neural processing letters manuscript No.
(will be inserted by the editor)

Fast and Stable Learning Utilizing Singular Regions of Multilayer Perceptron

Seiya Satoh · Ryohei Nakano

the date of receipt and acceptance should be inserted later

Abstract In the parameter space of MLP(J), multilayer perceptron with J hidden units, there exist flat areas called singular regions created by applying reducibility mappings to the optimal solution of MLP($J-1$). Since such singular regions cause serious stagnation of learning, a learning method to avoid singular regions has been desired. However, such avoiding does not guarantee the quality of the final solutions. This paper proposes a new learning method which does not avoid but makes good use of singular regions to stably and successively find excellent solutions commensurate with MLP(J). The proposed method worked well in our experiments using artificial and real data sets.

Keywords multilayer perceptron · learning method · reducibility mapping · singular region · search space

1 Introduction

Let MLP(J) be a multilayer perceptron having J hidden units. It is known in MLP(J) parameter space that a subspace having the same input-output map as the optimal solution of MLP($J-1$) can form a flat area called a singular region, and the singular region causes stagnation of learning [5]. Long time ago Hecht-Nielsen pointed out MLP parameter space is full of such flat areas and troughs [7], and recent experimental research [11] supported his insight to reveal most search points have huge condition numbers, e.g. more than 10^{15}.

Natural gradient [1,2] was once proposed to avoid the stagnation of learning, but even the method may get stuck in singular regions and is not guaranteed to find an excellent solution. It is reported [14] an adaptive learning

S. Satoh · R. Nakano
Department of Computer Science, Chubu University
1200 Matsumoto-cho, Kasugai, 487-8501 Japan
E-mail: nakano@cs.chubu.ac.jp

rate improves the performance of natural gradient. Recently an alternative constructive method to decide suitable weights has been proposed [8].

It is also known that many useful statistical models, such as MLP, Gaussian mixtures, and HMM, are singular models having singular regions where parameters are nonidentifiable. Theoretical research has been eagerly done to clarify mathematical characteristics of singular models and especially Watanabe has produced fruit of singular learning theory [15,16]; however, experimental research is rather insufficient to fully support the theories.

In MLP parameter space there are a number of local minima forming equivalence class [13]. Even if we exclude redundant solutions belonging to equivalence class, it is widely believed that there still remain local minima [4]. When we adopt an exponential function as an activation function [10], there surely exist local minima due to the expressive power of polynomials. In XOR problem, however, it was proved there is no strict local minima [6]. Thus, since we have no clear knowledge of MLP parameter space, we run a learning method repeatedly changing initial weights to find an excellent solution.

This paper proposes an extended version of SSF [11], called SSF1.2. SSF1.2 does not avoid but makes good use of singular regions to stably and successively find excellent solutions. The method starts with an MLP having one hidden unit and then gradually increases the number of hidden units until the intended number. When it increases the number of hidden units from $J-1$ to J, it utilizes the optimum of MLP($J-1$) to form two kinds of singular regions in MLP(J) parameter space. Each singular region forms a line, and the learning method can descend in the MLP(J) parameter space since most points along the line are saddles. Thus, we can always find a solution of MLP(J) better than the optimum of MLP($J-1$). Our method is evaluated by the experiments for sigmoidal and polynomial-type MLPs using artificial and real data sets.

2 Singular Regions of Multilayer Perceptron

This section explains that the optimum of MLP($J-1$) is used to form singular regions in MLP(J) parameter space [5]. This result is universal in the sense that it does not depend on the choice of an error function or an activation function.

Consider MLP(J) having J hidden units and one output unit. MLP(J) with parameters $\boldsymbol{\theta}_J$ outputs $f_J(\boldsymbol{x};\boldsymbol{\theta}_J)$ for input $\boldsymbol{x}$. Here $g(h)$ denotes an activation function, and $\boldsymbol{\theta}_J = \{w_0, w_j, \boldsymbol{w}_j, j = 1, \cdots, J\}$, where $\boldsymbol{w}_j = (w_{jk})$.

$$f_J(\boldsymbol{x};\boldsymbol{\theta}_J) = w_0 + \sum_{j=1}^{J} w_j z_j, \qquad z_j \equiv g(\boldsymbol{w}_j^T \boldsymbol{x}) \tag{1}$$

Let input vector $\boldsymbol{x} = (x_k)$ be K-dimensional. Given training data $\{(\boldsymbol{x}^\mu, y^\mu), \mu = 1, \cdots, N\}$, we want to find the parameter vector $\boldsymbol{\theta}_J$ which minimizes the fol-

lowing error function.

$$E_J = \frac{1}{2}\sum_{\mu=1}^{N}(f_J^\mu - y^\mu)^2, \qquad \text{where} \quad f_J^\mu \equiv f_J(\boldsymbol{x}^\mu;\boldsymbol{\theta}_J) \tag{2}$$

At the same time we consider the following MLP($J-1$) having $J-1$ hidden units, where $\boldsymbol{\theta}_{J-1} = \{u_0, u_j, \boldsymbol{u}_j, j = 2, \cdots, J\}$.

$$f_{J-1}(\boldsymbol{x};\boldsymbol{\theta}_{J-1}) = u_0 + \sum_{j=2}^{J} u_j v_j, \qquad v_j \equiv g(\boldsymbol{u}_j^T\boldsymbol{x}) \tag{3}$$

The error function of MLP($J-1$) is defined as follows.

$$E_{J-1}(\boldsymbol{\theta}) = \frac{1}{2}\sum_{\mu=1}^{N}(f_{J-1}^\mu - y^\mu)^2, \qquad \text{where} \quad f_{J-1}^\mu \equiv f_{J-1}(\boldsymbol{x}^\mu;\boldsymbol{\theta}_{J-1}) \tag{4}$$

Let $\widehat{\boldsymbol{\theta}}_{J-1}$ denote a critical point of MLP($J-1$), which satisfies the following

$$\frac{\partial E_{J-1}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0}. \tag{5}$$

The necessary conditions for the critical point of MLP ($J-1$) are shown below. Here $j = 2, \cdots, J$ and $v_j^\mu \equiv g(\boldsymbol{u}_j^T\boldsymbol{x}^\mu)$.

$$\frac{\partial E_{J-1}}{\partial u_0} = \sum_{\mu}(f_{J-1}^\mu - y^\mu) = 0 \tag{6}$$

$$\frac{\partial E_{J-1}}{\partial u_j} = \sum_{\mu}(f_{J-1}^\mu - y^\mu)\ v_j^\mu = 0 \tag{7}$$

$$\frac{\partial E_{J-1}}{\partial \boldsymbol{u}_j} = u_j \sum_{\mu}(f_{J-1}^\mu - y^\mu)\ g'(\boldsymbol{u}_j^T\boldsymbol{x}^\mu)\ \boldsymbol{x}^\mu = \mathbf{0} \tag{8}$$

Now we consider the following three reducibility mappings α, β, γ, and let $\widehat{\boldsymbol{\Theta}}_J^\alpha$, $\widehat{\boldsymbol{\Theta}}_J^\beta$, and $\widehat{\boldsymbol{\Theta}}_J^\gamma$ denote the regions obtained by applying these three mappings to the optimum $\widehat{\boldsymbol{\theta}}_{J-1} = \{\widehat{u}_0, \widehat{u}_j, \widehat{\boldsymbol{u}}_j, j = 2, \cdots, J\}$ of MLP($J-1$). Each reducibility mapping maps the optimal point of MLP($J-1$) to MLP(J) subspace called a singular region. Any two points in the region are I-O equivalent since they have the same input-output mapping from $\boldsymbol{x}$ to f_J. Moreover, MLP(J) at any point in the region is equivalent to MLP($J-1$) at the optimal point. These can be seen by examining the value setting of weights in the following mappings.

$$\widehat{\boldsymbol{\theta}}_{J-1} \xrightarrow{\alpha} \widehat{\boldsymbol{\Theta}}_J^\alpha, \quad \widehat{\boldsymbol{\theta}}_{J-1} \xrightarrow{\beta} \widehat{\boldsymbol{\Theta}}_J^\beta, \quad \widehat{\boldsymbol{\theta}}_{J-1} \xrightarrow{\gamma} \widehat{\boldsymbol{\Theta}}_J^\gamma \tag{9}$$

$$\widehat{\boldsymbol{\Theta}}_J^{\alpha} \equiv \{\boldsymbol{\theta}_J |\ w_0 = \widehat{u}_0,\ w_1 = 0, \\ w_j = \widehat{u}_j, \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, j = 2, \cdots, J\} \tag{10}$$

$$\widehat{\boldsymbol{\Theta}}_J^{\beta} \equiv \{\boldsymbol{\theta}_J |\ w_0 + w_1 g(w_{10}) = \widehat{u}_0,\ \boldsymbol{w}_1 = [w_{10}, 0, \cdots, 0]^T, \\ w_j = \widehat{u}_j, \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, j = 2, \cdots, J\} \tag{11}$$

$$\widehat{\boldsymbol{\Theta}}_J^{\gamma} \equiv \{\boldsymbol{\theta}_J |\ w_0 = \widehat{u}_0,\ w_1 + w_2 = \widehat{u}_2,\ \boldsymbol{w}_1 = \boldsymbol{w}_2 = \widehat{\boldsymbol{u}}_2, \\ w_j = \widehat{u}_j, \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, j = 3, \cdots, J\} \tag{12}$$

(1) region $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$ is $(K+1)$-dimensional since free vector $\boldsymbol{w}_1$ is $(K+1)$-dimensional.
(2) region $\widehat{\boldsymbol{\Theta}}_J^{\beta}$ is two-dimensional since all we have to do is to satisfy the following

$$w_0 + w_1\ g(w_{10}) = \widehat{u}_0. \tag{13}$$

(3) region $\widehat{\boldsymbol{\Theta}}_{\gamma}$ is a line since we have only to satisfy the following

$$w_1 + w_2 = \widehat{u}_2. \tag{14}$$

Here we review a critical point where the gradient $\partial E / \partial \boldsymbol{\theta}$ of an error function $E(\boldsymbol{\theta})$ gets zero. In the context of minimization, a critical point is classified into a local minimum and a saddle. A critical point $\boldsymbol{\theta}_0$ is classified as a local minimum when any point $\boldsymbol{\theta}$ in its neighborhood satisfies $E(\boldsymbol{\theta}_0) \leq E(\boldsymbol{\theta})$, otherwise is classified as a saddle.

In this paper we classify a local minimum into a wok-bottom and a gutter. A wok-bottom $\boldsymbol{\theta}_0$ is a strict local minimum where any point $\boldsymbol{\theta}$ in its neighborhood satisfies $E(\boldsymbol{\theta}_0) < E(\boldsymbol{\theta})$, and a gutter is a continuous subspace where any points $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ in the subspace satisfy $E(\boldsymbol{\theta}_1) = E(\boldsymbol{\theta}_2)$ or $E(\boldsymbol{\theta}_1) \approx E(\boldsymbol{\theta}_2)$, and any point $\boldsymbol{\theta}$ in its neighborhood satisfies $E(\boldsymbol{\theta}_1) < E(\boldsymbol{\theta})$.

The necessary conditions for the critical point of MLP (J) are shown below. Here $j = 2, \cdots, J$ and $z_j^{\mu} \equiv g(\boldsymbol{w}_j^T \boldsymbol{x}^{\mu})$.

$$\frac{\partial E_J}{\partial w_0} = \sum_{\mu} (f_J^{\mu} - y^{\mu}) = 0 \tag{15}$$

$$\frac{\partial E_J}{\partial w_1} = \sum_{\mu} (f_J^{\mu} - y^{\mu})\ z_1^{\mu} = 0 \tag{16}$$

$$\frac{\partial E_J}{\partial w_j} = \sum_{\mu} (f_J^{\mu} - y^{\mu})\ z_j^{\mu} = 0, \tag{17}$$

$$\frac{\partial E_J}{\partial \boldsymbol{w}_1} = w_1 \sum_{\mu} (f_J^{\mu} - y^{\mu})\ g'(\boldsymbol{w}_1^T \boldsymbol{x}^{\mu})\ \boldsymbol{x}^{\mu} = \mathbf{0} \tag{18}$$

$$\frac{\partial E_J}{\partial \boldsymbol{w}_j} = w_j \sum_{\mu} (f_J^{\mu} - y^{\mu})\ g'(\boldsymbol{w}_j^T \boldsymbol{x}^{\mu})\ \boldsymbol{x}^{\mu} = \mathbf{0} \tag{19}$$

Then we check if regions $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$, $\widehat{\boldsymbol{\Theta}}_J^{\beta}$, and $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ satisfy these necessary conditions. Note that in these regions we have $f_J^{\mu} = f_{J-1}^{\mu}$ and $v_j^{\mu} = z_j^{\mu},\ j = 2, \cdots, J$.

Thus, we see that the first, third, and fifth equations hold, and the second and fourth equations are needed to check.

(1) In region $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$, since weight vector $\boldsymbol{w}_1$ is free, the output of the first hidden unit z_1^{μ} is free, which means it is not guaranteed that the second equation holds. Thus, $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$ is not a singular region in general.
(2) In region $\widehat{\boldsymbol{\Theta}}_J^{\beta}$, since $z_1^{\mu}(= g(w_{10}))$ is independent on μ, the second equation can be reduced to the first one, and holds. However, the fourth equation does not hold in general unless $w_1 = 0$. Thus, the following area included in both $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$ and $\widehat{\boldsymbol{\Theta}}_J^{\beta}$ forms a singular region where w_{10} is free. This region is called $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$ and reducibility mapping from $\widehat{\boldsymbol{\theta}}_{J-1}$ to $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$ is called $\alpha\beta$.

$$\begin{aligned} &w_0 = \widehat{u}_0, \quad w_1 = 0, \quad \boldsymbol{w}_1 = [w_{10}, 0, \cdots, 0]^T \\ &w_j = \widehat{u}_j, \ \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, \quad j = 2, \cdots, J \end{aligned} \tag{20}$$

(3) In region $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$, since $z_1^{\mu} = v_2^{\mu}$, the second and fourth equations hold. Namely, $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ is a singular region. Here we have one degree of freedom since we only have the following restriction.

$$w_1 + w_2 = \widehat{u}_2 \tag{21}$$

3 SSF1.2 (Singularity Stairs Following, ver. 1.2) Method

This section proposes an extended version of SSF [11], which makes good use of the whole singular regions $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ and $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$ of MLP. The proposed method is called SSF1.2, while the original SSF [11] is called SSF1.0 hereafter.

SSF1.2 differs from SSF1.0 twofold. One is the extension of search areas; that is, SSF1.2 searches both $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ and $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$, although SSF1.0 searches only $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$. By searching the whole singular regions, we expect to find better solutions and will get more insight into MLP search space. The other is reduction of the load to search the region $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$; namely, SSF1.2 employs an efficient way as shown below, while SSF1.0 employs an exhaustive way.

Below we explain how to search these two singular regions. It is rather easy to search these regions since either region has only one degree of freedom and most points in the region are saddles [5], which means we surely find a solution of MLP(J) better than the optimum of MLP($J-1$).

As for $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$, SSF1.2 searches the region using the Hessian matrix $\boldsymbol{H}(= \partial^2 E/\partial \boldsymbol{w} \partial \boldsymbol{w}^T)$. Otherwise we cannot move the search point since the region is completely flat. Each negative eigen value of $\boldsymbol{H}$ is picked up, and its eigen vector $\boldsymbol{v}$ and its negative vector $-\boldsymbol{v}$ are selected as two search directions for the eigen value. The appropriate step length is decided using line search called golden section [9]. After the first move, the search is continued using BPQ.

As for $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$, SSF1.2 begins with focusing on the following three points in $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$: a middle interpolation point, an all-or-nothing point, and an extrapolation

point. These points correspond to $p = 0.5$, 1.0, and 1.5 respectively in the following equations. Note that the restriction eq. (21) is satisfied for each p.

$$w_1 = p\ \widehat{u}_2, \qquad w_2 = (1-p)\ \widehat{u}_2 \tag{22}$$

At each of the above three points, the Hessian $\boldsymbol{H}$ is calculated, each negative eigen value of $\boldsymbol{H}$ is picked up, and then its eigen vector $\boldsymbol{v}$ and its negative vector $-\boldsymbol{v}$ are selected as two search directions for the eigen value. The appropriate step length is decided using line search. After the first move, the search is continued using BPQ. The original SSF1.0 searches the region changing initial points many times in the form of interpolation or extrapolation of eq.(21).

The procedure of SSF1.2 is described below. It searches the space by ascending singularity stairs one by one, beginning with MLP(J=1) and gradually increasing J until the intended largest number J_{max}. The optimal MLP(J=1) is found just applying reducibility mapping $\alpha\beta$ to the optimal MLP(J=0); MLP(J=0) is only a constant model. Here $w_0^{(J)}$, $w_j^{(J)}$, and $\boldsymbol{w}_j^{(J)}$ denote weights of MLP(J). Compared with SSF1.0 [11], step 1 requires only two runs instead of many runs, step 2-1 is added to incorporate reducibility mapping $\alpha\beta$, and step 2-2 reduces the number of search points for reducibility mapping γ.

SSF1.2 (Singularity Stairs Following, ver. 1.2):
(step 1) Initialize weights of MLP(J=1) using reducibility mapping $\alpha\beta$:

$$w_0^{(1)} \leftarrow \widehat{w}_0^{(0)} (= \overline{y}), \qquad w_1^{(1)} \leftarrow 0, \qquad \boldsymbol{w}_1^{(1)} \leftarrow [0, 0, \cdots, 0]^T.$$

Pick up each negative eigen value of $\boldsymbol{H}$ and select its eigen vector $\boldsymbol{v}$ and $-\boldsymbol{v}$ as two search directions for the eigen value. Find the appropriate step length using golden section. Then perform MLP(J =1) learning and let the best be $\widehat{w}_0^{(1)}$, $\widehat{w}_1^{(1)}$, and $\widehat{\boldsymbol{w}}_1^{(1)}$. $J \leftarrow 1$.
(step 2) While $J < J_{max}$, repeat the following to get the optimal MLP(J+1) from the optimal MLP(J).
(step 2-1) Initialize weights of MLP(J+1) applying reducibility mapping $\alpha\beta$ to the optimal MLP(J):

$$w_j^{(J+1)} \leftarrow \widehat{w}_j^{(J)},\ j = 0, 1, \cdots, J, \qquad \boldsymbol{w}_j^{(J+1)} \leftarrow \widehat{\boldsymbol{w}}_j^{(J)},\ j = 1, \cdots, J$$
$$w_{J+1}^{(J+1)} \leftarrow 0, \qquad \boldsymbol{w}_{J+1}^{(J+1)} \leftarrow [0, 0, \cdots, 0]^T.$$

Pick up each negative eigen value of $\boldsymbol{H}$ and select its eigen vector $\boldsymbol{v}$ and $-\boldsymbol{v}$ as two search directions for the eigen value. Find the appropriate step length using golden section. Then perform MLP(J+1) learning and keep the best as the best MLP(J+1) of $\alpha\beta$.
(step 2-2) If there are more than one hidden units in MLP(J), repeat the following for each hidden unit $m (= 1, \cdots, J)$ to split.
Initialize weights of MLP(J+1) using reducibility mapping γ:

$$w_j^{(J+1)} \leftarrow \widehat{w}_j^{(J)},\ j \in \{0, 1, \cdots, J\} \setminus \{m\}, \quad \boldsymbol{w}_j^{(J+1)} \leftarrow \widehat{\boldsymbol{w}}_j^{(J)},\ j = 1, \cdots, J$$
$$\boldsymbol{w}_{J+1}^{(J+1)} \leftarrow \widehat{\boldsymbol{w}}_m^{(J)}.$$

Initialize $w_m^{(J+1)}$ and $w_{J+1}^{(J+1)}$ three times as follows with p=0.5, 1.0, and 1.5.

$$w_m^{(J+1)} \leftarrow p\ \widehat{w}_m^{(J)}, \qquad w_{J+1}^{(J+1)} \leftarrow (1-p)\ \widehat{w}_m^{(J)}$$

At each of the above three points, pick up each negative eigen value of $\boldsymbol{H}$ and

select its eigen vector $\boldsymbol{v}$ and $-\boldsymbol{v}$ as two search directions for the eigen value. Find the appropriate step length using line search. Then perform MLP(J+1) learning and keep the best as the best MLP(J+1) of γ for m.
(step 2-3) Among the best MLP(J+1) of $\alpha\beta$ and the best MLP(J+1)s of γ for different m, select the true best and let the weights be $\widehat{w}_0^{(J+1)}, \widehat{w}_j^{(J+1)}, \widehat{\boldsymbol{w}}_j^{(J+1)}$, j=1, $\cdots$, J+1. Then $J \leftarrow J$+1.

Now we claim the following, which will be evaluated in our experiments.
(1) Compared with existing methods such as BP, Newton's method, quasi-Newton methods, SSF1.2 will find excellent solutions of MLP(J) with much higher probabilities.
(2) The excellent solution of MLP(J) will be obtained one after another for $J = 1, \cdots, J_{max}$. These excellent solutions can be used for model selection. SSF1.2 guarantees that the solution of MLP(J+1) is better than that of MLP(J) since SSF1.2 descends in MLP(J+1) search space from the singular region corresponding to the optimal solution of MLP(J). Such monotonic decrease of training error is not guaranteed for the existing methods.
(3) Since SSF1.2 uses much few search points than the original SSF1.0, SSF1.2 will be much faster than SSF1.0. SSF1.2 will also be faster than the existing methods if they are performed many times changing initial weights.

4 Experiments

We evaluate the proposed SSF1.2 for sigmoidal and polynomial-type MLPs using artificial and real data sets. Activation functions $g(h)$ in eq. (1) for sigmoidal and polynomial-type MLPs are $g(h) = 1/(1 + e^{-h})$ and $g(h) = \exp(h)$ respectively. The output of polynomial-type MLP is written as follows.

$$f_J = \sum_{j=0}^{J} w_j z_j, \qquad z_j = \exp\left(\sum_{k=1}^{K} w_{jk} \ln x_k\right) \tag{23}$$

The above can be rewritten as below, representing a multivariate polynomial where w_j represents a coefficient and w_{jk} represents a power [10].

$$f_J = \sum_{j=0}^{J} w_j z_j, \qquad z_j = \prod_{k=1}^{K} (x_k)^{w_{jk}} \tag{24}$$

In performing SSF, since we have to move in singular flat regions, we employ weak weight decay. As a learning engine of SSF we use a kind of quasi-Newton method called BPQ [12] since any first-order method is too slow.

As existing learning methods we employed BP and BPQ for comparison. Here the learning rate of BP is set to be 0.001, since a relatively large learning rate does not work well. BP or BPQ is performed 100 times for each MLP(J) of each data set.

SSF or BPQ stops when a step length is less than 10^{-30} or the iteration exceeds 20,000 sweeps. BP stops when the training error is not improved any more or the iteration exceeds 200,000 sweeps. As for the initialization of MLP weights, w_{jk} and w_j are randomly selected from the range $[-1,\ 1]$, except a bias $w_0 = \overline{y}$.

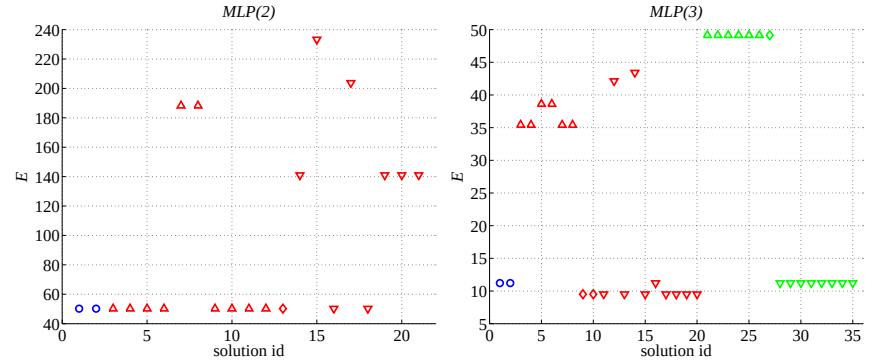
4.1 Experiment of Sigmoidal MLP using Artificial Data

An artificial data set for sigmoidal MLP was generated using MLP having the following weights. Values of each explanatory variable $x_1, x_2, \cdots, x_8$ were randomly selected from the range $[0,\ 1]$, while values of y were generated by adding small Gaussian noise $\mathcal{N}(0, 0.05^2)$ to MLP outputs. Note that four variables $x_5, \cdots, x_8$ are irrelevant. The sample size was 100 and the penalty coefficient was set to be $\lambda = 0.01$. The maximum number of hidden units was $J_{max} = 6$, which includes the correct number $J^* = 4$.

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 15 \\ -20 \\ 16 \\ 8 \end{pmatrix}, \qquad (\boldsymbol{w}_1\ \boldsymbol{w}_2\ \boldsymbol{w}_3\ \boldsymbol{w}_4) = \begin{pmatrix} 5 & 6 & 7 & 5 \\ -8 & 12 & 8 & 3 \\ -7 & 8 & 5 & 10 \\ 8 & -9 & -10 & 7 \\ 5 & -12 & -6 & -8 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{25}$$

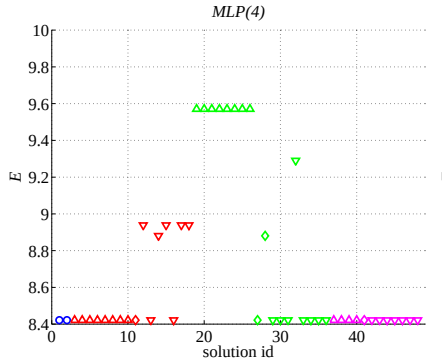
Figure 1 shows the result of SSF1.2. In each figure a horizontal axis simply indicates solution id (that is, search point id), and a vertical axis means final sum-of-squares error E. Throughout this paper a circle indicates $\alpha\beta$ search, while a triangle, diamond, and upside-down triangle indicate interpolation (p=0.5), all-or-nothing point (p=1.0), and extrapolation (p=1.5 in eq.(22)) of γ search respectively. We ran two MLP(J=1) learnings twice to obtain different solutions, and the better was used for the next step. The result for MLP(J=2) is shown in Fig. 1 (a). We have two search points for reducibility mapping $\alpha\beta$ search, and 19 points for reducibility mapping γ search. The best was used for the next step. The results for MLP(J=3, 4, and 6) are shown in Fig. 1 (b), (c), and (d) respectively. The total number of search points was 233 (=2+21+35+48+56+71). Since the number was 1615 (=100+101+202+303+404+505) for SSF1.0 [11], SSF1.2 reduced the total number of search points by 6.93 (=1615/233) times. We see the best solution was frequently obtained from different splitting.

As existing methods we ran BP and BPQ 100 times each. Table 1 compares the best training error E for each J. SSF1.2, SSF1.0 and BPQ achieved exactly the same training error for each J. This is probably because finding the best solution of each J seems relatively easy for this problem only if we use a 2nd-order method such as BPQ. However, we cannot get such best solutions if we use a 1st-order method such as BP. Moreover, all three showed monotonic

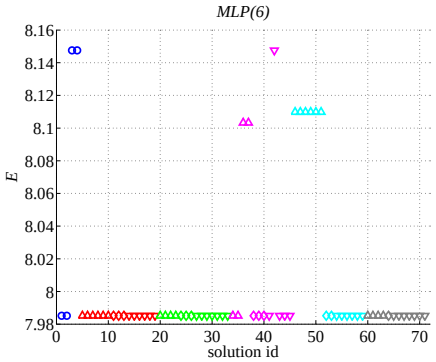


(a) from MLP(1) to MLP(2)

(b) from MLP(2) to MLP(3)



(c) from MLP(3) to MLP(4)



(d) from MLP(5) to MLP(6)

Fig. 1 Learning process of SSF1.2 for artificial sigmoidal data

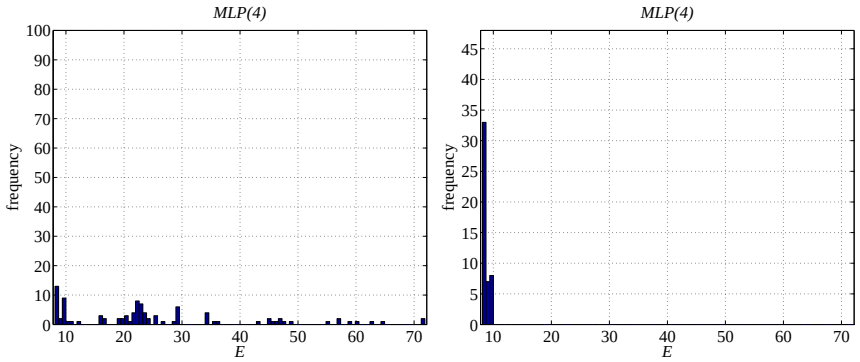
decrease of E as J increased. On the other hand, BP could not improve so much for $J \geq 3$ and did not show monotonic decrease of E as J increased.

Table 1 Best training error comparison for artificial sigmoidal data

J	BP	BPQ	SSF1.0	SSF1.2
1	321.3509	290.5380	290.5380	290.5380
2	53.9222	50.2203	50.2203	50.2203
3	44.6509	9.4990	9.4990	9.4990
4	45.9678	8.4217	8.4217	8.4217
5	45.0472	8.1476	8.1476	8.1476
6	43.7981	7.9851	7.9851	7.9851

Figure 2 compares histograms of BPQ and SSF1.2 solutions for MLP(J=4). Here each histogram is created by dividing the value range (max value - min value) of E into 100 equal parts. SSF1.2 reached the true solution 33 times out of 48 with probability 0.69, while BPQ reached the true solution 13 times out of 100 with probability 0.13. We see SSF1.2 found the excellent solution

5.3 times more stably. Moreover, many solutions of SSF1.2 are located very close to the true solution, while BPQ solutions are widely scattered.



(a) histogram of BPQ solutions (b) histogram of SSF1.2 solutions

Fig. 2 Histograms of solutions for artificial sigmoidal data

Table 2 compares CPU time spent for learning artificial sigmoidal data. SSF1.2 is 6.16 times faster than SSF1.0 mainly because the number of search points is greatly reduced, as stated above. For this data, SSF1.2 is 2.19 times faster than BPQ. BP was 505.7 times slower than SSF1.2 and all runs stopped by reaching the iteration upper bound.

Table 2 CPU time comparison for artificial sigmoidal data (sec)

J	BP	BPQ	SSF1.0	SSF1.2
1	1414.31	3.24	3.23	0.07
2	1576.20	3.19	4.74	1.00
3	1615.36	5.80	8.39	1.66
4	1652.16	7.98	23.74	3.80
5	1660.54	9.54	32.05	5.28
6	1664.44	11.82	44.53	7.15
total	9583.01	41.57	116.68	18.95

Table 3 compares validation error using leave-one-out. SSF1.2, SSF1.0 and BPQ showed the same validation performance since they reached the same weights in learning for each J. They indicated the best validation error at J=4, which is correct. BP showed excellent validation better than SSF or BPQ when J=4 and 6; however, BP indicated the best validation error at J=6, which is wrong. The validation performance of BP may not be trustworthy.

4.2 Experiment of Sigmoidal MLP using Real Data

As real data for sigmoidal MLP we used Computer Hardware data from UCI ML Repository. The number of explanatory variables is 6, and the sample size

Table 3 Validation error comparison for artificial sigmoidal data

J	BP	BPQ	SSF1.0	SSF1.2
1	338.6513	359.2869	359.2869	359.2869
2	92.7148	100.7066	100.7066	100.7066
3	71.1548	34.9033	34.9033	34.9033
4	24.6823	30.0909	30.0909	30.0909
5	35.4931	36.7126	36.4651	36.7126
6	24.5955	39.2102	39.2102	39.2102

is 209. The penalty coefficient was $\lambda = 0.001$, and the maximum number of hidden units was $J_{max} = 6$.

Figure 3 shows the result of SSF1.2. We ran MLP(J=1) learning twice and obtained the same solution, which was used for the next step. The result for MLP(J=2) is shown in Fig. 3 (a). We have two search points for reducibility mapping $\alpha\beta$ search, and 14 points for γ search. Here we have several solutions and the best was used for the next step. The results for MLP(J=3, 5, and 6) are shown in Fig. 3 (b), (c), and (d) respectively. The total number of search points was 189 (=2+16+29+38+48+56). Since the number was 1615 (=100+101+202+303+404+505) for SSF1.0 [11], SSF1.2 reduced the total number of search points by 8.54 (=1615/189) times. We see the best solution was frequently obtained from different splitting. For MLP(J=6), 43 search points out of 56 converged to much the same excellent solution.

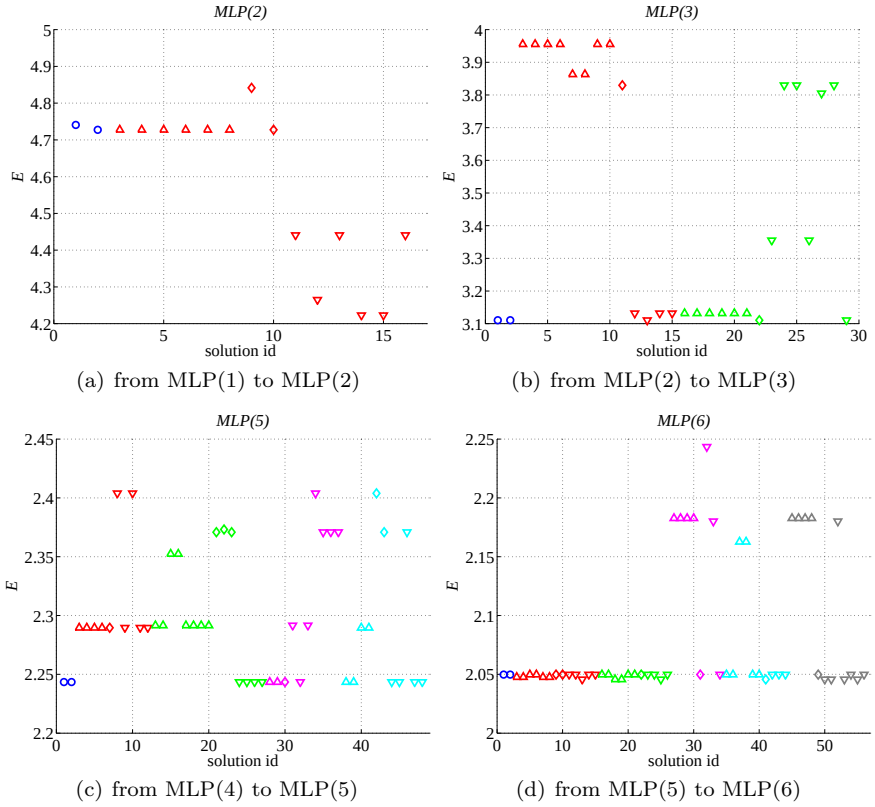
For comparison we ran BP and BPQ 100 times each. Table 4 compares the best training error E for each J. SSF and BPQ achieved much the same good training error for most J, while BP achieved rather poor results for each J. Moreover, SSF and BPQ showed preferable monotonic decrease as J increased; however, BP did not.

Table 4 Best training error comparison for Computer Hardware data

J	BP	BPQ	SSF1.0	SSF1.2
1	8.2336	6.6410	6.6410	6.6410
2	7.6651	4.2228	4.4408	4.2228
3	7.0291	3.0119	3.0976	3.1106
4	7.5801	2.3973	2.6093	2.4039
5	7.6931	2.1152	2.3028	2.2434
6	7.5201	2.0414	2.0414	2.0457

Figure 4 compares histograms of BPQ and SSF1.2 solutions for MLP(J=6). SSF1.2 reached the excellent solution 43 times out of 56 with probability 0.86, while BPQ found the excellent solution 5 times out of 100 runs with probability 0.05. Moreover, many solutions of SSF1.2 are located close to the excellent solution, while BPQ solutions are rather widely scattered. We see SSF1.2 found the excellent solution much more stably.

Table 5 compares CPU time spent for learning Computer Hardware data. SSF1.2 is 7.49 times faster than SSF1.0 mainly because the number of search points is greatly reduced, as stated previously. For this data, SSF1.2 is 3.53



(a) from MLP(1) to MLP(2)

(b) from MLP(2) to MLP(3)

(c) from MLP(4) to MLP(5)

(d) from MLP(5) to MLP(6)

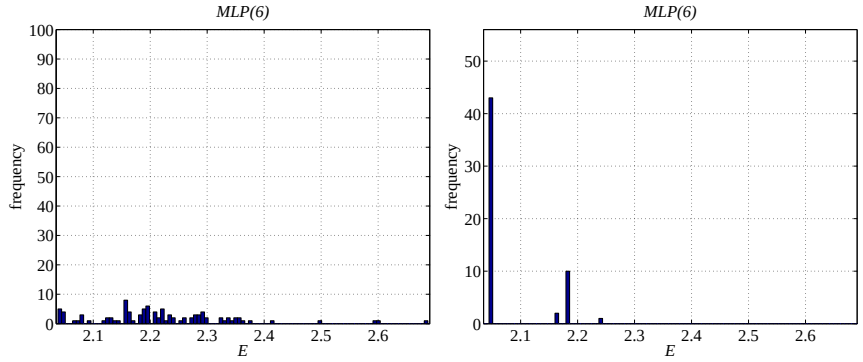
Fig. 3 Learning process of SSF1.2 for Computer Hardware data

times faster than BPQ. BP was quite slow and all runs stopped by reaching the iteration upper bound.

Table 5 CPU time comparison for Computer Hardware data (sec)

J	BP	BPQ	SSF1.0	SSF1.2
1	1355.73	2.54	2.22	0.15
2	1088.75	5.38	4.90	0.90
3	1097.04	11.41	18.90	2.53
4	1100.22	18.25	31.35	4.16
5	1110.52	28.75	51.61	7.76
6	1118.13	24.78	84.48	10.34
total	6870.39	91.13	193.45	25.84

Table 6 compares validation error using leave-one-out. SSF1.2 achieved the best validation error at J=4. BP also supports J=4, although BPQ and SSF1.0 indicate different J. The best validation error of SSF1.2 is better than that of SSF1.0.



(a) histogram of BPQ solutions (b) histogram of SSF1.2 solutions

Fig. 4 Histograms of solutions for Computer Hardware data

Table 6 Validation error comparison for Computer Hardware data

J	BP	BPQ	SSF1.0	SSF1.2
1	9.2725	11.4714	11.4714	11.4714
2	6.2712	10.7275	7.0776	10.7275
3	6.1397	12.5215	9.7083	15.4312
4	5.3707	7.5827	11.6553	4.4005
5	5.4525	5.6972	11.4055	6.6768
6	5.8105	5.9416	5.9416	5.9924

4.3 Experiment of Polynomial-type MLP using Artificial Data

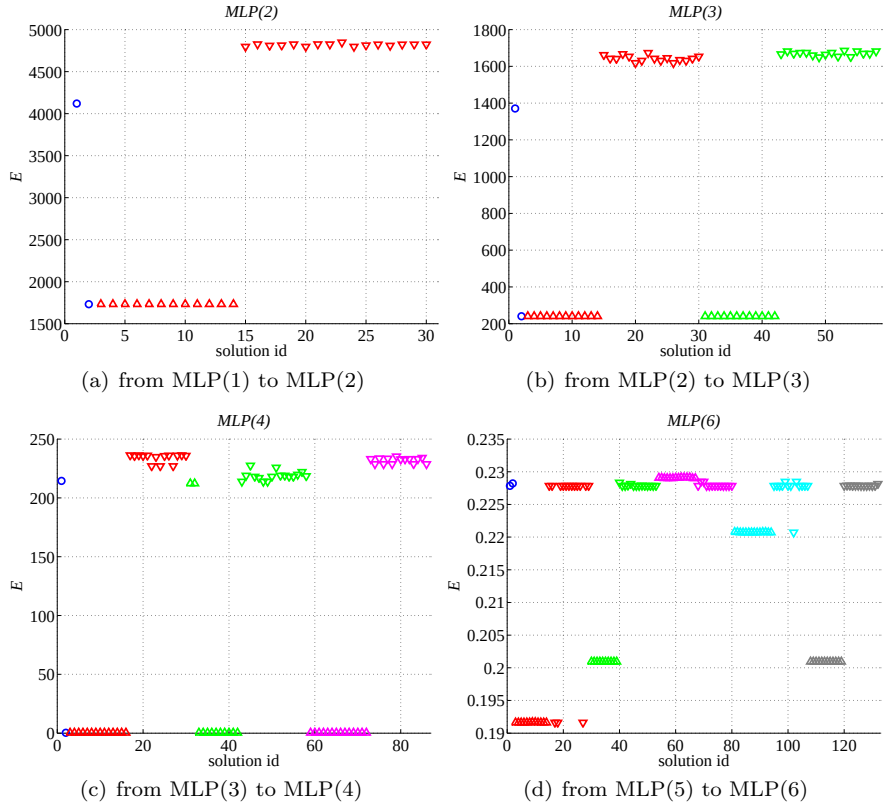
Here we consider the following multivariate polynomial.

$$y = 2 + 60\ x_1^3\ x_2^6 x_3 + 40\ x_4^8\ x_5 + 20\ x_6\ x_7^7 + 10\ x_2\ x_5^8 \tag{26}$$

Values of each variable $x_1, x_2, \cdots, x_{14}$ were randomly selected from the range $(0, 1)$, while y values were generated following eq. (26) with the addition of small Gaussian noise $\mathcal{N}(0, 0.01^2)$. Seven variables $x_8, \cdots, x_{14}$ are irrelevant. The sample size was 500 and the penalty coefficient was $\lambda = 0.0001$. The maximum number of hidden units was $J_{max} = 6$ to include the right $J^* = 4$.

Figure 5 shows the result of SSF1.2. We ran MLP(J=1) learning twice and obtained different solutions; the better was used for the next step. The result for MLP(J=2) is shown in Fig. 5 (a). We have two search points for reducibility mapping $\alpha\beta$ search, and 28 points for reducibility mapping γ search. The best solution was used for the next step. The results for MLP(J=3, 4, and 6) are shown in Fig. 5 (b), (c) and (d) respectively. The total number of search points was 415 (=2+30+58+86+107+132). Since the number was 1615 (=100+101+202+303+404+505) for SSF1.0 [11], SSF1.2 reduced the total number of search points by 3.89 (1615/415) times. The figures show the best solution was frequently obtained from different splitting.

For comparison we ran BP and BPQ 100 times each. Table 7 compares the best training error E for each J. SSF1.2 and SSF1.0 found exactly the same solution for each J, greatly reducing the training error when $J \geq 4$.



(a) from MLP(1) to MLP(2)

(b) from MLP(2) to MLP(3)

(c) from MLP(3) to MLP(4)

(d) from MLP(5) to MLP(6)

Fig. 5 Learning process of SSF1.2 for artificial polynomial data

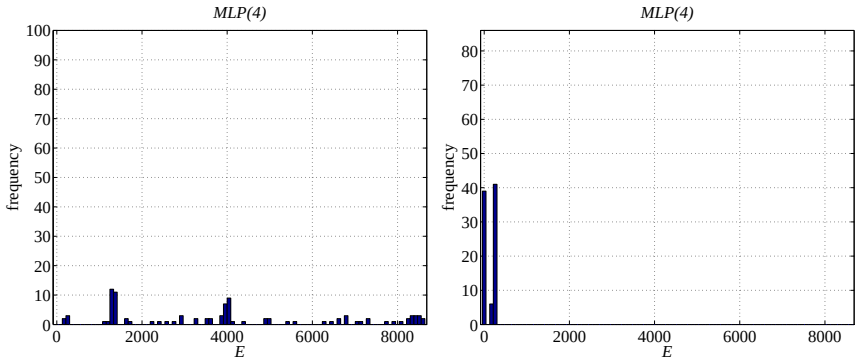
BPQ greatly reduced the training error when $J \geq 5$. BP stayed very poor even when J increased. Moreover, SSF and BPQ showed monotonic decrease for the increase of J; however, BP did not.

Table 7 Best training error comparison for artificial polynomial data

J	BP	BPQ	SSF1.0	SSF1.2
1	11281.7846	4951.5529	4951.5529	4951.5529
2	11325.9183	4120.1137	1731.6786	1731.6786
3	11549.9327	1370.8722	240.4430	240.4430
4	11142.7910	214.4255	0.3180	0.3180
5	11327.0229	0.3165	0.2292	0.2292
6	11455.7078	0.2278	0.1916	0.1916

Figure 6 compares histograms of BPQ and SSF1.2 solutions for MLP(J=4). SSF1.2 reached the true solution 39 times out of 86 with probability 0.45, while BPQ could not find the true solution for any 100 runs. Moreover, many solutions of SSF1.2 are located quite close to the true solution, while BPQ

solutions are very widely distributed. Compared with BPQ, SSF1.2 found the excellent solution much more stably.



(a) histogram of BPQ solutions (b) histogram of SSF1.2 solutions

Fig. 6 Histograms of solutions for artificial polynomial data

Table 8 compares CPU time spent for learning artificial polynomial data. SSF1.2 was 5.00 times faster than SSF1.0 due to the great reduction of the number of search points. SSF1.2 is 4.47 times faster than BPQ for this data. BP was fast but its training error was very poor, which means BP stopped in an early stage because BP could not improve the training error any more.

Table 8 CPU time comparison for artificial polynomial data (sec)

J	BP	BPQ	SSF1.0	SSF1.2
1	166.19	72.66	73.19	0.89
2	123.93	139.83	16.36	6.56
3	55.96	191.57	46.14	15.66
4	59.26	205.57	85.34	29.21
5	44.13	206.35	422.21	67.29
6	37.04	237.00	534.76	115.86
total	486.50	1052.97	1178.01	235.48

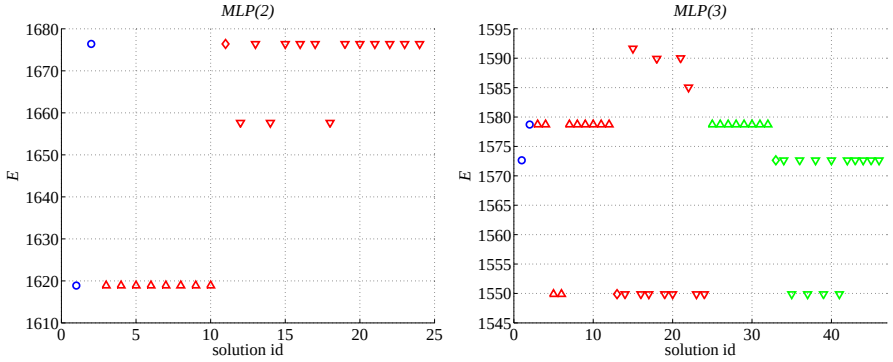
Table 9 compares validation error using leave-one-out. SSF1.2 and SSF1.0 showed the same performance reducing greatly at more than $J \geq 4$. Both SSF achieved the best validation error at J=4, which is correct. BPQ indicated wrong J=5 and validation error of BP remained rather poor.

4.4 Experiment of Polynomial-type MLP using Real Data

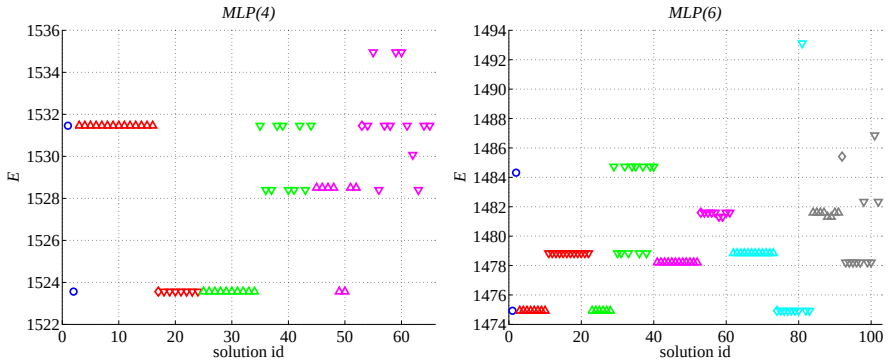
As real data for polynomial-type MLP we used White Wine data from UCI ML Repository. The number of explanatory variables is 10, and the sample size is 4,998. The penalty coefficient was $\lambda = 0.001$, and the maximum number of hidden units was $J_{max} = 6$.

Table 9 Validation error comparison for artificial polynomial data

J	BP	BPQ	SSF1.0	SSF1.2
1	12404.4112	5331.5387	5331.5387	5331.5387
2	12404.5434	4678.9544	2299.5675	2299.5675
3	12401.5252	2438.7316	462.4028	462.4028
4	12413.3403	511.3041	0.0434	0.0434
5	12416.8616	0.0540	0.0533	0.0533
6	12403.2163	0.0803	0.0646	0.0646



(a) from MLP(1) to MLP(2) (b) from MLP(2) to MLP(3)



(c) from MLP(3) to MLP(4) (d) from MLP(5) to MLP(6)

Fig. 7 Learning process of SSF1.2 for white wine data

Figure 7 shows the result of SSF1.2. We ran MLP(J=1) learning twice and obtained the same solution, which was used for the next step. The result for MLP(J=2) is shown in Fig. 7 (a). We have two search points for reducibility mapping $\alpha\beta$ search, and 22 points for γ search. Here we have three solutions and the best was used for the next step. The results for MLP(J=3, 4, and 6) are shown in Fig. 7 (b), (c), and (d) respectively. The total number of search points was 324 (=2+24+46+65+85+102). Since the number was 1615 (=100+101+202+303+404+505) for SSF1.0 [11], SSF1.2 reduced the total number of search points by 4.98 (=1615/324) times. We see the best solution

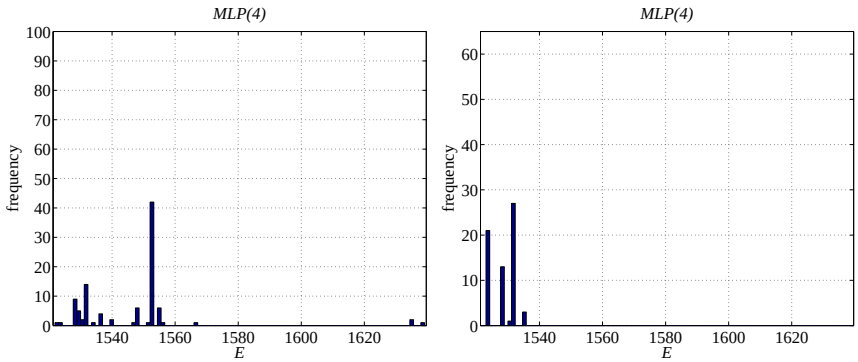
was obtained from different splitting. For MLP(J=4), 21 search points out of 65 converged to much the same excellent solution.

For comparison we ran BP and BPQ 100 times each. Table 10 compares the best training error E for each J. SSF and BPQ achieved much the same good training error for most J, while BP improved training error only gradually as J increased. Moreover, each method showed preferable monotonic decrease as J increased.

Table 10 Best training error comparison for white wine data

J	BP	BPQ	SSF1.0	SSF1.2
1	1793.83	1725.74	1725.74	1725.74
2	1722.96	1618.88	1618.88	1618.88
3	1699.03	1549.89	1549.89	1549.89
4	1689.21	1522.49	1523.56	1523.56
5	1680.47	1498.08	1498.86	1498.86
6	1665.78	1478.84	1474.93	1474.93

Figure 8 compares histograms of BPQ and SSF1.2 solutions for MLP(J=4). SSF1.2 reached the excellent solution 21 times out of 65 with probability 0.32, while BPQ did only once out of 100 runs with probability 0.01. Moreover, many solutions of SSF1.2 are located close to the excellent solution, while BPQ solutions scatter more widely. We see SSF1.2 found the excellent solution 32 times more stably.



(a) histogram of BPQ solutions (b) histogram of SSF1.2 solutions

Fig. 8 Histograms of solutions for white wine data

Table 11 compares CPU time spent for learning White Wine data. SSF1.2 is 4.96 times faster than SSF1.0 mainly because the number of search points is greatly reduced. For this data, SSF1.2 is 1.55 times faster than BPQ. BP was slow and most runs stopped by reaching the iteration upper bound.

Table 11 CPU time comparison for ball bearings data (sec)

J	BP	BPQ	SSF1.0	SSF1.2
1	996.41	23.41	23.36	0.48
2	1000.53	84.80	90.43	22.60
3	1021.26	139.71	275.37	74.72
4	1015.99	200.62	390.91	84.17
5	955.13	258.58	977.96	224.26
6	927.28	235.53	1249.01	200.53
total	5916.58	942.66	3007.05	606.76

Table 12 compares validation error using 10-fold cross-validation. SSF and BPQ achieved the best validation error at J=4, while BP supports J=6. The best validation error of SSF1.2 is the same as that of SSF1.0.

Table 12 Validation error comparison for White Wine data

J	BP	BPQ	SSF1.0	SSF1.2
1	1824.71	1739.79	1739.79	1739.79
2	1736.97	1655.12	1655.12	1655.12
3	1755.98	1611.81	1611.81	1611.81
4	1750.16	1569.16	1577.01	1577.01
5	1736.80	3799.64	4276.88	4276.87
6	1723.39	4356.03	5083.01	5083.05

5 Conclusion

This paper proposed a new MLP learning method called SSF1.2, which makes good use of the whole singular regions. It begins with MLP(J=1) and gradually increases J, the number of hidden units, one by one to successively find excellent solutions for each J. Our experiments using sigmoidal and polynomial-type MLP showed the following. Compared with the original SSF1.0, SSF1.2 is surely much faster and may improve solution quality as well. Moreover, compared with existing methods such as BP or quasi-Newton method, SSF1.2 more stably and more efficiently found excellent solutions commensurate with each J. Successive solutions obtained by SSF1.2 were useful for MLP model selection. In the future we plan to improve our method by applying it to data requiring heavy computation or having larger data size or different types of nonlinearity. Additionally, we will investigate model selection for singular models since successive excellent solutions for each J and learning processes can be used for such model selection.

Acknowledgments.

This work was supported by Grants-in-Aid for Scientific Research (C) 22500212 and Chubu University Grant 24IS27A.

References

1. Amari S (1998) Natural gradient works efficiently in learning. Neural Comput, 10(2): 251–276
2. Amari S, Park H, Fukumizu K (2000) Adaptive method of realizing natural gradient learning for multilayer perceptrons. Neural Comput, 12(6): 1399–1409
3. Cousseau F, Oseki T, Amari S (2008) Dynamics of learning in multilayer perceptrons near singularities. IEEE Trans Neural Networks, 19(8): 1313–1328
4. Duda RO, Hart PE, Stork DG (2001) Pattern classification, 2nd edition. John Wiley & Sons, Inc., New York
5. Fukumizu K, Amari S (2000) Local minima and plateaus in hierarchical structure of multilayer perceptrons. Neural Networks, 1(3): 317–327
6. Hamey LGC (1998) XOR has no local minima: a case study in neural network error surface. Neural Networks, 11(4): 669–681
7. Hecht-Nielsen H (1990) Neurocomputing. Addison-Wesley Publishing Company, Reading, Massachusetts
8. Minnett RCJ, Smith AT, Lennon Jr.WC, Hecht-Nielsen R (2011) Neural network tomography: network replication from output surface geometry. Neural Networks, 24(5): 484–492
9. Luenberger DG (1984) Linear and nonlinear programming. Addison-Wesley Publishing Company, Reading, Massachusetts
10. Nakano R, Saito K (2002) Discovering polynomials to fit multivariate data having numeric and nominal variables. LNAI, vol.2281, pp.482–493
11. Nakano R, Satoh S, Ohwaki T (2011) Learning method utilizing singular region of multilayer perceptron. In: Proceedings of the 3rd International Conference on Neural Computation Theory and Applications, Paris, pp.106–111
12. Saito K, Nakano R (1997) Partial BFGS update and efficient step-length calculation for three-layer neural networks. Neural Comput, 9(1): 239–257
13. Sussmann HJ (1992) Uniqueness of the weights for minimal feedforward nets with a given input-output map. Neural Networks, 5(4): 589–593
14. Wan W (2006) Implementing online natural gradient learning: problems and solutions. IEEE Trans Neural Networks, 17(2): 317–329
15. Watanabe S (2008) A formula of equations of states in singular learning machines. In: Proceedings of the International Joint Conference on Neural Networks 2008, Hong Kong, pp.2099–2106
16. Watanabe S (2009) Algebraic geometry and statistical learning theory. Cambridge University Press, Cambridge, UK