Orthogonal Gated Recurrent Unit With Neumann-Cayley Transformation

Vasily Zadorozhnyy
vasily.zadorozhnyy@sri.com
SRI International, Princeton, NJ 08540, U.S.A.

Edison Mucllar
edison.mucllar@uky.edu
Department of Mathematics, University of Kentucky, Lexington, KY 40506, U.S.A.

Cole Pospisil
pospisilcole@gmail.com
NSWC Crane, Crane, IN 47522, U.S.A.

Duc Nguyen
ducnguyen@utk.edu
The University of Tennessee, Knoxville, TN 37996, U.S.A.

Qiang Ye
qye3@uky.edu
Department of Mathematics, University of Kentucky, Lexington, KY 40506, U.S.A.

In recent years, using orthogonal matrices has been shown to be a promising approach to improving recurrent neural networks (RNNs) with training, stability, and convergence, particularly to control gradients. While gated recurrent unit (GRU) and long short-term memory (LSTM) architectures address the vanishing gradient problem by using a variety of gates and memory cells, they are still prone to the exploding gradient problem. In this work, we analyze the gradients in GRU and propose the use of orthogonal matrices to prevent exploding gradient problems and enhance long-term memory. We study where to use orthogonal matrices and propose a Neumann series–based scaled Cayley transformation for training orthogonal matrices in GRU, which we call Neumann-Cayley orthogonal GRU (NC-GRU). We present detailed experiments of our model on several synthetic and real-world tasks, which show that NC-GRU significantly outperforms GRU and several other RNNs.

Vasily Zadorozhnyy and Edison Mucllar contributed equally to this work. This work was done while Vasily Zadorozhnyy and Cole Pospisil were at the University of Kentucky.
1 Introduction

One of the preferred neural network models for working with sequential data is the recurrent neural network (RNN) (Rumelhart et al., 1986; Hopfield, 1982). RNNs can efficiently model sequential data through a hidden sequence of states. However, training vanilla RNNs has obstacles (Rumelhart et al., 1986; Hopfield, 1982), one of which is their susceptibility to vanishing and exploding gradients (Bengio et al., 1993). In the case of vanishing gradients, the optimization algorithm faces difficulty continuing to learn due to very small changes in the model parameters. In the case of exploding gradients, the training could overstep local minima, potentially causing instabilities such as divergence or oscillatory behavior. It may also lead to computational overflows.

There have been several works studying how to solve these problems. For example, gates have been introduced into the RNN architecture: long short-term memory (LSTM; Hochreiter & Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014). They can pass long-term information and help to overcome vanishing gradients. In practice, GRU and LSTM models are still prone to the problem of exploding gradients.

More recently, several RNN models have been proposed using unitary or orthogonal matrices for the recurrent weights (Helfrich et al., 2018; Dorado-Rojas et al., 2020; Vorontsov et al., 2017; Mhammedi et al., 2017; Jing et al., 2017; Arjovsky et al., 2016; Wisdom et al., 2016; Jing et al., 2019; Maduranga et al., 2019), along with methods to preserve those properties. Introducing such weights into RNN models brought new development into the RNN community. One of the main reasons behind this is a theoretical explanation of why the performance is better when using unitary or orthogonal weights (Arjovsky et al., 2016). The key step in these methods is preserving orthogonal or unitary properties at every training iteration. There have been several different techniques for updating the recurrent weights to preserve either orthogonal or unitary properties, including, for example, multiplicative updates (Wisdom et al., 2016), Givens rotations (Jing et al., 2017), Householder reflections (Mhammedi et al., 2017), Cayley transforms (Helfrich et al., 2018; Maduranga et al., 2019; Helfrich & Ye, 2020; Lezcano-Casado & Martínez-Rubio, 2019), and other similar ideas that have shown effective use of orthogonal or unitary matrices (Saxe et al., 2014; Arjovsky et al., 2016; Henaff et al., 2017; Hyland & Gunnar, 2017; Tagare, 2011; Vorontsov et al., 2017).

In this work, we study the benefits of applying orthogonal matrices to one of the most widely used RNN models, gated recurrent unit (GRU; Cho et al., 2014), both theoretically and experimentally. We analyze the gradients of the GRU loss, and based on this analysis, we propose the use of orthogonal matrices in several hidden state weights of the model. We introduce a Neumann series–based scaled Cayley transform for training the orthogonal weights. Our method uses a reliable and stable method of scaled Cayley
transforms, which was studied and used in Helfrich et al. (2018) and Madu-ranga et al. (2019) for training orthogonal weights for RNNs. In addition, we propose a Neumann series approximation of the matrix inverse inside the Cayley transform. Such substitution not only performs on the same or even better level as the traditional inverse (see section 4 for experiments) but also decreases computation time, which might be of particular assistance when working with larger models. We call our method Neumann-Cayley orthogonal gated recurrent unit (NC-GRU). Experiments show that the proposed method is more stable with faster convergence and produces better results on several synthetic and real-world tasks.

The rest of the article has the following sections. In section 2, we discuss some previous works that are most relevant to this article. In section 3, we introduce the NC-GRU method, together with the backpropagation method and gradient analysis of GRU (Cho et al., 2014) and NC-GRU. In section 4, we present experiments of our model on several synthetic and real-world problems: adding task (Hochreiter & Schmidhuber, 1997), copying task (Hochreiter & Schmidhuber, 1997), parenthesis task (Jing et al., 2019; Foerster et al., 2016), denoise task (Jing et al., 2019), as well as language models for the Penn Treebank (Marcus, Santorini, et al., 1993), and the WikiText-2 (Merity et al., 2016) datasets. Then, in section 5, we provide several experiments to justify the use of orthogonal matrices and the Neumann series, as well as which hidden states benefit from them. Finally, in section 6, we summarize our proposed method and contribution.

2 Related Work

Many models have been designed to improve classical RNN (Rumelhart et al., 1986; Hopfield, 1982) for sequential data. They include, for example, the establishment of gates (Hochreiter & Schmidhuber, 1997; Cho et al., 2014), normalization methods (Ioffe & Szegedy, 2015; Cooijmans et al., 2017; Wu & He, 2018; Ulyanov et al., 2017; Salimans & Kingma, 2016; Ba et al., 2016; Xu et al., 2019), and the introduction of unitary and orthogonal matrices into the RNN structure (Arjovsky et al., 2016; Jing et al., 2017; Mhammedi et al., 2017; Vorontsov et al., 2017; Jing et al., 2019; Dorado-Rojas et al., 2020). Improved training algorithms have been discussed in Nguyen et al. (2020). In this section, we discuss some of the works most relevant to our proposed method.

Unitary RNNs (uRNNs) (Arjovsky et al., 2016) presented an architecture that learns a unitary weight matrix. The construction of the recurrent weight matrix consists of a composition of diagonal matrices, reflection matrices in the complex domain, and Fourier transform. The uRNN model presented in Wisdom et al. (2016) is based on constrained optimization over the space of all unitary matrices rather than a product of parameterized matrices. Jing et al. (2017) is another work that uses the product of unitary matrices, the efficient unitary neural network (EUNN). The recurrent matrix in this
architecture is parameterized with products of Givens rotations. Also, the representation capacity of the unitary space is fully tunable and ranges from a subspace of unitary matrices to the entire unitary space.

Mhammedi et al. (2017) proposed orthogonal RNNs, or simply oRNNs, which involve the application of Householder reflections. Such parameterization of the transition matrix allows efficient training and maintains the orthogonality of the recurrent weights while training. The method introduced in Vorontsov et al. (2017) proposes a weight matrix factorization by bounding the matrix norms. Moreover, it controls the degree of gradient expansion during backpropagation. Besides that, this technique enforces orthogonality as well. GORU (Jing et al., 2019) presented an RNN, which is an extension of EUNN with a gating mechanism. It enforces orthogonality to hidden state loop matrices by using ideas from EURNN from Jing et al. (2017), where this matrix is decomposed into a sequence of 2-by-2 rotation matrices such that each of those rotation matrices contains one trainable rotation parameter. The results compared to GRU (Cho et al., 2014) are mixed, depending on the task. More recently, Dorado-Rojas et al. (2020) presented an embedding of a linear time-invariant system that contains Laguerre polynomials in the model.

2.1 Gated Recurrent Unit. We have studied in depth the gated recurrent unit (GRU) architecture proposed in Cho et al. (2014) as an alternative to the well-known LSTM (Hochreiter & Schmidhuber, 1997) cell. The structure of a GRU cell is

\begin{align*}
  r_t &= \sigma (W_r x_t + U_r h_{t-1} + b_r), \\
  u_t &= \sigma (W_u x_t + U_u h_{t-1} + b_u), \\
  c_t &= \Phi (W_c x_t + U_c (r_t \odot h_{t-1}) + b_c), \\
  h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t,
\end{align*}

where $W_r$, $W_u$, and $W_c$ are input weights in $\mathbb{R}^{n \times m}$; $U_r$, $U_u$, and $U_c$ are recurrent weights in $\mathbb{R}^{n \times n}$; and $b_r$, $b_u$, and $b_c$ are the bias parameters in $\mathbb{R}^n$. Here, $m$ represents the dimension of the input data, and $n$ represents the dimension of the hidden state. In equation 2.1, the activation functions $\sigma$ and $\Phi$ are sigmoid and hyperbolic tangent function (tanh), respectively, and $\odot$ is the Hadamard product. In addition, the initial hidden state $h_0$ is set to zero.

The main difference between GRU from LSTM is the implementation of the long-term memory not as a separate channel but inside the hidden state $h_t$ itself. GRU has a single gate $u_t$ that controls both forget and input gates. For example, if the output of $u_t$ is 1, then the forget gate is open, implying that the input gate is closed. Similarly, if $u_t$ is 0, the forget gate is closed and the input gate is open. This structure allows GRUs to discard random or insignificant information but at the same time grasp the important details.
2.2 Cayley Transform Orthogonal RNN. We have implemented and studied the effects of orthogonal matrices inside the GRU cell based on the Cayley transforms (Tagare, 2011). Some initial work to use the Cayley transform for orthogonal weights in RNNs was presented in Helfrich et al. (2018) together with the scoRNN model. This model introduced a skew-symmetric matrix $A$, which is used to define an orthogonal matrix $W$ via the scaled Cayley transform,

$$W = (I + A)^{-1} (I - A) D,$$

where matrix $D$ is a diagonal matrix of ones and negative ones, which scales the traditional Cayley transform (Tagare, 2011). Kahan (2006) proved that the matrix $D$, with a suitable choice on the number of negative ones, can avoid a potential problem of the eigenvalue(s) of $A$ being negative one(s), making matrix $I + A$ noninvertible. The number of negative ones in matrix $D$ can be considered as a tunable hyperparameter. Further, it guarantees that the skew-symmetric matrix $A$ that generates the orthogonal matrix will be bounded.

Helfrich et al. (2018) presents the following process to train the scoRNN model using scaled Cayley transforms:

$$A^{(k+1)} = A^{(k)} - \lambda \nabla_A L(U_{sco}(A^{(k)})),$$

$$U_{sco}^{(k+1)} = (I + A^{(k+1)})^{-1} (I - A^{(k+1)}) D,$$

where $\nabla_A L(U_{sco}(A))$ is computed using

$$\nabla_A L(U_{sco}(A)) = V^T - V,$$

with

$$V = (I + A)^{-T} \nabla_{U_{sco}} L(U_{sco}(A)) (D + U_{sco}^T),$$

in which $\nabla_{U_{sco}} L(U_{sco}(A))$ is computed using standard backpropogation methods.

Although rounding errors may accumulate over several repeated matrix multiplications, orthogonality in scoRNN (Helfrich et al., 2018) is maintained to the machine’s precision. This property helps to achieve significant improvements over other orthogonal/unitary RNNs for long sequences on several benchmark tasks (see the Experiments section in Helfrich et al., 2018, for details).

3 Efficient Orthogonal Gated Recurrent Unit

We now present an efficient orthogonal GRU. The proofs of all theoretical results presented are given in the supplementary materials section SM 7.
3.1 Gradient Analysis of Hidden States in GRU. Gradient behavior plays an important role in model training, convergence, stability, and, most important performance. However, when it comes to backpropagation through time for the GRU model from equation 2.1, the gradients of the loss function $L$ with respect to intermediate hidden states, weights, and biases can be found from the respective gradients of the final hidden state $h_T$, which is simplified to finding the gradient of $h_t$ with respect to $h_{t-1}$ for $t$ between 1 and $T$, namely,

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T} \prod_{t=i+1}^{T} \frac{\partial h_t}{\partial h_{t-1}}.$$  \hspace{1cm} (3.1)

Thus, to analyze the gradients, we consider the gradient of the hidden state $h_t$ with respect to the hidden state $h_{t-1}$, as well as its upper bound, in the following theorem:

**Theorem 1.** Let $h_{t-1}$ and $h_t$ be two consecutive hidden states from the GRU model stated in equation 2.1. Then

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \leq \alpha + \beta \| U_c \|_2,$$  \hspace{1cm} (3.2)

where

$$\alpha = \delta_u \left( \max_i \{ [h_{t-1}]_i \} + \max_i \{ [c_t]_i \} \right) \| U_u \|_2 + \max_i \{ (1 - [u_t]_i) \} \quad \text{(3.3)}$$

and

$$\beta = \max_i \{ [u_t]_i \} \left( \delta_r \| U_r \|_2 \max_i \{ [h_{t-1}]_i \} + \max_i \{ [r_t]_i \} \right).$$  \hspace{1cm} (3.4)

with constants $\delta_u$ and $\delta_r$ defined as follows:

$$\delta_u = \max_i \{ [u_t]_i (1 - [u_t]_i) \}$$  \hspace{1cm} (3.5)

and

$$\delta_r = \max_i \{ [r_t]_i (1 - [r_t]_i) \}.$$  \hspace{1cm} (3.6)

The following corollary provides some simple upper bounds for $\alpha$, $\beta$ obtained in theorem 1.

**Corollary 1.** For the hyperbolic tangent activation function in equation 2.1 (i.e., $\Phi = \tanh$), we have $\delta_u, \delta_r \leq \frac{1}{4}, [h_t]_i \leq 1$ for any $i$ and $t$, as well as
Orthogonal Gated Recurrent Unit

\[ \alpha \leq \frac{1}{2} \| U_u \|_2 + 1 \quad \text{and} \quad \beta \leq \frac{1}{4} \| U_r \|_2 + 1. \]  \hfill (3.7)

These bounds may be pessimistic because the gate elements may be expected to be close to 0 or 1. Consequently, the following corollary presents the relationship between the constants \( \alpha \) and \( \beta \) in theorem 1 when GRU’s gates are nearly closed or opened. Below, we use the notation \( x \lesssim y \) to denote that \( x \) is bounded by a quantity approximately equal to \( y \).

**Corollary 2.** When elements of GRU gates \( u_t \) and \( r_t \) are nearly either 0 or 1, then constants \( \alpha \) and \( \beta \) from theorem 1 satisfy the following inequality:

\[ \alpha + \beta \lesssim 2. \]  \hfill (3.8)

Moreover, if \( u_t \) and \( r_t \) are nearly either the zero vector or the vector of all ones, then

\[ \alpha + \beta \lesssim 1. \]  \hfill (3.9)

### 3.2 Neumann-Cayley Orthogonal Transformation.

Based on theorem 1 and corollary 2, we propose the use of orthogonal weights in the hidden parameters of GRU to obtain better-conditioned gradients. As discussed in section 2, different techniques have been proposed and used to initialize and preserve orthogonal weights while training—for example, Givens rotations (Jing et al., 2017) and Householder reflections (Mhammedi et al., 2017). In this work, we implement a version of the scaled Cayley transformation method discussed in section 2.2 with one key difference. The scaled Cayley transform method requires a calculation of the inverse of \( I + A^{(k)} \) to update the orthogonal matrix \( U^{(k)} \) in equation 2.4. When it comes to the computation of this inverse, classical numerical methods such as using LU-decomposition or solving the least squares problem can be implemented. These methods work well when the dimension of the matrix is small. However, if the matrix’s dimension is large, these methods are very expensive from both memory and computational time perspectives. Moreover, classical methods might overflow and not converge at all. We propose solving this possible complication using the Neumann series method to approximate the inverse of \( I + A^{(k)} \).

To derive the Neumann series approximation for the inverse of \( I + A^{(k)} \) in equation 2.4, we consider the following:

\[ (I + A^{(k)})^{-1} = (I + A^{(k-1)} - \delta A^{(k)})^{-1} \]  \hfill (3.10)

\[ = (I - (I + A^{(k-1)})^{-1} \delta A^{(k)})^{-1} (I + A^{(k-1)})^{-1} \]  \hfill (3.11)

\[ = \left( \sum_{i=0}^{\infty} ((I + A^{(k-1)})^{-1} \delta A^{(k)})^i \right) (I + A^{(k-1)})^{-1}, \]  \hfill (3.12)
where \( \delta A^{(k)} := \text{opt}_A (\nabla_A L = V^{(k)} T - V^{(k)}) \); here, \( \text{opt}_A \) includes a learning rate \( \lambda \) inside it. Note that the equality in equations 3.11 and 3.12 relies on the assumption that \( \| (I + A^{(k-1)})^{-1} \delta A^{(k)} \| < 1 \) for some operator norm \( \| \cdot \| \) (see Demmel, 1997, for details). We have conducted an ablation study that shows empirical evidence that this condition is indeed satisfied (see section 5.3 for details).

In our experiments, we have considered the first- and the second-order Neumann series approximations, estimating the series in equation 3.12 with two \( (i = 0, 1) \) and three \( (i = 0, 1, 2) \) terms, respectively. As expected, the model performs slightly better when using second-order approximation. However, it comes with a marginal increase in computational time (see section 5.1 for the ablation study regarding such approximations’ accuracy and computational time). Mathematically speaking, if we are using the second-order approximation, the error is of order \( \mathcal{O}((I + A^{(k-1)})^{-1} \delta A^{(k)})^3) \). Although this error is quite small, there is a chance that the errors from this approximation can accumulate and cause a loss of orthogonality. To avoid this issue, we recommend resetting orthogonality by computing the matrix inverse explicitly using a factorization method at the beginning of each epoch. However, it might be necessary to do it more often, particularly in the earlier training (e.g., every 100 iterations), due to more fluctuations in the gradients.

Algorithm 1 outlines the Neumann-Cayley orthogonal transformation method for training weight \( A \) and updating the corresponding orthogonal weight \( U \). It is important to note that during the initialization step, the weight \( A^{(0)} \) is defined to be skew-symmetric using the same initialization technique as in Helfrich et al. (2018), which is based on the idea from Henaff

---

**Algorithm 1: Update Rule for Orthogonal Weight \( U \)**

1. **Given:** \( D, A^{(0)}, U^{(0)}, \nabla_U L (U^{(0)} (A^{(0)})) \), \( \text{opt}_A \)
2. **Define:** \( \delta A^{(0)} := (I + A^{(0)})^{-1} \)
3. **for** \( k = 1, 2, \ldots \) **do**
   4. \( V^{(k)} := A^{(k-1)T} \nabla_U L (U^{(k-1)} (A^{(k-1)})) \left( D + U^{(k-1)T} \right) \)
   5. \( \delta A^{(k)} := \text{opt}_A \left( \nabla_A L = V^{(k)T} - V^{(k)} \right) \)
   6. \( A^{(k)} := A^{(k-1)} - \delta A^{(k)} \)
   7. \( \delta (k) := \left( I + A^{(k-1)} \delta A^{(k)} + \left( A^{(k-1)} \delta A^{(k)} \right)^2 \right) \delta (k-1) \)
   8. \( U^{(k)} := \delta (k) (I - A^{(k)}) D \)
9. **end**
et al. (2017). Then we apply the Cayley transform in $A^{(0)}$ to obtain $U^{(0)}$. Another peculiar detail that we want to point out is that algorithm 1 includes $\text{opt}_A$, the standard optimizer, such as SGD, RMSProp (Tieleman & Hinton, 2012), or Adam (Kingma & Ba, 2014), that takes $\nabla_{\mathcal{A}} L = V^{(k)} - V^{(k)}$ as an input. Moreover, the skew-symmetric property of the weight $\mathcal{A}$ and its gradient are preserved under such an optimizer.

### 3.3 Neumann-Cayley Orthogonal GRU

Finally, we introduce a Neumann-Cayley orthogonal GRU (NC-GRU) model that utilizes the proposed Neumann-Cayley orthogonal transform.

The structure of the NC-GRU cell is shown below:

$$
\begin{align*}
    r_t &= \sigma (W_r x_t + U_r (A_r) h_{t-1} + b_r), \\
    u_t &= \sigma (W_u x_t + U_u h_{t-1} + b_u), \\
    c_t &= \Phi (W_c x_t + U_c (A_c) (r_t \odot h_{t-1})). \\
    h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t,
\end{align*}
$$

(3.13)

here, $\sigma$, sigmoid function; $\odot$, Hadamard product; and $\Phi$, modReLU function defined in Arjovsky et al. (2016) as

$$
\Phi(x) := \text{modReLU}(x) := \text{sgn}(x) \cdot \text{ReLU}(|x| + b),
$$

(3.14)

with $b$ as a trainable bias.

We derive from our experiments that the best performance is achieved using orthogonality in $U_r$ and $U_l$ hidden weights. In addition, we present an ablation study in section 5.2 about the use and performance of orthogonal weights throughout the GRU model. Similar to the GRU cell, $W_r, W_u, W_c, U_u, b_r, b_u$, and $b$ are trainable parameters as well as $U_r$ and $U_c$ together with their associated weights $A_r$ and $A_c$, respectively. Moreover, all of them except $U_r$ with $A_r$ and $U_c$ with $A_c$ are trained using standard backpropagation algorithms such as stochastic gradient descent (SGD), RMSProp (Tieleman & Hinton, 2012), or Adam (Kingma & Ba, 2014) as in GRU (Cho et al., 2014), but $U_r, A_r, U_c$, and $A_c$ are trained using algorithm 1.

GORU (Jing et al., 2019) is a method that also employs orthogonal matrices in GRU. There are three differences from our model: (1) they use a variation of GRU where cell memory is generated by $c_t = \Phi(W_c x_t + r_t \odot (U_c h_{t-1}));$ (2) they impose orthogonality on $U_c$ only; and (3) they maintain orthogonality of $U_c$ by constructing a long product of Givens rotations. Our approaches lead to significant improvements over GORU in experiments.

As we mentioned, orthogonal weights lead to a better-conditioned gradient, and the following corollary summarizes this result.
Corollary 3. Let $h_{t-1}$ and $h_t$ be two consecutive hidden states from the NC-GRU model defined in equation 3.13. Then $\|U_r\|_2 = \|U_c\|_2 = 1$ and if elements of the gates $u_t$ and $r_t$ are nearly 0 or 1, then the following inequality is satisfied:

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 2. \quad (3.15)$$

Furthermore, if $u_t$ and $r_t$ are nearly either zero vector or vector of all ones,

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 1. \quad (3.16)$$

Bound 3.15 implies that the gradients are bounded by a constant independent of any model parameters. However, the constant 2 results in an exponentially growing upper bound for the loss gradients. Since the bound is a result of many inequalities that are not tight and, as a worst-case bound, it can be expected to be pessimistic. We further consider the cases when the gates are either approximately 0 or 1 vectors and obtain a bound of 1 in equation 3.16.

4 Experiments

We have performed various experiments to demonstrate the robustness and efficiency of our NC-GRU method. To this end, we apply NC-GRU to four commonly used synthetic tasks: parenthesis, denoise, adding, and copying tasks. In addition, we have considered nonsynthetic experiments, language modeling, with the character and word level tasks for the Penn Treebank (PTB) (Marcus, Santorini, et al., 1993) data set, as well as WikiText-2 (Merity et al., 2016).

All experiments were conducted for models with approximately equal numbers of trainable parameters (i.e., parameter-matching architecture). They were performed on a single NVIDIA Tesla V100 GPU with TensorFlow 1.13.2 (parenthesis, denoise, adding, and copying tasks), PyTorch 1.1.0 (character and word PTB, and WikiText-2), and Python 3.6.9. Codes for these experiments are available online at github.com/vasily789/NC-GRU and github.com/emucllari1/NC-GRU.

4.1 Parenthesis Task. This experiment derives from the descriptions in Jing et al. (2019) and Foerster et al. (2016). This task tests the ability of the network to remember the number of unmatched parentheses contained in our input data. The input data consist of 10 pairs of different types of parentheses combined with some noise data in between, and it is given as a one-hot encoding vector of length $T$. As stated in Jing et al. (2019), there are not more than 10 types of parentheses. The output data are given as a one
hot-encoding vector, counting the number of unpaired parentheses in the corresponding input data. The goal of our model is to forget the noise data and absorb information from the long-term dependencies related to the parentheses. These synthetic data require the model to develop a memory and to be able to select the most relevant information.

We present two versions of the NC-GRU model. The first one only has orthogonality in $U_r$ weight (NC-GRU($U_c$)); however, the second model uses orthogonality in both weights $U_r$ and $U_c$ (NC-GRU($U_r$, $U_c$)). Both models were trained using the Neumann series method with a reset every 50 iterations.

4.1.1 Implementation Details. All of the models consisted of a single-layer net with the following hidden dimensions for each model: LSTM (Hochreiter & Schmidhuber, 1997), 42; GRU (Cho et al., 2014), 50; scoRNN (Helfrich et al., 2018), 110; GORU (Jing et al., 2019), 64; and NC-GRU, 56. Furthermore, we trained all of the models for 200 epochs with a batch size of 16, and here were 20 and 40 negative ones for the $D$ matrix in scoRNN (Helfrich et al., 2018) and NC-GRU models. All models were trained using the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $10^{-3}$ including $A$ associated weights in scoRNN (Helfrich et al., 2018) and NC-GRU models. We conducted experiments using input length of $T = 100$ (see Figure 1a) and $T = 200$ (see Figure 1b).

4.1.2 Results. We observed and recorded the behavior of the five models on the parenthesis task when the input length is set to 100 and 200. Our results showed that both versions of NC-GRU models outperform GRU (Cho et al., 2014), LSTM (Hochreiter & Schmidhuber, 1997), scoRNN (Helfrich et al., 2018), and GORU (Jing et al., 2019) models with a significant gap (see Figure 1). On this task, the NC-GRU($U_r$, $U_c$) model performs better than the NC-GRU($U_c$); however, both perform better than the rest of the compared models. The minimum value of the loss attained during training is presented in Table 1.

4.2 Denoise Task. The denoise task (Jing et al., 2019) is another synthetic problem requiring filtering out the noise from a noisy sequence. This problem requires the forgetting ability of the network as well as learning long-term dependencies coming from the data (Jing et al., 2019). The input sequence of length $T$ contains 10 randomly located data points, and the other $T - 10$ points are considered noise data. These 10 points are selected from a dictionary $\{a_i\}_{i=0}^{n+1}$, where the first $n$ elements are data points, and the other two are the “noise” and the “marker,” respectively. The output data consist of the list of the data points from the input, and it should be output as soon as it receives the marker. The model task is to filter out the noise part and output the random 10 data points chosen from the input.
Figure 1: Parenthesis task results. NC-GRU($U_c$) denotes the NC-GRU model, equation 3.13, where the Neumann-Cayley method was only applied to the weight $U_c$; similarly, NC-GRU($U_r$, $U_c$) represents the NC-GRU model, equation 3.13, where both $U_r$ and $U_c$ weights were updated using the Neumann-Cayley method.

4.2.1 Implementation Details. We implemented one NC-GRU cell with a hidden size of 118 and the number of negative ones in the $D$ matrix to 50. The hidden size for the LSTM (Hochreiter & Schmidhuber, 1997) was 90; GRU (Cho et al., 2014), 100; scoRNN (Helfrich et al., 2018), 200; and GORU (Jing et al., 2019), 128. We implemented the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $10^{-3}$ to train all the models noted, including scoRNN (Helfrich et al., 2018) and NC-GRU $A$ weights. We trained all the models for 10,000 iterations with a batch size of 128. Similar to parenthesis task 4.1, we implemented the Neumann series method of approximation of
Table 1: Parenthesis Task Results.

| Sequence Length | Loss $\times 10^{-3}$ ↓ |
|-----------------|--------------------------|
|                 | $T = 100$                | $T = 200$                |
| LSTM            | 0.710 ± 0.005            | 14.706 ± 0.011           |
| GRU             | 31.066 ± 0.022           | 23.794 ± 0.397           |
| scoRNN          | 20.381 ± 0.234           | 19.180 ± 0.002           |
| GORU            | 1.065 ± 0.005            | 1.248 ± 0.012            |
| NC-GRU($U_c$) (ours) | 0.267 ± 0.001    | 0.217 ± 0.003           |
| NC-GRU($U_r, U_c$) (ours) | **0.188 ± 0.001** | 0.221 ± 0.003           |

Notes: Mean and a standard deviation of minimum attained loss values over three random seeds (↓ denotes the smaller, the better the result). All results are based on our tests, and the best results are highlighted in bold.

the $(I + A^{(k)})^{-1}$ when training the NC-GRU model, with the reset option to be applied every 50 iterations.

4.2.2 Results. Based on our experiments, NC-GRU($U_c$) and NC-GRU($U_r, U_c$) models significantly outperformed LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), scoRNN (Helfrich et al., 2018), and GORU (Jing et al., 2019) models on the denoised data (see Figure 2). Similar to what we observed from the parenthesis task, use of our orthogonal weights leads to better results. In addition, Table 2 provides a comparison of the attained minimum loss for each of the models. Furthermore, supplementary materials section SM 8 and Table 8 present the selected accuracies for this task.

4.3 Adding Problem. The adding problem is the third synthetic task that we considered. It was proposed in Hochreiter and Schmidhuber (1997) for recurrent networks. Our implementation of this problem is a variation of the original problem. The input in the network is a two-dimensional sequence of length $T$. In the first dimension, we have a sequence of all zeros except for two randomly placed ones, one in the first half of the sequence and one in the second. In the second dimension, we have a sequence of randomly selected numbers chosen uniformly from the interval $[0, 1)$. The goal of the adding task is to take the second-dimension numbers from the positions corresponding to the ones in the first dimension and output their sum. The sizes of the training and testing sets are 100,000 and 10,000, respectively.

4.3.1 Implementation Details. In this task, we worked with a single-layer cell with a hidden dimension set to be 68 for LSTM (Hochreiter & Schmidhuber, 1997), 70 for GRU (Cho et al., 2014), 190 for scoRNN (Helfrich et al., 2018), 128 for GORU (Jing et al., 2019), and 80 for NC-GRU. The negative ones for the $D$ matrix inside scoRNN and NC-GRU were set to 95 and 43,
Figure 2: Denoise task results.

Table 2: Denoise Task Results.

| Sequence Length | $T = 200$        | $T = 400$        |
|-----------------|------------------|------------------|
| LSTM            | $10.367 \pm 0.050$ | $5.551 \pm 0.001$ |
| GRU             | $9.852 \pm 0.001$  | $5.261 \pm 0.004$  |
| scoRNN          | $7.303 \pm 0.075$  | $4.023 \pm 0.002$  |
| GORU            | $3.709 \pm 0.196$  | $2.131 \pm 0.001$  |
| NC-GRU($U_r$) (ours) | $2.258 \pm 0.272$  | $0.952 \pm 0.001$  |
| NC-GRU($U_r$, $U_c$) (ours) | $2.278 \pm 0.110$ | $1.294 \pm 0.004$  |

Notes: Mean and a standard deviation of minimum attained loss values over three random seeds. All results are based on our tests, and the best results are highlighted in bold. See supplementary materials section SM 8 for corresponding results on accuracy.
We used the Adam optimizer (Kingma & Ba, 2014) with a learning rate of $10^{-3}$ in all of the experiments, including the training of the $A$ weight in scoRNN (Helfrich et al., 2018) and NC-GRU models. All models were trained for 10 epochs with a batch size of 50 and evaluated every 100 iterations. Neumann series were applied during the training of NC-GRU models, with a reset option implemented every 50 iterations.

4.3.2 Results. Figures 3a and 3b present the performances of all the interested methods on the tests for sequences of length 200 and 400, respectively. In this experiment, our NC-GRU($U_c$) model where the Neumann-Cayley method applied only to the $U_c$ weight showed the best performance out of all the compared models, including our NC-GRU($U_r, U_c$) model which produced comparable or marginally better results than others. Table 3 presents minimum attained validation loss values.

Figure 3: Adding problem results.
Table 3: Adding Task Results.

| Sequence Length | $T = 200$ | $T = 400$ |
|-----------------|-----------|-----------|
| LSTM            | 22.302 ± 11.884 | 145.089 ± 9.192 |
| GRU             | 3.137 ± 0.007 | 2.1927539 ± 0.016 |
| scoRNN          | 55.144 ± 0.035 | 16,301.960 ± 0.019 |
| GORU            | 4.829 ± 0.004 | 2.927 ± 0.007 |
| NC-GRU($U_c$) (ours) | **0.918 ± 0.009** | **1.009 ± 0.001** |
| NC-GRU($U_c, U_r$) (ours) | 4.086 ± 0.026 | 5.412 ± 0.016 |

Notes: Mean and a standard deviation of minimum attained loss values over three random seeds. All results are based on our tests, and the best results are highlighted in bold.

4.4 Copying Problem. The copying problem was proposed in Hochreiter and Schmidhuber (1997) as a synthetic task for testing recurrent neural networks. The setup of this problem consists of a string of 10 random digits that are sampled uniformly from the integers 1 through 8 and then fed into the recurrent model. These 10 digits are followed by a string of $T$ zeros and a digit 9, which marks the start of a string of 9 zeros. Therefore, the total length of the fed string is $T + 20$. The objective of the task is to output the initial sequence of 10 random digits beginning at the marker location, copying the first 10 elements of the sequence in order. For the evaluation of the model, the cross-entropy loss is used with an expected cross-entropy baseline of $10 \log(8) / (T + 20)$, which represents the random selection of digits 1 through 8 after the 9.

4.4.1 Implementation Details. We carried out our experiments on single-layer models with hidden sizes for each model to match the number of trainable parameters: 68 for LSTM (Hochreiter & Schmidhuber, 1997), 78 for GRU (Cho et al., 2014), 190 for scoRNN (Helfrich et al., 2018), 100 for GORU (Jing et al., 2019), and 96 for NC-GRU. All the models were trained using the Adam (Kingma & Ba, 2014) optimizer with the learning rate of $10^{-3}$ except the $A$ matrix in scoRNN (Helfrich et al., 2018) and NC-GRU models, where the Adam (Kingma & Ba, 2014) optimizer learning rate was set to $10^{-4}$. The training iterations and the batch size were set to 10,000 and 50, respectively. The negative ones for scaling matrix $D$ were set to 95 and 80 for scoRNN (Helfrich et al., 2018) and NC-GRU models, respectively. All of the models were evaluated every 50 iterations. While training the model using the NC-GRU models, we applied the Neumann series similarly to the other experiments with the reset option every 20 iterations.

4.4.2 Results. Figures 4a and 4b present the performance of the models for the copying problem with string sizes of $T = 1,000$ and $T = 2,000$, respectively. We have included a baseline for each of the string sizes.
Figure 4: Copying problem results.

Moreover, Table 4 shows the minimum validation loss values. Both NC-GRU models perform on the same level while outperforming LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), and GORU (Jing et al., 2019) models by a noticeable margin. However, for this task, scoRNN (Helfrich et al., 2018) performed better than all of the other models. Furthermore, supplementary materials section SM 8 and Table 9 present the selected accuracies for this task.

Language modeling is one of many natural language processing tasks. It is the development of probabilistic models that are capable of predicting the next character or word in a sequence using information that has preceded it. We have conducted two experiments on the language modeling with the character- and word-level tasks for the Penn Treebank (PTB) data set (Marcus, Marcinkiewicz, et al., 1993), both of which were based on the AWD-LSTM model (Merity et al., 2018).
Table 4: Copying Task Results.

| Sequence Length | Loss $\times 10^{-2}$ | $T = 1,000$ | $T = 2,000$ |
|-----------------|----------------------|------------|------------|
| LSTM            | 2.039 ± 0.001        | 1.030 ± 0.001 |
| GRU             | 1.278 ± 0.162        | 0.896 ± 0.001 |
| scoRNN          | 0.005 ± 0.001        | 0.028 ± 0.001 |
| GORU            | 1.577 ± 0.001        | 0.875 ± 0.003 |
| NC-GRU$(U_c)$ (ours) | 0.884 ± 0.001 | 0.447 ± 0.002 |
| NC-GRU$(U_c, U_r)$ (ours) | 0.919 ± 0.002 | 0.568 ± 0.001 |

Notes: Mean and a standard deviation of minimum attained loss values over three random seeds. All results are based on our tests, and the best results are highlighted in bold. See supplementary materials section SM 8 for corresponding results on accuracy.

4.5 Character-Level Penn Treebank. For this task, models were tested on their suitability for language modeling tasks using the character-level Penn Treebank data set (Marcus, Santorini, et al., 1993). This data set is a collection of English-language Wall Street Journal articles. It consists of a vocabulary of 10,000 words with other words replaced as $<$unk$>$, resulting in approximately 6 million characters that are divided into 5.1 million, 400,000, and 450,000 character sets for training, validation, and testing, respectively, with a character alphabet size of 50. The goal of the character-level language modeling task is to predict the next character given the preceding sequence of characters.

4.5.1 Implementation Details. For this task, we have considered three-layer models, where hidden dimensions for the NC-GRU$(U_c)$ model were set to (430, 1000, 430). All the dropout coefficients were set to 0.15. The learning rate of $5 \times 10^{-4}$ and the Adam (Kingma & Ba, 2014) optimizer were used to train the whole model, including matrix $A$. The number of negative ones for matrix $D$ for every layer was set to 50. The LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), and scoRNN (Helfrich et al., 2018) models were trained similarly with hidden dimensions being (350, 880, 350), (415, 950, 415), and (500, 2000, 500), respectively. The batch size for all the experiments was set to 32, and the backpropagation through time (bptt) window of 100 was used for all the models.

4.5.2 Results. Models were evaluated using the bits per character (bpc) metric. NC-GRU outperformed scoRNN, GRU, and even LSTM models, as shown in Table 5.

4.6 Word-Level Penn Treebank. We also tested our proposed Neumann-Cayley method on the word-level Penn Treebank data set.
Table 5: Character-Level PTB Results.

| Model          | bpc ↓          |
|----------------|---------------|
| LSTM           | 1.980 ± 0.003 |
| GRU            | 1.449 ± 0.016 |
| scoRNN         | 1.600 ± 0.009 |
| NC-GRU($U_c$) (ours) | 1.385 ± 0.002 |

Notes: Mean and a standard deviation of evaluated bits-per-character (bpc) for every model over three random seeds. All results are based on our tests, and the best result is highlighted in bold.

Table 6: Word-Level PTB Results.

| Model          | PPL ↓          |
|----------------|---------------|
| LSTM           | 78.93†        |
| GRU            | 92.48† (80.90 ± 0.13*) |
| scoRNN         | 123.48 ± 0.39 |
| NC-GRU($U_c$) (ours) | 77.05 ± 0.04  |

Notes: Mean and a standard deviation of evaluated perplexity (PPL) over three random seeds. †Result obtained from our experiments; *Result quoted from (Bai et al., 2018); the best result is highlighted in bold.

(Marcus, Marcinkiewicz, et al., 1993). The data set takes the same underlying corpus as the character-level task but with tokens representing words instead of characters. This results in a smaller data set with a larger vocabulary size, with 888,000, 70,000, and 79,000 words as training, validation, and testing sets, and a vocabulary of 10,000 words.

4.6.1 Implementation Details. We trained NC-GRU with three layers with dimensions (400, 1150, 400). We have a learning rate set to $5 \times 10^{-4}$ for both the $A$ matrix and the rest of the model, optimized using Adam (Kingma & Ba, 2014). The dropout after the NC-GRU cell has a coefficient of 0.4, the embedding layer dropout has a coefficient of 0.4, and the output dropout has a coefficient of 0.25. The number of negative ones for matrix $D$ for each layer was 50.

4.6.2 Results. Results were evaluated using the perplexity (PPL) metric and are shown in Table 6. We show improved performance over both baseline GRU and LSTM models.
Table 7: WikiText-2 Results.

| Model                      | PPL ↓         |
|----------------------------|---------------|
| LSTM                      | 65.76 ± 1.05  |
| GRU                       | 102.33 ± 0.65 |
| scoRNN                    | 167.75 ± 1.23 |
| NC-GRU($U_c$) (ours)      | 92.56 ± 0.04  |

Notes: Mean and a standard deviation of evaluated perplexity (PPL) for every model over three random seeds. All results are based on our tests, and the best result is highlighted in bold.

4.7 WikiText-2. Finally, we conducted experiments on testing our NC-GRU model on the WikiText-2 data set. This data set was introduced in Merity et al. (2016). It consists of preprocessed Wikipedia articles that keep their original punctuation and symbols and is almost twice the size of the Penn Treebank data set. The WikiText-2 data set has a vocabulary size of 33,278 words and contains approximately 2.2 million words—2 million for the training set and 200,000 for the validation and test sets, respectively. This word-level language modeling task aims to predict the subsequent word given the previous string of words.

4.7.1 Implementation Details. A three-layer NC-GRU model is implemented to train our proposed method in the WikiText-2 data set. The corresponding dimensions are 500, 1550, and 500. The learning rate for both the $A$ matrix and the model training is set to $7 \times 10^{-4}$ with the Adam (Kingma & Ba, 2014) optimizer. The batch size is 80, and the model is trained for 100 epochs. An $L_2$ regularization with coefficient 0.0008 and a temporal activation regularization with coefficient 0.012 is also applied in our model. The number of negative ones for every layer is set to 100, 310, and 100, respectively.

4.7.2 Results. Results are shown in Table 7, where the perplexity (PPL) metric is evaluated. Our proposed architecture outperforms scoRNN and GRU by a significant margin. However, the LSTM model seems to be better suited for this language modeling task.

5 Ablation Studies

This section considers several ablation studies that help us justify using the Neumann series, orthogonal matrices, and scaled Cayley transforms.
Figure 5: Inverse versus Neumann series. In this graph, Inverse represents the NC-GRU($U_c$) model with $(I + A^{(k-1)})^{-1}$ computed using the least squares method and Neumann $i$ represents the NC-GRU($U_c$) model with $(I + A^{(k-1)})^{-1}$ computed using the Neumann series method of order $i$. Average and standard derivation over three randomly picked seeds.

5.1 Neumann Series Method versus Inverse. In this experiment, we study the sharpness of approximating $(I + A^{(k-1)})^{-1}$ with the Neumann series in the NC-GRU($U_c$) model on the parenthesis task (see section 4.1 for implementation details and a description of NC-GRU($U_c$) model). We consider using the Neumann series approximation of orders 1, 2, and 3 and compare them to the least squares method for taking a matrix inverse, one of the widely used methods from deep learning libraries such as TensorFlow, PyTorch, and NumPy.

Our experiments showed that the Neumann series approximation method achieves better results than the classical least squares method for finding the matrix inverse. Figure 5 shows that the Neumann series method of order 2 performs marginally better than the order 1 and order 3 Neumann series methods and significantly better than the least squares method.

We have also compared the time it takes to train our models using the methods mentioned. Figure 6 shows the time it takes to train one epoch of the NC-GRU($U_c$) model on a character level PTB data set (see section 4.5.1 for the implementation details) on a single NVIDIA Tesla V100 GPU using inverse (least square) and Neumann 1, Neumann 2, and Neumann 3 methods with various hidden dimensions: 430-1000-430 (used in our experiment in section 4.5), 128-256-128, and 1024-2048-1024.

The observed behavior appears to be quite general, and we have conducted all of the experiments in section 4 using the second-order Neumann series method.

5.2 Orthogonality in the Hidden Weights of GRU. For our second ablation study, we studied the effect of Neumann-Cayley transformation
orthogonal weights applied to the hidden units inside the GRU cell, equation 2.1. We considered three models. The first model only had $U_c$ weight replaced with orthogonal weight preserved by the Neumann-Cayley method; we previously called such a model NC-GRU($U_c$). The second model, NC-GRU($U_r$, $U_c$), had two weights, $U_c$ and $U_r$, replaced with Neumann-Cayley transformation orthogonal weights, and finally, the third model, NC-GRU($U_r$, $U_u$, $U_c$), had all three weights $U_r$, $U_u$, and $U_c$ replaced.

The results are shown in Figure 7. We see that implementing one or two orthogonal weight models, NC-GRU($U_c$) or NC-GRU($U_r$, $U_c$), would have the most benefits, while the three orthogonal weights model, NC-GRU($U_r$, $U_u$, $U_c$), does not perform as well. This can also be seen in section 4, where both one and two orthogonal weight models are used.

5.3 Necessary Conditions for the Neumann Series Method. As noted in section 3.2, the assumption that allowed us to use the Neumann series is 
\[ \| (I + A^{(k-1)})^{-1} \delta A^{(k)} \| < 1, \] where $\| \cdot \|$ satisfies $\| AB \| \leq \| A \| \| B \|$ for some matrices $A$ and $B$ of appropriate dimensions.

For this experiment, we chose the spectral norm, that is, the $L_2$-norm, and recorded values $\| (I + A^{(k-1)})^{-1} \delta A^{(k)} \|_2$ during training the NC-GRU($U_c$)
Orthogonal Gated Recurrent Unit

Figure 7: Orthogonal matrices for different combinations of hidden weights. NC-GRU one hidden parameter orthogonal versus three hidden parameters orthogonal.

Figure 8: Norm condition for the Neumann approximation. Values of $\|(I + A^{(k-1)})^{-1}\delta A^{(k)}\|_2$ using the NC-GRU($U_c$) model with second-order the Neumann series approximation method.

model on the parenthesis task with the second-order Neumann series approximation. We observed that the norm values were changing during the training; however, they are well below one and satisfy the necessary condition, as depicted in Figure 8.

6 Conclusion

This article thoroughly analyzes the gated recurrent unit (GRU) model’s gradients. Based on this analysis, we introduced the Neumann-Cayley GRU model, NC-GRU. Our model incorporates orthogonal weights in the hidden states of the GRU model, which are trained using a newly proposed method of the Neumann-Cayley transformation for maintaining the desired
orthogonality in those weights. We have conducted experiments demonstrating the superiority of our proposed method, outperforming GRU, LSTM, s2oRNN, and GORU models on most synthetic and real-world (natural language processing) tasks. Moreover, we conducted several ablation studies that empirically confirmed our theoretical results.

Acknowledgments

We thank the University of Kentucky Center for Computational Sciences and Information Technology Services Research Computing for their support and use of the Lipscomb Compute Cluster and associated research computing resources. This research was supported in part by NSF under grants DMS-2053284, DMS-2151802, DMS-2208314, IIS-2327113, and the University of Kentucky Start-up fund.

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).
Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv:1507.06450.
Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.0127.
Bengio, Y., Frasconi, P., & Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In Proceedings of 1993 IEEE International Conference on Neural Networks (pp. 1183–1195).
Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv:1406.1078.
Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., & Courville, A. C. (2017). Recurrent batch normalization. In Proceedings of the 5th International Conference on Learning Representations.
Demmel, J. W. (1997). Applied numerical linear algebra. SIAM.
Dorado-Rojas, S. A., Vinzamuri, B., & Vanfretti, L. (2020). Orthogonal Laguerre current neural networks. In Proceedings of the 34th Conference on Neural Information Processing Systems.
Foerster, J. N., Gilmer, J., Chorowski, J., Sohl-Dickstein, J., & Sussillo, D. (2016). Input switched affine networks: An MN architecture designed for interpretability. arXiv:1611.09434.
Helfrich, K., Willmott, D., & Ye, Q. (2018). Orthogonal recurrent neural networks with scaled Cayley transform. In Proceedings of ICML 2018.
Helfrich, K., & Ye, Q. (2020). Eigenvalue normalized recurrent neural networks for short term memory. In Proceedings of the AAAI Conference on Artificial Intelligence (pp. 4115–4122).
Henaff, M., Szlam, A., & LeCun, Y. (2017). Recurrent orthogonal networks and long-memory tasks. In Proceedings of the 33rd International Conference on Machine Learning.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. 10.1162/neco.1997.9.8.1735

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, 79(8), 2554–2558. 10.1073/pnas.79.8.2554

Hyland, S. L., & Gunnar, R. (2017). Learning unitary operators with help from u(n). In Proceedings of the 31st AAAI Conference on Artificial Intelligence (pp. 2050–2058).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

Jing, L., Gulcehre, C., Peurifoy, J., Shen, Y., Tegmark, M., Soljacic, M., & Bengio, Y. (2019). Gated orthogonal recurrent units: On learning to forget. Neural Computation, 31(4), 765–783. 10.1162/neco_a_01174

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., . . . Soljačić, M. (2017). Tunable efficient unitary neural networks (EUNN) and their application to RNNs. In Proceedings of the 34th International Conference on Machine Learning (pp. 1733–1741).

Kahan, W. (2006). Is there a small skew Cayley transform with zero diagonal? Linear Algebra and Its Applications, 417(2), 335–341. 10.1016/j.laa.2005.08.027

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Lezcano-Casado, M., & Martínez-Rubio, D. (2019). Cheap orthogonal constraints in neural networks: A simple parameterization of the orthogonal and unitary group. In K. Chaudhuri & R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning (pp. 3794–3803).

Maduranga, K. D., Helfrich, K., & Ye, Q. (2019). Complex unitary recurrent neural networks using scaled Cayley transform. In Proceedings of the AAAI Conference on Artificial Intelligence (pp. 4528–4535).

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1993). Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19(2), 313–330.

Marcus, M. P., Santorini, B., & Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics, 19(2), 313–330.

Merity, S., Keskar, N. S., & Socher, R. (2018). Regularizing and optimizing LSTM language models. In Proceedings of the International Conference on Learning Representations.

Merity, S., Xiong, C., Bradbury, J., & Socher, R. (2016). Pointer sentinel mixture models. arXiv:1609.07843.

Mhammedi, Z., Hellicar, A. D., Rahman, A., & Bailey, J. (2017). Efficient orthogonal parameterisation of recurrent neural networks using householder reflections. In Proceedings of ICML 2017 (pp. 2401–2409).

Nguyen, T., Baraniuk, R., Bertozzi, A., Osher, S., & Wang, B. (2020). Momentum-RNN: Integrating momentum into recurrent neural networks. In H. Larochelle,
M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems*, 33 (pp. 1924–1936). Curran.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning internal representations by error propagation*. MIT Press.

Salimans, T., & Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems*, 29. Curran.

Saxe, A. M., McClelland, J. L., & Ganguli, S. (2014). Exact solutions to nonlinear dynamics of learning in deep linear neural networks. arXiv:1312.6120.

Tagare, H. D. (2011). *Notes on optimization on Stiefel manifolds*. Technical report, Yale University.

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).

Ulyanov, D., Vedaldi, A., & Lempitsky, V. (2017). *Instance normalization: The missing ingredient for fast stylization*. arXiv:1607.08022.

Vorontsov, E., Trabelsi, C., Kadoury, S., & Pal, C. (2017). On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning* (pp. 3570–3578).

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

Wu, Y., & He, K. (2018). Group normalization. In *Proceedings of the European Conference on Computer Vision*.

Xu, J., Sun, X., Zhang, Z., Zhao, G., & Lin, J. (2019). Understanding and improving layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems*, 32. Curran.

Received May 9, 2023; accepted June 24, 2024.