LETTER **Communicated by Qiang Ye**

Gated Orthogonal Recurrent Units: On Learning to Forget

Li Jing
ljing@mit.edu
Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Caglar Gulcehre
ca9lar@gmail.com
University of Montreal, Montreal H3T, 1J4, Quebec, Canada

John Peurifoy
jpeurifo@mit.edu
Yichen Shen
ycshen@mit.edu
Max Tegmark
tegmark@mit.edu
Marin Soljacic
soljacic@mit.edu
Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

Yoshua Bengio
yoshua.umontreal@gmail.com
University of Montreal, Montreal H3T 1J4, Quebec, Canada

We present a novel recurrent neural network (RNN)–based model that combines the remembering ability of unitary evolution RNNs with the ability of gated RNNs to effectively forget redundant or irrelevant information in its memory. We achieve this by extending restricted orthogonal evolution RNNs with a gating mechanism similar to gated recurrent unit RNNs with a reset gate and an update gate. Our model is able to outperform long short-term memory, gated recurrent units, and vanilla unitary or orthogonal RNNs on several long-term-dependency benchmark tasks. We empirically show that both orthogonal and unitary RNNs lack the ability to forget. This ability plays an important role in RNNs. We provide competitive results along with an analysis of our model on many natural sequential tasks, including question answering, speech spectrum prediction, character-level language modeling, and synthetic tasks that involve long-term dependencies such as algorithmic, denoising, and copying tasks.

Li Jing and Caglar Gulcehre contributed equally to this letter.

Neural Computation **31, 765–783** (2019)
doi:10.1162/neco_a_01174

1 Introduction

Recurrent neural networks (RNNs) with gating units—such as long short-term memory (LSTMs) (Hochreiter & Schmidhuber, 1997; Gers, 2001) and gated recurrent units (GRUs; Cho, Van Merriënboer, Gulcehre et al., 2014)—have led to rapid progress in different areas of machine learning, such as language modeling (Graves, Wayne, & Danihelka, 2014), neural machine translation (Cho et al., 2014; Sutskever, Vinyals, & Le, 2014), and speech recognition (Chan, Jaitly, Le, & Vinyals, 2016; Chorowski, Bahdanau, Serdyuk, Cho, & Bengio, 2015). These studies have proved the importance of gating units for RNNs.

The main advantage of using these gated units in RNNs is primarily due to the ease of optimization of the models using them and to reduce the learning degeneracies such as vanishing gradients that can cripple conventional RNNs (Pascanu, Mikolov, & Bengio, 2013). Most important, by designing special gates, it is easier to impose a particular behavior on the model, such as creating shortcut connections through time by using input and forget gates in LSTMs and resetting the memory via the reset gate of a GRU. These gates also bring modularity to the neural network design that seems to make training those models easier. Gated RNNs are also empirically shown to achieve better results for a wide variety of real-world tasks.

Recently, using unitary and orthogonal matrices (instead of general matrices) as the recurrence matrix of RNNs (Arjovsky, Shah, & Bengio, 2016; Jing et al., 2016; Henaff, Szlam, & LeCun, 2016) has attracted an increasing amount of attention in the machine learning community. This trend is motivated by the ability of these matrix constraints to effectively solve tasks involving long-term dependencies and the vanishing or exploding gradients problem (Bengio, Simard, & Frasconi, 1994; Hochreiter, 1991). Thus a unitary or orthogonal RNN can capture long-term dependencies more effectively in sequential data than a conventional RNN or LSTM. As a result, this type of model has been shown to perform well on tasks that would require rote memorization (Hochreiter, 1991) and simple reasoning, such as the copy task (Hochreiter & Schmidhuber, 1997) and sequential MNIST (Le, Jaitly, & Hinton, 2015). Those models can be viewed as an extension to vanilla RNNs (Jordan, 1997) that replaces the transition matrices with either unitary or orthogonal matrices.

In this letter, we refer to the ability of a model to omit parts of the input sequence that contain redundant information and to filter out the noise input in general as the means of a forgetting mechanism. Previously (Gers, Schmidhuber, & Cummins, 2000) showed the importance of the forgetting mechanism for LSTM networks. With similar motivations, we discuss the utilization of a forgetting mechanism for RNNs with orthogonal transitions. The importance of forgetting for those networks is mainly due to the fact that unitary or orthogonal RNNs can backpropagate the gradients without

vanishing through time, and it is very easy for them to just have an output that depends equally on all the elements of the entire input sequence. From this perspective, learning to forget can be difficult with unitary or orthogonal RNNs because these models can clog their memory with useless information. However, most real-world applications and natural tasks require the model to filter out irrelevant or redundant information from the input sequence. We argue that difficulties of forgetting can cause unitary and orthogonal RNNs to perform badly on many realistic tasks, and we demonstrate this empirically with several real-world tasks such as question answering and language modeling.

We propose a new architecture, the gated orthogonal recurrent unit (GORU), which combines the advantages of gated RNNs and unitary or orthogonal RNNs: the ability to capture long-term dependencies by using orthogonal matrices and the ability to "forget" by using a GRU structure. We demonstrate that GORU is able to learn long-term dependencies effectively, even in complicated data sets that require a forgetting ability. In this work, we focus on using orthogonal transition matrices, which are a subset of the unitary matrices.

GORU outperforms several variations of unitary or orthogonal RNNs (Arjovsky et al., 2016; Jing et al., 2016; Mhammedi, Hellicar, Rahman, & Bailey, 2017) on language modeling, denoising, and the question answering tasks. We show that they fail catastrophically on a denoising task that requires the model to forget. On question answering, speech spectrum prediction, algorithmic, and the denoising tasks, GORU achieves better accuracy on the test set over all other models that we compare against. We have attempted to use gates on the unitary matrices with complex numbers, but we encountered some training challenges of training gating mechanisms; thus, we focus solely on orthogonal matrices in this letter.

2 Background

Given an input sequence $\mathbf{x}_t \in \mathbb{R}^{d_x}, t \in \{1, 2, \ldots, T\}$, a vanilla RNN defines a sequence of hidden states $\mathbf{h}_t \in \mathbb{R}^{d_h}$ updated at each time step according to the rule

$$\mathbf{h}_t = \phi(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}), \tag{2.1}$$

where $\mathbf{W}_h \in \mathbb{R}^{d_h \times d_h}$, $\mathbf{W}_x \in \mathbb{R}^{d_x \times d_h}$, and $\mathbf{b} \in \mathbb{R}^{d_h}$ are model parameters and ϕ is a nonlinear activation function. RNNs have proven to be effective for solving sequential tasks due to their flexibility. However, a well-known problem, vanishing or exploding gradients, has prevented RNNs from efficiently learning long-term dependencies (Bengio et al., 1994). Several approaches have been developed to solve this problem, with LSTMs and GRUs being the most successful and widely used.

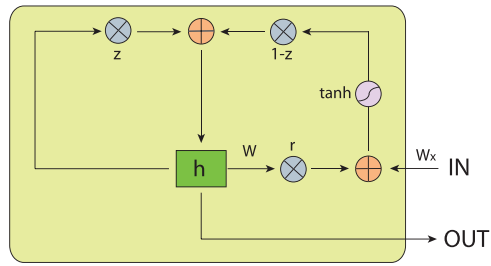

Figure 1: The GRU architecture. r and z are reset and update gates. h is the hidden state.

2.1 Gated Recurrent Unit. A big step forward from LSTM is the GRU, proposed by Cho, Van Merriënboer, Bahdanau, and Bengio (2014), which removes the extra memory state in LSTM. Specifically, the hidden state $\mathbf{h}_t$ in a GRU is updated as follows:

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tanh(\mathbf{W}_x \mathbf{x}_t + \mathbf{r}_t \odot \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{b}_h), \tag{2.2}$$

$$\mathbf{z}_t = \text{sigmoid}(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z), \tag{2.3}$$

$$\mathbf{r}_t = \text{sigmoid}(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r), \tag{2.4}$$

where $\mathbf{W}_{\{z,r\}} \in \mathbb{R}^{(d_h+d_x)\times d_h}$, $\mathbf{W}_x \in \mathbb{R}^{d_x \times d_h}$, $\mathbf{W}_h \in \mathbb{R}^{d_h \times d_h}$, and $\mathbf{b}_{\{z,r,h\}} \in \mathbb{R}^{d_h}$. Figure 1 demonstrates the architecture of the GRU model.

Although LSTMs and GRUs were proposed to solve the exploding and vanishing gradient problem (Hochreiter, 1991; Bengio et al., 1994), they can in practice still suffer from this issue for long-term tasks. As a result, gradient clipping (Pascanu et al., 2013) is usually required in the training process, although clipping addresses only gradient explosion.

2.2 Unitary and Orthogonal RNNs. A complex-valued matrix $\mathbf{U}$ is unitary when it satisfies $\mathbf{U}\mathbf{U}^{*T} = \mathbf{I}$. A matrix $\mathbf{U}$ is orthogonal if it is both unitary and real valued. Therefore, any vector $\mathbf{x}$ that multiplies a unitary or an orthogonal matrix satisfies

$$||\mathbf{U}\mathbf{x}|| = ||\mathbf{x}||. \tag{2.5}$$

Thanks to this property, a unitary or orthogonal matrix is able to preserve the norm of vectors that flow through and thus allow for the gradient to propagate through longer time steps. Recent papers (Arjovsky et al., 2016; Henaff et al., 2016) pointed out that unitary or orthogonal matrices can effectively prevent the gradient vanishing or explosion problem in conventional

RNNs. After this work, several other unitary or orthogonal RNN models have been proposed (Jing et al., 2016; Wisdom, Powers, Hershey, Le Roux, & Atlas, 2016; Hyland & Rtsch, 2017; Mhammedi et al., 2017), all showing promising abilities in capturing long-term dependencies in data.

A unitary or orthogonal RNN is simply defined as replacing the recurrence matrices in a vanilla RNN by unitary or orthogonal matrices:

$$\mathbf{h}_t = \sigma(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}_x\mathbf{x}_t, \mathbf{b}). \tag{2.6}$$

In practice, unitary or orthogonal matrices perform effectively when they are combined with a nonlinear function such as the modReLU (Arjovsky et al., 2016; Jing et al., 2016),

$$\text{modReLU}(\mathbf{z}, \mathbf{b})_i = \frac{z_i}{|z_i|}\text{ReLU}(|z_i| + b_i), \tag{2.7}$$

where the bias vector $\mathbf{b}$ is a shared trainable parameter and $|z_i|$ is the norm of the complex number z_i. Note that this nonlinearity preserves the phase in the complex domain.

2.3 Parameterization of Orthogonal Matrices. There are many unitary matrix parameterization methods (Arjovsky et al., 2016; Wisdom et al., 2016; Jing et al., 2016).

One of the most successful architectures, proposed by Arjovsky et al. (2016), parameterizes the hidden-to-hidden matrix as

$$\mathbf{W} = \mathbf{D}_3\mathbf{T}_2\mathcal{F}^{-1}\mathbf{D}_2\Pi\mathbf{T}_1\mathcal{F}\mathbf{D}_1. \tag{2.8}$$

Here $\mathbf{D}_{1,2,3}$ are diagonal matrices with each element $e^{i\omega_j}$, $j = 1, 2, \ldots, n$. $\mathbf{T}_{1,2}$ are reflection matrices, and $\mathbf{T} = I - 2\frac{\widehat{\mathbf{v}}\widehat{\mathbf{v}}^\dagger}{||\widehat{\mathbf{v}}||^2}$, where $\widehat{\mathbf{v}}$ is a vector with each of its entries as a parameter to be trained. Π is a fixed permutation matrix. $\mathcal{F}$ and $\mathcal{F}^{-1}$ are Fourier and inverse Fourier transform matrices, respectively. Since each factor matrix here is unitary, the product $\mathbf{W}$ is also a unitary matrix.

Instead of this parameterization, another approach is to use a geodesic gradient descent method proposed by Wisdom et al. (2016):

$$\mathbf{A}^{(t)} \equiv \mathbf{G}^{(t)^\dagger}\mathbf{W}^{(t)} - \mathbf{W}^{(t)^\dagger}\mathbf{G}^{(k)}, \tag{2.9}$$

$$\mathbf{W}^{(t+1)} \equiv \left(\mathbf{I} + \frac{\lambda}{2}\mathbf{A}^{(t)}\right)^{-1}\left(\mathbf{I} - \frac{\lambda}{2}\mathbf{A}^{(t)}\right)\mathbf{W}^{(t)}. \tag{2.10}$$

This method is able to span the entire unitary space, which gives full representativeness. It also empirically shows an advantage over the previous method (Arjovsky et al., 2016).

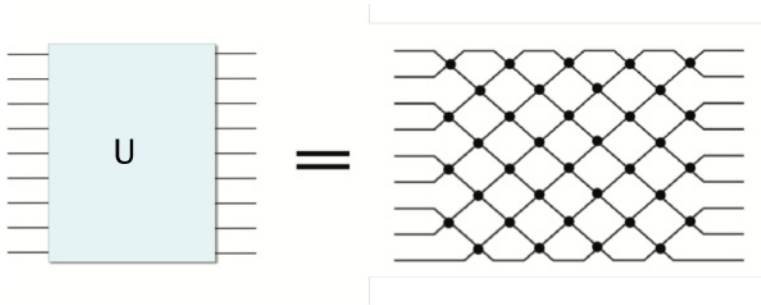

Figure 2: Orthogonal matrix parameterization by 2-by-2 rotation matrices. Each row represents one neuron in the hidden state. Each junction represents a 2-by-2 rotation matrix on the two corresponding neurons.

Jing et al. (2016) proposed another systematic parameterization method. The hidden-to-hidden matrix **U** is decomposed into a sequence of 2-by-2 rotation matrices, as shown in Figure 2. Each 2-by-2 rotation contains one trainable rotation parameter.

3 The Gated Orthogonal Recurrent Unit RNN

In this section, we discuss the forgetting problem of vanilla orthogonal RNNs and demonstrate the architecture of gated orthogonal recurrent unit (GORU) RNN.

3.1 The Problem of Forgetting in Orthogonal RNNs. First, we argue for the advantage of an RNN that can forget some of its past inputs. This is desirable because we seek a state representation that can capture the most important elements of the past sequence and can discard irrelevant details or noise. This ability becomes particularly critical when the dimensionality of the RNN state is smaller than the product of the sequence length with the input dimension, that is, when some form of compression is necessary. For this compression to be most useful for further processing, it is likely that it requires a nonlinear combination of the past input values, allowing the network to forget and ignore unnecessary elements from the past.

Now consider an RNN whose state is obtained as a sequence of orthogonal transformations, with each transformation being a function of the input at a given time step. Let us focus on the class of orthogonal transformations that are basically rotation for simplicity, which (noncommutativity aside) are analogous to addition in the space of angles. When we compose several orthogonal operators, we just add more angles together. So we forget in the mild sense that we get in the state a combination of several rotations (e.g., adding the angles), and we lose track of exactly which individual rotations

were applied. The advantage is that in the space of angles, the derivative of the final angle to any of the individually added angle is one, so there is no vanishing gradient. However, we cannot have complete forgetting, such as making the new state independent of the past inputs (or of some of them that we wish to forget). For a new rotation to cancel an old rotation, one would need the new rotation to "know" about the old rotation to cancel (i.e., it would need to be a function of the old rotation). But this is not what happens, because each new rotation is chosen before looking at the current state. Instead, in a regular RNN, the state update depends in a nonlinear way on the past state, so that, for example, when a particular value of the state is reached, it can be reset to zero. This would not be possible with just the composition of orthogonal transformations. These considerations motivate an architecture in which we combine orthogonal or unitary transformations with nonlinearities that can be trained to forget when and where it is appropriate.

3.2 GORU Architecture. This section introduces the GORU. In our architecture, we change the hidden state loop matrix into an orthogonal matrix and change the respective activation function to modReLU:

$$\begin{aligned}\mathbf{h}_t &= \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \text{modReLU}(\mathbf{W}_x \mathbf{x}_t \\ &\quad + \mathbf{r}_t \odot (\mathbf{U}\mathbf{h}_{t-1}) + \mathbf{b}_h), \qquad (3.1)\\ \mathbf{z}_t &= \text{sigmoid}(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{W}_{z,x}\mathbf{x}_t + \mathbf{b}_z), \qquad (3.2)\\ \mathbf{r}_t &= \text{sigmoid}(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{W}_{r,x}\mathbf{x}_t + \mathbf{b}_r), \qquad (3.3)\end{aligned}$$

where σ is a suitable nonlinear activation function and $\mathbf{W}_z, \mathbf{W}_r \in \mathbb{R}^{d_h \times d_h}$, $\mathbf{b}_z, \mathbf{b}_r, \mathbf{b}_h \in \mathbb{R}^{d_h}$, and $\mathbf{W}_{z,x}, \mathbf{W}_{r,x}, \mathbf{W}_x \in \mathbb{R}^{d_x \times d_h}$. $\mathbf{r}_t$ and $\mathbf{z}_t$ are the reset and update gates, respectively. $\mathbf{U} \in \mathbb{R}^{d_h \times d_h}$ is kept orthogonal. In fact, we have modified only the main loop that absorbs new information to the orthogonal while leaving the gates unchanged compared to the GRU. Figure 3 demonstrates the architecture of the GORU model.

The update gates of the GORU help the model filter out irrelevant or noise information coming from the input. It can be thought of as acting like a low-pass filter. The orthogonal transition matrices help the model prevent the gradients from vanishing through time. However, the ways an orthogonal transformation can interact with the hidden state of an RNN is limited to reflections and rotations. The reset gate enables the model to rescale the magnitude of the hidden state activations ($\mathbf{h}_t$).

In the following experiment, we follow the FFT-style parameterization method demonstrated in Jing et al. (2016). Similar to the tunable-style architecture shown in Figure 2, it is also built up by 2-by-2 rotational matrices. This architecture provides full symmetry with a minimum number of parameters, as shown in Figure 4.

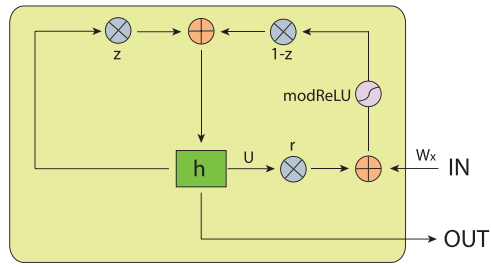

Figure 3: Architecture of the GORU model. h is the hidden state. For GORU, r and z are reset and update gates. It uses modReLU activation function instead of tanh.

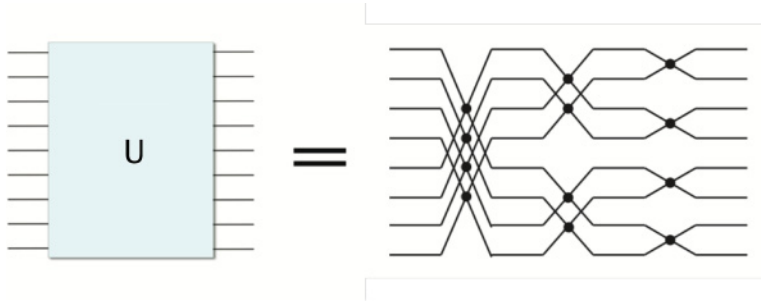

Figure 4: FFT-style orthogonal matrix parameterization by 2-by-2 rotation matrices. Each row represents one neuron in the hidden state. Each junction represents a 2-by-2 rotation matrix on the corresponding two neurons.

4 Experiments

We compare GORU with two unitary RNNs—EURNN (Jing et al., 2016) and uRNN (Arjovsky et al., 2016)—one orthogonal RNN—oRNN (Mhammedi et al., 2017)—and two other well-known gated RNNs (LSTMs and GRUs). Previous research on unitary or orthogonal RNNs has mainly focused on memorization tasks; in contrast, we focus on more realistic noisy tasks, which require the model to discard parts of the input sequence to be able to use its capacity efficiently. (GORU is implemented in Tensorflow, available at https://github.com/jingli9111/GORU-tensorflow.)

For synthetic tasks, including a copying memory task, a denoising task, and an algorithmic task, we set the hidden state size to match the total number of parameters. Specifically, we set the hidden size equal to 90, 100, and 128 for LSTM, GRU, and GORU, respectively, and 512 for all unitary or orthogonal RNNs. The total number of parameters is about 38,000 for each model in the copying memory and denoising tasks. We used 50,000

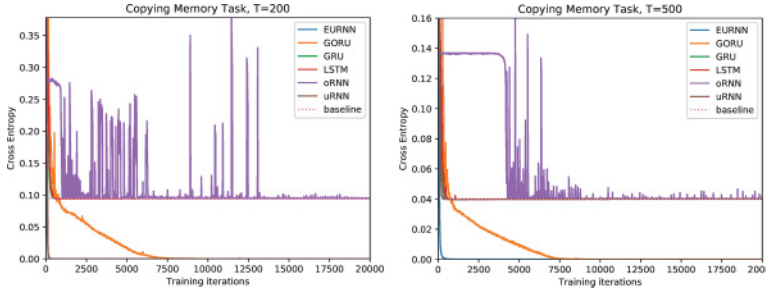

Figure 5: Copying memory task with delay time $T = 200$ and $T = 500$: cross entropy on validation set for each iteration. Hidden state sizes are set to 128, 512, 512, 512, 100, 90 for GORU, oRNN, uRNN, EURNN, GRU, LSTM, respectively, to match the total number of parameters. GORU is the only gated system to successfully solve this task; both GRU and LSTM get stuck at baseline. EURNN and uRNN are seen to converge within hundreds of iterations. oRNN also gets stuck at baseline.

training examples for each of these two tasks. We report the cross entropy on a separate validation data with 1000 examples for each training iteration.

4.1 Copying Memory Task. The first task we consider is the well-known copying memory task. This synthetic task is commonly used to test the network's ability to remember information seen T time steps earlier.

Specifically, the task is defined as follows. An alphabet consists of symbols $\{a_i\}$, $i \in \{0, 1, \ldots, n-1, n, n+1\}$, the first n of which represent data and the remaining two representing "blank" and "marker," respectively. Here we choose $n = 8$. The input sequence contains 10 data steps, followed by "blank." The RNN model is supposed to output "blank" and give the original sequence once it sees the "marker." Note that each instance has a different location for these 10 elements of data.

In this experiment, we use the RMSProp optimizer (Tieleman & Hinton, 2012). We apply a grid search on learning rate, decay rate, and minibatch size. The learning rate is chosen from {0.01, 0.001, 0.0001}, the decay rate from {0.5, 0.5, 0.99}, and the minibatch size from {64, 128, 256}.

This task only requires the model to efficiently overcome the gradient vanishing or explosion problem and does not require a forgetting ability. Unitary RNNs perform perfectly and go through to the baseline in no time—as previously seen. The vanilla orthogonal RNN gets stuck at the baseline. The GORU is the only gated system to successfully solve this task up to $T = 500$, while the GRU and LSTM get stuck at baseline, as shown in Figure 5. An ablation study on this task is in the appendix.

4.2 Denoising Task. We evaluate the forgetting ability of each RNN architecture on a synthetic denoising task. A list of data points is located

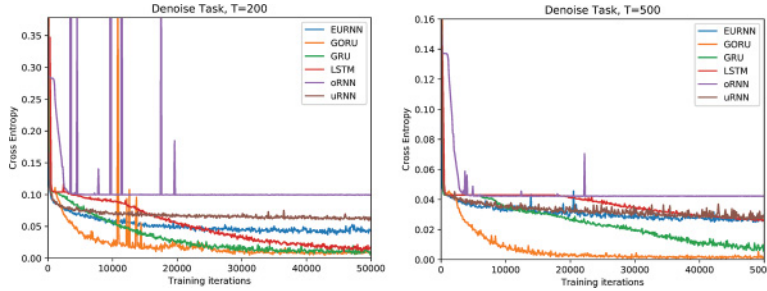

Figure 6: Denoising task with sequence length $T = 200$ and $T = 500$: cross entropy on validation set for each iteration. Hidden state sizes are set to 128, 512, 512, 512, 100, and 90 for GORU, oRNN, uRNN, EURNN, GRU, and LSTM, respectively, to match the total number of parameters. For $T = 200$, unitary or orthogonal RNNs get stuck at baseline because they lack a forgetting mechanism, while gated RNN models successfully solve the task. For T = 500, only GORU is able to solve the denoising task.

randomly in a long, noisy sequence. The RNN model is supposed to filter out the useless part ("noise") and output the remaining sequential labels.

Similarly to the labels of the copying memory task, an alphabet consists of symbols $\{a_i\}, i \in \{0, 1, \ldots, n-1, n, n+1\}$, the first n of which represent data and the remaining two represent "noise" and "marker," respectively. The input sequence contains 10 randomly located data steps, and the rest are filled by "noise." The RNN model is supposed to output those 10 data in a sequence after it sees the "marker." We use an RMSProp Optimizer. We use the same grid search as in the copying memory task.

This task requires the ability to learn long dependencies and to forget the noisy input. GORU significantly outperforms all other RNN models, as shown in Figure 6. Unitary or orthogonal RNNs, however, get stuck at baseline, just as we intuitively expected. An ablation study on this task is in the appendix.

4.3 Algorithmic Task. We tested the RNN models on the algorithmic task as described in Li, Tarlow, Brockschmidt, and Zemel (2015). The model is fed with a random graph as an input sequence and is required to output the shortest path at the end of the sequence. We have used the same setup and data provided in Li et al. (2015). The data are divided into a training set with 1000 examples, a validation set of 50 examples, and 1000 examples for test.

We used an Adam optimizer (Kingma & Ba, 2014) with a grid search on hyperparameters. The learning rate is chosen from {0.01, 0.001, 0.0001}, which is determined by the validation set performance for each model individually. The minibatch size is set to be 50 for all models.

Table 1: Algorithmic Test on GORU, GRU, LSTM, EURNN, oRNN, and uRNN.

Model	Accuracy
LSTM	70.3 ± 1.1
GRU	64.2 ± 2.1
uRNN	66.3 ± 2.2
oRNN	44.5 ± 2.4
EURNN	54.5 ± 0.9
GORU (EURNN FFT-style)	$\mathbf{73.9} \pm 1.4$
GORU (Householder)	73.6 ± 2.3

Notes: GORU significantly outperforms all other RNN models. The bold number represents the highest accuracy.

We summarize the test set results in Table 1. We find that the GORU outperforms GRU and LSTM and unitary and orthogonal RNNs.

4.4 bAbI: Episodic Question Answering. We tested the ability of our RNN models on a word-level episodic question answering task. The bAbI data set (Weston et al., 2015) examines RNN's ability to understand language and perform basic logical reasoning. Each training example is a set of statements that are logically related in some fashion. For instance, one training example consists of these three statements and a question: *Mary went to the bathroom. John moved to the hallway. Mary traveled to the office. Where is Mary?* Answer: *office*.

There are 20 types of questions that can be asked—some requiring deduction between lines and some requiring association. The bAbI data set is useful because it contains a small vocabulary and short sentences, and it requires one-word answers for each story. Thus, it is a good benchmarking test because the word mapping layers are not the dominant sources of parameters.

We test each task with a unidirectional RNN without any attention mechanism. In detail, we word-embed and then feed one RNN the sequence of statements. Another RNN is fed the word-embedded question. Then we concatenate the outputs of the two RNNs into a single input for a third RNN that then outputs the correct word.

For each task, we use 8000 training examples, 2000 validation examples, and 1000 test examples. The validation set is used to determine the best hyperparameters with early stopping for each individual model. The early stop criterion is used so that there is no decrease in validation loss for five epochs. We summarize the test set results in Table 2. We find that the GORU performs better on average than GRU and LSTM and unitary and orthogonal RNNs.

Table 2: Question Answering Task on bAbI Data Set.

Task	GORU	GRU	LSTM	EURNN	uRNN	oRNN
1 Single supporting fact	45.8	49.1	**49.3**	47.2	45.7	15.6
2 Two supporting facts	**39.5**	38.5	32.3	24.3	18.3	18.2
3 Three supporting facts	**33.5**	32.2	20.6	22.5	19.1	20.5
4 Two argument relations	62.7	64.6	**67.5**	56.1	56.1	18.0
5 Three argument relations	**87.0**	78.0	52.3	56.2	51.9	32.5
6 Yes/no questions	**53.6**	50.5	49.3	50.5	49.2	50.1
7 Counting	77.7	**79.5**	76.9	71.9	72.7	50.0
8 Lists/sets	75.0	75.5	**76.8**	56.5	47.9	63.4
9 Simple negation	62.9	**63.9**	63.5	60.6	61.8	**63.9**
10 Indefinite knowledge	45.4	44.8	**46.0**	42.6	43.3	43.6
11 Basic coreference	69.3	71.2	71.1	**72.1**	70.0	17.9
12 Conjunction	69.9	71.6	71.9	**72.7**	71.9	16.2
13 Compound coreference	92.7	**94.2**	93.8	92.4	93.2	17.2
14 Time reasoning	37.9	**39.2**	34.4	20.0	23.9	20.8
15 Basic deduction	55.2	**57.4**	20.9	25.0	27.1	25.4
16 Basic induction	44.0	**45.9**	**45.9**	43.3	43.9	26.2
17 Positional reasoning	**59.6**	50.5	51.6	51.2	49.5	50.6
18 Size reasoning	90.5	89.9	**91.8**	89.7	86.5	51.2
19 Path finding	8.9	9.6	8.2	9.0	7.0	**10.2**
20 Agent's motivations	**97.7**	**97.7**	96.5	93.3	93.3	77.6
Mean performance	**60.4**	58.2	56.0	52.9	51.6	34.5

Notes: Test accuracy (%) on GORU, GRU, LSTM, EURNN, uRNN, and oRNN. All RNN models are unidirectional without extra memory or attention mechanism. GORU achieves the highest average accuracy. The bold numbers represent the highest accuracy.

4.5 Language Modeling: Character-Level Prediction. We test each RNN on character-level language modeling. The RNN is fed one character for each step from a real context and is supposed to output the prediction for the next character. We use the Penn Treebank corpus (Marcus, Marcinkiewicz, & Santorini, 1993). The text is in English, and the vocabulary consists of 10,000 words. The train/val/test split is 5.1 million/400,000/450,000 characters, and rare words are replaced with *<unk>*.

We use RMSProp with a grid search of minibatch size in {32, 64} and learning rate in {0.001, 0.0001} for each model. Each training sequence is unfolded into 50 time steps. Similar to most other work in language modeling, at the end of each sequence, the hidden state is saved and used to initialize the hidden state for the next sequence. This allows the neural network to give consistent predictions even at the beginning of a sequence.

We used the validation set to choose the best hyperparameters with early stopping for each individual model. The early stopping criterion is so that there is no decrease in validation loss for five epochs. We show the final test performance in Table 3 by comparing their performance in terms of bits

Table 3: Penn Treebank Character-Level Modeling Test on GORU, GRU, LSTM, and EURNN.

Model	bpc	Number of Units
LSTM	1.656	184
GRU	1.639	216
EURNN	1.747	1024
uRNN	1.750	1024
oRNN (Mhammedi et al., 2017)	1.68	512 ($m = 510$)
GORU (EURNN FFT-style)	1.654	256
GORU (Householder)	1.652	256
GRU (w/ modReLU)	1.702	216
GORU (w/ ReLU)	1.785	256
GORU (w/ tanh)	1.780	256
GORU (w/o reset gate)	1.759	256
GORU (w/o update gate)	1.718	256

Notes: We use only single-layer models. We choose the size of the models to match the number of parameters, which is about 184,000 for each model. GORU is able to outperform unitary and orthogonal RNNs. We also tested the performance of restricted GORU, which shows the necessity of both reset and update gates.

per character. GORU is performing comparably to LSTM and GRU in our experiments, and it performs significantly better than unitary or orthogonal RNNs.

We have also done several ablation studies in this task:

1. Disabling the reset or update gates. When disabling the reset gate, equation 3.2 is removed and $\mathbf{z}_t$ become trainable parameters. When disabling the update gate, equation 3.3 is removed, and $\mathbf{r}_t$ become trainable parameters.
2. Using tanh or ReLU in GORU instead of modReLU.
3. Using modReLU in GRU instead of tanh.
4. Using Householder reflection orthgonal matrix architecture (Mhammedi et al., 2017) instead of EUNN FFT-style architecture (Jing et al., 2016)

The first three ablation models give significantly worse performance than GORU. These empirically prove the importance of gated mechanisms and the need for modReLU for GORU. Traditional activation functions such as ReLU or tanh still give vanishing gradients even with an orthogonal recurrence matrix. The last ablation model shows that using Householder reflection orthogonal matrix architecture in GORU gives a similar result as using EURNN FFT-style architecture. This proves that the advantage of GORU

Table 4: Speech Spectrum Prediction Test on LSTM, GRU, EURNN, GORU, uRNN, and oRNN.

Model	Number of Units	MSE(validation)	MSE(test)
LSTM	50	51.0	50.7
GRU	60	52.1	52.4
EURNN(Jing et al., 2016)	128	51.8	51.9
oRNN	128	46.2	46.9
uRNN	108	—	—
GORU (EURNN FFT-style)	64	45.5	45.7
GORU (Householder)	64	**40.9**	**43.0**
GORU (with ReLU)	64	45.8	47.4
GORU (with tanh)	64	59.7	59.6
GORU (without reset gate)	64	45.9	46.9
GORU (without update gate)	64	46.3	47.9

Note: The hidden size of each model is set to match the total number of parameters. uRNN failed to converge in this task. GORU significantly outperforms all other RNN models.

over oRNN comes from the additional gates instead of the parameterization method.

Since most of the relevant information for character-level prediction can be obtained only by using the recent rather than distant past (Karpathy, Johnson, & Li, 2015), the core of the character-prediction challenge does not involve the main strength of unitary or orthogonal RNNs.

4.6 Speech Spectrum Prediction. We tested the ability of our RNN models on a real-world speech spectrum prediction task in the log-magnitude short-time Fourier transform (STFT) (Wisdom et al., 2016; Jing et al., 2016). We used the TIMIT data set (Garofolo et al., 1993) sampled at 8 kHz. The audio file is initially divided into short time frames, Fourier-transformed into the frequency domain, transferred to log scale, and finally normalized to match the maximum value of each example. In our STFT operation, we used a Hann analysis window of 256 samples (32 milliseconds) and a window hop of 128 samples (16 milliseconds). In this task, the RNNs are required to predict the log-magnitude of the STFT frame at time $t + 1$, given all the log-magnitudes of STFT frames up to time t.

We used a training set with 2400 utterances, a validation set of 600 utterances, and a test set of 1000 utterances. We used an Adam optimizer with a grid search on learning rate in {0.01, 0.001} and batch size in {32, 64 128}. The hidden state size is set to keep the total number of parameters the same for each model, which is about 42,000.

We found that GORU significantly outperforms all other models with the same number of parameters as shown in Table 4.

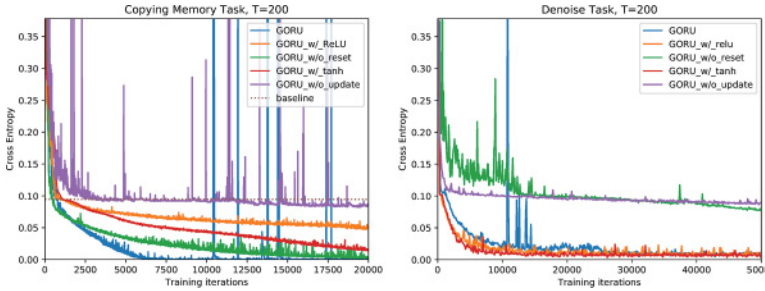

Figure 7: Copying memory task with delay time $T = 200$ and denoising task with delay time $T = 200$: cross-entropy on validation set for each iteration. In each task, we compare GORU, GORU with ReLU or tanh, and GORU without reset or update gate.

5 Conclusion

We have built a novel RNN that brings the benefits of orthogonal matrices to gated architectures: the gated orthogonal recurrent unit (GORU). By constraining the recurrence matrix of the GRU to be an orthogonal matrix and replacing the nonlinear activation with a modReLU, GORU gains the advantage of unitary or orthogonal RNNs since the gradient can pass through long time steps without exploding. Our empirical results showed that GORU is the only model we found that could solve both the synthetic copying task and the denoising task. Moreover, GORU is able to outperform GRU and LSTM in several benchmark tasks.

These results suggest that the GORU is the first step in bringing an explicit forgetting mechanism to the class of unitary or orthogonal RNNs. Our method demonstrates how to incorporate orthogonal matrices into a variety of neural network architectures. We are excited to open the gate of orthogonal matrix RNNs to real-world applications.

Appendix: Ablation Study on the Copying Task and Denoising Tasks

Here we show more results of the ablation study on the copying and denoising tasks, which test the long-term memory and forgetting ability, respectively, in Figure 7. For the copying task, the GORU without reset gate can still achieve optimum validation set performance in a small number of steps. GORU without an update gate gets stuck at baseline. GORU with other activation functions can still achieve optimum validation set performance, but the task requires significantly more training iterations. For the denoising task, the GORU with any conventional activation function can still achieve optimum validation set performance in a small number of

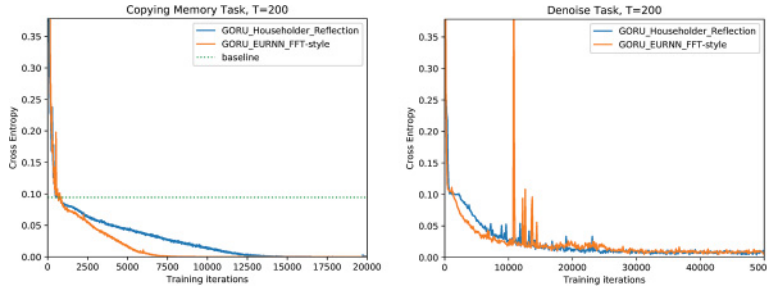

Figure 8: Copying memory task with delay time $T = 200$ and denoising task with delay time $T = 200$: cross entropy on validation set for each iteration. In each task, we compare the GORU with EURNN FFT-style parameterization and the GORU with Householder parameterization.

Table 5: Hyperparameters for the Copying Task.

Model	Learning Rate	Decay Rate	Batch Size
LSTM	—	—	—
GRU	—	—	—
uRNN	0.001	0.9	128
oRNN	—	—	—
EURNN	0.001	0.5	128
GORU	0.001	0.9	128

Note: The dash means the model fails the task with all possible hyperparameters.

steps. Both reset and update gates are crucial for this task. This matches our intuition that only the orthogonal recurrence matrix is responsible for solving the gradient vanishing or explosion problem. Gates are responsible for forgetting which is important in the denoising task.

In Figure 8, we show the performance of GORU with different parameterization methods. We find that GORU with both parameterizations is able to achieve optimum validation set performance.

A.1 Experiment Detail. Here we show the grid search detail for each model in each task. For synthetic tasks (copying memory task, denoising task, and algorithmic task), we choose hyperparameters with the lowest validation loss. For natural language tasks, we pick hyperparameters with the highest validation accuracy.

The best hyperparameters are listed in Tables 5 to 9.

Table 6: Hyperparameters for the Denoising Task.

Model	Learning Rate	Decay Rate	Batch Size
LSTM	0.001	0.9	128
GRU	0.001	0.9	128
uRNN	0.0001	0.9	128
oRNN	—	—	—
EURNN	0.0001	0.9	128
GORU	0.001	0.9	128

Note: The dash means the model fails the task with all possible hyperparameters.

Table 7: Hyperparameter for the Algorithmic Task.

Model	Learning Rate
LSTM	0.001
GRU	0.001
uRNN	0.0001
oRNN	0.0001
EURNN	0.0001
GORU	0.001

Table 8: Hyperparameters for the Language Modeling Task.

Model	Learning Rate	Decay Rate	Batch Size
LSTM	0.001	0.99	32
GRU	0.001	0.99	32
uRNN	0.0001	0.9	32
oRNN	0.0001	0.9	32
EURNN	0.0001	0.5	32
GORU	0.001	0.9	32

Table 9: Hyperparameters for the Speech Prediction Task.

Model	Learning Rate	Batch Size
LSTM	0.01	64
GRU	0.01	64
EURNN	0.001	32
oRNN	0.001	32
uRNN	0.001	32
GORU	0.001	32

Acknowledgments

This work was partially supported by the Army Research Office through the Institute for Soldier Nanotechnologies under contract W911NF-13-D0001, the National Science Foundation under grant CCF-1640012, and the Semiconductor Research Corporation under grant 2016-EP-2693-B.

References

Arjovsky, M., Shah, A., & Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on Machine Learning* (pp. 1120–1128). Berlin: Springer-Verlag.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Chan, W., Jaitly, N., Le, Q. V., & Vinyals, O. (2016). Listen, attend and spell. In *Proceedings of the IEEE Conference on Acoustics, Speech, and Signal Processing*. Piscataway, NJ: IEEE.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. In *Proceedings of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation* (pp. 103–111). Stroudsburg, PA: Association for Computational Linguistics.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014b). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 1724–1734). Stroudsburg, PA: Association for Computational Linguistics.

Chorowski, J. K., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). Attention-based models for speech recognition. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (pp. 577–585). Red Hook, NY: Curran.

Garofolo, J., Lamel, L., Fisher, W., Fiscus, J., Pallett, D., Dahlgren, N., & Zue, V. (1993). *Darpa timit acoustic-phonetic continous speech corpus* (Technical Report NISTIR 4930). Philadelphia: Linguistic Data Consortium.

Gers, F. (2001). *Long short-term memory in recurrent neural networks*. Ph.D. diss., Ecole Polytechnique Fédérale de Lausanne.

Gers, F. A., Schmidhuber, J., & Cummins, F. A. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, *12*, 2451–2471.

Graves, A., Wayne, G., & Danihelka, I. (2014). *Neural Turing Machines*. arXiv:1410.5401.

Henaff, M., Szlam, A., & LeCun, Y. (2016). Recurrent orthogonal networks and long-memory tasks. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd International Conference on Machine Learning* (pp. 2034–2042). Berlin: Springer-Verlag.

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen netzen*. *Diploma thesis Technische Universität München*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hyland, S., & Rtsch, G. (2017). Learning unitary operators with help from u(n). In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (pp. 2050–2058).

Jing, L., Shen, Y., Dubcek, T., John, P., Skirlo, S., LeCun, Y., . . . Soljacic, M. (2016). *Tunable efficient unitary neural networks (EUNN) and their application to RNNS*. arXiv:1612.05231.

Jordan, M. (1997). Serial order: A parallel distributed processing approach. *Advances in Psychology*, *121*, 471–495.

Karpathy, A., Johnson, J., & Li, F.-F. (2015). *Visualizing and understanding recurrent networks*. arXiv:1506.02078.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). *A simple way to initialize recurrent networks of rectified linear units*. arXiv:1504.00941.

Li, Y., Tarlow, D., Brockschmidt, M., & Zemel, R. (2015). *Gated graph sequence neural networks*. arXiv:1511.05493.

Marcus, M., Marcinkiewicz, M., & Santorini, B. (1993). Building a large annotated corpus of English: The penn treebank. *Computational Linguistics*, *19*, 313–330.

Mhammedi, Z., Hellicar, A. D., Rahman, A., & Bailey, J. (2017). Efficient orthogonal parameterisation of recurrent neural networks using householder reflections. In *Proceedings of the International Conference on Machine Learning*.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the International Conference on Machine Learning*.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems*, *27* (pp. 3104–3112). Red Hook, NY: Curran.

Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, *4*(2), 26–31.

Weston, J., Bordes, A., Chopra, S., Rush, A., van Merrinboer, B., Joulin, A., & Mikolov, T. (2015). *Towards AI-complete question answering: A set of prerequisite toy tasks*. arXiv:1502.05698.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., & Atlas, L. (2016). Full-capacity unitary recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*, *29* (pp. 4880–4888).

Received April 29, 2018; accepted December 8, 2018.