CONVERGENCE ANALYSIS OF MULTILAYER FEEDFORWARD NETWORKS TRAINED WITH PENALTY TERMS: A REVIEW

Jian Wang[1], Guoling Yang[1], Shan Liu[1], Jacek M. Zurada[2,3]

[1] College of Science
China University of Petroleum, Qingdao, Shandong, China
wangjiannl@upc.edu.cn
yangguolingfwz@163.com
liushan@upc.edu.cn

[2] Department of Electrical and Computer Engineering
University of Louisville, Louisville, Kentucky, USA

[3] Information Technology Institute, University of Social Sciences, Łódź, Poland
jacek.zurada@louisville.edu

Abstract

Gradient descent method is one of the popular methods to train feedforward neural networks. Batch and incremental modes are the two most common methods to practically implement the gradient-based training for such networks. Furthermore, since generalization is an important property and quality criterion of a trained network, pruning algorithms with the addition of regularization terms have been widely used as an efficient way to achieve good generalization. In this paper, we review the convergence property and other performance aspects of recently researched training approaches based on different penalization terms. In addition, we show the smoothing approximation tricks when the penalty term is non-differentiable at origin.

Key words: Gradient, feedforward neural networks, generalization, penalty, convergence, pruning algorithms.

1 Introduction

Multi-layer perceptron-type neural networks have been widely used in many real-life applications such as data analysis, trend analysis, classification, monitoring, control, clustering and pattern recognition. They automatically learn from observational data [1], [2]. Especially the so-called deep neural

networks that have been used since since 2006 [3] have been recognized to achieve outstanding performance on many important problems in speech recognition, natural language processing, pattern recognition and computer vision [4].

The promising advantages of neural networks lie in their attractive and biologically inspired model. The models encompass data-driven learning techniques that adjust the weights without a specified function for the underlying model. We note that neural networks are mostly nonlinear models, which readily makes them flexible in dealing with real-world and complex tasks. More importantly, neural networks have been rigorously proved to be universal functional approximators, that is, they can approximate any given function with arbitrary accuracy [5].

As commonly known, the backpropagation (BP) algorithm is one of the mostly used techniques for training multi-layer neural networks [6], [7]. There are two main popular modes to implement the BP algorithm: batch mode and incremental mode [8]: For batch learning, the weights of networks are adjusted once and after an entire presentation of the training samples, which corresponds to the standard gradient method. The incremental learning, however, is a variation of the standard gradient method, which updates the weights once and after the presentation of each training sample.

The incremental training strategy may be sub-divided into three different modes in terms of the order that the samples are fed to the network. The first strategy is cyclic learning, whose order is randomly chosen in the initial step and then remains fixed during the whole training procedure. The second strategy is the almost-cyclic learning, where each sample is randomly chosen once per cycle. The last strategy is online learning which selects the samples in completely stochastic order in the whole training procedure.

The training process of a neural network may be stated as a "curve-fitting" problem. A network with "good" generalization means that the input-output mapping computed by the network performs well for testing data never used in training of the network. A network that is constructed to generalize well will generate a suitable input-output mapping. When, however, the training of a network concentrates on training samples, the network may memorize the training examples very well rather than fit the testing data. Such a phenomenon of "overfitting" usually indicates worse generalization.

There are three main factors on influencing the generalization of a trained network: the size of the training data, the network architecture, and the inherent complexity of the problem. How to determine the best architecture of network to achieve a good generalization becomes here an attractive aspect for studying the training properties.

Generally speaking, the optimal network architecture is one with the number of hidden units large enough to learn the examples and small enough to generalize well. To achieve a good generalization, the optimal network design

depends on an appropriate tradeoff between reliability of training samples and goodness of the model. For BP learning, this tradeoff may be fulfilled by minimizing the total risk, expressed as a function of the weight vector $\mathbf{w}$ as follows

$$R(\mathbf{w}) = E_{av}(\mathbf{w}) + \lambda E_c(\mathbf{w}). \qquad (1)$$

The first term is the normal BP error term, which depends on both the network model and the input samples. The second term represents the complexity of the network as a function of the weights, which is so-called penalty term, and λ is the penalization coefficient [9], [10]. The aim of (1) is to find a network with the simplest architecture possible and that is adequate to classify its input patterns.

Gradient-based method is a simple and popular learning algorithm for BPNN training. Some deterministic convergence results of both batch and incremental gradient algorithms for neural networks have been established in [11-19]. Boundedness of the weights during training turns out to be an important assumption to assure the convergent behavior. Interestingly, this prerequisite condition can be usually guaranteed by adding penalty terms. Due to different penalty terms, corresponding learning algorithms demonstrate different performance. In this paper, we will focus on the convergence analysis of BPNN with penalties and claim the distinction among them.

The rest of this paper is organized as follows. A brief description of batch and incremental (three different strategies) gradient-methods for BPNN training is given in Section 2. We describe the generally used penalties and pruning schemes in Section 3, while some specialized tricks for non-differential penalty functions will be stated in the Section 4. The asymptotic and deterministic convergence results are separately listed for different network modes in Section 5 and 6. In the last section, we offer a brief conclusion and some remarks.

2 Gradient method based algorithm for BPNN

Gradient descent method is a first-order optimization algorithm. It is often used to train BPNNs. To find a local minimum of error function, one takes steps proportional to the negative of the gradient of the error function in the weight space.

We consider a feedforward networks with three layers. Suppose that the training sample set is $\{\mathbf{x}^j, \mathbf{o}^j\}_{j=1}^{J}$, where $\mathbf{x}^j$ and $\mathbf{o}^j$ are the input and the desired output of the j-th sample.

For given input $\mathbf{x}^{\boldsymbol{j}}$ $(j = 1, \cdots, J)$, the network output is denoted by $\mathbf{y}^{\boldsymbol{j}}$. Correspondingly, the error produced at the output of the j-th input sample is defined by

$$e(\mathbf{w},\mathbf{x}^{\boldsymbol{j}}) = \left\|\boldsymbol{o}^{\boldsymbol{j}} - \boldsymbol{y}^{\boldsymbol{j}}\right\|, \qquad (2)$$

where $\boldsymbol{y}^{\boldsymbol{j}}$ is the corresponding actual output, $\mathbf{w}$ is the total weights. Summing the error-energy contributions of all the neurons in the output layer, the total instantaneous error energy of the whole networks can be expressed as follows

$$\boldsymbol{E}(\mathbf{w})=\frac{1}{2}\sum_{j=1}^{J} e^2(\mathbf{w},\mathbf{x}^{\boldsymbol{j}}) = \frac{1}{2}\sum_{j=0}^{J-1}\left\|\boldsymbol{o}^{j} - \boldsymbol{y}^{j}\right\|_2^2. \qquad (3)$$

Naturally, the above error function is a function of all the adjustable weights of the multilayer networks. Depending on how the supervised learning of the BP neural networks is performed, two main different learning algorithms-namely, batch and incremental learning, as the following discussed in the context of gradient descent methods. We note that there are three specific incremental algorithms differ by the ordering of the training samples as online, cyclic and almost-cyclic learning [8].

2.2 Batch-mode learning

The popularity for the supervised training of multilayer neural networks has been enhanced by the development of the back-propagation algorithm. Gradient method is widely used to train the back-propagation algorithms.

In the bath-mode learning, adjustments to the weights of BPNN are performed after the presentation of all the J samples, or after an epoch. The cost function for batch learning is defined by the error function $\boldsymbol{E}(\mathbf{w})$, that is,

$$\boldsymbol{E}(\mathbf{w})=\frac{1}{2}\sum_{j=0}^{J-1} e\left(\mathbf{w},\mathbf{x}^{\boldsymbol{j}}\right) = \frac{1}{2}\sum_{j=0}^{J-1}\left\|\boldsymbol{o}^{j} - \boldsymbol{y}^{j}\right\|_2^2 \qquad (4)$$

Adjustments to the weights are carried on an epoch-by-epoch basis. The gradient of the error function is given by $\boldsymbol{\nabla E}_{\mathbf{w}}(\mathbf{w})$. Starting from an initial value $\mathbf{w}^0$, the weights $\{\mathbf{w}^m\}$ are interactively updated by

$$\mathbf{w}^{m+1} = \mathbf{w}^{m} - \eta \boldsymbol{E}_{\mathbf{w}}(\mathbf{w}^{\boldsymbol{m}}),\ m = 0,1,2,\cdots. \qquad (5)$$

We note that the batch-mode learning is completely deterministic and directly stems from the standard gradient method. It is clear to see that it requires large storage which is inconvenient in hardware applications.

2.2 Incremental learning

In the incremental method of supervised learning, adjustments to the weights of multilayer neural networks are performed on a sample-by-sample basis. That is, the weight updating takes place after each presentation of one drawn training sample. Due to the difference of ordering for the samples fed into the network, we distinguish three popular incremental learning strategies: cyclic learning, almost-cyclic learning and on-line learning.

2.2.1 Cyclic learning

Cyclic learning is a learning with a fixed cycle. Before training, the feeding order of training samples is randomly drawn and then fixed in the whole training procedure.

Given an initial weight $\mathbf{w}^0$, the cyclic learning updates the weights interactively by

$$\mathbf{w}^{mJ+j+1} = \mathbf{w}^{mJ+j} - \eta_m \nabla e\left(\mathbf{w}^{mJ+j}, \mathbf{x}^j\right), \quad (6)$$

where m is the m-th iteration epoch, $\nabla e\left(\mathbf{w}^{mJ+j}, \mathbf{x}^j\right)$ is the gradient of the instantaneous error function with respect to the total weight. We note that the $(j+1)$-th weights updating of the m-th cycle depends on the j-th training sample.

2.2.2 Almost-cyclic learning

The training process of almost-cyclic learning consists of training cycles in which each of the samples is fed into the network exactly once. In addition, the order of sample presentation is continually drawn at random after each learning cycle.

For any given initial weight vector $\mathbf{w}^0$, the almost-cyclic learning changes the weights as follows

$$\mathbf{w}^{mJ+j+1} = \mathbf{w}^{mJ+j} - \eta_m \nabla e\left(\mathbf{w}^{mJ+j}, \mathbf{x}^{m(j)}\right). \quad (7)$$

For the m-th training epoch, let $\left\{\mathbf{x}^{m(1)}, \mathbf{x}^{m(1)}, \cdots, \mathbf{x}^{m(J)}\right\}$ be a stochastic permutation of the $\{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^J\}$. This is the essential difference between cyclic and almost-cyclic learning.

2.2.3 On-line learning

For each online learning step, one of the training samples is randomly drawn from the training set and presented to the network. Online learning is a

special incremental learning where the weight updating takes place after each presentation of randomly chosen training samples.

For any initial weight $\mathbf{w}^0$, the weights are iteratively updated by the following formula

$$\mathbf{w}^{m+1} = \mathbf{w}^m - \eta_m \nabla e\left(\mathbf{w}^m, \mathbf{x}^{r(m)}\right), \qquad (8)$$

Where $m \in \mathbb{N}$ is the m-th iteration number, and r(m) is randomly chosen from $1,2,\cdots,J$. We make it clear that there is no training cycles for online learning. Each update of weight only depends on the randomly chosen sample.

3 Penalization and Network Pruning

To obtain a good generalization for a trained network, it is a popular strategy to add penalty terms to the standard error function. The penalties as below modify the objective function and the gradient based BP algorithm effectively prunes the network by pushing redundant weights to zero during training. Then the trained network performs as a smaller system with good generalization. The specific forms of these penalty functions are as follows:

$\phi(\mathbf{w}) = \sum_{w_{ij}\in\mathbf{w}} w_{ij}^2$, (9) Weight Decay ($L_2$ Regularizer) [11-16, 20-22]

$\phi(\mathbf{w}) = \sum_{w_{ij}\in\mathbf{w}} \frac{(w_{ij}/w_0)^2}{1+(w_{ij}/w_0)^2}$, (10) Weight Elimination [23-26]

$\phi(\mathbf{w}) = \sum_{w_{ij}\in\mathbf{w}} |w_{ij}|^{\frac{1}{2}}$, (11) $L_{1/2}$ Regularizer [27], [28]

$\phi(\mathbf{w}) = \sum_{\mathrm{w}_i\in\mathbf{w}} \|\mathrm{w}_i\|$, (12) Group Lasso penalty [29]

When a network is to be pruned, it is a common choice to add a penalty term with the sum of the squared weights. This quadratic penalty results in discouraging the weights from taking large values. However, this penalty term causes all weights to decay exponentially to zero at the same rate and disproportionately penalizes large weights.

To remedy this problem, the Weight Elimination penalty function has been proposed in [23, 24]. It penalizes small weights to decay at a higher rate than large weights by choosing suitable learning rate and penalization coefficients. However, a disadvantage of this penalty is that it can't distinguish between large and very large weights.

An $L_{1/2}$ regularizer was proposed in [27, 28] which is a nonconvex penalty. The $L_{1/2}$ regularizer is observed to have many promising properties such as unbiasedness, sparsity and Oracle properties. Particularly, the solution

of the $L_{1/2}$ regularizer delivers better sparsity than that of the L_1 regularizer. Then, many references employ the $L_{1/2}$ regularizer of the weights as a penalty term. The experiments shoe that $L_{1/2}$ penalty forms better pruning achievement than weight Decay and Weight Elimination.

We note that the $L_{1/2}$ regularizer penalizes the weights individually, that is, some of the weights which connecting a neuron are decreased to zero while other weights are still retain with a large value. Thus, it may not prune the neurons at a group manner.

Group Lasso has been introduced in [29] as an extension of the so-called Lasso, which encourages sparsity at a group level. It is an intermediate penalty function between the L_1 penalty in Lasso and the L_2 penalty (weight Decay). Naturally, a novel penalty term has been investigated by borrowing the Group Lasso idea to train the BP neural networks.

It is clear to see that the penalty terms of the above function (11) and (12) are not differentiable at the origin. This may lead to difficulties on both theoretical analysis and numerical simulations, when the weights are very close to zero. It is a popular strategy to approximate the non-differential penalty term with smoothing functions.

For any finite dimensional vector $\boldsymbol{z}$, we introduce following smoothing functions.

1)"Sqrt Form"

$$h(\mathbf{z})=\begin{cases} \|\mathbf{z}\| & \|\mathbf{z}\| \geq \alpha, \\ \sqrt{\|\mathbf{z}\|+\alpha}+\alpha-\sqrt{\alpha^2+\alpha}, & \|\mathbf{z}\| < \alpha, \end{cases} \quad (13)$$

2)"Quadratic Form"

$$h(\mathbf{z})=\begin{cases} \|\mathbf{z}\|, & \|\mathbf{z}\| \geq \alpha, \\ \frac{\|\mathbf{z}\|^2}{2\alpha}+\frac{\alpha}{2}, & \|\mathbf{z}\| < \alpha, \end{cases} \quad (14)$$

3)"Quartic Form"

$$h(\mathbf{z})=\begin{cases} \|\mathbf{z}\|, & \|\mathbf{z}\| \geq \alpha, \\ -\frac{\|\mathbf{z}\|^4}{8\alpha^3}+\frac{3\|\mathbf{z}\|^2}{4\alpha}+\frac{3\alpha}{8}, & \|\mathbf{z}\| < \alpha, \end{cases} \quad (15)$$

4)"Sextic Form"

$$h(\mathbf{z})=\begin{cases} \|\mathbf{z}\|, & \|\mathbf{z}\| \geq \alpha, \\ \frac{\|\mathbf{z}\|^6-15\alpha^2\|\mathbf{z}\|^4+75\alpha^4\|\mathbf{z}\|^2+35\alpha^6}{96\alpha^5}, & \|\mathbf{z}\| < \alpha, \end{cases} \quad (16)$$

where the smoothing parameter $\alpha \leq 1$ is a fixed positive constant. We note that the approximations are much closer to the original absolute function for the higher orders of the smoothing functions.

4 Asymptotic Convergence Analysis

For online learning strategy, the theoretical results mainly perform asymptotic convergent behaviors since the training sequence is absolute randomly generated from the training data set. In [13], the convergence results of online BP neural networks with L_2 penalty were based on the following assumptions.

A1) Each training sample is randomly chosen from the training set $\{\mathbf{x}^{\boldsymbol{j}}, \mathbf{y}^{\boldsymbol{j}}\}_{\boldsymbol{j=1}}^{\boldsymbol{J}}$ with independent identical distribution;

A2) The activation functions $\boldsymbol{g}$ and $\boldsymbol{f}$ are twice continuously differentiable on $\mathbb{R}$. Moreover, $\boldsymbol{g}, \boldsymbol{f}, \boldsymbol{g}', \boldsymbol{f}', \boldsymbol{g}''$ and $\boldsymbol{f}''$ are uniformly bounded on $\mathbb{R}$.

A2') The activation function $\boldsymbol{g}$ is twice continuously differentiable on $\mathbb{R}$. Moreover, $\boldsymbol{g}, \boldsymbol{g}', \boldsymbol{g}''$ are uniformly bounded on $\mathbb{R}$. $\boldsymbol{f}(\boldsymbol{t}) = \boldsymbol{t}$ for all $\boldsymbol{t} \in \mathbb{R}$.

A3) $\{\boldsymbol{\eta_m}\}$ is a decreasing positive sequence such that a) $\sum_{\boldsymbol{m=0}}^{\infty} \boldsymbol{\eta_m} = \infty$, b) $\lim_{m\to\infty} \sup(\boldsymbol{\eta_m^{-1}} - \boldsymbol{\eta_{m-1}^{-1}}) < \infty$, and c) $\sum_{\boldsymbol{m=0}}^{\infty} \boldsymbol{\eta_m^d} < \infty$, for some $d > 1$.

Theorem 4.1. Suppose that the above assumptions A1), A3), and either A2) or A2') hold. Let $\{\mathbf{w}^{\boldsymbol{m}}\}_{m\geq 1}$ be a sequence of weight vectors iteratively generated by (online equation) with arbitrary initial value $\mathbf{w}^{\mathbf{0}}$. Then, $\mathbf{w}^{\boldsymbol{m}} \to \mathbf{w}^*$ with probability 1, where $\mathbf{w}^*$ is the optimal weight.

Fault-tolerant BP NNs have been proposed for over two decades, which inject weight noise during training procedure. However, until recently, only a few sources discuss its convergent behavior [21], [22].

Actually, they focus on two kinds of online fault-tolerant BP NNs: node fault and weight noise injections. The online node fault injection-based algorithm is that the hidden nodes randomly output zeros during training. For weight noise injections, there are mainly two types of weight noise injection-based algorithms during each step of training: multiplicative weight noise and additive weight noise injections with weight decay penalty (L_2 regularizer).

On the basis of the different fault strategies in training process, the corresponding objective functions are established, the boundedness of weight sequence and the asymptotic convergence results are rigorously proved. For brevity, we only list the convergence results for weight noise injection-based BPNN under mild conditions of activation function and learning rates [21].

A1) The activation function is set to be the common sigmoid function $f(x) = \frac{1}{e^{-x}}, x \in \mathbb{R}$.

A2) The learning rates $\boldsymbol{\eta_m}$ satisfy that $\boldsymbol{\eta_m} \to \mathbf{0}, \sum_{\boldsymbol{m=0}}^{\infty} \boldsymbol{\eta_m} = \infty$ and $\sum_{\boldsymbol{m=0}}^{\infty} \boldsymbol{\eta_m^2} < \infty$.

Theorem 4.2. Suppose that the activation function is with the sigmoid function in A1). The weight sequence $\{\mathbf{w}^{\boldsymbol{m}}\}_{m\geq 1}$ is iteratively generated by (online equation) with arbitrary initial value $\mathbf{w}^{\mathbf{0}}$. The penalization coefficient α is some positive constant. In addition, if the assumption A2) is also valid. Then, $\lim_{\boldsymbol{m}\to\infty}\nabla E(\mathbf{w}^{\boldsymbol{m}})=\mathbf{0}$ with probability 1, where $E(\mathbf{w}^{\boldsymbol{m}})$ is the established objective function.

5 Deterministic Convergence Analysis

5.1 Batch-mode learning with $L_{1/2}$ regularizer

The idea of $L_{1/2}$ regularizer has been successfully applied in variable selection and feature extraction problems in high dimensional data analysis. It is a nonconvex penalty and possesses many promising properties such as unbiasedness, sparsity and oracle property. Thus, it has been introduced into the batch gradient learning algorithm for the pruning BP NNs in [19].

Consider a single hidden-layer network consisting of p input neurons, q hidden neurons and 1 output neuron. Let $w_0 = (w_{10}, w_{20}, \cdots, w_{q0})^T \in \mathbb{R}^{\boldsymbol{q}}$ be the weight vector which connects the hidden nodes and the output node, and denote $w_i = (w_{i1}, w_{i2}, \cdots, w_{ip})^T \in \mathbb{R}^{\boldsymbol{p}}$ as the weight vector between the input nodes and the i-th hidden node. Let g: $\mathbb{R} \to \mathbb{R}$ be the activation function of the hidden and output layers. Define a vector-value function G: $\mathbb{R}^{\boldsymbol{q}} \to \mathbb{R}^{\boldsymbol{q}}$,$\mathrm{G}(\mathbf{x}) = (g(x_1), g(x_2), \cdots, g(x_q))^T \in \mathbb{R}^{\boldsymbol{q}}$, for $\mathbf{x} = (x_1, x_2, \cdots, x_q)^T \in \mathbb{R}^{\boldsymbol{q}}$. Suppose that $\{\mathbf{x}^{\boldsymbol{j}}, \boldsymbol{o}^{\boldsymbol{j}}\}_{\boldsymbol{j=1}}^{\boldsymbol{J}}$ are the given bounded training samples. The error function with the $L_{1/2}$ regularization penalty term is denoted by the following:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{J}(\mathbf{o}^j - g(\mathbf{w}_0 \cdot G(\mathbf{v}\mathbf{x}^j)))^2 + \lambda\sum_{i=1}^{q}\sum_{k=0}^{p}|w_{ik}|^{\frac{1}{2}}, \qquad (17)$$

We note that the $L_{1/2}$ regularization term in (11) is non-differentiable at the origin, which leads to more difficulties in theoretical analysis. More importantly, it is inevitable that the oscillation phenomenon will appear in numerical simulations. To overcome this drawback, a modified $L_{1/2}$ regularization term is presented by employing a smoothing function to approximate the absolution value function, which results in the new error function.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{J}(\mathbf{o}^j - g(\mathbf{w}_0 \cdot G(\mathbf{v}\mathbf{x}^j)))^2 + \lambda\sum_{i=1}^{q}\sum_{k=0}^{p}\sqrt{f(w_{ik})}, \qquad (18)$$

where $f(x)$ is as the following piecewise polynomial function:

$$f(x)=\begin{cases}|x|, & if|x|\geq a,\\ -\frac{x^4}{8a^3}+\frac{3x^2}{4a}+\frac{3a}{8}, & if|x|<a,\end{cases} \quad (19)$$

Starting with an initial value $\mathbf{w}^0$, the weights $\{\mathbf{w}^n\}$ are iteratively updated by:

$$\mathbf{w}^{m+1}=\mathbf{w}^m-\eta\nabla E(\mathbf{w}^m), \quad (20)$$

where $\nabla E(\mathbf{w}^m)$ represents the gradient of $E(\mathbf{w}^m)$ with respect to $\mathbf{w}$, and the learning rate $\eta>0$ is a constant.

To show the convergence results of the batch gradient method with smoothing $L_{1/2}$ regularization penalty, some sufficient conditions are as follows:

A1). $|g(t)|, |g'(t)|, |g''(t)|$ are uniformly bounded for $t\in\mathbb{R}$;

A2). $\|\mathbf{w}_0^m\|$ (m=0, 1, 2,$\cdots$) are uniformly bounded;

A3). η and λ are chosen satisfy. $0<\eta<\frac{1}{M\lambda+c}$, where $M=\frac{\sqrt{6}}{4\sqrt{a^3}}$ and c is a given positive constant;

A4). There exists a compact set Φ such that $\mathbf{w}^m\in\Phi$ and the set $\Phi_0=\{\mathbf{w}\in\Phi:\nabla E(\mathbf{w})=0\}$ contains finite points.

Theorem 5.1. Let the error function be defined by (18), and the weight sequence $\{\mathbf{w}^m\}$ be generated by the iteration algorithm (20) for an arbitrary initial value. If assumption A1)-A3) are valid, then we have

(i). $E(\mathbf{w}^{m+1})\leq E(\mathbf{w}^m), \quad m=0,\ 1,\ 2,\ \cdots;$

(ii). There exists $E^*\geq 0,$ such that $\lim_{n\to\infty}E(\mathbf{w}^m)=E^*;$

(iii). $\lim_{n\to\infty}\|\nabla E(\mathbf{w}^m)\|=0.$

Furthermore, if assumption A4) also holds, then we have the following strong convergence;

(iv). There exists a point $\mathbf{w}^*\in\Phi_0$ such that $\lim_{n\to\infty}\mathbf{w}^m=\mathbf{w}^*$.

5.2 Cyclic learning with $L_{1/2}$ regularizer

Cyclic learning algorithm is one of the popular incremental algorithms compared with the batch mode training for BPNN. In addition, incremental learning algorithm is more efficient in term of both storage and computational burden. Based on the better pruning performance of $L_{1/2}$ regularizer, it is then presented as a penalty term for cyclic tearing of BP neural networks.

A modified smoothing $L_{1/2}$ regularizer has been proposed due to the non-differentiable ability at the origin. The smoothing error function is then defined for cyclic mode training procedure

$$E(\mathbf{w}) = \frac{1}{2}(\mathbf{o}^j - g(\mathbf{w}_0 \cdot G(\mathbf{v}\mathbf{x}^j)))^2 + \lambda \sum_{i=1}^{q} \sum_{k=0}^{p} \sqrt{f(w_{ik})}, \qquad (21)$$

Where $f(x)$ is identical to the above definition (3), $j = 1, \cdots, J$. For any given initial weight $\mathbf{w}_0$, the weight sequence $\{\mathbf{w}^m\}$ is iteratively generated by:

$$\mathbf{w}^{mJ+j+1} = \mathbf{w}^{mJ+j} - \eta_m \nabla E\left(\mathbf{w}^{mJ+j}\right), \qquad (22)$$

where $m = 0, 1, 2, \cdots; j = 1, \cdots, J;$ η_m is the learning rate in the m-th training epoch.

Let $\Phi_0 = \{\mathbf{w}: \nabla E(\mathbf{w}) = 0\}$ be the stationary point set of the error function $E(\mathbf{w})$. The following assumptions are imposed for the convergence of the cyclic BPNN with $L_{1/2}$ penalty term [18].

A1) $|g(t)|$ and $|g'(t)|$ are Lipschitz continuous for $t \in \mathbb{R}$;

A2) The learning rates η_m satisfy that

$$0 < \eta_m < 1 \text{ , and } \sum_{m=0}^{\infty} \eta_m < \infty.$$

Theorem 5.2. Let the error function $E(\mathbf{w})$ be defined by (21). $\mathbf{w}^0$ be an arbitrary initial value, and the weight sequence $\{\mathbf{w}^m\}$ be generated by the iteration algorithm (22). Assume the conditions A1) and A2) are valid, then there exists an unique $\mathbf{w}^* \in \Phi_0$, such that

$$\lim_{n\to\infty} \mathbf{w}^m = \mathbf{w}^*.$$

$$\lim_{m\to\infty} \|\nabla E(\mathbf{w}^m)\| = \|E(\mathbf{w}^*)\| = 0.$$

5.3. Almost cyclic learning with L_2 regularizer

For almost-cyclic learning with L_2 penalty, each training sample is chosen with a stochastic order and is fed exactly once in each training epoch. Let $\{\mathbf{x}^{m(1)}, \mathbf{x}^{m(2)}, \cdots, \mathbf{x}^{m(J)}\}$ be a stochastic permutation of the samples set $\{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^{J-1}\}$. The learning rate of training procedure is fixed as $\eta_m > 0$ in the m-th epoch. For fixed weight $\mathbf{w}$, the output error for the j-th iteration is defined as

$$E(\mathbf{w}) = \frac{1}{2}(\mathbf{o}^j - f(\mathbf{u} \cdot G(\mathbf{v}\mathbf{x}^{m(j)})))^2 + \lambda\|\mathbf{w}\|^2, \qquad (23)$$

The weights are iteratively updated as following

$$\mathbf{w}^{mJ+j+1} = \mathbf{w}^{mJ+j} - \eta_m \nabla E\left(\mathbf{w}^{mJ+j}, \mathbf{x}^{m(j)}\right), \qquad (24)$$

where $\nabla E\left(\mathbf{w}^{mJ+j}, \mathbf{x}^{m(j)}\right)$ is the gradient with respect to $\mathbf{w}$ of the j-th iteration in the m-th epoch

Let $\Omega_0 = \{\mathbf{w}: \nabla E(\mathbf{w}) = 0\}$ be the stationary point set of the error function (23). Denote $\Omega_{0,s} \subset \mathbb{R}$ be the projection of Ω_0 onto the $s-th$ coordinate axis, that is,

$$\Omega_{0,s} = \{\mathbf{w}_s \in \mathbb{R}: \mathbf{w} = (w_1, w_2, \cdots, w_s, \cdots w_{n(p+1)}) \in \Omega_0\}$$

For $s = 1,\ 2,\ \cdots, n(p+1)$. To guarantee the convergence of the algorithm, the following assumptions are required [16]:

A1). $g'(t)$ and $f'(t)$ are Lipschitz continuous on $\mathbb{R}$;

A2). $\eta_m > 0$, $\sum_{\mathrm{m}=0}^{\infty} \eta_m = \infty$ and $\sum_{\mathrm{m}=0}^{\infty} {\eta_m}^2 < \infty$;

A3). $\Omega_{0,s}$ does not contain any interior point for every $s = 1,\ 2,\ \cdots, n(p+1)$.

Theorem 5.3. Assume that conditions A1) and A2) are valid. Then starting from an arbitrary initial weight $\mathbf{w}^0$, the learning sequence $\{\mathbf{w}^m\}$ generated by (24) is uniformly bounded, that is, there exists a positive constant $C > 0$ such that

$$\|\mathbf{w}^m\| < C, \quad m = 0,\ 1,\ 2,\ \cdots$$

and satisfies the following weak convergence

$$\lim_{n\to\infty} \|\nabla E(\mathbf{w}^m)\| = 0.$$

Moreover, if the assumption A3) is also valid, there holds the strong convergence. There exists an unique $\mathbf{w}^* \in \Phi_0$, such that

$$\lim_{n\to\infty} \mathbf{w}^m = \mathbf{w}^*.$$

6 Conclusions

Different penalty terms that are applied for different learning modes demonstrate various convergence results. For online training, only L_2 regularizer is imposed to train the fault-tolerant BPNNs. The weight sequence is uniformly bounded during training. In addition, the asymptotic convergence results are obtained due to the absolute randomly chosen for the training samples and the randomly weight noise.

For batch-mode learning, it is simple to get the convergence results since the standard gradient descent method are carried out in the training process. Moreover, the learning rate can be selected as a small positive constant. It is a promising point that the $L_{1/2}$ regularizer as introduced during training improves the pruning ability and generalization.

For cyclic and almost cyclic learning, under assumptions deterministic convergence results have been obtained with L_2 and $L_{1/2}$ regularizer penalties under specific on the learning rates and the activation functions, respectively.

An observation can be made that a smaller network having similar approximation error for the training samples performs much better on generalization.

Adding penalization terms to an objective function is an efficient way to prune the redundant hidden neuron. It is important to pay special attention to the convergence analysis which guarantees the convergent network from theoretical point of view. In addition, we note that pruning performance should be a promising additional aspect to study networks convergence behavior.

Acknowledgments

The authors wish to thank the anonymous reviewers for careful error proofing of the manuscript and many insightful comments and suggestions which greatly improved this work.

This project was supported in part by the National Natural Science Foundation of China (No. 61305075), the China Postdoctoral Science Foundation (No. 2012M520624), Natural Science Foundation of Shandong Province (No. ZR2013FQ004), the Specialized Research Fund for the Doctoral Program of Higher Education of China (No. 20130133120014) and the Fundamental Research Funds for the Central Universities (No. 15CX05053A, 15CX08011A).

References

1. Hagan M. T., Demuth H. B., Beale M. H., 1996, *Neural networks design*. Boston ; London: PWS.
2. Haykin S. S., 1999, *Neural networks : a comprehensive foundation*, 2nd ed. Upper Saddle River, N.J. ; London: Prentice-Hall.
3. Hinton G. E.Salakhutdinov R. R., Jul 2006, *Reducing the dimensionality of data with neural networks*, Science, Vol. 313, No. 5786, pp. 504-507.
4. LeCun Y., Bengio Y., Hinton G., 05/28/ 2015, Deep learning, Nature, Vol. 521, No. 7553, pp. 436-444.
5. Sutskever I., Hinton G. E., Nov 2008, *Deep Narrow Sigmoid Belief Networks Are Universal Approximators*, Neural Computation, Vol. 20, No. 11, pp. 2629-2636.
6. Werbos P. J., 1974, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D., Harvard University, Cambridge, MA.
7. Rumelhart D. E., Hinton G. E., Williams R. J., Oct 9 1986, *Learning Representations by Back-Propagating Errors*, Nature, Vol. 323, No. 6088, pp. 533-536.
8. Nakama T., Dec 2009, *Theoretical analysis of batch and on-line training for gradient descent learning in neural networks*, Neurocomputing, Vol. 73, No. 1-3, pp. 151-159.
9. Reed R., 1993, *Pruning algorithms-a survey*, Neural Networks, IEEE Transactions on, Vol. 4, No. 5, pp. 740-747.

10. Bishop C. M., 1993, *Curvature-driven smoothing: a learning algorithm for feed-forward networks*, Neural Networks, IEEE Transactions on, Vol. 4, No. 5, pp. 882-884.
11. Wu W., Shao H., Li Z., 2006, Convergence of batch BP algorithm with penalty for FNN training, in *Neural Information Processing*, pp. 562-569.
12. Zhang H., Wu W., Yao M., 2007, Boundedness of a batch gradient method with penalty for feedforward neural networks, in *Proceedings of the 12th WSEAS International Conference on Applied Mathematics*, pp. 175-178.
13. Zhang H., Wu W., 2009, *Boundedness and convergence of online gradient method with penalty for linear output feedforward neural networks*, Neural Process Lett, Vol. 29, No. 3, pp. 205-212.
14. Zhang H., Wu W., Liu F., Yao M., 2009, *Boundedness and convergence of online gradient method with penalty for feedforward neural networks*, Neural Networks, IEEE Transactions on, Vol. 20, No. 6, pp. 1050-1054.
15. Shao H., Zheng G., 2011, *Boundedness and convergence of online gradient method with penalty and momentum*, Neurocomputing, Vol. 74, No. 5, pp. 765-770.
16. Wang J., Wu W., Zurada J. M., 2012, *Computational properties and convergence analysis of BPNN for cyclic and almost cyclic learning with penalty*, Neural Networks, Vol. 33, pp. 127-135.
17. Yu X., Chen Q., 2012, *Convergence of gradient method with penalty for Ridge Polynomial neural network*, Neurocomputing, Vol. 97, pp. 405-409.
18. Fan Q., Zurada J. M., Wu W., 2014, *Convergence of online gradient method for feedforward neural networks with smoothing L 1/2 regularization penalty*, Neurocomputing, Vol. 131, pp. 208-216.
19. Wu W., Fan Q., Zurada J. M., Wang J., Yang D., Liu Y., 2014, *Batch gradient method with smoothing L1/2 regularization for training of feedforward neural networks*, Neural Networks, Vol. 50, pp. 72-78.
20. Leung C. S., Tsoi A.-C., Chan L. W., 2001, *Two regularizers for recursive least squared algorithms in feedforward multilayered neural networks*, Neural Networks, IEEE Transactions on, Vol. 12, No. 6, pp. 1314-1332.
21. Sum J., Chi-Sing L., Ho K., 2012, *Convergence Analyses on On-Line Weight Noise Injection-Based Training Algorithms for MLPs*, Neural Networks and Learning Systems, IEEE Transactions on, Vol. 23, No. 11, pp. 1827-1840.
22. Sum J. P., Chi-Sing L., Ho K. I. J., 2012, *On-Line Node Fault Injection Training Algorithm for MLP Networks: Objective Function and Convergence Analysis*, Neural Networks and Learning Systems, IEEE Transactions on, Vol. 23, No. 2, pp. 211-222.
23. Weigend A. S., Rumelhart D. E., Huberman B., 1991, Generalization by weight-elimination applied to currency exchange rate prediction, in *Neural Networks, IJCNN 1991 International Joint Conference on*, Seattle, pp. 837-841.
24. Weigend A. S.Rumelhart D. E., 1992, *Generalization through minimal networks with application to forecasting*: Defense Technical Information Center.

25. Rakitianskaia A., Engelbrecht A., 2014, Weight regularization in particle swarm optimization neural network training, in *Swarm Intelligence (SIS), 2014 IEEE Symposium on*, pp. 1-8.
26. Thomas P., Suhner M. C., 2015, *A new multilayer perceptron pruning algorithm for classification and regression applications*, Neural Process Lett, pp. 1-22.
27. Xu Z., Zhang H., Wang Y., Chang X., 2010, L(1/2) regularization, Science China-Information Sciences, Vol. 53, No. 6, pp. 1159-1165.
28. Xu Z., Chang X., Xu F., Zhang H., 2012, *L(1/2) Regularization: A Thresholding Representation Theory and a Fast Solver*, Neural Networks and Learning Systems, IEEE Transactions on, Vol. 23, No. 7, pp. 1013-1027.
29. Yuan M., Lin Y., 2006, *Model selection and estimation in regression with grouped variables*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), Vol. 68, pp. 49–67.