Simplification of RNN and Its Performance Evaluation in Machine Translation*

Tomohiro FUI Johnston†, Zhiwei LUO†, Changqin QUAN† and Koheii MORI†

In this paper, we study on simplification of RNN and propose new structures which enable faster learning to improve the performance while reducing the number of learning parameters. We construct 4 types of RNNs with new gated structures and call these new RNNs “SGR (Simple Gated RNN)”. SGR have two or one gate and weight or no weight for input. Comparison studies are performed to verify the effectiveness of our proposal. As a result of machine translation in relatively small corpus, compared with LSTM and GRU, our proposed SGR can realize higher scores than LSTM and GRU. Furthermore, SGR can realize faster learning approximately 1.7 times than GRU. However, with the increase of learning layers and weights for input, the learning scores of SGR seems not increase as much as we expected, which should be studied in our future work. It is necessary to analyze in more detail a performance in larger dataset and a performance difference due to multi-layering, weight for input and the number of gates.

1. Introduction

This paper studies on simplification of RNN. RNN (Recurrent Neural Network), which is a type of neural network, is used to process time series data as well as text information [1].

RNN has a recursive structure inside to process time series data. For Simple RNN [2] which is a basic structure among RNNs, it is difficult to learn long-term time series data due to the vanishing gradient problem, etc. Therefore, Long Short-Term Memory (LSTM) [3,4] or Gated Recurrent Unit (GRU) [5], which control information by the gate structure, were proposed.

In order to train neural networks, a large amount of training data is usually required. Additionally, RNNs cannot be parallelized because they have an internal recursive structure. As a result, learning of RNNs requires a lot of time practically. Conventional neural networks such as LSTM and GRU have the problem that their learning is slower than Simple RNN due to their complicated structure and the increase of weight parameters. Also, the memory usage due to the increase of weight parameters becomes a problem.

Therefore, in this study, we propose a new Simple Gated RNN (SGR) that can learn at high speed and require a small amount of memory, by reducing the number of parameters and in turn the amount of calculation while keeping or even improving processing performance.

As experiments, we study 4 types of SGR and carry out performance evaluation experiment of machine translation from English to Japanese using “tanaka corpus” [6]. We also compare the performance with LSTM and GRU and verify the effectiveness of our proposal. As a result, it was found that some of our SGR outperforms the performance of LSTM and GRU in the machine translation in relatively small corpus with less weight parameters.

Previous researches also proposed, such as MUT [7], Minimal Gated Unit (MGU) [8], and Simple Recurrent Unit (SRU) [9]. These models attempt to reduce gates from LSTMs and GRUs, however they still have multiple gate calculation. In our study, we test whether the performance can be achieved even no weight for input and only a single gate using linear interpolation.

2. Related Works

2.1 LSTM (Long Short-Term Memory)

Simple RNN has vanishing gradient problem. The vanishing gradient problem is that when computing gradients in back propagation, the error becomes smaller rapidly as it goes back layers and the learning does not progress well. It is LSTM that partially solved this problem [3].

As be shown in Fig. 1, a layer of typical LSTM is defined by
\[ z_t = \tanh(W_x x_t + U_z h_{t-1} + b_z) \] (1)
\[ i_t = \sigma(W_{in} x_t + U_i h_{t-1} + b_i) \] (2)
\[ o_t = \sigma(W_{out} x_t + U_o h_{t-1} + b_o) \] (3)
\[ f_t = \sigma(W_{for} x_t + U_{for} h_{t-1} + b_{for}) \] (4)
\[ c_t = i_t \odot z_t + f_t \odot c_{t-1} \] (5)
\[ h_t = \tanh(c_t) \odot o_t \] (6)

where \( z_t \) is an input to be added to the cell, \( c_t \) is an internal cell state, \( h_t \) is an output at next time step. Also, \( i_t, o_t, f_t \) represent outputs from input gate, output gate, and forget gate, respectively. The operator symbol \( \odot \) represents the Hadamard product, \( W_z, W_{in}, W_{out}, W_{for}, U_z, U_{in}, U_{out}, U_{for} \) are weighting matrices, and \( b_z, b_{in}, b_{out}, b_{for} \) are biases. From here, unless otherwise specified, \( x_t \) is an input at time step \( t \), \( h_{t-1} \) is an output at time step \( t-1 \), and \( h_t \) is an output at time step \( t \).

LSTM solves the vanishing gradient problem by using a concept of CEC (Constant Error Carousel) which is back propagating without multiplying weighting matrices. The part of CEC is called a cell. The input gate controls input to the part of CEC, using the value of \( i_t (0 < i_t < 1) \) which is the output from equation (2) so that it can be selectively stored from the input. Also, the output gate controls the output from the part of CEC using the value of \( o_t (0 < o_t < 1) \) which is the output from equation (3).

The early LSTM had the problem that it could not deal with rapid data changes because the CEC had a linear structure. Therefore, in [4], a forget gate in LSTM is introduced and improved it to initialize internal cell by Gers et al. As with input gate and output gate, the forget gate controls the internal cell by \( f_t (0 < f_t < 1) \) which is the output from equation (4).

### 2.2 GRU (Gated Recurrent Unit)

Since LSTM has three gates, the number of parameters increase and the calculation cost is very high. Therefore, GRU is proposed to reduce the calculation cost.

As be shown in Fig. 2, a layer of GRU is defined by

\[ r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \] (7)
\[ u_t = \sigma(W_u x_t + U_u h_{t-1} + b_u) \] (8)
\[ \tilde{h}_t = \tanh(W_{gr} x_t + U_{gr}(r_t \odot h_{t-1}) + b_{gr}) \] (9)
\[ h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \tilde{h}_t \] (10)

where \( r_t, u_t \) are outputs from reset gate and update gate, \( W_r, W_u, W_{gr}, U_r, U_u, U_{gr} \) are weighting matrices, and \( b_r, b_u, b_{gr} \) are biases.

The reset gate uses gate output \( r_t (0 < r_t < 1) \) to control how much the previous hidden state \( h_{t-1} \) is considered to create a new hidden state \( h_t \). This gate allows to represent required information conceivably because of effective deleting information that has been found to be irrelevant in the future. The update gate uses the gate output \( u_t (0 < u_t < 1) \) to decide how much the new hidden state \( \tilde{h}_t \) is mixed to generate the next hidden state \( h_t \). Controlling the amount of information to be transferred from the previous hidden state \( h_{t-1} \) to the current hidden state helps to store long-term information [5].

GRU’s performance advantages with LSTM depends on learning task, while the required number of weight parameters is 3/4 that of LSTM.

### 2.3 Previous Researches on Simplification of RNN

In order to further reduce the computation cost, there are also many other previous proposes, such as MUT [7], MGU [8] and SRU [9]. The formulac of typical models in these papers are shown below:

For MUT

\[ z_t = \sigma(W_{zx} x_t + b_{zz}) \]
\[ r_t = \sigma(W_{zx} x_t + U_{zh} h_{t-1} + b_{rr}) \]
\[ h_t = \tanh(W_{hh}(r_t \odot h_{t-1}) + \tanh(x_t) + b_{hh} \odot z_t) \]
\[ + h_{t-1} \odot (1 - z_t) \]
3. Simplification of RNN with Gate Structure

To explain the SGR proposed in this study, firstly, we consider a Simple RNN which is a simple RNN with the weight of the recursive part fixed at 1.0 [12]. This Simple RNN can secure a certain level of performance. However, if the weight is fixed at 1.0, there is a problem that flexible processing cannot be performed when the input sequence changes suddenly or when a same time series appears after repeated several times. Since LSTM in [3] has the same problem, in order to solve this problem, a forget gate in [4] was also introduced in CEC of LSTM. Considering this fact, we suppose that performance of a RNN with a single gate using linear interpolation would be comparable to complex models with multiple gates such as LSTM, GRU, or MGU, SGU, and MUT. In our study, we verify whether the performance can be achieved even with a simple model that incorporates a single gate structure into Simple RNN, which has the weight of the recurrent part that is fixed at 1.0. Since the performance may change due to the difference in the gate structure between LSTM and GRU, we will test both gate structures.

The Simple RNN shown in Fig. 3 is defined by

\[ h_t = f(W_e x_t + h_{t-1} + b_e) \]

where \( W_e \) is a weighting matrix, \( b_e \) is bias, \( f \) is a nonlinear function. Here, we compare the gate part of LSTM and GRU with the Simple RNN so as to construct new gated RNN structures incorporating LSTM and GRU gate parts and based on the Simple RNN. In Figs. 1, 2, continuous lines and broken lines represent gate portion of LSTM or GRU. These lines

\[ f_t = \text{sigmoid}(W_f x_t + U_f h_{t-1} + b_f) \]
\[ h_t = \tanh(W_x x_t + U_h (r_t \circ h_{t-1}) + b_h) \]
\[ h_t = (1 - f_t) \circ h_{t-1} + f_t h_t \]

[SRU]

\[ f_t = \text{sigmoid}(W_s x_t + v_s c_{t-1} + b_s) \]
\[ c_t = f_t \circ c_{t-1} + (1 - f_t) \circ (W_x x_t) \]
\[ r_t = \text{sigmoid}(W_x x_t + v_r c_{t-1} + b_r) \]
\[ h_t = r_t \circ c_t + (1 - r_t) \circ x_t \]

where \( W_z, W_x, U_h, W_h, W_f, U_f, W_s, W_x, W_s, W_r, W_x, W_h, W_f, U_f, W_s, W_x, W_s, W_r \) are weighting matrices, \( b_z, b_x, b_h, b_f, b_s, b_r, b_x, b_h, b_f, b_s, b_r \) are biases and \( v_s, v_r \) are parameter vectors. These RNNs have less weights and gates than LSTM and GRU, thus improved memory usage and faster learning. However, they still use multiple gate calculation.

Papers such as [10] and [11] study to reduce the parameters by reducing the weight used for gate calculation, without reducing the number of gates. In this way, the simplification of RNN has been carried out in many studies, and the importance is apparent.

in the LSTM and GRU are inserted into corresponding portion of the Simple RNN in Fig. 3, respectively. We call these new RNN “SGR (Simple Gated RNN)” because they are created from most simple RNN and has gate structure. The outlines and formulas of the 4 types of SGR are as follows:

SG1R: It has one gate and no weight for input \( x_t \).

\[ g_{1t} = \text{sigmoid}(W_{g1} x_t + U_{g1} h_{t-1} + b_{g1}) \]  
\[ h_t = g_{1t} \circ x_t + (1 - g_{1t}) \circ h_{t-1} \]  
\[ a_t = \tanh(h_t) \]

SG1W: It has one gate and weight for input \( x_t \).

\[ g_{1t} = \text{sigmoid}(W_{g1} x_t + U_{g1} h_{t-1} + b_{g1}) \]  
\[ h_t = g_{1t} \circ (W_1 x_t + b_1) + (1 - g_{1t}) \circ h_{t-1} \]  
\[ a_t = \tanh(h_t) \]

SG2R: It has two gates and no weight for input \( x_t \).

\[ g_{1t} = \text{sigmoid}(W_{g1} x_t + U_{g1} h_{t-1} + b_{g1}) \]  
\[ g_{2t} = \text{sigmoid}(W_{g2} x_t + U_{g2} h_{t-1} + b_{g2}) \]  
\[ h_t = g_{1t} \circ x_t + g_{2t} \circ h_{t-1} \]  
\[ a_t = \tanh(h_t) \]

SG2RW: It has two gates and weight for input \( x_t \).

\[ g_{1t} = \text{sigmoid}(W_{g1} x_t + U_{g1} h_{t-1} + b_{g1}) \]  
\[ g_{2t} = \text{sigmoid}(W_{g2} x_t + U_{g2} h_{t-1} + b_{g2}) \]  
\[ h_t = g_{1t} \circ (W_1 x_t + b_1) + g_{2t} \circ h_{t-1} \]  
\[ a_t = \tanh(h_t) \]

where \( g_{1t}, g_{2t} \) are outputs from gates, \( W_1, W_{g1}, W_{g2}, U_{g1}, U_{g2} \) are weighting matrices for input and gates, \( b_1, b_{g1}, b_{g2} \) are biases, and \( a_t \) is the output after applying activation function to hidden state \( h_t \). Here, the subscripts use the same characters only for those with the same function. Fig. 4 shows proposed structures of each SGR.

In order to test which gate structure of LSTM or GRU is better, SG2RW and SG2R are created with
Fig. 4 Four types proposed structure of SGR

reference to LSTM, and on the other hand, SG1RW and SG1R are created with reference to GRU. Also, considering the necessity of weight parameters for input, we construct structures from which the weight for input was removed, because the gate calculation is weighted non-linear transformation. Therefore, SG2RW and SG1RW has weight \( W_1 \) for input \( x_t \), but SG2R and SG1R have no weight for input. The reason why the activation function is inserted after recurrent part is to avoid vanishing gradient problem as same as CEC of LSTM. Simply thinking, it should be assumed that the more weight parameters, the better the performance.

4. Performance Evaluation

4.1 Outline of Performance Evaluation

The experiment of machine translation is performed for performance evaluation. Machine translation is a mixed task that requires the performance of both a (encoder) task that uses only the output of RNN at the last time step as typified by binary classification and a (decoder) task that uses the output of RNN at each time step as typified by text generation. Therefore, we consider that it is an excellent task for performance evaluation.

Here, chainer (version 6.4) which is a machine learning library and Python 3 (version 3.7) is used. The execution environment is windows10 education (64bit) and cuda10.1, and we use i7 8700 (RAM 8GB) and gtx1080 of GPU (VRAM 8GB) for single precision calculations.

4.2 Performance Evaluation in Machine Translation

The machine translation task is to translate from English to Japanese, and the corpus used here is “tanaka corpus” [6]. There are 149786 translation pairs, and both English and Japanese sentences split into words using software libraries (Mecab and NLTK). The number of English words is 25,722, and the number of Japanese words is 31,809. These include punctuation marks and periods. We used 90% as training data, 5% as validation data, and 5% as test data. Also, we used all the words in the corpus and did not limit the number of words.

For machine translation experiments, we use seq2seq model [13], a machine translation model using RNN. The model learns input sentences by word and outputs sentences by word. Seq2seq model has encoder and decoder. Encoder consists of an input embedding layer and Several RNN layers. Decoder consists of an input embedding layer, Several RNN layers, an output layer and a softmax layer. In encoder, the input embedding layer maps each input word from one-hot vector into n-dimensional vector, and RNN layers are stacked by the number of layers. In decoder, the output layer is a feedforward neural network, and the softmax layer computes the probability distribution for the next output word. Here, if \( o_i \) is the output of unit \( i \), the softmax function is computed by \( \text{softmax}(o_i) = \frac{\exp(o_i)}{\sum_{j=1}^{N} \exp(o_j)} \) \( (N: \text{the number of units}) \). As a supplement, the decoder receives the encoder’s mapped hidden state for each corresponding RNN layer, and the output layer receives only the hidden state of the uppermost layer (the layer closest to the output side) of the decoder RNN. The method of word selection is to select the word with the highest probability among the probability of the word outputs by seq2seq model.

In our Seq2seq model used for machine translation, we set input and output dimension to 250 or 500 or 1000 for each RNN layer. The embedding layers and the output layer are set similarly to match the RNN dimensions. Also, the number of layers is changed according to the type of RNN. The algorithm of stochastic gradient descent in learning uses AdamW method [14]. In order to prevent over-fitting, the Dropout rate is 0.2 and the coefficient used for Weight Decay is 0.0001. Here, the dropout is ap-
Table 1 Average BLEU score of each model

| Model                  | Hidden units | 250 units | 500 units | 1000 units |
|------------------------|--------------|-----------|-----------|------------|
|                        | 1 layer      | 2 layers  | 3 layers  | 1 layer    | 2 layers  | 3 layers  | 1 layer    | 2 layers  | 3 layers  |
| Simple RNN with a fixed weight of 1.0 | 14.35±0.12 | 15.45±0.14 | 13.56±0.06 | 16.76±0.09 | 17.46±0.18 | 15.97±0.29 | 16.96±0.09 | 16.42±0.18 | 14.85±0.14 |
| Simple RNN             | 9.46±0.37   | 9.82±0.05 | 8.87±0.31 | 9.91±0.44 | 10.33±0.23 | 9.46±0.16 | 9.31±0.06 | 8.58±0.07 | 7.16±0.14 |
| SG1R                   | 16.73±0.15  | 18.08±0.09 | 17.11±0.09 | 20.68±0.20 | 22.16±0.08 | 21.66±0.08 | 23.04±0.05 | 24.62±0.03 | 24.83±0.04 |
| SG1RW                  | 19.09±0.11  | 18.09±0.09 | 19.77±0.26 | 21.42±0.08 | 22.55±0.02 | 22.71±0.21 | 22.50±0.24 | 22.90±0.15 | 22.29±0.39 |
| SG2R                   | 20.33±0.04  | 20.15±0.03 | 19.65±0.12 | 23.49±1.0 | 23.01±0.52 | 22.55±0.10 | 23.60±0.18 | 24.69±0.07 | 24.34±0.04 |
| SG2RW                  | 20.35±0.02  | 20.44±0.22 | 20.53±0.04 | 22.56±0.28 | 22.60±0.65 | 23.65±0.11 | 23.96±0.02 | 22.66±0.06 | 20.65±0.04 |
| LSTM                   | 16.35±0.36  | 18.00±0.09 | 16.56±0.21 | 19.47±0.05 | 20.42±0.56 | 18.92±0.45 | 20.28±0.34 | 20.99±0.58 | 19.97±0.52 |
| GRU                    | 19.55±0.04  | 20.69±0.12 | 21.79±0.09 | 23.04±0.07 | 22.77±0.06 | 24.67±0.07 | 22.93±0.10 | 23.31±0.31 | 23.44±0.21 |

Table 2 Translation examples

| English                | Japanese                  | Unsuccessful example                                                                 |
|------------------------|---------------------------|--------------------------------------------------------------------------------------|
| The storm had a bad impact on the crops. | は 作 物 に 大 き な 影 響 を 及 ば し た。 | 彼 は 明 日 ヨーロッパ へ 旅 行 に 来 る。 |
| I went to church with him. | 奥 と 一緒 に 出 行 に 行 っ た。 | 彼 女 は ほ こ に ペン を 持 っ て い る。 |
| You've run through all the butter? | バター を 全 品 使 っ て し ま っ た の ？ | 彼 男 で 平 混 な 男 で は な い。 |
| Since it's important, I'll attend to it. | は 重 要 だ か ら 決 が 引 き 受 け ま し ょ う。 | アインシュタイン は 科 学 者 で あ る。 |
| Who is the author of this book? | その 本 の 作 者 は 誰 で す か。 | 男 は 魚 の 味 が い い。 |
| I read the book up to page 80 yesterday. | 私 が 申 し 出 す を 受 け る 気 に は な れ な か っ た。 |

applied only to the input of each layer, but not to the recurrent part. In addition, to prevent exploding gradient, the coefficient of Gradient Clipping is set to 25. The epoch is 25 and the mini-batch size is 256 sequences. In this study, weights are initialized with a Gaussian distribution with an average of 0 and a standard deviation of $\sqrt{1/n}$ (n: number of units), and all biases are initialized to 0 [15]. Since all words including <EOS> are used as vocabularies, the input size is 25,723 and the output size is 31,810, and word embedding size is 250 or 500 or 1000. For learning seq2seq model, truncated BPTT is used. Regarding the memory usage, since truncated BPTT was used, if the total number of tokens in the source English sequence and the target Japanese sequence is $T$, and the number of parameters in model is $n$, the memory consumption is $O(nT)$. In order to prevent "OutOfMemoryError" in 3 layers LSTM and GRU experiments, a softmax cross entropy function, which is implemented with chainer, is set as so not to cache intermediate value in all models. In addition, the batch size for LSTM with 1000 units and 3 layers was set to 128.

We use BLEU as a performance evaluation index [16]. BLEU is the most mainstream evaluation criterion in machine translation and is evaluated based on n-gram matching rate. BLEU score takes a real number from 0 to 1. The closer to 1, the better the translation. The formula of BLEU are shown as follows:

$$BLEU = BP \cdot \exp \left( \sum_{n=1}^{N} W_n \log p_n \right)$$

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases}$$

where $c$ is a length of the candidate translation, and $r$ is an effective reference corpus length. Also, $p_n$ are n-gram precisions using n-grams up to length $N$, and $W_n$ are positive weights summing to 1. We used $N = 4$.
and uniform weights \( w_n = 1/N \).

In the experiment, machine translation is performed while increasing the number of layers for LSTM, GRU, and each SGR, and the performance is evaluated by the BLEU score.

### 4.3 Comparative Result in Machine Translation

Table 1 shows the results of the experiment. Also, Fig. 5 shows a graph of the time taken to complete experiments execution when hidden units of each RNN are 1000. Regarding the number of layers in seq2seq model, the same number of layers is used for both the encoder and the decoder. Table 1 shows the average of 2 times and standard deviation of the BLEU scores (%) for test data at the time when the scores for validation data is the highest in 25 epochs, and the third decimal place is rounded off. The experiment was performed 2 times each in order to take into account the randomness due to the stochastic gradient descent method and the initialization of weights. Table 2 shows examples of translation result with the highest BLEU score.

Fig. 6 shows required weight parameters of RNN increasing with the number of layers in case of 1000 units. As above, the weight parameters of the seq2seq model is twice of that shown in Fig. 6 because the number of layers in encoder and decoder is the same.

The execution speed of SGR cannot be compared correctly with LSTM, because LSTM is a single kernel implementation, and GRU and SGR are implemented by a combination of chainer functions. Therefore, only the execution time of GRU and SGR is shown in Fig. 5 due to implementation problem.

From Fig. 6, if the number of hidden units is the same, the weight parameters is smaller in SGR compared to LSTM and GRU. In SG2R with two gates, the number of weight parameters is 1/2 of LSTM and 2/3 of GRU. Similarly, in SG1R with one gate, the number of weight parameters is 1/4 of LSTM and 1/3 of GRU. Also, from Fig. 5, it can be seen that the execution time decreases as the number of SGR parameters decreases. Here, the reason why the ratio between the number of parameters and the execution time does not match completely is due to overhead such as transposing input sequences and batch learning.

Regarding performance, from Table 1, the best performance is SG1R with 1000 hidden units and three layer each for encoder and decoder. Also, the second performance is SG1R with 1000 hidden units and two layers each for encoder and decoder, but the difference from SG2R with 1000 hidden units and two layers is a little.

### 4.4 Discussion

As a result of the experiment, Table 1 shows that the overall performance of GRU is higher than that of SGR in the case of 250 units with a small number of units. When comparing SGRs, SGRs with weight for input (SG1RW, SG2RW) have higher performance than SGRs with no weight for input (SG1R, SG2R). This is generally considered to be a reasonable result considering the number of weights. In addition, even in the case of 500 units, GRU shows higher performance than SGR except SG2R with one and two layers.

However, comparing the case of 1000 units, the trend changes, and it can be seen that SG1R and SG2R that do not have a weight for the input show the highest performance. As can be seen from Fig. 7, SG1R and SG2R are more resistant to over-fitting than other models, and the performance improved as the number of units increase. Since the weight is updated only through the sigmoid function, we consider that this gentle gradient prevents model from over-fitting, and this is advantage of SG1R and SG2R. Therefore, SG1RW, SG2RW, and GRU models can be used when there are few units relatively, and SG1R and SG2R can be used for large models with many units to improve performance while suppressing over-fitting. As a result, in a larger model, the memory usage proportional to the parameter can be suppressed. In addition, when comparing GRU of 250 units and SG1R of 1000 units, the GRU has fewer total parameters than the SG1R, but the SG1R can calculate faster than the GRU using the GPU. Here, GRU of 250 units took 5200 seconds, and SG1R of 1000 units took 3468 seconds. This involves the parallelism of the model, which is considered to be an advantage of SGR.

With regard to superiority or inferiority depend-
ing on the number of gates, the performance in small weight parameters is higher in the case of 2 gates, however, in larger weight parameters results is the opposite. As the calculation cost and memory usage are about doubled too, therefore, it is difficult to simply determine superiority or inferiority of the number of gates. Further experimentation and verification should be studied.

With regard to the one gate structure, if the input to the gate is \(i_t\), the gate formula can be transformed as follows:

\[
h_t = g_{1t} \circ i_t + (1 - g_{1t}) \circ h_{t-1} = h_{t-1} + g_{1t} \circ (i_t - h_{t-1}).
\]

The reason why SGR was devised is because the gate structure was similar to shortcut connections [17]. The shortcut connections are a concept that has been attracting attention in recent years, and it relieves the vanishing gradient problem by letting a neural network learn residuals. The gate structure can relieve the vanishing gradient problem well, and the output of the gate with respect to \(g_{1t} \circ (i_t - h_{t-1})\) is a weighted non-linear transformation. Therefore, the performance of the single gate structure could be achieved like shortcut connections.

5. Conclusion

In this study, we studied on simplification of RNN structures that can learn at high speed while maintaining the same performance as LSTM and GRU, with a smaller number of parameters and reduction of calculation. As a result of machine translation in relatively small corpus, in certain cases, SGR with no weight for input outperforms LSTM and GRU in the task of machine translation.

The performance degradation due to multi-layering will be a future issue. Furthermore, since the superiority of SGRs performance of one gate or two gates differs according to multi-layering, detailed knowledge on these structures is still unclear. Analysis of larger dataset or other tasks such as image processing will be studied in the future.

References

[1] Y. Tsuboi: shizengoshorinokperudeipuraninguno hatten (Development of deep learning in natural language processing); April issue of Operations Research ɼ Vol. 60, No. 4, pp. 205–211 (2015)
[2] F. J. Pineda: Generalization of back-propagation to recurrent neural networks; Physical Review Letters, Vol. 59, pp. 2229–2232 (1987)
[3] S. Hochreiter and J. Schmidhuber: Long short-term memory; Neural Computation, Vol. 9, No. 8, pp. 1735–1780 (1997)
[4] F. A. Gers, J. Schmidhuber and F. Cummins: Learning to forget; Continual prediction with LSTM; Neural Computation, Vol. 12, No. 10, pp. 2451–2471 (2000)
[5] K. Cho, B. V. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio: Learning phrase representations using rnn encoder-decoder for statistical machine translation; Conference on Empirical Methods in Natural Language Processing, pp. 1724–1734 (2014)
[6] Y. Tanaka: Compilation of a multilingual parallel corpus; Proceedings of PACLING 2001, pp. 265–268 (2001)
[7] R. Jozefowicz, W. Zaremba and I. Sutskever: An empirical exploration of recurrent network architectures; International Conference on Machine Learning, pp. 2342–2350 (2015)
[8] G. B. Zhou, J. X. Wu, C. L. Zhang and Z. H. Zhou: Minimal gated unit for recurrent neural networks; International Journal of Automation and Computing, Vol. 13, No. 3, pp. 226–234 (2016)
Authors

Tomohiro Fujita (Student Member)

Tomohiro Fujita is a doctoral student of Graduate School of System Informatics, Kobe University. He received B. Eng. and Master of System Informatics from Kobe University in 2017 and 2019. His research interests include machine learning and deep learning at artificial intelligence.

Zhiwei Luo (Member)

Zhiwei Luo received his B.Eng. from Huazhong University of Science and Technology in 1984, and M. Eng., Dr. Eng. from Nagoya University in 1991 and 1992, respectively. He worked as an Assistant Professor of Toyohashi Institute of Technology from 1992 to 1994, a Frontier Researcher of The Institute of Physical and Chemical Research (RIKEN) from 1994 to 1999, an Associate Professor of Yamagata University from 1999 to 2001, a Team Leader of The Institute of Physical and Chemical Research (RIKEN) from 2001 to 2008, and a Professor of Kobe University since 2006. His research interests include robotics, system control, artificial intelligence and their applications for health engineering. He is a member of SCI, SICE, RSJ and VRSJ. He is honoured fellow of SICE.

Changqin Quan

Changqin Quan received the Ph.D. degree in 2011 from Faculty of Engineering, University of Tokushima, Japan. She is currently an associate professor in Kobe University. Her research interests include Natural Language Processing, Affective Computing, and Artificial Intelligence.

Kohei Mori (Member)

He received Ph.D. degree from Tokyo Institute of Technology in 2002. He is currently an assistant professor at Graduate School of System Informatics, Kobe University. His current research interests include mathematical programming, stability analysis of nonlinear dynamical systems, and some topics in applied mathematics.