Eigenvalue Normalized Recurrent Neural Networks for Short Term Memory

Kyle Helfrich, Qiang Ye
Mathematics Department, University of Kentucky
Lexington, Kentucky 40509, United States
{kyle.helfrich, qye3}@uky.edu

Abstract

Several variants of recurrent neural networks (RNNs) with orthogonal or unitary recurrent matrices have recently been developed to mitigate the vanishing/exploding gradient problem and to model long-term dependencies of sequences. However, with the eigenvalues of the recurrent matrix on the unit circle, the recurrent state retains all input information which may unnecessarily consume model capacity. In this paper, we address this issue by proposing an architecture that expands upon an orthogonal/unitary RNN with a state that is generated by a recurrent matrix with eigenvalues in the unit disc. Any input to this state dissipates in time and is replaced with new inputs, simulating short-term memory. A gradient descent algorithm is derived for learning such a recurrent matrix. The resulting method, called the Eigenvalue Normalized RNN (ENRNN), is shown to be highly competitive in several experiments.

1 Introduction

Recurrent neural networks (RNNs) are a type of deep neural network that are designed to handle sequential data. The underlying dynamical system carries temporal information from one time step to another and captures potential dependencies among the terms of a sequence. Like other deep neural networks, the weights of an RNN are learned by gradient descent. For the input at a time step to affect the output at a later time step, the gradients must back-propagate through each step. Since a sequence can be quite long, RNNs are prone to suffer from *vanishing* or *exploding* gradients as described in (Bengio, Frasconi, and Simard 1993) and (Pascanu, Mikolov, and Bengio 2013). One consequence of this well-known problem is the difficulty of the network to model input-output dependency over a large number of time steps.

There have been many different architectures that are designed to mitigate this problem. The most popular RNN architectures such as LSTMs (Hochreiter and Schmidhuber 1997) and GRUs (Cho et al. 2014), incorporate a gating mechanism to explicitly retain or discard information. More recently, several different RNNs have been developed to maintain either a unitary or orthogonal recurrent weight matrix such as the unitary evolution RNN (uRNN) (Arjovsky, Shah, and Bengio 2016), Full-Capacity uRNN (Wisdom et al. 2016), EUNN (Jing et al. 2016), oRNN (Mhammedi et al. 2017), scoRNN (Helfrich, Willmott, and Ye 2018; Maduranga, Helfrich, and Ye 2019), Spectral RNN (Zhang, Lei, and Dhillon 2018), nnRNN (Kerg et al. 2019), and EXPRNN (Lezcano-Casado 2019; Lezcano-Casado and Martinez-Rubio 2019). There is also work using unitary matrices in GRUs such as the GORU (Jing et al. 2017). For other work addressing the vanishing/exploding gradient problem, see (Henaff, Szlam, and LeCun 2016; Le, Jaitly, and Hinton 2015; Wolter and Yao 2018).

In spite of the promises shown in recent work, orthogonal/unitary RNNs still have some shortcomings. While an orthogonal RNN allows propagation of information over many time steps, it has an undesirable effect that all input information may be retained in all future states. Unlike gated architectures, orthogonal RNNs lack "forget" mechanisms (Jing et al. 2017) to discard unwanted information. This consumes model capacity, making it difficult to efficiently model sequences with both long and short-term dependency.

In this paper, we expand upon the orthogonal/unitary RNN architecture by incorporating a dissipative state to model short-term dependencies. We call this model the Eigenvalue Normalized RNN (ENRNN). Inspired by the work on the Spectral Normalized Generative Adversarial Network (SN-GAN) (Miyato et al. 2018), we construct a recurrent matrix with its spectral radius (i.e. the largest absolute value of the eigenvalues) less than 1 through normalizing another parametric matrix by its spectral radius. A gradient descent algorithm is also derived that maintains this spectral radius property. Any input to this state will dissipate in time with repeat multiplication by the recurrent matrix and will be replaced with new input information, emulating a short-term memory state. This state can be concatenated with another state with an orthogonal/unitary recurrent matrix to form an RNN that has a long and short-term memory component to efficiently model long sequences. The resulting architecture falls within the existing framework of the basic RNN and is shown to be highly competitive in several experiments.

2 Background and Related Work

An RNN takes an input sequence of length τ, denoted by $X_\tau = (x_1, x_2, ..., x_\tau)$, and produces an output sequence $Y_\tau = (y_1, y_2, ..., y_\tau)$ where $x_t \in \mathbb{R}^m$ and $y_t \in \mathbb{R}^p$. The basic architecture consists of an input weight matrix $U \in \mathbb{R}^{n\times m}$, recurrent weight matrix $W \in \mathbb{R}^{n\times n}$, bias vector $b \in \mathbb{R}^n$, output weight matrix $V \in \mathbb{R}^{p\times n}$, and output bias vector $c \in \mathbb{R}^p$. If $\sigma(\cdot)$ is an activation function that is applied pointwise, then the hidden state $h_t \in \mathbb{R}^n$ and output y_t at time t is given by

$$h_t = \sigma\left(Ux_t + Wh_{t-1} + b\right); y_t = Vh_t + c$$

A problem with RNNs is that the gradient of y_τ with respect to h_t involves repeat multiplication by the recurrent matrix W. If the spectral radius of W is less than one, gradients vanish but if the spectral radius is greater than one, gradients explode (Bengio, Frasconi, and Simard 1993; Pascanu, Mikolov, and Bengio 2013). One way to mitigate this issue is to use an orthogonal/unitary recurrent weight matrix which will preserve vector norms. Early work has shown simply initializing the recurrent matrix as identity or orthogonal may improve performance (Henaff, Szlam, and LeCun 2016; Le, Jaitly, and Hinton 2015). Several methods have also been developed that maintain an orthogonal/unitary recurrent matrix through different parameterizations. The uRNN (Arjovsky, Shah, and Bengio 2016) parameterizes W by a product of some special unitary matrices. The Full-Capacity uRNN (Wisdom et al. 2016) optimizes W along the manifold of unitary matrices. The EUNN (Jing et al. 2016) parameterizes W as a product of Givens rotation matrices, while the oRNN (Mhammedi et al. 2017) uses a product of Householder reflections. The scoRNN (Helfrich, Willmott, and Ye 2018; Maduranga, Helfrich, and Ye 2019) parameterizes W by using a skew-symmetric or skew-Hermitian matrix through a scaled Cayley transform. The EXPRNN (Lezcano-Casado and Martinez-Rubio 2019; Lezcano-Casado 2019) uses an exponential map. There has also been work in using recurrent matrices that are near orthogonal by constraining singular values within a small distance of 1; see (Vorontsov et al. 2017; Zhang, Lei, and Dhillon 2018). These models have demonstrated that orthogonal/unitary RNNs can mitigate the vanishing/exploding gradient problem and successfully model long sequences.

Gated networks, such as LSTM and GRU, are popular RNN architectures that use a gating mechanism to control passing of long or short-term memory. Although quite successful, the LSTM is still prone to exploding gradients and may still require gradient clipping. (Jing et al. 2017) considers GRU with an orthogonal recurrent matrix. However, using an orthogonal matrix with a gated network may not have the same benefits of passing long-term information as in an orthogonal RNN. Multiscale RNNs (Schmidhuber 1992; Hihi and Bengio 1995; Koutník et al. 2014) stack multiple layers of RNNs whose states are updated in different time scales at different layers to process short and long-term information, but the difficulty is their need to determine the boundary structures defining different layers. Hierarchical multiscale RNN (Chung, Ahn, and Bengio 2017) introduces a binary boundary state similar to a gate to dynamically determine the boundary structures. FS-RNN (Mujika, Meier, and Steger 2017) uses a similar approach but allows the time scale at the lower level to be finer than the native scale of the input sequences. The nnRNN (Kerg et al. 2019) uses a general non-normal matrix by constraining the modulus of all eigenvalues near 1. Compared with these methods, ENRNN uses two interacting states with different recurrent matrices to model long and short-term memory. The ENRNN only uses a non-normal matrix for the short-term memory component and constrains the spectral radius less than 1. With the eigenvalues of the ENRNN recurrent matrix distributed within the unit disk, the corresponding state can learn short-term dependencies at any unspecified time scale with the added simplicity of a basic RNN.

The learning algorithm of ENRNN is motivated by SN-GAN (Miyato et al. 2018). The SN-GAN normalizes the discriminator weight matrix by its spectral norm, i.e. its largest singular value. Here, we normalize the spectral radius of the recurrent matrix. Noting that the spectral radius is bounded by any matrix norm including the spectral norm, normalization by the spectral norm is expected to make the spectral radius of the matrix much less than 1. We emphasize the importance in our approach to constrain the eigenvalues of the recurrent matrix rather than its singular values because the eigenvalues affect the dynamical behavior of RNN but the singular values do not. See also (Bengio, Simard, and Frasconi 1994). For example, all orthogonal matrices have singular values equal to 1, but may define very different RNNs.

Additional work that supports the idea of modeling short-term dependencies by constraining the eigenvalues is (Kerg et al. 2019; Orhan and Pitkow 2019). Their analyses use a Fisher memory matrix to show non-normal networks may carry more memory than normal networks due to transient amplification. Even though input information is eventually diminished with spectral radius < 1, in the short-term it may increase. This theory shows that non-normal matrices may emulate short-term memory. This can be explained by the pseudo spectrum theory where the spectrum of a matrix is within the unit disk but the pseudo spectrum may extend outside it. In this case, the dynamics exhibit transient amplification (short-term increase) as determined by the pseudo spectrum but asymptotic (long-term) decay as determined by the spectrum.

3 Eigenvalue Normalized Recurrent Neural Network

Although an orthogonal/unitary recurrent weight matrix can help mitigate the vanishing/exploding gradient problem and hence allow an input to affect an output over long sequences, it does not have any mechanism to discard information that is no longer needed. In sequences where certain input information is only used for the states or outputs locally, the state may be consumed with such information, reducing its capacity for carrying other information.

In order to improve the capacity of orthogonal/unitary RNNs to capture short-term dependencies, we introduce a dissipative state. Let $h_t \in \mathbb{R}^n$ be the hidden state consisting of two components: $h_t^{(L)} \in \mathbb{R}^q$ that captures long-term dependencies and $h_t^{(S)} \in \mathbb{R}^{n-q}$ that captures short-term dependencies. In this scheme, q is considered a hyperparameter. Now let $W^{(L)} \in \mathbb{R}^{q\times q}$ be an orthogonal matrix used as the recurrent matrix for $h_t^{(L)}$ that is designed to propagate information over many time steps, and $W^{(S)} \in \mathbb{R}^{(n-q)\times(n-q)}$ which has a spectral radius less than one by normalizing with the spectral radius, see Section 3.1 for details. If we consider a recurrent weight matrix $W \in \mathbb{R}^{n\times n}$ of the form $W = \text{diag}\left(W^{(L)}, W^{(S)}\right)$, then a forward pass of the RNN will be:

$$\begin{cases} h_t^{(L)} = \sigma\left(U^{(L)}x_t + W^{(L)}h_{t-1}^{(L)} + b^{(L)}\right) \\ h_t^{(S)} = \sigma\left(U^{(S)}x_t + W^{(S)}h_{t-1}^{(S)} + b^{(S)}\right) \\ y_t = V^{(L)}h_t^{(L)} + V^{(S)}h_t^{(S)} + c \end{cases} \tag{1}$$

Since $W^{(S)}$ has a spectral radius less than 1, the effect of any input on $h^{(S)}$ will decay quickly from repeat multiplication by $W^{(S)}$ with the rate of decay controlled by the magnitude of the eigenvalues of $W^{(S)}$. Different eigenvalues with different magnitudes will then decay at different rates, emulating different lengths of memory.

In this model, the output y_t is determined from a combination of $h_t^{(L)}$ and $h_t^{(S)}$ where $h_t^{(S)}$ contains information of recent input data, see equation (1). In this way, short-term memory that is needed to determine y_t is stored in $h_t^{(S)}$, but once y_t is computed, $h_t^{(S)}$ will be gradually replaced by information from new inputs. This allows $h_t^{(L)}$ to store and carry only long-term memory information needed for the output.

In this architecture (1), the hidden states $h^{(L)}$ and $h^{(S)}$ are separate. They carry the long and short-term memory in parallel and the short-term state is directly used to determine output. If the task is to determine a single output from a sequence at the end of the entire sequence, then $h^{(S)}$ does not affect the output until near the end of the sequence. In this case, it may still be beneficial to have $h^{(S)}$ accumulate short-term memory but to feed it into $h^{(L)}$ to indirectly affect the final output. This can be done by adding a coupling block to the recurrent matrix,

$$W = \left[\begin{array}{c|c} W^{(L)} & W^{(C)} \\ \hline & W^{(S)} \end{array}\right] \tag{2}$$

where $W^{(C)} \in \mathbb{R}^{q\times(n-q)}$ is called a coupling matrix. Applying the recurrent matrix in (2) to a forward pass of the RNN, we obtain:

$$\begin{cases} h_t^{(L)} &= \sigma\left(U^{(L)}x_t + W^{(L)}h_{t-1}^{(L)} + W^{(C)}h_{t-1}^{(S)} + b^{(L)}\right) \\ h_t^{(S)} &= \sigma\left(U^{(S)}x_t + W^{(S)}h_{t-1}^{(S)} + b^{(S)}\right) \\ y_t &= V^{(L)}h_t^{(L)} + V^{(S)}h_t^{(S)} + c \end{cases} \tag{3}$$

In this case, $h^{(S)}$ is generated by the same recurrence as before and stores short-term information of the inputs. However, with the coupling block, $h^{(L)}$ is determined from the current input, the short-term hidden state $h^{(S)}$ and $h^{(L)}$. This interaction is similar to the update of the internal state of an LSTM. In particular, $h^{(S)}$ can be regarded as a preprocessing of several consecutive inputs designed to extract information to be used to update the long-term memory state $h^{(L)}$. As an example, one can think of character inputs in a language processing problem. The short-term memory state may process the character inputs to produce word or phrase information to be used in the long-term state $h^{(L)}$ so that $h^{(L)}$ can be devoted to processing the information at a higher level. We believe this separation of the processing of characters from the processing at a higher level of sentences or concepts will be more effective and efficient.

We note that since W is an upper triangular matrix, the eigenvalues of W consist of the eigenvalues of both $W^{(L)}$ and $W^{(S)}$ and so has a spectral radius of at most one and this coupling does not alter the spectral properties of the recurrent matrix. For this reason, we do not allow a coupling from $h^{(L)}$ to $h^{(S)}$ because the fully dense recurrent matrix would not preserve the desired spectral properties.

To illustrate how $h^{(S)}$ can simulate a short-term memory state, we note that since $\rho\left(W^{(S)}\right) < 1$, there exists some norm $\|\cdot\|$ such that $\|W^{(S)}\| < 1$. If we assume that this holds for the 2-norm, i.e. $\|W^{(S)}\|_2 < 1$, we formulate the following theorem.

Theorem 3.1 *For an RNN as defined in Equations 2 and 3 with a ReLU nonlinearity, if $\|W^{(S)}\|_2 < 1$ then*

$$\left\|\frac{\partial h_{t+\tau}^{(S)}}{\partial h_t^{(S)}}\right\| \leq \left\|W^{(S)}\right\|^{\tau} \textit{ and } \left\|\frac{\partial h_{t+\tau}^{(S)}}{\partial x_t}\right\| \leq \left\|W^{(S)}\right\|_2^{\tau}\left\|U^{(S)}\right\|$$

where $\|\cdot\|$ is the 2-norm.

We remark that as τ increases, the derivative bounds in Theorem 3.1 go to zero, indicating the dependence of $h_{t+\tau}^{(S)}$ on $h_t^{(S)}$ and x_t goes to zero.

3.1 ENRNN Gradient Descent

The training of $W^{(S)}$ by gradient descent can easily lead to a matrix with spectral radius greater than 1. To maintain $W^{(S)}$ with spectral radius less than 1, we parameterize it by another matrix $T \in \mathbb{R}^{(n-q)\times(n-q)}$ through the normalization

$$W^{(S)} = W^{(S)}(T) := \frac{T}{\rho(T) + \epsilon}$$

for some small $\epsilon > 0$, where $\rho(T) \in \mathbb{R}$ is the spectral radius of T. In this way, $W^{(S)}$ has eigenvalues with modulus less than one and the training of $W^{(S)}$ is carried out in T. Namely, for an RNN loss function $L = L(W^{(S)})$ in terms of $W^{(S)}$, we regard it as a function $L = L(W^{(S)}(T))$ of T. Instead of optimizing with respect to $W^{(S)}$, we optimize $L = L(W^{(S)}(T))$ with respect to T. The gradients of such a parameterization are given below.

Proposition 3.2 *Let $L = L(W) : \mathbb{R}^{m\times m} \to \mathbb{R}$ be some differentiable loss function for an RNN with a recurrent weight* *matrix W and let $\frac{\partial L}{\partial W} := \left[\frac{\partial L}{\partial W_{i,j}}\right] \in \mathbb{R}^{m\times m}$. Let W be parameterized by another matrix $T \in \mathbb{R}^{m\times m}$ as $W = \frac{T}{\rho(T)+\epsilon}$, where $\rho(T) \in \mathbb{R}$ is the spectral radius of T and $\epsilon > 0$ is a small positive number. If $\lambda = \alpha + i\beta$ (with $\alpha, \beta \in \mathbb{R}$) is a simple eigenvalue of T with $|\lambda| = \rho(T)$ and if $u \in \mathbb{C}^n$ and $v \in \mathbb{C}^n$ are corresponding right and left eigenvectors, i.e. $Tu = \lambda u$ and $v^*T = \lambda v^*$, then the gradient of $L = L(T)$ as a function of T is given by:*

$$\frac{\partial L}{\partial T} = \frac{1}{\tilde{\rho}(T)}\left[\frac{\partial L}{\partial W} - \frac{1}{\tilde{\rho}(T)} 1_m^T\left(\frac{\partial L}{\partial W}\odot W\right)1_m C\right]$$

where $C = \alpha\,\mathrm{Re}(S) + \beta\,\mathrm{Im}(S)$ with $S = \frac{\bar{v}u^T}{v^*u} \in \mathbb{C}^{m\times m}$, $1_m \in \mathbb{R}^m$ is a vector consisting of all ones, $\tilde{\rho}(T) = \rho(T) + \epsilon$, $*$ is the conjugate transpose operator, and $\odot$ is the Hadamard product.

Note that even though complex eigenvalues come in conjugate pairs, selecting either λ or $\overline{\lambda}$ in Proposition 3.2 will result in an identical derivative due to conjugation of u and v. In addition, when λ is a multiple eigenvalue, the computation of S involves a division by 0 or a number nearly 0. This is a rare situation and can be remedied in practice. First, it is unlikely to occur as the set of matrices with multiple eigenvalues lie on a low dimensional manifold in the space of $n \times n$ matrices and has a Lebesgue measure 0. Thus the probability of a random matrix having multiple eigenvalue is zero. Second, if a multiple or nearly multiple eigenvalue occurs, we may train using usual gradient descent without eigenvalue normalization for a few steps and return to $\frac{\partial L}{\partial T}$ when the eigenvalues are separated. This situation never occurred in our experiments.

Using Proposition 3.2, an optimizer with learning rate ζ is used to first update T which is then used to update $W^{(S)}$:

$$T_k \leftarrow T_{k-1} - \zeta\frac{\partial L}{\partial T_{k-1}}; \quad W_k^{(S)} \leftarrow \frac{T_k}{\rho(T_k)+\epsilon} \tag{4}$$

A naive approach may be to simply apply gradient descent on $W^{(S)}$ and then re-normalize $W^{(S)}$ by its spectral radius. The problem is that the computed gradients $\frac{\partial L}{\partial W^{(S)}}$ do not take into account the effects of the normalization. Thus a steepest descent step on $W^{(S)}$ will reduce the loss function, but it may not be the case after $W^{(S)}$ is re-normalized by the spectral radius. In contrast, our approach takes a gradient descent step on T, which decreases the loss function with the new W. Namely, the steepest descent direction $\frac{\partial L}{\partial T}$ has taken the eigenvalue normalization into account.

3.2 Complexity

The short-term memory matrix, $W^{(S)} \in \mathbb{R}^{(n-q)\times(n-q)}$, requires the computation of the spectral radius and the associated right/left eigenvectors as outlined in Section 3.1. This is done by using the Schur decomposition of the parameter matrix $T \in \mathbb{R}^{(n-q)\times(n-q)}$ which requires a complexity of $\mathcal{O}((n-q)^3)$ (Demmel 1997) per mini-batch training iteration. This is comparable in complexity to models that require $\mathcal{O}(n)^3$ complexity to maintain an orthogonal/unitary recurrent matrix such as scoRNN. Note that implementation of a standard RNN requires a complexity of $\mathcal{O}\left(BLn^2\right)$ where B is the batch size and L is the sequence length. This additional complexity of using the Schur decomposition will be comparable to a standard RNN when $n - q \leq BL$ which is typically the case. Alternatively, the complexity can be reduced to $\mathcal{O}((n-q)^2)$ by using the power-method instead of Schur decomposition, as discussed in SN-GAN (Miyato et al. 2018). In practice we found that the power method may require an uneven number of iterations and may actually be less efficient than Schur deomposition.

4 Other Architecture Details

We initialize T to be a random matrix with eigenvalues uniformly distributed on the complex unit disc. This is done in a way similar to (Helfrich, Willmott, and Ye 2018) as

$$T = \mathrm{diag}\left(B_1, .., B_{\lfloor n/2\rfloor}\right) B_j = \gamma_j\begin{bmatrix}\cos t_j & -\sin t_j\\ \sin t_j & \cos t_j\end{bmatrix} \tag{5}$$

where each t_j is sampled from $\mathcal{U}[0, \frac{\pi}{2})$ and each γ_j is sampled from $\mathcal{U}[-1.0, 1.0)$. This results in eigenvalues of the form $\gamma_j e^{\pm it_j}$ which are uniformly distributed on the complex unit disc. For the coupling matrix, $W^{(C)}$, initialization is Glorot Uniform (Glorot and Bengio 2010) unless indicated otherwise. The initial states of $h_0^{(L)}$ and $h_0^{(S)}$ are set to zero and are non-trainable.

It is unknown before hand if the largest eigenvalue should have a modulus near one, so we start by setting $W^{(S)} = T$ without eigenvalue normalization and train until $\rho(T) > 1$, at which point eigenvalue normalization is implemented. Namely, if $\rho(T) \leq 1$, then a standard gradient descent step is taken with $W^{(S)} = T$. Once an update step results in a $\rho(T) > 1$, equation (4 is used for all subsequent training steps.

Since the ENRNN is designed to expand upon orthogonal/unitary RNNs and many of these architectures use the modReLU as defined: $\sigma_{\mathrm{modReLU}}(z) = \frac{z}{|z|}\sigma_{\mathrm{ReLU}}(|z| + b)$. We also use it on most of our experiments.

5 Experiments

In this section, we present four experiments to compare ENRNN with LSTM and several orthogonal/unitary RNNs. Code for the experiments and hyperparameter settings for ENRNN are available at `https://github.com/KHelfrich1/ENRNN`. We compare models using single layer networks because implementation of multi-layer networks in the literature typically involves dropout, learning rate decay, and other multi-layer hyperparameters that make comparisons difficult. This is also the setting used in prior work on orthogonal/unitary RNNs. Each hidden state dimension is adjusted to match the number of trainable parameters, but ENRNN can be stacked in multiple layers. For ENRNN, the long-term recurrent matrix $W^{(L)}$ is parameterized using scoRNN (Helfrich, Willmott, and Ye 2018). For the short-term component state, we use $\epsilon = 0$ in Proposition 3.2. Unless noted otherwise, the activation function used was modReLU. For each method, the hyperparameters tuned included the optimizer {Adam, RMSProp, Adagrad}, and learning rates $\{10^{-3}, 10^{-4}, 10^{-5}\}$. For scoRNN, the number of negative ones used in the parameterization of the recurrent matrix is tuned in multiples of 10% of the hidden size. For ENRNN, the size of the short-term state $W^{(S)}$ is tuned in multiples of 10% of the entire hidden size up to 60%. For the LSTM, the forget gate bias initialization and gradient clipping threshold are tuned using integers in $[-4, 4]$ and in $[1, 10]$ respectively. These hyperparameters were selected using a gridsearch method. Experiments were run using Python3, Tensorflow, and CUDA9.0 on GPUs.

5.1 Adding Problem

The adding problem (Hochreiter and Schmidhuber 1997) has been widely used in testing RNNs. For this experiment, we implement a variation of the adding problem (Arjovsky, Shah, and Bengio 2016; Mhammedi et al. 2017; Helfrich, Willmott, and Ye 2018; Maduranga, Helfrich, and Ye 2019). The problem involves passing two sequences of length T concurrently into the RNN. The first sequence consists of entries sampled from $\mathcal{U}[0, 1)$ and the second sequence consists of all zeros except for two entries that are marked by the digit one. The first one is located uniformly in the first half of the sequence, $[1, \frac{T}{2})$ and the second one is located uniformly in the second half of the sequence, $[\frac{T}{2}, T)$. The network outputs the sum of the two entries in the first sequence that are marked by ones in the second sequence. The loss function used is the mean squared error (MSE). The baseline is an MSE of 0.167 which is the expected MSE for a network that predicts one regardless of the sequence. The sequence length used in this experiment was $T = 750$ with training and test sets of size $100,000$ and $10,000$ examples.

The hidden sizes for each model were adjusted so they each had $\approx$ 15k trainable parameters which results in a total hidden size of $n = 160, 60, 60, 170, 128,$ and 120 for the ENRNN, LSTM, Spectral RNN, scoRNN, oRNN, and Full-Capacity uRNN respectively. The ENRNN was comprised of an $h^{(L)}$ and an $h^{(S)}$ of respective sizes 96 and 64 with a coupling matrix, see Equation (3). The best hyperparameters for the oRNN were in accordance with (Mhammedi et al. 2017) with $\approx$ 2.6k trainable parameters. Figure 1 presents the convergence plots for 6 epochs. ENRNN converges towards 0 MSE before all other models with Spectral RNN asympototically achieving a slightly lower MSE with learning rate decay.

5.2 Copying Problem

The copying problem has also been used to test many orthogonal/unitary RNNs (Arjovsky, Shah, and Bengio 2016; Wisdom et al. 2016; Jing et al. 2016; Mhammedi et al. 2017; Helfrich, Willmott, and Ye 2018). In this experiment, a sequence of digits is passed into the RNN with the first 10 digits uniformly sampled from the digits 1 through 8 followed by the marker digit 9, a sequence of T zeros, and another marker digit 9. The network is to output the first 10 digits in the sequence once it sees the second marker 9, forcing the network to remember the original digits over the entire sequence. The total sequence length is $T + 20$. The cross-entropy loss function is used. The training and test sets were

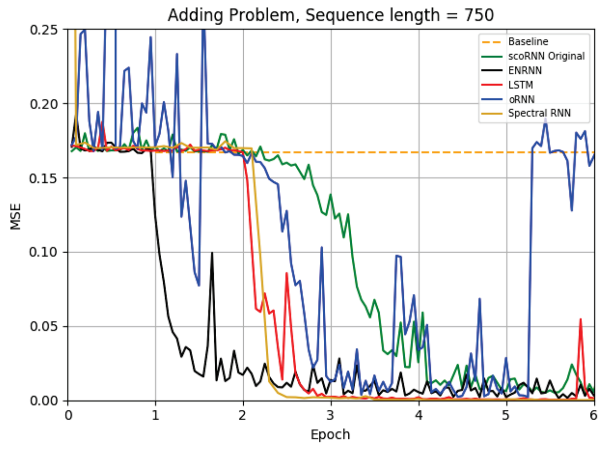

Figure 1: Test set MSE for the adding problem with sequence length of $T = 750$.

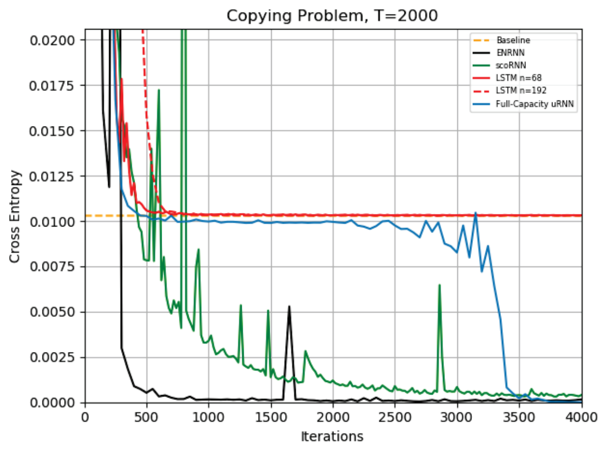

Figure 2: Cross-entropy for the copying problem with sequence length of $T = 2000$.

$20,000$ and $1,000$ sequences, respectively. Each model was trained for $4,000$ iterations with batch size 20. The baseline for this task is the expected cross-entropy of randomly selecting digits 1-8 after the last marker 9, $\frac{10\log(8)}{T+20}$.

The hidden sizes for each model were adjusted so that they each had $\approx 22k$ trainable parameters. This resulted in a hidden size of $n = 192, 68, 190,$ and 128 for the ENRNN, LSTM, scoRNN, and Full-Capacity uRNN respectively. The ENRNN had an $h^{(L)}$ and $h^{(S)}$ of size 172 and 20 with a coupling matrix $W^{(C)}$, see Equation (3). Figure 2 plots cross-entropy values for 4000 iterations. As a reference, the LSTM was also run with the same hidden size of ENRNN, $n = 192$, which has $\approx$ 7 times more trainable parameters than ENRNN and is still unable to drop below the baseline. Again, ENRNN outperforms other methods.

Table 1: TIMIT: Best validation MSE after 300 epochs with test MSE and perceptual metrics. N - dimension of h. (for ENRNN, dimensions of $h^{(L)}/h^{(S)}$)

MODEL	N	#PARAMS	VALID. MSE	TEST. MSE
ENRNN	374/94	$\approx$ 200K	**0.13**	**0.13**
SCORNN	425	$\approx$ 200K	1.56	1.52
LSTM	158	$\approx$ 200K	8.53	8.27
LSTM	468	$\approx$ 1200K	5.60	5.42

MODEL	N	SEGSNR (DB)	STOI	PESQ
ENRNN	374/94	**4.84**	**0.83**	**2.75**
SCORNN	425	4.55	0.82	2.72
LSTM	158	4.00	0.79	2.51
LSTM	468	4.82	0.81	2.75

5.3 TIMIT

The TIMIT dataset (Garofolo et al. 1993) consists of spoken sentences from 630 different speakers with eight major dialects of American English. We used the same setup as outlined in (Wisdom et al. 2016). The data set consisted of 3,696 training, 192 testing, and a validation set of 400 audio files. Each audio file was downsampled to 8kHz and a short-time Fourier transform (STFT) was applied (Wisdom et al. 2016). The sequence of the log magnitude of the STFT values are fed into RNNs and the output is the same sequence shifted forward in time by 1 to predict the next log magnitude STFT value in the sequence. Each sequence was padded with zeros to make uniform lengths. The loss function was the mean squared error (MSE) and was computed by taking the squared difference between the predicted and actual log magnitudes and applying a mask to zero out padded entries before computing the batch mean. The batch size was 28.

Following (Wisdom et al. 2016), the models were also analyzed using time-domain metrics consisting of the Signal-to-Noise Ratio (SegSNR), Perceptual Evaluation of Speech Quality (PESQ), and Short-Time Objective Intelligibility (STOI). For SegSNR, the higher the positive value indicates more signal than noise. The PESQ values range within $[1.0, 4.5]$ with a higher value indicating better signal quality. The STOI values range within $[0, 1]$ with a higher value indicating better human intelligibility. To compute the scores, the predicted log-magnitudes on the test set were used to reconstruct the sound waves and were compared with the original sound waves, see (Wisdom et al. 2016).

Each model was trained for 300 epochs. The ENRNN did not have a coupling matrix. Table 1 reports the results of the best epoch in validation MSE scores and Figure 3 plots convergence of these scores. As a secondary measure, we also show in Table 1 scores of three perceptual metrics. As a reference, the LSTM was also run with the same hidden size of ENRNN, $n = 468$, which has ≈ 6 times more trainable parameters than ENRNN and achieves worse scores except for PESQ where it is the same. Again, ENRNN significantly outperforms scoRNN and LSTM in the validation and testing MSEs and produces the overall best scores in the perceptual metrics.

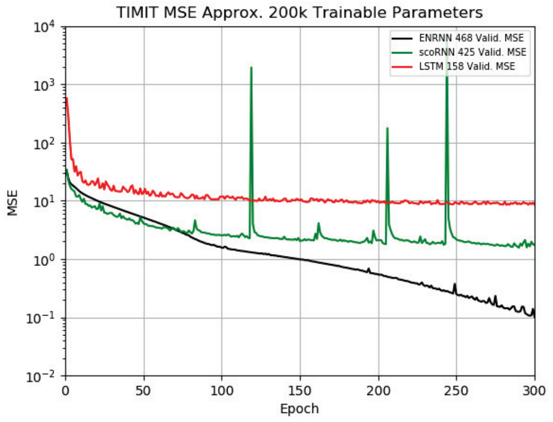

Figure 3: Validation set MSE for the TIMIT problem

5.4 Character PTB

The models were also tested on a character prediction task using the Penn Treebank Corpus (Marcus, Marcinkiewicz, and Santorini 1993). The dataset consists of a word vocabulary of 10k with all other words marked as <unk>, resulting in a total of 50 characters with the training, validation, and test sets consisting of approximately 5102k, 400k, and 450k respective characters. The batch size was set to 32. Due to the length of each sequence, the sequences were unrolled in length of 50 steps. Each sequence is fed into RNNs and the output is the same sequence shifted forward by one step to predict the next character. At the end of training of each sequence in a batch, the final hidden state is passed onto the next training sequence as the initial state. We use a linear embedding layer that maps each input character to R^N (N being the hidden state dimension). The loss function was cross-entropy. We report the customary performance metrics of bits-per-character (bpc) which is the cross-entropy loss with the natural logarithm replaced by the base 2 logarithm.

ENRNN uses a coupling matrix $W^{(C)}$ with a truncated orthogonal initialization, a fixed input weight matrix set to identity, and ReLU nonlinearity. We report the best results after 20 epochs training in Table 2. As a reference, the LSTM was also run with the same hidden size of ENRNN, $n = 1030$, which results in a better score but requires $\approx$ 8.5 times more trainable parameters than ENRNN. We see that ENRNN slightly outperforms LSTM when matching the number of trainable parameters and all other models.

6 Exploratory Experiments

We present exploratory studies to demonstrate short-term and long-term dependency of the states $h^{(S)}$ and $h^{(L)}$ respectively as intended.

6.1 Gradients

For each pair of time steps $t \leq \tau$, we use $||\frac{\partial h^{(S)}_\tau}{\partial x_t}||_2$ and $||\frac{\partial h^{(L)}_\tau}{\partial x_t}||_2$ to measure the dependency of $h^{(S)}_\tau$ and $h^{(L)}_\tau$ at

Table 2: Character PTB: Best testing MSE in BPC after 20 epochs. N - dimension of h (for ENRNN, dimensions of $h^{(L)}/h^{(S)}$). Entries marked by an asterix (*), (**), and (***) are reported from (Jing et al. 2017), (Mhammedi et al. 2017), and (Kerg et al. 2019), resp.

Model	N	# Param	Valid. BPC	Test BPC
ENRNN	310/720	$\approx$ 1016K	**1.475**	**1.429**
LSTM	350	$\approx$ 1016K	1.506	1.461
GRU	415	-	-	1.601*
EURNN	2048	-	-	1.715*
GORU	512	-	-	1.623*
oRNN	512	$\approx$ 183K	1.73**	1.68**
nnRNN	1024	$\approx$ 1320K	-	1.47***
LSTM	1030	$\approx$ 8600K	1.447	1.408

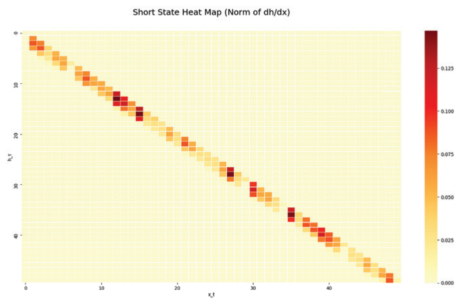

Figure 4: Gradient norms $\|\frac{\partial h_\tau^{(S)}}{\partial x_t}\|$ The x-axis is t from left to right and y-axis is τ from top to bottom. The column at t shows dependence of states $h_\tau^{(S)}/h_\tau^{(L)}$ on x_t.

time τ respectively on the input x_t at time t. We consider a small Adding Problem (Sec. 5.1) of sequence length $T = 50$ using an ENRNN of hidden size $n = 40$ with $h^{(L)}$ block size of 24 and $h^{(S)}$ block size of 16. We compute the gradient norms over the first random mini-batch at the beginning of the sixth epoch and plot them as a heat map in Figure 4 and 5. Here the x-axis is the input time step (going from left to right) and the y-axis is the hidden state time step (going from top to bottom).

As can be seen, the short-term state gradient (left) diminishes quickly as τ increases from t, demonstrating the short-term dependency of $h_\tau^{(S)}$. On the other hand, the long-term state gradient (right) may stay large for all τ showing long-term dependency of $h_\tau^{(S)}$. Of particular note, there appears to be a few vertical lines that have higher gradient norms relative to other input steps. It appears that these inputs have a greater effect on the model than others.

6.2 Hidden State Sizes

In this section, we explore the effect of different short-term hidden states on model performance on the adding problem using similar settings as discussed in Section 5.1. In Figure 6, we keep the total hidden state size fixed at $n = 160$

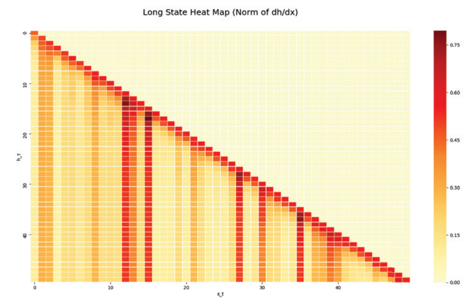

Figure 5: Gradient norms $\|\frac{\partial h_\tau^{(L)}}{\partial x_t}\|$ The x-axis is t from left to right and y-axis is τ from top to bottom. The column at t shows dependence of states $h_\tau^{(S)}/h_\tau^{(L)}$ on x_t.

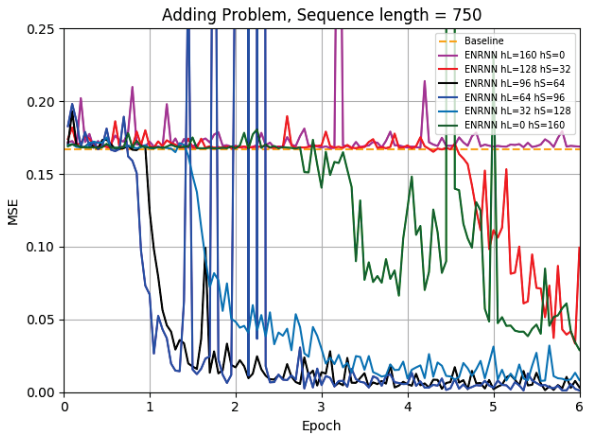

Figure 6: Test set MSE for the ENRNN on the adding problem with sequence length of $T = 750$ with fixed hidden state size of 160 and various short-term and long-term hidden state sizes $h^{(S)}$ and $h^{(L)}$.

and adjust the $h^{(L)}$ and $h^{(S)}$ sizes. In addition, we run the experiment with no $h^{(L)}$ and no $h^{(S)}$. As can be seen, having no long-term memory state, $h^{(L)} = 0$, the ENRNN is unable to approach zero MSE until the end and having no short-term memory state, $h^{(S)} = 0$, the ENRNN is unable to pass the baseline after 6 epochs. In general, as the size of $h^{(S)}$ increases, the performance increases with optimal performance occurring around $h^{(S)} = 64$ and $h^{(L)} = 96$ or $h^{(S)} = 96$ and $h^{(L)} = 64$.

7 Conclusion

We have introduced a new RNN architecture with two components to accumulate long and short-term memory information. We have developed a gradient descent algorithm for learning a short-term recurrent matrix with eigenvalues on the unit disc. Our exploratory study indicates that the long-term and short-term component states behave as intended. Experimental results in four widely used examples indicate that the ENRNN is competitive with orthogonal/unitary

RNNs and LSTM. Also important is that our method is entirely based on the basic RNN framework without the need of highly complex architectures that are more difficult to understand and implement.

Acknowledgments. This research was supported in part by NSF under grants DMS-1821144 and DMS-1620082.

References

Arjovsky, M.; Shah, A.; and Bengio, Y. 2016. Unitary evolution recurrent neural networks. In *Proceedings of ICML 2016*.

Bengio, Y.; Frasconi, P.; and Simard, P. 1993. The problem of learning long-term dependencies in recurrent networks. In *Proceedings of ICNN 1993*.

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

Cho, K.; van Merrienboer, B.; Bahdanau, D.; and Bengio, Y. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv e-prints* arXiv:1409.1259.

Chung, J.; Ahn, S.; and Bengio, Y. 2017. Hierarchical multiscale recurrent neural networks. In *Proceedings of ICLR 2017*.

Demmel, J. 1997. *Applied Numerical Linear Algebra*. 3600 Market Street, Philadelphia PA: Society for Industrial and Applied Mathematics.

Garofolo, J.; Lamel, L.; Fisher, W.; Fiscus, J.; Pallett, D.; Dahlgren, N.; and Zue, V. 1993. Timit acoustic-phonetic continuous speech corpus ldc93s1. Technical report, Philadelphia: Linguistic Data Consortium, Philadelphia, PA.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of AISTATS 2010*.

Helfrich, K.; Willmott, D.; and Ye, Q. 2018. Orthogonal recurrent neural networks with scaled cayley transform. In *Proceedings of ICML 2018*.

Henaff, M.; Szlam, A.; and LeCun, Y. 2016. Recurrent orthogonal networks and long-memory tasks. In *Proceedings of ICML 2016*.

Hihi, S. E., and Bengio, Y. 1995. Hierarchical recurrent neural networks for long-term dependencies. In *Proceedings of NIPS 1995*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9:1735–80.

Jing, L.; Shen, Y.; Dubček, T.; Peurifoy, J.; Skirlo, S.; Tegmark, M.; and Soljačić, M. 2016. Tunable efficient unitary neural networks (eunn) and their application to rnn. *arXiv e-prints* arXiv:1612.05231.

Jing, L.; Gulcehre, C.; Peurifoy, J.; Shen, Y.; Tegmark, M.; Soljačić, M.; and Bengio, Y. 2017. Gated orthogonal recurrent units: On learning to forget. *arXiv e-prints* arXiv:1706.02761.

Kerg, G.; Goyette, K.; Touzel, M.; Gidel, G.; Voronstov, E.; Bengio, Y.; and Lajoie, G. 2019. Non-normal recurrent neural network (nnrnn): Learning long time dependencies while improving expressivity with transient dynamics. In *Proceedings of NeurIPS2019*.

Koutník, J.; Greff, K.; Gomez, F.; and Schmidhuber, J. 2014. A clockwork rnn. In *Proceedings of ICML 2014*.

Le, Q. V.; Jaitly, N.; and Hinton, G. E. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv e-prints* arXiv:1504.00941.

Lezcano-Casado, M., and Martinez-Rubio, D. 2019. Cheap orthogonal constraints in neural networks: A simple parameterization of the orthogonal and unitary group. In *Proceedings of ICML 2019*.

Lezcano-Casado, M. 2019. Trivializations for gradient-based optimization on manifolds. In *Proceedings of NeurIPS 2019*.

Maduranga, K.; Helfrich, K.; and Ye, Q. 2019. Complex unitary recurrent neural networks using scaled cayley transform. In *Proceedings of AAAI 2019*.

Marcus, M.; Marcinkiewicz, M.; and Santorini, B. 1993. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics* 19(2):313–330.

Mhammedi, Z.; Hellicar, A.; Rhaman, A.; and Bailey, J. 2017. Efficient orthogonal parameterisation of recurrent neural networks using householder reflections. In *Proceedings of ICML 2017*.

Miyato, T.; Kataoka, T.; Koyama, M.; and Yoshida, Y. 2018. Spectral normalization for generative adverserial networks. In *Proceedings of ICLR 2018*.

Mujika, A.; Meier, F.; and Steger, A. 2017. Fast-slow recurrent neural networks. In *Proceedings of NIPS 2017*.

Orhan, E., and Pitkow, X. 2019. Improved memory in recurrent neural networks with sequential non-normal dynamics. *arXiv e-prints* arXiv:1905.13715v1.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of ICML 2013*.

Schmidhuber, J. 1992. Learning complex, extended sequences using the principle of history compression. *Neural Computation* 4(2):234–242.

Vorontsov, E.; Trabelsi, C.; Kadoury, S.; and Pal, C. 2017. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of ICML 2017*.

Wisdom, S.; Powers, T.; Hershey, J.; Roux, J. L.; and Atlas, L. 2016. Full-capacity unitary recurrent neural networks. In *Proceedings of NIPS 2016*.

Wolter, M., and Yao, A. 2018. Complex gated recurrent neural networks. In *Proceedings of NIPS 2018*.

Zhang, J.; Lei, Q.; and Dhillon, I. 2018. Stabilizing gradients for deep neural networks via efficient SVD parameterization. In *Proceedings of ICML 2018*.