Non-normal Recurrent Neural Networks

(Part of NeurIPS Reproducibility Challenge 2019)

Abhilash Jahagirdar
Department of Computer Science
University of Colorado Boulder
Boulder, CO 80309
abhilash.jahagirdar@colorado.edu

Karthik Siddarmanna
Department of Computer Science
University of Colorado Boulder
Boulder, CO 80309
karthik.siddaramanna@colorado.edu

Madhusudhan Aithal Mahabhaleshwara
Department of Computer Science
University of Colorado Boulder
Boulder, CO 80309
maai8912@colorado.edu

Siddhartha Shankar
Department of Computer Science
University of Colorado Boulder
Boulder, CO 80309
siddartha.shankar@colorado.edu

Abstract

Recurrent neural networks (RNNs) have been proven to suffer from Exploding and Vanishing Gradient Problem (EVGP). In the recent years, a class of RNNs known as Orthogonal RNNs [1] (or Unitary RNNs for complex values) have been developed which addresses the issue of exploding and vanishing gradient problem in RNNs. Orthogonal RNNs restrict the recurrent connectivity matrices to have unit norm eigen and singular spectra, thus ensuring that the signals (or gradients) can be propagated over long timescales. The expressivity of orthogonal RNNs is limited because it spans only a subset of all transformations with unit norm eigen spectra. Non-normal Recurrent Neural Networks (nnRNN) [2] solves this problem by relaxing the constraint on the orthogonal eigen basis. This allows to parameterize matrices with unit-norm eigenspectra without orthogonality constraints on eigenbasis. Thus nnRNN provides better expressivity with similar stable dynamics and training as Orthogonal RNNs.

1 Introduction

It is very hard to train Recurrent Neural Networks (RNNs) to perform computations over long time scales [3] because of the exploding and vanishing gradient problem (EVGP).

$$\begin{aligned} z_{t+1} &= V h_t + U x_{t+1} + b \\ h_{t+1} &= \phi(z_{t+1}) \end{aligned} \tag{1}$$

- $h_t \in \mathbb{R}^n$ is the hidden state vector at time t.
- x_t is input at time t.
- U matrix that projects input into dynamics.
- V is the recurrent connectivity matrix.

Let L be the loss we are trying to minimize. The rate of change of loss with respect to hidden state $\frac{\partial L}{\partial h_t}$ is responsible for the vanishing and exploding gradient problem as the value of t grows.

GitHub link of the source code *here*

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_T}\frac{\partial h_T}{\partial h_t} = \frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1}\frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1} D_{k+1}V_k^T \tag{2}$$

Where $D_{k+1} = diag(\phi^{'}(z_{k+1}))$ is the jacobian of the activation function. Let's define the norm of the matrix to be the largest absolute eigen value corresponding to the matrix and the norm of a vector is it's L_2 norm. The matrix D_k is diagonal hence

$$\|D_k\| = \max_{i=1\ldots n} |diag(\phi^{'}(z_{k+1}^i))|$$

$$\begin{aligned}\left\|\frac{\partial L}{\partial h_t}\right\| &= \left\|\frac{\partial L}{\partial h_T}\prod_{k=t}^{T-1} D_{k+1}V_k^T\right\| \\ &\leq \left\|\frac{\partial L}{\partial h_T}\right\|\prod_{k=t}^{T-1}\left\|D_{k+1}V_k^T\right\| \end{aligned} \tag{3}$$

Now we observe that norm of the back-propagated loss scales depending on the eigen spectra of V and jacobian of the activation function. If we constrain the matrix V to be **unitary** then all the eigen values will be 1. We can rewrite the above equation as follows.

$$\left\|\frac{\partial L}{\partial h_t}\right\| \leq \left\|\frac{\partial L}{\partial h_T}\right\|\prod_{k=t}^{T-1}\|D_{k+1}\| \tag{4}$$

- if $\phi^{'}$ takes some value $v > 1$, we see that $\left\|\frac{\partial L}{\partial h_t}\right\| \leq \left\|\frac{\partial L}{\partial h_T}\right\| v^{T-t}$. The gradient will grow exponentially with time.
- if $\phi^{'}$ takes some value $v < 1$, we observe that the gradient vanishes with time.

The above two properties call for the use of ReLU activation function. If ReLU is used, then $\|D_k\|$ will be one, unless all the activations are simultaneously switched off. We will discuss more about ReLU and how to use it in complex domain later. If we satisfy these two conditions we can write.

$$\left\|\frac{\partial L}{\partial h_t}\right\| \leq \left\|\frac{\partial L}{\partial h_T}\right\| \tag{5}$$

By using activation function like ReLU and constraining the matrix V to be unitary, we can prevent EVGP. This spectrally constrained optimization led to a class of RNN architectures called Orthogonal RNNs. Orthogonal RNNs prevent EVGP and perform well on tasks requiring recall over long time scales. Orthogonal matrices are a subset of all the matrices with eigen spectra one. This leads to limited expressivity in orthogonal RNNs and they struggle on tasks involving computation over long times scales. Non-normal RNNs address this problem by leveraging schur decomposition.

2 Non-normal RNN

Any diagonalizable matrix V can be decomposed into the below form using Eigen Decomposition,

$$V = P\Lambda P^T \tag{6}$$

where P contains orthogonal eigenbases of V and Λ is a diagonal matrix containing the eigen values of V. If a matrix does not have orthogonal eigenbases, we can express it in the above form by adding a strictly lower triangular part to Λ in eqn. (6) using Schur decomposition. This is the main idea behind the Non-normal RNN. For any $n \times n$ matrix V, we can use Schur Decomposition to decompose V as follows,

$$V = P\Theta P^T, \;\; \Theta = \Lambda + T \tag{7}$$

$$V = P \left(\begin{bmatrix} \mathbf{R_1} & 0 & \dots & 0 \\ 0 & \mathbf{R_1} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \mathbf{R_{N/2}} \end{bmatrix} + \begin{bmatrix} 0 & 0 & \dots & 0 \\ \alpha & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \beta & \beta & \alpha & 0 \end{bmatrix} \right) P^T$$

with

$$\mathbf{R_i}(\gamma_i, \theta_i) \stackrel{def}{=} \gamma_i \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix} \tag{8}$$

The matrix Θ is parametrized as follows.

$$(\Theta)_{i,j} = d\delta_{i,j} + \alpha\delta_{i,j+1} + \beta \sum_{2 \le k \le i} \delta_{i,j+k}$$

Here Λ is a diagonal matrix consisting of eigen values of V and T is a strictly lower triangular matrix. This essentially means that we can orthonormalize a non orthonormal P without changing the eigenspectrum of the matrix, in order to express a matrix according to the eqn. (7). One useful implication of this is that we can update T in eqn. (7) independently as the constraint of unit norm eigen spectra is satisfied.

Non-normal matrices can have complex eigen spectra. Complex values in computation can be avoided by representing Λ as block diagonal with $\mathbf{R_i}$ (2×2 blocks) these blocks represent complex conjugate eigenvectors. If real eigen values are present then the diagonal will contain 1×1 block.

Activation function ReLU has to be modified to work with complex values. If ReLU is applied separately on the real and imaginary parts then the phase of the complex number changes. The phase of a complex eigen value is speculated to form a key component for storing information across long time scales. So the authors use a variant of ReLU called modReLU[3] as an activation function in nnRNN. This activation affects only the magnitude of the complex value.

$$\sigma_{modReLU}(z) = \begin{cases} \sigma_{ReLU}(|z| + b)\dfrac{z}{|z|} & |z| + b \ge 0 \\ 0 & |z| + b < 0 \end{cases} \tag{9}$$

where $b \in \mathbb{R}$ is a bias parameter of the nonlinearity.

3 Transient Dynamics

The strictly lower triangular matrix T represents the purely feedforward connections of the network. We are able to parametrize any matrix in the form (7) where P is orthogonal by capturing the interactions between the *Schur modes*(columns of P) in T. In our case, α captures the interactions between the consecutive schur modes only and β represents the remaining interactions.

The transient dynamics produced in non normal networks is because of T and [4] does a detailed analysis of such dynamics introduced by a strictly lower trainglular matrix with non zero α and $\beta = 0$. They introduce the Fisher Memory Curve(FMC) , which uses Fisher Information to quantify the memory trace of a network. Although, Fisher information is generally used for studying static stimuli, it can be extended to dynamic time dependent stimuli.

3.1 Fisher Memory Matrix (FMM)

For a given scalar valued input $x_t = s_t + \xi_t$, $t \in \mathbb{N}$, composed of signal s_t and noise $\xi_t : \xi_t \sim \mathcal{N}(\mu_t, \epsilon_t)$ induces a probability distribution over trajectories of hidden states. The change in the likelihood of this probability distribution conditioned over the history of input sequences is captured by the *Fisher Memory Matrix* and this exactly gives us a measure of stored information.

The *Fisher Memory Matrix* is got by,

$$J_{k,l}(s_{:t}) = \left\langle \frac{\partial^2}{\partial s_k \partial s_l} \log P(h_{:t}|x_{:t}) \right\rangle_{P(h_{:t}|s_{:t})} \tag{10}$$

Here, $: t$ subscript is short hand for $(k : k \leq t)$ and $P(h_{:t}|s_{:t})$ is the distribution conditioned over the input sequence $s_{:t}$. $s_{:t}$ is considered to be a parameter of the model as the current hidden states of the model is influenced by $s_{:t}$.

The diagonal elements of the FMM $J(t) := \mathbf{J}_{t,t}$ make up the *Fisher Memory Curve(FMC)* of the network. The term $\mathbf{J}_{k,k}$ gives the information that our network retains about the signal s_{t-k}, that came in k timesteps back.

3.2 Interpreting Fisher Memory Matrix [8]

We can write our main recurrent equation as:

$$h_t = \sum_{k=0}^{\infty} V^k U s_k + \sum_{k=0}^{\infty} V^k \xi_{n-k} \tag{11}$$

by taking ϕ as an identity function and bias b as zero. In eq[2], as ϕ is linear, we can see that FMM does not depend on the signal. From this along with the fact that the noise is normally distributed, we can say that $P(h_{:t}|x_{:t})$ is also a Gaussian with mean linearly dependent on the signal ($\mu(s) = \sum_{k=0}^{\infty} V^k U s_k$). The variance of the distribution is independent of the signal and is exactly equal to variance of the noise ξ. Hence the co-variance matrix $C_t = \epsilon \sum_{k=0}^{\infty} V^k V^{kT}$ is also independent of signal. In such a case, the expression for FMM reduces to a much simpler form:

$$J_{k,l}(s) = \frac{\partial \mu(s)^T}{\partial s_k} C^{-1} \frac{\partial \mu(s)^T}{\partial s_l} = (V^k U)^T C^{-1} V^k U \tag{12}$$

3.3 Effects of α and β on the transients

Consider the multiplication of 3×3 matrix of the form P with schur modes $U = \{u_1, u_2, u_3\}$ and $\Theta = \Lambda + T$ as below:

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ u_1 & u_2 & u_3 \\ \vdots & \vdots & \vdots \end{bmatrix} * \left(\begin{bmatrix} d_1 & 0 & 0 \\ 0 & d_2 & 0 \\ 0 & 0 & d_3 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \alpha & 0 & 0 \\ \beta & \alpha & 0 \end{bmatrix} \right) = \begin{bmatrix} \vdots & \vdots & \vdots \\ m_1 & m_2 & m_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{13}$$

where

$$m_1 = d_1 * u_1 + \alpha * u_2 + \beta * u_3, m_2 = u_2 * d_2 + u_3 * \alpha \text{ and } m_3 = u_3$$

As we can see here clearly, α just controls the interactions between two adjacent *Schur modes* and β affects the interactions between all other modes 1(a).

To analyze the transient dynamics produced by purely feed-forward connections, we plot 1(b) the norm of hidden units and their variances over an ensemble of 1000 uniformly distributed input conditions. Input $x_t > 0$ at $t = 0$ and $x_t = 0$ for all $t > 0$. Λ is zero and T is initialized from the distribution. We see that α controls the exponential amplification of the transients and β affects the shape of the transients. This ability to control both the extent and shape provides expressivity for *nnRNN*. The sudden dip in the trajectories at $t = N$, where N is the dimension of the recurrent connectivity matrix is caused by the nil-potent property of the strictly lower triangular matrices. As

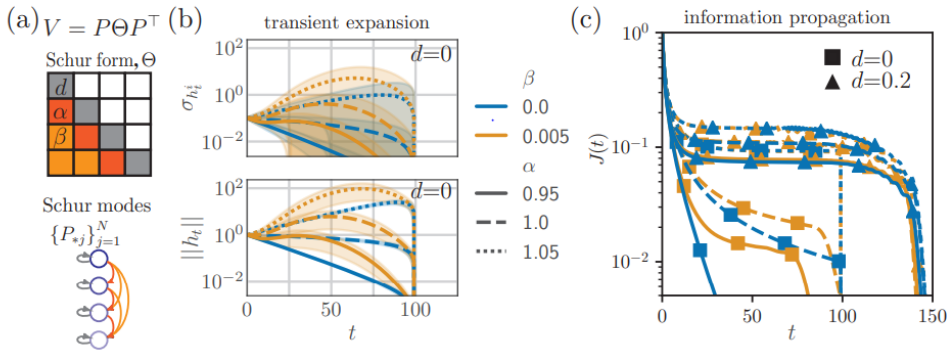


Figure 1

the diagonal elements are zero, in each iteration, the topmost element of every column vanishes.

The FMC trajectories 1(c) also behave in a similar way with varying α and β. Having diagonal elements as $d = 0.2$ leads to better retention and memory traces fail to exist only after $t = 150$. Here, d represents the eigen values of Θ. As we have no input after $t = 0$, according to [5] what we observe here is information decaying at a rate governed by eigen values.

4 Implementation Details

4.1 Copy Task

In this task, the networks are fed an input sequence x_t which has a length of $10 + T_c$. The values for the sequence is obtained from the set $S = \{a_i\}\forall i$, such that $1 \leq i \leq 10$.

Initially, for $t = 1, ..., 10$, x_t can take any one of the distinct values $\{a_i\}_{i=1}^8$. For the next $T_c - 1$ timesteps, the same input is given to the networks which has the value equal to a_9. At $t = T_c$, a_{10} is given as the input which acts as a cue and prompts the model to recall the first 10 inputs. The task is to output the first 10 inputs sequentially in the same order as they were given.

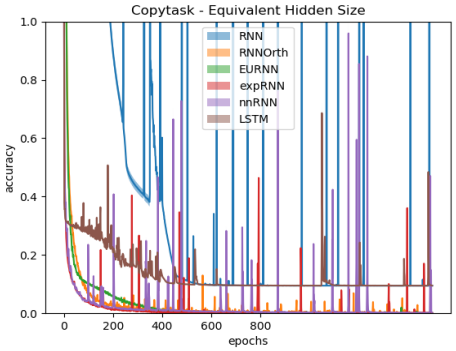
We conducted the experiment for copy task on the following models with the same hyperparameters as mentioned in the paper:

- RNN
- nnRNN
- LSTM
- expRNN

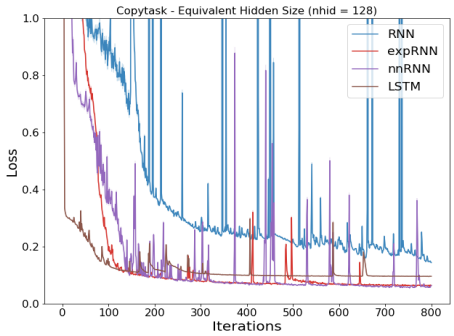
The task was trained for all the networks using mini batch sizes of 10. We were able to reproduce the results as mentioned in the paper for all the models listed above. We executed the author's code for Copy task for 6 evaluation runs and obtained the plot shown in 2a and 2b. We see that nnRNN and expRNN perform better than vanilla RNN and LSTM highlighting the advantages of preserving the orthogonal nature of transition matrices on tasks involving long-term memory or gradient propagation. These results are coherent with the authors' claim that nnRNN performs better than RNNs and LSTMs, and similar to expRNN. The values of the hyperparameters used in the experiments are available in Table 1.

4.2 Permuted sMINST

The sequential MNIST task measures the ability of an RNN to measure complex long term dependencies. The task is to identify the digit in an image and classify the same. The pixels of the image are



(a) Authors' Results



(b) Our Results

Figure 2: Performance of copy task with equivalent hidden size.

Model	hid	LR	LR orth	α	δ	T Decay	V init
RNN	128	0.001		0.9			Glorot Normal
nnRNN	128	0.0005	10^{-6}	0.99	0.0001	10^{-6}	Henaff
LSTM	128	0.0005		0.99			Glorot Normal
expRNN	128	0.001	0.0001	0.99			Henaff

Table 1: Hyperparameters for the copy task. Here, "hid" is hidden state size, "LR" is learning rate, "LR orth" is the learning rate of the orthogonal transition matrix (its skew symmetric matrix), α is the smoothing parameter of RMSprop, δ is the weight of L2 penalty applied to the rotation blocks', T decay is the weight of the L2 penalty applied on T, and "V init" is the initialization scheme for the state transition matrix.

given in a sequential manner to the networks. In order to add long term dependencies to the task the pixels are permuted and fed sequentially, thereby ensuring that the pixels which are dependent on each other are far from each other in the sequence.

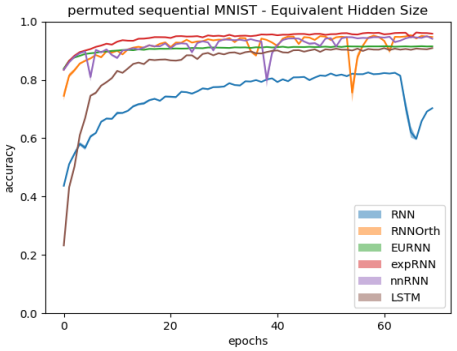
We conducted the experiment for copy task on the following models with the same hyperparameters as mentioned in the paper:

- RNN
- nnRNN
- LSTM
- expRNN

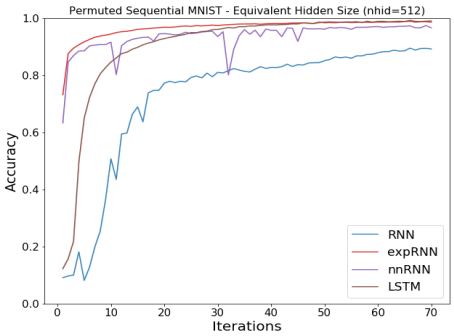
The task was trained for all the networks using mini batch sizes of 100. We were able to reproduce the results as mentioned in the paper for all the models listed above. For permuted Sequential MNIST task, we performed only one evaluation run. The results were plotted and is available in figure 3b. The values of the hyperparameters used in the experiments are available in Table 2.

Model	hid	LR	LR orth	α	δ	T Decay	V init
RNN	512	0.0001		0.9			Glorot Normal
nnRNN	512	0.0002	$2 * 10^{-5}$	0.99	0.1	0.0001	Cayley
LSTM	512	0.0005		0.9			Glorot Normal
expRNN	512	0.0005	$5 * 10^{-5}$	0.99			Cayley

Table 2: Hyperparameters for the permuted sequential MINST task. Here, "hid" is hidden state size, "LR" is learning rate, "LR orth" is the learning rate of the orthogonal transition matrix (its skew symmetric matrix), α is the smoothing parameter of RMSprop, δ is the weight of L2 penalty applied to the rotation blocks', T decay is the weight of the L2 penalty applied on T, and "V init" is the initialization scheme for the state transition matrix.



(a) Authors' Results



(b) Our Results

Figure 3: Permuted Sequential MINST.

4.3 Adding Task

In this task, the networks are provided with two sequences of length T. The first sequence consists of numbers sampled from the uniform random distribution between 0 and 1. The second sequence which acts as an indicator sequence, consists exactly of two entries of 1 and the remaining entries are 0. The position of 1 in the sequence is chosen uniformly at random. The task is to add the numbers from the first sequence whose position is the same as the positions where 1 is encountered in the second sequence.

We conducted the experiment for adding task on the following models:

- RNN
- nnRNN
- LSTM
- expRNN

The task was trained for all the networks using mini batch sizes of 10. The results were plotted and is available in figure 4. The values of the hyperparameters used in the experiments are available in Table 3.

Model	**hid**	**LR**	**LR orth**	α	δ	**T Decay**	**V init**	**Activation**
RNN	128	$2 * 10^{-4}$	$2 * 10^{-5}$	0.99		0.0001		tanh
nnRNN	512	$2 * 10^{-4}$	$2 * 10^{-5}$	0.99	0.0001	0.0001	Cayley	Mod ReLU
LSTM	128	$2 * 10^{-4}$	$2 * 10^{-5}$	0.99		0.0001		Mod ReLU
expRNN	512	$2 * 10^{-4}$	$2 * 10^{-5}$	0.99		0.0001	Cayley	Mod ReLU

Table 3: Hyperparameters for the adding task. Here, "hid" is hidden state size, "LR" is learning rate, "LR orth" is the learning rate of the orthogonal transition matrix (its skew symmetric matrix), α is the smoothing parameter of RMSprop, δ is the weight of L2 penalty applied to the rotation blocks', T decay is the weight of the L2 penalty applied on T, "V init" is the initialization scheme for the state transition matrix and Activation is the non-linear function applied.

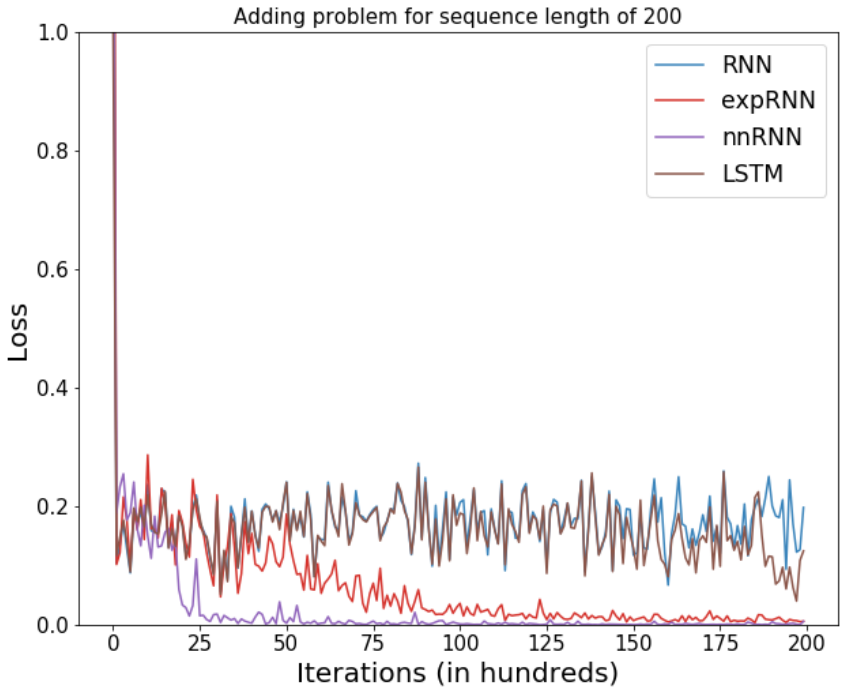


Figure 4: Adding Task.

5 Ablation Studies

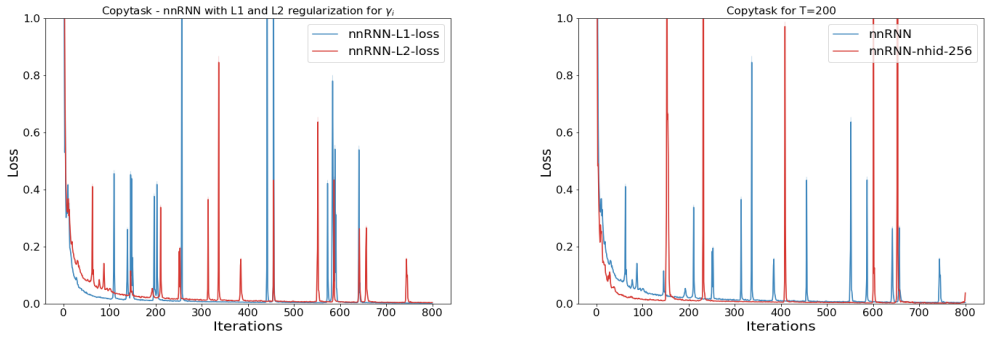
Our ablation studies is mainly restricted to copytask because permuted Sequential MNIST requires large computation time for training the models. We performed few experiments with the following hyperparameters.

- Optimization technique for gradient updates
- Orthogonal Initialization
- Regularization technique for γ parameters of nnRNN
- Hidden unit size

The γ parameters in nnRNN are optimized using Automatic Differentiation with an L2 regularization of $\delta||1-\gamma_i||_2^2$ applied on γ_i. Instead of L2 regularization, we applied an L1 regularization constraint of $\delta||1-\gamma_i||_1$. The resulting plot we obtained is shown in figure 5a. Our intuition behind the faster convergence of nnRNN with L1 regularization is that L1 penalizes more compared to L2 and causes the values of γ_i to be strictly around the value of 1 thereby not allowing to deviate much. This causes more eigen values to tend to 1.

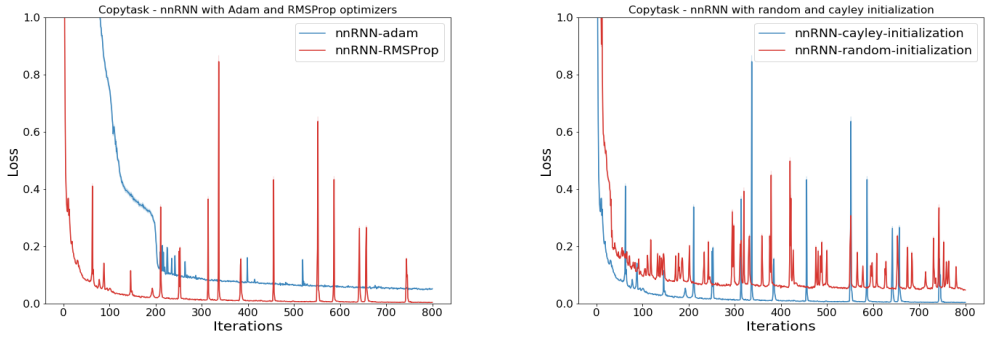
We increased the hidden unit size of nnRNN to 256 for the copytask. Our observation is that nnRNN converges faster compared to a lesser hidden unit size of 128. This is because increasing the hidden unit size leads to increase in the complexity of the model causing it learn faster.

We further experimented nnRNN with both Adam and RMSProp optimizers as mentioned in [6]. With RMSProp, the model converges faster and reaches a lower loss value, proving that RMSProp outperforms Adam in nnRNNs. Our plot is shown in figure 6a. As claimed in [2], cayley initialization performs better compared to random orthogonal initialization in nnRNNs as they converge slightly faster and have lower loss.



(a) (b)

Figure 5: Copytask experimented with gamma regularization (a) and hidden size(b)



(a) (b)

Figure 6: Copytask experimented with Adam optimizer (a) and Random orthogonal initialization (b)

6 Acknowledgements

We would like to thank the authors - Kyle Goyette, Giancarlo Kerg for their guidance. We would also like to thank Prof. Chenhao Tan and Kodur Krishna Chaitanya for their support. The compute resources required for this project was sponsored by CodeOcean.

References

[1] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[2] Kerg, Giancarlo and Goyette, Kyle and Touzel, Maximilian Puelma and Gidel, Gauthier and Vorontsov, Eugene and Bengio, Yoshua and Lajoie, Guillaume. Non-normal Recurrent Neural Network (nnRNN): learning long time dependencies while improving expressivity with transient dynamics. *arXiv preprint arXiv:1905.12080*, 2019.

[3] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[4] Surya Ganguli, Dongsung Huh, and Haim Sompolinsky. Memory traces in dynamical systems. *Proceedings of the National Academy of Sciences*, 105(48):18970–18975, 2008.

[5] Guillaume Hennequin, Tim P Vogels, and Wulfram Gerstner. Non-normal amplification in random balanced neuronal networks. *Physical Review E*, 86(1):011909, 2012.

[6] Mario Lezcano-Casado and David Martínez-Rubio. Cheap orthogonal constraints in neural networks: A simple parametrization of the orthogonal and unitary group. *arXiv preprint arXiv:1901.08428*, 2019.