International Journal of Automation and Computing X(X), X X, X-X
DOI: XXX

Minimal Gated Unit for Recurrent Neural Networks

Guo-Bing Zhou Jianxin Wu Chen-Lin Zhang Zhi-Hua Zhou
National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China, 210023

Abstract: Recurrent neural networks (RNN) have been very successful in handling sequence data. However, understanding RNN and finding the best practices for RNN learning is a difficult task, partly because there are many competing and complex hidden units (such as LSTM and GRU). We propose a gated unit for RNN, named as Minimal Gated Unit (MGU), since it only contains one gate, which is a minimal design among all gated hidden units. The design of MGU benefits from evaluation results on LSTM and GRU in the literature. Experiments on various sequence data show that MGU has comparable accuracy with GRU, but has a simpler structure, fewer parameters, and faster training. Hence, MGU is suitable in RNN's applications. Its simple architecture also means that it is easier to evaluate and tune, and in principle it is easier to study MGU's properties theoretically and empirically.

Keywords: Recurrent neural network, MGU, gated unit, GRU, LSTM

1 Introduction

In recent years, deep learning models have been particularly effective in dealing with data that have complex internal structures. For example, convolutional neural networks (CNN) are very effective in handling image data in which 2D spatial relationships are critical among the set of raw pixel values in an image [1;2]. Another success of deep learning is handling sequence data, in which the sequential relationships within a variable length input sequence is crucial. In sequence modeling, recurrent neural networks (RNN) have been very successful in language translation [3–5], speech recognition [6], image captioning, i.e., summarizing the semantic meaning of an image into a sentence [7–9], recognizing actions in videos [10;11], or short-term precipitation prediction [12].

There is, however, one important difference between RNN and CNN. The key building blocks of CNN, such as nonlinear activation function, convolution and pooling operations, etc., have been extensively studied. The choices are becoming *convergent*, e.g., ReLU for nonlinear activation, small convolution kernels and max-pooling. Visualization also help us understand the semantic functionalities of different layers [13], e.g., firing at edges, corners, combination of specific edge groups, object parts, and objects.

The community's understanding of RNN, however, is not as thorough, and the opinions are much less convergent. Proposed by Hochreiter and Schmidhuber [14], the Long Short-Term Memory (LSTM) model and its variants have been popular RNN hidden units with excellent accuracy. Typical LSTM is complex, having 3 gates and 2 hidden states. Using LSTM as an representative example, we may find some obstacles that prevent us from reaching a consensus on RNN:

- **Depth with unrolling.** Different from CNN which has concrete layers, RNN is considered a deep model only when it is unrolled along the time axis. This property helps in reducing parameter size, but at the same time hinders visualization and understanding. This difficulty comes from the complexity of sequence data itself. We do not have a satisfactory solution for solving this difficulty, but we hope this paper will help in mitigating difficulties caused by the next one.
- **Competing and complex structures.** Since its inception, the focus of LSTM-related research has been altering its structure to achieve higher performance. For example, a forget gate was added by Gers et al. [15] to the original LSTM, and a peephole connection made it even more complex by Gers et al. [16]. It was not until recently that simplified models were proposed and studied, such as the Gated Recurrent Unit (GRU) by Cho et al. [3]. GRU, however, is still relatively complex because it has two gates. Very recently there have been empirical evaluations on LSTM, GRU, and their variants [17–19]. Unfortunately, no consensus has yet been reached on the best LSTM-like RNN architecture.

Theoretical analysis and empirical understanding of deep learning techniques are fundamental. However, it is very difficult if there are too many components in the structure. Simplifying the model structure is an important step to enable the learning theory analysis in the future.

Complex RNN models not only hinder our understanding. It also means that more parameters are to be learned and more components to be tuned. As a natural consequence, more training sequences and (perhaps) more training time are needed. However, evaluations by Chung et al. [17] and Jozefwicz et al. [18] both show that more gates do not lead to better accuracy. On the contrary, the accuracy of GRU is often higher than that of LSTM, albeit the fact that GRU has one less hidden state and one less gate than LSTM.

In this paper, we propose a new variant of GRU (which is also a variant of LSTM), which has minimal number of gates–only one gate! Hence, the proposed method is named as the Minimal Gated Unit (MGU). Evaluations in [17–19] agreed that RNN with a gated unit works significantly better than a RNN with a simple tanh unit without any gate.

Invited paper
Manuscript received date; revised date
This work was supported by National Natural Science Foundation of China (No. 61422203 and No. 61333014), and National Key Basic Research Program of China (No. 2014CB340501).
Recommended by Associate Editor xxxx

The proposed method has the smallest possible number of gates in any gated unit, a fact giving rise to the name *minimal* gated unit.

With only one gate, we expect MGU will have significantly fewer parameters to learn than GRU or LSTM, and also fewer components or variations to tune. The learning process will be faster compared to them, which will be verified by our experiments on a diverse set of sequence data in Section 4. What is more, our experiments also showed that MGU has overall comparable accuracy with GRU, which once more concurs the observation that fewer gates reduces complexity but not necessarily accuracy.

Before we present the details of MGU, we want to add that we are not proposing a "better" RNN model in this paper.[1] The purpose of MGU is two-fold. First, with a simpler model, we can reduce the requirement for training data, architecture tuning and CPU time, while at the same time maintaining accuracy. This characteristic may make MGU a good candidate in various applications. Second, a minimally designed gated unit will (in principle) make our analyses of RNN easier, and help us understand RNN, but this will be left as a future work.

2 RNN: LSTM, GRU, and More

We start by introducing various RNN models, mainly LSTM and GRU, and their evaluation results. These studies in the literature have guided us in how to minimize a gated hidden unit in RNN.

A recurrent neural network uses an index t to indicate different positions in an input sequence, and assumes that there is a hidden state $\boldsymbol{h}_t$ to represent the system status at time t. We use boldface letters to denote vectors. RNN accepts input $\boldsymbol{x}_t$ at time t, and the status is updated by a nonlinear mapping f from time to time:

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)\,. \tag{1}$$

One usual way of defining the recurrent unit f is a linear transformation plus a nonlinear activation, e.g.,

$$\boldsymbol{h}_t = \tanh\left(W\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}\right)\,, \tag{2}$$

where we combined the parameters related to $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$ into a matrix W, and $\boldsymbol{b}$ is a bias term. The activation (tanh) is applied to every element of its input. The task of RNN is to learn the parameters W and $\boldsymbol{b}$, and we call this architecture the simple RNN. An RNN may also have an optional output vector $\boldsymbol{y}_t$.

LSTM. RNN in the simple form suffers from the vanishing or exploding gradient issue, which makes learning RNN using gradient descent very difficult in long sequences [20;14]. LSTM solved the gradient issue by introducing various gates to control how information flows in RNN, which are summarized in Figure 1, from Equation (4a) to (4f), in which

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{3}$$

is the logistic sigmoid function (applied to every component of the vector input) and $\odot$ is the component-wise product between two vectors.

[1] We also believe that: without a carefully designed common set of comprehensive benchmark datasets and evaluation criteria, it is not easy to get conclusive decisions as to which RNN model is better.

LSTM (Long Short-Term Memory)

$$\boldsymbol{f}_t = \sigma\left(W_f\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_f\right)\,, \tag{4a}$$
$$\boldsymbol{i}_t = \sigma\left(W_i\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_i\right)\,, \tag{4b}$$
$$\boldsymbol{o}_t = \sigma\left(W_o\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_o\right)\,, \tag{4c}$$
$$\tilde{\boldsymbol{c}}_t = \tanh\left(W_c\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_c\right)\,, \tag{4d}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t\,, \tag{4e}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t)\,. \tag{4f}$$

GRU (Gated Recurrent Unit)

$$\boldsymbol{z}_t = \sigma\left(W_z\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_z\right)\,, \tag{5a}$$
$$\boldsymbol{r}_t = \sigma\left(W_r\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_r\right)\,, \tag{5b}$$
$$\tilde{\boldsymbol{h}}_t = \tanh\left(W_h\left[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right)\,, \tag{5c}$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \tilde{\boldsymbol{h}}_t\,. \tag{5d}$$

MGU (Minimal Gated Unit, the proposed method)

$$\boldsymbol{f}_t = \sigma\left(W_f\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_f\right)\,, \tag{6a}$$
$$\tilde{\boldsymbol{h}}_t = \tanh\left(W_h\left[\boldsymbol{f}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_h\right)\,, \tag{6b}$$
$$\boldsymbol{h}_t = (1 - \boldsymbol{f}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{f}_t \odot \tilde{\boldsymbol{h}}_t\,. \tag{6c}$$

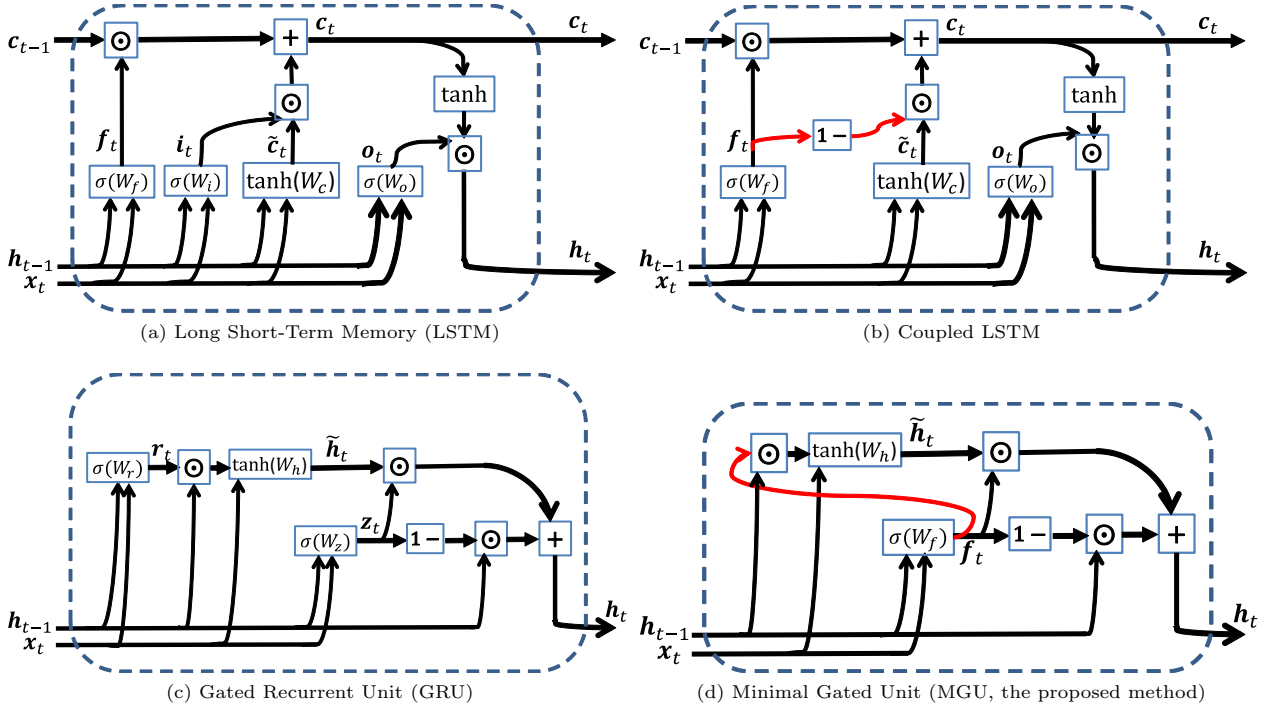
Figure 1: Summary of three gated units: LSTM, GRU, and the proposed MGU. σ is the logistic sigmoid function, and $\odot$ means component-wise product.

Figure 2a illustrates the data flow and operations in LSTM.

- There is one more hidden state $\boldsymbol{c}_t$ in addition to $\boldsymbol{h}_t$, which helps maintain long-term memories.
- The forget gate $\boldsymbol{f}_t$ decides the portion (between 0 and 1) of $\boldsymbol{c}_{t-1}$ to be remembered, determined by parameters W_f and $\boldsymbol{b}_f$.
- The input gate $\boldsymbol{i}_t$ determines which portion of time t's new information is to be added to $\boldsymbol{c}_t$ with parameter W_i and $\boldsymbol{b}_i$.
- The inputs are transformed as the update $\tilde{\boldsymbol{c}}_t$ with parameters W_c and $\boldsymbol{b}_c$. $\tilde{\boldsymbol{c}}_t$ (weighted by $\boldsymbol{i}_t$) and $\boldsymbol{c}_{t-1}$ (weighted by $\boldsymbol{f}_t$) form the new cell state $\boldsymbol{c}_t$.
- An output gate $\boldsymbol{o}_t$ is determined by parameters W_o and $\boldsymbol{b}_o$, and controls which part of $\boldsymbol{c}_t$ is to be output as the hidden state $\boldsymbol{h}_t$.

LSTM learning follows typical stochastic gradient descent and back propagation [21].

There are a lot of variants of LSTM. The original LSTM [14] does not include the forget gate, which was later introduced in [15]. Gers et al. [16] make LSTM even more complicated by allowing the three gates ($\boldsymbol{f}_t$, $\boldsymbol{i}_t$, $\boldsymbol{o}_t$) to take



(a) Long Short-Term Memory (LSTM)

(b) Coupled LSTM

(c) Gated Recurrent Unit (GRU)

(d) Minimal Gated Unit (MGU, the proposed method)

Figure 2: Data flow and operations in various gated RNN models. The direction of data flow are indicated by arrows, and operations on data are shown in rectangles. Five types of element wise operations (logistic sigmoid, tanh, plus, product and one minus) are involved. For operations with parameters (logistic sigmoid and tanh), we also include their parameters in the rectangle. These figures are different from diagrams that illustrate gates as switches, but match better to the equations in Figure 1.

$\boldsymbol{c}_{t-1}$ or $\boldsymbol{c}_t$ as an additional input, called the peephole connections. We choose the specific form of LSTM in Figure 1 because of two reasons. First, as will soon be discussed, the forget gate is essential. Second, the peephole connection does not seem necessary, but it complicates the learning process.

However, recently the trend is reversed: researchers found that simplifying the LSTM architecture may improve its performance.

LSTM with coupled gates. Greff et al.[19] evaluated a variant of LSTM, which couples the forget and input gates into one:

$$\boldsymbol{i}_t = 1 - \boldsymbol{f}_t\,, \quad \forall t\,, \tag{7}$$

as illustrated in Figure 2b. The coupling removed one gate and its parameters (W_i and $\boldsymbol{b}_i$), which leads to reduced computational complexity and slightly higher accuracy. Greff et al.[19] also observed that removing the peephole connection has similar effects.

GRU. The Gated Recurrent Unit (GRU) architecture further simplifies LSTM-like units[3]. GRU contains two gates: an update gate $\boldsymbol{z}$ (whose role is similar to the forget gate) and a reset gate $\boldsymbol{r}$ (whose role loosely matches the input gate). GRU's update rules are shown as Equation (5a) to (5d), and the data flow and operations are illustrated in Figure 2c. Beyond removing the output gate from LSTM, GRU also removed the hidden (slowly-changing) cell state $\boldsymbol{c}$. Note that GRU has appeared in different forms. When Cho et al.[3] originally proposed GRU, its form is different from Equation 5d, as

$$\boldsymbol{h}_t = \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \odot \tilde{\boldsymbol{h}}_t\,. \tag{8}$$

However, these two forms are mathematically equivalent. We adopt Equation 5d because it is more popular in the literature.

Evaluations by Chung et al.[17] found that when LSTM and GRU have the same amount of parameters, GRU slightly outperforms LSTM. Similar observations were also corroborated by Jozefwicz et al.[18].

SCRN. Instead of using gates to control information flow in RNN, the Structurally Constrained Recurrent Network (SCRN) added a hidden context vector $\boldsymbol{s}_t$ to simple RNN, which changes slowly over time if the parameter α is large[22], as

$$\boldsymbol{s}_t = \alpha \boldsymbol{s}_{t-1} + (1 - \alpha) B \boldsymbol{x}_t\,, \tag{9}$$

$$\boldsymbol{h}_t = \sigma(P \boldsymbol{s}_t + A \boldsymbol{x}_t + R \boldsymbol{h}_{t-1})\,, \tag{10}$$

in which α was fixed to 0.95, and the hidden state $\boldsymbol{h}_t$ hinges on three factors: $\boldsymbol{s}_t$, $\boldsymbol{x}_t$ and $\boldsymbol{h}_{t-1}$. SCRN has still fewer parameters than GRU, and has shown similar performance as LSTM by Mikolov et al.[22].

IRNN. Le et al.[23] showed that minor changes to the simple RNN architecture can significantly improve its accuracy. The key is to initialize the simple RNN's weight matrix to an identity matrix, and use ReLU (rectified linear unit) as the nonlinear activation function. This method

(named as IRNN) has achieved accuracy that is much closer to LSTM than that of simple RNN. Especially in the MNIST dataset [1], IRNN significantly outperforms LSTM. Similarly, Jozefwicz et al. [18] also showed that proper initialization is also important for LSTM. They showed that the bias of the forget gate should be set to a large value (e.g., 1 or 2). The same initialization trick was used by Le et al. in [23] too.

LSTM variants. Greff et al. [19] proposed, in their evaluation of the importance of LSTM components, 8 variants of the LSTM architecture. The evaluation results, however, showed that none of them can outperform the vanilla LSTM model. Their vanilla LSTM architecture has the peephole connections. Hence, it is slightly different from the LSTM architecture used in this paper (cf. Figure 1), and has more parameters too.

GRU variants. Jozefwicz et al. [18] proposed three variants of GRU. In these variants, they add the tanh nonlinearity in generating the gates, removing dependency to the hidden state in generating the gates, and make these changes while generating $\tilde{\boldsymbol{h}}_t$, etc. These variants sometimes achieve higher accuracy than GRU, but none of them can consistently outperform GRU.

Overall, with all these recent results and proposals we feel that among them GRU has some advantage in learning recurrent neural networks. Although it is not always the model with the highest accuracy or fewest parameters, it has stable performance (and it is usually one of the most accurate models) and relatively small amount of parameters. We will use GRU as the baseline method and compare it with the proposed MGU (minimal gated unit, cf. next section) in our experiments.

3 Minimal Gated Unit

As introduced in Section 1, we prefer an RNN architecture that has the smallest number of gates without losing LSTM's accuracy benefits. However, the choice of which gate to keep is not an easy job. Fortunately, several recent evaluations in the literature have helped us make this decision. Now we briefly summarize knowledge from these evaluations.

Jozefwicz et al. [18]: the forget gate is critical (and its biases $\boldsymbol{b}_f$ must be initialized to large values); the input gate is important, but the output gate is unimportant; GRU and LSTM have similar performance.

Greff et al. [19]: The forget and output gates are critical, and many variants of LSTM (mainly simplified LSTM variants) act similarly to LSTM.

Chung et al. [17]: Gated units work better than simple units without any gate; GRU and LSTM has comparable accuracy with the same number of parameters.

One notable thing is that different evaluations may lead to inconsistent conclusions, e.g., on the importance of the output gate Jozefwicz et al. [18] and Greff et al. [19] disagreed. This is inevitable because data with different properties have been used in different evaluations. However, at least we find the following consensus among these evaluations:

- Having a gated unit is beneficial to achieve high performance (e.g., higher accuracy or lower perplexity) of RNN architectures;
- The forget gate is considered important in all these evaluations; and,
- A simplified model may lower complexity and maintain comparable accuracy.

Hence, we propose the Minimal Gated Unit (MGU), which has the smallest possible number of gates in any gated unit. MGU only has 1 gate, which is the forget gate. MGU is based on GRU, and it further couples the input (reset) gate to the forget (update) gate, by specifying that

$$\boldsymbol{r}_t = \boldsymbol{f}_t\,,\ \ \forall t\,. \tag{11}$$

Note that we use $\boldsymbol{f}$ (instead of $\boldsymbol{z}$) to denote the only gate, because it is treated as the forget gate (which can be considered as a renaming of the update gate $\boldsymbol{z}$ in GRU). The equations that define MGU are listed in Figure 1 as Equations (6a) to (6c), and the data flow and operations are illustrated in Figure 2d.

In MGU, the forget gate $\boldsymbol{f}_t$ is first generated, and the element-wise product between $1 - \boldsymbol{f}_t$ and $\boldsymbol{h}_{t-1}$ becomes part of the new hidden state $\boldsymbol{h}_t$. The portion of $\boldsymbol{h}_{t-1}$ that is "forgotten" ($\boldsymbol{f}_t \odot \boldsymbol{h}_{t-1}$) is combined with $\boldsymbol{x}_t$ to produce $\tilde{\boldsymbol{h}}_t$, the short-term response. A portion of $\tilde{\boldsymbol{h}}_t$ (determined again by $\boldsymbol{f}_t$) form the second part of $\boldsymbol{h}_t$.

Comparing the equation sets in Figure 1 and the parameterized operations in Figure 2, it is obvious that MGU is more simplified than LSTM or GRU. While LSTM has four sets of parameters that determine $\boldsymbol{f}$, $\boldsymbol{i}$, $\boldsymbol{o}$ and $\tilde{\boldsymbol{c}}$, and GRU has three sets of parameters for $\boldsymbol{z}$, $\boldsymbol{r}$ and $\tilde{\boldsymbol{h}}$, MGU only has two sets of parameters, one for calculating $\boldsymbol{f}$, the other for $\tilde{\boldsymbol{h}}$. In other words, MGU has only roughly half the parameter size of that of LSTM, and 67% of that of GRU, because W_f (or W_z), W_i (or W_r), W_o, W_h have the same size. MGU also has slightly more parameters than SCRN, as will be shown in the example in Section 4.4. Since the parameter size of IRNN is the same as that of simple RNN, it must have the smallest number of parameters.

In short, MGU is a minimal design in any gated hidden unit for RNN. As we will show in the next section by experiments on a variety of sequence data, MGU also learns RNN for sequences without suffering from the gradient vanishing or gradient exploding problem (thanks to the forget gate in MGU). Because MGU only has few factors to tune, it is easier to find the best practices for MGU than for other gated units.

4 Experimental Results

In this section, we will evaluate the effectiveness of MGU using four datasets. The simple adding problem is used as a sanity check in Section 4.1. The IMDB dataset and the MNIST dataset are sentiment and image classification problems with sequence inputs, presented in Section 4.2 and 4.3, respectively. Finally, we evaluate MGU on the Penn TreeBank (PTB) language modeling dataset in Section 4.4.

As was shown in the evaluations [17;18], GRU has comparable accuracy with LSTM, and has fewer parameters than

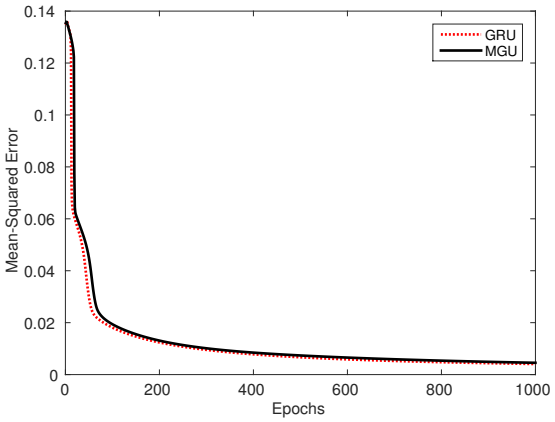


Figure 3: Test set mean squared error comparison (MGU vs. GRU) on the adding problem. Lower is better.

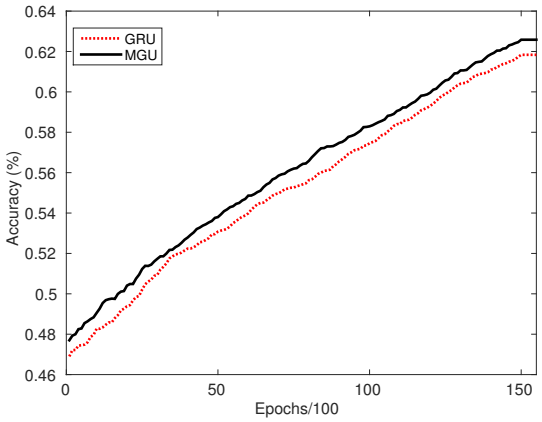


Figure 4: Test set classification accuracy comparison (MGU vs. GRU) on the IMDB dataset. Higher is better.

LSTM. We will use GRU as a baseline architecture, and compare the proposed MGU with GRU. If not otherwise specified, we compare these two algorithms with the same number of hidden units. All RNNs are implemented with the Lasagne package in the Theano library.[2]

The dropout technique is not used in either GRU or MGU. Because the focus of this paper is not absolutely high accuracy, we did not evaluate model averaging (ensemble) either.

The metrics that are compared in this paper include accuracy (or error, or perplexity) computed from the test sets, the average running time per epoch, and the number of parameters in the hidden units. We only count the parameters of the hidden unit, i.e., the preprocessing and fully connected regression or classification layer's parameters are not counted.

4.1 The Adding Problem

The adding problem was originally proposed by Hochreiter and Schmidhuber [14], and we use the variant proposed by Le et al. [23]. The input has two components: one random number in the range [0 1], and the other is a mask in $\{+1, 0, -1\}$. In the sequence (whose length ranges from 50 to 55), only 2 numbers are with mask $+1$, and the output should be the sum of these two. Both MGU and GRU use 100 hidden units; batch size is 100, and the learning rate is 10^{-3}. For this problem, we use a bidirectional network structure [21], which scans the sequence both from left to right and from right to left; the overall hidden state is the concatenation of the hidden state in both scans. On top of the last time step's hidden state, we add a fully connected layer to transform the last hidden state vector to a regression prediction.

We generated a training set of 10,000 examples and a test set of 1,000 examples. In Figure 3, we show the mean squared error of this simple regression task on the test set. MGU is slightly worse than GRU in the first 100 epochs. However, after 100 epochs these two algorithms have almost indistinguishable results. After 1,000 epochs, the mean squared regression error of both methods are below 0.005: 0.0041 for GRU and 0.0045 for MGU.

This simple experiment shows that MGU can smoothly deal with sequence input with a moderate length around 50. And, MGU has fewer parameters than GRU: 41,400 (MGU) vs. 62000 (GRU).

MGU also trains faster, which takes on average 6.85 seconds per epoch, while GRU requires 8.60 seconds.

4.2 IMDB

The second problem we study is sentiment classification in the IMDB movie reviews, whose task is to separate the reviews into positive and negative ones. This dataset was generated by Maas et al. in [24].[3] There are 25,000 movie reviews in the training set, another 25,000 for testing. We use the provided bag-of-words format as our sequence input. The maximum sequence length is 128. Both MGU and GRU have 100 hidden units; batch size is 16, and the learning rate is 10^{-8} with a 0.99 momentum. Similar to the adding problem, we use a fully connected layer on top of the last hidden state to classify a movie review as either positive or negative.

We show the accuracy of this binary classification problem in Figure 4, evaluated on the test set. In the x-axis of Figure 4, we show the epoch number divided by 100. Because both curves show that they converge after 15,000 epochs, it is too dense to show the results of the model after every training epoch.

MGU consistently outperforms GRU in this example, although by a small margin. After convergence, the accuracy of GRU is 61.8%. MGU achieves an accuracy of 62.6%, obtaining a 1.3% relative improvement. The input of IMDB is longer than that of the adding problem. In this larger and longer dataset, MGU verifies again it has comparable (or slightly better in this case) accuracy with GRU.

Again, MGU has two thirds of the number of parameters in GRU. MGU has 20,400 parameters, while GRU has 30,600 parameters. However, MGU trains much faster than GRU in this problem. The average running time per epoch

[2] `http://deeplearning.net/software/theano/`, `http://lasagne.readthedocs.org`.

[3] This dataset is available at `http://ai.stanford.edu/~amaas/data/sentiment/`.

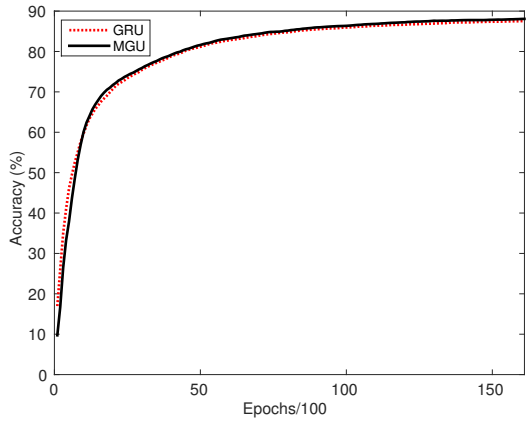


Figure 5: Test set classification accuracy comparison (MGU vs. GRU) on the MNIST dataset. This is the first task, where the sequence length is 28. Higher is better.

for MGU is 5.0 seconds, only 35% of that of GRU (which takes 14.1 seconds per epoch on average).

4.3 MNIST

The MNIST dataset by LeCun et al. [1] contains images of handwritten digits ('0'–'9'). All the images are of size 28×28.[4] There are 60,000 images in the training set, and 10,000 in the test set. The images are preprocessed such that the center of mass of these digits are at the central position of the 28×28 image.

MNIST has been a very popular testbed for deep neural network classification algorithms, but is not widely used in evaluating RNN models yet. We use this dataset in two ways. The first is to treat each row (28 pixels) as a single input in the input sequence. Hence, an image is a sequence with length 28, corresponding to the 28 image rows (from top to bottom). For this task, we use 100 hidden units and the batch size is 100. The learning rate is 10^{-8} with a 0.99 momentum. A fully connected layer transfer the last row's hidden state into an output vector with 10 elements. Accuracy of MGU and GRU on the test set in this task are shown in Figure 5, where the x-axis is the epoch number divided by 100.

The performance on MNIST is similar to that on the adding problem (cf. Figure 3), but the role of MGU and GRU has reversed. In the first 1,000 epochs (x-axis value up to 10), GRU is slightly better than our MGU. However, from then on till both algorithms' convergence, MGU's accuracy is higher than that of GRU. In the end (16,000 epochs), GRU achieves a 87.53% accuracy, while MGU is slightly higher at 88.07%. For this task, MGU has 25,800 parameters, while GRU has 38,700 parameters. The average per-epoch training time of MGU is 158 seconds, faster than that of GRU (182 seconds).

The second task on MNIST treats every pixel as one component in the input sequence. An image then becomes a sequence of length 784, with pixels scanned from left to right, and top to bottom. This task tests how MGU works when the sequence length is very long. In this task, GRU has 30,600 parameters while MGU has 20,400 parameters. Other settings (learning rate etc.) are the same as those in the first task.

After 16,000 epochs, MGU's test set accuracy is 84.25%, while with the same input length (784) and epoch number, IRNN's accuracy was below 65% [23] (cf. Figure 3 in that paper). After 32,000 epochs, IRNN's accuracy was roughly 80%, which is still worse than MGU's 84.25% at 16,000 epochs. Note that when the number of epochs continued till 500,000, IRNN reached a high accuracy of above 90%. Although we did not run MGU till this number of epochs, we have reasons to expect that MGU will similarly achieve a high accuracy if much longer training time is given. The accuracy of LSTM on this task is only around 65% even after 900,000 epochs [23] (cf. Figure 3 of that paper).

The average training time per epoch of MGU is 48.1 seconds. However, GRU is much slower in this long sequence task, which takes 145.3 seconds per epoch. We did not wait for GRU to run to a high epoch number on this task. The accuracy of GRU is higher than that of MGU in the early training stage. However, MGU takes over after 710 epochs. This trend is similar to that of Figure 5. If the same training time budget is allocated, MGU has significantly higher accuracy than GRU.

4.4 Penn TreeBank

The Penn TreeBank (PTB) dataset provides data for language modeling, which is released by Marcus et al. [25]. For this dataset, we work on the word-level prediction task, which is the same as the version by Wojciech et al. [26].[5] There are 10,000 words in the vocabulary. It has 929K words in the training set, 73K in the validation set, and 82K in the test set. As in [26], we use two layers and the sequence length is 35. The batch size is 20, and the learning rate is 0.01. Because dropout is not used, we tested MGU and GRU on small networks, whose number of hidden nodes range in the set $\{50, 100, 200, 300, 400, 500, 600\}$. Without dropout, both units overfit when the number of hidden units exceed 500. A fully connected layer predicts one of the 10,000 words. As a direct comparison, we show the perplexity of MGU and GRU in Figure 6 when there are 500 hidden units. The x-axis is the epoch number divided by 1,316. Note that two layers of MGU (or GRU) units are stacked to form one hidden unit.

GRU has a small advantage over MGU in this task. In the right end of Figure 6, GRU's perplexity on the test set is 101.64, while MGU's is higher at 105.59. We observe the same behavior on the validation set. We can also compare these two algorithms when they have roughly the same number of parameters, as was done in [17]. The comparison results are shown in Table 1.

MGU has one third less parameters than GRU. Thus, the number of parameters are roughly the same for MGU with 500 hidden units and GRU with 400 hidden units. When we compare these two algorithms using these settings, the gap between GRU and MGU becomes smaller (102.33 vs. 105.89 on the test set, and 107.92 vs. 110.68 on the validation set).

[4] This dataset is available at `http://yann.lecun.com/exdb/mnist/`.

[5] This dataset is available at `https://github.com/wojzaremba/lstm/`, or from `http://www.fit.vutbr.cz/00imikolov/rnnlm/simple-examples.tgz`.

Table 1: Results and statistics of GRU and MGU on the Penn TreeBank dataset. The average per-epoch training time is in seconds. The best result in each column is shown in boldface.

#hidden units	#parameters		Per epoch time		Validation perplexity		Test perplexity	
	GRU	MGU	GRU	MGU	GRU	MGU	GRU	MGU
50	37,800	25,200	177	174	172.71	185.34	165.52	177.39
100	90,600	60,400	180	163	130.94	140.35	125.24	135.33
200	241,200	160,800	184	175	111.28	119.77	105.75	114.14
300	451,800	301,200	197	181	108.08	113.21	102.55	108.27
400	722,400	481,600	201	190	**107.92**	110.92	102.33	106.02
500	1,053,000	702,000	211	198	108.30	**110.68**	**101.64**	**105.89**
600	1,443,600	962,400	218	205	111.42	113.50	104.88	108.22

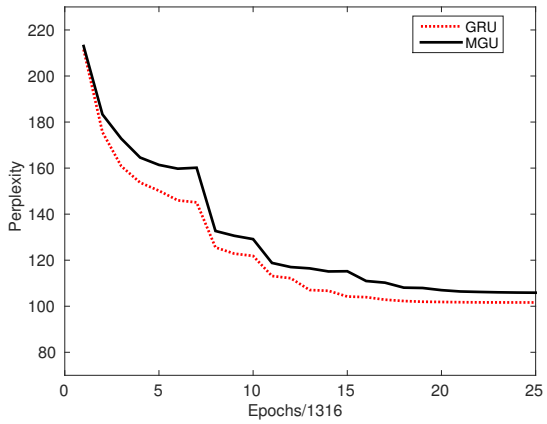


Figure 6: Test set perplexity comparison (MGU vs. GRU with 500 hidden units) on the PTB dataset. Lower is better.

If we compare these two algorithms with the same amount of training time, MGU is faster than GRU. MGU with 500 units is roughly as fast as GRU with 300 units; and, MGU with 300 units is similar to GRU with 100 units. When the numbers of hidden units are same (e.g., 500), the proposed MGU can run more epochs than GRU given the same amount of training time, which we expect will continue to decrease the perplexity of MGU.

We can also compare MGU with the results in [18]. When there are 400 hidden units, the total number of parameters (including all layers) of MGU is 4.8M, which can be fairly compared with the "5M-tst" result in [18], so is GRU with 300 units. Our GRU implementation (with 300 hidden units) has a test set perplexity of 102.55, lower than the GRU (with 5M parameters) result in [18], which is 108.42 (= exp(4.684)). The proposed MGU (with 400 hidden units) achieves a test set perplexity of 106.02, also lower than the GRU result in [18].

The SCRN method has still fewer parameters than the proposed MGU. When there are 100 hidden units, MGU has 60,400 parameters. A similar SCRN architecture has 100 hidden units and 40 context units [22], which has 48,200 parameters, amounting to roughly 80% of that of MGU. On this dataset, however, SCRN seems to saturate at test set perplexity 115, because SCRN with 100 and 300 hidden units arrived at this same perplexity. MGU gets lower perplexity than SCRN on this dataset.

One final comparison method is to test the combination of MGU and GRU. Since one hidden unit include two layers, we tested an additional network structure in which GRU is used as the bottom layer and MGU is the top layer. The combination GRU+MGU has fewer parameters than two layers of GRU (but more parameters than two layers of MGU). This hybrid structure achieved a test perplexity of 98.41 and a validation perplexity of 102.80 when 600 hidden units are used. Note that results of this hybrid architecture is significantly better than GRU alone or MGU alone.

4.5 Discussions

We have evaluated the proposed MGU on four different sequence data. The comparison is mainly against GRU, while results of IRNN and SCRN are also cited when appropriate. The input sequence length ranges short (35, 50–55), moderate (128), and long (784). The sequence data range from artificial to real-world, and the task domains are also diverse.

The proposed method is on par with GRU in terms of accuracy (or error, or perplexity). Given its minimal design of one gate, MGU has only two thirds of the parameters of GRU, and hence trains faster in all datasets. However, in some problems (e.g., Penn TreeBank), GRU converges faster than MGU. Overall, through these experimental results we believe MGU has proven itself as an attractive alternative in building RNN.

When we have accomplished this work we noticed a paper accepted into the ICCASP conference by Wu and King [27]. Although they also try to reduce the complexity of the LSTM model, the simplified LSTM (S-LSTM) model in [27] differs in two important aspects with MGU. S-LSTM is defined by the following equations:

$$\boldsymbol{f}_t = \sigma\left(W_f\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_f\right), \tag{12a}$$

$$\tilde{\boldsymbol{c}}_t = \tanh\left(W_c\left[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right] + \boldsymbol{b}_c\right), \tag{12b}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + (1 - \boldsymbol{f}_t) \odot \tilde{\boldsymbol{c}}_t, \tag{12c}$$

$$\boldsymbol{h}_t = \tanh(\boldsymbol{c}_t). \tag{12d}$$

First, they maintain two sets of hidden states $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$. Second, the output $\boldsymbol{h}$ is produced by $\boldsymbol{c}_t$ and a tanh nonlinearity.

S-LSTM has achieved similar performance as LSTM in speech synthesis [27]. Because MGU has similar performance as GRU (whose accuracy is similar to and sometimes

better than LSTM), it is reasonable to assume that S-LSTM and MGU will also perform similarly. However, MGU has a simpler structure than S-LSTM, which is beneficial to both practical applications and theoretical analyses.

5 Conclusions and Future Work

In this paper, we proposed a new hidden unit for recurrent neural network learning. The proposed Minimal Gated Unit (MGU) has the minimal design in any gated hidden unit for RNN. It has only one gate (the forget gate) and does not involve the peephole connection. Hence, the number of parameters in MGU is only half of that in the Long Short-Term Memory (LSTM), or two thirds of that in the Gated Recurrent Unit (GRU). We compared MGU with GRU on several tasks that deal with sequence data in various domains. MGU has achieved comparable accuracy with GRU, and (thanks to the minimal design) trains faster than GRU.

Based on our evaluations, MGU could be used as the hidden unit in an RNN, which may reduce memory footprint and training time in some applications. More importantly, the minimal design will facilitate our theoretical analysis or empirical observation (e.g., through visualization) of RNN models, and enhance our understanding and facilitate progresses in this domain. A minimal design also means that there are fewer possibilities of producing variants (which will complicate the analysis).

Ample ways are possible to further this line of research. Beyond analysis and understanding, we will also run MGU with more epochs, in more diverse and complex tasks, and regularize MGU to improve its accuracy.

References

[1] Yann LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. Advances in Neural Information Processing Systems 25*, pages 1097–1105, 2012.

[3] Kyunghyun Cho, Bart van Meriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proc. Empirical Methods in Natural Language Processing*, pages 1724–1735, 2014.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. In *Proc. Advances in Neural Information Processing Systems 27*, pages 3104–3112, 2014.

[5] Dzmitry Bahdanau, KyungHyun Cho, and Yoshua Bengio. Nueral Machine Translation by Jointly Learning to Align and Translate. In *Int'l Conf Learning Represenations*, 2015.

[6] Alan Graves, Abdel-Rahman Mohamed, and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. In *Proc. Int'l Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2013.

[7] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proc. Int'l Conf. on Machine Learning*, pages 2048–2057, 2015.

[8] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[9] Remi Lebret, Pedro O. Pinheiro, and Ronan Collobert. Phrase-based Image Captioning. In *Proc. Int'l Conf. on Machine Learning*, pages 2085–2094, 2015.

[10] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. In *Proc. IEEE Int'l Conf. on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[11] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised Learning of Video Representations using LSTMs. In *Proc. Int'l Conf. on Machine Learning*, pages 843–852, 2015.

[12] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-Chun Woo. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In *Proc. Advances in Neural Information Processing Systems 28*, pages 802–810, 2015.

[13] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *Proc. European Conf. Computer Vision*, volume LNCS 8689, pages 818–833, 2014.

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[15] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to Forget: Continual Prediction with LSTM. In *Proc. Int'l Conf Artificial Neural Networks*, volume 2, pages 850–855, 1999.

[16] Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research*, 3:115–143, 2002.

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[18] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An Empirical Exploration of Recurrent Network Architectures. In *Proc. Int'l Conf. on Machine Learning*, pages 2342–2350, 2015.

[19] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey . *arXiv preprint arXiv:1503.04069*, 2015.

[20] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning Long-term Dependencies with Gradient Descent is Difficult. *IEEE Trans.Neural Networks*, 5(2):157–166, 1994.

[21] Alex Graves and Jürgen Schmidhuber. Framewise Phoneme Classification with Bidirectional LSTM Networks. *Neural Networks*, 18(5-6):602–610, 2005.

[22] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato. Learning Longer Memory in Recurrent Neural Networks. In *Int'l Conf Learning Represenations*, 2015.

[23] 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.

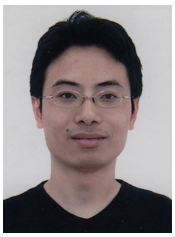
[24] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning Word Vectors for Sentiment Analysis. In *Proc. 49th Annual Meeting of the ACL*, pages 142–150, 2011.

[25] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

[26] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[27] Zhizheng Wu and Simon King. Investigating gated recurrent neural networks for speech synthesis. In *International Conference on Acoustics, Speech and Signal Processing*, 2016.

Guo-Bing Zhou received his B.S degree in computer science from Nanjing University. He is currently a postgraduate student in Nanjing University and will receive his M.S. degree in July 2016. His current research interest is machine learning.
E-mail: zhougb@lamda.nju.edu.cn



Jianxin Wu received his PhD degree in computer science from the Georgia Institute of Technology. He is currently a professor in the Department of Computer Science and Technology at Nanjing University, China. He has served as an area chair for ICCV 2015 and senior PC member for AAAI 2016. His research interests are computer vision and machine learning.
E-mail: wujx@lamda.nju.edu.cn (Corresponding author)

Chen-Lin Zhang was born in 1994. He is a candidate for the Bachelor's degree in the Department of Computer Science and Technology, Nanjing University. His main research interests include computer vision and machine learning.
E-mail: u-zhangcl@lamda.nju.edu.cn

Zhi-Hua Zhou is a Professor, Standing deputy director of the National Key Laboratory for Novel Software Technology, and Founding Director of the LAMDA Group at Nanjing University. His research interests are mainly in artificial intelligence, machine learning and data mining. He is a Fellow of the AAAI, IEEE, IAPR, IET/IEE, CCF, and an ACM Distinguished Scientist.
E-mail: zhouzh@lamda.nju.edu.cn