Faster RBF Network Learning Utilizing Singular Regions

Seiya Satoh[1] and Ryohei Nakano[2]
[1]*Tokyo Denki University, Ishizaka, Hatoyama-machi, Hiki-gun, Saitama 350-0394 Japan*
[2]*Chubu University, 1200 Matsumoto-cho, Kasugai, 487-8501 Japan*
seiya.satoh@mail.dendai.ac.jp, nakano_ryo@isc.chubu.ac.jp

Keywords: Neural Networks, RBF Networks, Learning Method, Singular Region, Reducibility Mapping

Abstract: There are two ways to learn radial basis function (RBF) networks: one-stage and two-stage learnings. Recently a very powerful one-stage learning method called RBF-SSF has been proposed, which can stably find a series of excellent solutions, making good use of singular regions, and can monotonically decrease training error along with the increase of hidden units. RBF-SSF was built by applying the SSF (singularity stairs following) paradigm to RBF networks; the SSF paradigm was originally and successfully proposed for multilayer perceptrons. Although RBF-SSF has the strong capability to find excellent solutions, it required a lot of time mainly because it computes the Hessian. This paper proposes a faster version of RBF-SSF called RBF-SSF(pH) by introducing partial calculation of the Hessian. The experiments using two datasets showed RBF-SSF(pH) ran as fast as usual one-stage learning methods while keeping the excellent solution quality.

1 INTRODUCTION

A radial basis function (RBF) network has the capability of universal approximation and is a popular alternative to a multilayer perceptron (MLP). An RBF network has the following parameters to learn: RBF centers (composed of weights between input and hidden layers), their widths such as variances in Gaussian basis functions, and weights between hidden and output layers.

So far, many methods have been proposed to learn RBF networks (Wu et al., 2012), and most methods can be classified into two kinds: one-stage and two-stage learnings. One-stage learning optimizes all the parameters at the same time by using gradient-based methods or the EM algorithm (Dempster et al., 1977). On the other hand, Two-stage learning goes in two stages; first, it selects suitable RBF centers together with their widths, and then optimizes only the remaining weights by solving linear regressions. (Bishop, 1995). Two-stage learning runs very fast and has been very popular. Although one-stage learning requires much more processing time than two-stage learning, it finds solutions having smaller training errors than two-stage learning for the same model complexity (Satoh and Nakano, 2018). Incidentally, three-stage learning has been investigated for classification task (Schwenker et al., 2001).

Recently a very powerful one-stage learning method called RBF-SSF has been proposed (Satoh and Nakano, 2018), which can stably find a series of excellent solutions, making good use of singular regions, and can monotonically decrease training error along with the increase of hidden units. RBF-SSF is built by applying the SSF (singularity stairs following) paradigm to RBF networks; the SSF paradigm was originally proposed for MLPs with great success (Satoh and Nakano, 2013) (Satoh and Nakano, 2015). Although RBF-SSF has the strong capability to find excellent solutions, it required a lot of processing time, even longer than a usual one-stage learning method mainly because it computes the Hessian.

This paper proposes a faster version of RBF-SSF called RBF-SSF(pH) by introducing partial calculation of the Hessian matrix, and compares its performance with those of the original RBF-SSF, a typical two-stage learning method, and usual one-stage learning methods.

This paper is organized as follows. Section 2 reviews RBF network formalization, existing learning methods, and singular regions as the background knowledge. Section 3 proposes RBF-SSF(pH), explaining a general flow, partial Hessian calculation, and other techniques for making SSF faster. Section 4 describes our experiments evaluating the solution quality and processing time of RBF-SSF(pH) in comparison with the original RBF-SSF and three existing methods. Section 5 summarizes the paper.

2 BACKGROUND

2.1 RBF Networks

Let RBF(J) be an RBF network with J hidden units and one output unit. In RBF(J) model, let $\mathbf{w}_j^{(J)} = (w_{j1}^{(J)}, \cdots, w_{jK}^{(J)})^{\mathrm{T}}$ be a vector of weights between all input units and hidden unit j ($= 1, \cdots, J$), and let $v_j^{(J)}$ be a weight between hidden unit j ($= 0, 1, \cdots, J$) and a single output unit. When Gaussian basis function is adopted and μ-th data point $\mathbf{x}^\mu = (x_1^\mu, \cdots, x_K^\mu)^{\mathrm{T}}$ is given as input, the output of RBF(J) can be defined as below. Here σ_j is a width parameter of Gaussian basis function at hidden unit j.

$$f_J^\mu = v_0^{(J)} + \sum_{j=1}^{J} v_j^{(J)} \exp\left(-\frac{||\mathbf{x}^\mu - \mathbf{w}_j^{(J)}||^2}{2(\sigma_j^{(J)})^2}\right) \quad (1)$$

The whole parameter vector of RBF(J) is given as below:

$$\theta^{(J)} = \left(v_0^{(J)}, \cdots, v_J^{(J)}, \left(\mathbf{w}_1^{(J)}\right)^{\mathrm{T}}, \cdots, \left(\mathbf{w}_J^{(J)}\right)^{\mathrm{T}}, \sigma_1^{(J)}, \cdots, \sigma_J^{(J)}\right)^{\mathrm{T}}. \quad (2)$$

Given training data $\{(\mathbf{x}^\mu, y^\mu),\ \mu = 1, \cdots, N\}$, the target function of RBF(J) learning is given as follows.

$$E_J = \frac{1}{2}\sum_{\mu=1}^{N} \left(\delta_J^\mu\right)^2, \qquad \delta_J^\mu \equiv f_J^\mu - y^\mu \quad (3)$$

2.2 Existing RBF Network Learning Methods

Many methods to learn RBF networks can be classified into two kinds: one-stage learning and two-stage learning.

In one-stage learning, whole weights are optimized at the same time by using gradient-based methods or the EM algorithm (Dempster et al., 1977). Gradient-based methods can be classified into 1st-order methods such as the steepest descent and 2nd-order methods such as quasi-Newton method.

In two-stage learning, under the condition that $\sigma_1^{(J)}, \cdots, \sigma_J^{(J)}$ are fixed to a certain constant throughout learning, first, select suitable $\mathbf{w}_1^{(J)}, \cdots, \mathbf{w}_J^{(J)}$, and then, optimize $v_0^{(J)}, \cdots, v_J^{(J)}$ by solving linear regressions (Bishop, 1995).

One-Stage Learning
One-stage learning can be classified into gradient-based or mixture-based (Lázaro et al., 2003). In the following experiments, we employ steepest descent (SD) as a 1st-order method and BFGS (quasi-Newton method with the BFGS update) (Fletcher, 1987) as a 2nd-order method.

Two-Stage Learning
At the first stage of this learning, one can apply clustering to explanatory data $\{\mathbf{x}^\mu\}$ to get J clusters, and then treat all the centroids as $\mathbf{w}_1^{(J)}, \cdots, \mathbf{w}_J^{(J)}$.

This paper, however, employs function newrb available in Neural Network Toolbox of MATLAB R2015b. The newrb algorithm gradually expands an RBF network by increasing the number J of hidden units one by one. At each cycle of model expansion, data point $\mathbf{x}^\mu$ that has the largest value of output error $|f_{J-1}^\mu - y^\mu|$ is added as $\mathbf{w}_J$. Width parameters $\{\sigma_j\}$ of basis functions are not optimized in this learning.

The general flow of the newrb algorithm is shown below. Let $J_{\max}$ be the maximum number of hidden units.

newrb algorithm

1: $v_0 \leftarrow (1/N)\sum_{\mu=1}^{N} y^\mu$
2: **for** $J = 1, \cdots, J_{\max}$ **do**
3: Compute outputs f_{J-1}^μ, $\mu = 1, \cdots, N$.
4: $i \leftarrow \mathrm{argmax}_{\mu \in \{1,\cdots,N\}} |f_{J-1}^\mu - y^\mu|$
5: $\mathbf{w}_J^{(J)} \leftarrow \mathbf{x}^i$
6: With $\mathbf{w}_1^{(J)}, \cdots, \mathbf{w}_J^{(J)}$ fixed, optimize $v_0^{(J)}, \cdots, v_J^{(J)}$.
7: **end for**

2.3 Singular Regions of RBF Networks

The search space of MLP has a continuous region where input-output equivalence holds and the gradient is zero (Fukumizu and Amari, 2000) (Nitta, 2013). Such regions can be generated by using reducibility mapping $\alpha\beta$ or γ (Satoh and Nakano, 2013) (Satoh and Nakano, 2015). We call such a region a singular region.

The search space of an RBF network has also singular regions, which are generated only by reducibility mapping γ.

Below we explain how to generate a singular region of RBF(J) based on the optimal solution of RBF($J-1$). The optimal solution must satisfy the following, where E_{J-1} denotes the target function of RBF($J-1$) learning, and $\theta^{(J-1)}$ denotes the whole parameter vector of RBF($J-1$).

$$\frac{\partial E_{J-1}}{\partial \theta^{(J-1)}} = \mathbf{0} \quad (4)$$

This can be broken down element-wise as follows,

where $j = 1, \cdots, J-1$.

$$\frac{\partial E_{J-1}}{\partial v_0^{(J-1)}} = \sum_{\mu=1}^{N} \delta_{J-1}^{\mu} = 0 \quad (5)$$

$$\frac{\partial E_{J-1}}{\partial v_j^{(J-1)}} = \sum_{\mu} \delta_{J-1}^{\mu} \exp\left(-\frac{||\mathbf{x}^{\mu} - \mathbf{w}_j^{(J-1)}||^2}{2(\sigma_j^{(J-1)})^2}\right) = 0 \quad (6)$$

$$\frac{\partial E_{J-1}}{\partial \mathbf{w}_j^{(J-1)}} = \sum_{\mu} \delta_{J-1}^{\mu} v_j^{(J-1)} \exp\left(-\frac{||\mathbf{x}^{\mu} - \mathbf{w}_j^{(J-1)}||^2}{2(\sigma_j^{(J-1)})^2}\right) \frac{\mathbf{x}^{\mu} - \mathbf{w}_j^{(J-1)}}{\left(\sigma_j^{(J-1)}\right)^2} = \mathbf{0} \quad (7)$$

$$\frac{\partial E_{J-1}}{\partial \sigma_j^{(J-1)}} = \sum_{\mu} \delta_{J-1}^{\mu} v_j^{(J-1)} \exp\left(-\frac{||\mathbf{x}^{\mu} - \mathbf{w}_j^{(J-1)}||^2}{2(\sigma_j^{(J-1)})^2}\right) \frac{||\mathbf{x}^{\mu} - \mathbf{w}_j^{(J-1)}||^2}{\left(\sigma_j^{(J-1)}\right)^3} = 0 \quad (8)$$

Let the optimal solution $\widehat{\theta}^{(J-1)}$ of RBF($J-1$) be $\left(\widehat{v}_0^{(J-1)}, \cdots, \widehat{v}_J^{(J-1)}, \left(\widehat{\mathbf{w}}_1^{(J-1)}\right)^{\mathrm{T}}, \cdots, \left(\widehat{\mathbf{w}}_J^{(J-1)}\right)^{\mathrm{T}}, \widehat{\sigma}_1^{(J-1)}, \cdots, \widehat{\sigma}_J^{(J-1)}\right)^{\mathrm{T}}$.
Now apply reducibility mapping γ to the optimal solution $\widehat{\theta}^{(J-1)}$ to get singular region $\widehat{\Theta}_{\gamma}^{(J)}$. Here $m \in \{1, \cdots, J-1\}$.

$$\widehat{\theta}^{(J-1)} \xrightarrow{\gamma} \widehat{\Theta}_{\gamma}^{(J)}$$

$$\begin{aligned}\widehat{\Theta}_{\gamma}^{(J)} \equiv \{\theta^{(J)} |\ & v_0^{(J)} = \widehat{v}_0^{(J-1)}, v_m^{(J)} + v_J^{(J)} = \widehat{v}_m^{(J-1)}, \\ & \mathbf{w}_m^{(J)} = \mathbf{w}_J^{(J)} = \widehat{\mathbf{w}}_m^{(J-1)}, \sigma_m^{(J)} = \sigma_J^{(J)} = \widehat{\sigma}_m^{(J-1)}, \\ & v_j^{(J)} = \widehat{v}_j^{(J-1)}, \mathbf{w}_j^{(J)} = \widehat{\mathbf{w}}_j^{(J-1)}, \sigma_j^{(J)} = \widehat{\sigma}_j^{(J-1)}, \\ & \text{for } j \in \{1, \cdots, J-1\} \backslash \{m\}\} \end{aligned} \quad (9)$$

Note that $v_m^{(J)}$ and $v_J^{(J)}$ cannot be determined uniquely since they only have the following constraint.

$$v_m^{(J)} + v_J^{(J)} = \widehat{v}_m^{(J-1)} \quad (10)$$

This equation can be rewritten using the following q.

$$v_m^{(J)} = q\, \widehat{v}_m^{(J-1)}, \quad v_J^{(J)} = (1-q)\, \widehat{v}_m^{(J-1)} \quad (11)$$

3 RBF-SSF(pH)

A search paradigm called Singularity Stairs Following (SSF) stably finds a series of excellent solutions by making good use of singular regions.

For MLPs two series of SSF methods have been proposed: the latest versions are SSF1.4 (Satoh and Nakano, 2013) for real-valued MLPs and C-SSF1.3 (Satoh and Nakano, 2015) for complex-valued MLPs.

By applying the SSF paradigm to RBF networks, RBF-SSF has been proposed (Satoh and Nakano, 2018). Although RBF-SSF stably finds excellent solutions having much simpler model complexities and much smaller training errors, it takes a lot of processing time. Its processing time is longer than a usual one-stage learning method such as steepest descent or BFGS (quasi-Newton with the BFGS update).

Since RBF-SSF employs BFGS as its basic search engine, the difference comes from whether to calculate the Hessian or not. Any SSF calculates the Hessian at its starting from a singular region to find search directions because the gradient is zero in a singular region. Note that the eigenvector corresponding to a negative eigenvalue of the Hessian indicates a descending direction from the singular region.

Thus, one way to make RBF-SSF much faster is to drastically reduce the calculation time of the Hessian. For this reason we introduce calculating the partial Hessian instead of the full Hessian. The full Hessian is a square matrix of size M_f, the number of all the parameters; $M_f = J(K+2)+1$. On the other hand, the partial Hessian is a square matrix of size $M_p = 2(K+2)+1$. When J gets large, calculation time of the full Hessian increases in the order of J^2, while that of the partial Hessian remains a small constant. Note that RBF-SSF optimizes $\{\sigma_j\}$ as well.

3.1 General Flow of RBF-SSF(pH)

The general flow of RBF-SSF(pH) is shown below. Let $J_{\max}$ be the maximum number of hidden units.

General flow of RBF-SSF(pH)

1: Get the best solution of RBF($J=1$) by repeating search with different initial weights.
2: **for** $J = 2, \cdots, J_{\max}$ **do**
3: Select starting points from the singular region obtained by applying reducibility mapping γ to the best solution of RBF($J-1$).
4: Calculate partial Hessian at each starting point, calculate eigenvalues and eigenvectors of the partial Hessian, and select eigenvectors corresponding to each negative eigenvalue.
5: Repeat search predefined times from the starting points in the direction and in the opposite direction of an eigenvector selected at Step 4, and get the best solution of RBF(J).
6: **end for**

In the following experiments, the starting points from

a singular region at the above Step 3 are obtained by setting q to 0.5, 1.0, and 1.5 in eq.(11). These three points correspond to interpolation, boundary, and extrapolation. Since m is changed in the range of $1, \cdots, J-1$, the total number of starting points in the search of RBF(J) amounts to $3(J-1)$.

3.2 Calculation of the Partial Hessian

The Hessian matrix is a symmetric matrix of second derivatives of the target function. Although the first derivatives are easy to calculate, second derivatives require complicated and heavy calculation.

The partial Hessian is shown below. Although the full Hessian is composed of 2nd derivatives with respect to all the parameters $\theta^{(J)}$ shown in eq.(2), the partial Hessian is limited to m and J.

$$\frac{\partial^2 E_J}{\partial \theta_{m,J}^{(J)} \, \partial \theta_{m,J}^{(J)\,\mathrm{T}}} \tag{12}$$

Here

$$\theta_{m,J}^{(J)} \equiv \left(v_0^{(J)}, v_m^{(J)}, v_J^{(J)}, \left(\mathbf{w}_m^{(J)}\right)^{\mathrm{T}}, \left(\mathbf{w}_J^{(J)}\right)^{\mathrm{T}}, \sigma_m^{(J)}, \sigma_J^{(J)} \right)^{\mathrm{T}}. \tag{13}$$

The number of elements of the full Hessian is $M_f^2 = (J(K+2)+1)^2$, while that of the partial Hessian is $M_p^2 = (2(K+2)+1)^2$. Since the Hessian is symmetric, the actual number of elements to calculate is $M_f(M_f+1)/2$ for the full Hessian, and $M_p(M_p+1)/2$ for the partial Hessian.

3.3 Other Techniques for Making SSF-SSFs Faster

The section explains the techniques for making SSF for MLPS faster (Satoh and Nakano, 2015), which can be applied to RBF-SSF and RBF-SSF(pH).

The number of starting points of RBF(J) gets large as J gets large. Moreover, the dimension of the search space also gets large as J gets large. Thus, if we perform search for every negative eigenvalue for large J, the number of searches gets very large, and consequently the processing time becomes huge. Hence, we introduce two accelerating techniques into RBF-SSF: one is search pruning and the other is to set upper limit to the number of searches.

Search pruning is an accelerating technique which discards a search if the search is found to proceed along much the same route experienced before (Satoh and Nakano, 2013).

Setting upper limit to the number of searches is an accelerating technique which selects the predefined number of starting points based on the preference. Here preference is given to smaller negative eigenvalues since such eigenvalues indicate bigger drop from a search point. In the following experiments the upper limit is set to be 10.

4 EXPERIMENTS

4.1 Design of Experiments

The performance of RBF-SSF(pH) was compared with four other learning methods. Tabel 1 describes these five methods. SD and BFGS are repeated 10 times changing initial weights. The number J of hidden units for newrb was changed as $10, 20, \cdots, 500$, while J of SD, BFGS, RBF-SSF and RBF-SSF(pH) was changed as $1, 2, \cdots, 50$. Note that BFGS and two RBF-SSFs adapt $\{\sigma_j\}$ through learning.

Table 1: Learning methods.

Learning method	Description ($j = 1, \cdots, J$)
newrb	two-stage learning $\sigma_j^{(J)}$: fixed to be $1/\sqrt{2}$
SD	one-stage learning batch steepest descent with adaptive step length the initial value of $v_j^{(J)}$: 0 the initial value of $\mathbf{w}_j^{(J)}$: randomly selected from $\{\mathbf{x}^\mu\}$ $\sigma_j^{(J)}$: fixed to be $1/\sqrt{2}$
BFGS	one-stage learning quasi-Newton with BFGS update the initial value of $v_j^{(J)}$: 0 the initial value of $\mathbf{w}_j^{(J)}$: randomly selected from $\{\mathbf{x}^\mu\}$ the initial value of $\sigma_j^{(J)}$: $1/\sqrt{2}$
RBF-SSF	one-stage learning by RBF-SSF
RBF-SSF(pH)	faster version of RBF-SSF

We used the following two datasets: Schwefel function dataset (Schwefel, 1981) and Parkinsons telemonitoring dataset from UCI ML Repository (Dua and Taniskidou, 2017). Each dataset was normalized as follows, where y_{mean} and y_{std} are the average and standard deviation of $\{y^\mu\}$ respectively.

$$\widetilde{x}_k^\mu \leftarrow \frac{x_k^\mu}{\max_\mu(|x_k^\mu|)}, \qquad \widetilde{y}^\mu \leftarrow \frac{y^\mu - y_{\mathrm{mean}}}{y_{\mathrm{std}}} \tag{14}$$

4.2 Experiments Using Schwefel Function Dataset

Schwefel function is given by eq.(15) (Schwefel, 1981). We set the function parameter K and the ranges of variables x_k as $K = 2$ and $x_k \in (-50, 50)$, and then generated 2000 data points for training and other 2000 data points for test. Generated data were normalized as shown above. After normalizing y^μ, Gaussian noise $\mathcal{N}(0, 0.01^2)$ was added for training data.

$$f(x_1, ..., x_K) = 418.982887K - \sum_{k=1}^{K} x_k \sin(\sqrt{|x_k|}) \quad (15)$$

Figure 1 shows the generated function. We can see this function has many peaks.

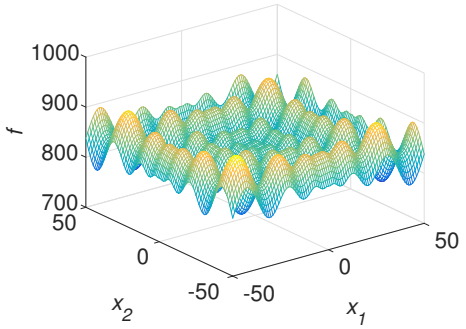

Figure 1: Schwefel function ($K = 2, x_1, x_2 \in (-50, 50)$)

Figure 2 illustrates how five learning methods, newrb, SD, BFGS, RBF-SSF, and RBF-SSF(pH), decreased training error with the increase of hidden units for Schwefel function dataset. RBF-SSF(pH) and RBF-SSF monotonically and sharply decreased training error to the lowest, showing much the same performance. BFGS and newrb could reach only more than two times larger error than the lowest; note that newrb used 10 times larger model complexity than the others. Two RBF-SSFs and BFGS massively outperformed newrb for any same model complexity. SD hardly decreased training error.

Figure 3 depicts test errors of five methods for Schwefel function dataset. Two RBF-SSFs showed much the same best performance. BFGS and newrb decreased only to the level more than two times larger test error than the best. Two RBF-SSFs and BFGS outperformed newrb for each same model complexity. SD hardly decreased test error.

Figure 4 shows total processing time of five methods for Schwefel function dataset. Newrb was the fastest, while the rest four required more processing time. Among the four, only the original RBF-SSF required more processing time than the other three.

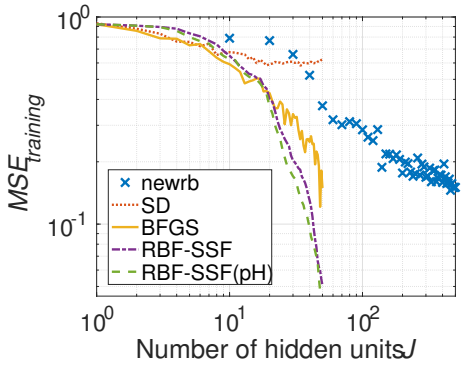

Figure 2: Training error for Schwefel function dataset

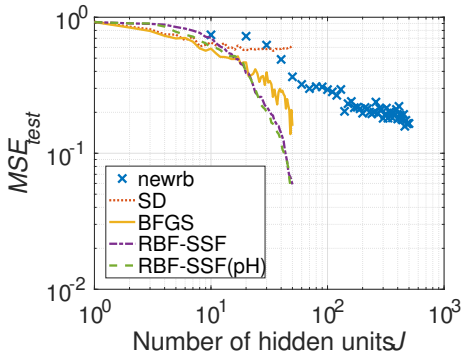

Figure 3: Test error for Schwefel function dataset

Figure 5 compares Hessian calculation time of two RBF-SSFs. The figure clearly shows how Hessian computation time was drastically reduced to a small amount of time by the proposed RBF-SSF(pH) compared with the original RBF-SSF.

Figure 6 shows the remaining processing time other than Hessian calculation of two RBF-SSFs. Two lines are closely overlapped, showing the rest were much the same.

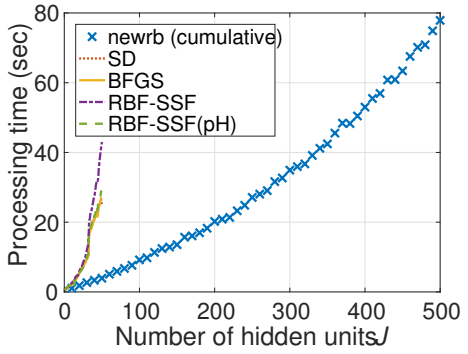

Figure 4: Total processing time for Schwefel function dataset

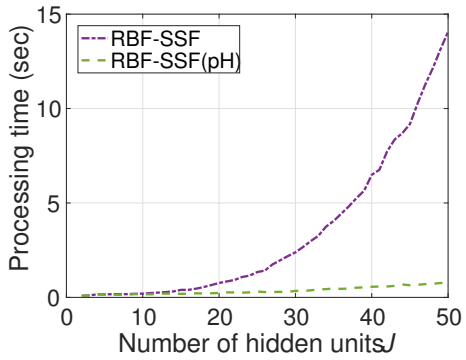

Figure 5: Processing time of Hessian calculation for Schwefel function dataset

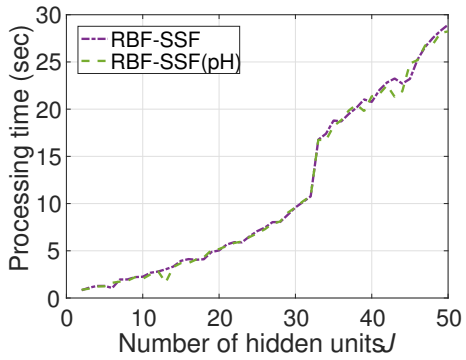

Figure 6: Processing time of other calculations for Schwefel function dataset

4.3 Experiments Using Parkinsons Telemonitoring Dataset

The problem for this dataset (Little et al., 2007) is to estimate motor UPDRS (unified Parkinson's disease rating score) based on 18 explanatory variables such as voice measures. The number N of data points is 5875; 90 % was used for training, and the remaining 10 % was used for test.

Figure 7 illustrates how five learning methods decreased training error for Parkinsons telemonitoring dataset. Two RBF-SSFs monotonically and rapidly decreased training error to the lowest, showing much the same performance. Newrb and BFGS could reach about 1.4 and 1.7 times larger error than the lowest respectively. Note that again newrb used 10 times larger model complexity than the others. Two RBF-SSFs and BFGS significantly outperformed newrb for any same model complexity. SD hardly decreased training error.

Figure 8 depicts test errors of five methods for Parkinsons telemonitoring dataset. Two RBF-SSFs showed much the same best performance. BFGS and newrb decreased test error to the level 1.2 and 1.3

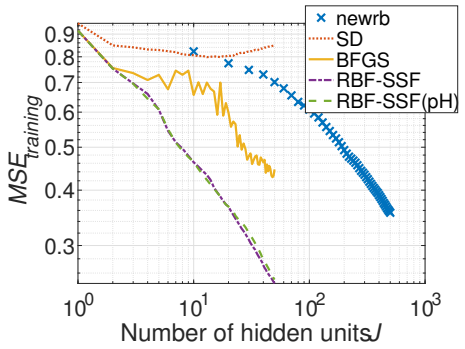

Figure 7: Training error for Parkinsons telemonitoring dataset

times larger than the best. Two RBF-SSFs and BFGS outperformed newrb for any same model complexity. SD hardly decreased test error.

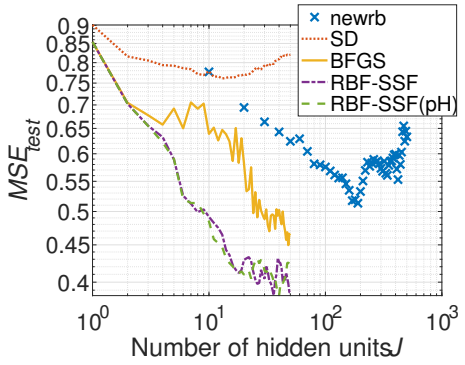

Figure 8: Test error for Parkinsons telemonitoring dataset

Figure 9 shows total processing time of five methods for Parkinsons telemonitoring dataset. Again newrb was the fastest, the original RBF-SSF was the slowest, and the remaining three required middle-level processing time. The proposed RBF-SSF(pH) ran slightly slower than SD and BFGS.

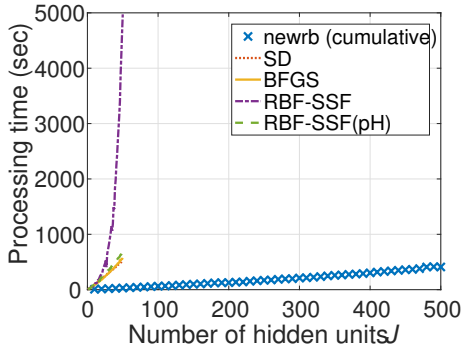

Figure 9: Total processing time for Parkinsons telemonitoring dataset

Figure 10 compares Hessian calculation time of

two RBF-SSFs. The figure clearly shows how Hessian computation time was drastically reduced to a negligible amount of time by the proposed RBF-SSF(pH) compared with the original RBF-SSF.

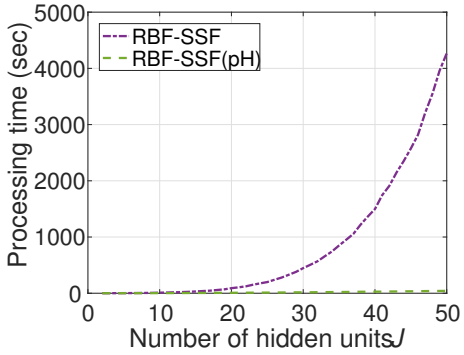

Figure 10: Processing time of Hessian calculation for Parkinsons telemonitoring dataset

Figure 11 shows the remaining processing time other than Hessian calculation of two RBF-SSFs. Two lines are mostly overlapped, showing the remaining processing time was much the same.

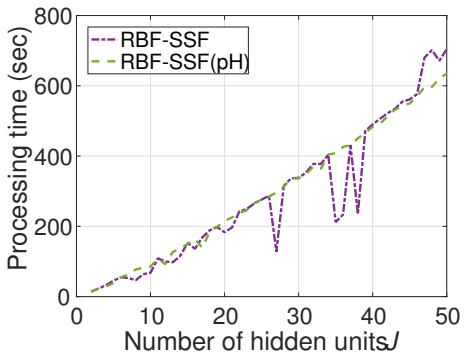

Figure 11: Processing time of other calculations for Parkinsons telemonitoring dataset

4.4 Comparison of Best Performances

Minimum training MSEs, minimum test MSEs, and total processing time through all Js are shown in Tables 2 and 3. These two tables indicate the following.

(1) Minimum Training Errors

As for minimum training MSEs through all Js for Schwefel function dataset, RBF-SSF(pH) achieved the smallest, then RBF-SSF, BFGS, newrb, and SD followed in ascending order, having 1.12, 2.70, 3.24, and 13.0 times larger than the smallest respectively. Note that newrb used 10 times larger model complexities than the other methods.

Table 2: Performance comparison for Schwefel function dataset

learning method	min training MSE	min test MSE	total processing time (sec)
newrb	0.1456	0.1579	77.88
SD	0.5823	0.5659	546.93
BFGS	0.1212	0.1382	534.34
RBF-SSF	0.0505	0.0591	715.84
RBF-SSF(pH)	0.0449	0.0527	560.85

Table 3: Performance comparison for Parkinsons telemonitoring dataset

learning method	min training MSE	min test MSE	total processing time (sec)
newrb	0.3558	0.5132	410
SD	0.7985	0.7609	12899
BFGS	0.4278	0.4496	13000
RBF-SSF	0.2463	0.3837	54902
RBF-SSF(pH)	0.2513	0.3834	15371

For Parkinsons telemonitoring dataset, RBF-SSF achieved the smallest, then RBF-SSF(pH), newrb, BFGS, and SD followed in ascending order, having 1.02, 1.44, 1.74, and 3.24 times larger training MSEs than the smallest respectively.

RBF-SSF(pH) and RBF-SSF obtained the comparable training MSEs, much smaller than those of the other methods.

(2) Minimum Test Errors

As for minimum test MSEs through all Js for Schwefel function dataset, RBF-SSF(pH) achieved the smallest, then RBF-SSF, BFGS, newrb, and SD followed in ascending order, having 1.12, 2.62, 3.00, and 10.7 times larger than the smallest respectively.

For Parkinsons telemonitoring dataset, RBF-SSF(pH) and RBF-SSF achieved the smallest, then BFGS, newrb, and SD followed in ascending order, having 1.17, 1.34, and 1.98 times larger test MSEs than the smallest respectively.

RBF-SSF(pH) and RBF-SSF obtained much the same smallest test MSEs. Although a model having

too small training error often has rather poor test error, suffering from overfitting, RBF-SSFs found solutions having minimum training and test errors. We think this is caused by the following; RBF-SSFs have the strong capability to find excellent solutions, and RBF networks are less prone to overfitting. We do not think this is caused by small-noise datasets because Parkinsons telemonitoring dataset has minimum training MSE 0.25 and minimum test MSE 0.38 for normalized data.

(3) Total Processing Time

As for total processing time for Schwefel function dataset, newrb was the fastest since it just solved linear regressions. Among the other four methods, BFGS was the fastest, and then, SD, RBF-SSF(pH), and RBF-SSF followed in order, requiring 1.02, 1.05, and 1.34 times longer time than BFGS respectively.

For Parkinsons telemonitoring dataset, newrb was the fastest, and among the other four, SD was the fastest, and then, BFGS, RBF-SSF(pH), and RBF-SSF followed in order, requiring 1.01, 1.19, and 4.26 times longer time than SD respectively.

The proposed RBF-SSF(pH) was 1.28 and 3.57 times faster than the original RBF-SSF, and was 1.05 and 1.18 times slower than BFGS. We can say RBF-SSF(pH) runs almost as fast as a good learning method BFGS. Moreover, to find excellent solutions of RBF networks, some amount of processing time will be needed.

5 CONCLUSIONS

Recently a very powerful one-stage learning method RBF-SSF has been proposed to find excellent solutions of RBF networks; however, it required a lot of time mainly because it computes the Hessian. This paper proposes a faster version of RBF-SSF called RBF-SSF(pH) by introducing partial calculation of the Hessian. The experiments using two datasets showed RBF-SSF(pH) ran as fast as usual one-stage learning methods while keeping the excellent solution quality. In the future, we plan to apply RBF-SSF(pH) to more data to prove its superiority.

ACKNOWLEDGMENT

This work was supported by Grants-in-Aid for Scientific Research (C) 16K00342.

REFERENCES

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Statist. Soc. Ser. B*, 39:1–38.

Dua, D. and Taniskidou, K. E. (2017). UCI Machine Learning Repository.

Fletcher, R. (1987). *Practical methods of optimization, 2nd edition*. John Wiley & Sons.

Fukumizu, K. and Amari, S. (2000). Local minima and plateaus in hierarchical structure of multilayer perceptrons. *Neural Networks*, 13(3):317–327.

Lázaro, M., Santamaría, I., and Pantaleón, C. (2003). A new EM-based training algorithm for RBF networks. *Neural Networks*, 16:69–77.

Little, M., McSharry, P., Roberts, S., Costello, D., and Moroz, I. (2007). Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. *Biomedical Engineering OnLine*, 6(23).

Nitta, T. (2013). Local minima in hierarchical structures of complex-valued neural networks. *Neural Networks*, 43:1–7.

Satoh, S. and Nakano, R. (2013). Multilayer perceptron learning utilizing singular regions and search pruning. In *Proc. Int. Conf. on Machine Learning and Data Analysis*, pages 790–795.

Satoh, S. and Nakano, R. (2015). A yet faster version of complex-valued multilayer perceptron learning using singular regions and search pruning. In *Proc. of 7th Int. Joint Conf. on Computational Intelligence (IJCCI)*, volume 3 NCTA, pages 122–129.

Satoh, S. and Nakano, R. (2018). A new method for learning RBF networks by utilizing singular regions. In *Proc. of 17th Int. Conf. on Artificial Intelligence and Soft Computing (ICAISC)*, pages 214–225.

Schwefel, H.-P., editor (1981). *Numerical optimization of computer models*. John Wiley & Sons, Inc.

Schwenker, F., Kestler, H., and Palm, G. (2001). Three learning phases for radial-basis-function networks. *Neural Networks*, 14(4-5):439–458.

Wu, Y., Wang, H., Zhang, B., and Du, K.-L. (2012). Using radial basis function networks for function approximation and classification. *ISRN Applied Mathematics*, 2012:1–34.