Complex Unitary Recurrent Neural Networks Using Scaled Cayley Transform

Kehelwala D. G. Maduranga, Kyle E. Helfrich, Qiang Ye

Mathematics Department, University of Kentucky

Lexington, KY, 40508, United States

{kdgmaduranga,kyle.helfrich,qye3}@uky.edu

Abstract

Recurrent neural networks (RNNs) have been successfully used on a wide range of sequential data problems. A well known difficulty in using RNNs is the *vanishing or exploding gradient* problem. Recently, there have been several different RNN architectures that try to mitigate this issue by maintaining an orthogonal or unitary recurrent weight matrix. One such architecture is the scaled Cayley orthogonal recurrent neural network (scoRNN) which parameterizes the orthogonal recurrent weight matrix through a scaled Cayley transform. This parametrization contains a diagonal scaling matrix consisting of positive or negative one entries that can not be optimized by gradient descent. Thus the scaling matrix is fixed before training and a hyperparameter is introduced to tune the matrix for each particular task. In this paper, we develop a unitary RNN architecture based on a complex scaled Cayley transform. Unlike the real orthogonal case, the transformation uses a diagonal scaling matrix consisting of entries on the complex unit circle which can be optimized using gradient descent and no longer requires the tuning of a hyperparameter. We also provide an analysis of a potential issue of the modReLU activiation function which is used in our work and several other unitary RNNs. In the experiments conducted, the scaled Cayley unitary recurrent neural network (scuRNN) achieves comparable or better results than scoRNN and other unitary RNNs without fixing the scaling matrix.

1 Introduction

Recurrent neural networks (RNNs) have been successfully used on a wide range of sequential data problems. A main difficulty when training RNNs using a gradient descent based optimizer is the *vanishing or exploding gradient* problem (Bengio, Simard, and Frasconi 1994). The exploding gradient problem refers to the large growth of gradients as they propagate backwards through time and the vanishing gradient problem occurs when the gradients tend toward zero. The exploding gradient case will cause the trainable parameters to vary drastically during training, resulting in unstable performance. For the vanishing gradient case, training will progress slowly, if at all.

A range of different architectures have been proposed to address this problem. Currently, the most common architectures involve gating mechanisms that control when information is retained or discarded such as the Long Short-Term Memory networks (LSTMs) (Hochreiter and Schmidhuber 1997) and Gated Recurrent Units (GRUs) (Cho et al. 2014). More recently, several architectures have been proposed that maintain a unitary recurrent weight matrix. The unitary evolution RNN (uRNN) architecture proposed by (Arjovsky, Shah, and Bengio 2016) maintains a unitary matrix by using a product of simple unitary matrices. The full-capacity uRNN (Wisdom et al. 2016) maintains a general unitary matrix by optimizing along a gradient descent direction on the Stiefel manifold. The tunable efficient unitary neural network (EURNN) by (Jing et al. 2016) constructs the unitary matrix using a product of complex rotation matrices. For additional work with unitary RNNs and complex valued networks, see (Hyland and Gunnar 2017), (Jing et al. 2017), (Vorontsov et al. 2017), (Wolter and Yao 2018) and (Trabelsi et al. 2018). There have also been several architectures that maintain an orthogonal recurrent weight matrix. The orthogonal RNN (oRNN) by (Mhammedi et al. 2017) uses a product of Householder reflection matrices while the scaled Cayley orthogonal RNN (scoRNN) architecture parameterizes the recurrent weight matrix by a skew-symmetric matrix and a diagonal matrix through the Cayley transform (Helfrich, Willmott, and Ye 2018). Compared with other unitary/orthogonal RNNs, the scoRNN architecture has a simple parameterization that has been shown to be advantageous in (Helfrich, Willmott, and Ye 2018). The exploding or vanishing gradient problem has also been examined in (Henaff, Szlam, and LeCun 2017) and (Le, Jaitly, and Hinton 2015).

In this paper, we address a difficulty of scoRNN. The scoRNN parameterization of the orthogonal recurrent weight matrix contains a diagonal matrix consisting of ± 1 on the diagonal. These discrete parameters are used to define the scaling matrix, which may critically affect performance, and can not be optimized by gradient descent. Thus scoRNN introduces a tunable hyperparameter that controls the number of negative ones on the diagonal. This hyperparameter is tuned for each particular task for optimal results. This causes the scaling matrix to remain fixed during training. We propose a method to overcome this difficulty by using a unitary recurrent weight matrix parameterized by a skew-Hermitian matrix and a diagonal scaling matrix through the scaled Cayley transform, where the entries of the diagonal matrix lie on the complex unit circle and have the form $e^{i\theta}$. This param-

eterization is differentiable with respect to the continuous θ variable and can be optimized using gradient descent. This eliminates the need for tuning a hyperparameter and having a fixed scaling matrix during training. We call this new architecture the scaled Cayley unitary recurrent neural network (scuRNN). We also develop the update scheme to train the skew-Hermitian and diagonal scaling matrices. The experiments performed show that scuRNN achieves better or comparable results than other unitary RNNs and scoRNN without the need for tuning an additional hyperparameter.

For many unitary RNNs, a popular activation function is the modReLU function (Arjovsky, Shah, and Bengio 2016). Known architectures that incorporate the modReLU function include works by (Arjovsky, Shah, and Bengio 2016), (Wisdom et al. 2016), (Jing et al. 2016), (Jing et al. 2017), (Helfrich, Willmott, and Ye 2018), and (Wolter and Yao 2018). We also use the modReLU activation function in this work but have noticed a singularity issue that may potentially impact performance. To the best of our knowledge, this singularity has not been previously discussed in the literature. In section 4, we provide an analysis of the modReLU function and discuss initialization schemes that may mitigate the singularity.

We note that there has been recent interest in complex networks outside of uRNN as discussed in the papers by (Trabelsi et al. 2018) and (Wolter and Yao 2018). Our work presents an additional case where complex networks can be advantageous over real networks.

2 Background

2.1 Real RNNs

A single hidden layer recurrent neural network (RNN) is a dynamical system that uses an input sequence $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_\tau)$ where each $\mathbf{x}_i \in \mathbb{R}^m$, to produce an output sequence $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_\tau)$ with $\mathbf{y}_i \in \mathbb{R}^p$ given recursively by the following:

$$\mathbf{h}_t = \sigma \left(U\mathbf{x}_t + W\mathbf{h}_{t-1} + \mathbf{b} \right) ; \ \mathbf{y}_t = V\mathbf{h}_t + \mathbf{c} \quad (1)$$

where $U \in \mathbb{R}^{n \times m}$ is the input to hidden weight matrix, $W \in \mathbb{R}^{n \times n}$ the recurrent weight matrix, $\mathbf{b} \in \mathbb{R}^n$ the hidden bias, $V \in \mathbb{R}^{p \times n}$ the hidden to output weight matrix, and $\mathbf{c} \in \mathbb{R}^p$ the output bias. Here m is the input data size, n is the number of hidden units, and p is the output data size. The sequence $\mathbf{h} = (\mathbf{h}_0, \ldots, \mathbf{h}_{\tau-1})$, is the sequence of hidden layer states with $\mathbf{h}_i \in \mathbb{R}^n$ and $\sigma(\cdot)$ is a pointwise nonlinear activation function, such as a hyperbolic tangent function or rectified linear unit (Nair and Hinton 2010).

For long sequence lengths, RNNs are prone to suffer from the exploding or vanishing gradient problem. As detailed in (Arjovsky, Shah, and Bengio 2016), the exploding gradient problem can occur when the spectral radius of the recurrent weight matrix is greater than one and the vanishing gradient problem can occur when it is less than one. Maintaining a strict unitary or orthogonal recurrent weight matrix with a spectral radius of one can help mitigate this problem.

2.2 Unitary RNNs

Similar to orthogonal matrices, unitary matrices are complex matrices $W \in \mathbb{C}^{n \times n}$ with spectral radius one and the property that $W^*W = I$, where $*$ denotes the conjugate transpose operator and I is the identity matrix. All unitary RNNs are designed to maintain a unitary recurrent weight matrix. In this section, we examine the unitary evolution RNN (uRNN), full-capacity uRNN, and the EURNN. For notational purposes, we follow the convention established in (Wisdom et al. 2016) and refer to the unitary evolution RNN as the restricted-capacity uRNN.

The restricted-capacity uRNN maintains a unitary recurrent weight matrix by using a parameterization consisting of diagonal matrices with entries lying on the complex unit disk, Householder reflection matrices, Fourier and inverse Fourier transform matrices, and a fixed permutation matrix (Arjovsky, Shah, and Bengio 2016). As shown by (Wisdom et al. 2016), this parameterization contains only $7n$ trainable parameters and is unable to represent all unitary matrices when the hidden size $n > 7$.

The full-capacity uRNN does not parameterize the recurrent weight matrix directly, but restricts the descent direction to the Stiefel manifold $\{W \in \mathbb{C}^{n \times n} | W^*W = I\}$. This is done by traveling along a curve of the tangent plane projected onto the Stiefel manifold using a multiplicative update scheme as outlined in (Wisdom et al. 2016). As shown in (Helfrich, Willmott, and Ye 2018), this multiplicative update scheme may result in a loss of orthogonality due to numerical rounding issues.

Similar to the restricted-capacity uRNN, the EURNN parameterizes the unitary recurrent weight matrix by a product of unitary matrices. Specifically, the product consists of a unitary diagonal matrix and complex Givens rotation matrices. Unlike the restricted-capacity uRNN, the EURNN parameterization has the capacity to represent all possible unitary matrices but requires a long product of matrices.

3 Scaled Cayley Unitary RNN (scuRNN)

3.1 Scaled Cayley Transform

Unlike other architectures that use a long product of simple matrices to parameterize the unitary recurrent weight matrix, the scuRNN architecture maintains a strictly unitary recurrent weight matrix by incorporating the following result.

Theorem 1 (Kahan; O'Dorney). *Every unitary matrix W can be expressed as*

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

where $A = [a_{ij}]$ is skew-Hermitian with $|a_{ij}| \leq 1$ and D is a unitary diagonal matrix. For an orthogonal W, the same result holds with A being skew-symmetric and D a diagonal matrix with entries consisting of ± 1.

In scoRNN, the orthogonal matrix is constructed using the orthogonal parameterization in Theorem 1. The scaling matrix D is not known a priori and needs to be determined for each particular task. Since D consists of discrete valued parameters, it can not be determined by gradient descent. However, D is essentially defined by the number of negative ones on the diagonal barring a permutation. Thus the number of negative ones is considered a hyperparameter that must be tuned for optimal results with the additional restriction that the scaling matrix must be fixed during training.

The scuRNN architecture uses the complex version in Theorem 1. It overcomes the constraints inherent with scoRNN since D in this case has entries of the form $D_{j,j} = e^{i\theta_j}$. This parameterization is differentiable with respect to the continuous θ_j variables and can be determined by gradient descent during training with D no longer being fixed.

3.2 Architecture Details

The scuRNN architecture is similar to a standard RNN, see (1), with the exception that the hidden bias is incorporated in the modReLU activation function $\sigma_{\text{modReLU}}(z)$, see section 4 for definition, and all matrices are complex valued.

$$\mathbf{h}_t = \sigma_{\text{modReLU}}\left(U\mathbf{x}_t + W\mathbf{h}_{t-1}\right) \; ; \; \mathbf{y}_t = V\mathbf{h}_t + \mathbf{c} \qquad (2)$$

Since the input to hidden, recurrent, and hidden to output weight matrices are complex valued, we follow the framework described in (Arjovsky, Shah, and Bengio 2016) to compute complex matrix vector products by separating all complex numbers in terms of their real and imaginary parts.

3.3 Training the Skew-Hermitian and Scaling Matrices

In order to train the skew-Hermitian matrix A and scaling matrix D that are used to parameterize the unitary recurrent weight matrix in scuRNN, we have to deal with complex derivatives. When we consider the loss function as a function of the complex matrix A or scaling matrix D with a range on the real-line, the loss function is nonholomorphic and thus not complex differentiable. To compute the necessary gradients, Wirtinger calculus is required (Kreutz-Delgado 2009).

In Wirtinger calculus, differentiable complex functions are viewed as differentiable functions over $\mathbb{R}^2$. In particular, given a nonholomorphic function, $f : \mathbb{C} \to \mathbb{R}$, the differential df is given by

$$df = \frac{\partial f}{\partial z}dz + \frac{\partial f}{\partial \overline{z}}d\overline{z},$$

where $z := x+iy \in \mathbb{C}$ and $\overline{z} := x-iy \in \mathbb{C}$ is the conjugate. Here the Wirtinger derivatives are given by

$$\frac{\partial f}{\partial z} = \frac{1}{2}\left(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}\right) \quad \text{and} \quad \frac{\partial f}{\partial \overline{z}} = \frac{1}{2}\left(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y}\right).$$

Results from (Hunger 2007) show that the steepest descent direction using Wirtinger calculus is $\frac{\partial f(z)}{\partial \overline{z}}$.

Using the above Wirtinger derivatives and steepest descent direction, we update the unitary recurrent weight matrix W by performing gradient descent on the associated skew-Hermitian parameterization matrix A and scaling matrix D. In order to compute gradients with respect to A, we must pass the gradients through the scaled Cayley transform. The desired gradients for A and diagonal arguments of D are given in Theorem 2. A proof is given in (Maduranga, Helfrich, and Ye 2018).

Theorem 2. *Let $L = L(W) : \mathbb{C}^{n \times n} \to \mathbb{R}$ be a differentiable cost function for an RNN with recurrent weight matrix W. Let $W = W(A, \boldsymbol{\theta}) := (I + A)^{-1}(I - A)D$ where $A \in \mathbb{C}^{n \times n}$ is skew-Hermitian, $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_n]^T \in \mathbb{R}^n$,*

and $D = diag\left(e^{i\theta_1}, e^{i\theta_2}, ..., e^{i\theta_n}\right) \in \mathbb{C}^{n \times n}$ is a unitary diagonal matrix. Then the gradient of $L = L(W(A, \boldsymbol{\theta}))$ with respect to $\overline{A}$ is

$$\frac{\partial L}{\partial \overline{A}} = C^T - \overline{C}$$

where $C := (I + A)^{-T}\frac{\partial L}{\partial W}(D + W^T)$, $\frac{\partial L}{\partial \overline{A}} = \left[\frac{\partial L}{\partial \overline{A}_{i,j}}\right] \in \mathbb{C}^{n \times n}$, and $\frac{\partial L}{\partial W} = \left[\frac{\partial L}{\partial W_{i,j}}\right] \in \mathbb{C}^{n \times n}$. Furthermore, the gradient of $L = L(W(A, \boldsymbol{\theta}))$ with respect to $\boldsymbol{\theta}$ is given by

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = 2Re\left(i\left(\left(\frac{\partial L}{\partial W}^T K\right) \odot I\right)d\right)$$

where $K = (I + A)^{-1}(I - A)$, $d = \left[e^{i\theta_1}, e^{i\theta_2}, ..., e^{i\theta_n}\right]^T$ is the diagonal vector of D and $\odot$ denotes entry-wise multiplication.

We use the above theorem to update the recurrent weight matrix. First we compute $\frac{\partial L}{\partial W}$, using the standard backpropagation algorithm. Then using $\frac{\partial L}{\partial W}$, we compute $\frac{\partial L}{\partial A}$, and $\frac{\partial L}{\partial \boldsymbol{\theta}}$, using Theorem 2. We then update the diagonal matrix D by first updating the argument vector $\boldsymbol{\theta} = [\theta_1, ..., \theta_n]^T$ using a standard optimizer, such as gradient descent, and re-forming D.

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - \alpha\frac{\partial L(A^{(k)}, \boldsymbol{\theta}^{(k)})}{\partial \boldsymbol{\theta}} \; ; \; D^{(k+1)} = \text{diag}\left(e^{i\boldsymbol{\theta}^{(k+1)}}\right)$$

where α is the learning rate and diag$(\cdot)$ forms a diagonal matrix. We then update the matrix A

$$A^{(k+1)} = A^{(k)} - \beta\frac{\partial L(A^{(k)}, \boldsymbol{\theta}^{(k)})}{\partial \overline{A}},$$

where β is the learning rate. We should note that for optimizers that involve squaring the entries of the gradient element-wise, such as RMSProp, Adam, and Adagrad, the update of A is split into updating the real component and imaginary component separately to maintain a skew-Hermitian matrix A. Since $\frac{\partial L}{\partial \overline{A}}$ is skew-Hermitian and skew-Hermitian matrices are closed under addition, $A^{(k+1)}$ will be skew-Hermitian. Finally, we construct the recurrent weight matrix using the scaled Cayley transform

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

4 ModReLU activation Function

The right selection of a nonlinear activation function plays a major role in avoiding the vanishing and exploding gradient problem. We use the modReLU activation function for a complex variable which is a modification of the ReLU activation function. The modReLU activation function was first proposed by (Arjovsky, Shah, and Bengio 2016) and also used in architectures by (Wisdom et al. 2016; Jing et al. 2016; 2017; Wolter and Yao 2018) to handle complex valued functions and weights and studied in (Trabelsi et al. 2018). The modReLU activation function is defined as

$$\sigma_{\text{modReLU}}(z) = \begin{cases} (|z| + b)\frac{z}{|z|} & \text{if } |z| + b \geq 0 \\ 0 & \text{if } |z| + b < 0 \end{cases} \qquad (3)$$

$$= \frac{z}{|z|}\sigma_{\text{ReLU}}(|z| + b), \qquad (4)$$

where b denotes a trainable bias. If $b > 0$, the modReLU activation function as defined above has a discontinuity at $z = 0$ no matter how $\sigma_{\text{modReLU}}(0)$ is defined. As a result, the derivative of modReLU has a singularity at $z = 0$ when $b > 0$ as follows: (see (Maduranga, Helfrich, and Ye 2018) for a proof)

Theorem 3. *For the modified rectified linear activation function $\sigma_{modReLU}(z)$, the Wirtinger derivatives are given by*

$$\frac{\partial \sigma_{modReLU}(z)}{\partial z} = \begin{cases} 1 + \frac{b}{2|z|} & \text{if } |z| + b \geq 0 \\ 0 & \text{if } |z| + b < 0, \end{cases}$$

$$\frac{\partial \sigma_{modReLU}(z)}{\partial \overline{z}} = \begin{cases} \frac{1}{2}\left[\frac{-bz^2}{|z|^3}\right] & \text{if } |z| + b \geq 0 \\ 0 & \text{if } |z| + b < 0. \end{cases}$$

In particular, if $b > 0$, then $\frac{\partial \sigma_{modReLU}(z)}{\partial z}$ and $\frac{\partial \sigma_{modReLU}(z)}{\partial \overline{z}}$ tend to infinity as $z \mapsto 0$.

Remark. *If b is positive and $|z| \ll b$, then $\left|\frac{\partial \sigma_{modReLU}(z)}{\partial \overline{z}}\right|$ is extremely large and will result in floating point exceptions such as NAN during training. On the other hand, if $b \leq 0$ then the derivatives are well defined and bounded for all z.*

In the implementations of the uRNNs by (Arjovsky, Shah, and Bengio 2016) and (Wisdom et al. 2016), the following approximate modReLU activation function is used:

$$\sigma_\epsilon(z) = \frac{z}{\hat{z} + \epsilon} \sigma_{\text{ReLU}}(\hat{z} + b) \qquad (5)$$

where $\epsilon = 10^{-5}$ and $\hat{z} := \sqrt{x^2 + y^2 + \epsilon}$. The idea behind this is to avoid division by zero during the forward pass and backward pass of the network. This version of the modReLU function is also implemented in the scuRNN model. Unfortunately, a large derivative can still occur if $\hat{z} + \epsilon \ll b$ as shown in the following theorem and Figure 1.

Theorem 4. *The Wirtinger derivatives of the approximate modReLU activation function can be given as:*

$$\frac{\partial \sigma_\epsilon(z)}{\partial z} = \begin{cases} \frac{\hat{z}+b}{\hat{z}+\epsilon} + \frac{|z|^2(\epsilon-b)}{2\hat{z}(\hat{z}+\epsilon)^2} & \text{if } \hat{z} + b \geq 0 \\ 0 & \text{if } \hat{z} + b < 0, \end{cases}$$

$$\frac{\partial \sigma_\epsilon(z)}{\partial \overline{z}} = \begin{cases} \frac{z^2(\epsilon-b)}{2\hat{z}(\hat{z}+\epsilon)^2} & \text{if } \hat{z} + b \geq 0 \\ 0 & \text{if } \hat{z} + b < 0. \end{cases}$$

In particular, we found that the unitary RNNs are sensitive to the initialization of the bias, **b**, and the initial state, $\mathbf{h}_0$. For example, in the MNIST experiment, see section 5.1, when the initial state $\mathbf{h}_0$ is set to $\mathbf{h}_0 = 0$ and non-trainable and **b** is initialized by sampling from the uniform distribution $\mathcal{U}[-0.01, 0.01]$, the gradient values of the loss function would become NAN before the end of the first training epoch. With a random initialization of **b**, many entries are positive and are much larger in magnitude then the corresponding entries of $\hat{\mathbf{z}}$ which results in large derivatives. These large derivatives over many time steps can lead to the exploding gradient problem. The cause for small $\hat{\mathbf{z}}$ in this experiment is due to the fact that the first several pixels of any given MNIST image will most likely have zero pixel value which combined with the zero initial state $\mathbf{h}_0$ will result in
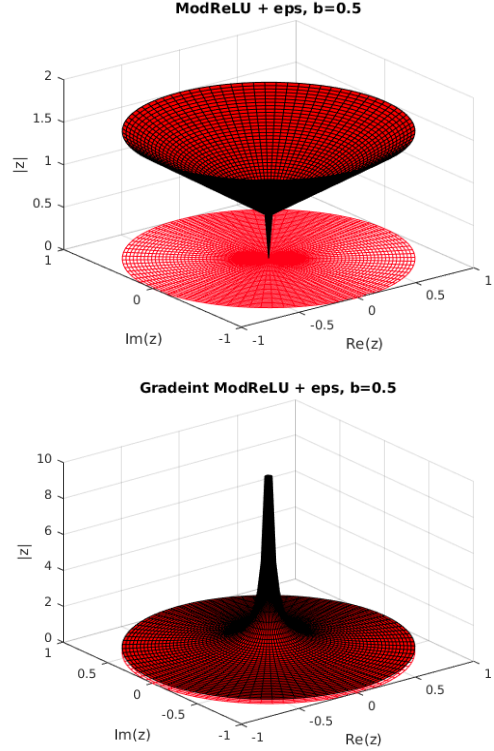

Figure 1: Surface plots of the modulus of the approximate modReLU activation function σ_ϵ (top) and the modulus of the gradient of σ_ϵ with respect to $\overline{z}$ (bottom). Both plots use a bias of b=0.5.

small values of $\hat{\mathbf{z}}$ compared to the corresponding entries in **b**.

To avoid small values of $\hat{\mathbf{z}}$, it is useful to initialize $\mathbf{h}_0$ away from zero. This will mitigate the effects when the first several sequence entries are zero. Based on experimentation, it is also advantageous to allow $\mathbf{h}_0$ to be trainable. If singularity is still encountered with initializing and/or training $\mathbf{h}_0$, constraining b to be nonpositive may be used, since the singularity only occurs when $b > 0$. For example, initializing $\mathbf{b} = \mathbf{0}$ will avoid the singularity at least initially so that training can proceed, regardless of the magnitude of $\hat{\mathbf{z}}$. However, subsequent training would typically turn some entries of **b** into positive numbers. On the other hand, we have experimented with schemes that maintain nonpositive **b**, which indeed eliminate the singularity but tend to hinder performace.

5 Experiments

In this section, we compare the performances between the restricted-capacity uRNN, full-capacity uRNN, EURNN, LSTM, scoRNN, oRNN and scuRNN architectures on a variety of tasks. Code for these experiments is available at https://github.com/Gayan225/scuRNN. For each model, the hidden size was adjusted to match the number of trainable parameters. For scuRNN, the real component of the skew-Hermitian matrix was initialized as a skew-symmetric ma-

trix using the initialization used in (Helfrich, Willmott, and Ye 2018) while the imaginary component was initialized to zero. The initial hidden state was initialized using the distribution $\mathcal{U}[-0.01, 0.01]$ and was trainable. The input to hidden matrix U and hidden to output matrix V were initialized using Glorot (Glorot and Bengio 2010). The θ values are sampled from $\mathcal{U}[0, 2\pi]$, which results in the diagonal entries $D_{j,j} = e^{i\theta_j}$ being uniformly distributed on the complex unit circle. The biases are initialized from the distribution $\mathcal{U}[-0.01, 0.01]$.

The parameterization used in scuRNN allows the use of different optimizers and different learning rates for the input and output weights, skew-Hermitian matrix, and the scaling matrix. We used several different combinations of optimizers and learning rates as noted under each experiment. The reasoning behind mixing different optimizers and learning rates is that the A and D matrices are implicit parameters that are not weights themselves and their entries may have different scales from those of the weights. An update on A and D using the same optimizer/learning rates as the non-recurrent weights may result in an update of W that is incompatible with the updates in the non-recurrent weights. However, scuRNN may be implemented with the same optimizer and learning rate for A and D matrices, which would involve no additional hyperparameters for tuning compared to scoRNN. In most cases, they produce competitive results (see Appendix in (Maduranga, Helfrich, and Ye 2018)).

Experiment settings for scoRNN and LSTM are in accordance with (Helfrich, Willmott, and Ye 2018), (Wisdom et al. 2016), and their corresponding codes. When not listed in their papers, we used the following settings with results that are consistent with the other papers. For LSTM we used an RMSProp optimizer with learning rate 10^{-3} on MNIST, permuted MNIST, and copying problems with forget gate bias initialize to 1.0. For the adding problem, we used an Adam optimizer with learning rate 10^{-2}. For TIMIT, an RMSProp optimizer with learning rate 10^{-3} with forget gate bias -4 were used. For scoRNN, the copying and adding problems used an RMSProp optimizer with learning rate 10^{-4} for A and an Adam optimizer with learning rate 10^{-3} for all other weights for the adding problem and an RMSProp optimizer with learning rate 10^{-3} for all other weights for the copying problem.

5.1 MNIST Classification

This experiment involves the classification of handwritten digits using the MNIST database (LeCun and Cortes 2010). The data set consists of 55,000 training images and 10,000 testing images with each image in the dataset consisting of a 28×28 pixel gray-scale image of a handwritten digit ranging from 0 to 9. Using the procedure outlined in (Le, Jaitly, and Hinton 2015), each image is flattened into a vector of length 784 with a single pixel sequentially fed into the RNN. The last sequence output is used to classify the digit. We refer to this experiment as the unpermuted MNIST experiment. A variation of this experiment is to apply a fixed permutation to both the training and test sequences and we refer to this version as the permuted MNIST experiment. All the models were trained for a total of 70 epochs in accordance

with (Helfrich, Willmott, and Ye 2018). Results of the experiments are given in Table 1, Figure 2, and Figure 3.

Table 1: Results for the MNIST classification problem. The best epoch test accuracy over the entire 70 epoch run are recorded. Entries marked by an asterix are reported results from (Mhammedi et al. 2017) and (Jing et al. 2016).

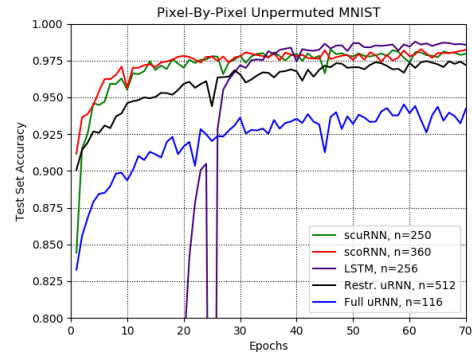| Model	n	# params	Unperm. Test Acc.	Perm. Test Acc.
scuRNN	116	$\approx$ 16k	0.976	0.949
scuRNN	250	$\approx$ 69k	0.983	0.962
scoRNN	170	$\approx$ 16k	0.973	0.943
scoRNN	360	$\approx$ 69k	0.983	0.962
LSTM	128	$\approx$ 68k	0.987	0.920
LSTM	256	$\approx$ 270k	0.989	0.929
Rest. cap. uRNN	512	$\approx$ 16k	0.976	0.945
Full. cap. uRNN	116	$\approx$ 16k	0.947	0.925
Full. cap. uRNN	512	$\approx$ 270k	0.974	0.947
oRNN	256	$\approx$ 11k	0.972*	-
EURNN	512	$\approx$ 9k	-	0.937*

Figure 2: Pixel-by-Pixel MNIST Results

For the unpermuted MNIST experiment, the scuRNN models used an RMSProp optimizer to update the skew-Hermitian matrix and an Adagrad optimizer to update the scaling matrix with all other parameters updated using the Adam optimizer. For hidden size $n = 116$, the learning rates were 10^{-4}, 10^{-3}, and 10^{-3} respectively. For hidden size $n = 250$, the learning rates were 10^{-5}, 10^{-4}, and 10^{-3} respectively. Although scuRNN was unable to outperform the LSTM architecture, the $n = 116$ scuRNN was able to match the performance of the $n = 512$ restricted-capacity uRNN and to outperform all other models. It should be noted that the scuRNN had a much smaller hidden size with less than 4 times the hidden size of the restricted-capacity uRNN. The $n = 250$ scuRNN was able to match the accuracy of the $n = 360$ scoRNN with a smaller hidden size.

For the permuted MNIST experiment, the optimizers for the scuRNN models were the same as the ones used in the unpermuted MNIST experiment. For hidden size $n = 116$, the learning rates for the scuRNN model were 10^{-4}, 10^{-3}, and 10^{-3} respectively. For hidden size $n = 250$, the learning
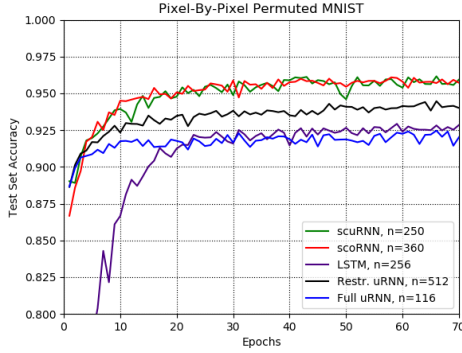
Figure 3: Permuted Pixel-by-Pixel MNIST Results

rates were the same except the skew-Hermitian matrix had a learning rate of 10^{-5}. In this task the $n = 250$ scuRNN matches the highest test accuracy of the $n = 360$ scoRNN which is higher than all other unitary RNN and LSTM models. It should be noted that the smaller $n = 116$ scuRNN outperforms the $n = 170$ scoRNN with the same order of trainable parameters.

5.2 Copying Problem

The experiment follows the setup described in (Arjovsky, Shah, and Bengio 2016), (Wisdom et al. 2016), and (Helfrich, Willmott, and Ye 2018). A sequence is passed into the RNN using the digits 0-9. The first ten entries are uniformly sampled from the digits 1-8. This is followed by a sequence of T zeros and a marker digit 9. At the marker digit, the RNN is to output the first ten entries of the sequence. This results in an entire sequence length of T+20. The baseline for this task is a network that outputs all zeros except for the last 10 digits which are uniformly sampled from the digits 1-8 for an expected categorical cross entropy of $\frac{10 \log(8)}{T+20}$.

For our experiment we adjust the number of hidden units of each network so that they all have approximately 22k trainable parameters. This results in an LSTM with hidden size $n = 68$, a restricted-capacity uRNN with $n = 470$, a full-capacity uRNN with $n = 128$, a EURNN with $n = 512$ and capacity $L = 2$, a scoRNN with $n = 190$, and a scuRNN with $n = 130$. We also tested the architectures using a sequence length of $T = 1000$ and $T = 2000$ zeros.

The T=1000 scuRNN used Adagrad optimizer with learning rate 10^{-4} for the skew-Hermitian matrix and for the diagonal scaling matrix, and Adam optimizer with learning rate 10^{-3} for all other weights. For the T=2000 task, scuRNN used Adam optimizer with learning rate 10^{-4} for the diagonal scaling matrix and RMSProp optimizer with learning rate 10^{-4} and 10^{-3} for the skew-Hermitian matrix and all other trainable weights respectively.

The results for this experiment are included in Figure 4. For each experiment, the restricted-capacity uRNN and LSTM converge rapidly to the base line but fail to drop below it. For T =1000, the scuRNN, full-capacity uRNN, and scoRNN quickly drop towards zero cross entropy while the EURNN drops below the baseline but does not converge to-

wards zero. For T=2000, scuRNN drops towards zero cross entropy before all other networks. In this case, the EURNN is unable to drop below the baseline.
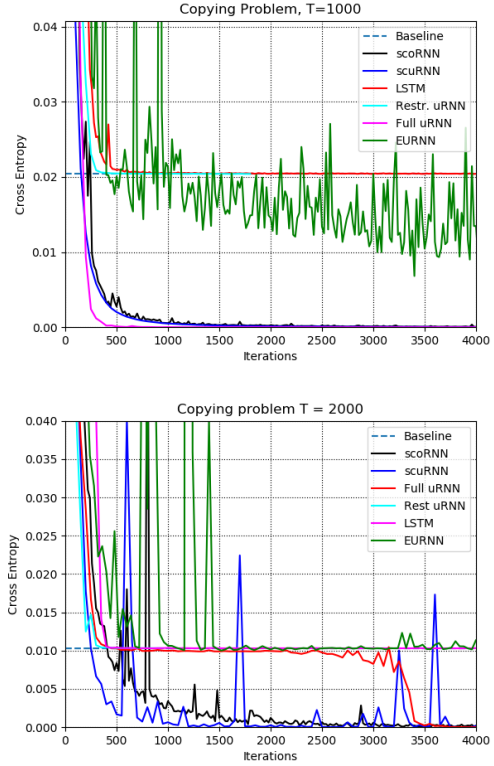

Figure 4: Cross entropy of the copying problem with T = 1000 and T=2000.

5.3 Adding Problem

The adding problem was first proposed by (Hochreiter and Schmidhuber 1997). In this experiment, we implement a slightly modified version as outlined in (Helfrich, Willmott, and Ye 2018). In this problem, two sequences of length T are simultaneously fed into the RNN. The first sequence consists of entries sampled from $\mathcal{U}[0, 1)$. The second sequence consists of all zeros except for two entries that are equal to one. The first one is uniformly located in the first half of the sequence, within the interval $\left[1, \frac{T}{2}\right)$, while the second one is located uniformly in the other half of the sequence, within the interval $\left[\frac{T}{2}, T\right)$. The goal of the network is to compute the sum of the two numbers from the first sequence that are marked by one in the second sequence. The loss function used for this task is the Mean Square Error (MSE). The baseline for this task is an expected MSE of 0.167 which is the expected MSE for a model that always outputs one.

The sequence lengths used were $T = 200$ and $T = 750$ with a training set size of 100,000 and a testing set size of 10,000 as in (Helfrich, Willmott, and Ye 2018). For T=200, scuRNN used the RMSProp optimizer with learning rate 10^{-3} for the skew-Hermitian matrix and the Adam optimizer with learning rate 10^{-3} for the diagonal scaling matrix

and all other weights. For $T = 750$, scuRNN used the Adam optimizer with learning rate 10^{-3} for the diagonal scaling matrix and the RMSProp optimizer for the skew-Hermitian matrix and all other weights with learning rates 10^{-4} and 10^{-3} respectively. For each model, the number of trainable parameters were matched to be approximately 14k. This results in a hidden size of $n = 116$ for scuRNN, $n = 170$ for scoRNN, $n = 60$ for LSTM, $n = 120$ for the full-capacity uRNN, and $n = 950$ for the restricted-capacity uRNN. For the EURNN, the tunable style model with a hidden size of $n = 512$ was used for $T = 200$ which results in $\approx$ 3k trainable parameters and the FFT style was used for $T = 750$ which results in $\approx$ 7k trainable parameters as outlined in (Helfrich, Willmott, and Ye 2018). For oRNN, a hidden size of $n = 128$ with 16 reflections which results in $\approx$ 2.6k trainable parameters was used in accordance with (Helfrich, Willmott, and Ye 2018) and (Mhammedi et al. 2017). Results are shown in Figure 5.

For sequence length $T = 200$, all the architectures start at or near the base line and eventually drop towards zero MSE with the exception of the EURNN model which appears to decrease below the baseline and eventually increases back towards it. The oRNN abruptly drops below the baseline first, followed by scuRNN. Although oRNN is the first to drop below the baseline, the descent curve is erratic with the oRNN bouncing back towards the baseline several training steps later. The LSTM also has a drastic drop towards the zero MSE solution but this occurs after the scuRNN curve passes below the baseline. Although the scuRNN and scoRNN architectures have similar performance, the scuRNN model descends below the baseline before the scoRNN model. For sequence length $T = 750$, the oRNN again drops below the base line first but has an erratic descent curve with the oRNN staying at the baseline near the end of training. The LSTM drops below the baseline next followed by the scoRNN and scuRNN models. It should be noted that although the scoRNN model curve descends below the scuRNN curve around the third epoch, the scuRNN model was able to descend towards zero MSE before the full-capacity uRNN and the restricted-capacity uRNN. The EURNN model appears to not be able to decrease below the baseline and the restricted-capacity uRNN model appears to not descend until around the tenth epoch.

5.4 TIMIT Speech Dataset

Experiments were conducted using the TIMIT data set (Garofolo et al. 1993). We use the same setup as described by (Wisdom et al. 2016) and (Helfrich, Willmott, and Ye 2018). TIMIT input values are the log magnitude of the modulus of STFT data and are real-valued. The core test set was used, consisting of a training set of 3,696 audio files, a testing set of 192 audio files, and a validation set of 400 audio files. Audio files were down sampled as detailed in (Wisdom et al. 2016). The hidden size of each model tested were adjusted to match the number of trainable parameters of approximately 83k and 200k. The best performance for scuRNN was achieved using the Adam optimizer for all trainable parameters. For both $n = 128$ and $n = 258$ the learning rates were 10^{-4} for the skew-Hermitian matrix and
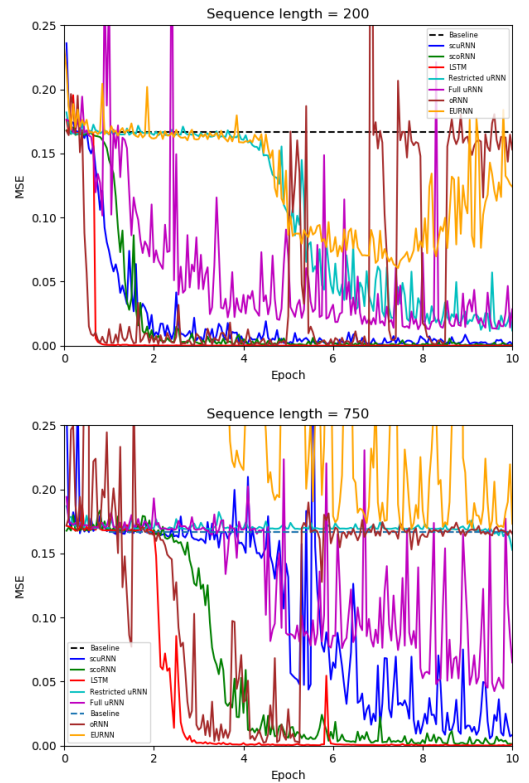

Figure 5: Test set MSE of each machine on the adding problem with sequence length T=200 (above) and sequence length T=750 (bottom).

10^{-3} for all other parameters.

In a minibatch, the sequences may have different lengths and they are padded with zeros so as to have a uniform length across the entire minibatch. Then the loss function for this experiment was the mean square error (MSE) as computed by taking the squared difference between the predicted and actual log magnitudes applied with a mask that zero out the padded entries, and then computing the resulting average over the entire batch including the zero entries. Table 2 includes the MSE for the validation and evaluation data sets for each model. As can be seen, the scuRNN model outperforms all other models. We suspect that scuRNN performs well on this data set because the complex architecture of scuRNN may be better suited to capture the complex dynamics of the underlying STFT sequences.

6 Conclusion

Orthogonal/unitary RNNs have shown promise in mitigating the well-known exploding or vanishing gradient problem. We have developed a new RNN architecture, scuRNN, that is designed to maintain a strict unitary recurrent weight matrix. A simple update scheme is used to optimize parameters using gradient descent or a related optimizer. This allows us to overcome the inherent difficulty of the scoRNN architecture by removing the need for an extra hyperparameter and allowing the diagonal scaling matrix parameters to

Table 2: Results for TIMIT speech set. Evaluation based on MSE

Model	n	# PARAMS	VALID MSE	EVAL. MSE
scuRNN	128	≈ 83k	3.94	3.56
scuRNN	258	≈ 200k	1.84	1.67
scoRNN	224	≈ 83k	4.76	4.31
scoRNN	425	≈ 200k	2.64	2.39
LSTM	84	≈ 83k	11.89	10.86
LSTM	158	≈ 200k	9.73	8.86
R.uRNN	158	≈ 83k	15.57	18.51
R.uRNN	378	≈ 200k	16.00	15.15
F.uRNN	128	≈ 83k	15.07	14.58
F.uRNN	256	≈ 200k	14.96	14.69

be trainable and not fixed during the training process. Our experiments show that scuRNN can achieve comparable or better results than scoRNN and other orthogonal and unitary RNN architectures. We have also discussed a potential singularity in the modReLU activation function, which may have implications on other complex neural networks involving the modReLU function.

References

Arjovsky, M.; Shah, A.; and Bengio, Y. 2016. Unitary evolution recurrent neural networks. In *Proceedings of ICML 2016*, volume 48, 1120–1128. JMLR.org.

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

Cho, K.; van Merrienboer, B.; Gülçehre, Ç.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR* abs/1406.1078.

Garofolo, J. S.; Lamel, L. F.; Fisher, W. M.; Fiscus, J. G.; Pallett, D. S.; Dahlgren, N. L.; and Zue, V. 1993. Timit acousic-phonetic coninuous speech corpus ldc93sl. *Technical report.Philadelphia: Linguistic Data Consortium.*

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS*, volume 9. Sardinia, Italy: JMLR: W&CP.

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.

Henaff, M.; Szlam, A.; and LeCun, Y. 2017. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of ICML 2017*, volume 48. New York, NY: JMLR: W&CP.

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

Hunger, R. 2007. An introduction to complex differential and complex differentiability.

Hyland, S. L., and Gunnar, R. 2017. Learning unitary operators with help from u(n). In *Proceedings of AAAI 2017*, 2050–2058.

Jing, L.; Shen, Y.; Dubcek, T.; Peurifoy, J.; Skirlo, S. A.; Tegmark, M.; and Soljacic, M. 2016. Tunable efficient unitary neural networks (EUNN) and their application to RNN. *CoRR* abs/1612.05231.

Jing, L.; Gülçehre, C.; Peurifoy, J.; Shen, Y.; Tegmark, M.; Soljačić, M.; and Bengio, Y. 2017. Gated orthogonal recurrent units: On learning to forget.

Kahan, W. 2006. Is there a small skew cayley transform with zero diagonal? *Linear algebra and its applications* 417(2-3):335–341.

Kreutz-Delgado, K. 2009. The complex gradient operator and the cr-calculus.

Le, Q. V.; Jaitly, N.; and Hinton, G. E. 2015. A simple way to initialize recurrent networks of rectified linear units.

LeCun, Y., and Cortes, C. 2010. MNIST handwritten digit database.

Maduranga, K. D. G.; Helfrich, K.; and Ye, Q. 2018. Complex unitary recurrent neural networks using scaled cayley transform. arXiv:1811.04142.

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, 2401–2409. PMLR.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of ICML 2010*, 807–814. USA: Omnipress.

O'Dorney, E. 2014. Minimizing the cayley transform of an orthogonal matrix by multiplying by signature matrices. 448:97–103.

Trabelsi, C.; Bilaniuk, O.; Zhang, Y.; Serdyuk, D.; Subramanian, S.; Santos, J. F.; Mehri, S.; Rostamzadeh, N.; Bengio, Y.; and Pal, C. 2018. Deep complex networks. arxiv:1705.09792.

Vorontsov, E.; Trabelsi, C.; Kadoury, S.; and Pal, C. 2017. On orthogonality and learning recurrent networks with long term dependencies. *arXiv preprint arXiv:1702.00071*.

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*. Curran Associates, Inc. 4880–4888.

Wolter, M., and Yao, A. 2018. Gated Complex Recurrent Neural Networks. *arXiv preprint arXiv:1806.08267*.