Stable Learning Algorithm Using Reducibility for Recurrent Neural Networks

Seiya Satoh(✉)

Tokyo Denki University, Hiki-gun, Saitama 350-0394, Japan
seiya.satoh@mail.dendai.ac.jp

Abstract. A multilayer perceptron (MLP), a feedforward neural network with one hidden layer, is called reducible if a hidden unit can be removed without changing the input–output function. In the MLP's search space, the MLP is reducible in some regions, and some regions in the reducible regions have zero gradients, where learning stagnates. Nonetheless, some methods have been proposed to leverage such regions. A reducible region of an MLP with J hidden units can be generated from an MLP with $J-1$ hidden units. To begin learning from a reducible region guarantees a monotonically decreasing training error as the number of hidden units increases. The evaluation experiments reveal that the methods using reducible regions stably determine better solutions than existing methods. In addition, methods using reducible regions can stably obtain high-quality solutions not only for MLPs, but also for complex-valued MLPs, and radial basis function networks. In this study, we show that the search space of a recurrent neural network also has reducible regions. In addition, we propose a method that utilizes reducible regions to obtain higher quality solutions in a stable manner, as compared to existing methods with randomly set initial weights.

Keywords: Recurrent neural network · Reducibility · Reducible region · Hessian matrix · Eigenvector

1 Introduction

Reducibility is a crucial characteristic of multilayer perceptrons (MLPs), feedforward neural networks with one hidden layer [5,24]. In several regions, *reducible* MLPs exist in the parameter space (referred to as *reducible regions* in this study). A reducible region of an MLP may contain a continuous region with zero gradient, where learning stagnates. However, methods utilizing such regions have been proposed [18,19], where a reducible MLP with J hidden units is generated from an MLP solution with $J-1$ hidden units, and learning is started from there. As these methods use reducible regions and start learning by increasing the number of hidden units in succession, the training error decreases monotonically as the number of hidden units increases. Previous experiments in the literature have

L. Iliadis et al. (Eds.): ICANN 2023, LNCS 14259, pp. 127–139, 2023.
https://doi.org/10.1007/978-3-031-44223-0_11

obtained solutions suitable for the number of hidden units, which were of better quality than those obtained from existing methods that randomly initialize the weights.

Complex-valued MLPs, whose parameters are all complex-valued, also have reducibility [10,11,16,17]. Methods using reducible regions of a complex-valued MLP have also been proposed, and like real-valued MLPs, they exhibit better solutions than those obtained using existing methods that randomly initialize the weights [20,21]. Similarly, radial basis function networks also have reducibility, and a method using reducible regions outperformed existing methods [22]. Further, research on the reducibility of quaternionic and hyperbolic neural networks has also been performed [12,13].

Recurrent neural networks (RNNs) are used in a variety of fields, including time series forecasting, speech recognition, and human action recognition [1,6,7]. Several methods have been proposed as RNN learning methods [3,9,25,27]. Previous studies showed that a quasi-Newton method tends to outperform a steepest descent method. However, the quasi-Newton method may not obtain reasonable quality solutions depending on the initial weights.

The present study shows that reducible regions also exist in an RNN parameter space. In addition, a learning method using reducible regions for an RNN is proposed herein. This method monotonically decreases the training error as the number of hidden units increases. Further, an experiment is conducted to demonstrate that better quality solutions can be stably obtained using the proposed method than those obtained using existing methods that randomly initialize the weights.

This study is organized as follows. Section 2 describes basic definitions. Section 3 shows that reducible regions exist in an RNN parameter space. Section 4 proposes a new learning method using reducible regions. Section 5 evaluates the proposed method. Finally, Sect. 6 concludes the study and discusses future work.

2 Basic Definitions

Among the various models of RNNs, Elman RNNs are considered herein [14,23,26]. Let K be the number of input units, J be the number of hidden units, and I be the number of output units. We use the following notation.

$\boldsymbol{w}_j^{(J)}$ Weights $\left(w_{0,j}^{(J)}, ..., w_{K,j}^{(J)}\right)^{\mathrm{tr}}$ from all input units to the jth hidden unit, where $w_{0,j}^{(J)}$ is the bias, and $\boldsymbol{a}^{\mathrm{tr}}$ denotes the transpose of a vector $\boldsymbol{a}$.

$\boldsymbol{W}^{(J)}$ Weights $\left(\boldsymbol{w}_1^{(J)}, ..., \boldsymbol{w}_J^{(J)}\right)$ from all input units to all hidden units.

$\boldsymbol{v}_i^{(J)}$ Weights $\left(v_{0,i}^{(J)}, ..., v_{J,i}^{(J)}\right)^{\mathrm{tr}}$ from all hidden units to the ith output unit, where $v_{0,i}^{(J)}$ is the bias term.

$\boldsymbol{V}^{(J)}$ Weights $\left(\boldsymbol{v}_1^{(J)}, ..., \boldsymbol{v}_I^{(J)}\right)$ from all hidden units to all output units.

$\boldsymbol{r}_j^{(J)}$ Recurrent weights $\left(r_{1,j}^{(J)}, ..., r_{J,j}^{(J)}\right)^{\mathrm{tr}}$ from all hidden units to the jth hidden unit.

$\boldsymbol{R}^{(J)}$ Recurrent weights $\left(\boldsymbol{r}_1^{(J)}, ..., \boldsymbol{r}_J^{(J)}\right)$ from all hidden units to all hidden units.

$\boldsymbol{\theta}^{(J)}$ A vector of all weights $\boldsymbol{W}^{(J)}$, $\boldsymbol{V}^{(J)}$, and $\boldsymbol{R}^{(J)}$.

$\boldsymbol{x}^{(t)}$ Input signals $\left(1, x_1^{(t)}, ..., x_K^{(t)}\right)^{\mathrm{tr}}$ at time t.

$\boldsymbol{X}_{t_0,t_1}$ Input signals $\left(\boldsymbol{x}^{(t_0)}, ..., \boldsymbol{x}^{(t_1)}\right)$ from time t_0 to $t_1 (\geq t_0)$.

Given input signals $\boldsymbol{X}_{t_0,t_1}$, the output of the jth hidden unit is

$$z_j\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right) = \begin{cases} \sigma\left(\boldsymbol{w}_j^{(J)\mathrm{tr}} \boldsymbol{x}^{(t_1)}\right), & \text{if } t_0 = t_1; \\ \sigma\left(\boldsymbol{w}_j^{(J)\mathrm{tr}} \boldsymbol{x}^{(t_1)} + \boldsymbol{r}_j^{(J)\mathrm{tr}} \boldsymbol{z}\left(\boldsymbol{X}_{t_0,t_1-1}; \boldsymbol{\theta}^{(J)}\right)\right), & \text{if } t_0 < t_1, \end{cases} \tag{1}$$

where $\boldsymbol{z}\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right) \equiv \left(z_1\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right), ..., z_J\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right)\right)^{\mathrm{tr}}$, and we consider a sigmoid function $\sigma(x) = 1/(1 + e^{-x})$ as the activation function. As hidden units have recurrent weights, the outputs typically depend on the values of $\boldsymbol{x}^{(t_0)}, ..., \boldsymbol{x}^{(t_1)}$, not just $\boldsymbol{x}^{(t_1)}$.

Herein, an identity function is considered as the activation function of output units. The output of the ith output unit is

$$f_i\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right) = \boldsymbol{v}_i^{(J)\mathrm{tr}} \widetilde{\boldsymbol{z}}\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right), \tag{2}$$

where $\widetilde{\boldsymbol{z}}\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right) \equiv \left(1, z_1\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right), ..., z_J\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right)\right)^{\mathrm{tr}}$. Using weights $\boldsymbol{V}^{(J)}$, the outputs $\boldsymbol{f}\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right) \equiv \left(f_1\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right), ..., f_I\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right)\right)^{\mathrm{tr}}$ of the 1st to Ith output units can be calculated as:

$$\boldsymbol{f}\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right) = \boldsymbol{V}^{(J)\mathrm{tr}} \widetilde{\boldsymbol{z}}\left(\boldsymbol{X}_{t_0,t_1}; \boldsymbol{\theta}^{(J)}\right). \tag{3}$$

3 Reducible Regions

An RNN is considered *reducible* when a hidden unit can be removed such that the RNN's input-output function does not change. This Section considers the regions of a reducible RNN, referred to as *reducible regions*.

First, consider a solution for an RNN with $J - 1$ hidden units. Here all weights of such a solution are denoted by symbols with hats, such as $\widehat{w}_{K,1}^{(J-1)}$, $\widehat{\boldsymbol{v}}_i^{(J-1)}$, $\widehat{\boldsymbol{R}}^{(J-1)}$, and $\widehat{\boldsymbol{\theta}}^{(J-1)}$.

Next, consider the following region.

$$\widehat{\boldsymbol{\Theta}}^{(J)}_{\gamma_j} = \Big\{\boldsymbol{\theta}^{(J)}\Big|\boldsymbol{W}^{(J)} = \left(\widehat{\boldsymbol{W}}^{(J-1)}, \widehat{\boldsymbol{w}}^{(J-1)}_j\right), v^{(J)}_{0,i} = \widehat{v}^{(J-1)}_{0,i}, v^{(J)}_{j',i} = \widehat{v}^{(J-1)}_{j',i},$$
$$v^{(J)}_{j,i} = p_i\widehat{v}^{(J-1)}_{j,i}, v^{(J)}_{J,i} = (1-p_i)\widehat{v}^{(J-1)}_{j,i}, r^{(J)}_{j',j''} = \widehat{r}^{(J-1)}_{j',j''}, r^{(J)}_{j',J} = \widehat{r}^{(J-1)}_{j',j},$$
$$r^{(J)}_{j,j''} = q_{j''}\widehat{r}^{(J-1)}_{j,j''}, r^{(J)}_{J,j''} = (1-q_{j''})\widehat{r}^{(J-1)}_{j,j''}, r^{(J)}_{J,J} = (1-q_J)\widehat{r}^{(J-1)}_{j,j},$$
$$r^{(J)}_{j,J} = q_J\widehat{r}^{(J-1)}_{j,j}, j'\in\{1,...,J-1\}\setminus\{j\}, j''\in\{1,...,J-1\}, i\in\{1,...,I\}\Big\}, (4)$$

where $j \in \{1,...,J-1\}$, and $p_1,...,p_I,q_1,...,q_J$ are all arbitrary real numbers. Figure 1 shows an example of a region $\widehat{\boldsymbol{\Theta}}^{(J)}_{\gamma_j}$. A region $\widehat{\boldsymbol{\Theta}}^{(J)}_{\gamma_j}$ is a reducible region because it satisfies the following theorem, the proof of which can be found in the appendix.

Theorem 1. $\boldsymbol{f}\left(\boldsymbol{X}_{t_0,t_1};\boldsymbol{\theta}^{(J)}\right) = \boldsymbol{f}\left(\boldsymbol{X}_{t_0,t_1};\widehat{\boldsymbol{\theta}}^{(J-1)}\right)$ where $\boldsymbol{\theta}^{(J)} \in \widehat{\boldsymbol{\Theta}}^{(J)}_{\gamma_j}$, and $t_0 \leq t_1$.

For MLPs, three types to generate reducible MLPs are known [5,24]. As for RNNs, other types, in addition to the aforementioned, may also generate reducible RNNs; however, the proposed method uses only the aforementioned type.

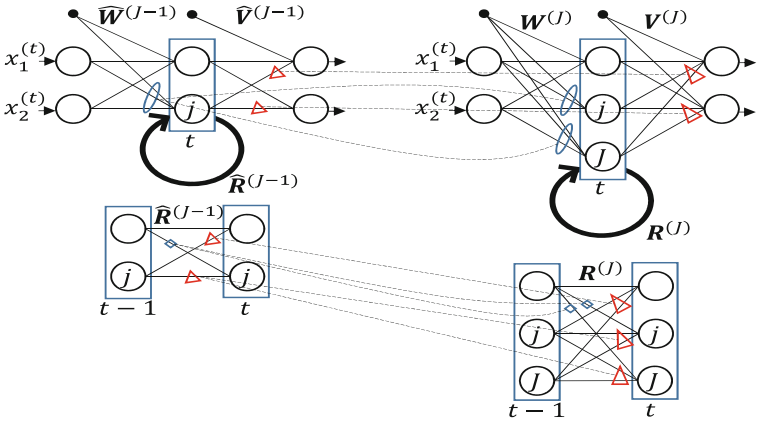

Fig. 1. Example of a region $\widehat{\boldsymbol{\Theta}}^{(J)}_{\gamma_j}$ when $K = 2, I = 2, j = 2$, and $J = 3$. The left side is the solution for an RNN with $J-1$ hidden units, whereas the right side is an RNN with J hidden units in the region $\widehat{\boldsymbol{\Theta}}^{(J)}_{\gamma_j}$. Weights across an ellipse on the right are the same as those on the left. A weight marked with a diamond on the right is the same as one marked with a diamond on the left. The sum of weights that touch a triangle is the same as one marked with a triangle on the left.

4 Reducible Region Descent

This section proposes reducible region descent (RRD), a method to start learning an RNN from a reducible region. When learning begins from a reducible region, the training error for an RNN with J hidden units becomes smaller than that of the solution with $J-1$ hidden units. Therefore, the training error monotonically decreases with increasing number of hidden units. This section explains the descent from a reducible region and the proposed method's processing flow.

4.1 Descent from a Reducible Region

Previous studies have proven that several reducible regions exist in the MLP parameter space and continuous regions with zero gradients exist in the reducible regions [5]. Similarly, points or regions with zero gradients may be in a reducible region $\widehat{\boldsymbol{\Theta}}_{\gamma_j}^{(J)}$. If the gradient is zero, learning cannot begin by a gradient-based method from the region. Therefore, herein, the eigenvalues and eigenvectors of the Hessian matrix were used, similar to the method of using reducible regions of an MLP [18]. Even if the gradient is zero, if a negative eigenvalue exists in the Hessian matrix, learning can be started in the direction of the eigenvector corresponding to the negative eigenvalue. Hence, herein, negative eigenvalues and their corresponding eigenvectors of Hessian matrices were used in the proposed method.

4.2 Procedure of the Proposed Method

Algorithm 1 shows the procedure of RRD. In addition, Fig. 2 shows a conceptual diagram of the RRD flow. In Step 1 of Algorithm 1, the initial values of the weights were randomly selected from the interval $(-1, 1)$, and the number of learning trials was set to 10 in the experiment performed herein. Further, Broyden–Fletcher–Goldfarb–Shanno (BFGS) [4], a type of quasi-Newton

Algorithm 1. RRD

1: Randomly initialize the weights of an RNN with one hidden unit and perform learning several times
2: Select a solution with the smallest training error and let the solution be denoted by $\widehat{\boldsymbol{\theta}}^{(1)}$
3: **for** $J = 2, ..., J_{\max}$ **do**
4: Generate reducible regions $\widehat{\boldsymbol{\Theta}}_{\gamma_1}^{(J)}, ..., \widehat{\boldsymbol{\Theta}}_{\gamma_{J-1}}^{(J)}$ from $\widehat{\boldsymbol{\theta}}^{(J-1)}$ and select several points on the reducible regions
5: Calculate the Hessian matrices at the selected points and calculate the eigenvalues and eigenvectors
6: Perform learning in the directions and the opposite directions of the eigenvectors corresponding to negative eigenvalues in the calculated eigenvalues
7: Let a solution with the smallest training error be $\widehat{\boldsymbol{\theta}}^{(J)}$
8: **end for**

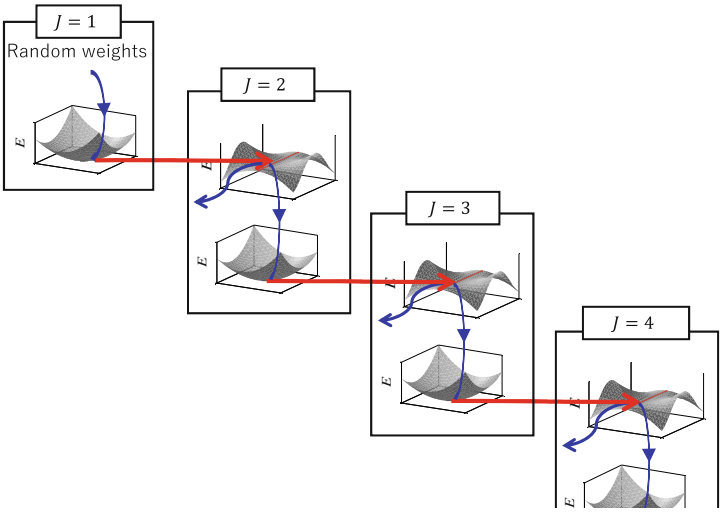

Fig. 2. Conceptual diagram of the RRD flow.

method, was used as the learning method. In BFGS, a line search was used where the initial value of the step length was set to 1. If the training error did not decrease, the step length was repeatedly halved. Learning by BFGS was stopped when the number of weight updates reached 1,000 or the step length for the line search became less than 10^{-16}. In Step 3, the maximum number $J_{\max}$ of hidden units was set to 20.

In Step 4, it is necessary to determine which points on the reducible regions will be used. In the experiment, the points where $p_1 = \cdots = p_I = q_1 = \cdots = q_J = 0.5$ and points where $p_1 = \cdots = p_I = q_1 = \cdots = q_J = 1.5$ were selected. Note that at 0.5, the signs of pair weights are the same, and at 1.5, the signs of pair weights are different. These points were selected for each $j \in \{1, ..., J-1\}$. Therefore, the number of points used in Step 4 was $2 \times (J-1)$.

The learning trials in Step 6 can be very time-consuming if numerous negative eigenvalues exist and learning trials are started in the directions of eigenvectors corresponding to all negative eigenvalues. Hence, herein, the eigenvalues were selected in order, starting with the smallest eigenvalue, and the upper limit of the number of learning trials was set to 10. In Step 6, BFGS was used after the weights were updated once in the direction (or in the opposite direction) of an eigenvector. In BFGS, the exact line search and stopping criteria were used as in Step 1.

5 Experiment

To evaluate the proposed method, an experiment was performed to predict the Lorenz attractor [15]. A computer with Ryzen 9, 3950X, and 64 GB RAM

was used. Further, Julia Version 1.8.5, with DifferentialEquations Version 7.7.0, Zygote Version 0.6.55, Statistics, LinearAlgebra, and Random packages, was used as the programming language.

5.1 Settings

The Lorenz equations are $\frac{da}{dt} = \alpha(b-a)$, $\frac{db}{dt} = a(\rho - c) - b$, and $\frac{dc}{dt} = ab - \beta c$. We set ρ, α, and β to 28, 10, and 8/3, respectively, and set the initial values of a, b, and c to –10, –10, and 30, respectively. Figure 3 shows the trajectory of the a, b and c at $t = 0, \Delta t, \cdots, 499\Delta t$. Here, Δt was set to 0.05 and Tsit5 solver was used to generate the trajectory by setting the tolerance option as reltol = abstol = 10^{-6}.

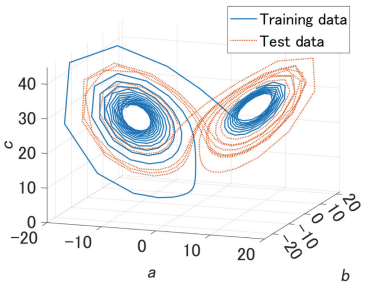

Fig. 3. Trajectory of the Lorenz attractor.

Given inputs a_t, b_t, and c_t, an RNN predicts one point ahead (Δt ahead) $a_{t+\Delta t}, b_{t+\Delta t}$, and $c_{t+\Delta t}$. Herein, 300 points were used as training data, as follows. $((a_0, b_0, c_0)$, $(a_{\Delta t}, b_{\Delta t}, c_{\Delta t})$, ..., $(a_{299\Delta t}, b_{299\Delta t}, c_{299\Delta t}))$:

$$\begin{aligned} \boldsymbol{X}_{1,299}^{(train)} = & \big((1, a_0, b_0, c_0)^{\mathrm{tr}}, (1, a_{\Delta t}, b_{\Delta t}, c_{\Delta t})^{\mathrm{tr}}, ..., \\ & (1, a_{298\Delta t}, b_{298\Delta t}, c_{298\Delta t})^{\mathrm{tr}}\big), \end{aligned} \tag{5}$$

$$\begin{aligned} \boldsymbol{Y}_{1,299}^{(train)} = & \big((1, a_{\Delta t}, b_{\Delta t}, c_{\Delta t})^{\mathrm{tr}}, (1, a_{2\Delta t}, b_{2\Delta t}, c_{2\Delta t})^{\mathrm{tr}}, ..., \\ & (1, a_{299\Delta t}, b_{299\Delta t}, c_{299\Delta t})^{\mathrm{tr}}\big). \end{aligned} \tag{6}$$

Further, 200 points were used as test data, as follows. $((a_{300\Delta t}, b_{300\Delta t}, c_{300\Delta t})$, $(a_{301\Delta t}, b_{301\Delta t}, c_{301\Delta t})$, ..., $(a_{499\Delta t}, b_{499\Delta t}, c_{499\Delta t}))$:

$$\begin{aligned} \boldsymbol{X}_{1,199}^{(test)} = & \big((1, a_{300\Delta t}, b_{300\Delta t}, c_{300\Delta t})^{\mathrm{tr}}, (1, a_{301\Delta t}, b_{301\Delta t}, c_{301\Delta t})^{\mathrm{tr}}, ..., \\ & (1, a_{498\Delta t}, b_{498\Delta t}, c_{498\Delta t})^{\mathrm{tr}}\big), \end{aligned} \tag{7}$$

$$\begin{aligned} \boldsymbol{Y}_{1,199}^{(test)} = & \big((1, a_{301\Delta t}, b_{301\Delta t}, c_{301\Delta t})^{\mathrm{tr}}, (1, a_{302\Delta t}, b_{302\Delta t}, c_{302\Delta t})^{\mathrm{tr}}, ..., \\ & (1, a_{499\Delta t}, b_{499\Delta t}, c_{499\Delta t})^{\mathrm{tr}}\big). \end{aligned} \tag{8}$$

In forecasting time series data, interests are often only in predicting the future. Hence, herein, the many-to-one architecture [2,8] was used, and the following equation was considered as the objective function of the many-to-one model.

$$E_{many-to-one}(\boldsymbol{X}_{1,T}, \boldsymbol{Y}_{1,T}) = \sum_{\tau_1=h}^{T} e\big(\boldsymbol{X}_{\tau_1-h+1,\tau_1}, \boldsymbol{y}_{\tau_1}; \boldsymbol{\theta}^{(J)}\big), \tag{9}$$

where $e\big(\boldsymbol{X}_{\tau_0,\tau_1}, \boldsymbol{y}_{\tau_1}; \boldsymbol{\theta}^{(J)}\big) \equiv \Big(\boldsymbol{f}\big(\boldsymbol{X}_{\tau_0,\tau_1}; \boldsymbol{\theta}^{(J)}\big) - \boldsymbol{y}_{\tau_1}\Big)^{\mathrm{tr}} \Big(\boldsymbol{f}\big(\boldsymbol{X}_{\tau_0,\tau_1}; \boldsymbol{\theta}^{(J)}\big) - \boldsymbol{y}_{\tau_1}\Big)$, and h is an integer greater than or equal to 1 and indicates the number of previous inputs that are reflected. In the experiment, h was set to

10. Here, we define $E_{tr} \equiv E_{many-to-one}(\boldsymbol{X}_{1,299}^{(train)}, \boldsymbol{Y}_{1,299}^{(train)})$, and $E_{test} \equiv E_{many-to-one}(\boldsymbol{X}_{1,199}^{(test)}, \boldsymbol{Y}_{1,199}^{(test)})$.

Moreover, steepest descent (SD) and BFGS with random initial weights were used as existing methods. Here, these two methods are called random initialization-SD (RI-SD) and RI-BFGS, respectively. In RI-SD and RI-BFGS, the initial values of the weights were randomly selected from the interval $(-1, 1)$. The exact line search and stopping criteria used in RRD were also used. Additionally, the range of hidden units was set to 1–20 in both RI-SD and RI-BFGS. Furthermore, the number of trials was set to 10 for each number of hidden units, matching the settings in RRD.

5.2 Results

Figures 4 and 5 show each method's smallest training and test errors for each number of hidden units.

Although the training error obtained by RI-BFGS was significantly smaller than RI-SD, the training error did not monotonically decrease with increasing number of hidden units. By contrast, the training error obtained by RRD decreased monotonically with increasing number of hidden units and was smaller than that obtained by RI-BFGS. The test error obtained by RRD was also the smallest.

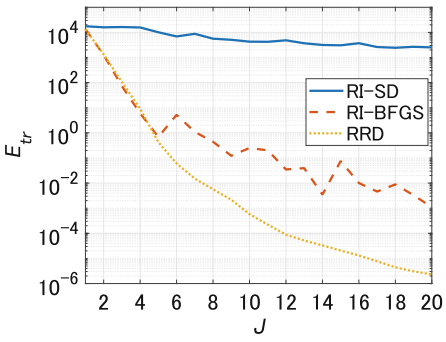

Fig. 4. Training error.

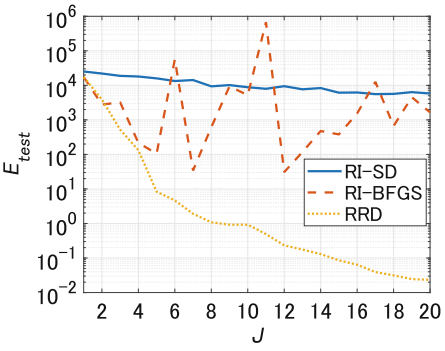

Fig. 5. Test error.

Further, the total processing times of the RI-SD, RI-BFGD, and RRD methods were 144, 51, and 65 min, respectively. RRD took more time compared with RI-BFGS, partly because RRD requires the calculation of the Hessian matrices on reducible regions. RI-SD took the longest time because the line search took a long time.

5.3 More Weight Updates for RI-BFGS

Since a solution with $J-1$ hidden units is used for learning an RNN with J hidden units in RRD, it can be interpreted that the number of weight updates

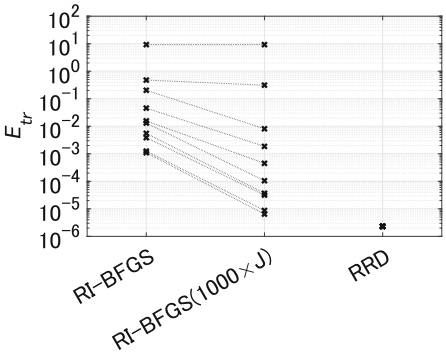

Fig. 6. Training errors when $J = 20$.

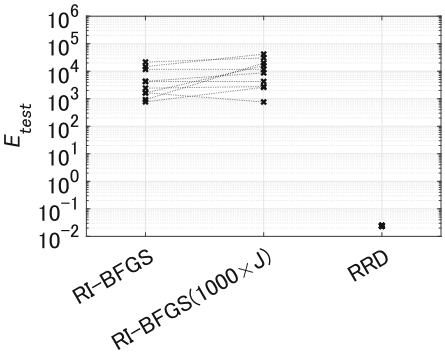

Fig. 7. Test errors when $J = 20$.

is accumulated, and the total number of weight updates is $1000 \times J$. Thus, RRD was compared with RI-BFGS, where the upper limit of weight updates was set to $1000 \times J$. Here, this method is called RI-BFGS($1000 \times J$). All other settings remained unchanged, consistent with those described in Sect. 5.1.

Figures 6 and 7 show the training and test errors when $J = 20$.

In RI-BFGS($1000 \times J$), the training error decreased with increasing number of weight updates; however, the smallest training error was still larger than an RRD training error. Interestingly, most of the test errors obtained by RI-BFGS($1000 \times J$) increased with increasing number of weight updates. By contrast, RRD enabled determining solutions with small training and test errors in a stable manner. The mean and standard deviation of the ten test errors obtained by RI-BFGS($1000 \times J$) when $J = 20$ were 1.39×10^4 and 1.35×10^4, respectively, whereas those obtained by RRD were 0.0242 and 6.93×10^{-4}, respectively.

6 Conclusion

This study shows that reducible regions exist in an RNN parameter space. In addition, a new learning method called RRD, which uses reducible regions generated from a solution, was proposed. Further, an evaluation experiment was conducted to predict the Lorenz attractor, wherein RRD stably determined quality solutions, thereby outperforming existing methods.

In the future, we plan to evaluate RRD using various datasets. We also plan to investigate the structure of reducible regions in the parameter space used in the procedure of RRD, in particular whether continuous regions exist where the gradient is zero. In addition, we plan to investigate other types to generate reducible regions.

Acknowledgements. This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

Appendix

First, we define, $z_{j''}\big(\boldsymbol{X}_{t_0,t_1};\widehat{\boldsymbol{\theta}}^{(J-1)}\big)\equiv \quad \widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_1})$, $z_{j''}\big(\boldsymbol{X}_{t_0,t_1};\boldsymbol{\theta}^{(J)}\big)\equiv z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_1})$ where $\boldsymbol{\theta}^{(J)}\in\widehat{\boldsymbol{\Theta}}_{\gamma_j}^{(J)}$. Next, the following lemma can be derived.

Lemma 1. $z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_1})=\widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_1})$, $z_J^{(J)}\big(\boldsymbol{X}_{t_0,t_1}\big)=\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_1})$ *where* $j''\in\{1,...,J-1\}$.

This lemma suggests that all outputs of the 1st to $J-1$th hidden units are the same as those of the solution $\widehat{\boldsymbol{\theta}}^{(J-1)}$, and the output of the Jth hidden unit is the same as that of the jth hidden unit of the solution $\widehat{\boldsymbol{\theta}}^{(J-1)}$.

Proof. Using mathematical induction, this lemme can be proven. More specifically, first, let us show that the outputs of hidden units are the same for $t_1=t_0+1$. Next, assuming that the outputs are the same when $t_1=t_0+h$, let us show that the outputs are the same when $t_1=t_0+h+1$.

To show that the outputs are the same when $t_1=t_0+1$, consider the case when $t_1=t_0$. From $\widehat{\boldsymbol{w}}_{j''}^{(J-1)}=\boldsymbol{w}_{j''}^{(J)},j''\in\{1,...,J-1\}$, and $\widehat{\boldsymbol{w}}_j^{(J-1)}=\boldsymbol{w}_J^{(J)}$, $\widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_0})=z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_0}),j''\in\{1,...,J-1\}$, and $\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0})=z_J(\boldsymbol{X}_{t_0,t_0})$.

When $t_1=t_0+1$, the following is noted.

$$
\begin{aligned}
\widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_0+1}) &= \sigma\Bigg(\widehat{\boldsymbol{w}}_{j''}^{(J-1)^{\mathrm{tr}}}\boldsymbol{x}^{(t)}+\sum_{j'=1}^{J-1}\widehat{r}_{j',j''}^{(J-1)}\widehat{z}_{j'}^{(J-1)}(\boldsymbol{X}_{t_0,t_0})\Bigg)\\
&= \sigma\Bigg(\widehat{\boldsymbol{w}}_{j''}^{(J-1)^{\mathrm{tr}}}\boldsymbol{x}^{(t)}+\sum_{j'\in\{1,...,J-1\}\setminus\{j\}}\widehat{r}_{j',j''}^{(J-1)}\widehat{z}_{j'}^{(J-1)}(\boldsymbol{X}_{t_0,t_0})\\
&\quad +b_{j''}\widehat{r}_{j,j''}^{(J-1)}\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0})+(1-b_{j''})\widehat{r}_{j,j''}^{(J-1)}\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0})\Bigg)\\
&= \sigma\Bigg(\boldsymbol{w}_{j''}^{(J)^{\mathrm{tr}}}\boldsymbol{x}^{(t)}+\sum_{j'\in\{1,...,J-1\}\setminus\{j\}}r_{j',j''}^{(J)}z_{j'}^{(J)}(\boldsymbol{X}_{t_0,t_0})\\
&\quad +r_{j,j''}^{(J)}z_j^{(J)}(\boldsymbol{X}_{t_0,t_0})+r_{J,j''}^{(J)}z_J^{(J)}(\boldsymbol{X}_{t_0,t_0})\Bigg)\\
&= z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_0+1}),
\end{aligned}
\tag{10}
$$

where $j''\in\{1,...,J-1\}$. Moreover,

$$
\begin{aligned}
\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0+1}) &= \sigma\Bigg(\widehat{\boldsymbol{w}}_j^{(J-1)^{\mathrm{tr}}}\boldsymbol{x}^{(t)}+\sum_{j'=1}^{J-1}\widehat{r}_{j',j}^{(J-1)}\widehat{z}_{j'}^{(J-1)}(\boldsymbol{X}_{t_0,t_0})\Bigg)\\
&= \sigma\Bigg(\widehat{\boldsymbol{w}}_j^{(J-1)^{\mathrm{tr}}}\boldsymbol{x}^{(t)}+\sum_{j'\in\{1,...,J-1\}\setminus\{j\}}\widehat{r}_{j',j''}^{(J-1)}\widehat{z}_{j'}^{(J-1)}(\boldsymbol{X}_{t_0,t_0})
\end{aligned}
$$

$$
\begin{aligned}
&+b_J \widehat{r}_{j,j}^{(J-1)} \widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0}) + (1-b_J)\widehat{r}_{j,j}^{(J-1)} \widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0})\Bigg) \\
&= \sigma\Bigg(\boldsymbol{w}_J^{(J)^{\mathrm{tr}}} \boldsymbol{x}^{(t)} + \sum_{j' \in \{1,\ldots,J-1\}\setminus\{j\}} r_{j',J}^{(J)} z_{j'}^{(J)}(\boldsymbol{X}_{t_0,t_0}) \\
&\quad + r_{j,J}^{(J)} z_j^{(J)}(\boldsymbol{X}_{t_0,t_0}) + r_{J,J}^{(J)} z_J^{(J)}(\boldsymbol{X}_{t_0,t_0})\Bigg) \\
&= z_J^{(J)}(\boldsymbol{X}_{t_0,t_0+1}). \qquad (11)
\end{aligned}
$$

Next, let us assume that $\widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_0+h}) = z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_0+h})$, $j'' \in \{1,\ldots,J-1\}$, $\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0+h}) = z_J^{(J)}(\boldsymbol{X}_{t_0,t_0+h})$, and consider the case when $t_1 = t_0+h+1$. If the equations are transformed in the same manner as Eqs. (10) and (11), the following is obtained.

$$
\widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_0+h+1}) = z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_0+h+1}), \qquad (12)
$$

$$
\widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_0+h+1}) = z_J^{(J)}(\boldsymbol{X}_{t_0,t_0+h+1}), \qquad (13)
$$

where $j'' \in \{1,\ldots,J-1\}$. From Eqs. (10), (11), (12), and (13), we obtain:

$$
\widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_1}) = z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_1}), \quad \widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_1}) = z_J^{(J)}(\boldsymbol{X}_{t_0,t_1}), \qquad (14)
$$

where $j'' \in \{1,\ldots,J-1\}$, and $t_1 \geq t_0$. □

Now, let us return to the proof of Theorem 1.

Proof. From Lemma 1, we obtain the following.

$$
\begin{aligned}
f_i\big(\boldsymbol{X}_{t_0,t_1};\boldsymbol{\theta}^{(J)}\big) &= \widehat{v}_{0,i}^{(J-1)} + \sum_{j''=1}^{J-1} \widehat{v}_{j'',i}^{(J-1)} \widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_1}) \\
&= \widehat{v}_{0,i}^{(J-1)} + \sum_{j'' \in \{1,\ldots,J-1\}\setminus\{j\}} \widehat{v}_{j'',i}^{(J-1)} \widehat{z}_{j''}^{(J-1)}(\boldsymbol{X}_{t_0,t_1}) \\
&\quad + p_i \widehat{v}_{j,i}^{(J-1)} \widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_1}) + (1-p_i)\widehat{v}_{j,i}^{(J-1)} \widehat{z}_j^{(J-1)}(\boldsymbol{X}_{t_0,t_1}) \\
&= v_{0,i}^{(J)} + \sum_{j'' \in \{1,\ldots,J-1\}\setminus\{j\}} v_{j'',i}^{(J)} z_{j''}^{(J)}(\boldsymbol{X}_{t_0,t_1}) \\
&\quad + v_{j,i}^{(J)} z_j^{(J)}(\boldsymbol{X}_{t_0,t_1}) + v_{J,i}^{(J)} z_J^{(J)}(\boldsymbol{X}_{t_0,t_1}) \\
&= f_i\big(\boldsymbol{X}_{t_0,t_1};\widehat{\boldsymbol{\theta}}^{(J-1)}\big) \qquad (15)
\end{aligned}
$$

where $i \in \{1,\ldots,I\}$. □

References

1. Baccouche, M., Mamalet, F., Wolf, C., Garcia, C., Baskurt, A.: Sequential deep learning for human action recognition. In: Salah, A.A., Lepri, B. (eds.) HBU 2011. LNCS, vol. 7065, pp. 29–39. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25446-8_4

2. Dadoun, A., Troncy, R.: Many-to-one recurrent neural network for session-based recommendation. arXiv preprint arXiv:2008.11136 (2020)
3. De Jeses, O., Hagan, M.T.: Backpropagation through time for a general class of recurrent network. In: International Joint Conference on Neural Networks, vol. 4, pp. 2638–2643. IEEE (2001)
4. Fletcher, R.: Practical Methods of Optimization, 2nd edn. John Wiley & Sons, Hoboken (1987)
5. Fukumizu, K., Amari, S.: Local minima and plateaus in hierarchical structures of multilayer perceptrons. Neural Netw. **13**(3), 317–327 (2000)
6. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: International Conference on Acoustics, Speech and Signal Processing, pp. 6645–6649. IEEE (2013)
7. Hewamalage, H., Bergmeir, C., Bandara, K.: Recurrent neural networks for time series forecasting: current status and future directions. Int. J. Forecast. **37**(1), 388–427 (2021)
8. Kaur, M., Mohta, A.: A review of deep learning with recurrent neural network. In: International Conference on Smart Systems and Inventive Technology, pp. 460–465. IEEE (2019)
9. Keskar, N.S., Berahas, A.S.: adaQN: an adaptive quasi-newton algorithm for training RNNs. In: Frasconi, P., Landwehr, N., Manco, G., Vreeken, J. (eds.) ECML PKDD 2016. LNCS (LNAI), vol. 9851, pp. 1–16. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46128-1_1
10. Kobayashi, M.: Exceptional reducibility of complex-valued neural networks. IEEE Trans. Neural Netw. **21**(7), 1060–1072 (2010)
11. Kobayashi, M.: Singularities of three-layered complex-valued neural networks with split activation function. IEEE Trans. Neural Netw. Learn. Syst. **29**(5), 1900–1907 (2017)
12. Kobayashi, M.: Uniqueness theorem for quaternionic neural networks. Signal Process. **136**, 102–106 (2017)
13. Kobayashi, M.: Reducibilities of hyperbolic neural networks. Neurocomputing **378**, 129–141 (2020)
14. Krichene, E., Masmoudi, Y., Alimi, A.M., Abraham, A., Chabchoub, H.: Forecasting using elman recurrent neural network. In: Madureira, A.M., Abraham, A., Gamboa, D., Novais, P. (eds.) ISDA 2016. AISC, vol. 557, pp. 488–497. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-53480-0_48
15. Lorenz, E.N.: Deterministic nonperiodic flow. J. Atmos. Sci. **20**(2), 130–141 (1963)
16. Nitta, T.: Reducibility of the complex-valued neural network. Neural Inf. Process.-Lett. Rev. **2**(3), 53–56 (2004)
17. Nitta, T.: Local minima in hierarchical structures of complex-valued neural networks. Neural Netw. **43**, 1–7 (2013)
18. Satoh, S., Nakano, R.: Fast and stable learning utilizing singular regions of multilayer perceptron. Neural Process. Lett. **38**(2), 99–115 (2013)
19. Satoh, S., Nakano, R.: Multilayer perceptron learning utilizing singular regions and search pruning. In: World Congress on Engineering and Computer Science, vol. 2 (2013)
20. Satoh, S., Nakano, R.: Complex-valued multilayer perceptron search utilizing singular regions of complex-valued parameter space. In: Wermter, S., et al. (eds.) ICANN 2014. LNCS, vol. 8681, pp. 315–322. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11179-7_40

21. Satoh, S., Nakano, R.: Complex-valued multilayer perceptron learning using singular regions and search pruning. In: International Joint Conference on Neural Networks, pp. 1–6. IEEE (2015)
22. Satoh, S., Nakano, R.: A new method for learning RBF networks by utilizing singular regions. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2018. LNCS (LNAI), vol. 10841, pp. 214–225. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91253-0_21
23. Şeker, S., Ayaz, E., Türkcan, E.: Elman's recurrent neural network applications to condition monitoring in nuclear power plant and rotating machinery. Eng. Appl. Artif. Intell. **16**(7–8), 647–656 (2003)
24. Sussmann, H.J.: Uniqueness of the weights for minimal feedforward nets with a given input-output map. Neural Netw. **5**(4), 589–593 (1992)
25. Tallec, C., Ollivier, Y.: Unbiasing truncated backpropagation through time. arXiv preprint arXiv:1705.08209 (2017)
26. Wang, J., Wang, J., Fang, W., Niu, H.: Financial time series prediction using elman recurrent random neural networks. Comput. Intell. Neurosci. **2016** (2016)
27. Williams, R.J., Peng, J.: An efficient gradient-based algorithm for on-line training of recurrent network trajectories. Neural Comput. **2**(4), 490–501 (1990)