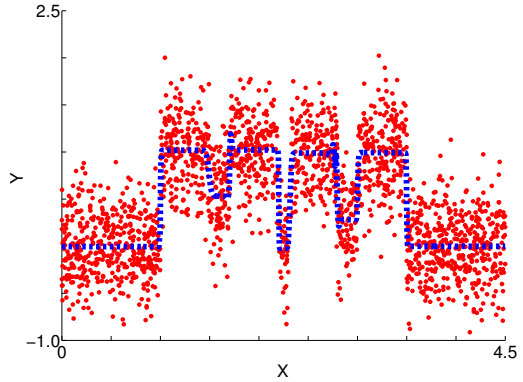
(a) MSE Training



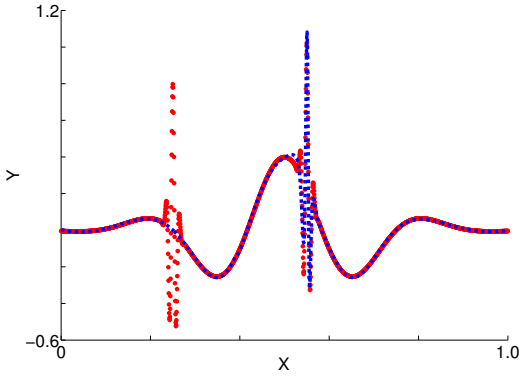
(b) NRAE-MSE Training



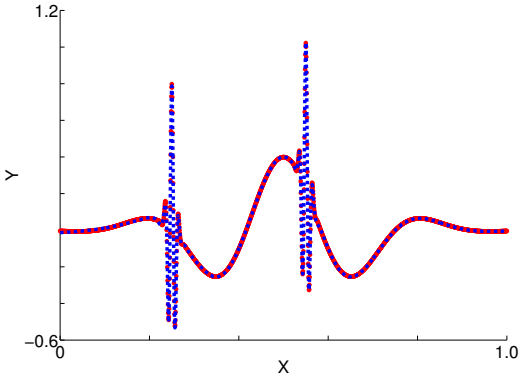
(c) MSE Training with Noises



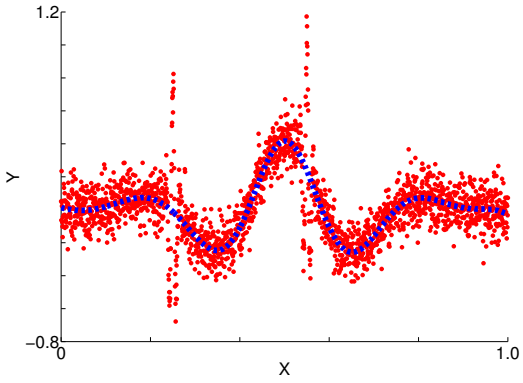
(d) NRAE-MSE Training with Noises



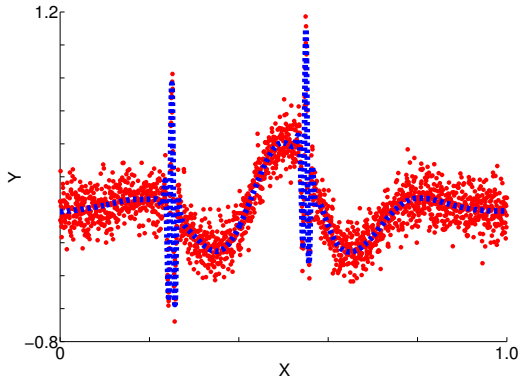
(e) MSE Training



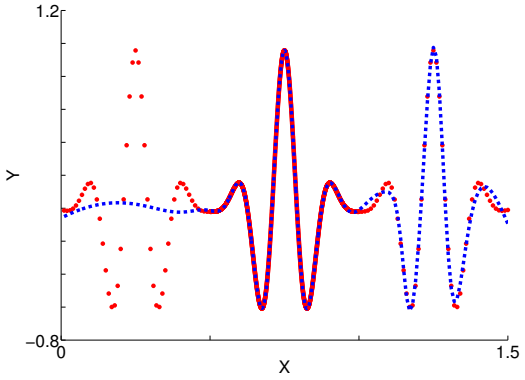
(f) NRAE-MSE Training



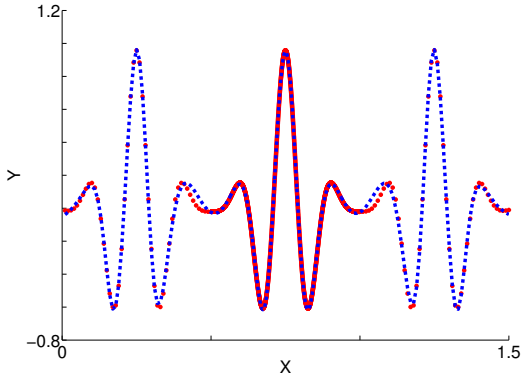
(g) MSE Training with Noises



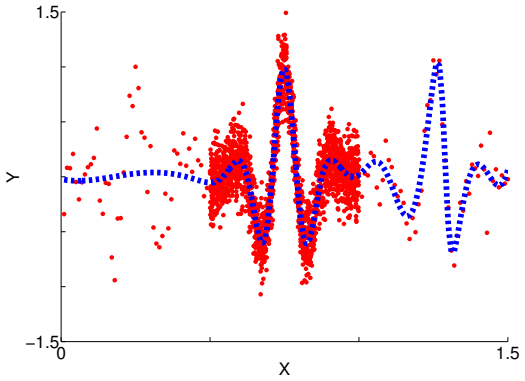
(h) NRAE-MSE Training with Noises



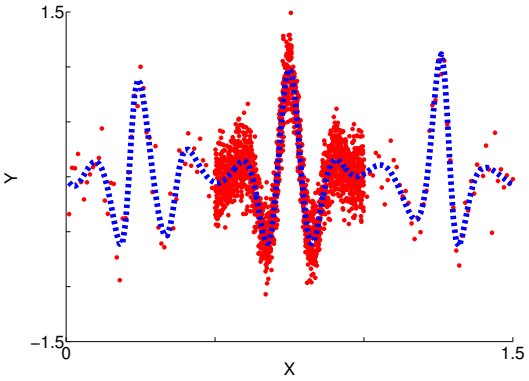
(i) MSE Training



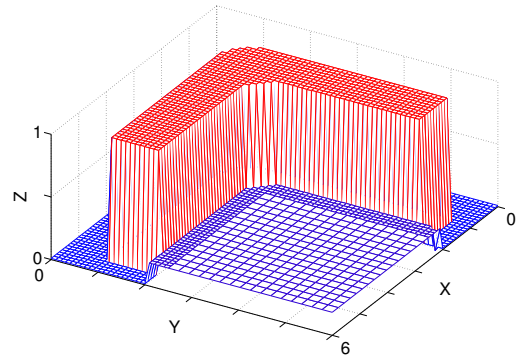
(j) NRAE-MSE Training



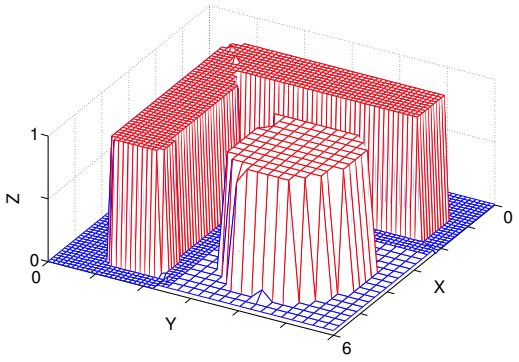
(k) MSE Training with Noises



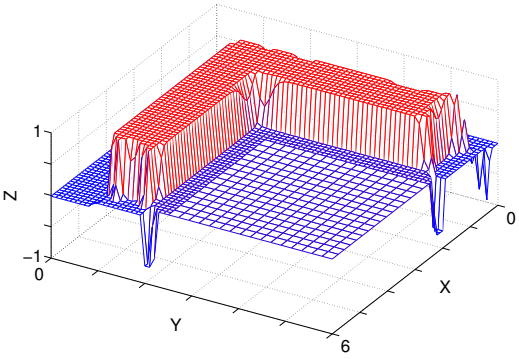
(l) NRAE-MSE Training with Noises



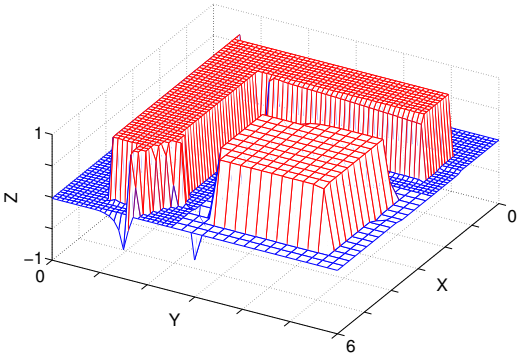
(m) MSE Training



(n) NRAE-MSE Training

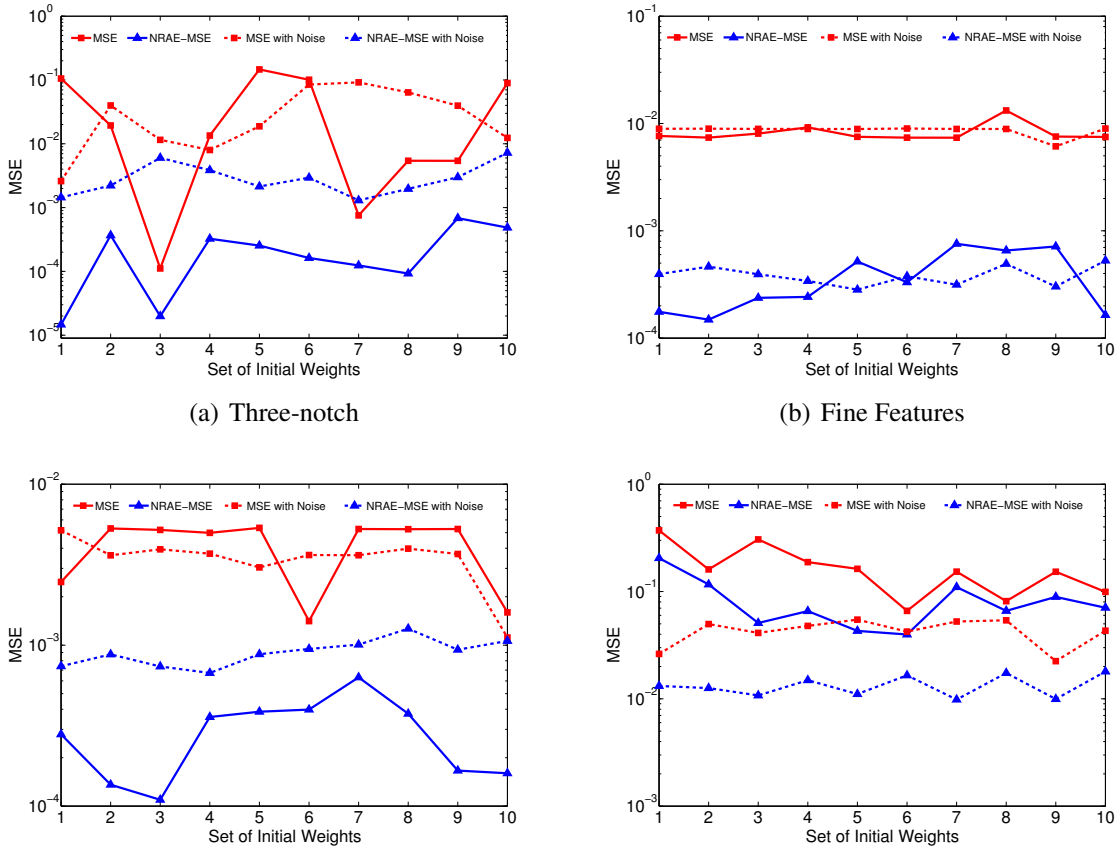


(o) MSE Training with Noises



(p) NRAE-MSE Training with Noises

FIG. 4.3. Fitting plots of four function approximation examples achieved by the MSE and NRAE-MSE training. For each example, the MSE and NRAE-MSE training method use the same set of initial weights. All NRAE-MSE training sessions presented here are performed with $\lambda = 10^6$. Numbers on horizontal and vertical axes in each subfigure represent the input and output of the function, respectively. In Fig. 4.3(a) - Fig. 4.3(l), red dots denote target training samples, and blue dash lines are MLP approximated function plots. In Fig. 4.3(m) - Fig. 4.3(p), only MLP approximated function plots are shown by using blue and red colors to distinguish different output values of the functions on vertical axes.



(a) Three-notch

(b) Fine Features

(c) Under-sampled Segments

(d) Under-sampled Square

FIG. 4.4. Training errors of 10 sets of initial weights for four function approximation examples achieved by the MSE and NRAE-MSE training. All NRAE-MSE training sessions presented here are performed with $\lambda = 10^6$. Colors and symbols denote different training methods of the MSE (red square) and the NRAE-MSE (blue triangle). Solid and dash lines represent different training sessions with noiseless and noisy data, respectively. In order to clearly show differences between MSE values obtained by the MSE and NRAE-MSE training, actual numbers in all vertical axes are converted to logarithmic numbers with respect to base 10.

4.3.2 Handwritten Digit Recognition

To test the capability of the NRAE criterion in training a large MLP with a real-world dataset, the NRAE-MSE training method is applied to train the MLP for recognizing handwritten digits with the MNIST dataset [64]. The MNIST dataset is commonly used as a benchmark to compare performances of different classifiers, including many MLP-based classifiers. The MNIST dataset contains 60,000 training samples and 10,000 test samples of handwritten digits. Each sample has 784 features which are obtained from a 28×28 black and white image. Each feature value is the anti-aliasing normalized gray level of the corresponding pixel in an image.

In our experiments, we test both the transformed and the original MNIST dataset. For the transformed MNIST dataset, we use the principle component analysis (PCA) to reduce the dimension of the original image from 784 to 40 (principle components), and then apply both the standard MSE training method and the NRAE-MSE training method to train a 2-layer MLP with the architecture of 40-300-10. Each of the ten output nodes is associated with one of the ten numerals, i.e., 1, 2, ..., 9, and 0. For the original MNIST dataset, we perform experiments on a 2-layer MLP with the architecture of 784-300-10. For training the MLPs, trainable weights are randomly selected from a uniform distribution between $-2.4/F_i$ and $2.4/F_i$, where F_i is the fan-in of the i-th node as we defined before. We initialized the weights in a small range close to 0 rather than $[-1, 1]$, because we do not want the initial weights to be too large to saturate the hidden nodes of MLP, especially when the MLP architecture is chosen very large. All experiments apply 50,000 out of 60,000 training samples to train the MLP and employ the early-stopping rule to detect the overfitting with the left 10,000 training samples as the validation data during the training. Once the overfitting is detected, the training stops and the trained MLP is tested by 10,000 test samples.

For training the MLP, the target value of the output node is 1 if the input is the correct digit associated with the node, and is -1 otherwise. After the MLP is trained, the digit associated with the output node, which has the highest value among the ten output nodes, is selected as the result of recognizing the input image. The classification accuracy of the trained MLP is calculated by the percentage of images that are correctly recognized in the test dataset, and the test error rate is defined as 100% minus the classification accuracy of the trained MLP. For all training sessions performed with the MNIST dataset, we set the global learning rate and the momentum term of GD equal to 0.001 and 0.5, respectively. In all the NRAE-MSE training sessions, we set $L_{\text{NRAE}} = 500$ for the NRAE training and $L_{\text{MSE}} = 500$ for the MSE excursion. The maximum number of iterations of the NRAE-MSE training is limited to 10. Therefore, the maximum number of NRAE-MSE training epochs including both the NRAE training sessions and the MSE excursions is 10^4. We set $\lambda = 10^3, 10^4, 10^5$ and 10^6 to see the effect of different values of λ on the NRAE-MSE training.

Ten different sets of initial weights for each λ are employed in our experiments. Table 4.2 presents the best classification results obtained by the NRAE-MSE training method among all testing sessions as well as some benchmark results. The total number of epochs of the NRAE-MSE training in all of our experiments is less than 2,000. The experimental results demonstrate that the MLP classifier trained by the NRAE-MSE method has a better generalization capability than many benchmark classifiers on the MNIST dataset with or without the PCA applied. It is worth noting that, the benchmark results presented in Table 4.2 were achieved by the most basic classifiers, such as the linear classifier, the quadratic classifier, the K-nearest-neighbors classifier, and the 2-layer MLP classifier, but they do not represent the best reported benchmark results obtained on the MNIST dataset. They are only used to demonstrate that the NRAE-MSE training method has the generalization ability that is better than the basic classifiers, while such the ability is comparable to the

Table 4.2. Test error rates achieved by basic classifiers on the original and transformed MNIST dataset. The best performances of the NRAE-MSE training method are achieved when $\lambda = 10^6$ and are highlighted in bold.

MNIST Dataset	Training Method	Test Error Rate (%)
Original	Linear classifier [64]	12.00
	K-nearest-neighbors + Euclidean L2 [64]	5.00
	784-300-10 MLP + MSE [64]	4.70
	784-300-10 MLP + NRAE-MSE (this dissertation)	**4.58**
Transformed	Linear classifier + deskewing [64]	8.40
	40-300-10 MLP + MSE + PCA (this dissertation)	3.84
	Quadratic classifier + PCA [64]	3.30
	40-300-10 MLP + NRAE-MSE + PCA (this dissertation)	**2.79**

MSE training method.

It is also observed that the NRAE-MSE training method consistently achieves a lower test error rate comparing to the MSE training method under the same experimental settings. It confirms that the NRAE-MSE training method has no difficulty in scaling up for a complex real-world task, where such the method has the ability to avoid the non-global local minimum and maintain a high level of generalization as well. In addition, we stress that the NRAE-MSE training can provide satisfactory generalization results with using random sets of initial weights, thus requiring no multiple trials on initial weights.

4.4 Stagnant Problem

4.4.1 Background

Based on Theorem 2, an intuitive method to maximumly use the potential of the NRAE criterion in training MLPs is to choose an extremely large λ to significantly expand the convexity region of RAE that is contained in the convexity region of NRAE, thus

providing the best chance to the local search method for avoiding the non-global local minimum and achieving a global or near-global minimum in theory. Accordingly, we wonder whether λ is able to be chosen as a very large value in the NRAE criterion for providing the best capability of the convexification in avoiding the non-global local minimum, especially when the training task involves a complex and non-convex error space that needs to be significantly convexified in practice. This idea motivates us to explore the NRAE criterion under the tremendous expansion of the convexity region of RAE and evaluate the NRAE training with large λ values.

However, some previously experimental results presented in Table 4.1 indicate that, applying extremely large λ values could affect the NRAE training somehow, thus impeding the training to converge on a global or near-global minimum normally. To further confirm this observation in a simple and elegant way, we perform the NRAE training with a very large $\lambda = 10^{20}$ to train the XOR problem using the same training samples and the same MLP architecture as described in Chapter 3. We choose 10 different sets of initial weights to start 10 training sessions, while training results from those sessions are shown in Fig. 4.5(a). It is observed that 6 out of 10 training sessions converge on the global minimum with the training errors around 10^{-10}, but the other 4 out of 10 training sessions cannot converge and stuck with the training errors closed to 1.0.

Therefore, it is confirmed that applying a significantly large λ can impedes the performance of the NRAE training, and we name this specific phenomenon existed in the NRAE training as a stagnant problem. In this section, we study the stagnant problem in the NRAE training with the XOR example and explore a feasible remedy to overcome it.

4.4.2 Theoretical Analysis

To uncover the theoretical reason of the stagnant problem in applying the NRAE criterion with large λ values to train MLPs, let us first review Eq. (3.8) for $\lambda \to \infty$ as

$$\begin{aligned}\lim_{\lambda\to\infty} C_\lambda(w) &= \lim_{\lambda\to\infty} \frac{1}{\lambda} \ln\left(\frac{1}{K} e^{\lambda\|\varepsilon_M(w)\|^2} \sum_{k=1}^{K} \eta_k(w)\right) \\ &= \lim_{\lambda\to\infty} \frac{1}{\lambda} \ln\left(\frac{1}{K} \sum_{k=1}^{K} \eta_k(w)\right) + \|\varepsilon_M(w)\|^2 \\ &= \|\varepsilon_M(w)\|^2\end{aligned} \tag{4.6}$$

It implies that if λ is numerically chosen as a very large value (e.g. $\lambda = 10^{20}$), the NRAE criterion is close to the minimax criterion, while $C_\lambda(w)$ is close to $\|\varepsilon_M(w)\|^2$. Generally, in training the MLP with the minimax criterion, i.e., the NRAE criterion under a very large λ, $\|\varepsilon_M(w)\|^2$ will be frequently changed by cycling different M defined in Eq. (3.6) for all $k = 1, ..., K$, and $C_\lambda(w)$ will be gradually minimized as $\|\varepsilon_M(w)\|^2$ decreases until a global or near-global minimum is achieved.

However, the solution of minimax generally tends to be an equilibrium rather an optimum. If the minimax criterion produces an equilibrium solution where M only covers K' but not K training samples where $K' < K$, then $\|\varepsilon_M(w)\|^2$ will be computed merely according to K' repeated values of M, and the entire MLP will be used to approximate the number of training samples decided by K' instead of K. In this case, the MLP could be redundant for such an approximation. Because the MLP is chosen to learn K training samples but it actually approximate K' training samples only, some trainable weights in the MLP will tend to become as similar as possible to each other, while some weights will be led to cancel effects of others during the training. As a consequence, the redundancy existed in trainable weights of the MLP causes linear dependencies of rows or columns in Jacobian $D_K(w)$, which is defined in Theorem 1, thereby generating the rank deficiency of

$D_K(w)$. Recall that a rank deficiency of Jacobian occurs when the number of linearly independent columns or rows in Jacobian is less than $\min(K, N)$, where K is the total number of training samples and N is the total number of trainable weights in the MLP.

According to Theorem 2, the NRAE criterion shares the same local and global optima with the RAE criterion only if the convexity of RAE is maintained, which requires the full rank of $D_K(w)$. If such a rank condition cannot be satisfied, the NRAE training may still be impeded by the non-global local minima, because the convexity of RAE cannot be maintained even through the convexity region of RAE is expected to be greatly expanded by a large λ. In other words, without the full rank of $D_K(w)$, the RAE criterion could yield a solution that exists in the complement set $(\cup_{\lambda>0}P)^c$, which is contained in the intersection I as defined in Theorem 1; as a result, the convexity region of RAE cannot be expanded monotonically as λ increases, thus it cannot be close to strictly convex even a very large λ is applied. Therefore, both the RAE and the NRAE criteria lose the benefit of the convexification in avoiding the non-global local minimum and have the risks of entering poor non-global local minima caused by the rank deficiency of $D_K(w)$.

Moreover, experimental evidences described in [94] pointed out that many training problems of MLPs are intrinsically ill conditioned, where Jacobian in those training problems are rank deficient. Any rank deficiency of Jacobian can cause BP to obtain only partial information of the possible search directions, potentially leading to non-global local minima with a slow convergence. Presumably, the stagnant problem is highly related to the rank deficiency of $D_K(w)$ in the NRAE training with large λ values. The four NRAE training sessions, which fail to reach the global minimum of the XOR problem with $\lambda = 10^{20}$, are possibly stuck on non-global local minima caused by the rank deficiency of $D_K(w)$. In other words, the rank deficiency of Jacobian could affect the ability of the NRAE training with large λ values in training MLPs for avoiding non-global local minima.

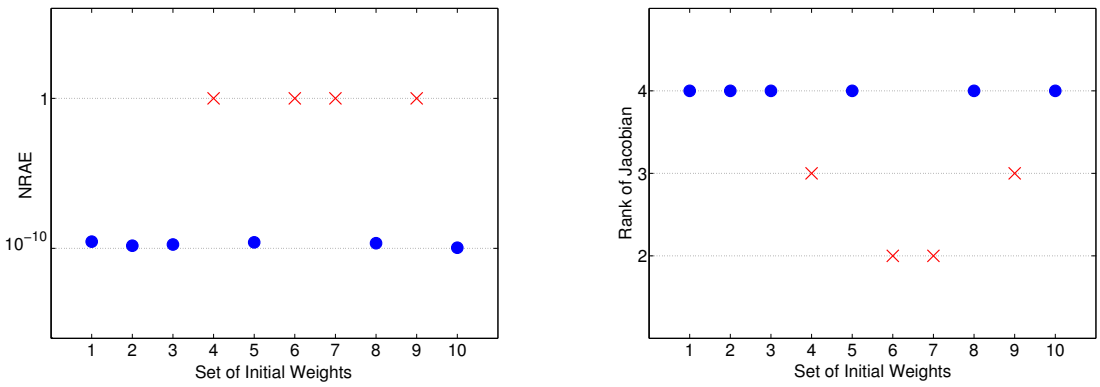
To verify our conjectures, we implement a rank detection method based on Gaussian

elimination (also known as row reduction in linear algebra) to monitor the rank of $D_K(w)$ and check whether the rank deficiency of Jacobian is occurred or not during the NRAE training. Because the rank of a matrix is equal to the number of pivots, i.e., the left-most non-zero entry of a row, in row echelon form of that matrix, our rank detection method applies a basic idea in linear algebra to calculate the rank of Jacobian in training MLPs as the following: compute $D_K(w)$ at first, then apply the Gaussian elimination to get a row echelon form of $D_K(w)$, and finally calculate the rank of $D_K(w)$ by counting the number of pivots in the row echelon form. With the rank detection method, we evaluate rank conditions of the ten NRAE training sessions with $\lambda = 10^{20}$ on the XOR problem and presented the results in Fig. 4.5(b).

Because the XOR problem has 4 training samples and we apply the 2-2-1 MLP with 9 trainable weights to perform the NRAE training, it is obviously that $K = 4$ and $N = 9$, thus $\min(K, N) = 4$. Experimental results in Fig. 4.5 illustrate that: for the NRAE training session that achieves the global minimum around the training error of 10^{-10}, $D_K(w)$ is always under a full rank condition where the rank of $D_K(w)$ equals to 4; for the NRAE training session that is stuck on non-global local minima with the training error close to 1.0, $D_K(w)$ has a rank deficiency where the rank of $D_K(w)$ is less than 4. It is a concrete evidence to confirm that the rank deficiency of $D_K(w)$ actually causes the stagnant problem and further results in the failure of the NRAE training with $\lambda = 10^{20}$ for achieving the global minimum of the XOR problem.

4.4.3 Experimental Study

To experimentally confirm our theoretical conjectures, we study the ten NRAE training sessions with $\lambda = 10^{20}$ performed on the XOR problem and evaluate corresponding experimental results under different circumstances in this section.

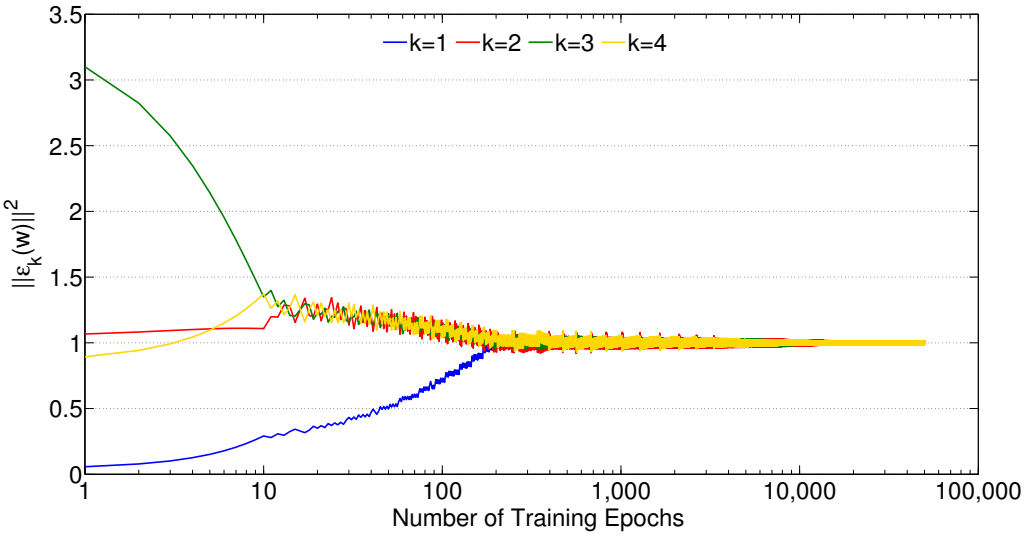


(a) Training Error (b) Rank Condition

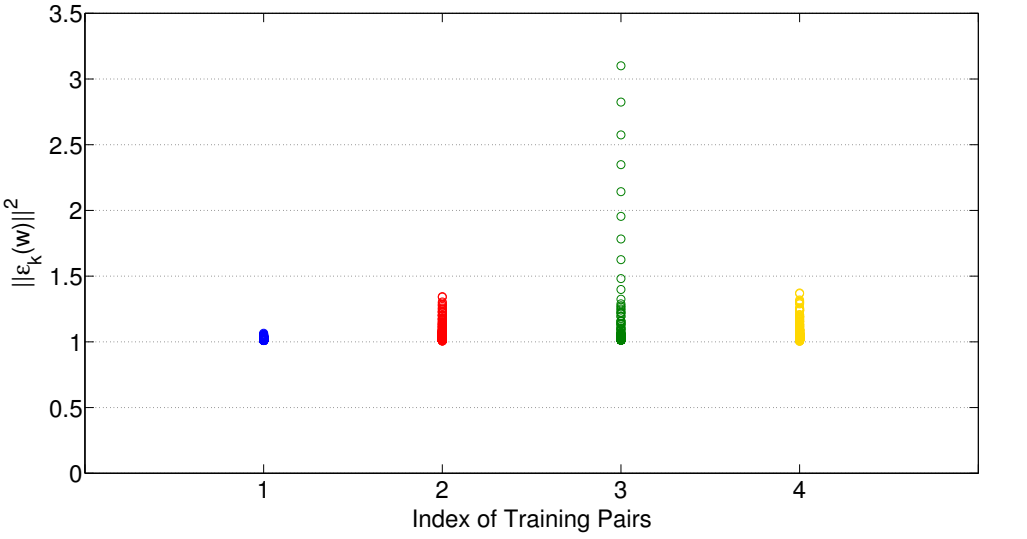
FIG. 4.5. Training errors and rank conditions of ten NRAE training sessions with $\lambda = 10^{20}$ for the XOR problem. Results are recorded corresponding to the convergence of the training sessions with 10 different sets of initial weights. Each blue dot represents the successfully converged training session, and each red cross indicates the training result involving the stagnant problem.

Failure NRAE Training We first study the four failure NRAE training results performed on the XOR problem, which use the 4th, 6th, 7th, and 9th set of initial weights as shown in Fig. 4.5. Here, we only discuss one result trained by the 4th set of initial weights with full details as the example.

For the mentioned NRAE training session on the XOR problem, we record the evolution of $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 3, 4$ with respect to the number of training epochs and the index of training samples separately as illustrated in Fig. 4.6. From the figure, we observe that the NRAE training merely focuses on approximating the training sample with $k = 3$ at the first 10 training epochs, thus dropping $\|\varepsilon_3(w)\|^2$ from the initial value near 3.0 down to a small value between 1.0 and 1.5. Meanwhile, the values of $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 4$ increase with different magnitudes from their initial values. It indicates that all trainable weights in the MLP are completely employed to fit the 3rd training sample at the beginning of the training, and such a fitting does not affect $\|\varepsilon_k(w)\|^2$ too much for $k = 1, 2, 4$. Thus,



(a)



(b)

FIG. 4.6. Evolution of $\|\varepsilon_k(w)\|^2$ in the NRAE training of the XOR problem using the 4th set of initial weights. Fig. 4.6(a) presents learning curves of $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 3, 4$ with respect to the number of training epochs. Fig. 4.6(b) indicates each value of $\|\varepsilon_k(w)\|^2$ corresponding to the index of training samples. Different colors represent different indices of XOR training samples in both figures.

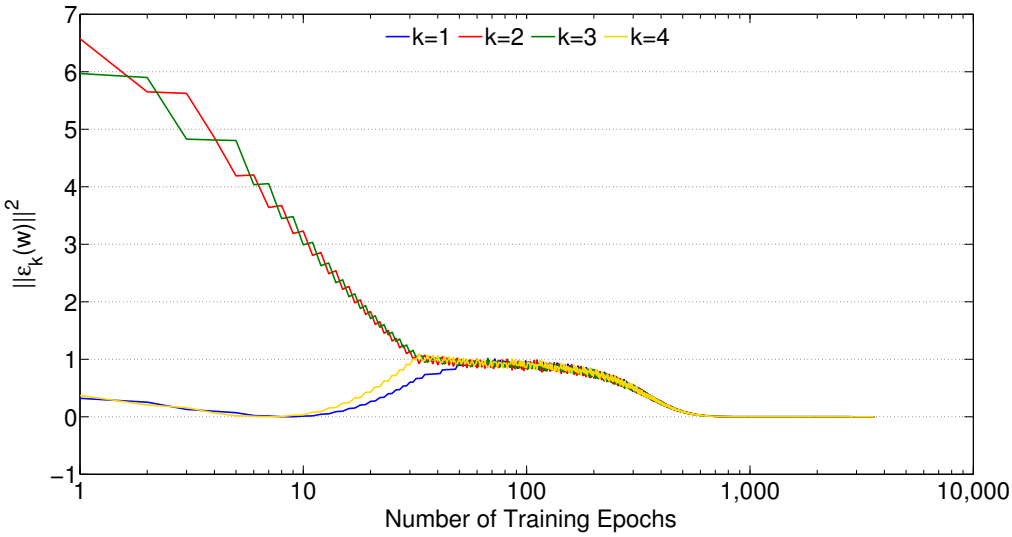
the redundancy in the MLP appears, and it leads to the rank deficiency of $D_K(w)$ at this moment. As a result, the NRAE training leads the searching path of the global minimum to enter the attraction basin of a non-global local minimum caused by the rank deficiency of Jacobian.

After that, although the MLP tries to resolve the rank deficiency through cycling the training samples with $k = 2, 3, 4$, it only brings minor fluctuations for fitting those training samples and gradually reduces $\|\varepsilon_k(w)\|^2$ for $k = 2, 3, 4$ down to 1.0. In this case, the NRAE training is unable to get away from the attraction basin of the mentioned non-global local minimum, and it leads to a continuous raising of $\|\varepsilon_1(w)\|^2$ because of the redundancy of weights. At last, the whole training is converged and stuck on the non-global local minimum with a training error close to 1.0.

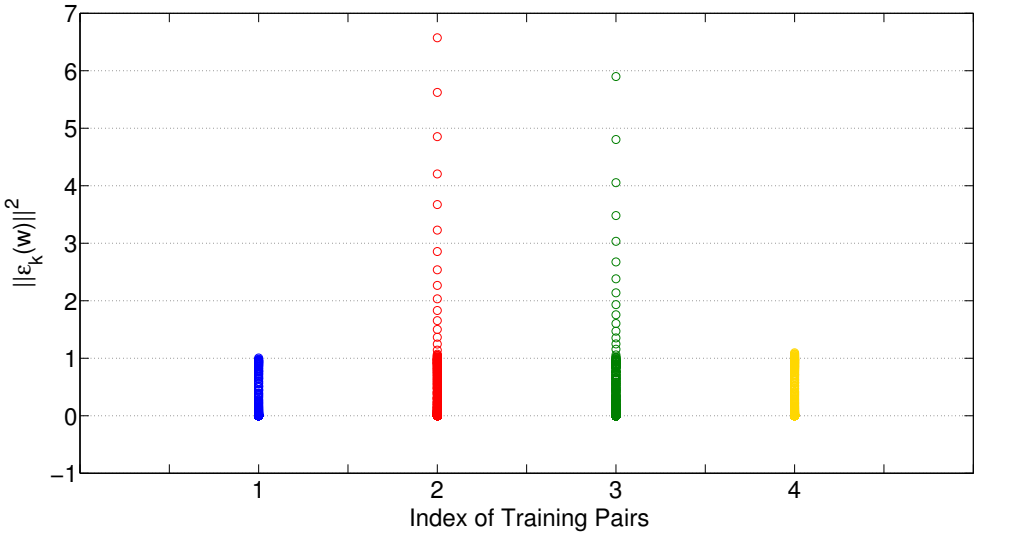
For the other failure NRAE training results trained by the 6th, the 7th, and the 9th set of initial weights, we perform the same analysis as we demonstrated for the 4th set of initial weights. We find out those failure NRAE training sessions always approximate the single XOR training sample at the beginning of the training, thus generating the rank deficiency of $D_K(w)$ and trapping the NRAE training into the non-global local minimum with the training error close to 1.0.

Success NRAE Training Comparing to the failure NRAE training results, we evaluate the six success NRAE training results, which use the 1st, 2nd, 3rd, 5th, 8th, and 10th set of initial weights as shown in Fig. 4.5. Here, we only discuss one result trained by the 1st set of initial weights with full details as the example.

For the mentioned NRAE training session on the XOR problem, we record the evolution of $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 3, 4$ with respect to the number of training epochs and the index of training samples separately as illustrated in Fig. 4.7. In contrast to the failure NRAE training results, the success NRAE training session approximates the training



(a)



(b)

FIG. 4.7. Evolution of $\|\varepsilon_k(w)\|^2$ in the NRAE training of the XOR problem using the 1st set of initial weights. Fig. 4.7(a) presents learning curves of $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 3, 4$ with respect to the number of training epochs. Fig. 4.7(b) indicates each value of $\|\varepsilon_k(w)\|^2$ corresponding to the index of training samples. Different colors represent different indices of XOR training samples in both figures.

samples with $k = 2, 3$ for almost 50 epochs at the beginning of the training. It reduces the training error by frequently switching between $\|\varepsilon_2(w)\|^2$ and $\|\varepsilon_3(w)\|^2$. After that, the MLP completely cycles through all training samples and gradually decreases $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 3, 4$ to a global minimum close to 0. Since the MLP does not focus on fitting a particular training sample, no rank deficiency of $D_K(w)$ appears, and the NRAE training successfully reaches the global minimum of the XOR problem. Based on our experiments, this phenomenon also explains other success NRAE training sessions.

Redundancy of MLP Weights Because the MLP applied to train the XOR problem has the 2-2-1 architecture including 9 free parameters composed by 6 trainable weights and 3 trainable biases, we analyze all NRAE training results through observing and comparing the exact values of these weights. In Table 4.3, we list all trained MLP weights with the same notations as presented in Fig. 3.1 after the convergence of the NRAE training sessions trained by 10 different sets of initial weights. Evidently, the redundancy of MLP weights caused by the rank deficiency of $D_K(w)$ can always be observed in the failure NRAE training sessions. For example, with the use of the 4th set of initial weights, w_{11}^1 and w_{12}^1 have the similar absolute value with different signs. In this case, if the input training samples are two values with the same sign, such like $(1, 1)$ or $(-1, -1)$, w_{11}^1 and w_{12}^1 cancel the effect with each other. Because of a very small b_1^1, it leads the result of the hidden node connected with w_{11}^1 and w_{12}^1 close to 0, thus the mentioned node together with its connected weights becomes redundant to the MLP. For another example, with the use of the 6th set of initial weights, the values of w_{11}^2 and w_{12}^2 are close to each other and very small. If the two hidden nodes provide the similar inputs with different signs to pass w_{11}^2 and w_{12}^2, then the calculation in the output node cancels the effect of these weights, which can also be considered as a redundancy.

Table 4.3. Values of trained MLP weights of ten NRAE training sessions after the convergence in the XOR problem. The failure NRAE training results are highlighted in bold.

Set of Initial Weights	Trainable Weights of the 2-2-1 MLP								
	w_{11}^1	w_{12}^1	w_{21}^1	w_{22}^1	w_{11}^2	w_{12}^2	b_1^1	b_2^1	b_1^2
1	−1.2720	1.3793	1.2069	−1.3021	1.2412	1.2436	−1.0429	−0.8859	1.4049
2	0.9629	1.0303	1.5613	1.7371	1.3954	−1.3196	3.7516	−1.5977	−1.3597
3	1.4614	−1.5513	−1.0982	1.1491	−1.2361	−1.2707	1.3453	0.6333	1.3821
4	**−0.0012**	**0.0012**	**−0.00011**	**0.00011**	**−0.0000058**	**0.000015**	**−0.00045**	**−0.0019**	**−0.0069**
5	−1.3226	1.3195	1.2243	−1.2216	−1.2186	−1.2296	1.0805	0.9117	1.4351
6	**−0.4612**	**−0.8291**	**−0.6689**	**0.1961**	**0.0000068**	**0.0000067**	**−0.1286**	**0.1703**	**0.0000069**
7	**0.0293**	**−0.2114**	**0.1211**	**−0.1967**	**0.0000086**	**0.0000086**	**−0.0567**	**0.5802**	**−0.0000087**
8	−1.3167	1.4431	1.1837	−1.2846	1.2488	1.2578	−1.0965	−0.8014	1.3816
9	**−0.0016**	**−0.0017**	**0.00084**	**−0.00084**	**−0.000039**	**−0.000014**	**0.0022**	**0.000087**	**−0.0069**
10	−1.3813	1.3920	1.2097	−1.2181	−1.1832	−1.2005	1.1755	0.8750	1.4109

Training Error on the Non-global Local Minimum In our experiments on the XOR problem, it is observed that all failure NRAE training sessions are always stuck on the non-global local minimum with the training error close to 1.0. An intuitive explanation is concluded by reviewing the trained MLP weights shown in Table 4.3. For each failure NRAE training session, because w_{11}^2, w_{12}^2, and b_1^2 are very small, the MLP outputs $\hat{f}\left(x_k, w\right)$ for $k = 1, 2, 3, 4$ with respect to the inputs x_k are very small. Since the target outputs of the XOR problem are either 1 or -1, the values of $\left\| y_k - \hat{f}\left(x_k, w\right) \right\|^2$ for $k = 1, 2, 3, 4$ are very close to 1.0. Therefore, based on Eq. (3.1), the corresponding NRAE value is close to 1.0 when $\lambda = 10^{20}$. However, it is unclear whether our experimental results represent a general case of the non-global local minimum caused by the rank deficiency of $D_K(w)$ on the XOR

problem. Fortunately, the XOR problem is so famous that the local minimum problem of XOR has already been discussed and analyzed comprehensively in many literatures. In [9, 108], experimental evidences indicate that if a local minimum of XOR is caused by exactly learning 1 out of 4 training sample with the 2-2-1 MLP, the MLP output for that training sample could be very small and close to 0. In this case, all partial derivatives with respect to the weights connected to the output node are close to 0. As a result, BP only propagates very small errors back to the hidden nodes where the small weights are unable to be updated effectively. This discussion explains our observations of the failure NRAE training results. Moreover, it further confirms that, if the NRAE training with a large λ tries to learn exactly one sample on the XOR problem, the rank deficiency of $D_K(w)$ will always occur and the training session will be stuck on the non-global local minimum with the training error close to 1.0.

4.4.4 Remedy

In order to solve the rank deficiency of Jacobian, a straightforward and effective way is to perform the perturbation, which randomly perturbs the variables corresponding to the linear dependent rows or columns in Jacobian until a full rank condition is satisfied. Based on this method, if the stagnant problem is detected during the NRAE training, the full rank condition of Jacobian can be restored by applying the perturbation to break linear dependencies between rows or columns in $D_K(w)$.

For validating the effectiveness of the perturbation method, we perform experiments with the same settings as the four failure NRAE training sessions on the XOR problem, and we randomly perturb all biases and weights that are smaller than 0.01 into $[-2.4/F_i, 2.4/F_i]$ for each MLP during the NRAE training. After that, we keep training the MLP and repeat the perturbation when the rank deficiency of $D_K(w)$ is detected until the NRAE training converges on the global minimum. The reason of merely perturbing the small biases and

weights of the MLP is that these small biases and weights in fitting the XOR are primarily caused by the rank deficiency of $D_K(w)$ according to what we observed in Table 4.3.

Fig. 4.8 presents the result after applying the perturbation to the failure NRAE training session with the 4th set of initial weights. It demonstrates that the perturbation breaks the cycling of training samples in the failure NRAE training session, thus eliminating the rank deficiency of $D_K(w)$ and leading the training to the global minimum. With applying the perturbation method, further experiments illustrate that all failure NRAE training sessions performed on the XOR problem are able to be improved by removing the rank deficiency of $D_K(w)$, and the training errors of these failure sessions can be reduced from 1.0 to 10^{-10}, which are similar to the success NRAE training sessions as we observed in Fig. 4.7. Therefore, our experimental results confirm that the stagnant problem is able to be resolved in the NRAE training with the large initial λ after the full rank condition of $D_K(w)$ is restored by the perturbation. Meanwhile, these experiments also indicate that maintaining the full rank condition of $D_K(w)$ in the NRAE training is significant to avoid the stagnant problem.

However, the computational efficiency of the perturbation method can be restricted if the training task is extremely complex, where a great number of training dataset (i.e., a large number of K) and a quite large MLP architecture (i.e., a large number of N) are required. Since the computational cost of $D_K(w)$ is directly proportional to the number of both K and N, evaluating the rank condition of $D_K(w)$ in each training epoch and perform the necessary perturbation to corresponding biases and weights could be tremendously expensive in computation. Therefore, it motives us to develop a more efficient and effective solution to handle the stagnant problem, and it will be comprehensively discussed in next chapter.

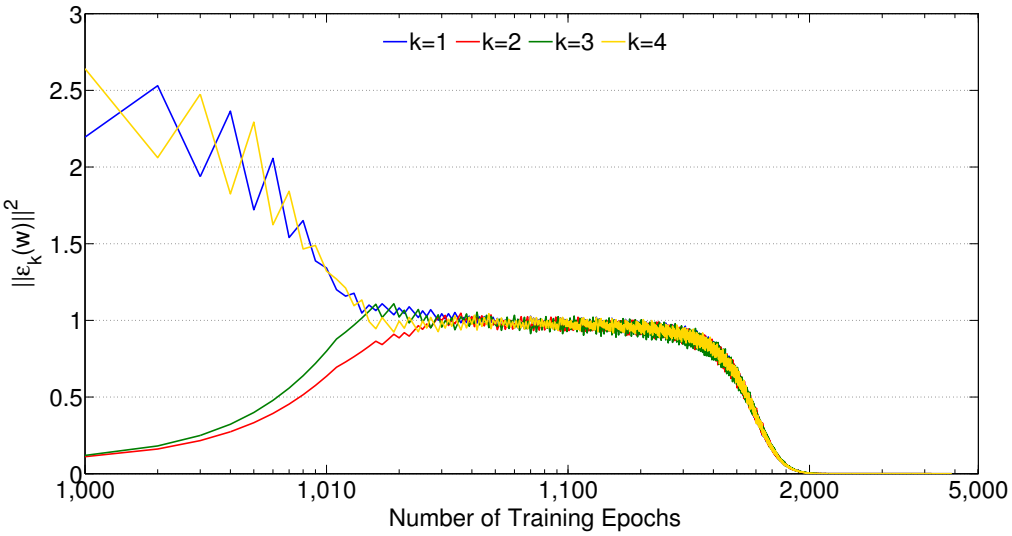


FIG. 4.8. Evolution of $\|\varepsilon_k(w)\|^2$ in the NRAE training of the XOR problem using the 4th set of initial weights after a perturbation. This figure presents learning curves of $\|\varepsilon_k(w)\|^2$ for $k = 1, 2, 3, 4$ with respect to the number of training epochs. The perturbation starts at the 1,000th training epoch of the previously failure NRAE training.

4.5 Summary

Insights to the properties of the NRAE criterion are gained in approximating functions by training MLPs with the NRAE training method at large values of λ. It is found that if λ is chosen in a properly working range, the NRAE training is able to achieve a much better performance than the standard MSE training in function approximation, where the tested functions contain fine features or under-sampled segments that specially designed to create undesirable non-global local minima of the MSE criterion. According to our experiments, those pre-designed non-global local minima are very hard for a training method to escape from, while most real-world tasks are not expected to be so "vicious". However, the effectiveness of the NRAE criterion in avoiding the non-global local minimum can be achieved in our experiments.

To perform the NRAE training with less computational costs, the NRAE-MSE training method is developed. It trains with the NRAE criterion at a large λ, but takes an excursion to train with the MSE criterion from time to time. Whenever the NRAE training brings the MLP weights within an "attraction basin" of a global or near-global minimum, the MSE excursion leads the training error close to zero. The NRAE-MSE training method succeeds every time in training MLPs over all trials with randomly selected sets of initial weights to approximate functions. Furthermore, the NRAE-MSE training method is applied to train the MLP for recognizing handwritten digits with the well-known MNIST dataset. The method outperforms basic benchmark methods without multiple trials of different sets of initial weights, indicating that the NRAE-MSE training can be scaled up to a complex real-world problem with a fairly good performance.

Although an intuitive approach to properly use the NRAE criterion in training MLPs is to apply the NRAE training method under a very large λ, a stagnant problem caused by the use of the large λ in the NRAE training is found. Accordingly, it impedes the applicability of the convexification methedology to avoid the non-global local minimum. Theoretical and experimental studies reveal the essence of the stagnant problem, which is connected to the notorious trouble of the rank deficiency of Jacobian in training MLPs. Concrete evidences indicate that the stagnant problem can be effectively fixed if the full rank condition of Jacobian in the NRAE training is maintained.

Chapter 5

DECONVEXIFICATION METHODOLOGY

According to the convexification methodology based on the NRAE criterion, the values of λ employed in both the NRAE and NRAE-MSE training method are primarily evaluated in the range 10^6-10^{11}. Presumably, for different training tasks with different datasets, different λ values are needed, thus selecting a fixed λ is neither practical nor efficient to be applied for performing various training tasks. Moreover, since the NRAE training has the difficulty of the stagnant problem as λ chooses very large, the best strategy to practically use the convexification methodology to perform the training is to select a range of the applicable λ rather than adopt a fixed large λ. Although the perturbation method can effectively solve the stagnant problem in the NRAE training, it could be limited by the tremendously computational cost if the number of the training dataset and the neural network architecture are large.

In order to alleviate the trouble of finding a proper value of λ and solve the stagnant problem in applying the convexification methodology, we demonstrate the deconvexification methodology in this chapter with proposing a new series of training methods based on the gradual deconvexification (GDC). To present the advantage of GDC, we evaluate the new training methods on the same function approximation and handwritten digit recognition tasks as being tested by the NRAE and the NRAE-MSE training methods.

5.1 Gradual Deconvexification

The essence of GDC [75] is to apply multiple NRAE training sessions with the convexification and deconvexification phases, which are associated with different λ values, to gradually convexify the error space of NRAE for avoiding the non-global local minimum during the training. GDC begins with the NRAE training under a very large initial λ to greatly expand the convexity region of RAE contained in the convexity region of NRAE. Then, if the convexification under the current λ is completed or the stagnant problem is detected, a deconvexification is performed to reduce the current λ to $\lambda \cdot DR$ with a deconvexifing rate $DR \in (0, 1)$. Afterward, GDC continues and repeats the above convexification and deconvexification phases if necessary until a satisfactory training error is achieved.

The training method with the use of GDC is described in Algorithm 3. In order to decide when the deconvexification should be performed in GDC, we apply d_{prev} and d_{next} to keep recording the values of $C_\lambda(w)$ before and after a preselected number of training epochs TE. If $\Delta d = |d_{\text{prev}} - d_{\text{next}}|$ is less than a threshold TS, GDC considers that the convexification under the current λ is completed or the stagnant problem is occurred at the moment. In addition, if the deconvexification brings the value of λ down to $\lambda < 1$, then according to Eq. (3.16), GDC will directly set $\lambda = 0$ to perform the MSE training for avoiding excessive deconvexification phases. Particularly, it is noticed that the overall converging speed of GDC is controlled by TE, TS and DR, where the larger (or smaller) values of TE, TS and DR lead to the slower (or faster) deconvexification in GDC. However, a drastic deconvexification in GDC could cause the rapidly shrinking of applicable convexity regions of NRAE, which limits the effect of the NRAE criterion in alleviating the non-global local minimum. Thus, it should be avoided if possible in practice.

With the use of the deconvexification in GDC, the error space of NRAE transforms correspondingly as λ decreases. As a consequence, the deconvexification prevents

Algorithm 3 Gradual Deconvexification

Require: Initialize the weight vector w randomly, choose TE, TS and DR, set a desired training error δ, select $\lambda >> 1$, and set $ctn \leftarrow 1$ and $flag \leftarrow 0$;

while $C_\lambda(w) \geq \delta$ **do**
  if $ctn = 1$ **then**
    Let $d_{\text{prev}} \leftarrow C_\lambda(w)$;
  end if
  Perform the NRAE training and update w to w^*;
  Set $ctn \leftarrow ctn + 1$;
  if $ctn = TE$ **then**
    Let $d_{\text{next}} \leftarrow C_\lambda(w)$ and set $flag \leftarrow 1$;
  end if
  if $flag = 1$ **then**
    $\Delta d \leftarrow |d_{\text{prev}} - d_{\text{next}}|$;
    Set $ctn \leftarrow 1$ and $flag \leftarrow 0$;
    if $\Delta d \leq TS$ **then**
      $\lambda \leftarrow \lambda \cdot DR$;
      if $\lambda < 1$ **then**
        Set $\lambda \leftarrow 0$;
      end if
    end if
  end if
  Set $w \leftarrow w^*$;
end while
return The optimal weight vector w^*.

$\|\varepsilon_M(w)\|^2$ to be locked on certain K' training samples repeatedly when λ is very large, thus avoiding the rank deficiency of $D_K(w)$. As long as the GDC parameters provide smooth deconvexification procedures, the NRAE trainings can be performed in adequate convexity regions as λ gradually decreases, assuring that the GDC method consistently works well without any stagnancy.

5.2 NRAE Training in Pairwise Mode

In training the MLP with BP, the training method can be performed in either batch or pairwise (also called ”on-line” or ”sequential”) mode. Batch training accumulates weight changes over presentation of all training samples before applying an update to weights, while pairwise training updates weights immediately after presentation of each training sample. Some research works [28, 87] indicate that batch training is at least theoretically superior to pairwise training, because batch training uses the true gradient, or follow the true gradient more closely, to update weights in training MLPs. Some other literatures [23, 37, 87] claim that batch training is as fast as or faster than pairwise training. However, many advantages of pairwise training have been noticed and discussed in [4, 7, 38], while several comprehensive summaries and comparisons between batch and pairwise training in [65, 115] point out that batch training is almost always slower by orders of magnitude than pairwise training especially in training with large datasets. Pairwise training is able to safely use a large learning rate to achieve a reasonably good result with a significantly fast convergence speed, but batch training can only follow the true gradient very well along the error space when the learning rate is small enough to yield an optimal results. As the size of the training dataset gets larger, the magnitude of accumulated weights changed in batch training becomes larger too. In this case, batch training must use a small learning rate to prevent weight oscillations across the weight space and avoid the risk of the node saturation in MLPs. Accordingly, this learning manner lowers the convergence speed of batch training. On the other hand, pairwise training applies weight changes as soon as they are calculated, thus it can handle different sizes of training datasets without requiring a small learning rate. Therefore, pairwise training is expected to improve the convergence speed of NRAE-based training methods, thus leading these methods to handle large datasets more efficiently.

In training an MLP on pairwise mode with K samples, a training epoch contains K training iterations where each iteration takes one training sample to evaluate the gradient of the training criterion. The NRAE criterion and its first-order derivative defined in Eq. (3.8) and Eq. (3.9) are restated as follows:

$$C_\lambda(w) = \frac{1}{\lambda}\ln\frac{1}{K} + \|\varepsilon_M(w)\|^2 + \frac{1}{\lambda}\ln\left[\sum_{k=1}^{K}\eta_k(w)\right]$$

$$\frac{\partial C_\lambda(w)}{\partial w_i} = \frac{-2\sum_{k=1}^{K}\eta_k(w)\,\varepsilon_k^T(w)\,\frac{\partial\hat{y}_k(w)}{\partial w_i}}{\sum_{k=1}^{K}\eta_k(w)}$$

It is noticed that the computation of $\frac{\partial C_\lambda(w)}{\partial w_i}$ can only be performed on batch mode, because the evaluations of both $\sum_{k=1}^{K}\eta_k(w)\,\varepsilon_k^T(w)\,\frac{\partial\hat{y}_k(w)}{\partial w_i}$ and $\sum_{k=1}^{K}\eta_k(w)$ require to compute summations with K training samples, while pairwise training cannot be performed by taking only one training sample to compute $\frac{\partial C_\lambda(w)}{\partial w_i}$ in a training iteration. Apparently, if the number of training samples K is very large, the evaluation of $\frac{\partial C_\lambda(w)}{\partial w_i}$ would be significantly inefficient on batch mode.

For a weight vector $w^{(k)}$ according to the k-th training sample, a formula to compute $\frac{\partial C_\lambda(w^{(k)})}{\partial w_i}$ in pairwise mode can be described as

$$\frac{\partial C_\lambda(w^{(k)})}{\partial w_i} = \frac{-2\eta_k(w^{(k)})\varepsilon_k^T(w^{(k)})\frac{\partial\hat{y}_k(w^{(k)})}{\partial w_i}}{\sum_{j=1}^{K}\eta_j(w^{(k)})} \tag{5.1}$$

where

$$\begin{aligned}\eta_k(w^{(k)}) &= e^{\lambda\left(\left\|\varepsilon_k(w^{(k)})\right\|^2-\left\|\varepsilon_M(w^{(k)})\right\|^2\right)}\\ \sum_{j=1}^{K}\eta_j(w^{(k)}) &= \sum_{j=1}^{K}e^{\lambda\left(\left\|\varepsilon_j(w^{(k)})\right\|^2-\left\|\varepsilon_M(w^{(k)})\right\|^2\right)}\end{aligned} \tag{5.2}$$

Thus, Eq. (5.1) can be applied to perform pairwise training with the NRAE criterion. How-

ever, evaluating $\eta_k(w^{(k)})$ and $\sum_{j=1}^{K} \eta_j(w^{(k)})$ in Eq. (5.2) could cost significantly computational time when K is very large. Because the computations in Eq. (5.2) have to decide the largest M of $\varepsilon_k(w^{(k)})$ for $k = 1, ..., K$ in K times per training iteration, leading to K^2 evaluations of $\varepsilon_M(w^{(k)})$ in computing both $\eta_k(w^{(k)})$ and $\sum_{j=1}^{K} \eta_j(w^{(k)})$ per training epoch.

In order to reduce the above evaluation cost in Eq. (5.2), we propose a method to estimate the true gradient $\frac{\partial C_\lambda(w^{(k)})}{\partial w_i}$ of the NRAE criterion in pairwise mode as

$$\frac{\partial C_\lambda(w^{(k)})}{\partial w_i} = \frac{-2\eta_k(w)\varepsilon_k^T(w^{(k)})\frac{\partial \hat{y}_k(w^{(k)})}{\partial w_i}}{\sum_{k=1}^{K} \eta_k(w)} \tag{5.3}$$

where $\eta_k(w)$ and $\sum_{k=1}^{K} \eta_k(w)$ are evaluated as same as in batch mode, providing the estimations of $\eta_k(w^{(k)})$ and $\sum_{j=1}^{K} \eta_j(w^{(k)})$ in Eq. (5.1). With such an approximation, $\eta_k(w)$ is only evaluated K times and $\sum_{k=1}^{K} \eta_k(w)$ is only evaluated once in Eq. (5.3) at the beginning of each training epoch, where the computation of $\frac{\partial C_\lambda(w^{(k)})}{\partial w_i}$ does not need to update $\eta_k(w)$ and $\sum_{k=1}^{K} \eta_k(w)$ anymore per training iteration. This method is applied as the pairwise NRAE training method [34] and it is described in Algorithm 4. Since the GDC method is composed of multiple NRAE training sessions, it can be performed in pairwise mode, called the pairwise GDC method, with integrating all the NRAE training in GDC as pairwise mode.

5.3 Experimental Evaluation

As we described in Chapter 4, training MLPs with the MSE criterion to approximate functions with fine features or under-sampled segments are known to have non-global local minima. In this section, the same examples of approximating functions designed to have non-global local minima as we tested before are applied to evaluate performances of the GDC method in batch and pairwise mode for training MLPs. Furthermore, we apply the

Algorithm 4 Pairwise NRAE Training Method

Require: Initialize the weight vector w^n randomly, choose a desired training error δ, select $\lambda >> 1$;

while $C_\lambda(w) \geq \delta$ **do**
  for $k = 1$ to K **do**
    Evaluate $\|\varepsilon_k(w)\|^2$ with respect to the weight vector w;
  end for
  Determine $\varepsilon_M(w)$;
  Set $\eta_{sum} \leftarrow 0$;
  for $k = 1$ to K **do**
    $\eta_k(w) \leftarrow e^{\lambda\left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2\right)}$;
    $\eta_{sum} \leftarrow \eta_{sum} + \eta_k(w)$;
  end for
  Set $w^{(1)} \leftarrow w$;
  for $k = 1$ to $K - 1$ **do**
    $\frac{\partial C_\lambda(w^{(k)})}{\partial w_i} \leftarrow \frac{-2\eta_k(w)\varepsilon_k^T(w^{(k)})\frac{\partial \hat{y}_k(w^{(k)})}{\partial w_i}}{\eta_{sum}}$;
    Update $w^{(k)}$ to $w^{(k+1)}$ using $\frac{\partial C_\lambda(w^{(k)})}{\partial w_i}$;
  end for
  Set $w^* \leftarrow w^{(K)}$ and $w \leftarrow w^*$;
end while
return The optimal weight vector w^*;

GDC method to train the MLP classifier for recognizing handwritten digits with the MNIST dataset, with demonstrating the superior performance of the MLP classifier trained by the pairwise GDC method comparing to many benchmark results.

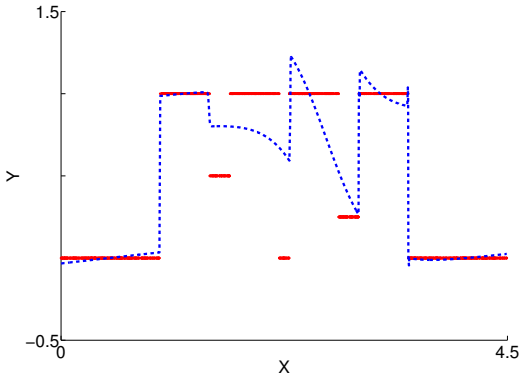
5.3.1 Function Approximation

Gradual Deconvexification The four function approximation tasks defined in Chapter 4 are applied to evaluate the performance of the GDC method in batch mode. Ten different sets of initial weights of the MLP are randomly chosen to start 10 training groups. In each group, a GDC training and a standard MSE training are performed on the same set of initial weights. For all GDC training sessions, we compute their corresponding

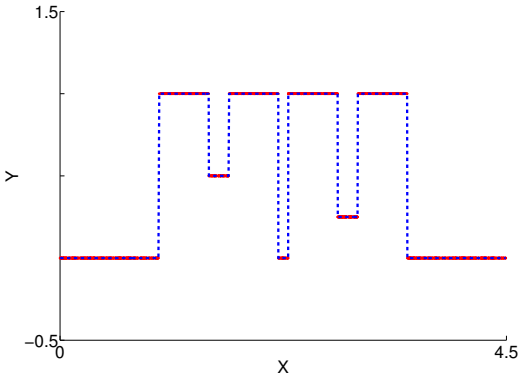
MSE values as training errors and compare them to the results of the MSE training. In all GDC training sessions, we set $\lambda = 10^{20}$ as the initial value, $TE = 1,000$ as the maximum training epochs to check the deviation $\Delta d = |d_{\text{prev}} - d_{\text{next}}|$, $TS = 10^{-10}$ as the threshold to decide whether the deconvexification performs, and $DR = 0.9$ as the deconvexifing rate.

Experimental results in Fig. 5.1 demonstrate that GDC has the capability to capture all significant features located on the target functions with avoiding non-global local minima, indicating a better performance than the MSE training. Moreover, training results in Fig. 5.2 present that all GDC training sessions with 10 different sets of initial weights lead all trained MLPs to achieve satisfactory training errors, which are consistently lower than the MSE training achieved. These observations are consistent with the experimental results achieved by the NRAE and NRAE-MSE training method comparing to the MSE training. Meanwhile, since GDC starts with a very large λ and gradually reduces the value of λ until the training converges, it eliminates the requirement of finding an proper initial λ in performing the NRAE or the NRAE-MSE training. At last, all the GDC training sessions for different function approximation tasks employ the same initial λ as well as the same settings of TE, TS, and DR, illustrating that these parameters in GDC are not sensitive to our training tasks.

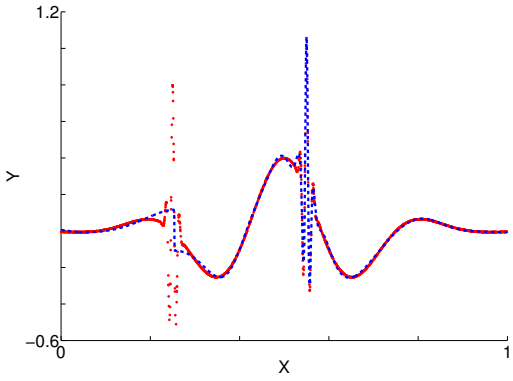
To compare performances between the GDC method and the MSE/NRAE/NRAE-MSE training method, we apply GDC to perform the three-notch function approximation with noiseless data under the same experimental settings as the MSE/NRAE/NRAE-MSE training method performed in Chapter 4. The results presented in Table 5.1 indicate that the best training errors over all experiments can be consistently achieved by the GDC method over 10 different sets of initial weights without selecting different initial values of λ. Therefore, as a NRAE-based training method, GDC is confirmed to have satisfactory performances better than both the NRAE and the NRAE-MSE training method in avoiding the non-global local minimum and achieving the global or near-global minimum.



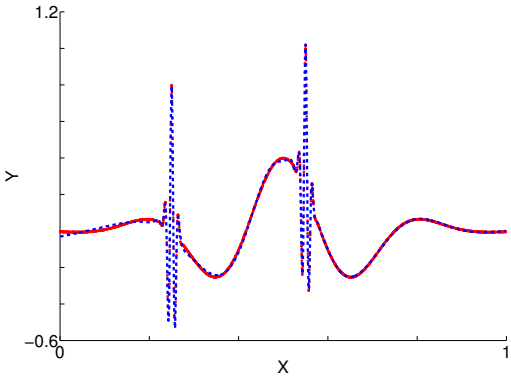
(a) MSE Training



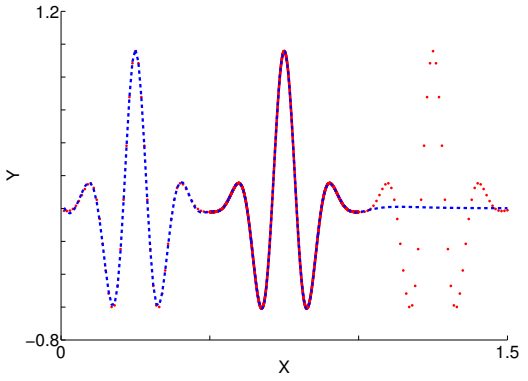
(b) GDC Training



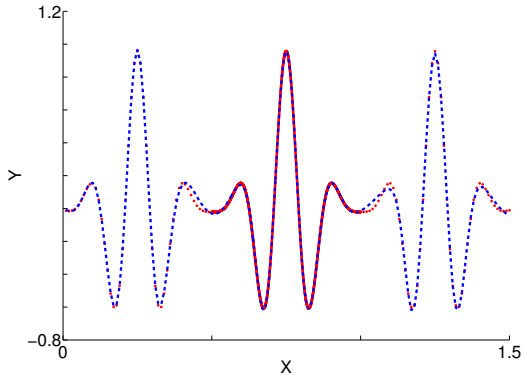
(c) MSE Training



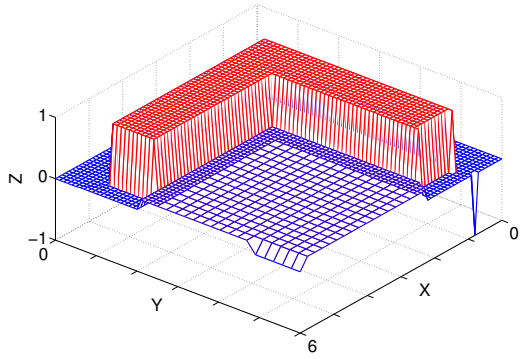
(d) GDC Training



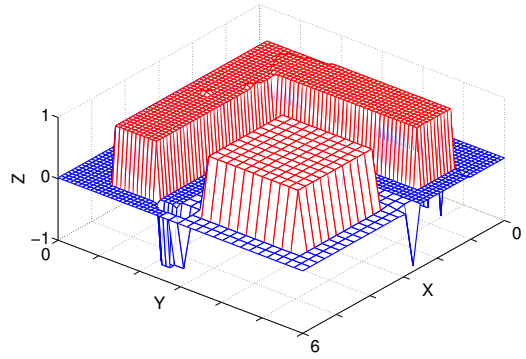
(e) MSE Training



(f) GDC Training

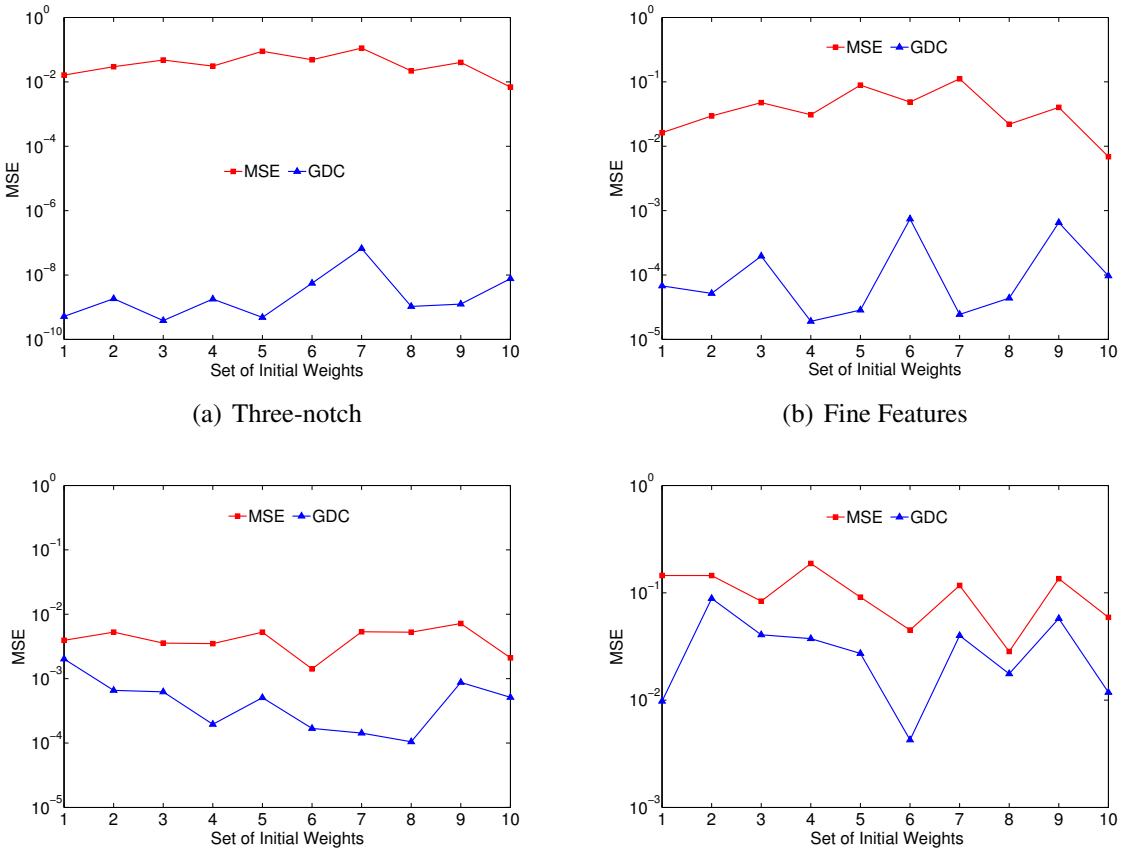


(g) MSE Training



(h) GDC Training

FIG. 5.1. Fitting plots of four function approximation examples achieved by the MSE and GDC training. Numbers on horizontal and vertical axes in each subfigure represent the input and output of the function, respectively. In Fig. 5.1(a) - Fig. 5.1(f), red dots denote target training samples, and blue dash lines are MLP approximated function plots. On Fig. 5.1(g) and Fig. 5.1(h), only MLP approximated function plots are shown by using blue and red colors to distinguish different function values on vertical axes.



(a) Three-notch

(b) Fine Features

(c) Under-sampled Segments

(d) Under-sampled Square

FIG. 5.2. Training errors of ten set of initial weights for four function approximation examples achieved by the MSE and GDC training. Solid lines with colors and symbols denote different training methods of the MSE (red line with squares) and the GDC (blue line with triangles). In order to clearly illustrate differences between MSE values obtained by the MSE and GDC training, actual numbers in all vertical axes are converted to logarithmic numbers with respect to base 10.

Table 5.1. MSEs of the three-notch function approximation with noiseless data achieved by the MSE, the NRAE, the NRAE-MSE, and the GDC training methods. The best performances achieved by the NRAE training method are highlighted by underlines, while the best performances achieved by the NRAE-MSE training method are highlighted in bold. The best performances achieved by the GDC method are highlighted by underlines in bold.

Three-notch Function Approximation		Set of Initial Weights									
		1	2	3	4	5	6	7	8	9	10
MSE		1.05×10^{-1}	1.94×10^{-2}	1.11×10^{-4}	1.35×10^{-2}	1.46×10^{-1}	1.01×10^{-1}	7.54×10^{-4}	5.42×10^{-3}	5.41×10^{-3}	8.97×10^{-2}
$\lambda = 10^6$	NRAE	7.14×10^{-8}	8.04×10^{-2}	4.55×10^{-2}	8.36×10^{-7}	9.25×10^{-1}	8.25×10^{-3}	9.36×10^{-8}	7.16×10^{-8}	1.43×10^{-2}	$\underline{5.23 \times 10^{-9}}$
	NRAE-MSE	1.47×10^{-5}	3.68×10^{-4}	1.99×10^{-5}	3.25×10^{-4}	2.54×10^{-4}	1.62×10^{-4}	1.24×10^{-4}	9.27×10^{-5}	6.83×10^{-4}	4.85×10^{-4}
$\lambda = 10^7$	NRAE	4.25×10^{-8}	2.86×10^{-2}	3.86×10^{-2}	9.14×10^{-4}	8.04×10^{-7}	4.27×10^{-2}	$\underline{8.01 \times 10^{-9}}$	9.02×10^{-8}	9.72×10^{-3}	7.08×10^{-7}
	NRAE-MSE	1.35×10^{-5}	1.57×10^{-5}	1.58×10^{-5}	2.48×10^{-5}	2.06×10^{-4}	1.54×10^{-5}	7.96×10^{-5}	5.05×10^{-5}	2.75×10^{-4}	3.99×10^{-4}
$\lambda = 10^8$	NRAE	$\underline{4.73 \times 10^{-9}}$	8.48×10^{-8}	$\underline{1.95 \times 10^{-8}}$	$\underline{1.53 \times 10^{-7}}$	9.54×10^{-7}	$\underline{7.64 \times 10^{-7}}$	5.26×10^{-7}	2.89×10^{-8}	1.74×10^{-7}	6.54×10^{-7}
	NRAE-MSE	4.75×10^{-6}	5.99×10^{-8}	9.00×10^{-9}	7.57×10^{-7}	5.56×10^{-7}	1.69×10^{-6}	7.29×10^{-8}	2.26×10^{-8}	6.74×10^{-8}	9.54×10^{-6}
$\lambda = 10^9$	NRAE	3.52×10^{-8}	$\underline{8.79 \times 10^{-9}}$	4.64×10^{-7}	7.23×10^{-7}	$\underline{5.54 \times 10^{-8}}$	8.66×10^{-7}	8.80×10^{-9}	$\underline{4.52 \times 10^{-9}}$	$\underline{7.37 \times 10^{-8}}$	9.36×10^{-7}
	NRAE-MSE	1.86×10^{-7}	6.66×10^{-8}	1.34×10^{-10}	4.83×10^{-8}	4.46×10^{-7}	4.63×10^{-7}	$\mathbf{1.63 \times 10^{-8}}$	9.35×10^{-9}	4.05×10^{-8}	1.35×10^{-7}
$\lambda = 10^{10}$	NRAE	8.43×10^{-8}	1.88×10^{-2}	8.78×10^{-3}	3.65×10^{-6}	1.17×10^{-6}	3.76×10^{-5}	2.01×10^{-3}	7.94×10^{-5}	5.68×10^{-7}	1.94×10^{-6}
	NRAE-MSE	7.33×10^{-8}	$\mathbf{1.96 \times 10^{-8}}$	9.34×10^{-11}	1.24×10^{-8}	7.53×10^{-8}	2.63×10^{-7}	1.74×10^{-8}	4.23×10^{-9}	1.72×10^{-8}	$\mathbf{3.78 \times 10^{-8}}$
$\lambda = 10^{11}$	NRAE	2.00×10^{-6}	6.07×10^{-6}	1.87×10^{-3}	2.16×10^{-4}	3.53×10^{-5}	9.85×10^{-2}	2.02×10^{-5}	4.39×10^{-5}	1.28×10^{-5}	1.27×10^{-7}
	NRAE-MSE	$\mathbf{2.57 \times 10^{-8}}$	2.16×10^{-8}	$\mathbf{5.50 \times 10^{-11}}$	$\mathbf{7.24 \times 10^{-9}}$	$\mathbf{2.54 \times 10^{-8}}$	$\mathbf{1.12 \times 10^{-7}}$	3.24×10^{-8}	$\mathbf{3.21 \times 10^{-9}}$	$\mathbf{9.63 \times 10^{-9}}$	7.78×10^{-8}
GDC		$\underline{\mathbf{3.85 \times 10^{-10}}}$	$\underline{\mathbf{8.74 \times 10^{-9}}}$	$\underline{\mathbf{1.45 \times 10^{-11}}}$	$\underline{\mathbf{5.27 \times 10^{-9}}}$	$\underline{\mathbf{2.27 \times 10^{-10}}}$	$\underline{\mathbf{1.28 \times 10^{-9}}}$	$\underline{\mathbf{5.92 \times 10^{-9}}}$	$\underline{\mathbf{2.11 \times 10^{-9}}}$	$\underline{\mathbf{8.19 \times 10^{-9}}}$	$\underline{\mathbf{3.46 \times 10^{-9}}}$

Pairwise Gradual Deconvexification In order to explore the effectiveness of pairwise training comparing to batch training in improving the convergence of training MLPs, we apply the four function approximation examples as same as we adopted in Chapter 4 to evaluate the performance of methods as the pairwise GDC training, the GDC training, the least mean square (LMS) training, and the MSE training. Accordingly, the pairwise GDC and the LMS training are performed in pairwise mode, while the GDC and the MSE training are performed in batch mode.

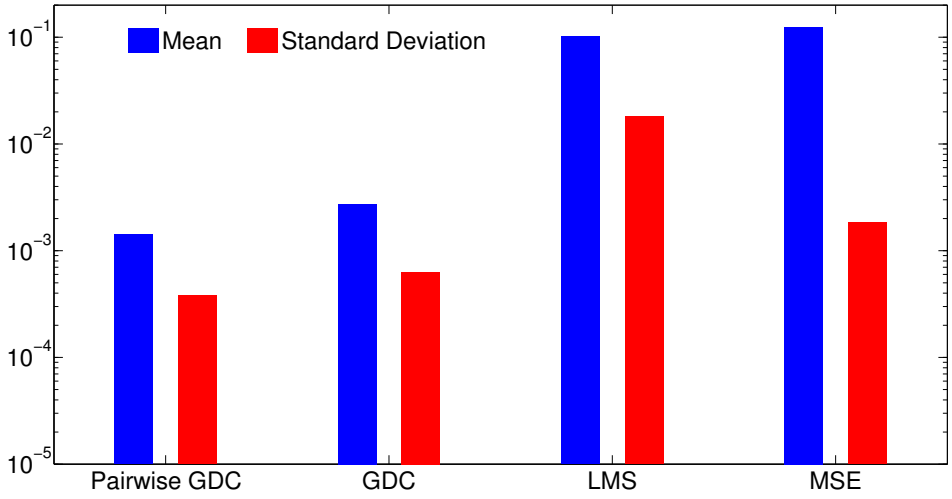
For each function approximation example, 10 different sets of initial weights of the MLP are randomly chosen. Starting with each set of initial weights, one pairwise GDC

training, one GDC training, one LMS training, and one MSE training are performed. These training sessions for the same set of initial weights are considered as a training group. The corresponding value of MSE of the MLP is recorded as the training error for each session after the training is converged, and we collect the mean and standard deviation of training errors among 10 training groups for each type of the training methods. In addition, we consider the time in seconds as the training cost of each session, and we compute the mean of training costs among 10 training groups as the average training cost for each type of the training methods. Notice that the training time of each session is always recorded after the corresponding training is converged.

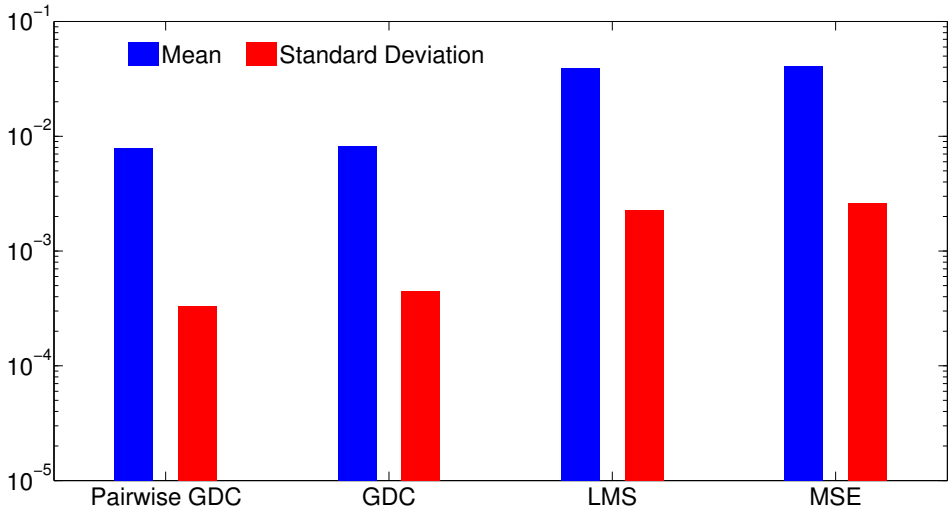
For all training sessions performed on function approximation examples, we apply GD rather than BFGS for batching training with setting the learning rate and the momentum term equal to 0.001 and 0.5, and we employ stochastic gradient descent (SGD) for pairwise training with setting the learning rate and the momentum term equal to 0.01 and 0.5, respectively. For the GDC method, we set $\lambda = 10^{20}$, $TE = 1,000$, $TS = 10^{-10}$, and $DR = 0.9$.

Fig. 5.3 illustrates the means and standard deviations of training errors achieved by the tested training methods in function approximation examples. The results demonstrate that both the pairwise GDC and the GDC training methods consistently achieve satisfactory average training errors and standard deviations lower than the LMS and the MSE training methods. It confirms that the GDC method has the ability to achieve training results better than the LMS and the MSE methods with avoiding non-global local minima in training MLPs. Furthermore, the pairwise GDC training reaches the same level of average training errors as the GDC training, implying that the pairwise GDC training method is capable to be applied to effectively solve function approximation tasks as same as the GDC training method.

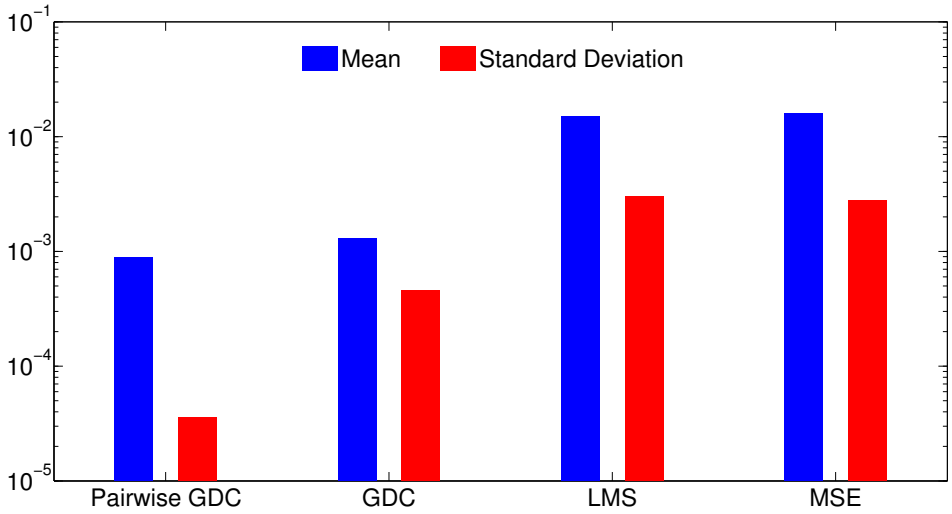
In Fig. 5.4, it presents average training costs of the tested training methods in func-



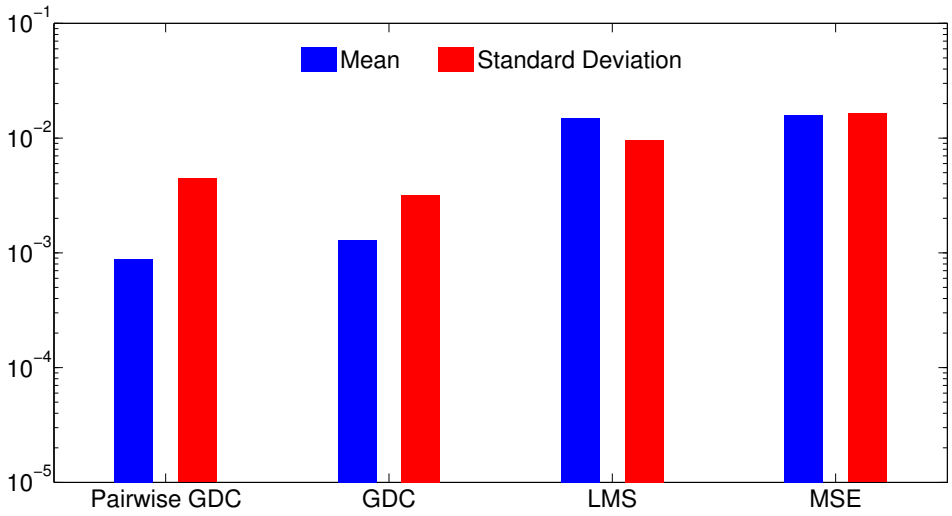
(a) Three-notch



(b) Fine Features



(c) Under-sampled Segments



(d) Under-sampled Square

FIG. 5.3. Values of mean and standard deviation of training errors for four function approximation examples achieved by the pairwise GDC, the GDC, the LMS, and the MSE training methods. Blue and red bars in each subfigure represent values of mean and standard deviation of training errors that are collected among 10 different training sessions, respectively. The corresponding value of MSE of the MLP is collected as the training error for each session after the training is converged. Vertical axes in each subfigure apply the logarithmic scale with the base 10.

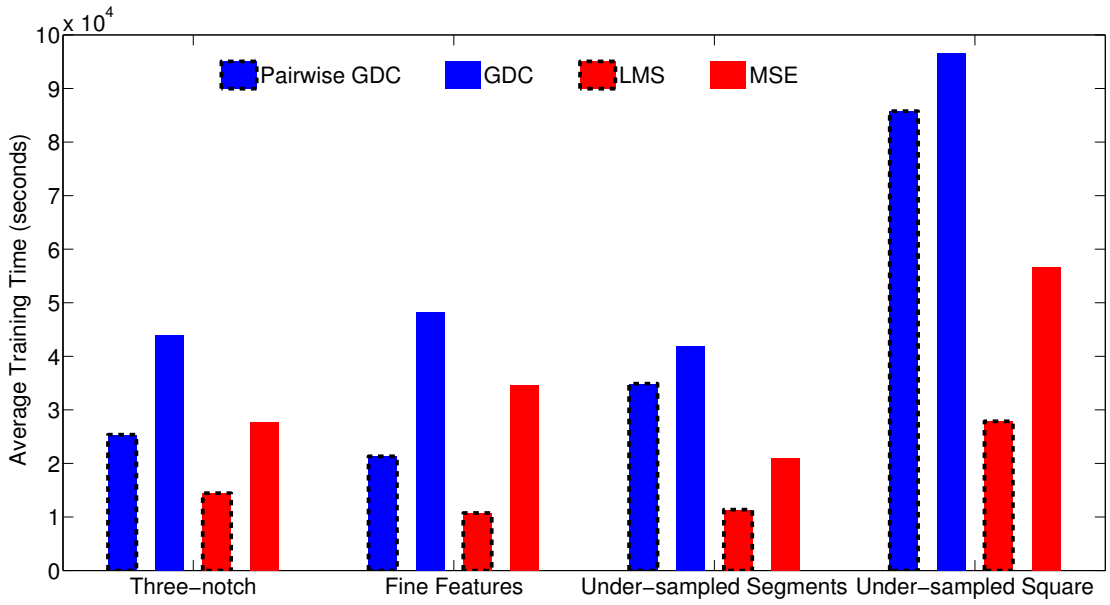


FIG. 5.4. Average training time of four function approximation examples achieved by the pairwise GDC, the GDC, the LMS, and the MSE training methods. Bars in blue denote the GDC training results, while other bars in red represent the MSE training results. Particularly, bars surrounding by dash lines represent pairwise training. Numbers on the vertical axis shows the average training time in seconds for different training methods.

tion approximation examples. Experimental results demonstrate that the pairwise GDC training reduces nearly 42%, 56%, 16%, and 11% average training costs comparing to the GDC training in approximating the three-notch, fine features, under-sampled segments, and under-sampled square functions, respectively. In addition, it is observed in Fig. 5.4 that the LMS (or the MSE) training take less computational time than the pairwise GDC (or the GDC) training. The reason is that the objective function and the gradient of the MSE criterion can be evaluated with less computational costs than the evaluation of the NRAE criterion in both batch and pairwise modes. However, training errors achieved with the use of the LMS (or the MSE) criterion never outperform the results obtained by the pairwise GDC (or the GDC) training over all of our experiments.

5.3.2 Handwritten Digit Recognition

To test the capability of GDC in training a large MLP on a real-world task, the pairwise GDC method is used to train the MLP for recognizing handwritten digits with the MNIST dataset. In our experiments, we evaluate the MNIST dataset on a 2-layer MLP with the architecture of 784-300-10. Ten different sets of initial weights are used to perform 10 training sessions. In each training session, 60,000 training samples are applied to train the MLP. After the corresponding training is converged, 10,000 test samples are employed to test the trained MLP with providing the test error rate for each testing session. For the pairwise GDC method, we set $\lambda = 10^4$, $TE = 10$, $TS = 10^{-5}$, and $DR = 0.9$. For all the pairwise NRAE training in GDC, we set the global learning rate and the momentum term of SGD equal to 0.0001 and 0.5, respectively. For all the LMS training sessions in GDC when $\lambda < 1$, we apply the learning rate decay to decrease the global learning rate by 50% of the original value in every 20 training epochs.

Table 5.2 presents test error rates of the MNIST dataset obtained by different MLP classifiers. Major advantages of the pairwise GDC method in training the MLP classifier are described in following:

1. Experimental results in Table 5.2 show that the MLP classifier trained by the pairwise GDC method has the lowest test error rate than many benchmark MLP classifiers. Based on our experiments, the MLP classifiers trained by the pairwise GDC method consistently achieve low test error rates over all testing sessions with the same settings of MLPs. The test error rates achieved in 10 testing sessions are separately as 2.61%, 2.67%, 2.70%, 2.73%, 2.88%, 2.71%, 2.68%, 2.65%, 2.74%, and 2.69%. It indicates that the pairwise GDC training method has the ability to provide satisfactory generalization results with different sets of initial weights, thus requiring no multiple selections of initial weights for the training. It is appropriate to note that the

pairwise GDC method also provides the test error rate lower than the best test error rate achieved by the NRAE-MSE training presented in Table 4.2.

2. For three benchmark MLP classifiers listed in Table 5.2, one has 1000 hidden nodes as the 784-1000-10 MLP, and the other two are built by 3-layer MLPs that are 784-300-100-10 and 784-500-150-10 respectively. However, the MLP classifier trained by the pairwise GDC method merely uses 300 hidden nodes on the 784-300-10 MLP. It demonstrates that the pairwise GDC method has the ability to avoid non-global local minima and maintain a high level of generalization without applying the larger MLP architecture comparing to the listed benchmark MLP classifiers.

3. Two benchmark results using the 784-300-10 and the 784-1000-10 MLPs are achieved with image distortions to increase the size of the original training dataset. This technique improves test error rates of the two benchmark MLP classifiers, but the results obtained by those MLP classifiers are still worse than the test error rate of the MLP classifier trained by the pairwise GDC method without image distortions. It indicates that the MLP classifier is able to be trained effectively by the pairwise GDC method in achieving low test error rate. In fact, through applying image distortions, the pairwise GDC method is expected to lead the MLP classifier to achieve a lower test error rate than the result of the MLP classifier trained without aiding of image distortions.

4. A GDC training in batch mode with the same experimental settings as the pairwise GDC training is also tested with the MNIST dataset in our experiments. Because the GDC training converges extremely slow comparing to the pairwise GDC training, we have not obtained any converged GDC training sessions with satisfactory performances better than the pairwise GDC training. Thus, no related result of the GDC training is shown in Table 5.2. In our experiments, the best test error rate achieved

Table 5.2. Test error rates achieved by MLP classifiers on the MNIST dataset. The best test error rate achieved by the pairwise GDC method is highlighted in bold.

Training Method	Test Error Rate (%)
784-300-10 MLP + MSE [64]	4.70
784-300-10 MLP + NRAE-MSE (this dissertation)	4.58
784-1000-10 MLP + MSE [64]	4.50
784-1000-10 MLP + MSE + Image distortions [64]	3.80
784-300-10 MLP + MSE + Image distortions [64]	3.60
784-300-100-10 MLP + MSE [64]	3.05
784-500-150-10 MLP + MSE [64]	2.95
784-300-10 MLP + Pairwise GDC (this dissertation)	**2.61**

by the GDC training is nearly 10%, and it costs almost 15,000 training epochs. However, in all pairwise GDC training sessions, the total number of training epochs for each session is less than 1,000. It presents a significant advantage of the pairwise GDC training comparing to the batch GDC training in saving computational costs on a large training dataset.

5.4 Summary

The GDC method is developed to solve the difficulty of the stagnant problem in the NRAE training with a large λ, thus leading to a global or near-global minimum of the NRAE criterion without a deliberate selection of the initial λ in training different MLPs. Numerical experiments in approximating functions with pre-designed non-global local minima demonstrate the effectiveness of the GDC method in searching a global minimum of $C_\lambda(w)$ of function approximation examples. Particularly, if the MLP has enough nodes, $\min_w C_\lambda(w)$ is virtually zero at a global minimum, implying that $\min_w Q(w)$ is also virtually zero where the global minimum or near-global minimum of the MSE criterion could

be achieved. Moreover, since the GDC method further improves the NRAE-MSE training, which is confirmed better than the NRAE training, it is the best NRAE-based training method over all others we proposed.

Although the GDC method can successfully solve function approximation examples, it is restricted by its batch training fashion, which becomes the difficulty in handling the large dataset with a fast convergence. To improve this shortage, a pairwise GDC training method is proposed and evaluated. Experimental results demonstrate that the pairwise GDC training method reaches an accuracy level similar to what the GDC training achieves in batch mode, but requires much less computation in approximating the same function example. Furthermore, in training a large MLP classifier with the MNIST dataset, the pairwise GDC method is able to achieve a performance better than many benchmark results of MLP classifiers.

Chapter 6

TRAINING DEEP NEURAL NETWORKS

Notable effectiveness of the GDC method are comprehensively evaluated in training MLPs for approximating functions with non-global local minima and recognizing handwritten digits with the MNIST dataset. Although the GDC method can gradually handle the non-convex function in distinct convex error spaces as an initially large λ reduces, the effectiveness of the GDC method in training DNNs with the high-dimensional non-convex error function is unclear. In fact, employing the GDC method to train neural networks with deep architectures and large datasets encounters tremendously computational costs. In this chapter, we explain the limitation of GDC in training DNNs and propose an enhanced gradual deconvexification (EGDC) method to train CNNs and deep MLPs in an efficient and effective manner. Numerical experiments for recognizing handwritten digits on the MNIST dataset are applied to evaluate the optimization and generalization performances of the EGDC method, with comparing to many benchmark results under the same experimental conditions.

6.1 Enhanced Gradual Deconvexification Method

6.1.1 GDC-RAE Training

So far as we discussed, the NRAE training has a stagnant problem when λ is chosen very large, and the problem can be resolved by applying the GDC method to gradually deconvexify the NRAE criterion in searching a global or near-global minimum. However, we have not discussed another issue in the NRAE training when λ is chosen very small.

Since the NRAE criterion $C_\lambda(w)$ is a normalization of the RAE criterion $J_\lambda(w)$ with a logarithmically concave function, $C_\lambda(w)$ has a smooth but a flat error space compared to $J_\lambda(w)$. Although $C_\lambda(w)$ shares the same local and global optima with $J_\lambda(w)$ when the convexity of $J_\lambda(w)$ is maintained, $C_\lambda(w)$ has a larger convexity region than $J_\lambda(w)$, where the error space of $C_\lambda(w)$ contains more planes and plateaus than the error space of $J_\lambda(w)$, even if $J_\lambda(w)$ is strictly convex as λ approaches infinity. When λ is close to zero, the planes and plateaus existed in the error space of $C_\lambda(w)$ become more severe, because the error space of $C_\lambda(w)$ is not convex anymore under this scenario. However, the error space of $J_\lambda(w)$ is less smooth than the error space of $C_\lambda(w)$, thus the former suffers less than the latter from the severe planes and plateaus, which potentially slow down the gradient-based method in searching the global or near-global minimum. Accordingly, it implies that the RAE training can converge faster than the NRAE training in avoiding the same planes and plateaus, if λ is close to zero.

The RAE training has limitations in choosing λ only with small values because of the exponential computation of $J_\lambda(w)$. However, the RAE training can be safely performed in the GDC method, if the deconvexification in GDC brings down the initial large λ to a very small value, which is smaller enough for the RAE training to be properly performed without any computational overflow. Therefore, it inspires us to integrate the RAE training into the GDC method and compose a new GDC-RAE training to attain both the fast convergence

and the good optimization. The basic idea of the GDC-RAE training performs GDC by switching the NRAE training to the RAE training when $\lambda < \lambda_{\text{c}}$, where λ_{c} is a critical value of λ that can be handled by $J_\lambda(w)$ without any computational overflow. The selection approach of λ_{c} is illustrated as follows.

Based on the defination from Eq. (3.4) to Eq. (3.7), we can rewrite Eq. (2.1) as

$$\begin{aligned} J_\lambda(w) &= \sum_{k=1}^{K} e^{\left(\lambda \left\| y_k - \hat{f}(x_k, w) \right\|^2\right)} \\ &= e^{\lambda \|\varepsilon_M(w)\|^2} \sum_{k=1}^{K} \eta_k(w) \end{aligned} \tag{6.1}$$

where $0 < J_\lambda(w) \leq K e^{\lambda \|\varepsilon_M(w)\|^2}$. It is noticed that the evaluation of $J_\lambda(w)$ will not cause any computational overflow if $K e^{\lambda \|\varepsilon_M(w)\|^2} < F_{\text{max}}$, where F_{max} is the largest positive floating point number that can be handled by the computer. Based on this fact, the selection of λ_{c} can be decided by

$$\lambda_{\text{c}} := \frac{\ln F_{\text{max}} - \ln K}{\|\varepsilon_M(w)\|^2} \tag{6.2}$$

As we choose a proper λ_{c} to apply the RAE training, the computation of the first-order derivative of the RAE criterion as presented in Eq. (3.14) could yield a significant large magnitude of the RAE gradient

$$\nabla J_\lambda(w) := \left[\frac{\partial J_\lambda(w)}{\partial w_i} \right]_{1 \times N} \tag{6.3}$$

To overcome this difficulty, let us first rewrite Eq. (3.14) as

$$\begin{aligned}\frac{\partial J_{\lambda}\left(w\right)}{\partial w_{i}} &= -2\lambda\sum_{k=1}^{K}e^{\lambda\left\|\varepsilon_{k}(w)\right\|^{2}}\varepsilon_{k}^{T}\left(w\right)\frac{\partial\hat{y}_{k}\left(w\right)}{\partial w_{i}}\\ &= -2\lambda e^{\lambda\left\|\varepsilon_{M}(w)\right\|^{2}}\sum_{k=1}^{K}\eta_{k}\left(w\right)\varepsilon_{k}^{T}\left(w\right)\frac{\partial\hat{y}_{k}\left(w\right)}{\partial w_{i}}\end{aligned} \tag{6.4}$$

It is noticed that, if λ_{c} is chosen as a very large value, the calculation of $e^{\lambda\left\|\varepsilon_{M}(w)\right\|^{2}}$ in each component $\frac{\partial J_{\lambda}(w)}{\partial w_{i}}$ of $\nabla J_{\lambda}(w)$ will result in a very large magnitude. In this case, $\nabla J_{\lambda}(w)$ would easily cause significant fluctuations for adjusting trainable weights in BP, if the learning rate of the gradient-based optimization method is not small enough. This phenomenon could further bring serious saturations of the activation function to most training nodes, and eventually lead the neural network training to fail. In order to avoid this issue, we normalize the RAE gradient in Eq. (6.3) as $\frac{\nabla J_{\lambda}(w)}{\left\|\nabla J_{\lambda}(w)\right\|}$, where $\left\|\cdot\right\|$ is the Euclidean norm. With such a normalization, all components of $\frac{\nabla J_{\lambda}(w)}{\left\|\nabla J_{\lambda}(w)\right\|}$ are bounded in $(-1,1)$, therefore it can be safely applied for BP.

6.1.2 Fast NRAE Gradient Evaluation

The NRAE training in the GDC method evaluates the first-order derivative of the NRAE criterion in Eq. (3.9) as

$$\frac{\partial C_{\lambda}\left(w\right)}{\partial w_{j}} = \frac{-2\sum_{k=1}^{K}\eta_{k}\left(w\right)\varepsilon_{k}^{T}\left(w\right)\frac{\partial\hat{y}_{k}(w)}{\partial w_{j}}}{\sum_{k=1}^{K}\eta_{k}\left(w\right)}$$

where the evaluation of $\eta_{k}(w)$ involves an exponential operation of $\varepsilon_{k}\left(w\right)$ and $\varepsilon_{M}\left(w\right)$ based on Eq. (3.7), costing the most computational resources during the training.

To improve the computation of Eq. (3.9) with a notational simplicity, we define

$$
\begin{aligned}
E_k(w) &:= \lambda \left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2 \right) \\
T_k(w) &:= \eta_k(w)\, \varepsilon_k^T(w)\, \frac{\partial \hat{y}_k(w)}{\partial w_i}
\end{aligned}
\tag{6.5}
$$

where $\|\varepsilon_k(w)\|^2 \leq \|\varepsilon_M(w)\|^2$ and $0 < \eta_k(w) \leq 1$.

Based on the calculation in Eq. (3.7), if $\|\varepsilon_k(w)\|^2 = \|\varepsilon_M(w)\|^2$, then $\eta_k(w) = 1$ and $\eta_k(w)$ is independent of λ. On the other hand, if $\|\varepsilon_k(w)\|^2 < \|\varepsilon_M(w)\|^2$, then the evaluation of $E_k(w)$ in Eq. (6.5) as λ approaches infinity provides

$$
\lim_{\lambda \to \infty} E_k(w) = -\infty \tag{6.6}
$$

which leads to

$$
\begin{aligned}
\lim_{\lambda \to \infty} \eta_k(w) &= \lim_{\lambda \to \infty} e^{E_k(w)} \\
&= \frac{1}{e^{\infty}}
\end{aligned}
\tag{6.7}
$$

Considering the numerical computation in computer, if λ is chosen as a very large value and $\|\varepsilon_k(w)\|^2 < \|\varepsilon_M(w)\|^2$ is satisfied, the calculation of $\eta_k(w)$ will provide a result with an extremely small magnitude that is close to zero. In other words, if $\eta_k(w) < F_{\text{min}}$, which implies $E_k(w) < \ln F_{\text{min}}$, the result of $\eta_k(w)$ will be rounded to zero during a numerical approximation in computer, where F_{min} is the smallest positive floating point number that is able to be handled by computer. In this case, $T_k(w) = 0$.

This phenomenon inspired us to develop an approach to fast evaluate the NRAE gradient

$$
\nabla C_\lambda(w) := \left[\frac{\partial C_\lambda(w)}{\partial w_i} \right]_{1 \times N}
$$

as defined in Eq. (3.2). This approach is described in Algorithm 5. With applying the ap-

proach to the NRAE training, the evaluation of $\eta_k(w)$ according to Eq. (3.7) is not always required for computing $\frac{\partial C_\lambda(w)}{\partial w_i}$ per training epoch, thus saving the computational costs significantly. Moreover, because the calculation of $T_k(w)$ in Eq. (6.5) can be safely ignored by direct setting $T_k(w) = 0$ when $E_k(w) < \ln F_{\text{min}}$, it effectively avoids massive computations of $\frac{\partial \hat{y}_k(w)}{\partial w_i}$ in BP, especially when the neural network has a very large number N of trainable weights.

Algorithm 5 Fast NRAE Gradient Evaluation

Require: Initialize the NRAE training with selecting the weight vector w randomly;

for $k = 1$ to K **do**
  if $\|\varepsilon_k(w)\|^2 = \|\varepsilon_M(w)\|^2$ **then**
    Set $\eta_k(w) \leftarrow 1$;
    $T_k(w) \leftarrow \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i}$;
  else
    $E_k(w) \leftarrow \lambda \left(\|\varepsilon_k(w)\|^2 - \|\varepsilon_M(w)\|^2 \right)$;
    if $E_k(w) < \ln F_{\text{min}}$ **then**
      Set $\eta_k(w) \leftarrow 0$ and $T_k(w) \leftarrow 0$;
    else
      $\eta_k(w) \leftarrow e^{E_k(w)}$;
      $T_k(w) \leftarrow \eta_k(w) \varepsilon_k^T(w) \frac{\partial \hat{y}_k(w)}{\partial w_i}$;
    end if
  end if
end for
for $i = 1$ to N **do**
  $\frac{\partial C_\lambda(w)}{\partial w_i} \leftarrow \frac{-2 \sum_{k=1}^K T_k(w)}{\sum_{k=1}^K \eta_k(w)}$;
end for
return $\nabla C_\lambda(w) \leftarrow \left[\frac{\partial C_\lambda(w)}{\partial w_i} \right]_{1 \times N}$

6.2 Experimental Settings

To experimentally verify the effectiveness of the EGDC method in training DNNs on a real-world dataset, we first evaluate it on LeNet-5 [64] for classifying handwritten digits

with the MNIST dataset. LeNet-5 is generally considered as a classic CNN model with deep architectures. It comprise 8 layers including 1 input layer, 3 convolutional layers, 2 pooling layers, 1 fully connected layer, and 1 output layer in sequence. In our experiments, we apply the same experimental settings, including the organization of training and testing samples, the architecture of LeNet-5, and the selection of convolutional and pooling operations, as described in [64] to evaluate the EGDC method in pairwise mode. Moreover, we perform another pairwise GDC training with using the same parameters as the EGDC training on LeNet-5 for demonstrating the advantages of EGDC compared to GDC. At last, the LMS training is applied to provide a baseline for measuring differences between distinct training methods under the same experimental settings. For the organization of the MNIST dataset, we perform the EGDC training, the GDC training, and the LMS training with the full 60,000 training samples and test the trained LeNet-5 models with the standard 10,000 testing samples.

To further demonstrate the advantage of the EGDC method comparing to more deep learning methods that are aided by unsupervised layer-wise pre-training, we perform EGDC simply in the supervised manner to train distinct MLPs on the MNIST dataset and compare the achieved test error rates to benchmark results reported in using MLPs, SAEs, DBNs, and DBMs under the same experimental settings. Particularly, we perform EGDC on one shallow MLP with the 784-1000-10 architecture, one deep MLP with the 784-500-1000-10 architecture, and another deep MLP with the 784-500-500-1000-10 architecture. To properly use the MNIST dataset as the same as the benchmark methods employed, we randomly choose 50,000 out of 60,000 samples to train MLPs, and apply the left 10,000 samples as the validation set to select the best performed weights with the lowest validation error. Then, we adopt the optimal weights to evaluate the 10,000 testing samples and provide the final test error rate.

As for training parameters of the EGDC and GDC methods, we set $\lambda = 10^4$, $TE = 10$,

$TS = 10^{-5}$, and $DR = 0.9$. For all the NRAE and RAE training sessions, we fix the global learning rate and the momentum term in SGD equal to 0.0001 and 0.5, respectively. For all the LMS training sessions, we apply the learning rate decay to decrease the global learning rate by 50% in every 10 training epochs, i.e., 0.0001 for the first 10 epochs, then 0.00005 for the next 10 epochs, and so on. The training session is considered as convergence once the training error rate in ten consecutive epochs is less than 0.01%. For all EGDC experiments, we set $F_{\text{max}} = 10^{300}$ and $F_{\text{min}} = 10^{-300}$.

6.3 Evaluation on Convolutional Neural Networks

6.3.1 Experiments with Training Method

Table 6.1 presents the training and test error rates obtained by applying different training methods to LeNet-5 on the MNIST dataset. The best result achieved by the EGDC method illustrates a test error rate of 0.90%, which is better than the famous benchmark result with a test error rate of 0.95% achieved by the Stochastic Diagonal Levenberg-Marquardt (SDLM) method on LeNet-5. Meanwhile, the benchmark result with the test error rate of 0.95% was obtained under a training error rate of 0.35%, but the EGDC method is able to bring the training error rate down to 0.23%, which is about 34% reduction of 0.35%, for obtaining the test error rate of 0.90%. Furthermore, both the EGDC and GDC training results are better than the LMS training results with the use of the same initial weights and training parameters. These observations indicate that the EGDC method has the capability to find a better solution with the training error lower than other training results located on different non-global local minima, while maintaining a generalization level better than others. We would like to stress that our experiments of EGDC are performed by using exactly the same network architecture of LeNet-5 as that being used to achieve the benchmark result (i.e., the test error rate of 0.95% reported in [64]). We did not

Table 6.1. Training and test error rates achieved by LeNet-5 on the MNIST dataset. Best performances achieved by the pairwise EGDC method are highlighted in bold.

Training Method	Error Rate (%)	
	Training	Test
LeNet-5 + LMS (this dissertation)	0.40	1.10
LeNet-5 + SDLM [64]	0.35	0.95
LeNet-5 + Pairwise GDC (this dissertation)	0.27	0.93
LeNet-5 + Pairwise EGDC (this dissertation)	**0.23**	**0.90**

find any reported results that have the training and test error rates better than ours under the same experimental conditions.

As an enhanced method of GDC, EGDC contains the main procedure of GDC but performs in a more efficient way. Therefore, the major effectiveness of GDC, which is confirmed in avoiding the non-global local minimum, could be maintained in EGDC. Although the EGDC method did not provide a notable improvement to the GDC method based on their test error rates shown in Table 6.1, EGDC costs significantly less training time than GDC. According to our estimation, LeNet-5 needs to be trained about 4 weeks by the GDC method to achieve the reported result, but the EGDC method can reduce almost a half of that training time to 15 days, which is also comparable to the LMS training in 10 days. This observation confirms that the EGDC method has the ability to significantly speedup the GDC method with the use of the GDC-RAE training and the fast NRAE gradient evaluation.

There is no doubt that much better results in training very deep CNNs can be achieved quite fast by using GPUs together with many well-known deep learning techniques, such as stochastic pooling, rectified linear units, and dropout. However, we stress that our experiments are performed without using GPU-acceleration, which can potentially reduce the training time of LeNet-5 from several days down to few hours though. Moreover, the

EGDC method consistently achieves better performances than the standard benchmark on LeNet-5 with employing neither additional tricks nor excessive trials of initial weights. Based on our experiments, it is fair to argue that the EGDC method has the potential to outperform more benchmark results with a faster computation, if EGDC is integrated with appropriate training tricks together with GPU-accelerated computing.

It is proper to mention that the benchmark result with the test error rate of 0.95% was achieved by the SDLM method on LeNet-5, which takes advantage of the diagonal terms of an estimation of the Gauss-Newton approximation to Hessian matrix [64]. It is noticed that both the EGDC and the GDC methods provide satisfactory performances better than the SDLM method, merely relying on SGD. This observation demonstrates that both the EGDC and GDC methods are better than or at least comparable to the SDLM method in training LeNet-5 with the MNIST dataset for achieving good optimization and generalization. Meanwhile, both the EGDC and GDC methods achieve satisfactory performances in pairwise mode better than the LMS method.

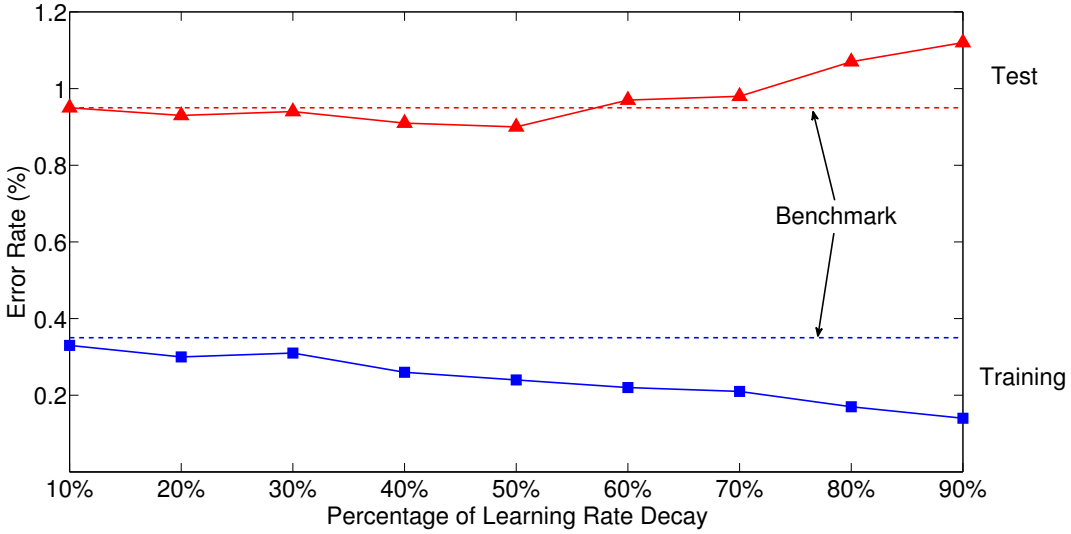
6.3.2 Experiments with Learning Rate Decay

As described in [64], a specific schedule of learning rate decay has been applied to train LeNet-5 for achieving a stable test error rate of 0.95%. This schedule chose an initial learning rate and gradually reduced it after certain training epochs until the learning rate is very small. Although it stated that the overfitting was not observed in training LeNet-5 with the specific schedule of learning rate decay, the reason of selecting that particular schedule has not been discussed in [64]. We expand our experiments through exploring the relation between the learning rate decay and the performance of EGDC as follows: we first choose 9 different percentages of learning rate decay to perform the EGDC training with the same initial weights, then we fix the percentage of learning rate decay and perform the EGDC training on 10 different sets of initial weights. In our experiments, the learning rate decay is

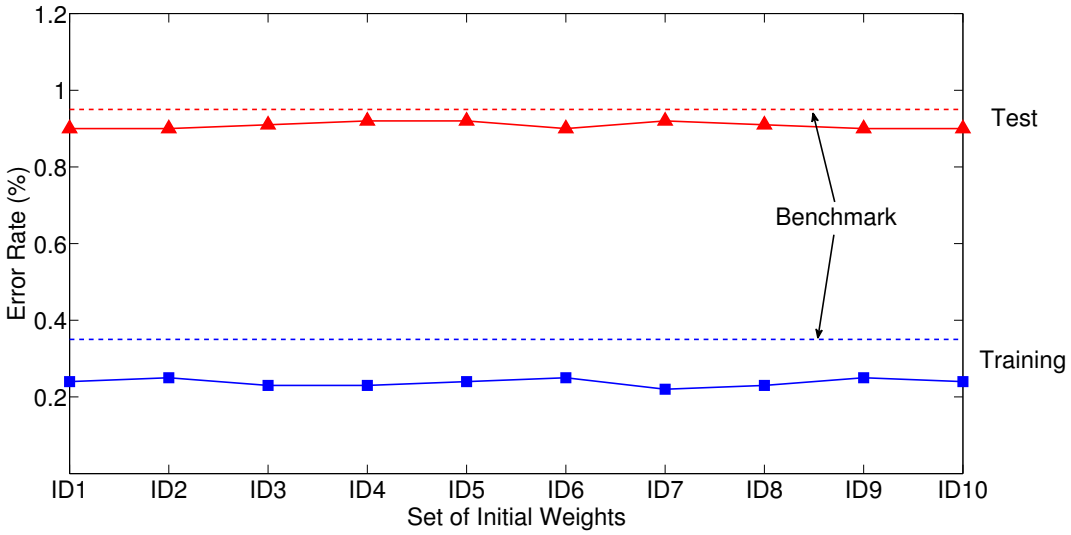
applied in every 10 epochs to the LMS training sessions in EGDC. Experimental results as presented in Fig. 6.1 demonstrate the variation of EGDC performances affected by distinct schedules of learning rate decay.

Generally, a learning process tends to find the optimal weights to approximate training samples, but it may easily escape from the ”sweet spot” for generalizing testing samples once the overfitting is occurred. Experimental results in Fig. 6.1(a) show that the test error rates achieved by small percentages of learning rate decay from 10% to 50% are smaller than the benchmark test error rate of 0.95%, while the 50% learning rate decay provides the lowest test error rate as 0.90%. Our explanation is that, if a large percentage of learning rate decay is applied, the corresponding learning rates will be kept as large numbers during the training. Thus, the learning procedure could suffer from many fluctuations in searching optimal weights and would converge on a solution with overfitting. In contrast, a small percentage of learning rate decay provides a quick drop of learning rates with less fluctuations. It can lock the optimal weights into the lower positions of wide open minima in the training error space, while stabilizing the learning procedure on good generalization levels with less overfitting.

Furthermore, the training sessions with 10 different sets of initial weights are performed to study whether the EGDC method is able to achieve good performances consistently when the learning rate decay is fixed as 50%. As illustrated in Fig. 6.1(b), the EGDC method achieves the training and test error rates in ranges of 0.22% - 0.25% and 0.90% - 0.92% over 10 different sets of initial weights, respectively. These results are consistently better than the benchmark results with the 0.35% training error rate and the 0.95% test error rate, implying that no multiple trials of initial weights are necessary for the EGDC method to achieve satisfactory performances.



(a) Learning rate decay with different percentages



(b) Learning rate decay with a fixed percentage (50%)

FIG. 6.1. Error rates of LeNet-5 achieved by the EGDC method on the MNST dataset. The blue color denotes the training result, while the red color shows the test result. Benchmark results of the 0.35% training error rate and the 0.95% test error rate are represented as dash lines in both figures.

6.4 Evaluation on Multilayer Perceptrons

The success of CNNs, such like LeNet-5, is benefited by a combination of convolution and pooling layers, where the combination captures a spatial topology that produces a rich feature representation of the input information with the weight sharing technique. Comparing to the traditional MLP trained by the non-convex error function, the CNN reduces the complexity of the error function in the parameter space by using significantly less trainable weights, which effectively lead the gradient-based method to converge on a better solution without serious overfitting. Because of the intrinsic advantages of CNNs, the high-dimensional non-convex optimization is not the vital issue in training CNNs, thereby the effectiveness of EGDC method for fighting the non-convex error function in training CNNs is limited.

However, many deep learning methods have focused on attacking the high-dimensional non-convex optimization problem by applying unsupervised layer-wise pre-training together with supervised fine-tuning to train DNNs, such as SAEs, DBNs, and DBMs. Therefore, to further demonstrate the effectiveness of the EGDC method comparing to deep learning methods that employ unsupervised pre-training, we perform EGDC to train distinct shallow and deep MLPs on the MNIST dataset merely in the supervised fashion and compare the test error rates achieved by these MLPs to the results achieved by architectures and methods of the benchmark training as presented in Table 6.2.

6.4.1 Comparison on MLP and SAE

Experimental results in Table 6.2 demonstrate that the EGDC method consistently achieves the lowest test error rates, i.e., 1.36% on the shallow MLP and 1.29% on the deep MLP, comparing to both the MSE and the cross-entropy (CE) method under the use of the same or similar benchmark architecture of MLPs. It indicates that the EGDC method,

Table 6.2. Test error rates achieved by shallow and deep neural networks on the MNIST dataset. Best performances achieved by the pairwise EGDC method are highlighted in bold.

Neural Network	Training Method	Test Error Rate (%)
Shallow Architecture	784-300-10 MLP + MSE [64]	4.70
	784-300-10 MLP + NRAE-MSE (this dissertation)	4.58
	784-1000-10 MLP + MSE [64]	4.50
	784-1000-10 MLP + MSE + Image distortions [64]	3.80
	784-300-10 MLP + MSE + Image distortions [64]	3.60
	784-300-10 MLP + Pairwise GDC (this dissertation)	2.61
	Shallow MLP + CE [3]	1.90
	784-1000-10 SAE + CE [90]	1.78
	784-1000-10 SAE + CE + Weight-decay regularization [90]	1.68
	784-1000-10 Denosing SAE + CE + Binary masking noise [90]	1.57
	784-1000-10 MLP + Pairwise EGDC (this dissertation)	**1.37**
	784-1000-10 RBM + Contrastive divergence [90]	1.30
Deep Architecture	784-300-100-10 MLP + MSE [64]	3.05
	784-500-150-10 MLP + MSE [64]	2.95
	784-300-100-10 MLP + MSE + Image distortions [64]	2.50
	784-500-150-10 MLP + MSE + Image distortions [64]	2.45
	Deep MLP + CE [3]	2.40
	Deep MLP + CE + Supervised pre-training [3]	2.00
	Deep SAE + CE + Unsupervised pre-training [3]	1.40
	784-500-1000-10 MLP + Pairwise EGDC (this dissertation)	1.31
	784-500-500-1000-10 MLP + Pairwise EGDC (this dissertation)	**1.29**
	Deep DBN + CE [3]	1.20
	784-500-500-1000-10 DBM + Discriminative fine-tuning [95]	1.01
	784-500-1000-10 DBM + Discriminative fine-tuning [95]	0.95

which contains the theoretical essences of both NRAE and RAE criteria in convexifying the non-convex error function, is better than the benchmark methods based on the MSE and the CE criteria in achieving the optimum with a good generalization performance. Moreover, the EGDC method is able to be applied to train both the shallow and deep MLPs, indicating that the method is completely applicable to train DNNs without any numerical and scaling-up problems.

The application of SAEs has been widely reported as a successful deep learning model for learning the high-level representation from the raw data thus generalizing the new data effectively, with involving unsupervised layer-wise pre-training followed by supervised fine-tuning. In our experiments, the shallow and deep MLPs trained by EGDC are compared to the SAEs trained by CE, demonstrating the significant effectiveness of the convexification and deconvexification methodology that is better than unsupervised pre-training approach in handling the high-dimensional non-convex optimization problems (i.e., training DNNs). Although unsupervised pre-training can guide the learning of SAEs towards attraction basins of the local minima that contain good generalization results from the training dataset, EGDC applies the proper convexification and deconvexification approach to the non-convex training criterion and provides an even better error space to reach the optimum with the superior generalization. Meanwhile, without the aid of unsupervised pre-training, the EGDC method merely needs to work in the supervised fashion, thus it is expected to be more efficient in computation.

It is worth noticing that the benchmark results reported in [3] are selected by different performances of the validation set with choosing the number of hidden nodes between 500 and 1000, where the shallow MLP has 1 hidden layer and the deep MLP/SAE/DBN has 3 hidden layers. We stress that our experimental results are not selected by performing the EGDC method on many MLP architectures then reporting the best. In fact, we only perform the EGDC method on each tested MLP with a single trial, then we report our experimental

results of MLPs and compare them to the benchmark results. It evidently implies that, with an adequate MLP architecture to fully express the training data and with a learning method like EGDC to effectively train the model, multiple trials on selecting different network architectures for the best performance are not necessary in training DNNs.

6.4.2 Comparison on DBN and DBM

As presented in Table 6.2, the shallow MLP trained by EGDC does not achieve a test error rate lower than the RBM trained by contrastive divergence, while the deep MLPs trained by EGDC also cannot provide test error rates better than the DBN trained by CE and the DBM trained with discriminative fine-tuning. Because our tests are correspondingly performed and compared to the benchmark results under the same experimental settings, the shallow and deep MLPs trained by EGDC have not demonstrated effective advantages comparing to the models of RBM, DBN, and DBM.

However, it is worth stressing that the generalization ability of our tested MLPs is restricted by the intrinsic nature of MLP as the discriminative learning model, but not by the EGDC method. Comparing to MLP, RBM is a generative model that is designed to express more complex relationships between the target and observed data as a full probabilistic model of variables, while DBN is basically composed of stacked RBMs, and DBM is intensively built with Boltzmann machines that are more generalizing than RBM. Although it has been claimed in [58, 84] that discriminative models are able to yield superior performances than generative models in some regression and classification tasks that do not require the joint distribution of the input data, recent research works in training DNNs [2, 5, 61, 100] implies that generative models are typically more flexible than discriminative models in expressing dependencies in complex learning tasks, especially with the aid of unsupervised layer-wise pre-training followed by supervised fine-tuning. Therefore, it is fairly to argue that the generative model, such as RBM, DBN, and DBM, has more ben-

efits than the discriminative model, such as MLP, in generalizing the MNIST dataset thus providing the lower test error rate under our experimental settings.

In fact, experimental results in Table 6.2 illustrate that the 784-1000-10 MLP trained by EGDC achieves a comparable generalization ability to the RBM trained with the same network architecture (i.e., 1.36% versus 1.30%), and the 784-500-500-1000-10 MLP trained by EGDC also presents a test error rate that is comparable to the DBN trained with the similar network setting (i.e., 1.29% versus 1.20%). However, the MLPs trained by MSE or CE can hardly compare to the performance achieved by EGDC under the same experimental settings. It demonstrates that the EGDC method has the ability to improve the generalization of discriminative model better than the MSE and the CE methods, while achieving the comparable performance to generative model such as RBM and DBN in the meantime. Therefore, it is reasonable to expect that, if the EGDC method is applied to generative models, it would effectively achieve the generalization performance better than other training methods, which do not apply the convexification and deconvexification methodology.

6.5 Summary

The EGDC method is proposed to improve the GDC method with the use of GDC-RAE training and the fast NRAE gradient evaluation for training DNNs. Since EGDC contains the main procedures of GDC but performs in a more efficient way, the major effectiveness of GDC, which is confirmed in avoiding the non-global local minimum and achieving a global or near-global minimum, is maintained in EGDC.

The theoretical essence of EGDC is derived from the NRAE and RAE criteria, which are proposed to alleviate the local minimum problem by convexifying the non-convex error function. With a proper use of the convexification and deconvexification methodology,

the non-convex optimization can be transformed to a convex optimization problem, while many difficulties in training the non-convex error criterion, such as local minima, large flat regions, and saddle points, are expected to be effectively alleviated or fundamentally solved by the convex optimization. Therefore, instead of applying a deeper network architecture or developing a tricky technique to perform deep learning, the proposed EGDC method provides a new perspective to solve the fundamental difficulties of the high-dimensional non-convex optimization in training DNNs.

Experimental results of training LeNet-5 with the MNIST dataset confirm that the EGDC method is able to reach the same accuracy level as the GDC method but with significantly less computational costs, while attaining good performances compared to the LMS training with the same experimental settings. Meanwhile, the EGDC method demonstrates its capability in training LeNet-5 with the MNIST dataset for achieving both training and test error rates better than those reported in benchmark results. To consistently achieve satisfactory performances, it is confirmed that multiple trials of initial weights are unnecessary for the EGDC method.

Furthermore, the shallow and deep MLPs trained by the EGDC method are evaluated merely in the supervised fashion with the MNIST dataset, illustrating significant advantages in achieving satisfactory generalization levels compared to many benchmark MLPs and SAEs that are trained by MSE or CE. Corresponding experiments indicate that the EGDC method based on the convexification and deconvexification methodology, which properly handles the non-convex error criterion, is suitable to train DNNs without the aid of unsupervised pre-training. More comparisons between the MLPs trained by EGDC and the DBN/DBM trained by CE present different performances caused by discriminative and generative models, demonstrating that the EGDC method can effectively improve the generalization of discriminative model better than the MSE and the CE methods.

Chapter 7

STATISTICAL NEURAL NETWORK PRUNING

This chapter introduces a statistical neural network pruning approach based on the hypothesis testing to improve the generalization of ANNs. Two pruning methods are developed with the essence of the statistical neural network pruning. They are experimentally studied by pruning ANNs after applying the EGDC method with the MNIST dataset, demonstrating that the statistical neural network pruning is not only able to be applied to enhance the generalization of shallow MLPs, but also applicable to improve the generalization of DNNs, such as CNNs and deep MLPs.

7.1 Background

In training ANNs, the optimal number of hidden nodes is hard to be determined before the training starts, while it is commonly estimated by the trial-and-error fashion. Many research works have been developed mainly in view of two strategies named as network growing and network pruning for deciding the size of hidden layers [26]. A network pruning strategy is the most important way to achieve an proper ANN, which uses less trainable parameters and thus has better generalization capability. This strategy first selects an ANN with a large number of hidden nodes, then it removes the redundant nodes during the training based on different approaches. The implementation of these approaches is

often categorized into two groups corresponding to the sensitivity-based methods and the penalty-based methods [16, 89]. Here, we only focus on the pruning strategy according to the sensitivity-based methods.

A sensitivity-based pruning method applies the relevance or sensitivity measure to quantify the contribution of trainable weights (or hidden nodes) in ANN for solving a task, where the less relevant or sensitive weights (or nodes) can be removed from the network. This method is often required to integrate with a network retraining manner [103]. For example, if two hidden nodes in a network produce the same or proportional outputs for a training dataset, one of them can be safely removed. Or, if several small weights are irrelevant in training the network, they should be pruned properly. Although it is difficult to decide which weights or nodes are least important for pruning the network, several techniques and heuristic approaches have been developed to resolve this issue, such as the iterative pruning algorithm for feedforward neural networks [14], the Karnin's pruning method [55], the pruning based on orthogonal transforms like the singular value decomposition and QR decomposition with column pivoting [54], the principal components pruning [69], and the method based on the perturbation analysis of the second-order Taylor expansion of the objective function like optimal brain damage (OBD) [66] and optimal brain surgeon (OBS) [36].

Particularly, the OBD and the OBS methods together with their variations are the most popular methods to prune ANNs through determining the saliency of weights with the aid of Hessian. However, pruning large ANNs with these methods involves intensive calculations of Hessian, which cost enormous amount of computational time and memory spaces on the real-world datasets in practice. A statistical neural network pruning method proposed in [77] described an innovated approach, which can reduce the total amount of computation in OBS with the use of a sensitivity measurement that is closely related to the saliency of weights defined by OBS under certain demonstrated conditions. In this chapter,

we study this statistical neural network pruning approach with more details in both theory and practice.

7.2 Statistical Neural Network Pruning

The basic idea of the statistical neural network pruning proposed in [77] is derived from a hypothesis testing, which is a standard procedure of statistical inferences applied for testing a statistical hypothesis. If the hypothesis testing declares a null and an alternative hypothesis as

$$\begin{aligned} H_0 &: \mu = 0 \\ H_1 &: \mu \neq 0 \end{aligned} \tag{7.1}$$

where μ is a population mean. If a sample mean x taken from the population that has a normal distribution with variance s^2, then the test z-statistic for the mean under H_0 is

$$z = \frac{x - \mu}{s} \tag{7.2}$$

which describes the distance of the sample mean x away from the population mean $\mu = 0$. A significance level is selected as a probability threshold to test the z-statistic, where the smaller significance level means the stronger evidence to reject the null hypothesis.

The z-statistic, which is applied to test the statistical hypothesis, is an estimation of each component of the weight vector in ANN. The z value is calculated by Eq. (7.2) with the use of the absolute value of weight as x and the estimation of s as

$$s = \sqrt{Q(w) \Big/ \left(\frac{\partial Q(w)}{\partial w_i} \right)^2} \tag{7.3}$$

where $Q(w)$ is the MSE criterion described in Eq. (2.2) and $\frac{\partial Q(w)}{\partial w_i}$ is the first-order derivative of the MSE criterion defined by

$$\frac{\partial Q\left(w\right)}{\partial w_i} = -\frac{2}{K}\sum_{k=1}^{K}\left\|y_k - \hat{f}\left(x_k, w\right)\right\|\frac{\partial \hat{f}\left(x_k, w\right)}{\partial w_i} \tag{7.4}$$

If the tested z value is greater than or equal to a chosen critical value z_{c}, such a z-statistic is regarded as the sufficient evidence to support that the evaluated component of the weight vector is not zero. Otherwise, the sensitivity of the evaluated weight is zero, which indicates that the corresponding connection associated with the evaluated weight in the network should be pruned.

7.2.1 Statistical Pruning Method

A statistical pruning (SP) method directly calculates z values for all components of the weight vector and prunes the weight if its corresponding z value is smaller than a critical value z_{c}. With choosing the proper z_{c}, the ANN after pruning is able to be retained in RT epochs for achieving a better generalization level with less number of weights comparing to the original network. The critical value z_{c} is commonly chosen as the value with respect to a small significance level in the complementary cumulative convention of the standard normal table (also referred to as the Z table), which gives a probability that a statistic is greater than z_{c}. For example, when we choose $z_{\mathrm{c}} = 2.00$, the corresponding significant level in the Z table is 4.55%, which is generally accepted as a sufficiently small probability in practice to reject the null hypothesis, if any evaluated z value is greater than or equal to the z_{c}. The SP method is described in Algorithm 6.

Algorithm 6 Statistical Pruning Method

Require: Obtain the weight vector w from the previously trained neural network, select a critical value z_{c}, and set RT;

for $i = 1$ to N **do** {Perform the pruning}
  Compute $z \leftarrow |w_i|/s$ where $s \leftarrow \sqrt{Q(w) \Big/ \left(\frac{\partial Q(w)}{\partial w_i}\right)^2}$;
  if $z < z_{\mathrm{c}}$ **then**
    Set $w_i \leftarrow 0$ and $flag_i \leftarrow 0$;
  else
    Set $flag_i \leftarrow 1$;
  end if
end for
Obtain the pruned weight vector w;
for $j = 1$ to RT **do** {Retrain the network}
  for $i = 1$ to N **do**
    if $flag_i = 1$ **then**
      Update w_i to ${w_i}^*$ using the first-order derivative of the objective function;
    else
      Set ${w_i}^* \leftarrow w_i$;
    end if
  end for
end for
return The optimal weight vector w^*.

7.2.2 Gradual Statistical Pruning Method

Although a proper critical value z_{c} is generally selected by a small significance level to reject the null hypothesis in practice, the ANN could be pruned incorrectly by an inappropriate selection of z_{c}. First, if z_{c} is too large, the linearization in the derivation of the statistical neural network pruning is invalid, leading to incorrect pruning results of the significant weights without the linearization or the critical statistic against the null hypothesis. Moreover, if z_{c} is too large, more sensitive weights with large z values could be removed excessively because of $z < z_{\mathrm{c}}$. It may cause the model underfitting, where the weights after pruning with the large z_{c} are insufficient to properly express the training data. On the other

hand, if z_c is too small, it could cause ineffective pruning, where the weights may not be pruned enough thus the overfitting of the original network is not properly alleviated.

Therefore, a gradual statistical pruning (GSP) method is proposed for choosing the best critical value z_c adaptively via repeating the pruning and retraining phases with the aid of a validation data. Like the SP method, a critical value z_c is initially selected to prune the trained network with the weight vector w_{old} and the error of the validation data v_{old}. Then, the network is retrained after pruning in RT epochs. In the GSP method, we always record the weight vector w_{new} according to the recently lowest validation error v_{new}, and then w_{new} is pruned by an updated critical value $z_c = z_c + z_{inc}$, where z_{inc} is an increment for gradually raising z_c as each pruning phase applies. The described pruning and retraining phases will repeat until the maximum critical value z_{max} or a satisfactory validation error v_{opt} is reached. The GSP method is described in Algorithm 7.

Algorithm 7 Gradual Statistical Pruning Method

Require: Obtain the weight vector w_{old} and its related validation error v_{old} from the previously trained neural network, choose a desired validation error v_{opt}, select z_{max} and z_{inc}, and set $z_c << z_{max}$;

while $z_c < z_{max}$ or $v_{old} > v_{opt}$ **do**
  Start SP with w_{old} and z_c;
  Record the lowest validation error v_{new} during the retraining phase in SP;
  Save the corresponding weight vector as w_{new};
  if $v_{new} < v_{old}$ **then**
    Let $w_{old} \leftarrow w_{new}$ and $v_{old} \leftarrow v_{new}$;
  end if
  $z_c \leftarrow z_c + z_{inc}$;
  Set $w^* \leftarrow w_{old}$;
end while
return The optimal weight vector w^*.

7.3 Experimental Evaluation

7.3.1 Experimental Settings

The effectiveness of the statistical neural network pruning method is evaluated by applying the SP and the GSP methods to remove redundant weights and to improve the generalization of the previously evaluated MLPs and CNNs with the MNIST dataset. For the experiments with shallow MLPs, we separately perform SP and GSP to prune ten 784-300-10 MLPs trained by GDC, denoting as sMLP-1, sMLP-2, ... , and sMLP-10 respectively. Moreover, for the experiments with CNNs, we separately perform SP and GSP to prune ten LeNet-5 models trained by EGDC, denoting as LN-1, LN-2, ..., and LN-10 respectively. At last, for the experiments with deep MLPs, we perform GSP to prune the 784-500-1000-10 and the 784-500-500-1000-10 MLPs trained by EGDC.

To perform SP, we first choose 15 critical values of z_{c} from 0.2 to 3.0 with an interval of 0.2. Then, we apply one critical value to prune one network each time, and we test all 15 critical values over 10 MLPs and 10 CNNs. At last, we perform the LMS training on each pruned network with setting the global learning rate and the momentum term as 0.0001 and 0.5 in SGD. In addition, the learning rate decay is applied to decrease the global learning rate by 50% in every 10 training epochs, and the training session is considered as convergence once the total number of retraining epochs $RT = 100$ is reached. To perform the experiments with the MNIST dataset, the full 60,000 training samples are applied to retrain the network, and the 10,000 testing samples are employed to test the retrained network after the corresponding session is converged.

For employing GSP, we choose $z_{\text{c}} = 0.2$, $z_{\text{max}} = 3.0$, $z_{\text{inc}} = 0.1$, $RT = 10$, and $v_{\text{opt}} = 0.80\%$ in our experiments. Since GSP provides the best pruned and retrained network with the aid of the validation data, the network does not need to convergence in the retraining phase after pruning. Accordingly, it would avoid excessive training time if the network is

large. Therefore, in each retraining phase, we always apply SGD by using the fixed global learning rate and the momentum term as 0.00001 and 0.5 without employing the learning rate decay. To properly use the MNIST dataset, we apply 50,000 out of 60,000 training samples to retain the network, and employ the left 10,000 samples as the validation data to maintain the network with the lowest validation error. After that, the pruned network selected by the best validation error is tested by 10,000 test samples, with providing the final test error rate.

7.3.2 Experiments with Shallow Multilayer Perceptrons

Evaluating the SP Method In this section, the SP method is evaluated on ten 784-300-10 MLPs from sMLP-1 to sMLP-10 with choosing 15 different critical values. We present one experiment in sMLP-1 and show the results in Fig. 7.1 as an example to demonstrate the performance of SP on the shallow MLPs. Based on our experiments, the other 784-300-10 MLPs from sMLP-2 to sMLP-10 have the similar performances of SP as we state below:

1. Experimental results presented in Fig. 7.1(a) demonstrate that the SP method is able to achieve a new test error rate, which is lower than or equal to the original test error rate, when $z_{\mathrm{c}} < 1.8$. As presented in Table 7.1, the best test error rate 2.49%, which is lower than the original test error rate 2.61%, is achieved by the SP method when $z_{\mathrm{c}} = 1.8$. It confirms the effectiveness of SP in improving the generalization of the shallow MLP with the use of the statistical neural network pruning method under the proper critical value of z_{c}.

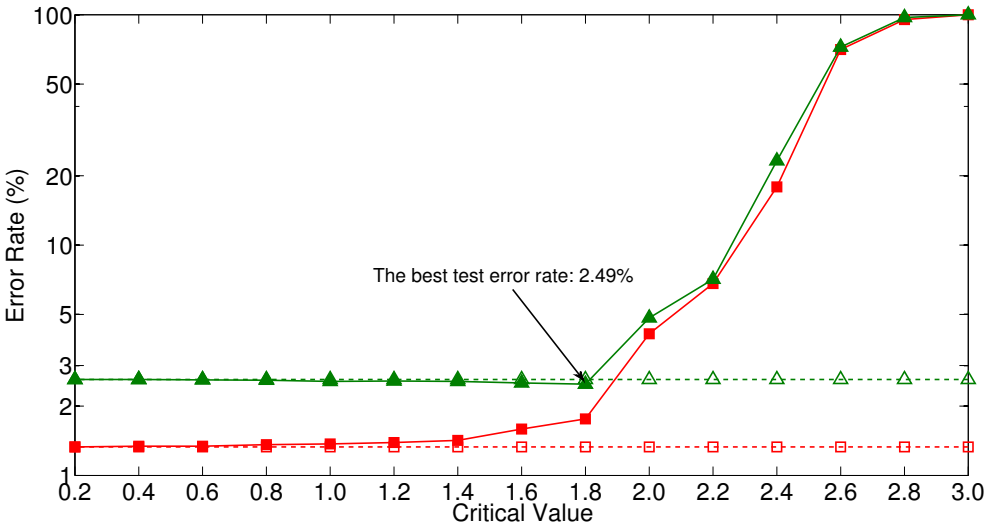
2. For the experimental results after applying the SP method with $z_{\mathrm{c}} > 1.8$, we observe several unusual large training and test error rates in Fig. 7.1(a). The reason can be explained by the curve of pruning percentages illustrated in Fig. 7.1(b). It

Table 7.1. Test error rates achieved by the 784-300-10 MLP classifiers on the MNIST dataset. The best test error rate of the MLP classifier trained by the pairwise GDC method and then pruned by the SP method is highlighted in bold.

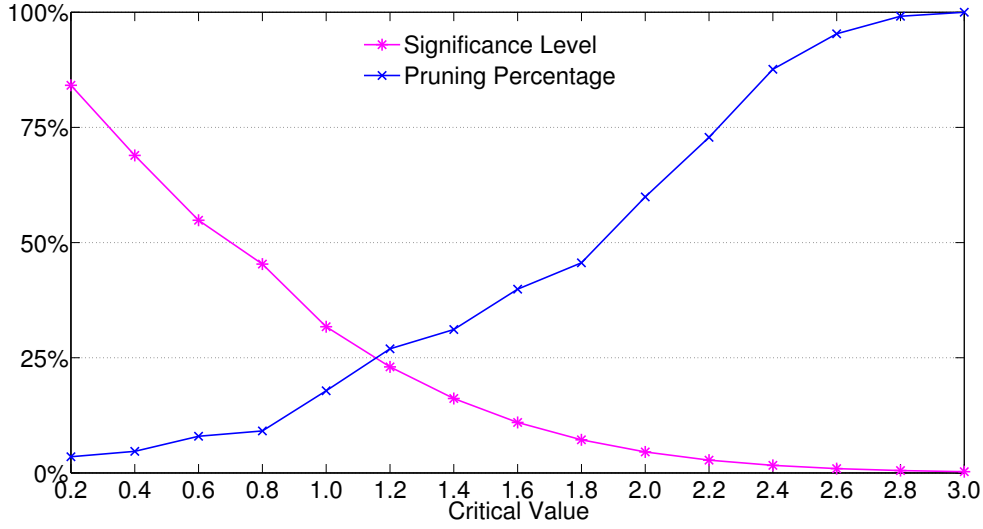
Training Method	Test Error Rate (%)
784-300-10 MLP + MSE [64]	4.70
784-300-10 MLP + NRAE-MSE (this dissertation)	4.58
784-300-10 MLP + MSE + Image distortions [64]	3.60
784-300-10 MLP + Pairwise GDC (this dissertation)	2.61
784-300-10 MLP + Pairwise GDC + SP (this dissertation)	**2.49**

demonstrates that the large critical values with $z_{\mathrm{c}} > 1.8$ yield intensive pruning percentages that are larger than 50%, where such large pruning percentages can easily cause underfitting in retraining the 784-300-10 MLPs. Without a proper ability to express the training error space, the model underfitting brings a poor training result together with an inappropriate generalization. In order to avoid any possible model underfitting, too large critical values should not be taken to perform SP.

3. In Fig. 7.1(a), we merely observe slight improvement of the original test error rate when $z_{\mathrm{c}} < 1.0$. The reason is that such a small critical value often associates with a small pruning percentage, which is less than 20%. It indicates that if the pruning percentage is selected too small to apply SP, the effectiveness of reducing the network redundancy and improving the generalization could be limited, because such a small pruning percentage is not enough to properly remove overfitting of the original network. Nevertheless, it is worth stressing that, even though choosing a small critical value of z_{c} in SP may not produce a satisfactory pruning result, it does not lead to a performance worse than the original network after all.



(a) Error rates according to 15 critical values



(b) Significance levels and pruning percentages according to 15 critical values

FIG. 7.1. Experimental results of evaluating the SP method on sMLP-1 with 15 critical values. In Fig. 7.1(a), the training and test results are marked as red squares and green triangles, while the results before and after applying SP are denoted by dash and solid lines, respectively. The original training and test error rates of sMLP-1 before applying SP are 1.33% and 2.61%.

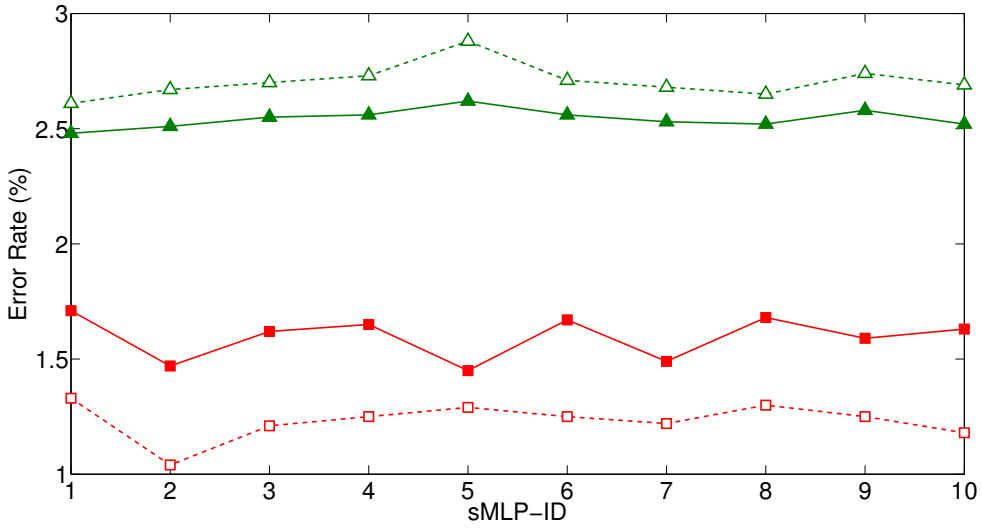
Evaluating the GSP Method Unlike the SP method, GSP adaptively chooses the best critical value during the pruning, thus it does not need to be tested with choosing different critical values of z_c. The GSP method is evaluated on the same ten 784-300-10 MLPs from sMLP-1 to sMLP-10 as we applied for the evaluation of SP. Experimental results are presented in Fig. 7.2.

Fig. 7.2(a) demonstrates that the GSP method applied on ten 784-300-10 MLPs consistently achieves the new test error rates, which are lower than the original test error rates achieved by the shallow MLPs. Since the GSP method is evaluated with applying the same parameters as the evaluation of shallow MLPs adopt, our obtained results confirm the effectiveness of GSP for improving the generalization of the shallow MLPs with the use of adaptive critical values. Comparing to the SP method, GSP removes the requirement of selecting the proper critical value of z_c in pruning a network, thus avoiding too large or too small critical values to be chosen during the pruning.

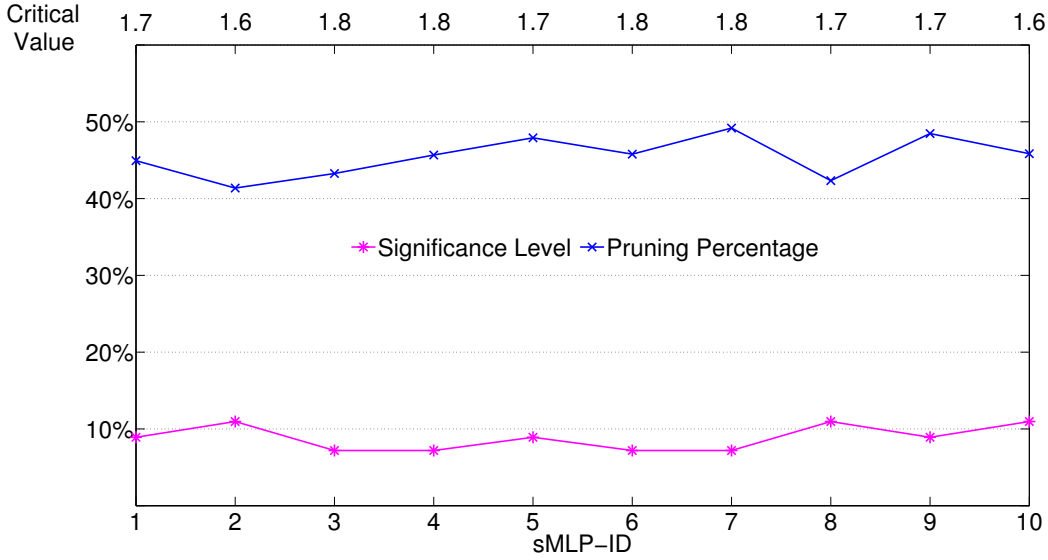
Furthermore, it is observed in Fig. 7.2(b) that the experimental results achieved by the GSP method are associated with the critical values of z_c in the range between 1.6 and 1.8, the significance levels close to 10%, and the pruning percentages in the range between 40% and 50%. These results are comparable to the best test error rate achieved by the SP method with choosing the critical value $z_c = 1.8$ as shown in Fig. 7.1. It illustrates that the GSP method has the ability to preserve the best performance of the SP method, thus reaching the same generalization level as SP achieves without selecting the proper critical value of z_c by trial and error in pruning shallow MLPs.

7.3.3 Experiments with Convolutional Neural Networks

Evaluating the SP Method In this section, the SP method is evaluated on ten LeNet-5 models from LN-1 to LN-10 with choosing 15 different critical values. We present one experiment in LN-1 and show the results in Fig. 7.3 as an example to demonstrate the



(a) Error rates according to ten sMLPs



(b) Significance levels and pruning percentages according to ten sMLPs

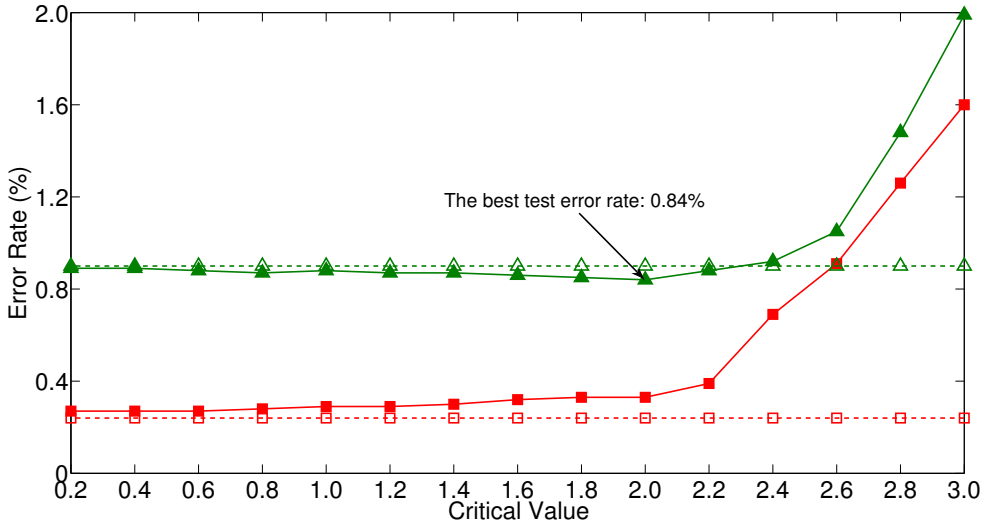
FIG. 7.2. Experimental results of evaluating the GSP method on ten 784-300-10 MLPs. In Fig. 7.2(a), the training and test results are marked as red squares and green triangles, while the results before and after applying GSP are denoted by dash and solid lines, respectively. In Fig. 7.2(b), the exact critical values which produce the corresponding results are shown on top of the figure.

performance of SP on the CNNs. Based on our experiments, the other tests of the CNNs from LN-2 to LN-10 behave the similar performances of SP as we state below:

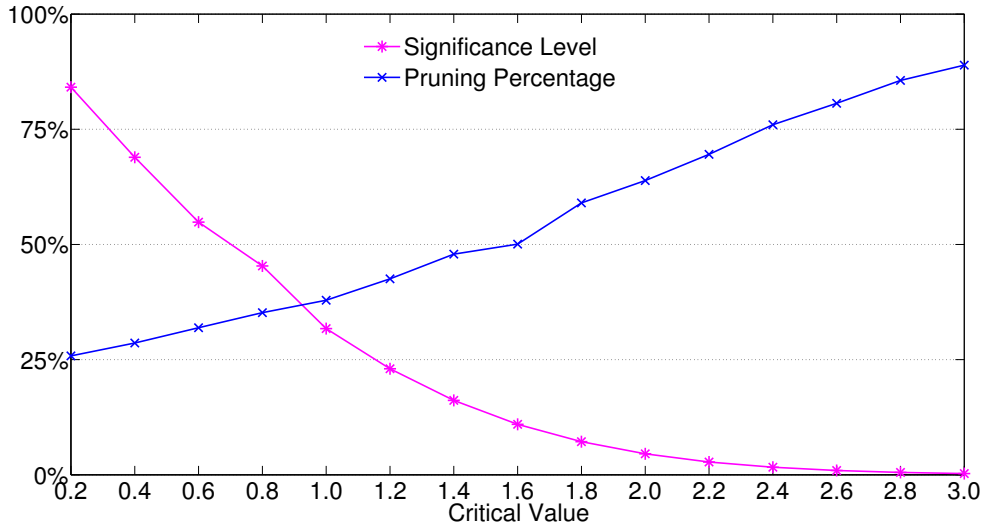
1. Experimental results shown in Fig. 7.3(a) demonstrate that the SP method is able to achieve a new test error rate, which is lower than or equal to the original test error rate, when $z_{\mathrm{c}} < 2.2$. As presented in Table 7.2, the best test error rate 0.84%, which is lower than the original test error rate 0.90%, is achieved by the SP method when $z_{\mathrm{c}} = 2.0$. Moreover, the test error rate of 0.84% is also close and comparable to a benchmark result 0.80% achieved by applying a huge training data distortion. It confirms the effectiveness of SP in improving the generalization of the CNN with the use of the statistical neural network pruning under proper critical value of z_{c}.

2. As similar as what we observed in pruning the shallow MLPs, several unusual large training and test error rates appear after applying SP with $z_{\mathrm{c}} > 2.2$. The reason can be explained by the curve of pruning percentages illustrated in Fig. 7.3(b). It demonstrates that the large critical values with $z_{\mathrm{c}} > 2.2$ yield intensive pruning percentages that are larger than 70%, where such large pruning percentages can easily cause underfitting in retraining the LeNet-5. It further confirms that too large critical values should not be taken to perform SP for avoiding any possible model underfitting. Furthermore, for the achieved test error rates according to small pruning percentages closed to 25% as shown in Fig. 7.3(b), we only observe small improvement of the original test error rate, because the overfitting of the original network is not properly removed by small critical values of z_{c}.

Evaluating the GSP Method In this section, the GSP method is evaluated on the same ten LeNet-5 models from LN-1 to LN-10 as we applied for the evaluation of SP, and the experimental results are presented in Fig. 7.4.



(a) Error rates according to 15 critical values



(b) Significance levels and pruning percentages according to 15 critical values

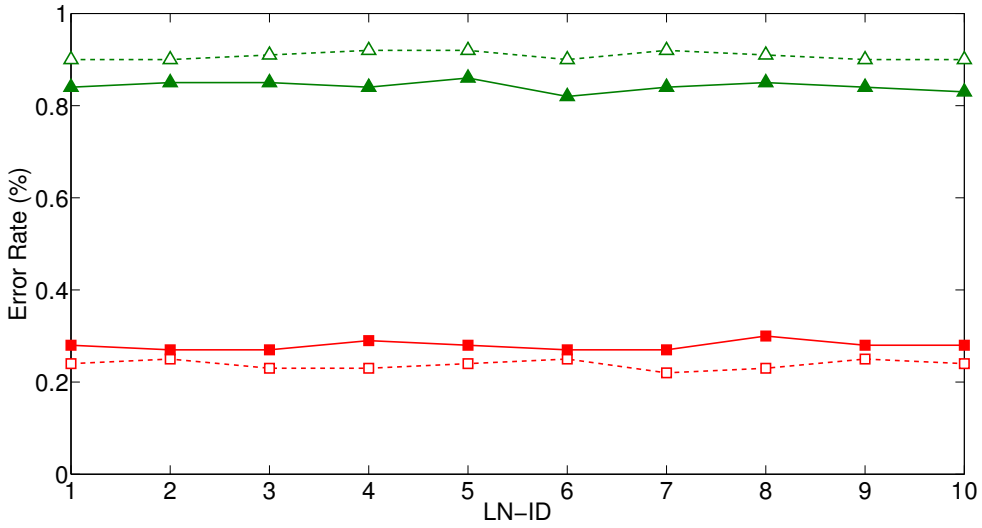
FIG. 7.3. Experimental results of evaluating the SP method on LN-1 with 15 critical values. In Fig. 7.3(a), the training and test results are marked as red squares and green triangles, while the results before and after applying SP are denoted by dash and solid lines, respectively. The original training and test error rates of LN-1 before applying SP are 0.24% and 0.90%.

Table 7.2. Training and test error rates achieved by LeNet-5 on the MNIST dataset. The best test error rate of LeNet-5 trained by the pairwise EGDC method and then pruned by the SP method is highlighted in bold.

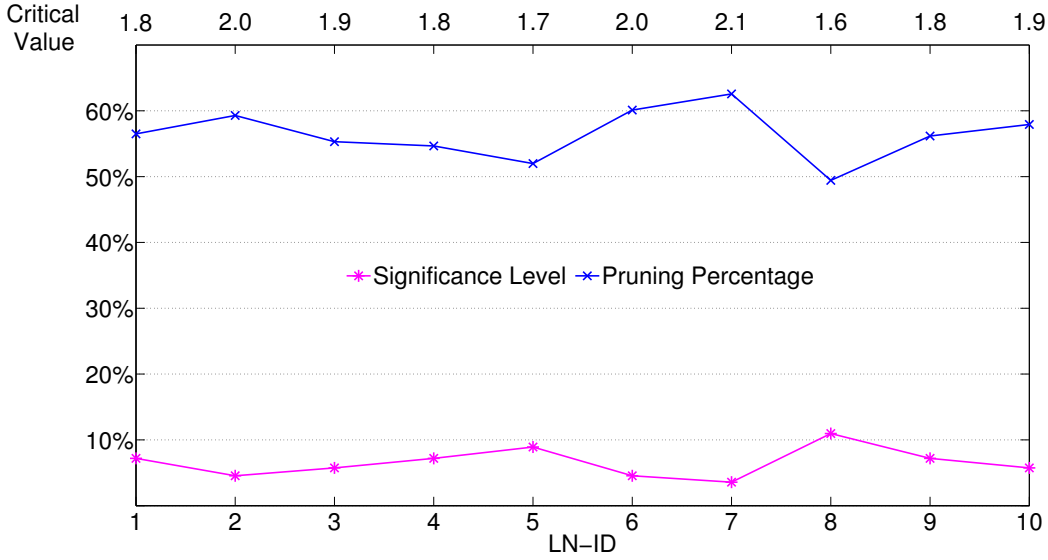
Training Method	Error Rate (%)	
	Training	Test
LeNet-5 + LMS (this dissertation)	0.40	1.10
LeNet-5 + SDLM [64]	0.35	0.95
LeNet-5 + Pairwise GDC (this dissertation)	0.27	0.93
LeNet-5 + Pairwise EGDC (this dissertation)	0.23	0.90
LeNet-5 + Pairwise EGDC + GSP (this dissertation)	0.28	**0.84**
LeNet-5 + SDLM + Image distortions [64]	N/A	0.80

Fig. 7.4(a) demonstrates that the GSP method applied on ten LeNet-5 models consistently achieve the new test error rates, which are lower than the original test error rates achieved by the LeNet-5. Since the GSP method is evaluated with applying the same parameters as the evaluation of LeNet-5 employs, the experimental results confirm the the effectiveness of GSP for improving the generalization of CNNs with the use of adaptive critical values.

Furthermore, it is observed in Fig. 7.4(b) that the results achieved by GSP are associated with the critical values of z_{c} in the range between 1.8 and 2.1, the significance levels close to 10%, and the pruning percentages in the range between 50% and 60%. These results are comparable to the best test error rate achieved by the SP method with choosing the critical value $z_{\mathrm{c}} = 2.0$ as shown in Fig. 7.4. It further confirms that GSP has the ability to preserve the best performance of SP and reach the same generalization level as SP does, without manually selecting the proper critical value of z_{c} in pruning CNNs.



(a) Error rates according to ten LeNet-5 models



(b) Significance levels and pruning percentages according to ten LeNet-5 models

FIG. 7.4. Experimental results of evaluating the GSP method on ten LeNet-5 models. In Fig. 7.4(a), the training and test results are marked as red squares and green triangles, while the results before and after applying GSP are denoted by dash and solid lines, respectively. In Fig. 7.4(b), the exact critical values which produce the corresponding results are shown on top of the figure.

Table 7.3. Test error rates achieved by deep neural networks on the MNIST dataset. The best performance of the deep MLP trained by the pairwise EGDC method and then pruned by the GSP method is highlighted in bold.

Training Method	Test Error Rate (%)
784-300-100-10 MLP + MSE [64]	3.05
784-500-150-10 MLP + MSE [64]	2.95
784-300-100-10 MLP + MSE + Image distortions [64]	2.50
784-500-150-10 MLP + MSE + Image distortions [64]	2.45
Deep MLP + CE [3]	2.40
Deep MLP + CE + Supervised pre-training [3]	2.00
Deep SAE + CE + Unsupervised pre-training [3]	1.40
784-500-1000-10 MLP + Pairwise EGDC (this dissertation)	1.31
784-500-1000-10 MLP + Pairwise EGDC + GSP (this dissertation)	1.29
784-500-500-1000-10 MLP + Pairwise EGDC (this dissertation)	1.29
784-500-500-1000-10 MLP + Pairwise EGDC + GSP (this dissertation)	**1.27**
Deep DBN + CE [3]	1.20
784-500-500-1000-10 DBM + Discriminative fine-tuning [95]	1.01
784-500-1000-10 DBM + Discriminative fine-tuning [95]	0.95

7.3.4 Experiments with Deep Multilayer Perceptrons

Since the SP method requires to select the proper critical value of z_c to achieve the best performance, and it will greatly cost computational efforts to evaluate SP in pruning deep MLPs. In fact, based on what we observed from our previous experiments in pruning shallow MLPs and CNNs, the GSP method can perform at least as good as the SP method without manually selecting the proper critical value of z_c for the best result. Therefore, we only apply the GSP method to prune the 784-500-1000-10 and the 784-500-500-1000-10 MLPs in this section for evaluating the performance of the statistical neural network pruning approach on the deep MLPs.

Table 7.3 shows the performances of two deep MLPs trained by the pairwise EGDC

method and then pruned by the GSP method. It demonstrates the effectiveness of GSP in achieving the lower test error rate of deep MLPs with the use of adaptive critical values. Particularly, the test error rate of 1.29% and 1.27% are achieved by the critical value of $z_c = 1.8$ and $z_c = 2.0$, the significance level of 7.19% and 4.55%, the pruning percentage of 20% and 30%, respectively. Although the best test error rate that is achieved by the 784-500-500-1000-10 MLP with performing the pairwise EGDC training and GSP cannot outperform the test error rates achieved by the DBN or the DBM, the effectiveness of GSP in pruning deep MLPs for a better generalization with a smaller network architecture is certainly verified.

7.4 Summary

A statistical neural network pruning approach based on the hypothesis testing is implemented and evaluated as the SP and the GSP methods to enhance the generalization of ANNs. Experimental results of pruning MLPs and CNNs on the MNIST dataset with separately employing SP and GSP successfully demonstrate the effectiveness of the statistical neural network pruning approach in improving the generalization of MLPs and CNNs. Furthermore, comparing to the SP method, the GSP method removes the requirement of selecting a proper critical value to perform a pruning, thus avoiding too large or too small critical values to be chosen during the pruning. Meanwhile, it confirms that GSP has the ability to preserve the best performance of SP and reach the same generalization level as SP does, without selecting the proper critical value by trial and error in pruning MLPs and CNNs.

Chapter 8

CONCLUSION AND FUTURE WORK

8.1 Conclusion

The local minimum problem has attracted much attention since its occurrence for decades in training artificial neural networks (ANNs) with backpropagation (BP). Nowadays, this conundrum still impedes the development and the application of ANNs via affecting the performance of BP with the presence of local minima in addition to global minima. Although numerous learning methods have been developed to avoid or alleviate poor local minima by using a wide array of training techniques together with practical heuristics, the fundamental difficulty caused by the use of the non-convex objective function in training ANNs has not been solved thoroughly. Even poor local minima are rarely a serious problem in training neural networks with deep architectures, the intrinsic quality of local minimum still leads to more severe difficulties corresponding to the high-dimensional non-convex optimization in training deep neural networks (DNNs).

This dissertation proposes a series of methodologies with applying convexification and deconvexification to avoid non-global local minima and achieve the global or near-global minima with satisfactory optimization and generalization performances. The theoretical essence of the convexification and deconvexification methodologies is developed based on the normalized risk-averting error (NRAE) criterion, which can be properly applied to

handle the fundamental difficulty of the local minimum problem by transforming a non-convex optimization to a convex optimization in a certain convexity region.

The NRAE criterion is derived from its predecessor, the risk-averting error (RAE) criterion, with resolving the difficulty of computational overflow in practically applying the RAE criterion to convexify the standard mean squared error (MSE) criterion. The theoretical foundation of the NRAE criterion is justified by its bounded computation and convexification property, indicating the ability to avoid or alleviate non-global local minima in training the ANN with an non-convex objective function. Meanwhile, the effectiveness of the NRAE criterion is evaluated by the NRAE training and the NRAE-MSE training methods, which are developed according to the convexification methodology. These methods demonstrate significantly better performances comparing to the MSE method in training multilayer perceptrons (MLPs) for approximating functions with poor non-global local minima, illustrate the benefit of handling the local minimum problem in using the convexification methodology based on the NRAE criterion.

Although the convexification methodology works well in solving problems with involving non-global local minima, it is constrained by properly selecting distinct parameters of λ for applying the NRAE criterion to different tasks. Since the convexification property of the NRAE criterion is limited by a stagnant problem, it hinders the convexity region to be expanded thoroughly by using a fixed but a very large λ. However, according to a deconvexification methodology, the stagnant problem can be overcome by gradually reducing λ to control the expansion of the convexity region for employing the NRAE criterion in practice. Following this idea, a gradual deconvexification (GDC) method is developed. Based on the experiments of approximating functions with non-global local minima and recognizing handwritten digits with the well-know MNIST dataset, the GDC method presents a capability that is superior to the MSE method in training MLPs to achieve satisfactory optimization and generalization performances.

To evaluate the effectiveness of the NRAE criterion in challenging real-world tasks when the problem size becomes larger as well as the neural network has deep architectures in practice, an enhanced gradual deconvexification (EGDC) method is developed to handle the high-dimensional non-convex optimization with a large dataset in training deep neural networks. Under the same experimental settings, the EGDC method outperforms many benchmark approaches in training convolutional neural networks (CNNs) and deep MLPs with the MNIST dataset for better performances, without adopting excessive trials of initial weights, network architectures, and even without the aid of unsupervised pre-training. Such the results illustrate that the non-convex optimization can be transformed to a convex optimization problem with the proper use of the convexification and deconvexification methodology. As a consequence, many difficulties of using the high-dimensional non-convex error criterion, such as local minima, large flat regions, and saddle points, are expected to be effectively alleviated or fundamentally solved by the convex optimization in training DNNs.

Last but not least, a statistical neural network pruning approach based on the hypothesis testing is developed to improve the generalization of ANNs. Experimental results indicate that the statistical pruning can be applied to remove the redundancy of weights in training MLPs and CNNs with achieving better generalization results.

8.2 Future Work

With the development of the high performance computing on graphics processing units (GPUs), many deep learning frameworks, such as Theano, TensorFlow, Torch, Caffe, MXNet, etc., are tremendously benefited by applying GPU-acceleration instead of the traditional CPU-based computation to train DNNs. Although training with the NRAE criterion presents remarkable performances comparing to many benchmark results in solving

function approximation and data classification tasks, the NRAE-based training method is merely based on the old fashion of central processing unit (CPU) computing without GPU-acceleration. Since GPU-accelerated computing is expected to deliver approximately the same accuracy as the standard CPU computing, the NRAE-based method is expected to potentially achieve a much faster computation with the aid of GPU.

In order to acquire a fast computing speed, we integrate the NRAE-based training methods into Theano with GPU-acceleration for training both shallow and deep MLPs. Experimental results demonstrate significant improvement of training speed compared to our CPU-based framework. Our next future work will focus on integrating the NRAE-based training methods into Theano or Caffe for training CNNs with very deep architectures (e.g., a CNN with more than 10 non-linear layers) and challenging more complicated datasets (e.g., CIFAR-10/CIFAR-100, SVHN, ImageNet, etc.).

Bibliography

[1] Aarts, E., and Korst, J. 1989. *The Neuron*. Oxford, UK: Oxford University Press.

[2] Bengio, Y., and LeCun, Y. 2007. Scaling learning algorithms towards AI. In *Large-Scale Kernel Machines*. Cambrigde, MA, USA: MIT Press.

[3] Bengio, Y.; Lamblin, P.; Popovici, D.; and Larochelle, H. 2007. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*, 153–160. Cambrigde, MA, USA: NIPS.

[4] Bengio, Y. 1996. *Neural Networks for Speech and Sequence Recognition*. New York, NY: International Thomson Computer Press.

[5] Bengio, Y. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning* 2(1):1–127.

[6] Bennett, K. P., and Campbell, C. 2000. Support vector machines: Hype or hallelujah? *SIGKDD Explorations* 2(2):1–13.

[7] Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. New York, NY: Springer.

[8] Blake, A., and Zisserman, A. 1987. *Visual Reconstruction*. Cambrigde, Massachussetts, USA: The MIT Press.

[9] Bland, R., and University of Stirling, Department of Computing Science and Mathematics. 1998. Learning XOR: Exploring the space of a classic problem. Technical report, Department of Computing Science and Mathematics, University of Stirling.

[10] Blum, A. L., and Rivest, R. L. 1992. Training a 3-node neural network is NP-complete. *Neural Networks* 5:117–127.

[11] Blum, E. K. 1989. Approximation of boolean functions by sigmoidal networks: Part 1: Xor and other two-variable functions. *Neural Computation* 1(4):532–540.

[12] Bourlard, H., and Kamp, Y. 1988. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics* 59(4-5):291–294.

[13] Broyden, C. G. 1970. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications* 6:76–90.

[14] Castellano, G.; Fanelli, A. M.; and Pelillo, M. 1997. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks* 8(3):519–531.

[15] Černý, V. 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45:41–51.

[16] Chandrasekaran, H.; Chen, H.-H.; and Manry, M. T. 2000. Pruning of basis functions in nonlinear approximators. *Neurocomputing* 34(1-4):29–53.

[17] Choromanska, A.; Henaff, M.; Mathieu, M.; Arous, G. B.; and LeCun, Y. 2015. The loss surfaces of multilayer networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, 192–204.

[18] Cireşan, D. C.; Meier, U.; Masci, J.; Gambardella, L. M.; and Schmidhuber, J. 2011. Flexible, high performance convolutional neural networks for image classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence - Volume Volume Two*, volume 2, 1237–1242.

[19] Cireşan, D. C.; Meier, U.; and Schmidhuber, J. 2012. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3642–3649.

[20] Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*, 160–167.

[21] Colorni, A.; Dorigo, M.; and Maniezzo, V. 1991. Distributed optimization by ant colonies. In *Actes de la première conférence européenne sur la vie artificielle*, 134–142.

[22] Dauphin, Y. N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; and Bengio, Y. 2014. Identifying and attacking the saddle point problem in high-dimensional nonconvex optimization. In *Advances in Neural Information Processing Systems*, 2933–2941.

[23] Demuth, H., and Beale, M. 1994. *Neural Network Toolbox User's Guide*. Natick, MA: MathWorks, Inc.

[24] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255.

[25] Dorigo, M. 1992. *Optimization, Learning and Natural Algorithms*. Ph.D. Dissertation, Politecnico di Milano.

[26] Du, K.-L., and Swamy, M. 2006. *Neural Networks in a Softcomputing Framework*. New York, NY: Springer.

[27] Erhan, D.; Bengio, Y.; Courville, A.; Manzagol, P.; Vincent, P.; and Bengio, S. 2010. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* 11:625–660.

[28] Fausett, L. 1994. *Fundamentals of Neural Networks*. Englewood Cliffs, New Jersey: Prentice Hall.

[29] Fischer, A., and Igel, C. 2014. Training restricted boltzmann machines: An introduction. *Pattern Recognition* 47(1):25–39.

[30] Fletcher, R. 1970. A new approach to variable metric algorithms. *Computer Journal* 13(3):317–322.

[31] Fukushima, K., and Miyake, S. 1982. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition* 15(6):455–469.

[32] Fukushima, K. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics* 36(4):193–202.

[33] Goldfarb, D. 1970. A family of variable metric updates derived by variational means. *Mathematics of Computation* 24(109):23–26.

[34] Gui, Y.; Lo, J. T.-H.; and Peng, Y. 2014. A pairwise algorithm for training multilayer perceptrons with the normalized risk-averting error criterion. In *Proceedings of the 2014 International Joint Conference on Neural Networks*, 358–365.

[35] Hadsell, R.; Erkan, A.; Sermanet, P.; Scoffier, M.; Muller, U.; and LeCun, Y. 2008. Deep belief net learning in a long-range vision system for autonomous off-road driving. In *Proceedings of the 2008 International Conference on Intelligent Robots and Systems*, 628–633.

[36] Hassibi, B.; Stork, D. G.; and Wolff, G. J. 1992. Optimal brain surgeon and general network pruning. In *Proceedings of the 1993 IEEE International Conference on Neural Networks*, 293–299.

[37] Hassoun, M. H. 1995. *Fundamentals of Artificial Neural Networks*. Cambridge, Massachusetts: MIT Press.

[38] Haykin, S. 2008. *Neural Networks and Learning Machines*. Upper Saddle River, New Jersey: Prentice Hall, third edition.

[39] Hestenes, M. R., and Stiefel, E. 1952. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards* 49(6):409–436.

[40] Hinton, G. E., and Salakhutdinov, R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.

[41] Hinton, G. E., and Salakhutdinov, R. 2012. A better way to pretrain deep boltzmann machines. In Pereira, F.; Burges, C.; Bottou, L.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc. 2447–2455.

[42] Hinton, G. E., and Zemel, R. S. 1994. Autoencoders, minimum description length and helmholtz free energy. In Cowan, J.; Tesauro, G.; and Alspector, J., eds., *Advances in Neural Information Processing Systems 6*. Morgan-Kaufmann. 3–10.

[43] Hinton, G. E.; Osindero, S.; and Teh, Y.-W. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18(7):1527–1554.

[44] Hinton, G. E. 2002. Training product of experts by minimizing contrastive divergence. *Neural Computation* 14(8):1771–1800.

[45] Hinton, G. E. 2012. A practical guide to training restricted boltzmann machines. *Lecture Notes in Computer Science in Neural Networks: Tricks of the Trade* 7700:599–619.

[46] Hirose, Y.; Yamashita, K.; and Huiya, S. 1991. Back-propagation algorithm which varies the number of hidden units. *Neural Networks* 4:61–66.

[47] Hubel, D. H., and Wiesel, T. N. 1962. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology* 160(1):106–154.

[48] Hubel, D. H., and Wiesel, T. N. 1968. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology* 195(1):215–243.

[49] Ingber, L. 1989. Very fast simulated re-annealing. *Mathematical and Computer Modelling* 12(8):967–973.

[50] Iyer, M. S., and Rhinehart, R. R. 1999. A method to determine the required number of neural network training repetitions. *IEEE Transactions on Neural Networks* 10(2):427–432.

[51] Japkowicz, N.; Hanson, S. J.; and Gluck, M. A. 2000. Nonlinear autoassociation is not equivalent to pca. *Neural Computation* 12(3):531–545.

[52] Jarrett, K.; Kavukcuoglu, K.; Ranzato, M.; and LeCun, Y. 2009. What is the best multi-stage architecture for object recognition? In *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision*, 2146–2153.

[53] Joachims, T. 1999. Making large-scale support vector machine learning practical. In *Advances in kernel methods*. Cambridge, MA, USA: MIT Press. 169–184.

[54] Kanjilal, P. P., and Banerjee, D. N. 1995. On the application of orthogonal transformation for the design and analysis of feedforward networks. *IEEE Transactions on Neural Networks* 6(5):1061–1070.

[55] Karnin, E. 1990. A simple procedure for pruning back-propagation trained neural networks. *IEEE Transactions on Neural Networks* 1(2):239–242.

[56] Kirkpatrick, S.; Gelatt Jr., C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220(4598):671–680.

[57] Krizhevsky, A. 2009. Learning multiple layers of features from tiny images. Master's thesis, Computer Science Department, University of Toronto.

[58] Lafferty, J. D.; McCallum, A.; and Pereira, F. C. N. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning*, 282–289.

[59] Larochelle, H.; Erhan, D.; Courville, A.; Bergstra, J.; and Bengio, Y. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, 473–480. ACM.

[60] Larochelle, H.; Bengio, Y.; Louradour, J.; and Lamblin, P. 2009. Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research* 10:1–40.

[61] LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature* 521:436–444.

[62] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation* 1(4):541–551.

[63] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; and Jackel, L. D. 1990. Handwritten digit recognition with a back-propagation network. In Touretzky, D. S., ed., *Advances in Neural Information Processing Systems*. Morgan Kaufmann. 396–404.

[64] LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998a. Gradient-based learning applied to document recognition. *IEEE* 86(11):2278–2324.

[65] LeCun, Y.; Bottou, L.; Orr, G. B.; and Muller, K.-R. 1998b. Efficient backprop. *Lecture Notes in Computer Science in Neural Networks: Tricks of the Trade* 1524:9–50.

[66] LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In Touretzky, D. S., ed., *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann. 598–605.

[67] LeCun, Y.; Huang, F. J.; and Bottou, L. 2004. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, volume 2, 97–104.

[68] Lee, H.; Ekanadham, C.; and Ng, A. 2008. Sparse deep belief net model for visual area V2. In Platt, J.; Koller, D.; Singer, Y.; and Roweis, S., eds., *Advances in Neural Information Processing Systems 20*. Curran Associates, Inc. 873–880.

[69] Levin, A. U.; Leen, T. K.; and Moody, J. E. 1994. Fast pruning using principal

components. In Cowan, J.; Tesauro, G.; and Alspector, J., eds., *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann. 35–42.

[70] Lisboa, P. J. G., and Perantonis, S. J. 1991. Complete solution of the local minima in the xor problem. *Network - Computation In Neural Systems 2* 1:119–124.

[71] Liu, W., and Floudas, C. 1993. A remark on the gop algorithm for global optimization. *Journal of Global Optimization* 3:519–531.

[72] Lo, J. T.-H., and Bassu, D. 2001a. An adaptive method of training multilayer perceptrons. In *Proceedings of the 2001 International Joint Conference on Neural Networks*, volume 3, 2013–2018.

[73] Lo, J. T.-H., and Bassu, D. 2001b. Robust identification of dynamic systems by neurocomputing. In *Proceedings of the 2001 International Joint Conference on Neural Networks*, volume 2, 1285–1290.

[74] Lo, J. T.-H.; Gui, Y.; and Peng, Y. 2012. Overcoming the local-minimum problem in training multilayer perceptrons with the NRAE training method. In *Advances in Neural Networks - ISNN 2012*, volume Part I, 440–447.

[75] Lo, J. T.-H.; Gui, Y.; and Peng, Y. 2013a. Overcoming the local-minimum problem in training multilayer perceptrons by gradual deconvexification. In *Proceedings of the 2013 International Joint Conference on Neural Networks*, 1–6.

[76] Lo, J. T.-H.; Gui, Y.; and Peng, Y. 2013b. Overcoming the local-minimum problem in training multilayer perceptrons with the NRAE-MSE training method. In *Advances in Neural Networks - ISNN 2013*, volume Part I, 83–90.

[77] Lo, J. T.-H. 1999. Statistical method of pruning neural networks. In *Proceedings of the 1999 International Joint Conference on Neural Networks*, volume 3, 1678–1680.

[78] Lo, J. T.-H. 2010. Convexification for data fitting. *Journal of Global Optimization* 46(2):307–315.

[79] Locatelli, M. 2000. Convergence and first hitting time of simulated annealing algorithms for continuous global optimization. *Mathematical Methods of Operations Research* 54(2):171–199.

[80] Michalewicz, Z. 1999. *Genetic Algorithms + Data Structures = Evolution Programs, third edition*. New York: Springer.

[81] Mitchell, M. 1996. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press.

[82] Mobahi, H.; Collobert, R.; and Weston, J. 2009. Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 737–744. ACM.

[83] Moré, J. J. 1978. The levenberg-marquardt algorithm: Implementation and theory. In Watson, G. A., ed., *Numerical Analysis, Lecture Notes in Mathematics*, volume 630. Berlin, Heidelberg, Germany: Springer-Verlag. 105–116.

[84] Ng, A. Y., and Jordan, M. I. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In Dietterich, T.; Becker, S.; and Ghahramani, Z., eds., *Advances in Neural Information Processing Systems 14*. MIT Press. 841–848.

[85] Osindero, S., and Hinton, G. E. 2008. Modeling image patches with a directed hierarchy of markov random fields. In Platt, J.; Koller, D.; Singer, Y.; and Roweis, S., eds., *Advances in Neural Information Processing Systems 20*. Curran Associates, Inc. 1121–1128.

[86] Press, W. H.; Teukolsky, S. A.; Vetterling, W. T.; and Flannery, B. P. 2007. *Numerical Recipes in C: The Art of Scientific Computing*. New York, NY: Cambridge University Press, third edition.

[87] Principe, J. C.; Euliano, N. R.; and Lefebvre, W. C. 2000. *Neural and Adaptive Systems: Fundamentals through Simulations*. New York: John Wiley and Sons, Inc.

[88] Ranzato, M.; Poultney, C. S.; Chopra, S.; and LeCun, Y. 2006. Efficient learning of sparse representations with an energy-based model. In *Neural Information Processing Systems - NIPS*, 1137–1144.

[89] Reed, R. 1993. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks* 4(5):740–747.

[90] Rifai, S.; Vincent, P.; Muller, X.; Glorot, X.; and Bengio, Y. 2011. Contractive auto-encoders: Explicit invariance during feature extraction. In *In Proceedings of the 28 International Conference on Machine Learning*.

[91] Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning representations by back-propagating errors. *Nature* 323:533–536.

[92] Rumelhart, D. E.; McClelland, J. L.; and the PDP Research Group. 1986. *Parallel Distributed Processing: explorations in the microstructure of Parallel Distributed Processing: explorations in the microstructure of cognition*. Cambrigde, Massachussetts, USA: MIT Press.

[93] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; Berg, A. C.; and Fei-Fei, L. 2015. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)* abs/1409.0575.

[94] Saarinen, S.; Bramley, R.; and Cybenko, G. 1993. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing* 14(3):693–714.

[95] Salakhutdinov, R., and Hinton, G. E. 2009a. Deep boltzmann machines. *Journal of Machine Learning Research - Proceedings Track* 5:448–455.

[96] Salakhutdinov, R., and Hinton, G. E. 2009b. Semantic hashing. *International Journal of Approximate Reasoning* 50(7):969–978.

[97] Salakhutdinov, R., and Hinton, G. 2012. An efficient learning procedure for deep boltzmann machines. *Neural Computation* 24:1967–2006.

[98] Salakhutdinov, R., and Larochelle, H. 2010. Efficient learning of deep boltzmann machines. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9, 693–700.

[99] Salakhutdinov, R.; Mnih, A.; and Hinton., G. E. 2007. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*, 791–798. ACM.

[100] Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61:85–117.

[101] Schwenk, H., and Milgram, M. 1995. Transformation invariant autoassociation with application to hand- written character recognition. In Tesauro, G.; Touretzky, D. S.; and Leen, T. K., eds., *Advances in Neural Information Processing Systems 7*. MIT Press. 992–998.

[102] Shanno, D. F. 1970. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation* 24(111):647–656.

[103] Sietsma, J., and Dow, R. J. 1991. Creating artificial neural networks that generalize. *Neural Networks* 4(1):67–79.

[104] Sima, J. 1996. Back-propagation is not efficient. *Neural Networks* 9(6):1017–1023.

[105] Simard, P. Y.; Steinkraus, D.; and Platt, J. C. 2003. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the 2003 International Conference on Document Analysis and Recognition*, 958–963.

[106] Sprinkhuizen-Kuyper, I. G., and Boers, E. J. W. 1994. A comment on a paper of blum: Blum's "local minima" are saddle points. Technical report, Department of Computer Science, Leiden University, Leiden, The Netherlands.

[107] Sprinkhuizen-Kuyper, I. G., and Boers, E. J. W. 1996. The error surface of the simplest xor network has only global minima. *Neural Computation* 8(6):1301–1320.

[108] Sprinkhuizen-Kuyper, I. G., and Boers, E. J. 1999. The local minima of the error surface of the 2-2-1 XOR network. *Annals of Mathematics and Artificial Intelligence* 25(1-2):107–136.

[109] Szu, H. H. 1987. Nonconvex optimization by fast simulated annealing. *Proceedings of the IEEE* 75(11):1538–1540.

[110] Teh, Y.-W., and Hinton, G. E. 2001. Rate-coded restricted boltzmann machines for face recognition. In *Advance in Neural Information Processing Systems*, volume 13. Cambrigde, MA, USA: The MIT Press.

[111] Tsallis, C., and Stariolo, D. A. 1996. Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications* 233(1-2):395–406.

[112] Vincent, P.; Larochelle, H.; Bengio, Y.; and Manzagol, P.-A. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, 1096–1103. ACM.

[113] Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research* 11:3371–3408.

[114] Weston, J.; Ratle, F.; and Collobert, R. 2008. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, 1168–1175. ACM.

[115] Wilson, D. R., and Martinez, T. R. 2003. The general inefficiency of batch training for gradient descent learning. *Neural Networks* 16(10):1429–1451.

[116] Zlobec, S. 2005. On the Liu-Floudas convexification of smooth programs. *Journal of Global Optimization* 32:401–407.

[117] Zurada, J. M. 1992. *Introduction to Artificial Neural Networks*. St. Paul, MN: West Publishing Company.