# Orthogonal Gated Recurrent Unit with Neumann-Cayley Transformation

**Vasily Zadorozhnyy**[1,*] **Edison Mucllari**[1,*] **Cole Pospisil**[1]
**Duc Nguyen**[1] **Qiang Ye**[1]

[1] Department of Mathematics, University of Kentucky, Lexington, Kentucky, USA

{vasily.zadorozhnyy, edison.mucllari, Cole.Pospisil, ducnguyen, qye3}@uky.edu

[*] These authors contributed equally to this work

**Keywords:** RNN, Neumann-Cayley Transformation, GRU, NC-GRU, Parenthesis, Denoise, Adding, Copying, Penn TreeBank, Wikitext-2

## Abstract

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 usage 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, or simply 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.

# 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 have 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 and 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 and Ye, 2020; Lezcano-Casado and Martínez-Rubio, 2019), and other similar ideas that have shown effective usage of orthogonal or unitary matrices (Saxe et al., 2014; Arjovsky et al., 2016; Henaff et al., 2017; Hyland and 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 usage 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 utilizes a reliable and stable method of Scaled Cayley transforms, which was studied and used in (Helfrich et al., 2018; Maduranga 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 come of particular assistance when working with larger models. We call our method *Neumann-Cayley Orthogonal Gated Recurrent Unit* or *NC-GRU* for simplicity. 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 paper has the following layout. In Section 2, we discuss some previous works that are most relevant to this paper. In Section 3, we introduce Neumann-Cayley Orthogonal Gated Recurrent Unit (NC-GRU) method together with the back-propagation 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 and Schmidhuber, 1997), Copying task (Hochreiter and Schmidhuber, 1997), Parenthesis task (Jing et al., 2019; Foerster et al., 2016), and Denoise task (Jing et al., 2019) as well as character and word level Penn TreeBank prediction tasks (Marcus et al., 1993b). 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 and Schmidhuber, 1997; Cho et al., 2014), normalization methods (Ioffe and Szegedy, 2015; Cooijmans et al., 2017; Wu and He, 2018; Ulyanov et al., 2017; Salimans and Kingma, 2016; Ba et al., 2016; Xu et al., 2019), and the introduction of unitary and orthogonal matrices into 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), etc. Improved training algorithms have been discussed in (Nguyen et al., 2020). In this section, we will 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. EUNN (Jing et al., 2017) is another work that utilizes the product of unitary matrices. 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.

Work by (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 matrix 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 contain 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 (GRU)

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

$$\begin{aligned} r_t &= \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \\ u_t &= \sigma\left(W_u x_t + U_u h_{t-1} + b_u\right) \\ c_t &= \Phi\left(W_c x_t + U_c\left(r_t \odot h_{t-1}\right) + b_c\right) \\ h_t &= \left(1 - u_t\right) \odot h_{t-1} + u_t \odot c_t \end{aligned} \tag{1}$$

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 (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

$$W = (I + A)^{-1} (I - A) D, \tag{2}$$

where matrix $D$ is a diagonal matrix of ones and negative ones, which scales the traditional Cayley transform (Tagare, 2011). It is proved in (Kahan, 2006) 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 $I + A$ non-invertible. 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:

$$A^{(k+1)} = A^{(k)} - \lambda \nabla_A L\left(U_{sco}\left(A^{(k)}\right)\right) \tag{3}$$

$$U_{sco}^{(k+1)} = \left(I + A^{(k+1)}\right)^{-1}\left(I - A^{(k+1)}\right) D \tag{4}$$

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

$$\nabla_A L\left(U_{sco}\left(A\right)\right) = V^T - V, \tag{5}$$

with

$$V = \left(I + A\right)^{-T} \nabla_{U_{sco}} L\left(U_{sco}\left(A\right)\right)\left(D + U_{sco}^T\right) \tag{6}$$

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

Even though 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 more 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 a very important role in model training, convergence, stability, and most importantly performance. However, when it comes to backpropagation through time for the GRU model from (1), the gradients of the loss function $\mathcal{L}$ 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,

$$\frac{\partial \mathcal{L}}{\partial h_i} = \frac{\partial \mathcal{L}}{\partial h_T} \prod_{t=i+1}^{T} \frac{\partial h_t}{\partial h_{t-1}}. \tag{7}$$

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 (1). Then*

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right|_2 \leq \alpha + \beta \left|\left|U_c\right|\right|_2 \tag{8}$$

*where*

$$\alpha = \delta_u \left(\max_i \left\{[h_{t-1}]_i\right\} + \max_i \left\{[c_t]_i\right\}\right) \left|\left|U_u\right|\right|_2 + \max_i \left\{(1 - [u_t]_i)\right\} \tag{9}$$

*and*

$$\beta = \max_i \left\{[u_t]_i\right\} \left(\delta_r \left|\left|U_r\right|\right|_2 \max_i \left\{[h_{t-1}]_i\right\} + \max_i \left\{[r_t]_i\right\}\right), \tag{10}$$

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

$$\delta_u = \max_i \left\{[u_t]_i \left(1 - [u_t]_i\right)\right\} \tag{11}$$

*and*

$$\delta_r = \max_i \left\{[r_t]_i \left(1 - [r_t]_i\right)\right\}. \tag{12}$$

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

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

$$\alpha \leq \frac{1}{2} ||U_u||_2 + 1 \quad \textit{and} \quad \beta \leq \frac{1}{4} ||U_r||_2 + 1. \tag{13}$$

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 $\beta$ presented in Theorem 1 when GRU's gates are nearly closed or opened. Below we use a 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:*

$$\alpha + \beta \lesssim 2. \tag{14}$$

*Moreover, if* $u_t$ *and* $r_t$ *are nearly either the zero vector or the vector of all ones, then*

$$\alpha + \beta \lesssim 1. \tag{15}$$

## 3.2 Neumann-Cayley Orthogonal Transformation

Based on Theorem 1 and Corollary 2, we propose the usage of orthogonal weights in the hidden parameters of GRU to obtain better-conditioned gradients. As discussed in section 2, there have been different techniques proposed and used to initialize and preserve orthogonal weights while training, for example, Givens rotations (Jing et al., 2017),

Householder reflections (Mhammedi et al., 2017) etc. In this work, we implement a version of the Scaled-Cayley transformation method discussed in 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 (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 (4), we consider the following:

$$\left(I + A^{(k)}\right)^{-1} = \left(I + A^{(k-1)} - \delta A^{(k)}\right)^{-1} \tag{16}$$

$$= \left(I - \left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right)^{-1} \left(I + A^{(k-1)}\right)^{-1} \tag{17}$$

$$= \left(\sum_{i=0}^{\infty} \left(\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right)^{i}\right) \left(I + A^{(k-1)}\right)^{-1} \tag{18}$$

where $\delta A^{(k)} := opt_A \left(\nabla_A \mathcal{L} = {V^{(k)}}^T - V^{(k)}\right)$, here $opt_A$ includes a learning rate $\lambda$ inside of it. Note that the equality in Equations (17) and (18) relies on the assumption that $\left|\left|\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right|\right| < 1$ for some operator norm $||\cdot||$; see (Demmel, 1997) for more details. We have conducted an ablation study that shows empirical evidence that this condition is indeed satisfied; see section 5.3 for more details.

In our experiments, we have considered the first and the second-order Neumann

series approximations, estimating the series in (18) 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 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 $\mathcal{O}\left(\left(\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right)^3\right)$. Even though 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:** Update Rule for Orthogonal Weight $U$

**Given:** $D$, $A^{(0)}$, $U^{(0)}$, $\nabla_U \mathcal{L}\left(U^{(0)}\left(A^{(0)}\right)\right)$, $opt_A$

**Define:** $\tilde{A}^{(0)} := \left(I + A^{(0)}\right)^{-1}$

**for** $k = 1, 2, \ldots$ **do**

$\quad V^{(k)} := \tilde{A}^{(k-1)^T} \nabla_U \mathcal{L}\left(U^{(k-1)}\left(A^{(k-1)}\right)\right)\left(D + U^{(k-1)^T}\right)$

$\quad \delta A^{(k)} := opt_A\left(\nabla_A \mathcal{L} = V^{(k)^T} - V^{(k)}\right)$

$\quad A^{(k)} := A^{(k-1)} - \delta A^{(k)}$

$\quad \tilde{A}^{(k)} := \left(I + \tilde{A}^{(k-1)}\delta A^{(k)} + \left(\tilde{A}^{(k-1)}\delta A^{(k)}\right)^2\right)\tilde{A}^{(k-1)}$

$\quad U^{(k)} := \tilde{A}^{(k)}\left(I - A^{(k)}\right) D$

**end**

We present Algorithm 1 that 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 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 $opt_A$, which is the standard optimizer such as SGD, RMSProp (Tieleman and Hinton, 2012), or Adam (Kingma and Ba, 2014), etc., that takes $\nabla_A \mathcal{L} = V^{(k)^T} - V^{(k)}$ as an input. Moreover, the skew-symmetric property of the weight $A$ and its gradient are preserved under such an optimizer.

### 3.3 Neumann-Cayley Orthogonal GRU (NC-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:

$$\begin{aligned} r_t &= \sigma\left(W_r x_t + U_r(A_r) h_{t-1} + b_r\right) \\ u_t &= \sigma\left(W_u x_t + U_u h_{t-1} + b_u\right) \\ c_t &= \Phi\left(W_c x_t + U_c(A_c)\left(r_t \odot h_{t-1}\right)\right) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t \end{aligned} \tag{19}$$

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

$$\Phi(x) := \text{modReLU}(x) := \text{sgn}(x) \cdot \text{ReLU}\left(|x| + b\right) \tag{20}$$

with $b$ as a trainable bias.

We derive from our experiments that the best performance is achieved using orthogonality in $U_c$ and $U_r$ hidden weights. In addition, we present an ablation study in 5.2 about the usage and performance of orthogonal weights throughout the GRU model. Similar to the GRU Cell, $W_r$, $W_u$, $W_c$, $U_u$, $b_r$, $b_u$, $b$, 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 and Hinton, 2012), or Adam (Kingma and Ba, 2014) similarly 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 $c_t = \Phi\left(W_c x_t + r_t \odot \left(U_c h_{t-1}\right)\right)$; 2) they impose orthogonality on $U_c$ only; 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 (19). 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:*

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right|_2 \lesssim 2. \tag{21}$$

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

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 1. \tag{22}$$

Bound (21) 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 (22).

# 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 non-synthetic experiments, Language Modeling with the character and word level tasks for the Penn TreeBank (PTB) (Marcus et al., 1993b) dataset as well as WikiText-2 (Merity et al., 2016).

All the 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.

## 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 consists 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 is given as 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. This synthetic data requires 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 $U_c$ weight (NC-GRU($U_c$)); however, the second model utilizes 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.

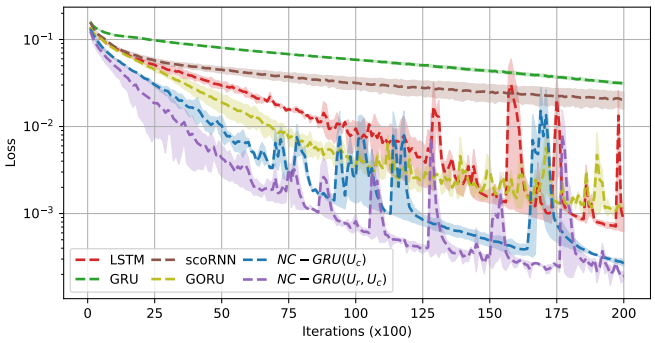
*Implementation Details:* All of the models consisted of a single layer net with the following hidden dimensions for each model: LSTM (Hochreiter and Schmidhuber, 1997) - 42, GRU (Cho et al., 2014) - 50, scoRNN (Helfrich et al., 2018) - 110, GORU (Jing et al., 2019) - 64, and NC-GRU - 56. Furthermore,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

Table 1: **Parenthesis Task Results:** Mean and a standard deviation of minimum attained loss values over 3 random seeds (↓ - denotes the smaller, the better the result). All results are based on our tests, and the best results are highlighted in **bold**.

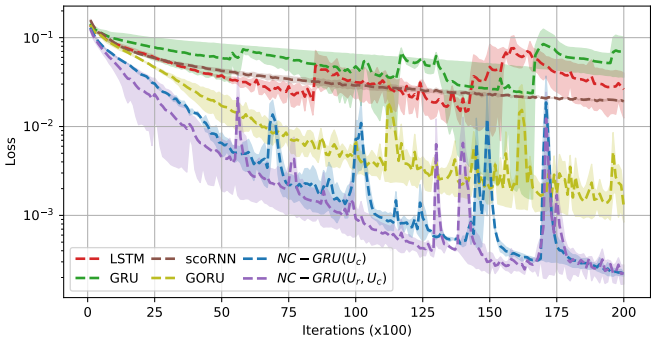
| | Loss $\times 10^{-3}$ ↓ | |
|---|---|---|
| Sequence Length | $T = 100$ | $T = 200$ |
| LSTM | $0.710 \pm 0.005$ | $14.706 \pm 0.011$ |
| GRU | $31.066 \pm 0.022$ | $23.794 \pm 0.397$ |
| scoRNN | $20.381 \pm 0.234$ | $19.180 \pm 0.002$ |
| GORU | $1.065 \pm 0.005$ | $1.248 \pm 0.012$ |
| NC-GRU($U_c$)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

2018) and NC-GRU models. We conducted experiments using input length of $T = 100$, see Figure 1a, and $T = 200$, see Figure 1b.

*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 and 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.




(a) $T = 100$




(b) $T = 200$

Figure 1: **Parenthesis Task Results:** NC-GRU$(U_c)$ denotes the NC-GRU model (19) 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 (19) where both $U_r$ and $U_c$ weights were updated using the Neumann-Cayley method.

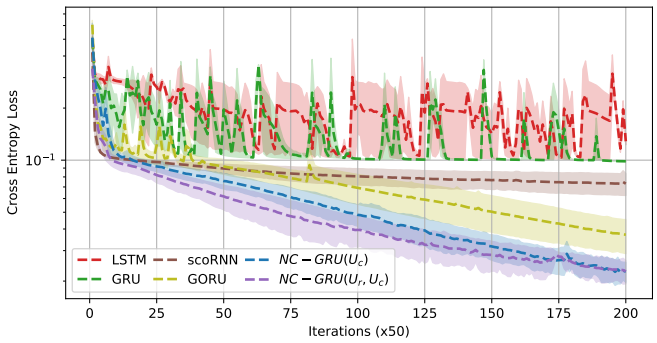
## 4.2 Denoise Task

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

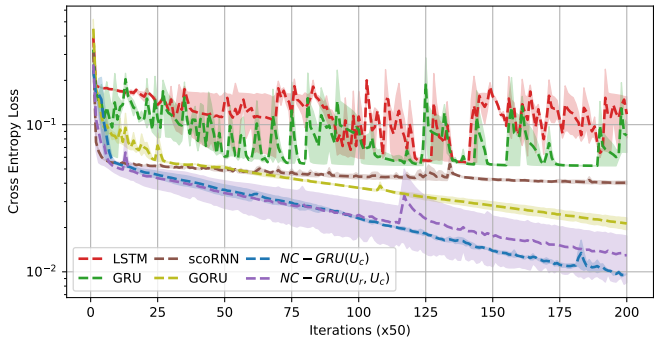
"$noise$" and the "$marker$" respectively. The output data consists of the list of the data points from the input, and it should be outputted as soon as it receives the "$marker$". The model task is to filter out the noise part and output the random 10 data points chosen from the input.

*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 and 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 Adam optimizer (Kingma and Ba, 2014) with a learning rate of $10^{-3}$ to train all the aforementioned models, 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 the Parenthesis Task 4.1, we implemented the Neumann series method of approximation of the $\left(I + A^{(k)}\right)^{-1}$ when training the NC-GRU model with the reset option to be applied every 50 iterations.

*Results:* Based on our experiments, NC-GRU($U_c$) and NC-GRU($U_r, U_c$) models significantly outperformed LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014), scoRNN (Helfrich et al., 2018), and GORU (Jing et al., 2019) models on the denoise data; see Figure 2. Similar to what we observed from the parenthesis task, the usage of two orthogonal weights leads to better results. In addition, Table 2 provides a comparison of the attained minimum loss for each of the models. ACCURACY REFERENCE




(a) $T = 200$




(b) $T = 400$

Figure 2: **Denoise Task Results**

## 4.3 Adding Problem

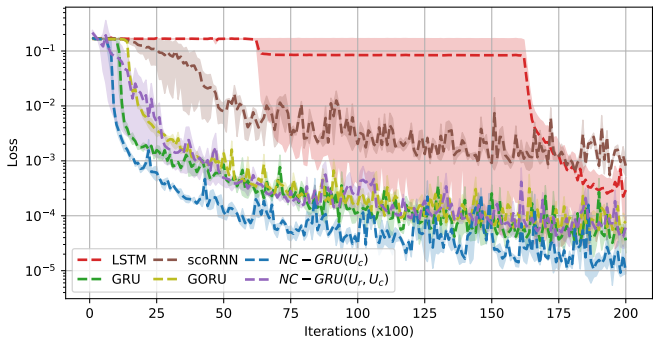
The adding problem is the third synthetic task that we considered. It was proposed in (Hochreiter and Schmidhuber, 1997) for recurrent networks. Our implementation of this problem is a variation of the original problem. The input in the network is a 2-dimensional sequence of length $T$. In the first 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 size of the training and testing sets are $100,000$

Table 2: **Denoise Task Results:** Mean and a standard deviation of minimum attained loss values over 3 random seeds (↓ - denotes the smaller, the better the result). All results are based on our tests, and the best results are highlighted in **bold**.

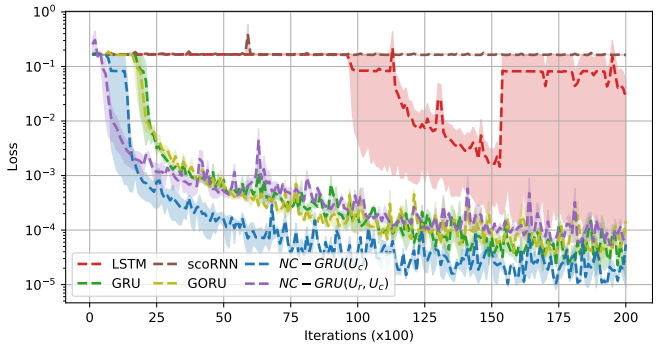
| | Loss $\times 10^{-2}$ ↓ | |
|---|---|---|
| Sequence Length | $T = 200$ | $T = 400$ |
| LSTM | $10.367 \pm 0.050$ | $5.551 \pm 0.001$ |
| GRU | $9.852 \pm 0.001$ | $5.261 \pm 0.004$ |
| scoRNN | $7.303 \pm 0.075$ | $4.023 \pm 0.002$ |
| GORU | $3.709 \pm 0.196$ | $2.131 \pm 0.001$ |
| NC-GRU($U_c$) (**ours**) | $\mathbf{2.258 \pm 0.272}$ | $\mathbf{0.952 \pm 0.001}$ |
| NC-GRU($U_c, U_r$) (**ours**) | $2.278 \pm 0.110$ | $1.294 \pm 0.004$ |

and $10,000$, respectively.

*Implementation Details:* In this task, we worked with a single layer cell with a hidden dimension set to be 68 for LSTM (Hochreiter and 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 number of negative ones for the $D$ matrix inside scoRNN and NC-GRU was set to 95 and 43, respectively. We used the Adam optimizer (Kingma and 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.




(a) $T = 200$




(b) $T = 400$

Figure 3: **Adding Problem Results**

*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.

## 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

Table 3: **Adding Task Results:** Mean and a standard deviation of minimum attained loss values over 3 random seeds (↓ - denotes the smaller, the better the result). All results are based on our tests, and the best results are highlighted in **bold**.

| | Loss $\times 10^{-5}$ ↓ | |
|---|---|---|
| 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 |

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 ten 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 $\frac{10 \log(8)}{T+20}$ which represents the random selection of digits 1-8 after the 9.

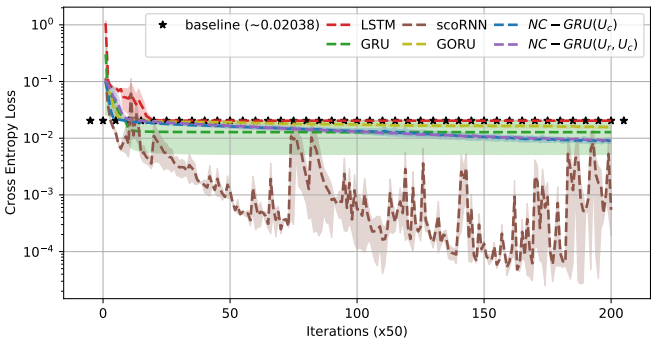
*Implementation Details:* We carried out our experiments on single-layer models with hidden sizes for each model to match the number of trainable parameters was 68 for LSTM (Hochreiter and Schmidhuber, 1997), 78 for GRU (Cho et al., 2014),

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 Adam (Kingma and Ba, 2014) optimizer with the learning rate of $10^{-3}$ except $A$ matrix in scoRNN (Helfrich et al., 2018) and NC-GRU models where Adam (Kingma and Ba, 2014) optimizer learning rate was set to $10^{-4}$. The number of training iterations and the batch size were set to 10,000 and 50, respectively. The number of negative ones for scaling matrix $D$ was 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.

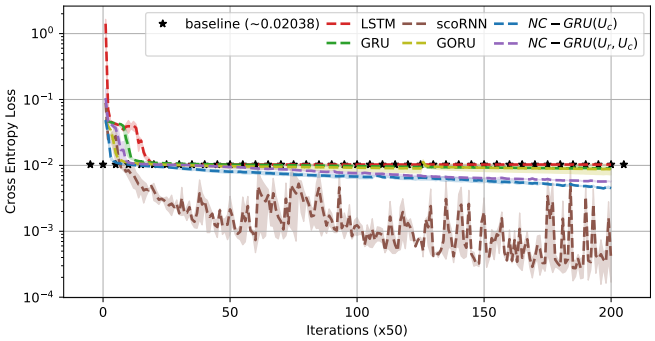
*Results:* Figures 4a and 4b present the performance of the aforementioned models for the Copying Problem with string sizes of $T = 1,000$ and $T = 2,000$, respectively. We have included a baseline for each of the string sizes. Moreover, Table 4 shows minimum validation loss values. Both NC-GRU models perform on the same level while outperforming LSTM (Hochreiter and 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.

ACCURACY REFERENCE

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) dataset (Marcus et al., 1993a), both of which were based on the




(a) $T = 1,000$




(b) $T = 2,000$

Figure 4: **Copying Problem Results**

AWD-LSTM model (Merity et al., 2018).

## 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 dataset (Marcus et al., 1993b). This dataset is a collection of English-language Wall Street Journal articles. The dataset 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 thousand, and 450 thousand character sets for training, validation, and testing, respectively with a character alphabet size of 50. The goal of the character-level Language Modeling task

Table 4: **Copying Task Results:** Mean and a standard deviation of minimum attained loss values over 3 random seeds (↓ - denotes the smaller, the better the result). All results are based on our tests, and the best results are highlighted in **bold**.

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

is to predict the next character given the preceding sequence of characters.

*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 $5\times10^{-4}$ and the Adam (Kingma and Ba, 2014) optimizer were used to train the whole model, including the matrix $A$. The number of negative ones for matrix $D$ for every layer was set to $50$. LSTM (Hochreiter and Schmidhuber, 1997), GRU (Cho et al., 2014), and scoRNN (Helfrich et al., 2018) model 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 be $32$, and the Back Propagation Through Time (`bptt`) window of 100 was used for all the models.

Table 5: **Character Level PTB Results:** Mean and a standard deviation of evaluated bits-per-character (bpc) for every model over 3 random seeds. (↓ - denotes the smaller, the better the result). All results are based on our tests, and the best result is highlighted in **bold**.

| | bpc ↓ |
|---|---|
| LSTM | $1.980 \pm 0.003$ |
| GRU | $1.449 \pm 0.016$ |
| scoRNN | $1.600 \pm 0.009$ |
| NC-GRU($U_c$) (**ours**) | $\mathbf{1.385 \pm 0.002}$ |

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

*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 and Ba, 2014). The dropout after the

Table 6: **Word Level PTB Results:** Mean and a standard deviation of evaluated perplexity (PPL) over 3 random seeds (↓ - denotes the smaller, the better the result). * - result obtained from our experiments; † - result quoted from (Bai et al., 2018); the best result is highlighted in **bold**.

| | 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** |

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.

*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.

## 4.7 WikiText-2

Finally, we conducted experiments on testing our NC-GRU model on the WikiText-2 dataset. The WikiText-2 dataset 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 dataset. The WikiText-2 dataset 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.

Table 7: **WikiText-2 Results:** Mean and a standard deviation of evaluated perplexity (PPL) for every model over 3 random seeds (↓ - denotes the smaller, the better the result). All results are based on our tests, and the best result is highlighted in **bold**.

| | 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 |

This word-level Language Modeling task aims to predict the subsequent word given the previous string of words.

*Implementation Details:* A three-layer NC-GRU model is implemented to train our proposed method in the WikiText-2 dataset. 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 Adam (Kingma and 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.

*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.

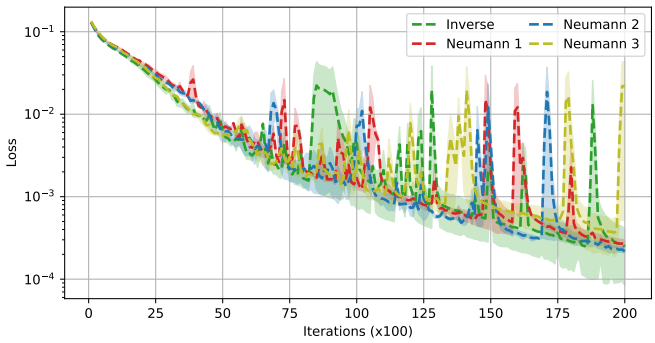



Figure 5: **Inverse vs Neumann series:** In this graphs, `Inverse` represents NC-GRU($U_c$) model with $\left(I + A^{(k-1)}\right)^{-1}$ computed using the Least Squares method and `Neumann i` represents NC-GRU($U_c$) model with $\left(I + A^{(k-1)}\right)^{-1}$ computed using the Neumann series method of order $i$. Average and standard derivation over 3 randomly picked seeds.

# 5 Ablation Studies

This section considers several ablation studies that help us justify using the Neumann series, orthogonal matrices, and Scaled-Cayley transforms.

## 5.1 Neumann Series Method vs Inverse

In this experiment, we study the sharpness of approximating $\left(I + A^{(k-1)}\right)^{-1}$ with the Neumann series in the NC-GRU($U_c$) model on the Parenthesis task; see 4.1 for implementation details and 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

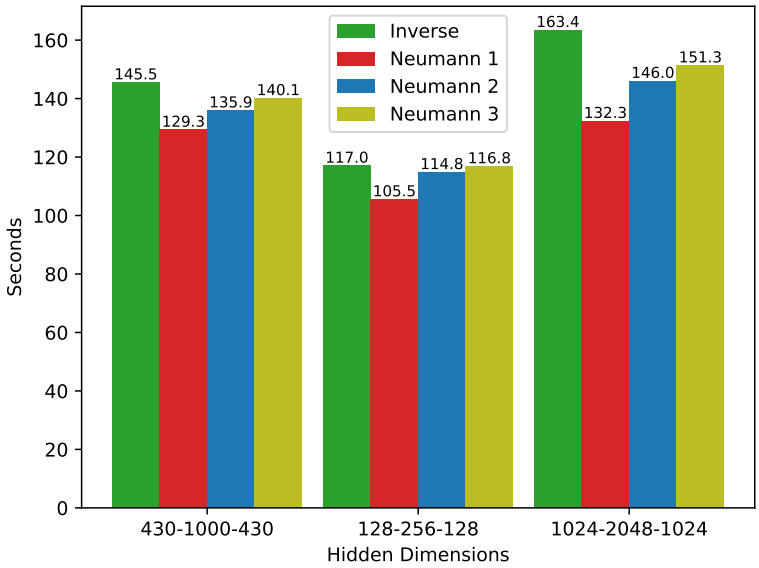



Figure 6: **Computational time comparisons:** The amount of compute time (in seconds) it takes to train one epoch of NC-GRU($U_c$) model on Character Level PTB dataset using a single NVIDIA® Tesla® V100 GPU with various hidden dimensions.

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 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 dataset (see Implementation Details of 4.5) on a single NVIDIA® Tesla® V100 GPU using Inverse (Least Square), Neumann 1, Neumann 2, and Neumann 3 methods with various hidden dimensions: 430-1000-430 (used in our experiment in 4.5), 128-256-128, and 1024-2048-1024.

The observed behavior appears to be quite general, and we have conducted all of

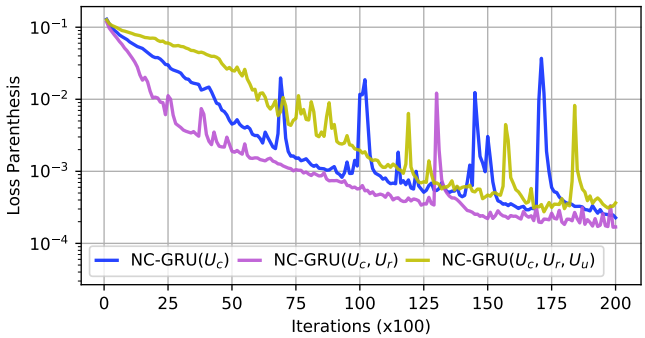



Figure 7: **Orthogonal matrices for different combinations of hidden weights:** NC-GRU one hidden parameter orthogonal vs. three hidden parameters orthogonal

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 have studied the effect of Neumann-Cayley transformation orthogonal weights applied to the hidden units inside the GRU cell (1). We have 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.

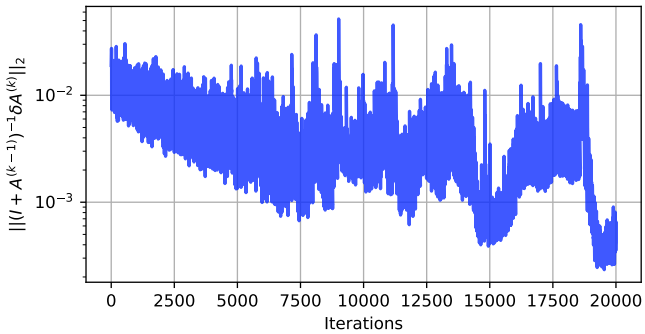



Figure 8: **Norm condition for the Neumann approximation:** Values of $\left|\left|\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right|\right|_2$ using NC-GRU($U_c$) model with second-order Neumann series approximation method.

### 5.3 Necessary Conditions for the Neumann Series Method

As it was mentioned in 3.2, the assumption that allowed us to use the Neumann series is $\left|\left|\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right|\right| < 1$ where $||\cdot||$ satisfies $||AB|| \leq ||A||\,||B||$ for some matrices $A$ and $B$ of appropriate dimensions.

For this experiment, we chose the Spectral Norm, i.e., $L_2$-norm, and recorded values $\left|\left|\left(I + A^{(k-1)}\right)^{-1} \delta A^{(k)}\right|\right|_2$ during training the NC-GRU($U_c$) 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 paper thoroughly analyzes the Gated Recurrent Unit (GRU) model's gradients. Based on this analysis, we introduced the Neumann-Cayley Gated Recurrent Unit 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 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 ( MOST OF THE TASKS mainly in the nlp domain), scoRNN, and GORU models on different tasks. Moreover, we conducted several ablation studies that empirically confirmed our theoretical results.

## Acknowledgments

We would like to 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., and Bengio, Y. (2016). Unitary evolution recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1120–1128. JMLR.org.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Bai, S., Kolter, J. Z., and Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.

Bengio, Y., Frasconi, P., and Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. In *Proceedings of 1993 IEEE International Conference on Neural Networks (ICNN '93)*, pages 1183–1195, San Francisco, CA. IEEE Press.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., and Courville, A. C. (2017). Recurrent batch normalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Demmel, J. W. (1997). *Applied Numerical Linear Algebra*. SIAM.

Dorado-Rojas, S. A., Vinzamuri, B., and Vanfretti, L. (2020). Orthogonal laguerre

recurrent neural networks. *34th Conference on Neural Information Processing Systems*.

Foerster, J. N., Gilmer, J., Chorowski, J., Sohl-Dickstein, J., and Sussillo, D. (2016). Input switched affine networks: An rnn architecture designed for interpretability.

Helfrich, K., Willmott, D., and Ye, Q. (2018). Orthogonal Recurrent Neural Networks with Scaled Cayley Transform. *In Proceedings of ICML 2018, Stockholm, Sweden, PMLR 80*.

Helfrich, K. and Ye, Q. (2020). Eigenvalue normalized recurrent neural networks for short term memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:4115–4122.

Henaff, M., Szlam, A., and LeCun, Y. (2017). Recurrent orthogonal networks and long-memory tasks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML 2017)*, volume 48.

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

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.

Hyland, S. L. and Gunnar, R. (2017). Learning unitary operators with help from u(n). In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAI 2017)*, pages 2050–2058, San Francisco, CA.

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

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

Jing, L., Shen, Y., Dubcek, T., Peurifoy, J., Skirlo, S., LeCun, Y., Tegmark, M., and Soljačić, M. (2017). Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1733–1741. JMLR.org.

Kahan, W. (2006). Is there a small skew cayley transform with zero diagonal? *Linear Algebra and its Applications*, 417(2):335–341. Special Issue in honor of Friedrich Ludwig Bauer.

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

Lezcano-Casado, M. and Martínez-Rubio, D. (2019). Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3794–3803. PMLR.

Maduranga, K. D., Helfrich, K., and Ye, Q. (2019). Complex unitary recurrent neural

networks using scaled cayley transform. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4528–4535.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. (1993a). Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330.

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

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

Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2016). Pointer sentinel mixture models.

Mhammedi, Z., Hellicar, A. D., Rahman, A., and Bailey, J. (2017). Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of ICML 2017*, volume 70, pages 2401–2409. PMLR.

Nguyen, T., Baraniuk, R., Bertozzi, A., Osher, S., and Wang, B. (2020). Momentumrnn: Integrating momentum into recurrent neural networks. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1924–1936. Curran Associates, Inc.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, page 318–362. MIT Press.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

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

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

Tieleman, T. and 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., and Lempitsky, V. (2017). Instance normalization: The missing ingredient for fast stylization.

Vorontsov, E., Trabelsi, C., Kadoury, S., and Pal, C. (2017). On orthogonality and learning recurrent networks with long term dependencies. ICML'17, page 3570–3578. JMLR.org.

Wisdom, S., Powers, T., Hershey, J., Le Roux, J., and Atlas, L. (2016). Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems 29*, pages 4880–4888.

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

Xu, J., Sun, X., Zhang, Z., Zhao, G., and Lin, J. (2019). Understanding and improving layer normalization. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

# Orthogonal Gated Recurrent Unit with Neumann-Cayley Transformation

## Supplementary Materials

## SM 7 Proofs of Theorem 1 and Corollaries 1, 2, 3

In this section, we present the proofs for the theorem and corollaries stated in section 3.

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

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right|_2 \leq \alpha + \beta \left|\left|U_c\right|\right|_2 \tag{23}$$

*where*

$$\alpha = \delta_u \left(\max_i \left\{[h_{t-1}]_i\right\} + \max_i \left\{[c_t]_i\right\}\right) \left|\left|U_u\right|\right|_2 + \max_i \left\{(1 - [u_t]_i)\right\} \tag{24}$$

*and*

$$\beta = \max_i \left\{[u_t]_i\right\} \left(\delta_r \left|\left|U_r\right|\right|_2 \max_i \left\{[h_{t-1}]_i\right\} + \max_i \left\{[r_t]_i\right\}\right), \tag{25}$$

*with constants $\delta_u$ and $\delta_r$ be as follows:*

$$\delta_u = \max_i \left\{[u_t]_i \left(1 - [u_t]_i\right)\right\} \tag{26}$$

*and*

$$\delta_r = \max_i \left\{[r_t]_i \left(1 - [r_t]_i\right)\right\}. \tag{27}$$

*Proof.* Since $h_t$ depends on $u_t$ and $c_t$ (which is also depends on $r_t$), we will start by finding $\frac{\partial r_t}{\partial h_{t-1}}$, and $\frac{\partial u_t}{\partial h_{t-1}}$, and $\frac{\partial c_t}{\partial h_{t-1}}$ as well as corresponding bounds of these gradients. Recall that

$$r_t = \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) \tag{28}$$

then

$$\frac{\partial r_t}{\partial h_{t-1}} = diag\left(\sigma'\left(W_r x_t + U_r h_{t-1} + b_r\right)\right) U_r = diag\left(r_t \odot (1 - r_t)\right) U_r \tag{29}$$

since $\sigma'(x) = \sigma(x)\,(1 - \sigma(x))$. Moreover, we can bound $\frac{\partial r_t}{\partial h_{t-1}}$ in the following way

$$\left|\left|\frac{\partial r_t}{\partial h_{t-1}}\right|\right|_2 \leq \delta_r \left|\left|U_r\right|\right|_2 \tag{30}$$

with

$$\delta_r := \max_i \left\{[r_t]_i \left(1 - [r_t]_i\right)\right\}. \tag{31}$$

The constant $\delta_r$ is defined to be the largest entry of the vector $r_t \odot (1 - r_t)$ and it is bounded by $\frac{1}{4}$. Similarly,

$$\frac{\partial u_t}{\partial h_{t-1}} = diag\left(\sigma'\left(W_u x_t + U_u h_{t-1} + b_u\right)\right) U_u = diag\left(u_t \odot (1 - u_t)\right) U_u, \tag{32}$$

and

$$\left|\left|\frac{\partial u_t}{\partial h_{t-1}}\right|\right|_2 \leq \delta_u \left|\left|U_u\right|\right|_2 \tag{33}$$

with

$$\delta_u := \max_i \left\{[u_t]_i \left(1 - [u_t]_i\right)\right\}. \tag{34}$$

The definition of the vector $c_t$ is

$$c_t = \Phi\left(W_c x_t + U_c\left(r_t \odot h_{t-1}\right) + b_c\right) \tag{35}$$

and

$$\frac{\partial c_t}{\partial h_{t-1}} = diag\left(\Phi'\left(W_c x_t + U_c(r_t \odot h_{t-1}) + b_c\right)\right) U_c \left(diag\left(h_{t-1}\right) \frac{\partial r_t}{\partial h_{t-1}} + diag\left(r_t\right)\right), \tag{36}$$

with $\Phi'$ applied entrywise, and

$$\left|\left|\frac{\partial c_t}{\partial h_{t-1}}\right|\right|_2 \leq \max_i \left\{\left[\Phi'\left(W_c x_t + U_c\left(r_t \odot h_{t-1}\right) + b_c\right)\right]_i\right\}$$

$$\times \left( \left|\left| \frac{\partial r_t}{\partial h_{t-1}} \right|\right|_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\} \right) ||U_c||_2 \tag{37}$$

$$\leq \left( \delta_r \, ||U_r||_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\} \right) ||U_c||_2 \tag{38}$$

649 since $\Phi'$ is bounded by 1. Finally,

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot c_t \tag{39}$$

650 with

$$\frac{\partial h_t}{\partial h_{t-1}} = -diag\,(h_{t-1}) \frac{\partial u_t}{\partial h_{t-1}} + diag\,(1 - u_t) + diag\,(c_t) \frac{\partial u_t}{\partial h_{t-1}} + diag\,(u_t) \frac{\partial c_t}{\partial h_{t-1}} \tag{40}$$

and

$$\begin{aligned} \left|\left| \frac{\partial h_t}{\partial h_{t-1}} \right|\right|_2 \leq \max_i \{[h_{t-1}]_i\} \left|\left| \frac{\partial u_t}{\partial h_{t-1}} \right|\right|_2 + \max_i \{(1 - [u_t]_i)\} \\ + \max_i \{[c_t]_i\} \left|\left| \frac{\partial u_t}{\partial h_{t-1}} \right|\right|_2 + \max_i \{[u_t]_i\} \left|\left| \frac{\partial c_t}{\partial h_{t-1}} \right|\right|_2 . \end{aligned} \tag{41}$$

Furthermore, using (30), (33), and (38), we get

$$\begin{aligned} \left|\left| \frac{\partial h_t}{\partial h_{t-1}} \right|\right|_2 &\leq \delta_u \left( \max_i \{[h_{t-1}]_i\} + \max_i \{[c_t]_i\} \right) ||U_u||_2 + \max_i \{(1 - [u_t]_i)\} \\ &\quad + \max_i \{[u_t]_i\} \left( \delta_r \, ||U_r||_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\} \right) ||U_c||_2 \qquad (42) \\ &=: \alpha + \beta \, ||U_c||_2 \qquad (43) \end{aligned}$$

651 where

$$\alpha = \delta_u \left( \max_i \{[h_{t-1}]_i\} + \max_i \{[c_t]_i\} \right) ||U_u||_2 + \max_i \{(1 - [u_t]_i)\} \tag{44}$$

652 and

$$\beta = \max_i \{[u_t]_i\} \left( \delta_r \, ||U_r||_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\} \right) . \tag{45}$$

653 □

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

$$\alpha \leq \frac{1}{2} \left|\left|U_u\right|\right|_2 + 1 \quad \textit{and} \quad \beta \leq \frac{1}{4} \left|\left|U_r\right|\right|_2 + 1. \tag{46}$$

*Proof.* The function $\sigma'(x) = \sigma(x)\left(1 - \sigma(x)\right)$ is bounded above by $\frac{1}{4}$, thus both $\delta_u, \delta_r \leq \frac{1}{4}$.

Now, to show that $[h_t]_i \leq 1$ for any $i$ and $t$, we first need to note that $h_0$ is initialized to zero (i.e., $[h_0]_i = 0$ for all $i$) and $0 \leq [u_t]_i, [c_t]_i \leq 1$ for all $i$ and $t$ from the definition of GRU cell in (1). Then for any fixed $i$

$$[h_0]_i = 0 \tag{47}$$

$$[h_1]_i = [1 - u_1]_i \cdot [h_0]_i + [u_1]_i \cdot [c_1]_i \tag{48}$$

$$= [u_1]_i \cdot [c_1]_i \leq 1 \tag{49}$$

Furthermore, if we assume that $[h_\tau]_i \leq 1$ for some $\tau \geq 1$, then

$$[h_{\tau+1}]_i = [1 - u_{\tau+1}]_i \cdot [h_\tau]_i + [u_{\tau+1}]_i \cdot [c_{\tau+1}]_i \tag{50}$$

$$\leq [1 - u_{\tau+1}]_i \cdot 1 + [u_{\tau+1}]_i \cdot 1 \tag{51}$$

$$= [1 - u_{\tau+1}]_i + [u_{\tau+1}]_i = 1. \tag{52}$$

Thus, by induction, we can conclude that $[h_t]_i \leq 1$ for any $i$ and $t$. Finally, using these obtained bounds, we can bound constants $\alpha$ and $\beta$ as follows

$$\alpha = \delta_u \left(\max_i \left\{[h_{t-1}]_i\right\} + \max_i \left\{[c_t]_i\right\}\right) \left|\left|U_u\right|\right|_2 + \max_i \left\{(1 - [u_t]_i)\right\} \tag{53}$$

$$\leq \frac{1}{4} \cdot (1 + 1) \cdot \left|\left|U_u\right|\right|_2 + 1 \tag{54}$$

$$= \frac{1}{2} \left|\left|U_u\right|\right|_2 + 1 \tag{55}$$

and

$$\beta = \max_i \{[u_t]_i\} \left(\delta_r \left|\left|U_r\right|\right|_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\}\right) \tag{56}$$

$$\leq 1 \cdot \left(\frac{1}{4} \cdot \left|\left|U_r\right|\right|_2 \cdot 1 + 1\right) \tag{57}$$

$$= \frac{1}{4} \left|\left|U_r\right|\right|_2 + 1. \tag{58}$$

Note, that all of these bounds are independent of $i$ and $t$.

□

The following two corollaries use a notation $x \approx y$ and $x \lesssim y$ to denote that $x$ is approximately equal to $y$ and $x$ is bounded by a quantity that is approximately equal to $y$, respectively.

**Corollary 2.** *When the 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:*

$$\alpha + \beta \lesssim 2. \tag{59}$$

*Moreover, if $u_t$ and $r_t$ are nearly either the zero vector or the vector of all ones, then*

$$\alpha + \beta \lesssim 1. \tag{60}$$

*Proof.* Recall the definition of $\alpha$ and $\beta$

$$\alpha = \delta_u \left(\max_i \{[h_{t-1}]_i\} + \max_i \{[c_t]_i\}\right) \left|\left|U_u\right|\right|_2 + \max_i \{(1 - [u_t]_i)\} \tag{61}$$

and

$$\beta = \max_i \{[u_t]_i\} \left(\delta_r \left|\left|U_r\right|\right|_2 \max_i \{[h_{t-1}]_i\} + \max_i \{[r_t]_i\}\right). \tag{62}$$

If we assume that $[u_t]_i \approx 0$ and $[u_t]_j \approx 1$ for some $i \neq j$, then $\delta_u \approx 0$ and $\alpha \lesssim 1$. Additionally, if we assume that the elements of $r_t$ are nearly either 0 or 1, then $\delta_r \approx 0$ and $\beta \lesssim 1$. Putting these two inequalities together yields

$$\alpha + \beta \lesssim 2. \tag{63}$$

Now, if we assume that $u_t$ is nearly a zero vector (i.e., $\max_i \{[u_t]_i\} \approx 0$), then constant $\delta_u \approx 0$, $\alpha \approx 1$, and $\beta \approx 0$. On the other hand, if we assume that $u_t$ is nearly the vector of all ones (i.e., $\min_i \{[u_t]_i\} \approx 1$) then $\delta_u \approx 0$ and $\alpha \approx 0$. Moreover, if we also assume that $r_t$ is nearly a zero vector (i.e., $\max_i \{[r_t]_i\} \approx 0$) then $\delta_r \approx 0$ and $\beta \approx 0$. However, if we assume that $r_t$ approaches a vector of all ones (i.e., $\min_i \{[r_t]_i\} \approx 1$), then $\delta_r \approx 0$ but $\beta \approx 1$ for this case. Putting all of these cases together, we obtain

$$\alpha + \beta \lesssim 1. \tag{64}$$

□

**Corollary 3.** *Let $h_{t-1}$ and $h_t$ be two consecutive hidden states from the NC-GRU model defined in (19). Then $||U_r||_2 = ||U_c||_2 = 1$ and if elements of the gates $u_t$ and $r_t$ are nearly either 0 or 1, then the following inequality is satisfied:*

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right|_2 \lesssim 2. \tag{65}$$

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

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right|_2 \lesssim 1. \tag{66}$$

*Proof.* By the definition of NC-GRU model, weights $U_r$ and $U_c$ are orthogonal which implies $||U_r||_2 = ||U_c||_2 = 1$, and by Theorem 1 and Corollary 2, we conclude

$$\left|\left|\frac{\partial h_t}{\partial h_{t-1}}\right|\right|_2 \lesssim 2 \tag{67}$$

when element of the gates $u_t$ and $r_t$ approach either 0 or 1; and

$$\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\|_2 \lesssim 1 \tag{68}$$

if $u_t$ and $r_t$ approach either zero vector or vector of all ones.

□

# SM 8 Selected Accuracies

Table 8: **Denoise Task** $T = 400$ **Accuracy**

| | Accuracy ↑ |
|---|---|
| LSTM | 96.01% |
| GRU | 96.32% |
| scoRNN | 97.56% |
| GORU | 99.08% |
| NC-GRU($U_c$) (**ours**) | 99.46% |
| NC-GRU($U_c, U_r$) (**ours**) | 99.72% |

Table 9: **Copying Task** $T = 2,000$ **Accuracy**

| | Accuracy ↑ |
|---|---|
| LSTM | 99.56% |
| GRU | 99.64% |
| scoRNN | 99.99% |
| GORU | 99.57% |
| NC-GRU($U_c$) (**ours**) | 99.76% |
| NC-GRU($U_c, U_r$) (**ours**) | 99.82% |