ORTHOGONAL RECURRENT NEURAL NETWORKS WITH SCALED CAYLEY TRANSFORM

Kyle E. Helfrich*, **Devin Willmott***, **& Qiang Ye**
Department of Mathematics
University of Kentucky
Lexington, KY 40506, USA
{kyle.helfrich,devin.willmott,qye3}@uky.edu

ABSTRACT

Recurrent Neural Networks (RNNs) are designed to handle sequential data but suffer from vanishing or exploding gradients. Recent work on Unitary Recurrent Neural Networks (uRNNs) have been used to address this issue and in some cases, exceed the capabilities of Long Short-Term Memory networks (LSTMs). We propose a simpler and novel update scheme to maintain orthogonal recurrent weight matrices without using complex valued matrices. This is done by parametrizing with a skew-symmetric matrix using the Cayley transform. Such a parametrization is unable to represent matrices with negative one eigenvalues, but this limitation is overcome by scaling the recurrent weight matrix by a diagonal matrix consisting of ones and negative ones. The proposed training scheme involves a straightforward gradient calculation and update step. In several experiments, the proposed scaled Cayley orthogonal recurrent neural network (scoRNN) achieves superior results with fewer trainable parameters than other unitary RNNs.

1 INTRODUCTION

Deep neural networks have been used to solve numerical problems of varying complexity. RNNs have parameters that are reused at each time step of a sequential data point and have achieved state of the art performance on many sequential learning tasks. Nearly all optimization algorithms for neural networks involve some variant of gradient descent. One major obstacle to training RNNs with gradient descent is due to *vanishing* or *exploding* gradients, as described in Bengio et al. (1993) and Pascanu et al. (2013). This problem refers to the tendency of gradients to grow or decay exponentially in size, resulting in gradient descent steps that are too small to be effective or so large that the network oversteps the local minimum. This issue significantly diminishes RNNs' ability to learn time-based dependencies, particularly in problems with long input sequences.

A variety of architectures have been introduced to overcome this difficulty. The current preferred RNN architectures are those that introduce gating mechanisms to control when information is retained or discarded, such as LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Cho et al., 2014), at the cost of additional trainable parameters. More recently, a number of papers have explored constraints and schemes that force the recurrent weight matrix to remain orthogonal or unitary throughout training. The first breakthrough appears in a paper by Arjovsky et al. (2016), which uses unitary matrices of a special form constructed from diagonal and rotation matrices in tandem with the discrete Fourier transform. The resulting network, called unitary evolution recurrent neural networks (uRNNs), exhibited superior performance to LSTMs on a variety of synthetic and real-world tasks. For clarity, we follow the convention of Wisdom et al. (2016) and refer to this network as the restricted-capacity uRNN.

Since the introduction of uRNNs, orthogonal and unitary RNN schemes have increased in both popularity and complexity. Wisdom et al. (2016) use a multiplicative update method detailed in Tagare (2011) and Wen & Yin (2013) to expand uRNNs' capacity to include all unitary matrices. These networks are referred to as full-capacity uRNNs. Jing et al. (2016)'s EURNN parametrizes this same

*Equal contributor

space with Givens rotations, while Jing et al. (2017)'s GORU introduces a gating mechanism for unitary RNNs to enable short term memory. Vorontsov et al. (2017) introduced modified optimization and regularization methods that restrict the singular values of the recurrent matrix to an interval around 1. Each of these methods involve complex valued recurrent weights. For additional related work in addressing the vanishing and exploding gradient problem, see Henaff et al. (2017) and Le et al. (2015).

In this paper, we consider RNNs with a recurrent weight matrix taken from the set of all orthogonal matrices. To construct the orthognal weight matrix, we parametrize it with a skew-symmetric matrix through a scaled Cayley transform. This scaling allows us to avoid the singularity issue occuring for -1 eigenvalues that may arise in the standard Cayley transform. With the parameterization, the network optimization involves a relatively simple gradient descent update. The resulting method achieves superior performance on sequential data tasks with a smaller number of trainable parameters and hidden sizes than other unitary RNNs and LSTMs.

The method we present in this paper works entirely with real matrices, and as such, our results deal only with orthogonal and skew-symmetric matrices. However, the method and all related theory remain valid for unitary and skew-Hermitian matrices in the complex case. The experimental results in this paper indicate that state of the art performance can be achieved without the increased complexity of optimization along the Stiefel manifold and using complex matrices.

2 BACKGROUND

2.1 ORTHOGONAL & UNITARY MATRICES

A real matrix W is orthogonal if it satisfies $W^T W = I$. The complex analog of orthogonal matrices are unitary matrices, which satisfy $W^* W = I$, where $*$ denotes the conjugate transpose. Orthogonal and unitary matrices have the desirable property that $\|Wx\|_2 = \|x\|_2$ for any vector x. This property motivates the use of orthogonal or unitary matrices in RNNs to avoid vanishing and exploding gradients, as detailed in Arjovsky et al. (2016).

2.2 RECURRENT NEURAL NETWORKS

A recurrent neural network (RNN) is a function with input parameters $U \in \mathbb{R}^{n \times m}$, recurrent parameters $W \in \mathbb{R}^{n \times n}$, recurrent bias $b \in \mathbb{R}^n$, output parameters $V \in \mathbb{R}^{p \times n}$, and output bias $c \in \mathbb{R}^p$ where m is the data input size, n is the number of hidden units, and p is the output data size. From an input sequence $x = (x_1, x_2, ..., x_T)$ where $x_i \in \mathbb{R}^m$, the RNN returns an output sequence $y = (y_1, y_2, ..., y_T)$ where each $y_i \in \mathbb{R}^p$ is given recursively by

$$
\begin{aligned}
h_t &= \sigma\left(U x_t + W h_{t-1} + b\right) \\
y_t &= V h_t + c
\end{aligned}
$$

where $h = (h_0, \ldots, h_{T-1})$, $h_i \in \mathbb{R}^n$ is the hidden layer state at time i and $\sigma(\cdot)$ is the activation function, which is often a pointwise nonlinearity such as a hyperbolic tangent function or rectified linear unit (Nair & Hinton, 2010).

2.3 UNITARY RNNS

Arjovsky et al. (2016) follow the framework of the previous section for their restricted-capacity uRNN, but introduce a parametrization of the recurrent matrix W using a product of simpler matrices. This parametrization is given by

$$
W = D_3 R_2 \mathcal{F}^{-1} D_2 P R_1 \mathcal{F} D_1
$$

where D are diagonal matrices with complex norm 1 diagonal elements, R are complex Householder reflection matrices, $\mathcal{F}$ is a discrete Fourier transform, and P is a fixed permutation matrix. Because W is a product of both $\mathcal{F}$ and $\mathcal{F}^{-1}$, as well as other unitary matrices, W itself is unitary.

Wisdom et al. (2016) note that this representation has only $7n$ parameters, which is insufficient to represent all unitary matrices for $n > 7$. In response, they present the full-capacity uRNN, which uses a multiplicative update step that is able to reach all unitary matrices of order n.

The full-capacity uRNN aims to construct a unitary matrix $W^{(k+1)}$ from $W^{(k)}$ by moving along a curve on the Stiefel manifold $\{W \in \mathbb{C}^{n\times n} \mid W^*W = I\}$. For the network optimization, it is necessary to use a curve that is in a descent direction of the cost function $L := L(W)$. In Tagare (2011), Wen & Yin (2013), and Wisdom et al. (2016), a descent direction is constructed as $B^{(k)}W^{(k)}$, which is a representation of the derivative operator $DL(W^{(k)})$ in the tangent space of the Stiefel manifold at $W^{(k)}$, where

$$B^{(k)} = \left(\frac{\partial L}{\partial W}\right)^* W^{(k)} - \left(W^{(k)}\right)^* \frac{\partial L}{\partial W}.$$

Then, with $B^{(k)}W^{(k)}$ defining the direction of a descent curve, an update along the Stiefel manifold is obtained as

$$W^{(k+1)} = \left(I + \frac{\lambda}{2}B^{(k)}\right)^{-1} \left(I - \frac{\lambda}{2}B^{(k)}\right) W^{(k)} \tag{1}$$

where λ is the learning rate.

3 Scaled Cayley Orthogonal RNN

In this section, we present an RNN with an orthogonal recurrent weight matrix. First we discuss the scaled Cayley transform and then we derive a differentiation formula for updating the recurrent weight matrix.

3.1 Cayley Transform

Given a (real) skew-symmetric matrix A (i.e. $A^T = -A$), the Cayley transform

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

defines an orthogonal matrix. Such an orthogonal matrix W cannot have -1 as an eigenvalue. Conversely, if W is an orthogonal matrix that does not have -1 as an eigenvalue, it is the Cayley transform of the skew-symmetric matrix A defined by the same transform

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

Thus, the Cayley transform forms a bijection between the set of orthogonal matrices without -1 eigenvalues and the set of skew-symmetric matrices.

We can use this bijection to parametrize the set of orthogonal matrices without -1 eigenvalues using skew-symmetric matrices. This parametrization is attractive from a machine learning perspective because it is closed under addition: the sum or difference of two skew-symmetric matrices is also skew-symmetric, so we can use gradient descent algorithms like RMSprop or Adam to train parameters.

However, this parametrization cannot represent orthogonal matrices with -1 eigenvalues, since in this case $I + W$ is not invertible. Theoretically, we can still represent matrices with eigenvalues that are arbitrarily close to -1; however, it can require large entries of A. For example, suppose we would like to learn the following 2x2 orthogonal matrix W with eigenvalues $\approx -0.99999 \pm 0.00447i$, as parametrized by its Cayley transform, A:

$$W = \begin{bmatrix} -0.99999 & -\sqrt{1 - 0.99999^2} \\ \sqrt{1 - 0.99999^2} & -0.99999 \end{bmatrix} \qquad A \approx \begin{bmatrix} 0 & 447.212 \\ -447.212 & 0 \end{bmatrix}$$

Gradient descent algorithms will learn this A matrix very slowly, if at all.

This difficulty can be overcome through a suitable diagonal scaling according to results from O'Dorney (2014) and Kahan (2006).

Theorem 3.1 *Every orthogonal matrix W can be expressed as*

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

where $A = [a_{ij}]$ is real-valued, skew-symmetric with $|a_{ij}| \leq 1$, and D is diagonal with all nonzero entries equal to ± 1. Similarly, every unitary matrix U can be expressed as

$$U = (I + S)^{-1}(I - S)\Phi$$

where $S = [s_{ij}]$ is skew-Hermitian with $|s_{i,j}| \leq 1$ and Φ is a unitary diagonal matrix.

Proof: See O'Dorney (2014) for the orthogonal case, and Kahan (2006) for the unitary case.■

We call the transform in Theorem 3.1 the scaled Cayley transform. Then, with an appropriate choice of D, the scaled Cayley transform can reach any orthogonal matrices including those with -1 eigenvalues. Further, it ensures that the skew-symmetric matrix A that generates the orthogonal matrix will be bounded. This allows us to construct any orthogonal matrix using a skew-symmetric A with moderately sized entries.

Our proposed network, the scaled Cayley orthogonal recurrent neural network (scoRNN), is based on this theorem. We parametrize the recurrent weight matrix W through a skew-symmetric matrix A, which results in $\frac{n(n-1)}{2}$ trainable weights. The recurrent matrix W is formed by the scaled Cayley transform: $W = (I+A)^{-1}(I-A)D$. The scoRNN then operates identically to the set of equations given in Section 2.2, but during training we update the skew-symmetric matrix A using gradient descent, while D is fixed throughout the training process. The number of -1s on the diagonal of D, which we call ρ, is considered a hyperparameter in this work and is manually chosen based on the task.

3.2 Update Scheme

We now derive the update scheme for the recurrent weight matrix W. This is done through an update on the skew-symmetric parameterization matrix A. We compute the gradients by backpropagating through the Cayley transform. The next theorem describes how to compute these gradients with respect to A.

Theorem 3.2 *Let $L = L(W) : \mathbb{R}^{n\times n} \to \mathbb{R}$ be some differentiable cost function for an RNN with the recurrent weight matrix W. Let $W = W(A) := (I + A)^{-1}(I - A)D$ where $A \in \mathbb{R}^{n\times n}$ is skew-symmetric and $D \in \mathbb{R}^{n\times n}$ is a fixed diagonal matrix consisting of -1 and 1 entries. Then the gradient of $L = L(W(A))$ with respect to A is*

$$\frac{\partial L}{\partial A} = V^T - V \tag{2}$$

where $V := (I - A)^{-1}\frac{\partial L}{\partial W}\left(D + W^T\right)$ and we have used the notation $\frac{\partial L}{\partial A} = \left[\frac{\partial L}{\partial A_{i,j}}\right] \in \mathbb{R}^{n\times n}$ and $\frac{\partial L}{\partial W} = \left[\frac{\partial L}{\partial W_{i,j}}\right] \in \mathbb{R}^{n\times n}$

Proof: Let $Z := (I + A)^{-1}(I - A)$. We consider the (i, j) entry of $\frac{\partial L}{\partial A}$. Taking the derivative with respect to $A_{i,j}$ where $i \neq j$ we obtain:

$$\frac{\partial L}{\partial A_{i,j}} = \sum_{k,l=1}^{n} \frac{\partial L}{\partial W_{k,l}}\frac{\partial W_{k,l}}{\partial A_{i,j}} = \sum_{k,l=1}^{n} \frac{\partial L}{\partial W_{k,l}} D_{l,l}\frac{\partial Z_{k,l}}{\partial A_{i,j}} = \operatorname{tr}\left[\left(\frac{\partial L}{\partial W}D\right)^T \frac{\partial Z}{\partial A_{i,j}}\right]$$

Using the identity $(I + A)\,Z = I - A$ and taking the derivative with respect to $A_{i,j}$ to both sides we obtain:

$$\frac{\partial Z}{\partial A_{i,j}} + \frac{\partial A}{\partial A_{i,j}} Z + A \frac{\partial Z}{\partial A_{i,j}} = -\frac{\partial A}{\partial A_{i,j}}$$

and rearranging we get:

$$\frac{\partial Z}{\partial A_{i,j}} = -\left(I + A\right)^{-1} \left(\frac{\partial A}{\partial A_{i,j}} + \frac{\partial A}{\partial A_{i,j}} Z \right)$$

Let $E_{i,j}$ denote the matrix whose (i,j) entry is 1 with all others being 0. Since A is skew-symmetric, we have $\frac{\partial A}{\partial A_{i,j}} = E_{i,j} - E_{j,i}$. Combining everything, we have:

$$\begin{aligned}
\frac{\partial L}{\partial A_{i,j}} &= -\operatorname{tr}\left[\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \left(E_{i,j} - E_{j,i} + E_{i,j} Z - E_{j,i} Z \right) \right] \\
&= -\operatorname{tr}\left[\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} E_{i,j} \right] + \operatorname{tr}\left[\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} E_{j,i} \right] \\
&\quad - \operatorname{tr}\left[\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} E_{i,j} Z \right] + \operatorname{tr}\left[\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} E_{j,i} Z \right] \\
&= -\left[\left(\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \right)^T \right]_{i,j} + \left[\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \right]_{i,j} \\
&\quad - \left[\left(\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \right)^T Z^T \right]_{i,j} + \left[Z \left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \right]_{i,j} \\
&= \left[\left(I + Z\right) \left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \right]_{i,j} - \left[\left(\left(\frac{\partial L}{\partial W} D \right)^T \left(I + A\right)^{-1} \right)^T \left(I + Z^T\right) \right]_{i,j} \\
&= \left[\left(I + Z\right) \left(\frac{\partial L}{\partial W} D \right)^T \left(I - A^T\right)^{-1} \right]_{i,j} - \left[\left(I - A\right)^{-1} \frac{\partial L}{\partial W} D \left(I + Z^T\right) \right]_{i,j}
\end{aligned}$$

Using the above formulation, $\frac{\partial L}{\partial A_{j,j}} = 0$ and $\frac{\partial L}{\partial A_{i,j}} = -\frac{\partial L}{\partial A_{j,i}}$ so that $\frac{\partial L}{\partial A}$ is a skew-symmetric matrix. Finally, by the definition of V we get the desired result. ■

From the above theorem, at each training step of scoRNN, we first use the standard backpropagation algorithm to compute $\frac{\partial L}{\partial W}$, from which $\frac{\partial L}{\partial A}$ is computed by (2). We then update the A matrix by implementing an optimizer such as gradient descent

$$A^{(k+1)} = A^{(k)} + \gamma \frac{\partial L}{\partial A^{(k)}}$$

where γ is the learning rate. Then we construct the recurrent weight matrix as

$$W^{(k+1)} = \left(I + A^{(k+1)}\right)^{-1} \left(I - A^{(k+1)}\right) D \tag{3}$$

Since $\frac{\partial L}{\partial A}$ is skew-symmetric, then $A^{(k+1)}$ will be skew-symmetric and, in turn, $W^{(k+1)}$ will be orthogonal.

Comparing our scoRNN and the full-capacity uRNNs from Section 2.3, they both have the capacity to optimize the network with respect to a general orthogonal or unitary matrix. However, the training iterations are different; namely, from the same weight matrix $W^{(k)}$, the new weight matrix $W^{(k+1)}$

as constructed by either the full-capacity uRNN via (1), or the scoRNN by (3) are different. In particular, the full-capacity uRNN performs a multiplicative update that goes in the direction of a certain projection of the gradient $\frac{\partial L}{\partial W}$ in the tangent space of the Stiefel manifold. This projection can be shown to be a descent direction but not necessary the steepest one. In contrast, scoRNN performs an additive update in the direction of steepest descent with respect to its parametrization.

4 Other Architecture Details

The basic architecture of scoRNN is very similar to the standard RNN as presented in Section 2.2. From a network layer perspective, one can think of the application of the recurrent weight in a three layer process. Let $h_t \in \mathbb{R}^n$ be the current state of the scoRNN at a particular time step, t. We then pass h_t through the following layers:

- Layer 1: $h_t \to Dh_t =: h_t^{(1)}$
- Layer 2: $h_t^{(1)} \to (I - A)\, h_t^{(1)} =: h_t^{(2)}$
- Layer 3: $h_t^{(2)} \to (I + A)^{-1}\, h_t^{(2)} =: h_t^{(3)}$

Note that the above scheme is the same as taking $h_t \to Wh_t$ as discussed previously.

For each experiment, the number of negative ones on the diagional of D were manually adjusted to get the best results. In the unitary case, the diagonals would be of the form $D_{j,j} = e^{i\theta_j}$ where θ_j could be set as a trainable variable.

During the update step, it is possible to update the input and output weights using a different optimizer and learning rate than the recurrent matrix. During the experiments, several different combinations of optimizers were used with varying degress of success.

4.1 modReLU Activation Function

The modReLU function was first implemented by Arjovsky et al. (2016) to handle complex valued functions and weights. Unlike previous methods, our method only uses real-valued functions and weights. Nevertheless, we have found that the modReLU function in the real case also performed better than other activation functions. The function is defined as follows:

$$\begin{aligned}\sigma_{\text{modReLU}}(z) &= \begin{cases} (|z| + b)\,\frac{z}{|z|} & \text{if } |z| + b \geq 0 \\ 0 & \text{if } |z| + b < 0 \end{cases} \\ &= \frac{z}{|z|}\sigma_{\text{ReLU}}\left(|z| + b\right)\end{aligned}$$

In the real case, this simplifies to $\text{sign}(z)\sigma_{\text{ReLU}}(|z| + b)$. We believe that the improved performance of the modReLU over other activation functions is because it admits both positive and negative activation values, which is important for the state transition in orthogonal RNNs. This is similar to the hyperbolic tangent function, but unlike the hyperbolic tangent function the modReLU does not have vanishing gradient issues.

4.2 Initializations

We found that modifying the initialization of our parameter matrices, in particular our recurrent parameter matrix A, had a significant effect on performance. We found two initializations to be particular effective. The first initializes all elements of A to 0. This will generate an initial recurrent matrix of $W = (I - 0)^{-1}(I - 0)D = D$. With a scaling matrix of all 1s, this will give us an initial W equal to the identity; a recurrent matrix initialized in this manner has been shown to be particularly effective in certain contexts by Le et al. (2015).

Our other initialization method, hereafter referred to as unit circle initialization, uses a similar technique as Henaff et al. (2017). We initialize all of the entries of A to be 0 except for 2x2 blocks along the diagonal, which are given as

$$A = \begin{bmatrix} B_1 & & \\ & \ddots & \\ & & B_{\lfloor n/2 \rfloor} \end{bmatrix} \quad \text{where} \quad B_j = \begin{bmatrix} 0 & s_j \\ -s_j & 0 \end{bmatrix}$$

with $s_j = \sqrt{\frac{1-\cos(t_j)}{1+\cos(t_j)}}$, where t_j is sampled uniformly from $\left[0, \frac{\pi}{2}\right]$. The Cayley transform of this A will have eigenvalues equal to $\pm e^{it_j}$ for each j, which will be distributed uniformly along the right unit half-circle. Multiplication by the scaling matrix D will reflect ρ of these eigenvalues across the imaginary axis.

5 Experiments

Several experiments were performed to compare scoRNN with LSTM, restricted capacity uRNN, and full-capacity uRNN as discussed previously. Different optimizers, learning rates, and initializers were used as identified below.

5.1 Copying Problem

This experiment follows descriptions found in Arjovsky et al. (2016) and Wisdom et al. (2016), and tests an RNN's ability to reproduce a sequence seen many timesteps earlier. In the problem setup, there are 10 input classes, which we denote using the digits 0-9, with 0 being used as a 'blank' class and 9 being used as a 'marker' class. The RNN receives an input sequence of length $T + 20$. This sequence consists of entirely zeros, except for the first ten elements, which are uniformly sampled from classes 1-8, and a 9 placed ten timesteps from the end. The goal for the machine is to output zeros until it sees a 9, at which point it should output the ten elements from the beginning of the input sequence.

A baseline strategy with which to compare machine performance is that of outputting 0 until the machine sees a 9, and then outputting 10 elements randomly sampled from classes 1-8. The expected cross-entropy for such a strategy is $\frac{10 \log(8)}{T+20}$. In practice, it is common to see gated RNNs such as LSTMs converge to this local minimum.

We vary the number of hidden units of the machines to match the number of parameters, approximately 22k each. This results in an LSTM with $n = 68$, a restricted-capacity uRNN with $n = 470$, a full-capacity uRNN with $n = 128$, and a scoRNN with $n = 190$. We found the best performance

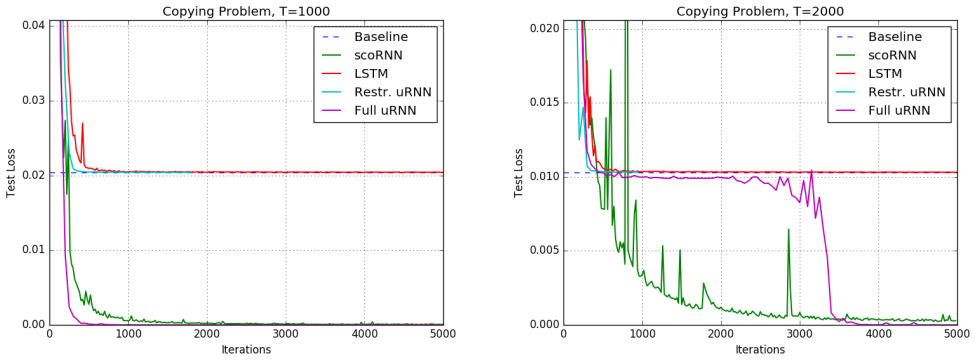

Figure 1: Cross entropy of each machine on the copying problem with time gaps of 1000 (left) and 2000 (right).

with the scoRNN came from unit circle initialization and $\rho = n/2$, which gives an initial W with eigenvalues distributed uniformly on the entire unit circle.

Figure 1 compares each machine's performance for $T = 1000$ and $T = 2000$, with the baseline solution cross entropy given as a dashed blue line. In both cases, the restricted-capacity uRNN and LSTM fail to find a zero entropy strategy, instead converging quickly to the baseline solution. For the $T = 1000$ test, the full-capacity uRNN and scoRNN converge immediately to zero entropy solutions. For $T = 2000$, the scoRNN error converges more gradually, but is able to bypass the baseline strategy entirely, which even the full-capacity uRNN remains stuck at for several thousand iterations before finding a correct solution.

5.2 Adding Problem

We examined a variation of the adding problem as proposed by Arjovsky et al. (2016) which is based on the work of Hochreiter & Schmidhuber (1997). This variation involves passing two sequences concurrently into the RNN, each of length T. The first sequence is a sequence of digits sampled uniformly with values ranging in a half-open interval, $\mathcal{U}[0, 1)$. The second sequence is a marker sequence consisting of all zeros except for two entries that are marked by one. The first 1 is located uniformly within the interval $[1, \frac{T}{2})$ of the sequence and the second 1 is located uniformly within the interval $[\frac{T}{2}, T)$ of the sequence. The label for each pair of sequences is the sum of the two entries that are marked by one. Naively predicting one regardless of the sequence gives an expected mean squared error (MSE) of approximately 0.167. This will be considered as the baseline.

The number of hidden units for each network were adjusted so that each had approximately 14k trainable parameters. This results in $n = 170$ for the scoRNN, $n = 60$ for the LSTM, $n = 120$ for the Full-Capacity uRNN, and $n = 950$ hidden units for the restricted-capacity uRNN.

The test set MSE results for sequence lengths $T = 200$, $T = 400$, and $T = 750$ can be found in Figure 2. A training set size of 100,000 and a testing set size of 10,000 were used for each sequence length. For each case, the networks start at or near the baseline MSE and drop towards zero after a few epochs. As the sequence length increases, the number of epochs before the drop increases. We found the best initialization for the scoRNN was the unit circle with $\rho = n/2$. As can be seen, the LSTM appears to drop first followed by the full-capacity uRNN, scoRNN, and the restricted-capacity uRNN. Although the full-capacity uRNN drops below the baseline before the scoRNN, the full-capacity uRNN does not drop as quickly and has a more irregular descent curve.

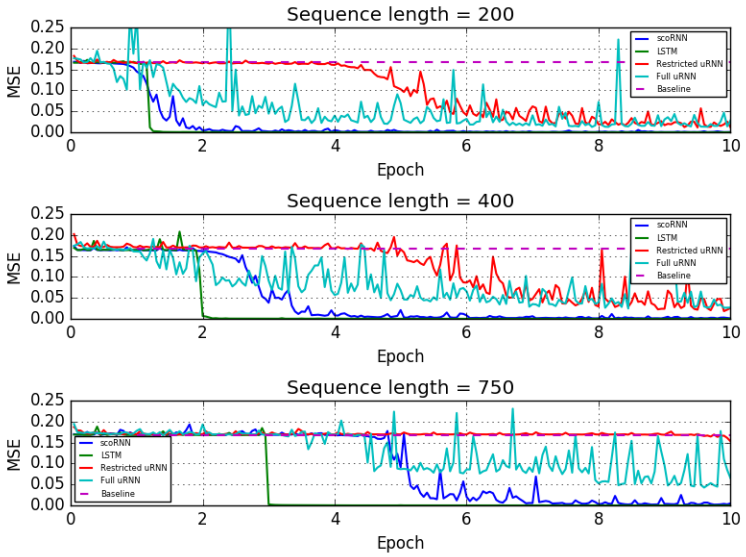

Figure 2: Test set MSE of each machine on the adding problem with sequence lengths of 200 (top), 400 (middle), and 750 (bottom).

5.3 PIXEL-BY-PIXEL MNIST

This experiment involves classifying samples from the well-known MNIST dataset (LeCun et al.). Following the implementation of Le et al. (2015), each pixel is fed into the RNN sequentially. Since each image is 28 pixels by 28 pixels, this results in a single pixel sequence length of 784. All machines were trained with the RMSProp optimization algorithm (Hinton, 2012). The recurrent parameters in the scoRNN with $n = 170$ had a learning rate of 10^{-4}, while the recurrent parameters in the scoRNN with $n = 360$ hidden units had a learning rate of 10^{-5}. All other parameters had a learning rate of 10^{-3}. We also used unit circle initialization and $\rho = n/10$.

Our experiment uses a training set of 55,000 images and a test set of 10,000 testing images. Each machine was trained for 70 epochs, and test set accuracy was evaluated at the conclusion of each epoch. Figure 3 shows test set accuracy over time for each machine, and the best performance over all epochs by each machine is given in Table 1.

As in experiments presented in Arjovsky et al. (2016) and Wisdom et al. (2016), orthogonal and unitary RNNs are unable to outperform the LSTM on this task. However, the 360 hidden unit scoRNN performs the best of these, and the 170 hidden unit scoRNN gives comparable performance to both of the 512 hidden unit uRNNs using a much smaller hidden dimension and in the case of the full-capacity uRNN, an order of magnitude fewer parameters.

Table 1: Results for unpermuted pixel-by-pixel MNIST. Evaluation accuracies are based on the best test accuracy at the end of every epoch.

Model	n	# parameters	Test Accuracy
scoRNN	170	$\approx$ 16k	0.973
scoRNN	360	$\approx$ 69k	0.983
LSTM	128	$\approx$ 68k	0.987
LSTM	256	$\approx$ 270k	0.989
Restricted-capacity uRNN	512	$\approx$ 16k	0.976
Full-capacity uRNN	116	$\approx$ 16k	0.947
Full-capacity uRNN	512	$\approx$ 270k	0.974

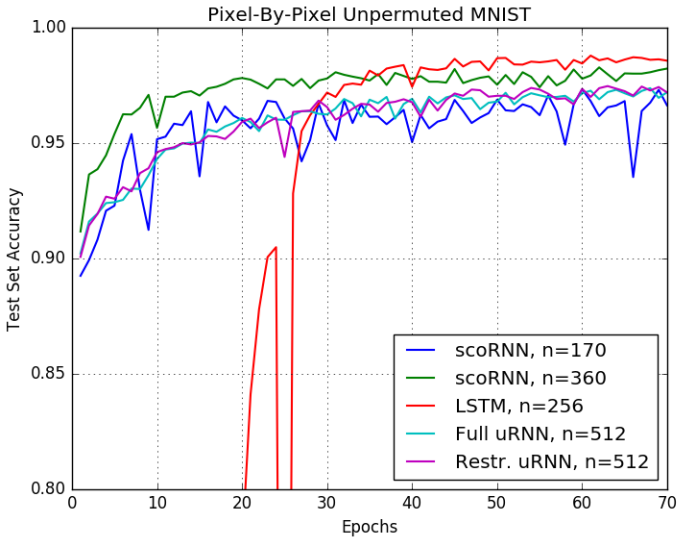

Figure 3: Test accuracy for unpermuted MNIST over time.

5.4 PERMUTED PIXEL-BY-PIXEL MNIST

This experiment follows the same setup as the pixel-by-pixel MNIST experiment except that a fixed permutation is applied to the training and test data sets. The architecture of the scoRNN was identical to the unpermuted case except we found that setting ρ to $n/2$, resulting in a scaling matrix diagonal with half ones and half negative ones, increased accuracy. We suspect that this difference from the unpermuted case, where we found $\rho = n/10$ to be optimal, comes from the different types of dependencies in each: the unpermuted MNIST experiment has mostly local dependencies, which seem to benefit from a lower proportion of -1s in D, while permuted MNIST requires learning many long-term dependencies, which appears to be more easily modeled when the diagonal of D is half 1s and half -1s.

The experiment setup was the same as in the unpermuted task. Results can be found in Table 2 and Figure 4. As in the unpermuted case, the smaller scoRNN does as well as restricted-capacity and full-capacity uRNNs with triple the number of hidden units. The 360 hidden unit scoRNN achieves a test-set accuracy of 96.2%, outperforming all of the uRNNs and both sizes of LSTM. We believe this is a state of the art result on this task.

Table 2: Results for permuted pixel-by-pixel MNIST. Evaluation accuracies are based on the best test accuracy at the end of every epoch.

Model	n	# parameters	Test Accuracy
scoRNN	170	$\approx$ 16k	0.943
scoRNN	360	$\approx$ 69k	0.962
LSTM	128	$\approx$ 68k	0.920
LSTM	256	$\approx$ 270k	0.929
Restricted-capacity uRNN	512	$\approx$ 16k	0.945
Full-capacity uRNN	116	$\approx$ 16k	0.925
Full-capacity uRNN	512	$\approx$ 270k	0.947

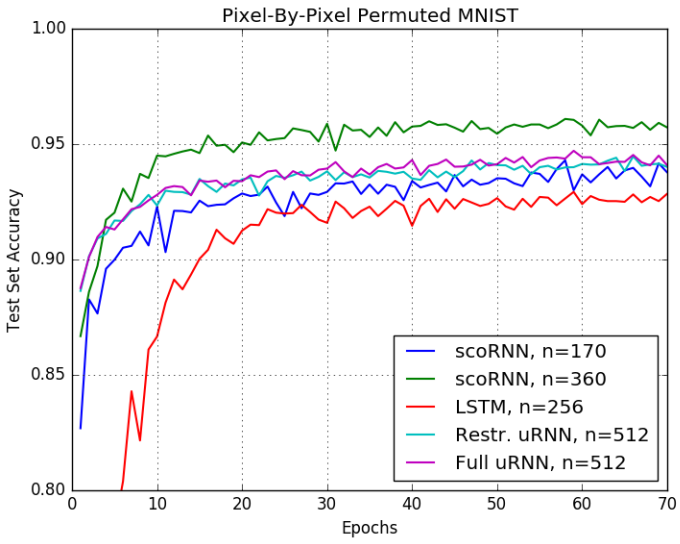

Figure 4: Test accuracy for permuted MNIST over time.

6 CONCLUSION

There have been recent breakthroughs with RNNs using unitary recurrent weight matrices. These uRNNs are implemented with complex valued matrices and so require additional complexity in architectures and computations. Unlike the uRNNs, the scoRNN developed here can use real valued orthogonal recurrent weight matrices with a simpler implementation scheme. Results from the copying, unpermuted and permuted MNIST, and adding problem tasks show that scoRNNs can achieve superior performance. Its mathematical simplicity and ease with implementation is another important advantage in applications.

Acknowledgements: The research of Qiang Ye was supported in part by NSF Grants DMS-1317424 and DMS-1620082.

REFERENCES

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *Proceedings of the 2016 International Conference on Machine Learning*, volume 48, pp. 1120–1128, New York, New York, USA, 2016. JMLR.

Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. pp. 1183–1195, San Francisco, CA, USA, 1993. IEEE Press.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014. URL `https://arxiv.org/abs/1409.1259`.

Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48, New York, NY, USA, 2017. JMLR: W&CP.

Geoffery Hinton. Neural networks for machine learning. Coursera, video lectures, 2012.

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

Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Max Tegmark, and Marin Soljačić. Tunable efficient unitary neural networks (eunn) and their application to rnn. *arXiv preprint arXiv:1612.05231*, 2016.

Li Jing, Çalar Gülçehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljačić, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. 2017. URL `https://arxiv.org/abs/1706.02761`.

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

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units, 2015. URL `https://arxiv.org/abs/1504.00941`.

Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database. URL `http://yann.lecun.com/exdb/mnist/`.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *27th International Conference on Machine Learning*, Haifa, Israel, 2010.

Evan O'Dorney. Minimizing the cayley transform of an orthogonal matrix by multiplying by signature matrices. 448:97103, 05 2014.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *30th International Conference on Machine Learning*, Atlanta, Georgia, USA, 2013.

Hemant D. Tagare. Notes on optimization on stiefel manifolds. Technical report, Yale University, 2011.

Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. *arXiv preprint arXiv:1702.00071*, 2017.

Zaiwen Wen and Wotao Yin. A feasible method for optimization with orthogonality constraints. In *Mathematical Programming*, volume 142(1-2), pp. 397–434. 2013.

Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4880–4888. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6327-full-capacity-unitary-recurrent-neural-networks.pdf`.