Deep Gate Recurrent Neural Network

JMLR: Workshop and Conference Proceedings 63:350–365, 2016 ACML 2016

Deep Gate Recurrent Neural Network

Yuan Gao GAOYUANKIDULT@GMAIL.COM
University of Helsinki

Dorota Glowacka GLOWACKA@CS.HELSINKI.FI
University of Helsinki

Editors: Robert J. Durrant and Kee-Eung Kim

Abstract

This paper explores the possibility of using multiplicative gate to build two recurrent neural network structures. These two structures are called Deep Simple Gated Unit (DSGU) and Simple Gated Unit (SGU), which are structures for learning long-term dependencies. Compared to traditional Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), both structures require fewer parameters and less computation time in sequence classification tasks. Unlike GRU and LSTM, which require more than one gate to control information flow in the network, SGU and DSGU only use one multiplicative gate to control the flow of information. We show that this difference can accelerate the learning speed in tasks that require long dependency information. We also show that DSGU is more numerically stable than SGU. In addition, we also propose a standard way of representing the inner structure of RNN called RNN Conventional Graph (RCG), which helps to analyze the relationship between input units and hidden units of RNN.

Keywords: Recurrent Neural Networks(RNN); Deep Neural Networks; Neural Network Representation

1. Introduction

The use of advanced architectures of RNNs, such as Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber (1997)) and Gated Recurrent Unit (GRU) (Cho et al. (2014a)) for learning long dependencies has led to significant improvements in various tasks, such as Machine Translation (Bahdanau et al. (2015)) or Robot Reinforcement Learning (Bakker (2001)). The main idea behind these networks is to use several gates to control the information flow from previous steps to the current steps. By employing the gates, any recurrent unit can learn a mapping from one point to another. Hochreiter proposed using two gates, namely an input gate and an output gate in the original LSTM (Hochreiter and Schmidhuber (1997)), while Gers added a forget gate to the original LSTM to form the well-known LSTM network (Gers et al. (2000)). Similarly, in GRU a reset gate and an update gate are used for the same purpose. In this paper, we simplify GRU to contain only one gate, which we name Simple Gated Unit (SGU). Then, by adding one layer to SGU, we form Deep SGU (DSGU). Both models use multiplicative operation to control the flow of information from the previous step to the current step. By doing so, we can reduce one-third (for SGU) and one-sixth (for DSGU) of parameters needed and accelerate the learning process compared to GRU. The results also indicate that adding layers in RNN's multiplication gate is an interesting direction of optimizing RNN structure.

In the following sections, we first describe the standard graph for representing detailed inner structure of RNN, i.e. relationships between the input units and recurrent units of each time step. We call this graph RNN Conventional Graph (RCG). Next, we introduce the structures of LSTM and GRU using RCG, followed by a description of the proposed network SGU and its variant DSGU. Finally, we present experimental results showing the performance of the proposed networks in several tasks, including IMDB semantic analysis, pixel-by-pixel MNIST classification task and a text generation task.

2. RNN Conventional Graph

RNN is a neural network with recurrent connections. The first step of training an RNN network is to unfold recurrent connections in time, which results in a deep hierarchy consisting of layers of the inner structure of RNN. In most cases, researchers produce their own graphs or use only mathematical equations to represent the structure of an RNN network. However, these representations can be rather complicated and idiosyncratic to each researcher. We designed RCG in order to provide a clear and standard view of the inner structure of RNN.

RCG consists of an input unit, an output unit, default activation functions and gates. It is easy to see how many and what kind of gates an RNN has and the graph can be easily translated into formulas. Normally, RCG takes $\boldsymbol{x}_t$ (sometimes also $\boldsymbol{x}_1$ to $\boldsymbol{x}_{t-1}$) and $\boldsymbol{h}_{t-1}$ (sometimes also $\boldsymbol{h}_1$ to $\boldsymbol{h}_{t-2}$) as inputs and produces $\boldsymbol{h}_t$ as an output. An RCG represents the input on the left side and the output on the right side, which enables the graph to show clearly how the information from the left side flows through different structures to the right side.

In the following sections, we use RCG to describe different structures of RNN.

2.1. RCG example: Vanilla Recurrent Neural Network

Vanilla Recurrent Neural Network (VRNN) is the simplest form of RNN. It consists of one input node and several hidden nodes. Hidden nodes are recurrent units, which means the current value $\boldsymbol{h}_t$ is updated according to the previous unit $\boldsymbol{h}_{t-1}$ and the current input i_t. Figure 1 shows the relationship between $\boldsymbol{x}_t$, $\boldsymbol{h}_{t-1}$ and $\boldsymbol{h}_t$ in VRNN.

In the form of RCG, VRNN can be represented as:

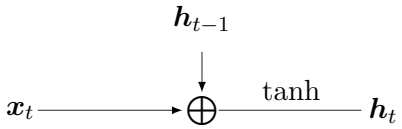

Figure 1: RCG of Vanilla Recurrent Neural Network. Recurrent input $\boldsymbol{h}_{t-1}$ is drawn from the upper side of the graph. Similarly, the input of the current step $\boldsymbol{x}_t$ is drawn from the left side. With an arrow $\rightarrow$ indicating a matrix multiplication operation, the information from two different sources goes into the addition node $\bigoplus$ in the middle of the graph. Followed by a non-linear function tanh, it outputs the current value of hidden units $\boldsymbol{h}_t$

Mathematically, the updating rule of VRNN is defined as:

$$\boldsymbol{h}_t = \sigma\left(W_{xh}\boldsymbol{x} + W_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}\right) \tag{1}$$

In Figure 1, an arrow $\rightarrow$ represents multiplication with a matrix, an addition node $\bigoplus$ indicates an addition operation to all the inputs. For example, $\boldsymbol{x}_t \rightarrow$ represents $W_{xh}\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1} \rightarrow$ represents $W_{hh}\boldsymbol{h}_{t-1}$. As a consequence, the whole graph can be directly transformed into Formula 1. The bias vector $\boldsymbol{b}$ is ignored in the graph as it can be integrated into the multiplication matrix.

3. LSTM and GRU

LSTM and GRU were developed to tackle a major problem suffered by traditional VRNN, namely the exploding and vanishing gradient problem for long-term dependency tasks (Pascanu et al. (2012)). They both use gated units to control the information flow through the network. However, LSTM differs from GRU in that LSTM uses three gates to control the information flow of the internal cell unit, while GRU only uses gates to control the information flow from the previous time steps.

3.1. LSTM

LSTM contains three gates: an input gate, an output gate and a forget gate – illustrated in Figure 2. At each iteration, the three gates try to remember when and how much the information in the memory cell should be updated. Mathematically, the process is defined by Formulas 2 to 6. Similarly to VRNN, LSTM can also be represented by RCG.

Figure 2 shows the RCG representation of LSTM. In the figure, $\boldsymbol{x}_t \rightarrow$ is defined as $W_{xi}\boldsymbol{x}$, $\boldsymbol{c}_{t-1} \rightarrow$ is defined as $W_{ci}\boldsymbol{c}_t$, $\boldsymbol{h}_{t-1} \rightarrow$ is defined as $W_{hi}\boldsymbol{h}_{t-1}$, *-var-* means that the output from the left side is named var and passed to the right side, $\bigoplus$ means summation over all the inputs, $\bigotimes$ means multiplication over all the inputs. For symbols $\bigoplus$ and $\bigotimes$, the input connections are normally defined as left and up connections, but if there are four connections to the node, then only the right connection is an output connection and the rest of the connections are input connections.

Mathematically, the relationship between the input and the output of LSTM is defined by a set of the following equations.

$$\boldsymbol{i}_t = \sigma\left(W_{xi}\boldsymbol{x}_t + W_{hi}\boldsymbol{h}_{t-1} + W_{ci}\boldsymbol{c}_{t-1} + \boldsymbol{b}_i\right) \tag{2}$$

$$\boldsymbol{f}_t = \sigma\left(W_{xf}\boldsymbol{x}_t + W_{hf}\boldsymbol{h}_{t-1} + W_{cf}\boldsymbol{c}_{t-1} + \boldsymbol{b}_f\right) \tag{3}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t\boldsymbol{c}_{t-1} + \boldsymbol{i}_t \tanh\left(W_{xc}\boldsymbol{x}_t + W_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c\right) \tag{4}$$

$$\boldsymbol{o}_t = \sigma\left(W_{xo}\boldsymbol{x}_t + W_{ho}\boldsymbol{h}_{t-1} + W_{co}\boldsymbol{c}_t + \boldsymbol{b}_o\right) \tag{5}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \tanh(\boldsymbol{c}_t) \tag{6}$$

Formula 2 describes the update rule of the input gate. It takes the output $\boldsymbol{h}_{t-1}$ of the last time step of the system, the input for the current time step $\boldsymbol{x}_t$, the memory cell value of the last time step $\boldsymbol{c}_{t-1}$ and a bias term $\boldsymbol{b}_t$ into its updating rule. Similarly, Formulas 3 to 6 use these parameters to update their values. Generally, the input gate controls the information flow into the memory cell, the forget gate controls the information flow out of

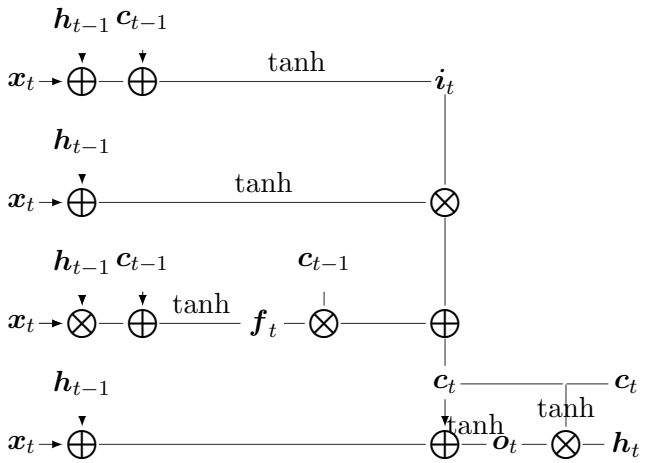

Figure 2: LSTM represented in the RCG form. The input $\boldsymbol{x}_t$ is fed in from the left side and recurrent connections are fed from either down or up. The outputs $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$ of this time step are output on the left side.

the system and the output gate controls how the information can be translated into the output. These three gates form a path of remembering the long-term dependency of the system. A direct relationship between the RCG representation of LSTM and Formulas 2 to 6 can be easily observed. The first line of Figure 2 shows Fomula 2. The second line of Figure 2 shows how Formula 4 calculates $\boldsymbol{c}_t$. Similarly, the third and fourth lines of the figure map to Formula 3 and Formula 5, respectively. In the RCG representation of LSTM, three multiplication gates are contained in the graph. It is an important characteristic of LSTM.

3.2. GRU

GRU was first designed by Kyunghyun Cho in his paper about Neural Machine Translation (Bahdanau et al. (2015)). This structure of RNN only contains two gates. The update gate controls the information that flows into memory, while the reset gate controls the information that flows out of memory. Similarly to the LSTM unit, GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell. Figure 3 shows the RCG representation of GRU.

The elements in Figure 3 are similar to the elements in Figure 2. However, the activation $\boldsymbol{h}_t$ of GRU at time t is a linear interpolation between the previous activation $\boldsymbol{h}_{t-1}$ and the candidate activation $\tilde{\boldsymbol{h}}_t$, which is represented by $\boxed{\sum\otimes}$ in the RCG representation of GRU and defined mathematically as:

$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \cdot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \cdot \tilde{h}_t, \tag{7}$$

where an update gate $\boldsymbol{z}_t$ decides how much the unit updates its activation, or information from the previous step.

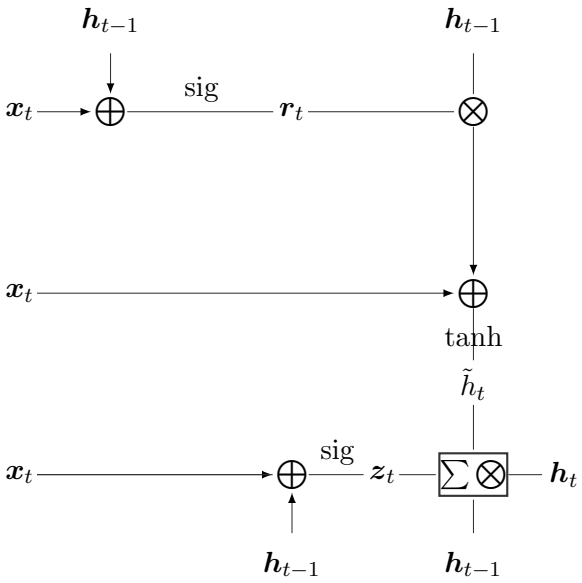

Figure 3: RCG representation of GRU. The input $\boldsymbol{x}_t$ is fed in from the left side. Recurrent connections are fed from either down or up, and output $\boldsymbol{h}_t$ is passed to the left. The special gate $\boxed{\sum\otimes}$ in the right corner is defined in Equation 7.

$$\boldsymbol{z}_t = \sigma_1(W_{hz}\boldsymbol{h}_{t-1} + W_{xz}\boldsymbol{x}_t) \tag{8}$$
$$\boldsymbol{r}_t = \sigma_1(W_{hr}\boldsymbol{h}_{t-1} + W_{xr}\boldsymbol{x}_t) \tag{9}$$
$$\tilde{h}_t = \sigma_2(W_{chx}\boldsymbol{x}_t + W_{chr}(\boldsymbol{r}_t \cdot \boldsymbol{h}_{t-1})) \tag{10}$$
$$\boldsymbol{h}_t = (\mathbf{1} - \boldsymbol{z}_t)h_{t-1} + \boldsymbol{z}_t\tilde{h}_t \tag{11}$$

The RCG representation of GRU can be directly translated into Formulas 8 to 11. Formula 8 represents the update gate, Formula 9 represents the reset gate and Formula 11 shows how the output $\boldsymbol{h}_t$ is calculated.

4. SGU and DSGU

In this section we describe the proposed Simple Gated Unit (SGU) and Deep Simple Gated Unit (DSGU).

4.1. SGU

SGU is a recurrent structure designed for learning long-term dependencies. Its aim is to reduce the amount of parameters needed to train and to accelerate the training speed in temporal classification tasks. As we observed earlier, GRU uses two gates to control the information flow from the previous time step to the current time step. However, compared

to GRU, SGU uses only one gate to control the information flow in the system, which is simpler and faster in terms of computation time.

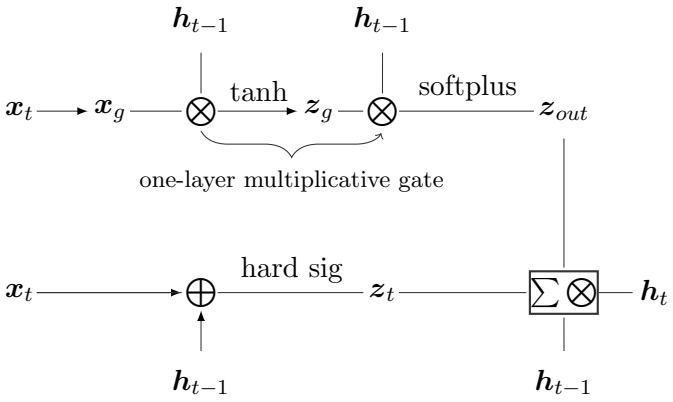

Figure 4: SGU in the form of RCG. This structure receives information from the current step and amplifies it by multiplying it with the current hidden states.

Figure 4 shows the structure of SGU. The input is fed into two different function units of the structure. The first line of the graph represents the gate to the recurrent neural network and the second line is the normal recurrent operation. Mathematically, Figure 4 represents the following formulas:

$$\boldsymbol{x}_g = W_{xh}\boldsymbol{x}_t + \boldsymbol{b}_g \tag{12}$$
$$\boldsymbol{z}_g = \sigma_1(W_{zxh}(\boldsymbol{x}_g \cdot \boldsymbol{h}_{t-1})) \tag{13}$$
$$\boldsymbol{z}_{out} = \sigma_2(\boldsymbol{z}_g \cdot \boldsymbol{h}_{t-1}) \tag{14}$$
$$\boldsymbol{z}_t = \sigma_3(W_{xz}\boldsymbol{x}_t + \boldsymbol{b}_z + W_{hz}\boldsymbol{h}_{t-1}) \tag{15}$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t)\boldsymbol{h}_{t-1} + \boldsymbol{z}_t \cdot \boldsymbol{z}_{out} \tag{16}$$

Compared to GRU, SGU needs fewer parameters. From Figure 4, we can observe that six weight matrices are needed for GRU, but SGU only needs four weight matrices. Inspired by IRNN, which is a Recurrent Neural Network (RNN) with rectifier as inner activation function (Le et al. (2015)), the structure uses softplus activation function for input, which intuitively enables the network to learn faster.

4.2. DSGU

DSGU is also an RNN structure designed for classification tasks. DSGU is designed to tackle a problem associated with SGU – we observed that if SGU is continuously trained, the process might drop dramatically. This is probably due to the shallowness of the gate and the nature of the softmax activation function.

Adding an extra weight matrix to $\boldsymbol{z}_{out}$ would make controlling the gate with a more complicated structure easier and the network more stable. The structure of DSGU is shown

in Figure 5. The only difference compared to SGU is that before $\boldsymbol{z}_{out}$ one weight matrix is added to the previous output.

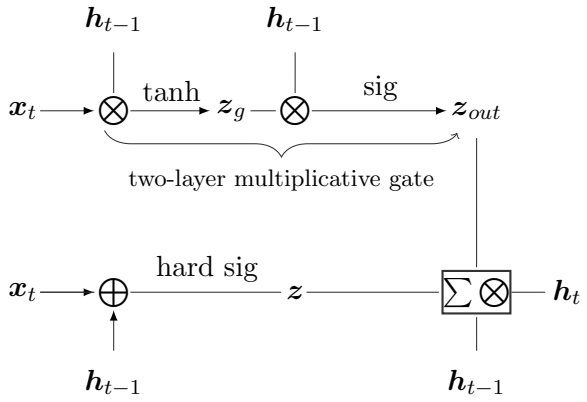

Figure 5: The structure of DSGU. Similarly to SGU, the input is fed into two different function units of the structure, namely $\boldsymbol{z}$ and $\boldsymbol{z}_{out}$.

The first line of the graph represents the gate to the recurrent neural network and the second line is the normal recurrent operation. Mathematically, Figure 5 represents the following formulas:

$$\boldsymbol{x}_g = W_{xh}\boldsymbol{x}_t + \boldsymbol{b}_g \tag{17}$$
$$\boldsymbol{z}_g = \sigma_1(W_{zxh}(\boldsymbol{x}_g \cdot \boldsymbol{h}_{t-1})) \tag{18}$$
$$\boldsymbol{z}_{out} = \sigma_2(W_{go}(\boldsymbol{z}_g \cdot \boldsymbol{h}_{t-1})) \tag{19}$$
$$\boldsymbol{z} = \sigma_3(W_{xz}\boldsymbol{x}_t + \boldsymbol{b}_z + W_{hz}\boldsymbol{h}_{t-1}) \tag{20}$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t)\boldsymbol{h}_{t-1} + \boldsymbol{z}_t \cdot \boldsymbol{z}_{out} \tag{21}$$

5. Experimental Results

In this section we report on the performance of SGU and DSGU in three classification tasks.

5.1. IMDB Sentiment Classification Task

We use a collection of 50,000 reviews from IMDB, extracted and provided by Stanford University (Maas et al. (2011)). Each movie has no more than 30 reviews and the whole dataset contains an even number of positive and negative reviews. As a consequence, a totally random algorithm yields 50% accuracy.

We use a system made by a stack of two layers. The first layer is the RNN layer (GRU, SGU, DSGU or LSTM) and the second layer is a dense layer with one output neuron. In this experiment, we use sigmoid as the activation function of the output layer and the binary cross-entropy as the cost function. All the tests were performed using Titan X GPU architecture.

We used standard initializations for the LSTM network, including the forget gate. The network uses glorot uniform as the initialization method of the input matrices and orthogonal initialization for the recurrent matrices. See Table 1 for initialization methods and activation functions for all the parameters in LSTM.

	update gate	reset gate	output gate	cell
W	glorot uniform	glorot uniform	glorot uniform	None
U	orthogonal	orthogonal	orthogonal	None
activation	hard sigmoid	hard sigmoid	hard sigmoid	tanh

Table 1: The initialization method and the activation function of each gate of LSTM in the IMDB sentiment analysis task.

Our implementation of GRU uses the standard structure mentioned above. W is the matrix used for multiplication of $\boldsymbol{x}$, and U is the matrix used for multiplication of the hidden units. Table 2 shows all the configurations of the initialization of different parameters.

	input gate	forget gate
W	glorot uniform	glorot uniform
U	orthogonal	orthogonal
activation	sigmoid	hard sigmoid

Table 2: The initialization method and activation function of each gate of GRU in the IMDB sentiment analysis task.

We ran GRU, SGU, DSGU and LSTM 50 times on the IMDB sentiment analysis dataset and calculated the mean and variance over all the experiments. The test results comparing SGU, DSGU, GRU and LSTM are shown in Figure 6 and Figure 7. Figure 6 compares the models in terms of the number of iterations, while Figure 7 compares the models in terms of time. Compared with LSTM and GRU, SGU can converge in a very short period (in approximately 2000 iterations or 180 seconds). In this task, we can also observe high variance in the testing phase when learning with LSTM, which makes LSTM less stable in practice. Both figures use the mean values for comparisons of SGU, DSGU, GRU and LSTM. In both cases, SGU and DSGU learn faster than GRU and LSTM.

5.2. MNIST Classification from a Sequence of Pixels

Image classification is a major problem in the field of image processing. MNIST is the simplest and the most well-studied dataset that has been used in many image classification tasks (LeCun et al. (1998)). In recent studies, pixel-by-pixel MNIST (Le et al. (2015)) was used to train RNN networks in order to test their ability to classify temporal data. Below, we follow research presented in Quoc Lee's paper (Le et al. (2015)) and compare SGU and DSGU against two models used in the paper (Le et al. (2015)) i.e. LSTM and IRNN.

In our experiments, we use a system made by a stack of two layers of neural networks. The first layer is a corresponding RNN layer (GRU, IRNN, SGU, DSGU or LSTM) and the second layer is a dense layer. Rmsprop optimization algorithm, softmax activation function

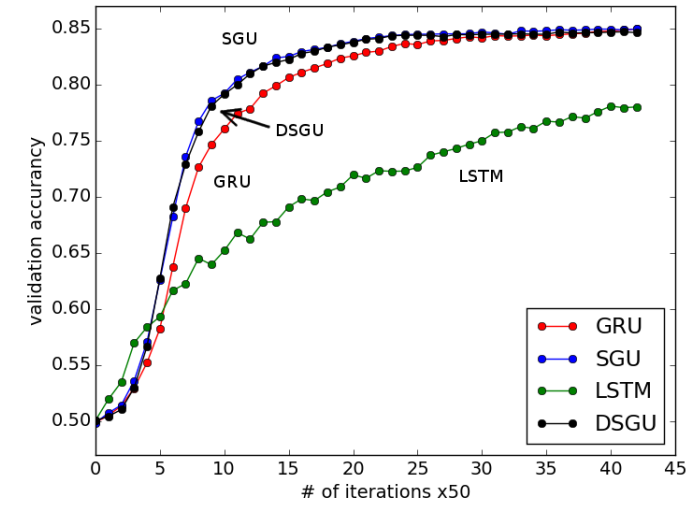

Figure 6: Comparison of SGU, DSGU, GRU and LSTM in the IMDB sentiment classification task. The y-axis shows validation accuracy of the model, whist the x-axis represents the number of iterations.

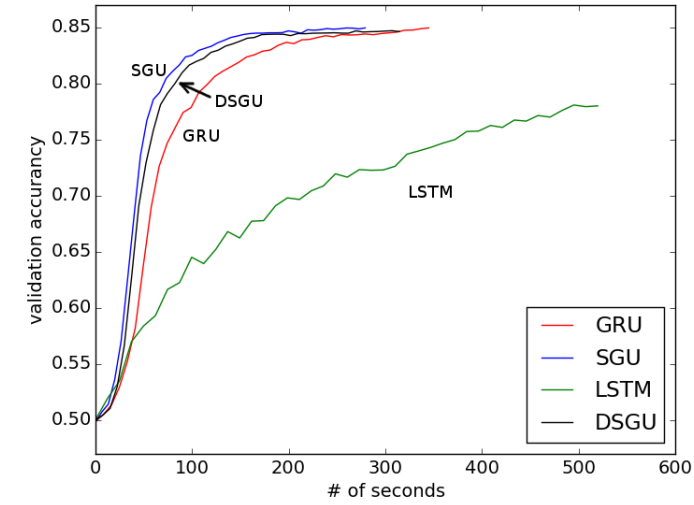

Figure 7: Comparison of SGU, DSGU, GRU and LSTM in the IMDB sentiment classification task in terms of seconds required to perform the task.

and categorical cross entropy were used for LSTM and IRNN in order to match the settings used by Quoc Lee (Le et al. (2015)). For IRNN, we used a relatively high learning rate of 1e-6 in order to speed up the learning process (In original paper (Le et al. (2015)), the

learning rate is 10^{-8}). We also tried using IRNN with Adam as the optimization algorithm, however the system did not learn in this case. In GRU, SGU and DSGU experiments, we used Adam (Kingma and Ba (2014b)) as the optimization algorithm, sigmoid function as the final layer activation function and categorical cross-entropy as the cost function. We would like to point out that using a sigmoid function as the final layer activation function does not give a proper probability distribution over different classes but it does provide a right prediction of class. Using sigmoid function as final layer activation function is essential for systems built using SGU and DSGU. All experiments were performed without cutting gradient.

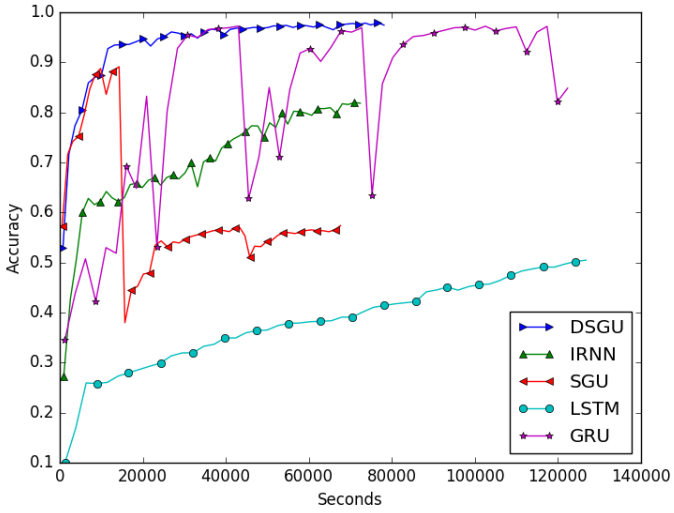

Figure 8: Validation accuracy of GRU, DSGU, IRNN and SGU and LSTM in terms of time in the MNIST classification task. Both DSGU and SGU reached a very high accuracy within a short period of time. However, SGU dropped after a short period, which might be due to the fact that it is too simple for learning this task.

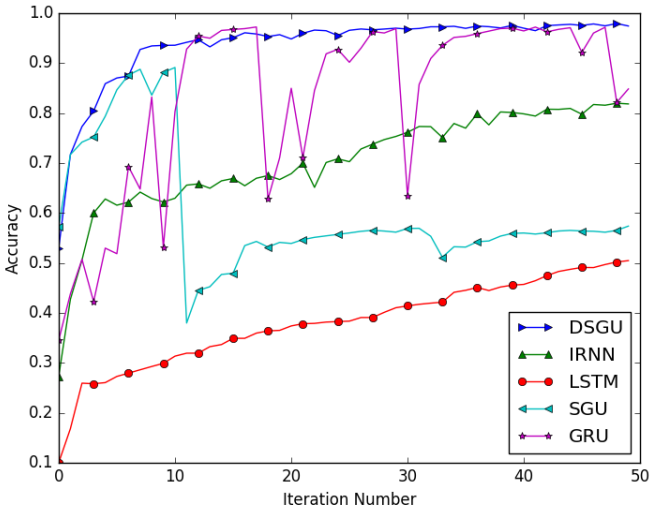

Figure 9: Validation accuracy of GRU, DSGU, IRNN and SGU and LSTM in terms of number of iterations in the MNIST classification task. Both DSGU and SGU reached a very high accuracy within a short period of time. However, SGU dropped after a short period, which might be due to the fact that it is too simple for learning this task.

The results, presented in Figures 8 and 9, show that both GRU and DSGU reached the best validation error accuracy (0.978) within a relatively short period of time (around 42000 seconds). (The best result of IRNN in paper (Le et al. (2015)) is 97% with a relatively long training time.) However, SGU failed to increase after around 30 iterations, which indicates hidden units in SGU might not be enough to keep the information in its structure in this particular task. This problem could be fixed by cutting the gradient in the structure, however, in order to provide a more general model and to avoid this problem, we propose using DSGU. GRU also behaved unstable in this task. We suspect using sigmoid function as an activation function of the output layer is not very suitable for GRU. This problem could also be fixed by cutting the gradient. After this paper was accepted, we noticed our result was surpassed by using LSTM with batch normalization (Cooijmans et al. (2016)).

5.3. Text Generation

Text generation is a specific task designed for the testing performance of recurrent neural networks. According to Graves (Graves (2013)), the recurrent neural network needs to be trained with a large number of characters from the same distribution, e.g. a particular book.

In this experiment, we use a collection of writings by Nietzsche to train our network. In total, this corpus contains 600901 characters and we input one character at a time in order to train the network to find a common pattern in the writing style of Nietzsche.

The structure for learning includes an embedding layer, a corresponding recurrent layer, and an output layer. For this experiment, we vary the recurrent layer and the activation function of the output layer. We tested DSGU, GRU and SGU with the sigmoid activation function in the output layer, while in LSTM, we used both sigmoid and softmax function in the output layer. The optimization algorithm for the models is Adam. We run each configuration 15 times and average the results.

Figure 10 shows the results of the text generation task in terms of the number of iteration. Figure 11 represents the results of the text generation task in terms of time. We can observe that DSGU reached the best accuracy (0.578) the fastest. SGU is also relatively fast. However, the best it can get is less than GRU (0.555 vs 0.556).

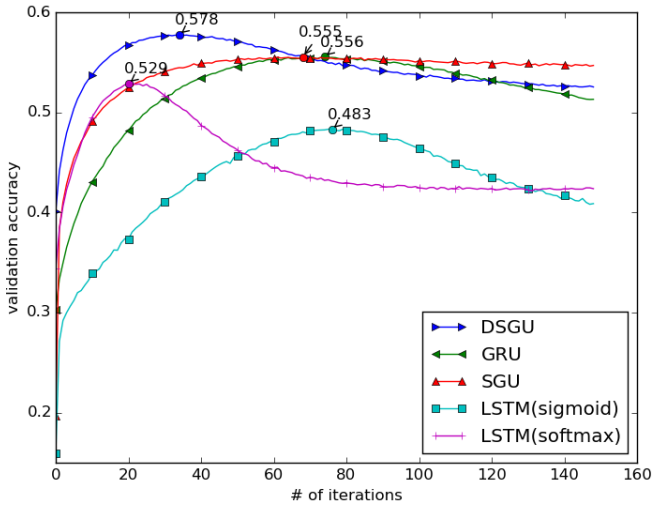

Figure 10: Validation accuracy of DSGU, GRU and SGU and LSTM in terms of the number of iterations in the text generation task. Each line is drawn by taking the mean value of 15 runs of each configuration.

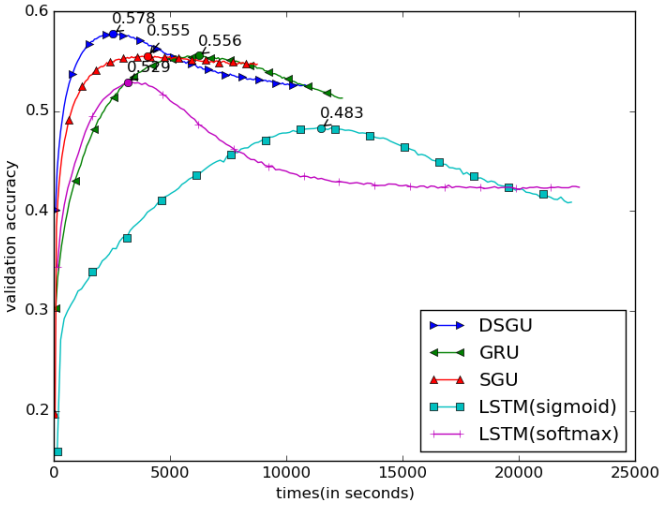

Figure 11: Validation accuracy of DSGU, GRU and SGU and LSTM in terms of time in the text generation task. Each line is drawn by taking the mean value of 15 runs of each configuration.

6. Conclusion

In this paper, we explored the possibility of using multiplicative gate to build two recurrent neural network structures, namely Deep Simple Gated Unit (DSGU) and Simple Gated Unit (SGU). Both structures require fewer parameters than GRU and LSTM.

In experiments, we noticed that both DSGU and SGU are very fast and often more accurate than GRU and LSTM. However, unlike DSGU, SGU seems to sometimes lack the ability to characterize accurately the mapping between two time steps, which indicates that DSGU might be more useful for general applications.

We also found deep multiplication gate to be an intriguing component of RNN. On one hand, with properly designed multiplication gate, RNN can learn faster than other models but become more fragile due to the fluctuation of data, on the other hand, adding a layer to the multiplication gate can make RNN more stable, while keeping the learning speed.

Regarding the potential drawbacks of SGU and DSGU, one thing we would like to point out is that both SGU and DSGU use sigmoid activation function (instead of softmax) as the output activation function for binary and multi-class classification tasks. We should note here that although binary classification using SGU and DSGU provides a probabilistic distribution, in multi-class case these models only provide the best class instead of a probabilistic distribution.

In the future, we would like to explore the usefulness of deep multiplication gate mathematically, test the performance with a deeper gate as well as perform experiments on more tasks.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

B Bakker. Reinforcement learning with long short-term memory. In *In Advances in Neural Information Processing Systems 15 (NIPS-2002)*, pages 1475–1482, 2003.

Bram Bakker. Reinforcement learning with long short-term memory. In *Neural Information Processing Systems*, pages 1475–1482, 2001.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014a.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv*, 2014b. URL http://arxiv.org/abs/1406.1078.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*, 2015.

Tim Cooijmans, Nicolas Ballas, Csar Laurent, alar Glehre, and Aaron Courville. Recurrent batch normalization. 2016.

C Daniel, G Neumann, and J Peters. Hierarchical Relative Entropy Policy Search. In *Fifteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2012)*, pages 1–9, 2012. URL http://www.is.tuebingen.mpg.defileadmin/user_upload/files/publications/2012/AISTATS-2012-Daniel.pdf.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. pages 1–26, October 2014. URL http://arxiv.org/abs/1410.5401.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. pages 1–13, December 2014a. URL http://arxiv.org/abs/1412.6980.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014b.

J. Kober, J. a. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 2013. ISSN 0278-3649. doi: 10.1177/0278364913495721. URL http://ijr.sagepub.com/cgi/doi/10.1177/0278364913495721.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012. ISSN 10495258.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep Learning for Detecting Robotic Grasps. *CoRR*, abs/1301.3, 2013. URL http://arxiv.org/abs/1301.3592.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P11-1015.

Maja J Matari, Complex Systems, and M.J. Matarić. Reinforcement learning in the multi-robot domain. *Autonomous Robots*, 4:73–83, 1997. ISSN 0929-5593. doi: 10.1023/A:1008819414322. URL http://www.springerlink.com/index/K662611455651Q42.pdf.

Hermann Mayer, Faustino Gomez, Daan Wierstra, Istvan Nagy, Alois Knoll, and Jürgen Schmidhuber. A system for robotic heart surgery that learns to tie knots using recurrent neural networks. In *IEEE International Conference on Intelligent Robots and Systems*, pages 543–548, 2006. ISBN 142440259X. doi: 10.1109/IROS.2006.282190.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

Jan Peters and Stefan Schaal. Policy gradient methods for robotics. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2219–2225, 2006. ISBN 142440259X. doi: 10.1109/IROS.2006.282564.

Anton Maximilian Schäfer, Steffen Udluft, et al. Solving partially observable reinforcement learning problems with recurrent neural networks. In *Workshop Proc. of the European Conf. on Machine Learning*, pages 71–81, 2005.

Yi Sun, Daan Wierstra, Tom Schaul, and Juergen Schmidhuber. Efficient natural evolution strategies. *Proceedings of the 11th Annual conference on Genetic and evolutionary computation GECCO 09*, pages 539–545, 2009. doi: 10.1145/1569901.1569976. URL http://portal.acm.org/citation.cfm?doid=1569901.1569976.

Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *In Advances in Neural Information Processing Systems 12*, pages 1057–1063, 1999. doi: 10.1.1.37.9714.

Paul J. Werbos. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78:1550–1560, 1990. ISSN 00189219. doi: 10.1109/5.58337.

Yezhou Yang, Y Li, and Y Aloimonos. Robot Learning Manipulation Action Plans by "Watching" Unconstrained Videos from the World Wide Web. *Under Review*, 2015. URL http://www.umiacs.umd.edu/~yzyang/paper/YouCookMani_CameraReady.pdf.