VARIATIONS OF TRAINING PROCESS IN VANILLA RECURRENT NEURAL NETWORK FRAMEWORK

D. Yi[*], *I. Kim*[†], *S. Bu*[‡]

Abstract: Recurrent neural networks (RNNs) are capable of learning features and long term dependencies from sequential and time series data and show outstanding performance in sequential modeling tasks. However, training process in RNNs is troubled by issues in learning processes such as slow inference, vanishing gradients and difficulties in capturing long term dependencies. In this paper, we introduce a new learning technique to update the weight set as we change the input sequence which is shifted by certain amount of time in training process, instead of using a traditional way to calculate one set of the weights and bias in training time series with sequences shifted by certain amount of time series. We also consider an algorithm for an evaluation process. In the traditional way, the evaluation process is executed by using final weights and biases calculated in the training process. Instead, during the testing process, the weights and biases are iteratively updated in each sequence as done in the training process. Several numerical experiments demonstrate the efficiency of the proposed techniques.

Key words: *recurrent neural network, time series, machine learning, weight*

Received: April 17, 2019 **DOI:** 10.14311/NNW.2024.34.005
Revised and accepted: April 22, 2024

1. Introduction

Recurrent neural network (RNN) has become the standard approach for machine learning tasks involving sequential time series data. Such success has been enabled by the appearance of larger datasets, more powerful computing resources and improved architectures and training algorithms. The RNNs have a stack of non-linear units where at least one connection between units form a directed cycle. Also RNN is made from layers of connected units called neurons. As the number of layers increases, the network becomes more complex. Many number of layers or recurrent connections are increasing the depth of the network referred to as "deep learning", and it enables to represent more complex models. However, the complex models

[*]Dokkyun Yi; DU University College, Daegu University, Kyungsan, Republic of Korea, E-mail: dkyi@daegu.ac.kr,

[†]Inmi Kim; Mechatronics research center, Hongik University, Sejong, Republic of Korea, E-mail: inmikim@gmail.com

[‡]Sunyoung Bu – Corresponding author; Department of liberal arts, Hongik University, Sejong, Republic of Korea, E-mail: syboo@hongik.ac.kr

based on the many layers may produce many difficulties induced from abstract representation of data at the higher layer, unwanted variabilities and complicated back-propagation in the optimization process [4, 8, 9, 11, 14, 16, 17]. Note that to avoid the unnecessary discussion of these complexities and difficulties, we restrict our attention in this paper to a discrete time series with a single size as an input data.

An efficient training is a major issue in RNN and there have been several efforts to improve the efficiency of a training process in RNN, such as choosing a proper initialization, selecting optimization algorithm, etc. These efforts are for finding more appropriate weights and biases to minimize the training cost. In other aspects, there are some difficulties on learning for long-term dependencies with gradient descent. Previous researches [1, 13] show that calculated parameters are only suitable for sub-optimal solutions that take into account only short-term dependencies but not long term on. Also, current back-propagations are not sufficiently powerful enough to follow the whole behavior of long term time series data. Several variations such as the long short-term memory (LSTM) [5,7] and the gated recurrent unit (GRU) [2], etc, were designed to deal with the vanishing gradients problems with long term dependencies, occurring in basic RNNs. These variation models [3,6,10,12,15] have been widely used due to more efficient results in a variety of tasks in machine translations, language modelings and speech recognitions, etc.

A purpose of this paper is to modify the basic RNN architectures to resolve the learning of long-term dependencies by giving an alternative technique to find the weights and biases in the basic vanilla RNN. In the traditional RNN, the weights and biases are calculated at once in the whole training process. Instead of calculating once for one whole time sequence, first we chopped the whole sequence into several subsequence using an appropriate sequence length. A new technique is to find the weights and biases in each subsequence with having the previous weights and biases calculated in the previous subsequence during the training process.

For the testing process to evaluate the trained model on samples it has not seen during training in order to evaluate its ability to generalize, the test data set is the last part of the given time series. The testing process is executed to predict a next value after the training data set, with having the weights and biases calculated from the training process. Also, for predicting a sequence of test data set, the training and evaluation processes are alternatively executed. That is, one new predicted value obtained from each testing iteration is added as the last value of training data set so it can make a new training data set. The new training data set is trained again and calculated newly weights and biases. The testing process predicts the next value and the value becomes the last value of training data set again. These processes are iteratively repeated until a sequence of test data set is fully predicted.

The paper is organized as follows. In Section 2.1, we briefly review RNN and its properties and in Section 2.2, a new technique for RNN is introduced. Especially, in Section 2.2, a new training process is described and its algorithm is introduced. Additionally, evaluation algorithms used in this work are also described in Section 2.3. Several numerical results are presented in Section 3 to show the efficiency of the proposed techniques. Finally, in Section 4, a summary of the proposed algorithms and some discussions are given.

2. Vanilla RNN and Modified RNN

RNN is a class of neural network where connections between units form a direct cycle along a given sequence. This connection can show a dynamic temporal behavior for a time series. In this section, we briefly explain the basic vanilla RNN and introduce a modification of the basic RNN.

2.1 Vanilla RNN

The basic model of the RNN can be written as follows:

$$\begin{aligned} h_t &= \sigma(w_{hh}h_{t-1} + w_{xh}x_t + b_h), \\ y_t &= w_{hy}h_t + b_y, \end{aligned} \tag{1}$$

where $w_{hh}, w_{xh}, w_{hy}, b_h$ and b_y are weights and biases and σ denotes a sigmoidal function $\tanh(\cdot)$. In RNN, it uses one set of weights and biases, $w_{hh}, w_{xh}, w_{hy}, b_h$ and b_y and many sequences created by shifting certain length of sequence by one. That is, we use a sequence $(x_1, x_2, \ldots, x_{seq})$ to estimate x_{seq+1} and then use $(x_2, x_3, \ldots, x_{seq+1})$ to estimate x_{seq+2} and so on. Here seq is the sequence length. With size N_t of the training set, we have $N_t - seq$ number of sequences to put into the RNN system. In the basic RNN, the *cost* is calculated by the average of squares of differences between the result y and the data x from seqth element, as follows:

$$cost = \frac{1}{m}\sum_{i=1}^{m}\left[y_{seq+i-1} - x_{seq+i}\right]^2, \tag{2}$$

where $m = N_t - seq$. The *cost* formula is used to do learning and get weights and biases $w_{hh}, w_{xh}, w_{hy}, b_h$ and b_y by an appropriate optimizing process [14] such as Adam optimization method [8]. Note that we only consider full back propagation in the Adam optimization. That is, in the partial derivative of *cost* with respect to each variables w_{hh} and w_{xh}, it goes all the way down to the first input data and uses them all as described below. Based on Eqs. (1) and (2),

$$\begin{aligned} \frac{\partial}{\partial w_{hh}}cost &= \frac{\partial}{\partial w_{hh}}\left(\frac{1}{m}\sum_{i=1}^{m}\left[y_{seq+i-1} - x_{seq+i}\right]^2\right) \\ &= \frac{\partial}{\partial w_{hh}}\left(\frac{1}{m}\sum_{i=1}^{m}\left[w_{hy}h_{seq+i-1} + b_y - x_{seq+i}\right]^2\right) \\ &= \frac{\partial}{\partial w_{hh}}\left(\frac{1}{m}\sum_{i=1}^{m}[w_{hy}\sigma(w_{hh}h_{seq+i-2} + w_{xh}x_{seq+i-1} + b_h)\right. \\ &\quad \left.+b_y - x_{seq+i}]^2\right) \\ &= \frac{2}{m}\sum_{i=1}^{m}\left[w_{hy}\sigma(\cdot) + b_y - x_{seq+i}\right]w_{hy}\frac{\partial}{\partial w_{hh}}\sigma(\cdot), \end{aligned} \tag{3}$$

where $\sigma(\cdot) = \sigma(w_{hh}h_{seq+i-2} + w_{xh}x_{seq+i-1} + b_h)$ and $m = N_t - seq$.

By the first equation in Eq. (1), $\sigma(\cdot) = \sigma(w_{hh}h_{seq+i-2} + w_{xh}x_{seq+i-1} + b_h) = h_{seq+i-1}$, so Eq. (3) becomes

$$\begin{aligned}\frac{\partial}{\partial w_{hh}}cost &= \frac{2}{m}\sum_{i=1}^{m}\left[w_{hy}\sigma(\cdot) + b_y - x_{seq+i}\right]w_{hy}\frac{\partial}{\partial w_{hh}}\sigma(\cdot)\\ &= \frac{2}{m}\sum_{i=1}^{m}\left[w_{hy}\sigma(\cdot) + b_y - x_{seq+i}\right]w_{hy}\frac{\partial}{\partial w_{hh}}h_{seq+i-1}.\end{aligned} \tag{4}$$

For detail derivation of $\frac{\partial}{\partial w_{hh}}h_{seq+i-1}$in Eq. (4), we need the following calculations:

$$\begin{aligned}\frac{\partial}{\partial w_{hh}}h_{seq+i-1} &= \frac{\partial}{\partial w_{hh}}\sigma(w_{hh}h_{seq+i-2} + w_{xh}x_{seq+i-1} + b_h)\\ &= \left(1 - h_{seq+i-1}^2\right)\left(h_{seq+i-2} + w_{hh}\frac{\partial}{\partial w_{hh}}h_{seq+i-2}\right)\\ &\vdots\\ \frac{\partial}{\partial w_{hh}}h_1 &= \frac{\partial}{\partial w_{hh}}\sigma(w_{hh}h_0 + w_{xh}x_1 + b_h)\\ &= \left(1 - h_1^2\right)\left(h_0 + w_{hh}\frac{\partial h_0}{\partial w_{hh}}\right).\end{aligned} \tag{5}$$

Similar calculation can be done for $\frac{\partial}{\partial w_{xh}}cost$. Note that in both partial derivatives $\frac{\partial}{\partial w_{hh}}cost$ and $\frac{\partial}{\partial w_{xh}}cost$, every $\frac{\partial h_t}{\partial w_{hh}}$ and $\frac{\partial h_t}{\partial w_{xh}}$ for $t = 0, \ldots, N_t - 2$ are used, since it is full back-propagation.

2.2 Modified RNN

In this subsection, we modify the traditional RNN training process by updating weights and biases in each subsequence step based on the previous weights and biases calculated in the previous subsequence. Additionally, for the testing process, we present an evaluation technique to update the weights and biases calculated in every processes. Algorithms for all new schemes are provided.

The modified RNN method considering the *cost* in each sequence has a learning process to obtain optimized weights and bias in the sequence. After learning the sequence of data, the weight and bias are tossed as the initial value of the next sequence so that the next RNN system can use a better initial condition than the basic RNN. Also, to avoid over-fitting issues, all calculated parameters are saved and used for calculations at the next subsequence. In the modified RNN, the *cost* is calculated in each sequence similar to Eq. 2 as follows:

$$cost(j) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i(j) - x_{i+1}(j)\right)^2, \tag{6}$$

where $x_{n+1}(j) = y_n(j)$ and $n = seq$, the length of each sequence. Then the cost is calculated to get optimized weights and biases by Adam optimization. Note that

in this training, only s-length of back propagation is used. For example, for jth sequence $(x_j, x_{j+1}, \ldots, x_{j+n-1}) = (x_1(j), x_2(j), \ldots, x_n(j))$,

$$\begin{aligned}
\frac{\partial}{\partial w_{hh}} cost &= \frac{2}{n} \sum_{i=1}^{n} \big(y_i(j) - x_{i+1}(j)\big) \frac{\partial}{\partial w_{hh}} h_i w_{hy}, \\
\frac{\partial}{\partial w_{hh}} h_i &= \big(1 - h_i^2\big) \left(h_{i-1} + w_{hh} \frac{\partial}{\partial w_{hh}} h_{i-1} \right), \\
&\vdots \\
\frac{\partial}{\partial w_{hh}} h_1 &= \big(1 - h_1^2\big) \left(h_0 + w_{hh} \frac{\partial}{\partial w_{hh}} h_0 \right),
\end{aligned} \tag{7}$$

where $h_i = \sigma(w_{hh} h_{i-1} + w_{xh} x_i + b_h)$ and $h_0 = h_1(j-1)$ for $j > 1$. Note that initial h_0 in jth sequence is called from the first h, which is h_1 in the previous sequence and it is called as constant. Unlike the basic RNN which uses the full back propagation, the modified RNN uses only sequence length-amount back propagations.

Based on above calculations, we obtain algorithm.

Algorithm 1 Algorithm for modified vanilla RNN.

1. Remark: The algorithm is designed to modify vanilla RNN.
Input: data with having a N_d size vector, training set length N_t, sequence length *seq*, iteration number of optimising process *times*
Output: weight set $(w_{hh}, w_{xh}, w_{hy}, b_h, b_y)$ of basic RNN after learning, the last h_{train} for testing the next element.
2. Take the training data with length N_t by chopping the data from the first element.
3. Take the first data sequence of length *seq* from the first element of training data and calculate the *seq*th element of h_t $(=h_{seq})$ by computing Eq. 1.
4. Perform the Adam optimization scheme with appropriate initial conditions and given iteration number *times*. Note that in this learning scheme, the object function is the cost function defined as

$$cost(w_{hh}, w_{xh}, w_{hy}, b_h, b_y) = \frac{1}{seq} \sum_{i=1}^{seq} \big(y_i(j) - x_{i+1}(j)\big)^2,$$

where $x_{seq+1}(j) = x_1(j+1)$. Here, $x_i(j)$ means ith element of jth sequence data.
5. Calculate h_t and y_t for all indices t in current sequence in Eq. 1 with the result of step 4.
6. Take the next sequence data taken by shifting by one and use the result of step 5 as the initial conditions of weights and biases.
7. Repeat step 4, 5, and 6 until all the training data is used.
8. As changing an object sequence of data, repeat step 4 and 5 until use up all the training data.

2.3 Evaluation of Training Process

In this subsection, we present an algorithm for the testing process with the weights and biases calculated in the training process. In the traditional evaluation process, the test data set is only used for comparison with the expectations. For the traditional testing method, the cost is defined as follows:

$$cost = \frac{1}{n}\sum_{i=1}^{n-1}\big(x_{i+1} - y_i\big)^2, \tag{8}$$

where

$$\begin{aligned} y_i &= w_{hy}h_i + b_y, \\ h_i &= \sigma(w_{hh}h_{i-1} + w_{xh}y_{i-1} + b_h). \end{aligned} \tag{9}$$

Note that y_{i-1} is the calculated result from x_{i-1} which targets x_i.

The algorithm for testing process used in this paper is a process to measure the difference between real data and expectation by iteratively updating the weights and biases during the testing process. For an evaluation of a training algorithm, first we predict an expected value using weights and biases calculated in the training process. After that, the expected value is added to the original training data. New weights and biases are calculated for the updated training data. Using the new weights and biases, we predict the next value, again. Note that the new weights and biases are used for the initial condition for the next input training data. By repeating this process, we have the expected evaluation data, and compare them with testing data.

Based on the above calculations and explanations, we present the following algorithm.

Algorithm 2 Evaluation algorithm for modified vanilla RNN.

1. Remark: The algorithm is designed to evaluate the proposed vanilla RNN.
2. **Input:** data with having a N_d size vector, training set length N_t, sequence length *seq*, weight set $(w_{hh}, w_{xh}, w_{hy}, b_h, b_y)$ and the last h from learning process, iteration number *times* for optimization.
Output: cost of testing data from given weights and biases and expected data
3. Take the testing data with size $N_d - N_t$ by extracting the training data from the whole data.
4. Get data of $(seq - 1)$ amount before the testing data, and take an expected value from the training process
5. Initiate $h_0 = h$.
6. From the first sequence of data in step 4, perform the Adams optimization scheme with appropriate initial conditions and given iteration number *times* with the object function

$$cost_j(w_{hh}, w_{xh}, w_{hy}, b_h, b_y) = \frac{1}{seq}\sum_{i=1}^{seq}\big(y_i(j) - x_{i+1}(j)\big)^2,$$

where $x_{seq+1}(j) = x_1(j+1)$. Here, $x_i(j)$ means ith element of jth sequence data.

7. Calculate h_t and y_t in Eq. 1 with the result of step 6 then calculate the expected data $y_{seq}(j)$, the seqth element of y_t in current sequence.
8. Take the next sequence of data by shifting by one and use the result of step 7 as the initial conditions of weights and biases. Here, the last element of the sequence would be $y_{seq}(j)$.
9. Repeat step 6 and 7 until the expected data length is the same length as the testing data.

3. Numerical Results

In this section, preliminary numerical results are presented to examine the efficiency of the proposed scheme, compared to the basic RNN. For this experiment, there are tests – stock data, virtual currency data and precipitation data in Korea. Although these data has own units and values, these data should be rescaled from 0 to 1 by own maximum and minimum values in each data set.

For the experiments, the initial conditions for all cases are given as follows $h_0(1) = 0, (w_{hh}, w_{xh}, w_{hy}, b_h, b_y) = (1, 1, 1, 0, 0)$. As mentioned above, the sigmoidal function is set to a hyperbolic tangent tanh function.

3.1 Stock Data

For the first example, we test the proposed scheme on a time series data obtained from a stock market. The data is composed of 744 values over time and divided into two parts – one for training and the other for evaluation. The training data length is set as 700 for convenience and the remaining 44 is automatically for the testing. Here, a sequence length of 50 is used as default.

To show the efficiency of the proposed scheme, we calculate the cost used for the learning process of the proposed scheme and compare it with that of the basic RNN. The result is plotted in Fig. 1 by varying the iteration numbers from 1 to 3000, used for optimization. Fig. 1 shows that training cost of the basic RNN is decreasing slower than the cost of the proposed scheme and the final training cost is also greater than the proposed scheme.

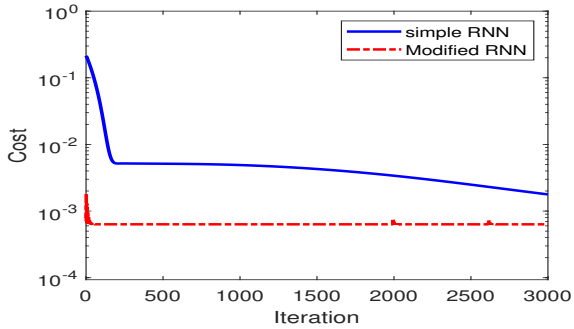

Fig. 1 *Training error.*

After the learning process, we need to check if the learning process is working reasonably by testing the learning results – weights and biases. Therefore, to investigate the testing process, we plug the weights and biases obtained from the last learning process back into Eq. 9 and examine how the results approach the behaviors of real data. All conditions are the exact same as those used above. First, we calculate a cost of a testing scheme and compare it to that of the real testing data. All results are plotted in Fig. 2. As shown in Fig. 2, the expectation of the proposed scheme result resembles a similar pattern to the real data, compared with the results from the basic RNN. However, both results from the proposed and basic RNN are insufficient in catching the details of the real data.

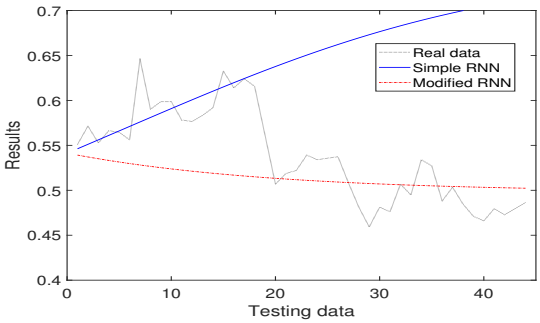

Fig. 2 *Comparison of traditional testing results of proposed RNN and basic RNN.*

To avoid this drawback, we apply the proposed testing scheme to the proposed RNN and compare it with that of the given real data set. For further comparison, we also calculate the traditional testing scheme for the proposed RNN. All results are plotted in Fig. 3. One can see that the proposed testing scheme follows the pattern of the given data set, while the traditional testing scheme seems to generate a linear pattern to fit the given real data.

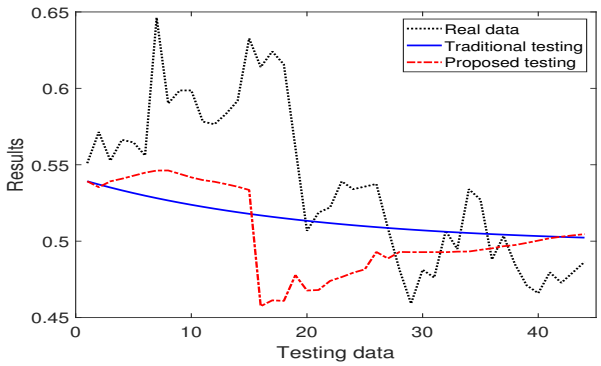

Fig. 3 *Comparison of traditional testing scheme with proposed testing scheme.*

To emphasize the effectiveness of the proposed testing process, we apply the testing process into both the basic RNN and the proposed RNN to see the effect of

the proposed testing process according to testing methods. Fig. 4 shows that the difference of real testing data and expectation data constructed by the proposed scheme is much smaller than the difference from the basic RNN result.

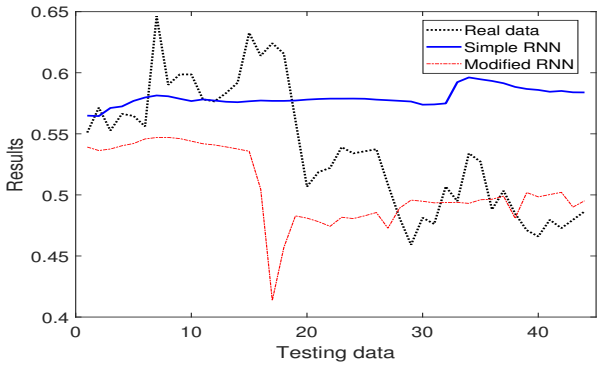

Fig. 4 *Comparison of proposed testing result of proposed RNN and basic RNN.*

Note that there are several flexible factors such as sequence length, iteration numbers to control the optimising module, an appropriate portion of training data and testing data among the given data set, etc. To investigate the effect of the factors, we calculate the learning costs depending on the sequence length and plot the results in Fig. 5(a). For further investigation, we plot the testing error in Fig. 5(b). In Fig. 5(a), we can see that there are suitable sequence lengths for a given data set. Additionally, through the comparison of Fig. 5(a) with Fig. 5(b), one can see that the testing error also becomes quite high when we use unfavorable sequence length which makes learning cost relatively high. Hence, from Fig. 5, one can conclude that the most appropriate sequence length can be found depending on the given data set, and the bigger the learning cost is, the higher the testing error is.

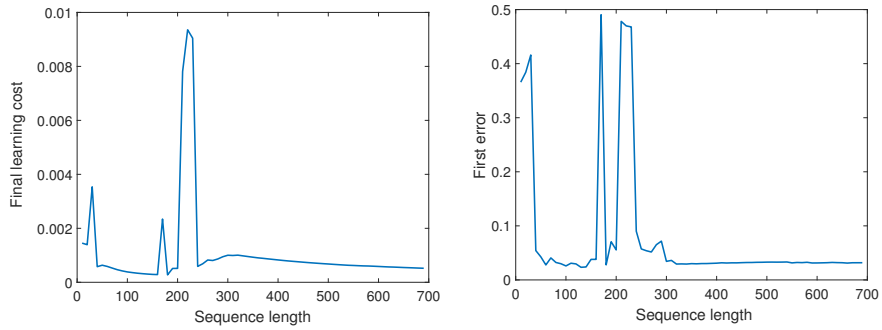

Fig. 5 *(a) Final learning cost per sequence length and (b) testing error per sequence length.*

In addition, to examine the relation between iteration numbers of the optimiser and expectation ability, we calculate the learning cost and testing error according to the iteration number of the optimiser and plot the results in Fig. 6. Fig. 6(a) shows that using small iteration numbers results in high training costs. Also, after using a certain amount of iteration numbers, the learning cost is stably reduced. Similarly, in Fig. 6(b), one can see that testing error is also decreased after using a sufficient number of iterations, although just a few exceptional iteration numbers are observed to make the testing error high.

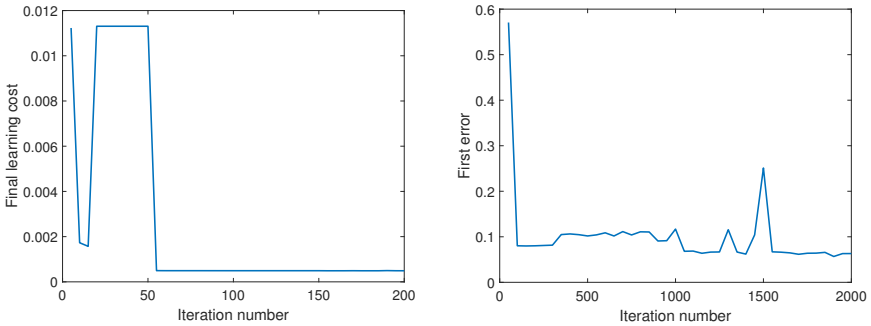

Fig. 6 *(a) Final learning cost per iteration number and (b) testing error per sequence length.*

To examine the efficiency of the method combining proposed modified RNN and proposed testing schemes, we compare it with the simple RNN with traditional testing scheme, GRU and LSTM in Fig. 7. Time series data set obtained from Korea Composite Stock Price Index (Kospi) during last 20 years is composed of 4146 data, and 4140 data and 6 data are used for training set and testing, respectively. Iteration number is set to 5000 with 10 sequence length for all schemes.

Fig. 7 shows that the proposed scheme has good performance compared with the existing schemes. Therefore, one can conclude that either the proposed training

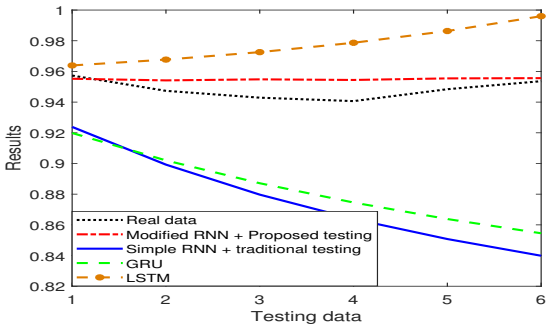

Fig. 7 *Comparing proposed scheme with simple RNN and GRU.*

scheme or the proposed testing scheme can increase the efficiency of the original technique. Therefore, by applying this idea to existing schemes such as LSTM or GRU, we infer that better performance is likely to be obtained.

3.2 Virtual Currency Data

As the second example, we apply the proposed scheme to a time series data obtained from altcoin (Ripple) price fluctuation which was measured from March, 2017 to March, 2018. Each data point is the ripple price at noon of each day. For the experiment, 330 data and 25 data are used for training and testing process, respectively. Also the sequence length is set to 40.

First, the training cost is observed by varying iteration numbers used in optimization module from 1 to 10000. The results are plotted in Fig. 8. Unlike the

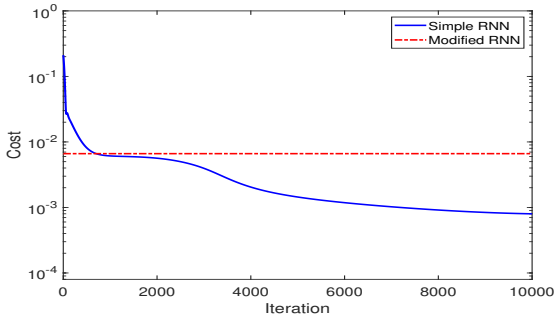

Fig. 8 *Training error for virtual currency data.*

results of other experiments, the cost of the proposed RNN is not decreasing as iteration numbers are increasing, while the cost of the basic RNN is quite reduced.

To show the comparison of training schemes, we plot the results of testing process generated by the basic RNN and the proposed RNN and compare them with the given real data set in Fig. 9.

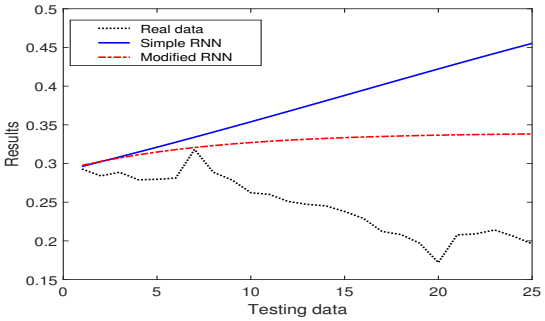

Fig. 9 *Comparison of testing result of proposed RNN and basic RNN for virtual currency data.*

One can see that both schemes generate quite accurate results for the first testing data point. However, both are not sufficient to catch the given data set, but the proposed algorithm is much closer to the real data set.

To improve the results, we apply the proposed testing scheme to the proposed RNN and compare it with the traditional testing scheme in the proposed RNN framework. The results are plotted in Fig. 10(a). It shows that proposed testing scheme can generate quite accurate results and catch the pattern of the give real data. For a further investigation of the proposed testing scheme, the proposed scheme is applied to both the basic RNN and the proposed RNN. Fig. 10(b) shows the results of effectiveness for the proposed testing scheme. One can see that using the proposed scheme can catch the pattern of the real data for both training schemes. Therefore it can be concluded that the proposed testing scheme is quite efficient in catching the pattern of the given real data.

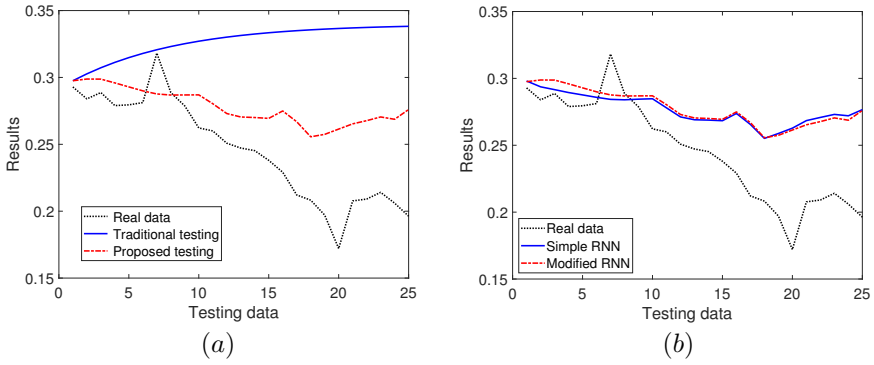

Fig. 10 *(a) Comparison testing results of proposed testing scheme and traditional testing scheme in proposed training framework and (b) comparison of proposed testing result of proposed RNN and basic RNN for virtual currency.*

3.3 Precipitation

As the last example, we apply the proposed scheme to a time series data obtained from the precipitation of Seoul, the capital of Republic of Korea, during the last 10 years. Each data represents the precipitation of each month. Similarly, we calculate the learning cost and plot it in Fig. 11 by varying the iteration numbers from 1 to 10000. For this experiment, of among 110 data, 100 data are used for the training test. Since data is related to the month, the sequence length is set to 12.

Fig. 11 shows that the proposed scheme is much efficient than the basic RNN.

To examine the effectiveness of the calculated weights and biases from the testing process, the testing results using the same parameters and conditions above are calculated and plotted in Fig. 12. It shows that the results of the proposed scheme are much closer to the real data, while those of the basic RNN have totally different pattern, compared with the given real data. However, the results from the proposed RNN cannot catch the peak of the given data.

To improve the results, the proposed testing scheme is applied to the proposed RNN and compared to the traditional testing scheme in the proposed RNN. The

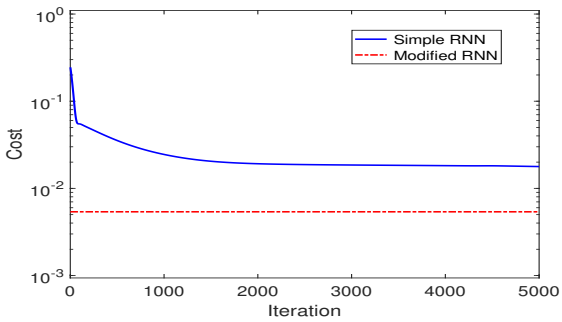

Fig. 11 *Training error for precipitation data.*

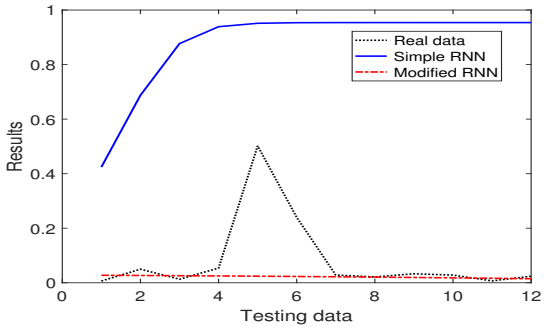

Fig. 12 *Comparison testing result of proposed RNN and basic RNN for precipitation data.*

results are plotted in Fig. 13(a). One can see that the proposed testing algorithm can catch the pattern of the given real data and provide more accurate results, compared to the traditional testing scheme in the proposed RNN framework.

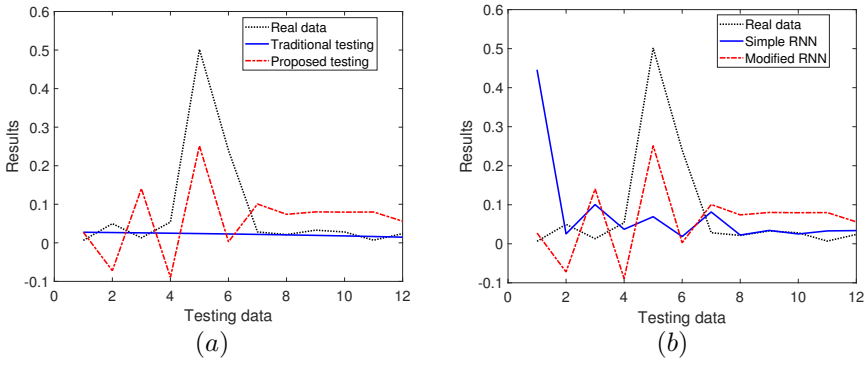

Fig. 13 *(a) Comparison testing results of proposed testing scheme and traditional testing scheme in proposed training framework and (b) comparison of proposed testing result of proposed RNN and basic RNN for precipitation data.*

To show the effect of the testing scheme, we apply the proposed scheme to the basic RNN and plot the results in Fig. 13(b). The figure shows that the testing scheme in the basic RNN also tries to catch the pattern of the real data but is less accurate, compared with the result of the proposed RNN. Therefore, it can be concluded that the proposed testing algorithm in the proposed RNN is superior to the traditional methods.

4. Conclusion

New variations of learning and testing techniques are introduced in the RNN framework. Unlike the basic RNN to calculate the weights and biases only once for the whole sequence, the new technique updates the weights and biases in each subsequence by using the previous weights and biases calculated in previous sequence. Also, for the testing process, we consider a traditional testing algorithm and introduce another variation of the testing algorithm. Traditionally the testing scheme is to test using weights and biases obtained from the training process, while the proposed testing scheme is to test by updating weights and biases in each test data set based on the previous weights biases calculated from the previous test data set. Also we examine several factors effecting the efficiency of the proposed scheme, such as sequence length and iteration number of the used optimiser.

Throughout the several numerical experiments, it is shown that the proposed learning scheme is more efficient than the basic RNN learning process. Also, the proposed testing process is much better than the traditional testing process in that the proposed testing process iteratively calculates the weights and biases fitted to the given data. Additionally, it is seen that depending on the sequence lengths and iteration numbers, the learning error and testing error are changed. Therefore, the most favorable sequence length and iteration numbers suited to the given data set can be found.

Acknowledgement

This work was supported by basic science research program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number NRF-2017R1E1A1A03070311). The corresponding author Bu was supported by basic science research program through the National Research Foundation of Korea (NRF) funded by the Korea government (MSIT) (grant number NRF-2022R1A2C1004588).

References

[1] BENGIO Y., SIMARD P., FRASCONI P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*. 1994, 5(2), pp. 157–166, doi: 10.1109/72.279181.

[2] CHO K., MERRIENBOER B.V., GULCEHRE C., BAHDANAU D., BOUGARES F., SCHWENK H., BENGIO Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014, arXiv:1406.1078

[3] CHO K., MERRIENBOER B.V., BAHDANAU D., BENGIO Y. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *arXiv:1409.1259*, 2014.

[4] ELMAN J.L. Finding structure in time *Cognitive Science*. 1990, 14(2), pp. 179–211, doi: 10.1016/0364-0213(90)90002-E.

[5] GERS F.A., SCHRAUDOLPH N.N., SCHMIDHUBER J. Learning Precise Timing with LSTM Recurrent Networks, *The Journal of Machine Learning Research*, 2002, 3, pp. 115–143.

[6] GRAVES A., SCHMIDHUBER J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures, *Neural Networks*, 2005, 18(5), pp. 602–610, doi: 10.1016/j.neunet.2005.06.042.

[7] HOCHREITER S., SCHMIDHUBER J. Long Short-Term Memory, *Neural Computation*. 1997, 9(8), pp. 1735–1780, doi: 10.1162/neco.1997.9.8.1735.

[8] KINGMA D.P., BA Adam J.L. A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference for Learning Representations, ICLR*, San Diego, USA, 2015.

[9] PASCANU R., MIKOLOV T., BENGIO Y. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, Atlanta, GA, USA, 2013, pp. 1310–1318.

[10] ROHWER R. The moving targets training algorithm. *Advances in neural information processing systems 2*. Touretzky, Ed. San Matteo, CA: Morgan Kaufmann, 1990, pp. 558–565

[11] RUMELHART D.E., HINTON G.E., WILLIAMS R.J. Learning representations by back-propagating errors. *Nature*, 1986, 323, pp. 533–536, doi: 10.1038/323533a0.

[12] SCHMIDHUBER J. A Local Learning Algorithm for Dynamic Feedforward and Recurrent Networks, *Connection Science*, 1989, 1(4), pp. 403–412, doi: 10.1080/09540098908915650.

[13] SCHMIDHUBER J. A Fixed Size Storage O(n3) Time Complexity Learning Algorithm for Fully Recurrent Continually Running Networks, *Neural Computation*, 1992, 4(2), pp. 243–248, doi: 10.1162/neco.1992.4.2.243.

[14] WAN E.A., BEAUFAYS F. Diagrammatic derivation of gradient algorithms for neural networks, *Neural Computation*, 1996, 8, pp. 182–201, doi: 10.1162/neco.1996.8.1.182

[15] WERBOS P.J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*. 1988, 1(4), pp. 339–356, doi: 10.1016/0893-6080(88)90007-X.

[16] YI D., BU S., KIM I. An Enhanced Algorithm of RNN Using Trend in Time-Series, *Symmetry*, 2019, 11(7), 912, doi: 10.3390/sym11070912

[17] YI D., JI S., BU S. An Enhanced Optimization Scheme Based on Gradient Descent Methods for Machine Learning. *Symmetry*, 2019, 11(7), 942, doi: 10.3390/sym11070942.