Multilayer Perceptron Learning Utilizing Singular Regions and Search Pruning

Seiya Satoh and Ryohei Nakano

***Abstract*—In a search space of a multilayer perceptron having J hidden units, MLP(J), there exist flat areas called singular regions. Since singular regions cause serious stagnation of learning, a learning method to avoid them was once proposed, but was not guaranteed to find excellent solutions. Recently, SSF1.2 was proposed which utilizes singular regions to stably and successively find excellent solutions commensurate with MLP(J). However, SSF1.2 has a problem that it takes longer as J gets larger. This paper proposes a learning method SSF1.3 that enhances SSF1.2 by attaching search pruning so as to discard a search whose route is similar to one of previous searches. Our experiments showed SSF1.3 ran several times faster than SSF1.2 without degrading solution quality.**

***Index Terms*—multilayer perceptron, learning method, singular region, reducibility mapping, search pruning**

I. Introduction

In a parameter space of MLP(J), a multilayer perceptron with J hidden units, there exist flat areas called singular regions created by reducibility mapping [1], and such a region causes stagnation of learning [2]. Hecht-Nielsen once pointed out MLP parameter space is full of flat areas and troughs [3], and recent experiments [4] revealed most points have huge condition numbers ($> 10^6$).

Natural gradient [5] was once proposed to avoid singular regions, but even the method may get stuck in singular regions and is not guaranteed to find an excellent solution.

It is known that many useful statistical models, such as MLP, GM, and HMM, are singular models having singular regions. Intensive theoretical research has been done to clarify mathematical features of singular models and especially Watanabe has produced his own singular learning theory [6]; however, experimental research has been rather insufficient so far to fully support the theories.

Recently a rather unusual learning method called SSF (Singularity Stairs Following) [4], [7] was proposed for MLP learning, which does not avoid but makes good use of singular regions of MLP search space. The latest version SSF1.2 [7] successively and stably finds excellent solutions, starting with MLP(J=1) and then gradually increases J. When increasing J, SSF1.2 utilizes the optimum of MLP($J-1$) to form two kinds of singular regions in MLP(J) parameter space. Thus, SSF1.2 can monotonically improve the solution quality along with the increase of J. The processing time of SSF1.2, however, gets very long as J gets large due to the increase of search routes. Moreover, it was observed that many SSF runs converged into very limited number of solutions, indicating the great potential for search pruning.

This work was supported in part by Grants-in-Aid for Scientific Research (C) 25330294 and Chubu University Grant 24IS27A.

S. Satoh and R. Nakano are with the Department of Computer Science, Graduate School of Engineering, Chubu University, 1200 Matsumoto-cho, Kasugai 487-8501, Japan. email: tp13801-3493@sti.chubu.ac.jp and nakano@cs.chubu.ac.jp

This paper proposes a learning method called SSF1.3 which enhances SSF1.2 by attaching search pruning so as to prune any search going through a route similar to one of previous searches. Our experiments using artificial and real data sets showed SSF1.3 ran several times faster than SSF1.2 without degrading solution quality.

II. Singular Regions of Multilayer Perceptron

This section explains how the optimum of MLP($J-1$) can be used to form singular regions in MLP(J) parameter space [2], [7]. Hereafter the boldface indicates a vector.

Consider MLP(J) having J hidden units and one output unit. MLP(J) having parameters $\boldsymbol{\theta}_J$ outputs $f_J(\boldsymbol{x};\boldsymbol{\theta}_J)$ for input vector $\boldsymbol{x} = (x_k)$. Let $\boldsymbol{x}$ be K-dimensional.

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

Here $g(h)$ is an activation function and $\boldsymbol{a}^T$ is the transpose of $\boldsymbol{a}$; $\boldsymbol{\theta}_J = \{w_0, w_j, \boldsymbol{w}_j, j = 1, \cdots, J\}$, where $\boldsymbol{w}_j = (w_{jk})$.

Given training data $\{(\boldsymbol{x}^\mu, y^\mu), \mu = 1, \cdots, N\}$, we consider finding $\boldsymbol{\theta}_J$ that minimizes the following error function.

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

Moreover we consider MLP($J-1$) having $J-1$ hidden units and one output unit. MLP($J-1$) with parameters $\boldsymbol{\theta}_{J-1} = \{u_0, u_j, \boldsymbol{u}_j, j = 2, \cdots, J\}$ outputs the following:

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

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

$$E_{J-1}(\boldsymbol{\theta}) \;=\; \frac{1}{2}\sum_{\mu=1}^{N}(f_{J-1}^\mu - y^\mu)^2 \tag{4}$$

Here $f_{J-1}^\mu \equiv f_{J-1}(\boldsymbol{x}^\mu;\boldsymbol{\theta}_{J-1})$. Then, let $\widehat{\boldsymbol{\theta}}_{J-1} = \{\widehat{u}_0, \widehat{u}_j, \widehat{\boldsymbol{u}}_j, j = 2, \cdots, J\}$ be the optimum of MLP($J-1$).

Now we introduce the following 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 $\widehat{\boldsymbol{\theta}}_{J-1}$ of MLP($J-1$). Here let $m = 2, \cdots, J$ in the last mapping.

$$\widehat{\boldsymbol{\theta}}_{J-1} \xrightarrow{\alpha} \widehat{\boldsymbol{\Theta}}_J^\alpha, \quad \widehat{\boldsymbol{\theta}}_{J-1} \xrightarrow{\beta} \widehat{\boldsymbol{\Theta}}_J^\beta, \quad \widehat{\boldsymbol{\theta}}_{J-1} \xrightarrow{\gamma} \widehat{\boldsymbol{\Theta}}_J^\gamma \tag{5}$$

$$\begin{aligned}\widehat{\boldsymbol{\Theta}}_J^\alpha \;\equiv\; & \{\boldsymbol{\theta}_J |\; w_0 = \widehat{u}_0,\; w_1 = 0,\\ & w_j = \widehat{u}_j, \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, j = 2, \cdots, J\}\end{aligned} \tag{6}$$

$$\begin{aligned}\widehat{\boldsymbol{\Theta}}_J^\beta \;\equiv\; & \{\boldsymbol{\theta}_J |\; w_0 + w_1 g(w_{10}) = \widehat{u}_0,\; \boldsymbol{w}_1 = [w_{10}, 0, \cdots, 0]^T,\\ & w_j = \widehat{u}_j, \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, j = 2, \cdots, J\}\end{aligned} \tag{7}$$

$$\begin{aligned}\widehat{\boldsymbol{\Theta}}_J^\gamma \;\equiv\; & \{\boldsymbol{\theta}_J |\; w_0 = \widehat{u}_0,\; w_1 + w_m = \widehat{u}_m,\; \boldsymbol{w}_1 = \boldsymbol{w}_m = \widehat{\boldsymbol{u}}_m,\\ & w_j = \widehat{u}_j, \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j, j = \{2, \cdots, J\} \setminus \{m\}\}\end{aligned} \tag{8}$$

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

By checking the necessary conditions for the critical point of MLP(J), we have the following result [7], which means there are two kinds of singular regions $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$ and $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ in MLP(J) parameter space.

(1) Region $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$ is $(K+1)$-dimensional since free vector $\boldsymbol{w}_1$ is $(K+1)$-dimensional. In this region since $\boldsymbol{w}_1$ is free, the output of the first hidden unit z_1^{μ} is free, which means the necessary conditions do not always hold. Thus, $\widehat{\boldsymbol{\Theta}}_J^{\alpha}$ is not a singular region in general.

(2) Region $\widehat{\boldsymbol{\Theta}}_J^{\beta}$ is two-dimensional since all we have to do is to satisfy $w_0 + w_1\ g(w_{10}) = \widehat{u}_0$. In this region $z_1^{\mu}(= g(w_{10}))$ is independent on μ; however, some necessary condition 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 only w_{10} is free. The 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 $\alpha\beta$.

$$\begin{aligned} &w_0 = \widehat{u}_0,\ w_1 = 0,\ \boldsymbol{w}_1 = [w_{10}, 0, \cdots, 0]^T, \\ &w_j = \widehat{u}_j,\ \boldsymbol{w}_j = \widehat{\boldsymbol{u}}_j,\ j = 2, \cdots, J \end{aligned} \tag{9}$$

(3) Region $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ is a line since we have only to satisfy $w_1 + w_m = \widehat{u}_m$. In this region all the necessary conditions hold since $z_1^{\mu} = v_m^{\mu}$. Namely, $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$ is a singular region. Here we have a line since we only have the following restriction.

$$w_1 + w_m = \widehat{u}_m \tag{10}$$

III. Singularity Stairs Following

This section explains the framework of SSF (singularity stairs following) [4], [7]. SSF is a search method which makes good use of singular regions.

Here, we consider the following four technical points of SSF. SSF1.2 [7] and the proposed SSF1.3 share the first three points, but they are different in the last point.

The first point is on search areas; that is, SSF1.2 and 1.3 search whole two kinds of singular regions $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$ and $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$. By searching the whole singular regions, SSF1.2 and 1.3 will find better solutions than SSF1.0 [4].

The second point is how to start search from singular regions. Since a singular region is flat, any method based on gradient cannot move. Thus, SSF1.0 employs weak weight decay, distorting the original search space, while SSF1.2 and 1.3 employ the Hessian $\boldsymbol{H}(= \partial^2 E/\partial \boldsymbol{w} \partial \boldsymbol{w}^T)$. Figure 1 illustrates an initial point on the singular region. Since most points in singular regions are saddles [2], we surely find a descending path which corresponds to a negative eigen value of $\boldsymbol{H}$. That is, each negative eigen value of $\boldsymbol{H}$ is picked up, and its eigen vector $\boldsymbol{v}$ and its negative $-\boldsymbol{v}$ are selected as two search directions. The appropriate step length is decided using line search [8]. After the first move, the search is continued using a quasi-Newton called BPQ [9].

The third point is on the number of initial search points; namely, SSF1.2 and 1.3 start search from singular regions in a very selective way, while SSF1.0 tries many initial points in a rather exhaustive way. As for region $\widehat{\boldsymbol{\Theta}}_J^{\gamma}$, SSF1.2 and 1.3 start from three initial points: a middle interpolation point, a boundary point, and an extrapolation point. These correspond to $q = 0.5$, 1.0, and 1.5 respectively in the following. Note that the restriction eq. (10) is satisfied.

$$w_1 = q\ \widehat{u}_m, \qquad w_m = (1-q)\ \widehat{u}_m \tag{11}$$

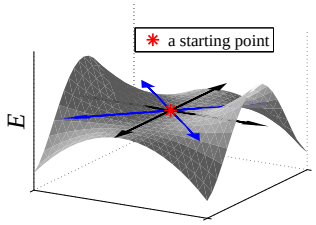


Fig. 1. Conceptual diagram of a singular region.

On the other hand, SSF1.0 exhaustively tries many initial points in the form of interpolation or extrapolation of eq.(10). As for region $\widehat{\boldsymbol{\Theta}}_J^{\alpha\beta}$, SSF1.2 and 1.3 use only one initial point in the region, while SSF1.0 does not search this region.

The final point is on *search pruning*, which is newly introduced in SSF1.3; neither SSF1.0 nor SSF1.2 has such feature. Our previous experiences on SSF indicated that although we started many searches from singular regions, we obtained only limited kinds of solutions. This means many searches join together through their search processes. Thus, we consider the search pruning described below will surely accelerate the whole search.

Now our search pruning is described in detail. Let $\boldsymbol{\theta}^{(t)}$ and $\boldsymbol{\phi}^{(i)}$ be a current search node and a previous search node respectively. Here a search node means a search point to be checked or recorded at certain intervals, say at every 100 moves. We introduce the following normalization of weights in order to make the pruning effective. The normalization will prevent large weight values from overly affecting the similarity judging. Let $\boldsymbol{d}$ be a normalization vector.

$$d_m \leftarrow \begin{cases} 1/|\theta_m^{(t-1)}| & (1 < |\theta_m^{(t-1)}|) \\ 1 & (|\theta_m^{(t-1)}| \le 1) \end{cases} \tag{12}$$

$$\boldsymbol{v}^{(t)} \leftarrow \mathrm{diag}(\boldsymbol{d})\ \boldsymbol{\theta}^{(t)} \tag{13}$$

$$\boldsymbol{v}^{(t-1)} \leftarrow \mathrm{diag}(\boldsymbol{d})\ \boldsymbol{\theta}^{(t-1)} \tag{14}$$

$$\boldsymbol{r}^{(i)} \leftarrow \mathrm{diag}(\boldsymbol{d})\ \boldsymbol{\phi}^{(i)},\ i = 1, \cdots, I \tag{15}$$

Here $m = 1, \cdots, M$, and M is the number of weights. I is the number of previous search nodes stored in memory, and diag($\boldsymbol{d}$) denotes a diagonal matrix whose diagonal elements are $\boldsymbol{d}$ and the other elements are zero.

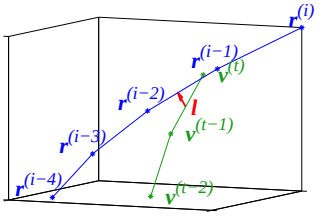


Fig. 2. Conceptual diagram of search pruning.

Figure 2 illustrates a conceptual diagram of search pruning. We consider previous line segment vectors from $\boldsymbol{r}^{(i-1)}$ to $\boldsymbol{r}^{(i)}$ for $i = 1, \cdots, I$ and the current line segment vector from $\boldsymbol{v}^{(t-1)}$ to $\boldsymbol{v}^{(t)}$. If the current line segment is

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

close enough to any previous line segment, then the current search is pruned at once. We define two difference vectors $\boldsymbol{\Delta r}^{(i)} \equiv \boldsymbol{r}^{(i)} - \boldsymbol{r}^{(i-1)}$, and $\boldsymbol{\Delta v}^{(t)} \equiv \boldsymbol{v}^{(t)} - \boldsymbol{v}^{(t-1)}$. Then consider a segment vector perpendicular to both lines which include the above difference vectors. The segment vector is described as below.

$$\boldsymbol{\ell} = (\boldsymbol{r}^{(i-1)} + a_1 \triangle \boldsymbol{r}^{(i)}) - (\boldsymbol{v}^{(t-1)} + a_2 \triangle \boldsymbol{v}^{(t)}) \qquad (16)$$

Unknown variables $\boldsymbol{a} \equiv (a_1, a_2)^T$ can be determined by solving the following minimization problem.

$$\min_{\boldsymbol{a}} \ \boldsymbol{\ell}^T \boldsymbol{\ell} \qquad (17)$$

The solution $\boldsymbol{a}$ can be written as below. Here $b_1 \equiv \|\triangle \boldsymbol{r}^{(i)}\|^2$, $b_2 \equiv \triangle \boldsymbol{r}^{(i)T} \triangle \boldsymbol{v}^{(t)}$, $b_3 \equiv \|\triangle \boldsymbol{v}^{(t)}\|^2$, $b_4 \equiv (\boldsymbol{r}^{(i-1)} - \boldsymbol{v}^{(t-1)})^T \triangle \boldsymbol{r}^{(i)}$, $b_5 \equiv (\boldsymbol{r}^{(i-1)} - \boldsymbol{v}^{(t-1)})^T \triangle \boldsymbol{v}^{(t)}$.

$$\boldsymbol{a} \ = \ - \frac{1}{b_1 b_3 - b_2^2} \left[\begin{array}{c} b_3 b_4 - b_2 b_5 \\ b_2 b_4 - b_1 b_5 \end{array} \right] \qquad (18)$$

Both endpoints of the line segment $\boldsymbol{\ell}$ are on two difference vectors $\boldsymbol{\Delta r}^{(i)}$ and $\boldsymbol{\Delta v}^{(t)}$ if and only if the following equations hold.

$$0 \le a_1 \le 1, \qquad 0 \le a_2 \le 1 \qquad (19)$$

If both of the above hold and each element ℓ_m of $\boldsymbol{\ell}$ satisfies $\ell_m < \epsilon$, then we consider the current search joins some previous search, and prune it. Otherwise, we consider the current search is different from any previous searches. In our experiments we set $\epsilon = 0.3$.

The procedure of SSF1.3 is described below, which is much the same as that of SSF1.2. The only difference is presence or absence of search pruning. SSF1.3 searches MLP parameter spaces by ascending singularity stairs one by one, beginning with J=1 and gradually increasing J until J_{max}. The optimal MLP(J=1) can be found just applying reducibility mapping $\alpha\beta$ to the optimal MLP(J=0); MLP(J=0) is a constant model. Step 1 embodies such search. Step 2-1 and step 2-2 search MLP(J+1) parameter space starting from singular regions $\widehat{\boldsymbol{\Theta}}^{\alpha\beta}_{J+1}$, and $\widehat{\boldsymbol{\Theta}}^{\gamma}_{J+1}$ respectively. Here $w_0^{(J)}$, $w_j^{(J)}$, and $\boldsymbol{w}_j^{(J)}$ denote weights of MLP(J).

SSF1.3 (Singularity Stairs Following, ver. 1.3):
(step 1) Initialize weights of MLP(J=1) using reducibility mapping $\alpha\beta$:
$w_0^{(1)} \leftarrow \widehat{w}_0^{(0)} = \overline{y}, \quad w_1^{(1)} \leftarrow 0, \quad \boldsymbol{w}_1^{(1)} \leftarrow [0, 0, \cdots, 0]^T$.
Pick up each negative eigen value of the Hessian $\boldsymbol{H}$ and select its eigen vector $\boldsymbol{v}$ and $-\boldsymbol{v}$ as two search directions. Find the appropriate step length using line search. Then perform MLP(J=1) learning with search pruning and keep the best as $\widehat{w}_0^{(1)}$, $\widehat{w}_1^{(1)}$, and $\widehat{\boldsymbol{w}}_1^{(1)}$. $J \leftarrow 2$.
(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):
$w_j^{(J+1)} \leftarrow \widehat{w}_j^{(J)}, \ j = 0, 1, \cdots, J,$
$\boldsymbol{w}_j^{(J+1)} \leftarrow \widehat{\boldsymbol{w}}_j^{(J)}, \ j = 1, \cdots, J$
$w_{J+1}^{(J+1)} = 0, \quad \boldsymbol{w}_{J+1}^{(J+1)} \leftarrow [0, 0, \cdots, 0]^T$.
Find the reasonable search directions and their appropriate step lengths by using the procedure shown in step 1. Then perform MLP(J+1) learning with search pruning 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:
$w_j^{(J+1)} \leftarrow \widehat{w}_j^{(J)}, \ j \in \{0, 1, \cdots, J\} \setminus \{m\},$
$\boldsymbol{w}_j^{(J+1)} \leftarrow \widehat{\boldsymbol{w}}_j^{(J)}, \ j = 1, \cdots, J$
$\boldsymbol{w}_{J+1}^{(J+1)} \leftarrow \widehat{\boldsymbol{w}}_m^{(J)}$.
Initialize $w_m^{(J+1)}$ and $w_{J+1}^{(J+1)}$ three times as shown below with q = 0.5, 1.0, and 1.5.
$w_m^{(J+1)} = q \ \widehat{w}_m^{(J)}, \quad w_{J+1}^{(J+1)} = (1-q) \ \widehat{w}_m^{(J)}$
For each of the above three, find the reasonable search directions and their appropriate step lengths by using the procedure shown in step 1. Then perform MLP(J+1) learning with search pruning 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, quasi-Newtons, SSF1.3 will find excellent solutions with much higher probabilities.
(2) Excellent solutions will be obtained one after another for $J = 1, \cdots, J_{max}$. SSF1.3 guarantees the monotonic improvement of training solution quality, which is not guaranteed for most existing methods.
(3) SSF1.3 will be faster than SSF1.0 or SSF1.2 due to the search pruning. SSF1.3 will also be faster than most existing methods if they are performed many times changing initial weights.

IV. Experiments

We evaluated the proposed SSF1.3 for sigmoidal and polynomial MLPs using artificial and real data sets. We used a PC with Intel Core i7-2600 (3.4GHz). Forward calculations of sigmoidal and polynomial MLPs are shown in eq. (20) and eq. (21) respectively.

$$f \ = \ w_0 + \sum_{j=1}^{J} w_j z_j, \quad z_j = \sigma(\boldsymbol{w}_j^{\mathrm{T}} \boldsymbol{x}) \qquad (20)$$

$$f \ = \ w_0 + \sum_{j=1}^{J} w_j z_j,$$
$$z_j = \prod_{k=1}^{K} (x_k)^{w_{jk}} = \exp(\sum_{k=1}^{K} w_{jk} \ln x_k) \qquad (21)$$

For comparison, we employed batch BP and a quasi-Newton called BPQ [9] as existing methods. The learning rate of batch BP was adaptively determined using line search. BP or BPQ was performed 100 times for each J changing initial weights; i.e., w_{jk} and w_j were randomly selected from the range $[-1, +1]$ except $w_0 = \overline{y}$. SSF or BPQ stops when a step length is less than 10^{-16} or the iteration exceeds 10,000. For artificial data, generalization was evaluated using 1,000

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

test data points independent of training data. For real data 10-fold cross-validation was used, and each x_k and y were normalized as $x_k/max(x_k)$ and $(y-\overline{y})/std(y)$ respectively.

Experiment of Sigmoidal MLP using Artificial Data 1

An artificial data set for sigmoidal MLP was generated using MLP having the following weights. Values of each input variable $x_1, x_2, \cdots, x_{10}$ 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 five variables $x_6, \cdots, x_{10}$ are irrelevant. The size of training data was $N = 1,000$, and J_{max} was set to be 8.

$$(w_0, w_1, w_2, w_3, w_4, w_5, w_6) = (-6, -6, -5, -3, 2, 10, 9), \quad (22)$$

$$(\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3, \boldsymbol{w}_4, \boldsymbol{w}_5, \boldsymbol{w}_6) = \begin{pmatrix} -1 & -1 & 1 & -3 & 3 & -6 \\ 0 & 7 & -9 & 9 & 1 & 4 \\ 6 & -5 & 8 & -8 & -8 & 3 \\ -9 & 5 & 6 & 3 & -4 & 5 \\ -9 & -1 & 7 & -10 & -6 & -1 \\ 9 & 7 & 3 & -7 & -2 & 6 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (23)$$

Table I shows the numbers of sigmoidal MLP search routes of SSF1.2 and 1.3 for artificial data 1. For SSF1.3, initial and final routes correspond to search routes before and after search pruning respectively. For SSF1.3, the total numbers of initial and final routes were 908 and 392 respectively, meaning 56.8 % of initial routes were pruned on their ways.

TABLE I
NUMBERS OF SIGMOIDAL MLP SEARCH ROUTES FOR ARTIFICIAL DATA 1

J	SSF1.2	SSF1.3	
		initial routes	final routes
1	5	5	5
2	29	29	21
3	65	65	48
4	83	83	44
5	122	122	49
6	161	165	76
7	195	200	80
8	245	239	69
total	905	908	392

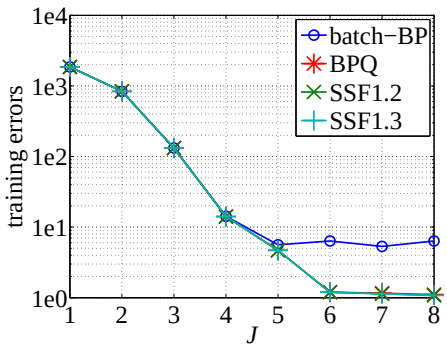
Table II shows CPU time required by each method for sigmoidal MLP using artificial data 1. SSF1.3 was 2.61 times faster than SSF1.2 owing to search pruning. SSF1.3 finished search the fastest among the four.

Figure 3 shows how training and test errors changed along with the increase of J. BP stopped decreasing at $J = 5$, while SSF and BPQ monotonically decreased training error. As for test error, BP selected $J = 7$ as the best model, while SSF and BPQ indicated $J = 6$ is the best, which is correct.

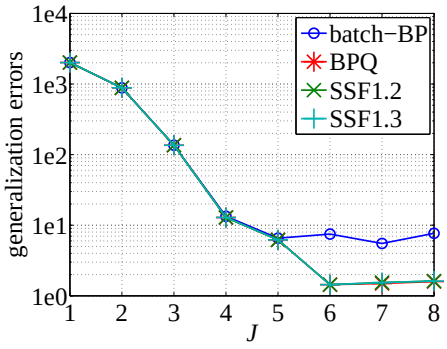
Figure 4 compares histograms of BPQ and SSF1.3 solutions for MLP(J=6). BPQ found the excellent solution only once out of 100 runs, while SSF1.3 found it eight times out of 97 search routes. Moreover, BPQ solutions are widely scattered, while SSF1.3 solutions are densely located around the excellent solution. The tendencies that SSF1.3 finds excellent solutions with much higher probability and SSF1.3 solutions are more densely located close to an excellent solution were observed in other experiments, which are omitted due to space limitation.

TABLE II
CPU TIME FOR SIGMOIDAL MLP USING ARTIFICIAL DATA 1 (HR:MIN:SEC)

J	batch BP	BPQ	SSF1.2	SSF1.3
1	00:04:39	00:00:04	00:00:00	00:00:01
2	00:06:35	00:00:11	00:00:09	00:00:08
3	00:08:38	00:01:54	00:02:13	00:00:39
4	00:11:19	00:06:40	00:05:43	00:01:08
5	00:13:34	00:10:01	00:08:14	00:02:39
6	00:15:08	00:14:18	00:14:29	00:08:43
7	00:17:13	00:18:02	00:34:07	00:13:21
8	00:18:59	00:21:26	00:58:34	00:20:38
total	01:36:05	01:12:36	02:03:29	00:47:16

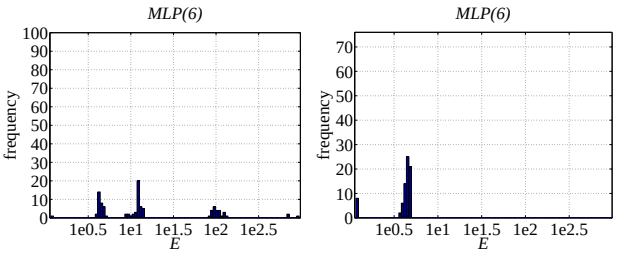


(a) training error



(b) test error

Fig. 3. Training and test errors of sigmoidal MLP for artificial data 1



(a) BPQ (b) SSF1.3

Fig. 4. Histograms of sigmoidal MLP(J=6) solutions for artificial data 1.

Experiment of Polynomial MLP using Artificial Data 2

Here we consider the following multivariate polynomial. Values of $x_1, \cdots, x_{15}$ were randomly selected from $[0, +1]$,

while values of y were generated by adding small Gaussian noise $\mathcal{N}(0, 0.05^2)$ to MLP outputs. Here five variables $x_{11}, \cdots, x_{15}$ are irrelevant. The size of training data was $N = 1,000$. J_{max} was set to be 8.

$$y = 3 - 50x_1^3x_2^4 - 28x_2^9x_3^6 - 23x_4^{10}x_5^{10}x_6^5 - 18x_7^4x_8^4 - 12x_8^6x_9^{10} + 36x_{10} \quad (24)$$

Table III shows the numbers of sigmoidal MLP search routes of SSF1.2 and 1.3 for artificial data 2. For SSF1.3, the total numbers of initial and final routes were 1432 and 610 respectively, meaning 57.4 % of initial routes were pruned on their ways.

TABLE III
NUMBERS OF POLYNOMIAL MLP SEARCH ROUTES FOR ARTIFICIAL DATA 2

J	SSF1.2	SSF1.3	
		initial routes	final routes
1	14	14	13
2	47	47	10
3	93	103	41
4	159	146	59
5	210	194	81
6	261	265	97
7	324	274	108
8	386	389	201
total	1494	1432	610

Table IV shows CPU time required by each method for sigmoidal MLP using artificial data 2. SSF1.3 was 2.37 times faster than SSF1.2 owing to search pruning. SSF1.3 finished search the fastest among the four.

TABLE IV
CPU TIME FOR POLYNOMIAL MLP USING ARTIFICIAL DATA 2 (HR:MIN:SEC)

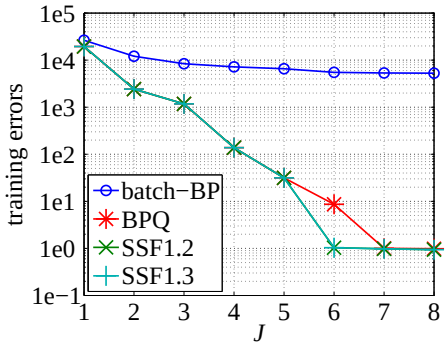
J	batch BP	BPQ	SSF1.2	SSF1.3
1	00:03:34	00:02:46	00:00:05	00:00:03
2	00:04:18	00:02:18	00:01:09	00:00:10
3	00:05:20	00:03:01	00:00:59	00:00:40
4	00:06:24	00:03:23	00:03:18	00:01:59
5	00:07:24	00:05:17	00:08:27	00:04:02
6	00:07:28	00:05:38	00:13:13	00:03:55
7	00:08:16	00:06:48	00:21:28	00:08:10
8	00:08:43	00:07:03	00:20:51	00:10:21
total	00:51:27	00:36:15	01:09:32	00:29:22

Figure 5 shows how training and test errors changed when J was increased. BP could not decrease training error efficiently, while SSF and BPQ monotonically decreased at a steady pace. Specifically, SSF outperformed BPQ at J=6. As for test error, BP stayed at the same level, and BPQ reached the bottom at $J = 5$ and 6, and rose sharply for $J \geq 7$, while SSF1.3 showed a steady move and indicated $J = 6$ is the best model, which is correct.

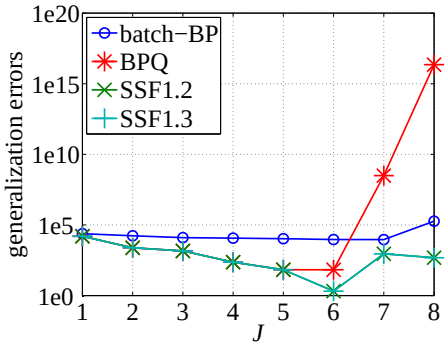
Experiment of Sigmoidal MLP using Real Data

As real data for sigmoidal MLP we used concrete compressive strength data from UCI ML repository. The number of input variables is 8, and the data size is $N = 1,030$. From our preliminary experiment, J_{max} was set to be 18.

Table V shows the numbers of sigmoidal MLP search routes of SSF1.2 and 1.3 for concrete data. For SSF1.3, the total numbers of initial and final routes were 5523 and 981 respectively, meaning 82.2 % of initial routes were pruned on their ways.



(a) training error



(b) test error

Fig. 5. Training and test errors of polynomial MLP for artificial data 2

TABLE V
NUMBERS OF SIGMOIDAL MLP SEARCH ROUTES USING CONCRETE DATA

J	SSF1.2	SSF1.3	
		initial routes	final routes
1	6	6	6
2	30	30	9
3	62	51	14
4	101	87	24
5	107	103	36
6	145	139	44
7	226	176	40
8	209	223	49
9	211	275	75
10	329	422	75
11	300	329	64
12	377	411	84
13	332	484	88
14	340	547	89
15	665	429	68
16	467	692	68
17	612	500	75
18	476	619	73
total	4995	5523	981

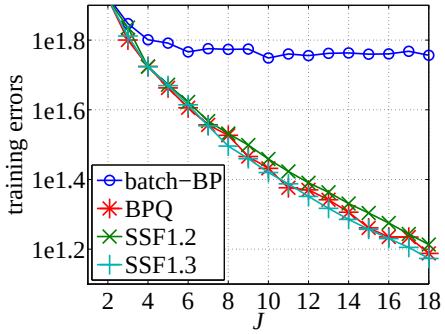
Table VI shows CPU time required by each method for sigmoidal MLP using concrete data. SSF1.3 was 4.06 times faster than SSF1.2 owing to search pruning. Again, SSF1.3 finished search the fastest among the four.

Figure 6 shows how training and test errors changed along with the increase of J. BP could not decrease training error efficiently for $J \geq 6$, while SSF and BPQ monotonically decreased training error at a steady pace. As for test error, BP stayed at a poor level, and BPQ fluctuated very widely for $J > 16$, while SSF showed a steady move and SSF1.3 indicated $J = 16$ as the best model, but SSF1.2 showed a monotonic tendency in this range of J.

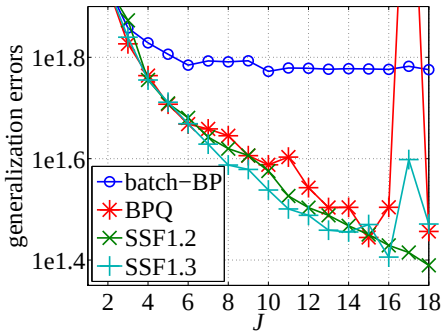
ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE VI
CPU TIME FOR SIGMOIDAL MLP USING CONCRETE DATA (HR:MIN:SEC)

J	batch BP	BPQ	SSF1.2	SSF1.3
1	00:05:40	00:00:05	00:00:00	00:00:01
2	00:07:14	00:07:40	00:02:01	00:00:23
3	00:09:25	00:10:22	00:06:06	00:01:34
4	00:12:33	00:13:38	00:09:00	00:03:53
5	00:14:59	00:16:27	00:16:11	00:06:22
6	00:16:48	00:18:39	00:25:17	00:08:51
7	00:19:04	00:21:28	00:45:15	00:10:10
8	00:21:07	00:23:56	00:46:36	00:15:05
9	00:23:46	00:26:47	00:52:35	00:23:12
10	00:16:18	00:20:17	01:02:19	00:19:01
11	00:18:39	00:23:16	01:03:56	00:18:13
12	00:20:49	00:26:10	01:31:29	00:27:22
13	00:23:08	00:29:18	01:30:04	00:32:26
14	00:25:13	00:32:10	01:42:01	00:37:02
15	00:27:31	00:35:11	03:40:07	00:30:21
16	00:22:53	00:31:52	02:19:24	00:28:05
17	00:24:17	00:34:22	03:17:28	00:31:36
18	00:24:16	00:34:55	02:39:15	00:34:01
total	05:33:37	06:46:31	22:09:05	05:27:35



(a) training error



(b) test error

Fig. 6. Training and test errors of sigmoidal MLP for concrete data

V. CONCLUSION

This paper proposed a new MLP learning method called SSF1.3, which makes good use of the whole singular regions and has the search pruning feature. Beginning with MLP(J=1) it gradually increases J one by one to successively and stably find excellent solutions for each J. Compared with existing methods such as BP or a quasi-Newton method, SSF1.3 successively and more stably found excellent solutions commensurate with each J. In the future we plan to apply our method to 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) 25330294 and Chubu University Grant 24IS27A.

REFERENCES

[1] H. J. Sussmann, "Uniqueness of the weights for minimal feedforward nets with a given input-output map," *Neural Networks*, vol. 5, no. 4, pp. 589–593, 1992.
[2] K. Fukumizu and S. Amari, "Local minima and plateaus in hierarchical structure of multilayer perceptrons," *Neural Networks*, vol. 13, no. 3, pp. 317–327, 2000.
[3] H. Hecht-Nielsen, *Neurocomputing*. Addison-Wesley, 1990.
[4] R. Nakano, S. Satoh, and T. Ohwaki, "Learning method utilizing singular region of multilayer perceptron," in *Proc. 3rd Int. Conf. on Neural Computation Theory and Applications*, 2011, pp. 106–111.
[5] S. Amari, "Natural gradient works efficiently in learning," *Neural Computation*, vol. 10, no. 2, pp. 251–276, 1998.
[6] W. S., *Algebraic geometry and statistical learning theory*. Cambridge University Press, 2009.
[7] S. Satoh and R. Nakano, "Fast and stable learning utilizing singular regions of multilayer perceptron," *Neural Processing Letters*, 2013, online DOI:10.1007/s11063-013-9283-z.
[8] D. G. Luenberger, *Linear and nonlinear programming*. Addison-Wesley, 1984.
[9] K. Saito and R. Nakano, "Partial BFGS update and efficient step-length calculation for three-layer neural networks," *Neural Computation*, vol. 9, no. 1, pp. 239–257, 1997.

ISBN: 978-988-19253-1-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)