Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization

Jiong Zhang[1] **Qi Lei**[1] **Inderjit S. Dhillon**[1 2]

Abstract

Vanishing and exploding gradients are two of the main obstacles in training deep neural networks, especially in capturing long range dependencies in recurrent neural networks (RNNs). In this paper, we present an efficient parametrization of the transition matrix of an RNN that allows us to stabilize the gradients that arise in its training. Specifically, we parameterize the transition matrix by its singular value decomposition (SVD), which allows us to explicitly track and control its singular values. We attain efficiency by using tools that are common in numerical linear algebra, namely Householder reflectors for representing the orthogonal matrices that arise in the SVD. By explicitly controlling the singular values, our proposed Spectral-RNN method allows us to provably solve the exploding gradient problem and we observe that it empirically solves the vanishing gradient issue to a large extent. We note that the SVD parameterization can be used for any rectangular weight matrix, hence it can be easily extended to any deep neural network, such as a multi-layer perceptron. Theoretically, we demonstrate that our parameterization does not lose any expressive power, and show how it controls generalization of RNN for the classification task. Our extensive experimental results also demonstrate that the proposed framework converges faster, and has good generalization, especially in capturing long range dependencies, as shown on the synthetic addition and copy tasks, as well as on the MNIST and Penn Tree Bank data sets.

1. Introduction

Deep neural networks have achieved great success in various fields, including computer vision, speech recognition, natural language processing, etc. Despite their tremendous capacity to fit complex functions, optimizing deep neural networks remains a contemporary challenge. Two main obstacles are vanishing and exploding gradients, that become particularly problematic in Recurrent Neural Networks (RNNs) since the transition matrix is identical along the temporal dimension, and any slight change to it is amplified through recurrent cells (Bengio et al., 1994).

Several methods have been proposed to solve the issue, for example, Long Short Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and residual networks (He et al., 2016). Another recently proposed class of methods is designed to enforce orthogonality of the square transition matrices, such as unitary and orthogonal RNNs (oRNN) (Arjovsky et al., 2016; Mhammedi et al., 2017). However, while these methods solve the exploding gradient problem, they limit the expressivity of the network.

In this paper, we present an efficient parametrization of transition matrices that arise in a deep neural network, thus allowing us to stabilize the gradients that arise in its training, while retaining the desired expressive power of the network. In more detail we make the following contributions:

- We propose a method to parameterize transition matrices through their singular value decomposition (SVD). Inspired by (Mhammedi et al., 2017), we attain efficiency by using tools that are common in numerical linear algebra, namely Householder reflectors for representing the orthogonal matrices that arise in the SVD. The SVD parametrization allows us to retain the desired expressive power of the network, while enabling us to explicitly track and control singular values.
- We apply our SVD parameterization to recurrent neural networks to exert spectral constraints on the RNN transition matrix. Our proposed Spectral-RNN method enjoys similar space and time complexity as the vanilla RNN. We empirically verify the superiority of Spectral-RNN over RNN/oRNN, in some case even LSTMs, over an exhaustive collection of time series classifica-

[1] University of Texas at Austin [2] Amazon.com. Correspondence to: Jiong Zhang <zhangjiong724@utexas.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018. Copyright 2018 by the author(s).

tion tasks and the synthetic addition and copy tasks, especially when the network depth is large.

- Theoretically, we prove that the generalization gap in margin loss of general RNN is bounded by the t-th power of the spectral norm of the transition matrix, where t is the recurrent temporal dimension. Therefore by controlling singular values we can reduce the population risk.
- Our parameterization is general enough to eliminate the gradient vanishing/exploding problem not only in RNNs, but also in various deep networks. We illustrate this by applying SVD parametrization to problems with non-square weight matrices, specifically multi-layer perceptrons (MLPs) and residual networks.

We now present the outline of our paper. In Section 2, we discuss related work, while in Section 3 we introduce our SVD parametrization and demonstrate how it spans the whole parameter space and does not limit expressivity. In Section 4 we propose the Spectral-RNN model that is able to efficiently control and track the singular values of the transition matrices, and we extend our parameterization to non-square weight matrices and apply it to MLPs in Section 5. Section 6 provides the theoretical analysis to show our method ensures good generalization for RNN. Experimental results on synthetic addition and copy tasks, and on MNIST and Penn Tree Bank data are presented in Section 7. Finally, we present our conclusions and future work in Section 8.

2. Related Work

Numerous approaches have been proposed to address the vanishing and exploding gradient problem. Long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) attempts to address the vanishing gradient problem by adding additional memory gates. Residual networks (He et al., 2016) pass the original input directly to the next layer in addition to the original layer output. (Mikolov, 2012) performs gradient clipping, while (Pascanu et al., 2013) apply spectral regularization to the weight matrices. Other approaches include introducing L_1 or L_2 penalization on successive gradient norm pairs in back propagation (Pascanu et al., 2013).

Recently the idea of restricting transition matrices to be orthogonal has drawn some attention. (Kanai et al., 2017) propose to constrain the leading singular value of the transition matrix during training of GRUs. (Le et al., 2015) propose initializing recurrent transition matrices to be identity or orthogonal (IRNN). This strategy shows better performance when compared to vanilla RNN and LSTM. However, there is no guarantee that the transition matrix is close to orthogonal after a few iterations. The unitary RNN (uRNN) algorithm proposed in (Arjovsky et al., 2016) parameterizes the transition matrix with reflection, diagonal and Fourier transform matrices. By construction, uRNN ensures that the transition matrix is unitary at all times. Although this algorithm performs well on several small tasks, (Wisdom et al., 2016) showed that uRNN only covers a subset of possible unitary matrices and thus detracts from the expressive power of RNN. An improvement over uRNN, the orthogonal RNN (oRNN), was proposed by (Mhammedi et al., 2017). oRNN uses products of Householder reflectors to represent an orthogonal transition matrix, which is rich enough to span the entire space of orthogonal matrices. Meanwhile, (Vorontsov et al., 2017) empirically demonstrate that the strong constraint of orthogonality limits the model's expressivity, thereby hindering its performance. Therefore, they parameterize the transition matrix by its SVD, $W = U\Sigma V^\top$ (factorized RNN) and restrict Σ to be in a range close to 1; however, the orthogonal matrices U and V are updated by geodesic gradient descent using the Cayley transform, thereby resulting in time complexity cubic in the number of hidden nodes which is prohibitive for large scale problems. Motivated by the shortcomings of the above methods, our work in this paper attempts to answer the following question: *Is there an efficient way to solve the gradient vanishing/exploding problem without hurting expressive power?*

Generalization is a major concern in training deep neural networks. (Neyshabur et al., 2017) and (Bartlett et al., 2017) provide margin-based generalization bounds for feedforward neural networks by a spectral Lipschitz constant, namely the product of spectral norm of each layer. We extend the analysis to recurrent neural networks and show our scheme of restricting the spectral norm of weight matrices reduces generalization error in the same setting as (Neyshabur et al., 2017). As supported by the analysis in (Cisse et al., 2017), since our SVD parametrization allows us to develop an efficient way to constrain the weight matrix to be a tight frame (Tropp et al., 2005), we consequently are able to reduce the sensitivity of the network to adversarial examples.

3. SVD parameterization

The SVD of the transition matrix $W \in \mathbb{R}^{n\times n}$ of an RNN is given by $W = U\Sigma V^T$, where Σ is the diagonal matrix of singular values, and $U, V \in \mathbb{R}^{n\times n}$ are orthogonal matrices, i.e., $U^TU = UU^T = I$ and $V^TV = VV^T = I$ (Trefethen & Bau III, 1997). During the training of an RNN, our proposal is to maintain the transition matrix in its SVD form. However, in order to do so efficiently, we need to maintain the orthogonal matrices U and V in compact form, so that they can be easily updated by forward and backward propagation. In order to do so, as in (Mhammedi et al., 2017), we use a tool that is commonly used in numerical linear algebra, namely Householder reflectors (which, for example, are used in computing the QR decomposition of a matrix).

Given a vector $u \in \mathbb{R}^k, k \leq n$, we define the $n \times n$ Householder reflector $\mathcal{H}_k^n(u)$ to be:

$$\mathcal{H}_k^n(u) = \begin{cases} \begin{pmatrix} I_{n-k} & \\ & I_k - 2\frac{uu^\top}{\|u\|^2} \end{pmatrix} & , \quad u \neq \mathbf{0} \\ I_n & , \quad \text{otherwise.} \end{cases} \tag{1}$$

The Householder reflector is clearly a symmetric matrix, and it can be shown that it is orthogonal, i.e., $H^2 = I$ (Householder, 1958). Further, when $u \neq 0$, it has $n-1$ eigenvalues that are 1, and one eigenvalue which is -1 (hence the name that it is a reflector) . In practice, to store a Householder reflector, we only need to store $u \in \mathbb{R}^k$ rather than the full matrix.

Given a series of vectors $\{u_i\}_{i=k}^n$ where $u_k \in \mathbb{R}^k$, we define the map:

$$\begin{aligned} \mathcal{M}_k : \mathbb{R}^k \times ... \times \mathbb{R}^n &\mapsto \mathbb{R}^{n \times n} \\ (u_k, ..., u_n) &\mapsto \mathcal{H}_n(u_n) ... \mathcal{H}_k(u_k), \end{aligned} \tag{2}$$

where the right hand side is a product of Householder reflectors, yielding an orthogonal matrix (to make the notation less cumbersome, we remove the superscript from $\mathcal{H}_k^n$ for the rest of this section). As shown below, any orthogonal matrix can be generated by this map.

Theorem 1. *The image of $\mathcal{M}_1$ is the set of all $n \times n$ orthogonal matrices.*

The proof of Theorem 1 is an easy extension of the Householder QR factorization Theorem, and is presented in Appendix A. Although we cannot express all $n \times n$ matrices with $\mathcal{M}_k$, any $W \in \mathbb{R}^{n \times n}$ can be expressed as the product of two orthogonal matrices U, V and a diagonal matrix Σ, i.e. by its SVD: $W = U \Sigma V^\top$. Given $\sigma \in \mathbb{R}^n$ and $\{u_i\}_{i=k_1}^n, \{v_i\}_{i=k_2}^n$ with $u_i, v_i \in \mathbb{R}^i$, we finally define our proposed SVD parametrization:

$$\begin{aligned} \mathcal{M}_{k_1,k_2} : & \mathbb{R}^{k_1} \times ... \times \mathbb{R}^n \times \mathbb{R}^{k_2} \times ... \times \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^{n \times n} \\ & (u_{k_1}, ..., u_n, v_{k_2}, ..., v_n, \sigma) \\ & \quad \mapsto \mathcal{H}_n(u_n) ... \mathcal{H}_{k_1}(u_{k_1}) diag(\sigma) \mathcal{H}_{k_2}(v_{k_2}) ... \mathcal{H}_n(v_n). \end{aligned} \tag{3}$$

Theorem 2. *The image of $\mathcal{M}_{1,1}$ is the set of $n \times n$ real matrices,*
i.e., $\mathbb{R}^{n \times n} = \mathcal{M}_{1,1}\left(\mathbb{R}^1 \times ... \times \mathbb{R}^n \times \mathbb{R}^1 \times ... \times \mathbb{R}^n \times \mathbb{R}^n\right)$

The proof of Theorem 2 is based on the singular value decomposition and Theorem 1, and is presented in Appendix A. The astute reader might note that $\mathcal{M}_{1,1}$ seemingly maps an input space of $n^2 + 2n$ dimensions to a space of n^2 dimensions; however, since $\mathcal{H}_k^n(u_k)$ is invariant to the norm of u_k, the input space also has exactly n^2 dimensions. Although Theorems 1 and 2 are simple extensions of well known linear algebra results, they ensure that our parameterization has the ability to represent any matrix and so the full expressive power of the RNN is retained.

Theorem 3. *The image of $\mathcal{M}_{k_1,k_2}$ includes the set of all orthogonal $n \times n$ matrices if $k_1 + k_2 \leq n + 2$.*

Theorem 3 indicates that if the total number of reflectors is greater than n: $(n - k_1 + 1) + (n - k_2 + 1) \geq n$, then the parameterization covers all orthogonal matrices. Note that when fixing $\sigma = \mathbf{1}$, $\mathcal{M}_{k_1,k_2}(\{u_i\}_{i=k_1}^n, \{v_i\}_{i=k_2}^n, \mathbf{1}) \in \mathbf{O}(n)$, where $\mathbf{O}(n)$ is the set of $n \times n$ orthogonal matrices. Thus when $k_1 + k_2 \leq n + 2$, we have $\mathbf{O}(n) = \mathcal{M}_{k_1,k_2}\left[\mathbb{R}^{k_1} \times ... \times \mathbb{R}^n \times \mathbb{R}^{k_2} \times ... \times \mathbb{R}^n \times \mathbf{1}\right]$.

4. Spectral-RNN

In this section, we apply our SVD parameterization to RNNs and describe the resulting Spectral-RNN algorithm in detail. Given a hidden state vector from the previous step $h^{(t-1)} \in \mathbb{R}^n$ and input $x^{(t)} \in \mathbb{R}^{n_i}$, RNN computes the next hidden state $h^{(t)}$ and output vector $\hat{y}^{(t)} \in \mathbb{R}^{n_y}$ as:

$$h^{(t)} = \phi(W h^{(t-1)} + M x^{(t)} + b), \tag{4}$$

$$\hat{y}^{(t)} = Y h^{(t)}. \tag{5}$$

In Spectral-RNN we parametrize the transition matrix $W \in \mathbb{R}^{n \times n}$ using $m_1 + m_2$ Householder reflectors as:

$$\begin{aligned} W &= \mathcal{M}_{k_1,k_2}(u_{k_1}, ..., u_n, v_{k_2}, ..., v_n, \sigma) \\ &= \mathcal{H}_n(u_n) ... \mathcal{H}_{k_1}(u_{k_1}) diag(\sigma) \mathcal{H}_{k_2}(v_{k_2}) ... \mathcal{H}_n(v_n) \end{aligned} \tag{6}$$

where $k_1 = n - m_1 + 1$, $k_2 = n - m_2 + 1$. This parameterization gives us several advantages over the regular RNN. First, we can select the number of reflectors m_1 and m_2 to balance expressive power versus time and space complexity. By Theorem 2, the choice $m_1 = m_2 = n$ gives us the same expressive power as vanilla RNN. Notice oRNN could be considered a special case of our parametrization, since when we set $m_1 + m_2 \geq n$ and $\sigma = \mathbf{1}$, we can represent all orthogonal matrices, as proven by Theorem 3. Most importantly, we are able to explicitly control the singular values of the transition matrix. In most cases, we want to constrain the singular values to be within a small interval near 1. The most intuitive method is to clip the singular values that are out of range. Another approach would be to initialize all singular values to 1, and add a penalty term $\|\sigma - 1\|^2$ to the objective function. Here, we have applied another parameterization of σ proposed in (Vorontsov et al., 2017):

$$\sigma_i = 2r(f(\hat{\sigma}_i) - 0.5) + \sigma^*, \; i \in [n] \tag{7}$$

where f is the sigmoid function and $\hat{\sigma}_i$ is updated from u_i, v_i via stochastic gradient descent. The above allows us to constrain σ_i to be within $[\sigma^* - r, \sigma^* + r]$. In practice, σ^* is usually set to 1 and $r \ll 1$. Note that we are not incurring more computation cost or memory for the parameterization. For regular RNN, the number of parameters is $(n_y + n_i + n + 1)n$, while for Spectral-RNN it is $(n_y + n_i + m_1 + m_2 + 2)n - \frac{m_1^2 + m_2^2 - m_1 - m_2}{2}$. In the extreme case where $m_1 = m_2 = n$, it becomes $(n_y + n_i + n + 3)n$. Later we will show that the computational cost of Spectral-RNN is also of the same order as RNN.

4.1. Forward/backward propagation

In forward propagation, we need to iteratively evaluate $h^{(t)}$ from $t = 0$ to L using (17). The only different aspect from a regular RNN in the forward propagation is the computation of $Wh^{(t-1)}$. Note that in Spectral-RNN, W is expressed as product of $m_1 + m_2$ Householder matrices and a diagonal matrix.Thus $Wh^{(t-1)}$ can be computed iteratively using $(m_1 + m_2)$ inner products and vector additions. Denoting $\hat{u}_k = \binom{\mathbf{0}_{n-k}}{u_k}$, we have:

$$\mathcal{H}_k(u_k)h = \left(I_n - \frac{2\hat{u}_k\hat{u}_k^\top}{\hat{u}_k^\top \hat{u}_k}\right)h = h - 2\frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k}\hat{u}_k \tag{8}$$

Thus, the total cost of computing $Wh^{(t-1)}$ is $O((m_1 + m_2)n)$ floating point operations (flops). Detailed analysis can be found in Section 4.2. Let $L(\{u_i\}, \{v_i\}, \sigma, M, Y, b)$ be the loss function, $\tilde{h}^{(t)} = Wh^{(t)}, \hat{\Sigma} = diag(\hat{\sigma})$. Given $\frac{\partial L}{\partial \tilde{h}^{(t)}}$, we define:

$$\frac{\partial L}{\partial u_k^{(t)}} := \left[\frac{\partial \tilde{h}^{(t)}}{\partial u_k^{(t)}}\right]^\top \frac{\partial L}{\partial \tilde{h}^{(t)}}; \; \frac{\partial L}{\partial v_k^{(t)}} := \left[\frac{\partial \tilde{h}^{(t)}}{\partial v_k^{(t)}}\right]^\top \frac{\partial L}{\partial \tilde{h}^{(t)}}; \tag{9}$$

$$\frac{\partial L}{\partial \Sigma^{(t)}} := \left[\frac{\partial \tilde{h}^{(t)}}{\partial \Sigma^{(t)}}\right]^\top \frac{\partial L}{\partial \tilde{h}^{(t)}}; \; \frac{\partial L}{\partial \hat{\Sigma}^{(t)}} := \left[\frac{\partial \Sigma^{(t)}}{\partial \hat{\Sigma}^{(t)}}\right]^\top \frac{\partial L}{\partial \Sigma^{(t)}}; \tag{10}$$

$$\frac{\partial L}{\partial h^{(t-1)}} := \left[\frac{\partial \tilde{h}^{(t)}}{\partial h^{(t-1)}}\right]^\top \frac{\partial L}{\partial \tilde{h}^{(t)}} \tag{11}$$

Back propagation for Spectral-RNN requires $\frac{\partial \tilde{h}^{(t)}}{\partial u_k^{(t)}}$, $\frac{\partial \tilde{h}^{(t)}}{\partial v_k^{(t)}}$, $\frac{\partial \tilde{h}^{(t)}}{\partial \hat{\Sigma}^{(t)}}$ and $\frac{\partial \tilde{h}^{(t)}}{\partial h^{(t-1)}}$. These partial gradients can also be computed iteratively by computing the gradient of each Householder matrix at a time. We drop the superscript (t) now for ease of exposition. Given $\hat{h} = \mathcal{H}_k(u_k)h$ and $g = \frac{\partial L}{\partial \hat{h}}$, we have

$$\frac{\partial L}{\partial h} = \left[\frac{\partial \hat{h}}{\partial h}\right]^\top \frac{\partial L}{\partial \hat{h}} = \left(I_n - \frac{2\hat{u}_k\hat{u}_k^\top}{\hat{u}_k^\top \hat{u}_k}\right)g = g - 2\frac{\hat{u}_k^\top g}{\hat{u}_k^\top \hat{u}_k}\hat{u}_k, \tag{12}$$

$$\begin{aligned}\frac{\partial L}{\partial \hat{u}_k} &= \left[\frac{\partial \hat{h}}{\partial \hat{u}_k}\right]^\top \frac{\partial L}{\partial \hat{h}} \\ &= -2\left(\frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k}I_n + \frac{1}{\hat{u}_k^\top \hat{u}_k}h\hat{u}_k^\top - 2\frac{\hat{u}_k^\top h}{(\hat{u}_k^\top \hat{u}_k)^2}\hat{u}_k\hat{u}_k^\top\right)g \\ &= -2\frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k}g - 2\frac{\hat{u}_k^\top g}{\hat{u}_k^\top \hat{u}_k}h + 4\frac{\hat{u}_k^\top h}{\hat{u}_k^\top \hat{u}_k}\frac{\hat{u}_k^\top g}{\hat{u}_k^\top \hat{u}_k}\hat{u}_k. \end{aligned}\tag{13}$$

Details of forward and backward propagation can be found in Appendix B.

4.2. Complexity Analysis

Table 1 gives the time complexity of various algorithms. $Hprod$ and $Hgrad$ are defined in Algorithm 2 and 3 (see Appendix B). Algorithm 2 needs $6k$ flops, while Algorithm 3 uses $(3n + 10k)$ flops. Since $\|u_k\|^2$ only needs to be computed once per iteration, we can further decrease the flops to $4k$ and $(3n + 8k)$. Also, in back propagation we can reuse α in forward propagation to save $2k$ flops. The efficiency of training Spectral-RNN can be improved by adopting the level 3 BLAS, blocked Householder QR algorithm (Andrew & Dingle, 2014) to exploit GPU computing power.

	flops
$Hprod(h, u_k)$	$4k$
$Hgrad(h, u_k, g)$	$3n + 6k$
Spectral-RNN-Local FP(n, m_1, m_2)	$4n(m_1 + m_2) - 2m_1^2 - 2m_2^2 + O(n)$
Spectral-RNN-Local BP(n, m_1, m_2)	$6n(m_1 + m_2) - 1.5m_1^2 - 1.5m_2^2 + O(n)$
oRNN-Local FP(n, m)	$4nm - m^2 + O(n)$
oRNN-Local BP(n, m)	$7nm - 2m^2 + O(n)$

Table 1. Time complexity across algorithms

5. Extending SVD Parameterization to General Weight Matrices

In this section, we extend the parameterization to non-square matrices and use Multi-Layer Perceptrons(MLP) as an example to illustrate its application to general deep networks. For any weight matrix $W \in \mathbb{R}^{m \times n}$(without loss of generality $m \leq n$), its reduced SVD can be written as:

$$W = U(\Sigma|0)(V_L|V_R)^\top = U\Sigma V_L^\top, \tag{14}$$

where $U \in \mathbb{R}^{m \times m}$, $\Sigma \in diag(\mathbb{R}^m)$,$V_L \in \mathbb{R}^{n \times m}$. There exist $u_n, ..., u_{k_1}$ and $v_n, ..., v_{k_2}$ s.t. $U = \mathcal{H}_m^m(u_m)...\mathcal{H}_{k_1}^m(u_{k_1})$, $V = \mathcal{H}_n^n(v_n)...\mathcal{H}_{k_2}^n(v_{k_2})$, where $k_1 \in [m], k_2 \in [n]$. Thus we can extend the SVD parameterization for any non-square matrix:

$$\begin{aligned}\mathcal{M}^{m,n}_{k_1,k_2} :&\mathbb{R}^{k_1}\times ... \times\mathbb{R}^{m}\times\mathbb{R}^{k_2}\times ...\times\mathbb{R}^{n}\times\mathbb{R}^{\min(m,n)}\\ &\mapsto \mathbb{R}^{m\times n}\\ &(u_{k_1},...,u_m,v_{k_2},...,v_n,\sigma)\\ &\mapsto \mathcal{H}^m_m(u_m)\cdots\mathcal{H}^m_{k_1}(u_{k_1})\hat{\Sigma}\mathcal{H}^n_{k_2}(v_{k_2})\cdots\mathcal{H}^n_n(v_n).\end{aligned} \tag{15}$$

where $\hat{\Sigma} = (diag(\sigma)|0)$ if $m < n$ and $(diag(\sigma)|0)^\top$ otherwise. Next we show that we only need $2\min(m,n)$ reflectors (rather than $m+n$) to parametrize any $m\times n$ matrix. By the definition of $\mathcal{H}^n_k$, we have the following lemma:

Lemma 1. *Given $\{v_i\}_{i=1}^n$, define $V^{(k)} = \mathcal{H}^n_n(v_n)...\mathcal{H}^n_k(v_k)$ for $k\in[n]$. We have:*

$$V^{(k_1)}_{*,i} = V^{(k_2)}_{*,i},\ \forall k_1,k_2\in[n],\ i\le\min(n-k_1,n-k_2).$$

Here $V_{*,i}$ indicates the ith column of matrix V. According to Lemma 1, we only need at most first m Householder vectors to express V_L, which results in the following Theorem:

Theorem 4. *If $m\le n$, the image of $\mathcal{M}^{m,n}_{1,n-m+1}$ is the set of all $m\times n$ matrices; else the image of $\mathcal{M}^{m,n}_{n-m+1,1}$ is the set of all $m\times n$ matrices.*

Similarly if we constrain u_i, v_i to have unit length, the input space dimensions of $\mathcal{M}^{m,n}_{1,n-m+1}$ and $\mathcal{M}^{m,n}_{m-n+1,1}$ are both mn, which matches the output dimension. Thus we extend Theorem 2 to the non-square case, which enables us to apply SVD parameterization to not only the RNN transition matrix, but also to general weight matrices in various deep learning models. For example, the Multilayer perceptron (MLP) model is a class of feedforward neural network with fully connected layers:

$$h^{(t)} = f(W^{(t-1)}h^{(t-1)} + b^{(t-1)}) \tag{16}$$

Here $h^{(t)}\in\mathbb{R}^{n_t}$, $h^{(t-1)}\in\mathbb{R}^{n_{t-1}}$ and $W^{(t)}\in\mathbb{R}^{n_t\times n_{t-1}}$. Applying SVD parameterization to $W^{(t)}$ say $n_t < n_{t-1}$, we have:

$$\begin{aligned}W^{(t)} =&\mathcal{H}^{n_t}_{n_t}(u_{n_t})...\mathcal{H}^{n_t}_1(u_1)\Sigma\\ &\cdot\mathcal{H}^{n_{t-1}}_{n_{t-1}-n_t+1}(v_{n_{t-1}-n_t+1})...\mathcal{H}^{n_{t-1}}_{n_{t-1}}(v_{n_{t-1}}).\end{aligned}$$

We can use the same forward/backward propagation algorithm as described in Algorithm 1. Besides RNN and MLP, our SVD parameterization also applies to more advanced frameworks, such as Residual networks and LSTM, which we will not describe in detail here.

6. Generalization Analysis

Since we can control and upper bound the singular values of the transition matrix in Spectral-RNN, we can clearly eliminate the exploding gradient problem. In this section, we provide the first generalization analysis for RNNs for the classification task, and prove that by upper bounding the singular values of the transition matrix, our Spectral-RNN approach ensures good generalization.

To study the generalization of general recurrent neural network, we simplify the network by absorbing the bias term b in M and consider:

$$\begin{aligned}&h^{(t)} = \phi(Wh^{(t-1)} + Mx^{(t)}), h^{(0)} = 0,\\ &\hat{y}^{(t)} = Yh^{(t)}.\end{aligned} \tag{17}$$

Recall from Section 4 we assume $x^{(t)}\in\mathbb{R}^{n_i}, \hat{y}^{(t)}\in\mathbb{R}^{n_y}$, and $h^{(t)}\in\mathbb{R}^n$. Therefore $W\in\mathbb{R}^{n\times n}, M\in\mathbb{R}^{n_i\times n}, Y\in\mathbb{R}^{n\times n_y}$. We let $h = \max\{n_i,n,n_y\}$ and write $w = \text{vec}(\{W,M,Y\})$ for ease of notation. Throughout the paper, we use $\|\cdot\|$ to denote l_2 norm for vectors and spectral norm for matrices unless otherwise specified. For the classification, we consider the following *Margin Loss* defined in (Neyshabur et al., 2017):

Definition 1. *For any distribution $\mathcal{D}$ and margin $\gamma > 0$, we define the expected margin loss as follows:*

$$L_\gamma(f_w) = \mathbb{P}_{(x,y)\sim\mathcal{D}}\left[f_w(x)[y]\le\gamma + \max_{j\ne y} f_w(x)[j]\right],$$

where $f_w(x)[y]$ is the probability of predicting y given input x with weight w. For recurrent neural network, $x = [x^{(1)}, x^{(2)}, \cdots x^{(T)}]$. We use $\hat{L}_\gamma$ to represent the empirical margin loss.

Assumption 1. *We consider the recurrent neural network (17) with the following assumptions:*

a *Data is bounded: $\|x^{(t)}\|\le B, t = 0,1,\cdots,$ for some constant B.*

b *Activation ϕ satisfies $\|\phi(x)\|_2\le\|x\|_2$, and $\|\phi(x)-\phi(y)\|_2\le\|x-y\|_2, \forall x\in\mathbb{R}^n$.*

The assumptions are natural, since common data like image pixels or embedding of words satisfy Assumption 1(a). Most common activations like ReLU, tanh and sigmoid function all satisfy Assumption 1(b). Under such assumptions, we get the following generalization bound for the recurrent neural network for classification task:

Theorem 5. *For any $B,T,n,n_i,n_y > 0$, let $f_w:\mathbb{R}^{n_i\times T}\to\mathbb{R}^{n_y}$ be a recurrent neural network with T time steps and n hidden nodes. Suppose the network satisfies Assumption 1 where data is bounded by B. Then for any $\delta,\gamma > 0$, with probability $\ge 1-\delta$ over a training set of size m, for any w, we have:*

$$L_0(f_w)\le\hat{L}_\gamma(f_w) + \mathcal{O}\left(\sqrt{\frac{G(w)+\ln\frac{m}{\delta}}{m}}\right),$$

where $G(w) = \frac{B^2T^4hln(h)}{\gamma^2} \cdot(\|W\|_F^2 + \|M\|_F^2 + \|Y\|_F^2) \max\{\|W\|^{2T-2}, 1\} \max\{\|M\|_2^2, 1\} \max\{\|Y\|_2^2, 1\}$, *and* $h = \max\{n, n_y, n_i\}$.

From Theorem 5, we can see that W plays a huge role since the generalization gap grows exponentially with $\|W\|$, i.e. the largest singular value of W. It is easy to see that our proposed Spectral-RNN approach, which bounds the singular radius of W in $[1-r, 1+r]$, ensures good generalization:

Corollary 1. *With the update rule in* (7)*, Spectral-RNN has generalization gap bounded by* $\mathcal{O}(\sqrt{\frac{G(w)+\ln\frac{m}{\delta}}{m}})$ *with probability* $\geq 1-\delta$*, where* $G(w) = \frac{B^2T^4h\ln(h)}{\gamma^2}(1+r)^{2T-2} \cdot \max\{\|M\|_2^2, 1\} \cdot \max\{\|Y\|_2^2, 1\}\|w\|_2^2$, $h = \max\{n, n_y, n_i\}$*, and* $\|w\|_2^2 = \|W\|_F^2 + \|M\|_F^2 + \|Y\|_F^2$.

The proof of Theorem 5 is presented in Appendix A, and uses the PAC-Bayes (McAllester, 2003) strategy as in (Neyshabur et al., 2017): a combination of the PAC-Bayes margin analysis (Lemma 2) and our perturbation analysis of the neural network in Lemma 3. (See Appendix A for Lemmas 2 and 3.)

Theorem 5 implies that a smaller matrix norm of the parameters leads to better generalization. This is easy to understand if we take an extreme example: when the norm of the matrices W, M and Y shrinks to 0 (norm), the output of the neural network will be constant and the generalization gap is obviously 0, but comes at the cost of expressivity of the network. Meanwhile, when the parameters are allowed to grow larger, the network will have higher expressivity but poorer generalization, meaning it could overfit the training data while not preserving the good performance on the test data. In this sense, when we control the range of the singular values of the matrix W, we are trying to reach a balance between expressivity and generalization gap of the network.

7. Experimental Results

In this section, we provide empirical evidence that shows the advantages of SVD parameterization in both RNNs and MLPs. For RNN models, we compare our Spectral-RNN algorithm with (vanilla) RNN, IRNN (Le et al., 2015), oRNN (Mhammedi et al., 2017) and LSTM (Hochreiter & Schmidhuber, 1997). The transition matrix in IRNN is initialized to be orthogonal while other matrices are initialized by sampling from a Gaussian distribution. For MLP models, we implemented vanilla MLP, Residual Network (ResNet) (He et al., 2016) and applied SVD parameterization on both of them. We used a residual block of two layers in ResNet. In most cases $leaky\_Relu$ is used as activation function except for LSTM. To train these models, we applied Adam optimizer with stochastic gradient descent (Kingma & Ba, 2014). These models are implemented with Tensorflow (Abadi et al., 2015).[1][2] Other than the experiments reported in this section, we provide UCR time series classification and multi-label learning results in Appendix C.

7.1. Addition and Copy tasks

We tested RNN models on the Addition and Copy tasks with the same settings as (Arjovsky et al., 2016).

Addition task: The Addition task requires the network to remember two marked numbers in a long sequence and add them. Each input data includes two sequences: top sequence whose values are sampled uniformly from $[0, 1]$ and bottom sequence which is a binary sequence with only two 1's. The network is asked to output the sum of the two values. From the empirical results in Figure 1, we can see that when the network is not deep (temporal dimension L=30 in (a)(d)), every model outperforms the baseline of 0.167 (i.e. always output 1 regardless of the input). Also, the gradients w.r.t. first cell do not vanish for all models. However, on longer sequences (L=100 in (b)(e)), IRNN fails and LSTM converges much slower than Spectral-RNN and oRNN. If we further increase the sequence length (L=300 in (c)(f)), only Spectral-RNN and oRNN are able to beat the baseline within a reasonable number of iterations. We can also observe that the gradient w.r.t. first cell of oRNN/Spectral-RNN does not vanish regardless of the depth, while IRNN/LSTM's gradients vanish as L becomes lager.

Copy task: Let $A = \{a_i\}_{i=0}^9$ be the alphabet and the input data sequence $x \in A^{T+20}$ where T is the time lag. $x_{1:10}$ are sampled uniformly from $\{a_i\}_{i=0}^7$ and x_{T+10} is set to a_9. The rest of x_i are set to a_8. The network is asked to output $x_{1:10}$ after seeing a_9, that is, to copy $x_{1:10}$ from the beginning to the end with time lag T.

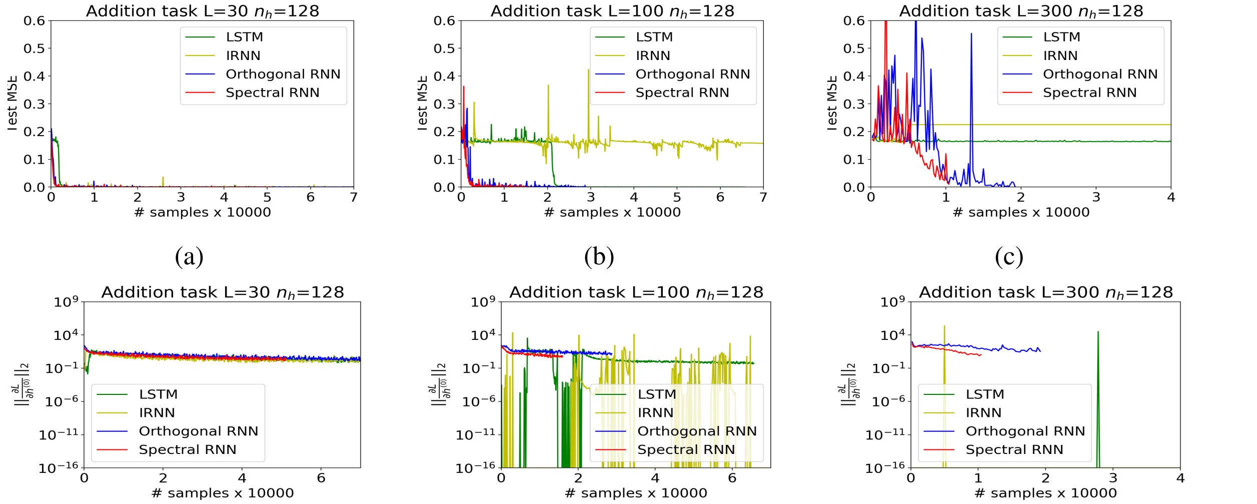
A baseline strategy is to predict a_8 for $T+10$ entries and randomly sample from $\{a_i\}_{i=1}^7$ for the last 10 digits. From the empirical results in Figure 2, Spectral-RNN consistently outperforms all other models. IRNN and LSTM models are not able to beat the baseline when the time lag is large. In fact, the test MSE for RNN/LSTM is very close to the baseline (memoryless strategy) indicating that they do not memorize any useful information throughout the larger time lag.

7.2. pixel-MNIST and permute-MNIST

In this experiment, we compare different models on the MNIST image dataset. The dataset was split into a training set of 60000 instances and a test set of 10000 instances.

[1] we thank Mhammedi for providing their code for oRNN (Mhammedi et al., 2017)

[2] The source code is available at `https://github.com/zhangjiong724/spectral-RNN`



(a) (b) (c)

(d) (e) (f)

Figure 1. RNN models on the addition task with sequence length L and hidden dimension of n_h. The top plots show the test MSE, while the bottom plots show the magnitude of the gradient at each corresponding step.

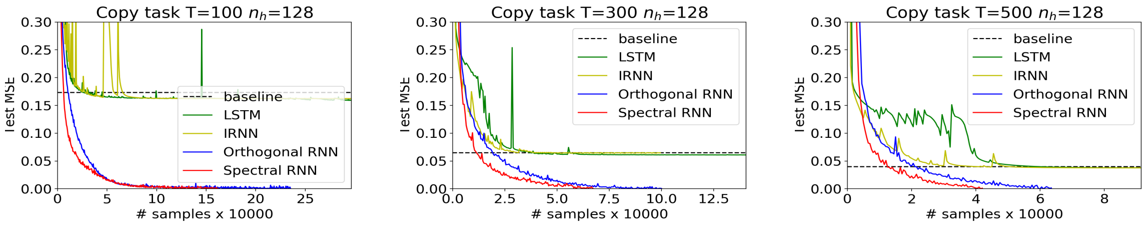


Figure 2. RNN models on the Copy task with time lag T and hidden dimension n_h.

The 28×28 MNIST pixels are flattened into a vector and then traversed by the RNN models. Table 2 shows test accuracy across multiple models. Spectral-RNN reaches the highest 97.7% accuracy on pixel-MNIST with only 128 hidden dimensions and 6k parameters.

Models	Hidden dimension	# parameters	Test accuracy
Spectral-RNN	128($m_1, m_2 = 16$)	$\approx 6k$	**97.7**
oRNN (Mhammedi et al., 2017)	256($m = 32$)	$\approx 11k$	97.2
RNN (Vorontsov et al., 2017)	128	$\approx 35k$	94.1
uRNN (Arjovsky et al., 2016)	512	$\approx 16k$	95.1
RC uRNN (Wisdom et al., 2016)	512	$\approx 16k$	97.5
FC uRNN (Wisdom et al., 2016)	116	$\approx 16k$	92.8
factorized RNN (Vorontsov et al., 2017)	128	$\approx 32k$	94.6
LSTM (Vorontsov et al., 2017)	128	$\approx 64k$	97.3

Table 2. Results for pixel MNIST across multiple algorithms

Figure 3(a)(b) plots the test accuracy on networks with 392 and 784 temporal steps respectively. We also tested models on the permuted-MNIST dataset, where we apply a fixed random permutation to the pixels before training. We performed a grid search over several learning rates $\rho = \{0.1, 0.01, 0.001, 0.0001\}$, decay rate $\alpha = \{0.9, 0.8, 0.5\}$ and batch size $B = \{64, 128, 256, 512\}$. The reported results are the best one among them. Figure 3(c) shows the test accuracy on permuted MNIST dataset. Also we explored the effect of different spectral constraints and explicitly tracked the spectral margin ($\max_i |\sigma_i - 1|$) of the transition matrix.

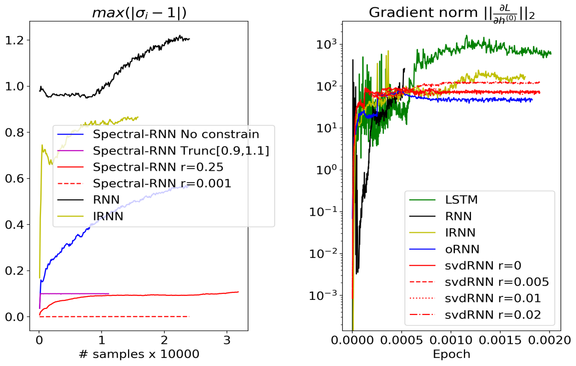
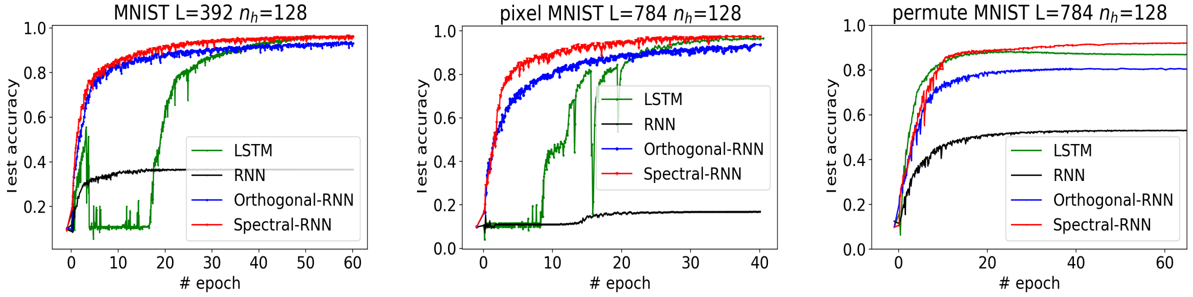


Figure 4. σ deviation and gradient magnitude

Figure 4 shows the spectral margin of different RNN models. Although IRNN has small spectral margin at first few iterations, it quickly deviates from being orthogonal. Figure 4 shows the magnitude of the gradient w.r.t. first cell $\|\frac{\partial L}{\partial h^{(0)}}\|_2$



(a) (b) (c)

Figure 3. RNN models on pixel-MNIST and permute-MNIST. Spectral-RNN constantly yields the highest test accuracy.

during training. RNN suffers from vanishing gradient at first several epochs, LSTM's gradient explode after several epochs while oRNN and Spectral-RNN have much more stable gradients. For the MLP models, each instance is flattened to a vector of length 784 and fed to the input layer. After the input layer there are 30-100 layers with hidden dimension 128 (Figure 5). On a shallow network, Spectral-MLP and Spectral-ResNet achieve similar performance as ResNet while MLP's convergence is slower. However, when the network is deeper, both MLP and ResNet start to fail. MLP is not able to function around $L \sim 35$ and ResNet with $L \sim 70$. On the other hand, the SVD based methods are resilient to increasing depth and thus achieve higher precision.

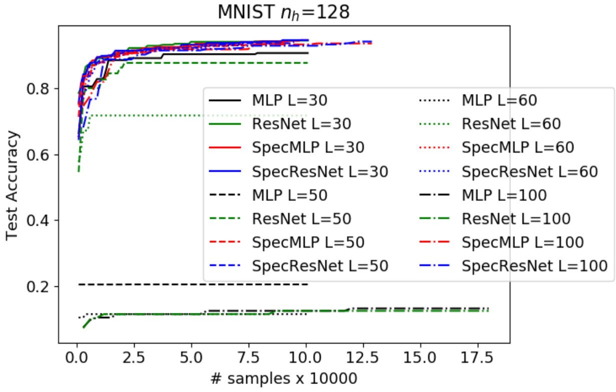


Figure 5. MLP models on MNIST with L layers and n_h hidden dimension. Spectral-based methods are resilient to increasing depth.

7.3. Penn Tree Bank dataset

We tested different models on Penn Tree Bank (PTB) (Marcus et al., 1993) dataset for word-level prediction tasks. The dataset contains 929k training words, 73k validation words, and 82k test words with 10k vocabulary. We trained 1- and 2-layered RNN models on word sequences of length 300. We adopted the successive mini-batches method (Zaremba et al., 2014), that use the final hidden state of the previous mini-batch as the initial state of the next one. We use initial learning rate of 0.1 and decay by factor of 0.8 at each epoch, and 80% dropout is applied on 2-layered models.

Models(n_l,n_h)	# parameters	Train perplexity	Test perplexity
RNN(1,128)	$\approx 16k$	68.1	144.7
LSTM(1,128)	$\approx 64k$	69.1	130.7
Spectral-RNN(1,512)	$\approx 31k$	65.4	130.2
RNN(2,128)	$\approx 32k$	62.6	142.5
LSTM(2,128)	$\approx 128k$	26.1	122.7
Spectral-RNN(2,512)	$\approx 63k$	36.0	121.3

Table 3. Penn Tree Bank word level prediction

As seen in Table 3, Spectral-RNN achieves better performance than LSTM with about half the number of parameters. Note that 2-layered Spectral-RNN achieves lower test perplexity with higher training perplexity, which shows its generalization ability.

8. Conclusions

In this paper, we have proposed an efficient SVD parametrization of weight matrices in deep neural networks, which allows us to explicitly track and control their singular values. This parameterization does not restrict the network's expressive power, while simultaneously allowing fast forward as well as backward propagation. The method is easy to implement and has the same time and space complexity as compared to original methods like RNN and MLP. The ability to control singular values helps in avoiding the gradient vanishing and exploding problems, and as we have empirically shown, gives good performance. However, further experimentation is required to fully understand the influence of using different number of reflectors in our SVD parameterization. Also, the underlying structures of the image of $\mathcal{M}_{k_1,k_2}$ when $k_1, k_2 \neq 1$ is a subject worth investigating.

Acknowledgement This research was supported by NSF grants CCF1320746, IIS-1546452 and CCF-1564000.

References

Abadi, Martin et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Andrew, Robert and Dingle, Nicholas. Implementing QR factorization updating algorithms on GPUs. *Parallel Computing*, 40(7):161–172, 2014.

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

Bartlett, Peter, Foster, Dylan J, and Telgarsky, Matus. Spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1706.08498*, 2017.

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994.

Chen, Yanping, Keogh, Eamonn, Hu, Bing, Begum, Nurjahan, Bagnall, Anthony, Mueen, Abdullah, and Batista, Gustavo. The UCR time series classification archive, July 2015. www.cs.ucr.edu/~eamonn/time_series_data/.

Cisse, Moustapha, Bojanowski, Piotr, Grave, Edouard, Dauphin, Yann, and Usunier, Nicolas. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine Learning*, pp. 854–863, 2017.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Householder, Alston S. Unitary triangularization of a nonsymmetric matrix. *Journal of the ACM (JACM)*, 5(4): 339–342, 1958.

Hüsken, Michael and Stagge, Peter. Recurrent neural networks for time series classification. *Neurocomputing*, 50: 223–235, 2003.

Kanai, Sekitoshi, Fujiwara, Yasuhiro, and Iwamura, Sotetsu. Preventing gradient explosions in gated recurrent units. In *Advances in Neural Information Processing Systems*, pp. 435–444, 2017.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Marcus, Mitchell P, Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a large annotated corpus of english: The Penn Treebank. *Computational linguistics*, 19(2):313–330, 1993.

McAllester, David. Simplified PAC-Bayesian margin bounds. In *Learning theory and Kernel machines*, pp. 203–215. Springer, 2003.

Mhammedi, Zakaria, Hellicar, Andrew, Rahman, Ashfaqur, and Bailey, James. Efficient orthogonal parametrisation of recurrent neural networks using Householder reflections. In *International Conference on Machine Learning*, pp. 2401–2409, 2017.

Mikolov, Tomáš. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

Neyshabur, Behnam, Bhojanapalli, Srinadh, McAllester, David, and Srebro, Nathan. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564*, 2017.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

Trefethen, Lloyd N and Bau III, David. *Numerical linear algebra*, volume 50. SIAM, 1997.

Tropp, Joel A, Dhillon, Inderjit S, Heath, Robert W, and Strohmer, Thomas. Designing structured tight frames via an alternating projection method. *IEEE Transactions on information theory*, 51(1):188–209, 2005.

Vorontsov, Eugene, Trabelsi, Chiheb, Kadoury, Samuel, and Pal, Chris. On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning*, pp. 3570–3578, 2017.

Wisdom, Scott, Powers, Thomas, Hershey, John, Le Roux, Jonathan, and Atlas, Les. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 4880–4888, 2016.

Zaremba, Wojciech, Sutskever, Ilya, and Vinyals, Oriol. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

A. Proofs

A.1. Proof of Proposition 1

Proposition 1. *(Householder QR factorization) Let $B \in \mathbb{R}^{n\times n}$. There exists an upper triangular matrix R with positive diagonal elements, and vectors $\{u_i\}_{i=1}^n$ with $u_i \in \mathbb{R}^i$, such that $B = \mathcal{H}_n^n(u_n)...\mathcal{H}_1^n(u_1)R$. (Note that we allow $u_i = 0$, in which case, $H_i^n(u_i) = I_n$ as in (1))*

Proof of Proposition 1. For $n = 1$, note that $\mathcal{H}_1^1(u_1) = \pm 1$. By setting $u_1 = 0$ if $B_{1,1} > 0$ and $u_1 \neq 0$ otherwise, we have the factorization desired.

Assume that the result holds for $n = k$, then for $n = k + 1$ set $u_{k+1} = B_1 - \|B_1\|e_1$. Here B_1 is the first column of B and $e_1 = (1, 0, ..., 0)^\top$. Thus we have

$$\mathcal{H}_{k+1}^{k+1}(u_{k+1})B = \begin{pmatrix} \|B_1\| & \hat{B}_{1,2:k+1} \\ 0 & \hat{B} \end{pmatrix},$$

where $\hat{B} \in \mathbb{R}^{k\times k}$. Note that $\mathcal{H}_{k+1}^{k+1}(u_{k+1}) = I_{k+1}$ when $u_{k+1} = 0$ and the above still holds. By assumption we have $\hat{B} = \mathcal{H}_k^k(u_k)...\mathcal{H}_1^k(u_1)\hat{R}$. Notice that $\mathcal{H}_i^{k+1}(u_i) = \begin{pmatrix} 1 & \\ & \mathcal{H}_i^k(u_i) \end{pmatrix}$, so we have that

$$\mathcal{H}_1^{k+1}(u_1)...\mathcal{H}_k^{k+1}(u_k)\mathcal{H}_{k+1}^{k+1}(u_{k+1})B = \begin{pmatrix} \|B_1\| & \tilde{B}_{1,2:k+1} \\ 0 & \hat{R} \end{pmatrix} = R$$

is an upper triangular matrix with positive diagonal elements. Thus the result holds for any n by the theory of mathematical induction. □

A.2. Proof of Theorem 1

Proof. Observe that the image of $\mathcal{M}_1$ is a subset of $\mathbf{O}(n)$, and we now show that the converse is also true. Given $A \in \mathbf{O}(n)$, by Proposition 1, there exists an upper triangular matrix R with positive diagonal elements, and an orthogonal matrix Q expressed as $Q = \mathcal{H}_n^n(u_n)...\mathcal{H}_1^n(u_1)$ for some set of Householder vectors $\{u_i\}_{i=1}^n$, such that $A = QR$. Since A is orthogonal, we have $A^\top A = AA^\top = I_n$, thus:

$$A^\top A = R^\top Q^\top QR = R^\top R = I_n;\; Q^\top AA^\top Q = Q^\top QRR^\top Q^\top Q = RR^\top = I_n$$

Thus R is orthogonal and upper triangular matrix with positive diagonal elements. So $R = I_n$ and $A = Q = \mathcal{H}_n^n(u_n)...\mathcal{H}_1^n(u_1)$. □

A.3. Proof of Theorem 2

Proof. It is easy to see that the image of $\mathcal{M}_{1,1}$ is a subset of $\mathbb{R}^{n\times n}$. For any $W \in \mathbb{R}^{n\times n}$, we have its SVD, $W = U\Sigma V^\top$, where $\Sigma = diag(\sigma)$. By Theorem 1, for any orthogonal matrix $U, V \in \mathbb{R}^{n\times n}$, there exists $\{u_i\}_{i=1}^n\{v_i\}_{i=1}^n$ such that $U = \mathcal{M}_1(u_1, ..., u_n)$ and $V = \mathcal{M}_1(v_1, ..., v_n)$, then we have:

$$\begin{aligned} W &= \mathcal{H}_n^n(u_n)...\mathcal{H}_1^n(u_1)\Sigma\mathcal{H}_1^n(v_1)...\mathcal{H}_n^n(v_n) \\ &= \mathcal{M}_{1,1}(u_1, ..., u_n, v_1, ..., v_n, \sigma) \end{aligned}$$

□

A.4. Proof of Theorem 3

Proof. Let $A \in \mathbb{R}^{n\times n}$ be an orthogonal matrix. By Theorem 1, there exist $\{a_i\}_{i=1}^n$, such that $A = \mathcal{M}_1(a_1, ..., a_n)$. Since $A^\top$ is also orthogonal, for the same reason, there exist $\{b_i\}_{i=1}^n$, such that $A^\top = \mathcal{M}_1(b_1, ..., b_n)$. Thus we have:

$$A = \mathcal{H}_n(a_n)...\mathcal{H}_1(a_1) = \mathcal{H}_1(b_1)...\mathcal{H}_n(b_n)$$

Observe that one of $k_2 \geq k_1 - 1$ and $k_1 \geq k_2 - 1$ must be true. If $k_2 \geq k_1 - 1$, set

$$\begin{aligned} u_k &= a_k, k = n, n-1, ..., k_1, \\ v_{k_2+k_1-k-1} &= a_k, k = k_1 - 1, ..., 1, \\ v_t &= \mathbf{0}, t = k_2 + k_1 - 2, ..., n, \end{aligned} \tag{18}$$

and then we have:

$$\begin{aligned}\mathcal{M}_{k_1,k_2}(u_{k_1},...,u_n,v_{k_2},...,v_n,\mathbf{1}) &= \mathcal{H}_n(u_n)...\mathcal{H}_{k_1}(u_{k_1})I_n\mathcal{H}_{k_2}(v_{k_2})...\mathcal{H}_n(v_n)\\ &= \mathcal{H}_n(a_n)...\mathcal{H}_{k_1}(a_{k_1})I_n\mathcal{H}_{k_1-1}(a_{k_1-1})...\mathcal{H}_1(a_1)\\ &= A \qquad (19)\end{aligned}$$

Else, assign:

$$\begin{aligned} v_k &= b_k, k = n, n-1, ..., k_2,\\ u_{k_2+k_1-k-1} &= b_k, k = k_2-1, ..., 1, \qquad (20)\\ u_t &= \mathbf{0}, t = k_2+k_1-2, ..., n,\end{aligned}$$

and then we have:

$$\begin{aligned}\mathcal{M}_{k_1,k_2}(u_{k_1},...,u_n,v_{k_2},...,v_n,\mathbf{1}) &= \mathcal{H}_1(b_1)...\mathcal{H}_{k_2-1}(b_{k_2-1})I_n\mathcal{H}_{k_2}(b_{k_2})...\mathcal{H}_n(b_n)\\ &= A \qquad (21)\end{aligned}$$

□

A.5. Proof of Theorem 4

Proof. It is easy to see that the image of $\mathcal{M}^{m,n}_{*,*}$ is a subset of $\mathbb{R}^{m\times n}$. For any $W \in \mathbb{R}^{m\times n}$, we have its SVD, $W = U\Sigma V^\top$, where Σ is an $m \times n$ diagonal matrix. By Theorem 1, for any orthogonal matrix $U \in \mathbb{R}^{m\times m}, V \in \mathbb{R}^{n\times n}$, there exists $\{u_i\}_{i=1}^m\{v_i\}_{i=1}^n$ such that $U = \mathcal{H}^m_m(u_m)...\mathcal{H}^m_1(u_1)$ and $V = \mathcal{H}^n_n(v_n)...\mathcal{H}^n_1(v_1)$. By Lemma 1, if $m < n$ we have:

$$\begin{aligned} W &= \mathcal{H}^m_n(u_n)...\mathcal{H}^m_1(u_1)\Sigma\mathcal{H}^n_1(v_1)...\mathcal{H}^n_n(v_n)\\ &= \mathcal{H}^m_n(u_n)...\mathcal{H}^m_1(u_1)\Sigma\mathcal{H}^n_{n-m+1}(v_{n-m+1})...\mathcal{H}^n_n(v_n).\end{aligned}$$

Similarly, for $n < m$, we have:

$$\begin{aligned} W &= \mathcal{H}^m_n(u_n)...\mathcal{H}^m_1(u_1)\Sigma\mathcal{H}^n_1(v_1)...\mathcal{H}^n_n(v_n)\\ &= \mathcal{H}^m_n(u_n)...\mathcal{H}^m_{m-n+1}(u_{m-n+1})\Sigma\mathcal{H}^n_1(v_1)...\mathcal{H}^n_n(v_n).\end{aligned}$$

□

A.6. Proof of Theorem 5

Notations: Recall from Definition 1 that L_0 is the expected error with margin $\gamma = 0$, and we write $\hat{L}_\gamma$ as the empirical error when margin equals γ with m samples, i.e.,

$$\hat{L}_\gamma(f_w) = \frac{1}{m}\sum_{i=1}^m\left[f_w(x_i)[y_i] \le \gamma + \max_{j\neq y_i} f_w(x_i)[j]\right].$$

We are looking at a recurrent neural network with T time steps:

$$\begin{aligned} &h^{(t)} = \phi(Wh^{(t-1)} + Mx^{(t)}), h^{(0)} = 0, t = 1, 2, \cdots T\\ &\hat{y}^{(t)} = Yh^{(t)},\end{aligned}$$

where ϕ is the activation function. The dimensions are as follows: $x^{(t)} \in \mathbb{R}^{n_i}, \hat{y}^{(t)} \in \mathbb{R}^{n_y}$, and $h^{(t)} \in \mathbb{R}^n$. Therefore $W \in \mathbb{R}^{n\times n}, M \in \mathbb{R}^{n_i\times n}, Y \in \mathbb{R}^{n\times n_y}$. To incorporate the different parameters W, M, Y into the neural network, we write $w = \text{vec}(\{W, Y, M\})$ and use subscript w to denote dependence on the parameter w. For instance, $h^{(t)}_w$ denotes the activation that takes $w = \text{vec}(\{W, Y, M\})$ as parameters, and similar notation also holds for the output $\hat{y}^{(t)}_w$. We use $\|\cdot\|$ to denote l_2 norm for vectors and spectral norm for matrices when there is no ambiguity.

To get a generalization bound for RNN, we need to use the following lemma from (Neyshabur et al., 2017).

Lemma 2. *(Neyshabur et al., 2017) Let $f_w(x) : \mathcal{X} \to \mathbb{R}^k$ be any predictor (not necessarily a neural network) with parameters w, and P be any distribution on the parameters that is independent of the training data. Then, for any $\gamma, \delta > 0$, with probability $\geq 1 - \delta$ over the training set of size m, for any w, and any random perturbation u s.t. $\mathbb{P}_u[\max_{x \in \mathcal{X}} \|f_{w+u}(x) - f_w(x)\|_\infty < \frac{\gamma}{4}] \geq \frac{1}{2}$, we have:*

$$L_0(f_w) \leq \hat{L}_\gamma(f_w) + 4\sqrt{\frac{KL(w+u||P) + \ln \frac{6m}{\delta}}{m-1}}$$

Here $KL(P||Q)$ is the Kullback-Leibler divergence of two continuous random variables P and Q:

$$KL(P||Q) := \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx,$$

where p and q denote the density of P and Q. In order for the random variable u to satisfy the probability property in Lemma 2, we study the change in output with respect to perturbation u.

Lemma 3. *Write $w = vec(\{W, Y, M\})$, and perturbation $u = vec(\{\delta W, \delta Y, \delta M\})$ such that $\|\delta W\| \leq \frac{1}{T}\|W\|$, $\|\delta Y\| \leq \frac{1}{T}\|Y\|$, $\|\delta M\| \leq \frac{1}{T}\|M\|$. For a recurrent neural network (17) with T time steps that satisfies Assumption 1, the perturbation in the activation is bounded by*

$$\|h_{w+u}^{(T)} - h_w^{(T)}\| \leq BTe(T\|M\|\|\delta W\| + \|\delta M\|) \max\{\|W\|^{T-1}, 1\}, \tag{22}$$

while the perturbation in the output satisfies:

$$\|\hat{y}_{w+u}^{(T)} - \hat{y}_w^{(T)}\| \leq TB \max\{\|W\|^{T-1}, 1\} \cdot (\|Y\|\|\delta W\|\|M\|Te + \|Y\|\|\delta M\|e + \|\delta Y\|\|M\|).$$

Here e is the natural logarithm base.

Proof of Lemma 3. First we bound the norm of $h_w^{(t)}$,

$$\begin{aligned}
\|h_w^{(t)}\| &= \|\phi(Wh_w^{(t-1)} + Mx^{(t)})\| \\
&\leq \|Wh_w^{(t-1)} + Mx^{(t)}\| && \text{(by Assumption 1.2)} \\
&\leq \|W\|\|h_w^{(t-1)}\| + \|M\|\|x^{(t)}\| && \text{(by triangle inequality)} \qquad (23) \\
&\leq \|W\| \left(\|W\|\|h_w^{(t-2)}\| + \|M\|\|x^{(t-1)}\|\right) + \|M\|\|x^{(t)}\| \\
& && \text{(applying (23) to } \|h_w^{(t-1)}\|) \\
&\leq \cdots \\
&\leq \|W\|^t\|h_w^{(0)}\| + \|M\| \sum_{j=0}^{t-1} \|W\|^{t-1-j}\|x^{(j+1)}\| \\
&= \|M\| \sum_{j=0}^{t-1} \|W\|^{t-1-j}\|x^{(j+1)}\| && \text{(since } h_w^{(0)} = 0) \\
&\leq B\|M\| \sum_{j=0}^{t-1} \|W\|^{t-1-j} && \text{(by Assumption 1.1)} \\
\implies \|h_w^{(t)}\| &\leq B\|M\|t \max\{\|W\|^{t-1}, 1\} && (24) \\
& && \text{(since } \sum_{i=0}^{t-1} \|W\|^i \leq t \max\{\|W\|^{t-1-i}, 1\})
\end{aligned}$$

Denoting $\Delta_t = \|h_{w+u}^{(t)} - h_w^{(t)}\|$ for short, in order to prove (22), we now prove the following tighter result by induction,

$$\Delta_t \leq Bt(1 + \frac{1}{T})^{t-1}(\|\delta W\|\|M\|T + \|\delta M\|) \max\{\|W\|^{t-1}, 1\}, \forall t \leq T \tag{25}$$

Clearly $\Delta_0 = 0$ satisfies the inequality. Suppose Δ_{t-1} satisfies the assumption, then,

$$
\begin{aligned}
\Delta_t &= \|\phi\left((W+\delta W)h_{w+u}^{(t-1)} + (M+\delta M)x^{(t)}\right) - \phi\left(Wh_w^{(t-1)} + Mx^{(t)}\right)\| \\
&\leq \|\left((W+\delta W)h_{w+u}^{(t-1)} + (M+\delta M)x^{(t)}\right) - \left(Wh_w^{(t-1)} + Mx^{(t)}\right)\| \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Assumption 1.2)} \\
&= \|(W+\delta W)(h_{w+u}^{(t-1)} - h_w^{(t-1)}) + \delta W h_w^{(t-1)} + \delta M x^{(t)}\| \\
&\leq (\|W\| + \|\delta W\|\Delta_{t-1} + \|\delta W\|\|h_w^{(t-1)}\| + \|\delta M\|\|x^{(t)}\| \qquad \text{(by triangle inequality)} \\
&\leq (1+\frac{1}{T})\|W\|\Delta_{t-1} + \|\delta W\|\|h_w^{(t-1)}\| + \|\delta M\|B \\
&\qquad\qquad\qquad\qquad\qquad\qquad \text{(by Assumption 1.1 and requirement of } \|\delta W\|\text{)}
\end{aligned}
$$

Then by induction and the bound of the activations, we have:

$$
\begin{aligned}
\Delta_t \leq &(1+\frac{1}{T})\|W\|\left(B(t-1)(1+\frac{1}{T})^{t-2}(\|\delta W\|\|M\|T + \|\delta M\|)\max\{\|W\|^{t-2},1\}\right) && \text{(by induction)} \\
&+ \|\delta W\|\left(B(t-1)\|M\|\max\{\|W\|^{t-2},1\}\right) + B\|\delta M\| && \text{(by activation bound (24))} \\
=&B(t-1)T(1+\frac{1}{T})^{t-1}\|\delta W\|\|M\|\|W\|\max\{\|W\|^{t-2},1\} + B(t-1)\|\delta W\|\|M\|\max\{\|W\|^{t-2},1\} \\
&+ (1+\frac{1}{T})^{t-1}\|W\|B(t-1)\max\{\|W\|^{t-2},1\}\|\delta M\| + B\|\delta M\| \\
=&B(t-1)\|\delta W\|\|M\|\max\{\|W\|^{t-2},1\}\left((1+\frac{1}{T})^{t-1}T\|W\| + 1\right) \\
&+ B\|\delta M\|\left((1+\frac{1}{T})^{t-1}\|W\|(t-1)\max\{\|W\|^{t-2},1\} + 1\right) \\
\leq&B(t-1)\|\delta W\|\|M\|\max\{\|W\|^{t-2},1\}\left((1+\frac{1}{T})^{t-1}T + 1\right)\max\{\|W\|,1\} \\
&+ B\|\delta M\|\left((1+\frac{1}{T})^{t-1}(t-1)\max\{\|W\|^{t-2},1\} + 1\right)\max\{\|W\|,1\} && \text{(both } 1, \|W\| \leq \max\{\|W\|,1\}\text{)} \\
\leq&B\|\delta W\|\|M\|tT(1+\frac{1}{T})^{t-1}\max\{\|W\|^{t-1},1\} \\
&+ B\|\delta M\|t(1+\frac{1}{T})^{t-1}\max\{\|W\|^{t-1},1\} && \text{(since } (t-1)a+1 \leq ta \text{ for } a \geq 1\text{)} \\
=&Bt(1+\frac{1}{T})^{t-1}(T\|\delta W\|\|M\| + \|\delta M\|)\max\{\|W\|^{t-1},1\}
\end{aligned}
$$

Since $(1+\frac{1}{T})^{T-1} \leq e$, therefore $\Delta_T \leq BTe(T\|M\|\|\delta W\| + \|\delta M\|)\max\{\|W\|^{T-1},1\}$. Meanwhile for the perturbation of output $\hat{y}$,

$$
\begin{aligned}
&\|\hat{y}_{w+u}^{(T)} - \hat{y}_w^{(T)}\| \\
=&\|(Y+\delta Y)h_{w+u}^{(T)} - Yh_w^{(T)}\| \\
=&\|(Y+\delta Y)(h_{w+u}^{(T)} - h_w^{(T)}) + (Y+\delta Y)h_w^{(T)} - Yh_w^{(T)}\| \\
\leq&\|(Y+\delta Y)\|\Delta_T + \|\delta Y h_w^{(T)}\| && \text{(by triangle inequality)} \\
\leq&\|Y\|(1+\frac{1}{T})BT(1+\frac{1}{T})^{T-1}(T\|\delta W\|\|M\| + \|\delta M\|)\max\{\|W\|^{T-1},1\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by perturbation bound (25))} \\
&+ \|\delta Y\|TB\|M\|\max\{\|W\|^{T-1},1\} && \text{(by activation bound (24))} \\
\leq&TB\max\{\|W\|^{T-1},1\}(\|Y\|\|\delta W\|\|M\|Te + \|Y\|\|\delta M\|e + \|\delta Y\|\|M\|) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(since } (1+\frac{1}{T})^T \leq e\text{)}
\end{aligned}
$$

□

Finally we are able to prove Theorem 5:

Proof of Theorem 5. In order to finish the proof, we first calculate the maximum allowed perturbation u that satisfies the requirement in Lemma 2, and we define the prior P and calculate the KL divergence of P and $w + u$.

Let $\beta = \max\{\|W\|_2^{T-1}, 1\} \max\{\|Y\|_2, 1\} \max\{\|M\|_2, 1\}$. We choose the distribution of the prior $P = \mathcal{N}(0, \sigma^2 I)$ and consider the random perturbation $u = \text{vec}(\{\delta W, \delta Y, \delta M\})$ with the same zero mean Gaussian distribution, where σ will be assigned later according to β. More precisely, since the prior cannot depend on the β which is associated with the learned parameters W, M and Y, we will set σ based on some discrete choices of $\tilde{\beta}$ that approximates β. For each value of $\tilde{\beta}$ of our choice, we will compute the PAC-Bayes bound, establishing the generalization guarantee for all w for which $\frac{1}{e}\beta \le \tilde{\beta} \le e\beta$, and ensuring that each relevant value of β is covered by some $\tilde{\beta}$ on the grid. We will then take a union bound over all $\tilde{\beta}$ of our choice.

For a random matrix $X \in \mathbb{R}^{n_1 \times n_2}$ with individual entries following normal distribution, (Tropp et al., 2005) provides the following bound of its spectral norm:

$$\mathbb{P}_{X \sim \mathcal{N}(0, \sigma^2 I)}[\|X\|_2 > t] \le 2ne^{-t^2/2n\sigma^2}, \forall n \ge n_1, n_2 \tag{26}$$

Therefore for $\delta W, \delta M, \delta Y$, the probability of their spectral norm being greater than t is bounded by $2he^{-t^2/2h\sigma^2}$, where $h = \max\{n, n_i, n_y\}$. Therefore with probability $\ge \frac{1}{2}$, $\|\delta W\|_2, \|\delta Y\|_2, \|\delta M\|_2 \le \sigma\sqrt{2h\ln(12h)}$.

Plugging into Lemma 3 we have with probability at least $\frac{1}{2}$,

$$\begin{aligned}
& \max_{\|x^{(t)}\| \le B, \forall t \le T} \|\hat{y}_{w+u} - \hat{y}_w\| \\
\le \quad & TB\max\{\|W\|^{T-1}, 1\}(\|Y\|\|\delta W\|\|M\|Te + \|Y\|\|\delta M\|e + \|\delta Y\|\|M\|) \\
\le \quad & TB\max\{\|W\|^{T-1}, 1\}\max\{\|Y\|, 1\}\max\{\|M\|, 1\}(\|\delta W\|Te + \|\delta M\|e + \|\delta Y\|) \\
\le \quad & TB\sqrt{2h\ln(12h)}\tilde{\beta}\phi(Te + e + 1) \\
\le \quad & \frac{\gamma}{4},
\end{aligned}$$

where we choose $\sigma = \frac{\gamma}{12\sqrt{2h\ln(12h)}TB(Te+e+1)\tilde{\beta}}$. Therefore now the perturbation u satisfies assumptions in Lemma 2.

We next compute the KL-divergence of distributions for P and u for the sake of Lemma 2.

$$\begin{aligned}
& KL(w + u \,\|\, P) \le \frac{\|w\|^2}{2\sigma^2} \\
& \le \mathcal{O}\left(\frac{B^2T^4h\ln(h)\max\{\|W\|^{2T-2}, 1\}\max\{\|M\|_2^2, 1\}\max\{\|Y\|_2^2, 1\}}{\gamma^2}(\|W\|_F^2 + \|M\|_F^2 + \|Y\|_F^2)\right)
\end{aligned}$$

□

Hence, with probability $\ge 1 - \delta$ and for all w such that, $\frac{1}{e}\beta \le \tilde{\beta} \le e\beta$, we have:

$$L_0(\hat{y}_w) \le \hat{L}_\gamma(\hat{y}_w) + \mathcal{O}(\sqrt{\frac{B(w) + \ln\frac{m}{\delta}}{m}}), \tag{27}$$

where $B(w) = \frac{B^2T^4h\ln(h)\max\{\|W\|^{2T-2}, 1\}\max\{\|M\|_2^2, 1\}\max\{\|Y\|_2^2, 1\}}{\gamma^2}(\|W\|_F^2 + \|M\|_F^2 + \|Y\|_F^2)$.

Since $\tilde{\beta}$ should be independent of the learned models. We finally take a union bound over different choices of the parameter. We will choose discrete set of $\tilde{\beta}$ such that they cover the real W, M, Y that satisfies $\frac{1}{e}\beta \le \tilde{\beta} \le e\beta$. Firstly we notice for some range of β inequality (27) holds trivially, when either term of its RHS is greater or equal to 1, since the expected margin loss is less or equal to 1.

$\hat{y}^{(T)} = Yh_w^{(T)}$, therefore if $\beta \leq \frac{\gamma}{2BT}$,

$$\begin{aligned}
\|\hat{y}\|_\infty &\leq \|\hat{y}\|_2 && \text{(by definition of } \ell_\infty \text{ norm and } \ell_2 \text{ norm)} \\
&\leq \|Y\|\|h_w^{(T)}\| && \text{(by definition of spectral norm)} \\
&\leq \|Y\|B\|M\|T\max\{\|W\|^{t-1}, 1\} \\
& && \text{(by activation bound (24))} \\
&\leq BT\beta < \frac{\gamma}{2}
\end{aligned}$$

Therefore $\hat{L}_\gamma = 1$ from definition of margin loss and the bound is satisfied trivially. Meanwhile, when $\beta \geq \frac{\gamma\sqrt{m}}{2BT}$, then the second term of (27) ≥ 1 and it also holds trivially. Therefore, we only need to consider $\tilde{\beta}$ such that $\tilde{\beta} \in [\frac{\gamma}{2BT}, \frac{\gamma\sqrt{m}}{2BT}]$. Therefore we could respectively set $\tilde{\beta}$ to be $\frac{\gamma}{2BT} + se\frac{\gamma}{2BT}, s = 0, 1, 2, \cdots$, and the size of the cover we need to consider is only $\frac{\sqrt{m}}{e}$. Therefore we replace δ by $e\frac{\delta}{\sqrt{m}}$ in (27) and take a union bound over all the $\tilde{\beta}$ on the grid to complete the proof.

B. Details of Forward and Backward Propagation Algorithms

Algorithm 1 Local forward/backward propagation

```
Input: $h^{(t-1)}, \frac{\partial L}{\partial \tilde{h}^{(t)}}, U = (u_n|...|u_{n-m_1+1})$,
$\Sigma, V = (v_n|...|v_{n-m_2+1})$
Output: $\tilde{h}^{(t)} = Wh^{(t-1)}, \frac{\partial L}{\partial U}, \frac{\partial L}{\partial V}, \frac{\partial L}{\partial \hat{\sigma}}, \frac{\partial L}{\partial h^{(t-1)}}$
// Begin forward propagation
$h_{n+1}^{(v)} \leftarrow h^{(t-1)}$
for $k = n, n-1, ..., n-m_2+1$ do
    $h_k^{(v)} \leftarrow Hprod(h_{k+1}^{(v)}, v_k)$    // Compute $\hat{V}^\top h$
end for
$h_{k_1-1}^{(u)} \leftarrow \Sigma h_{k_2}^{(v)}$    // Compute $\Sigma \hat{V}^\top h$
for $k = n-m_1+1, ..., n$ do
    $h_k^{(u)} \leftarrow Hprod(h_{k-1}^{(u)}, u_k)$    // Compute $\hat{U}\Sigma\hat{V}^\top h$
end for
$\tilde{h}^{(t)} \leftarrow h_n^{(u)}$
//Begin backward propagation
$g \leftarrow \frac{\partial L}{\partial \tilde{h}^{(t)}}$
for $k = n, n-1, ..., n-m_1+1$ do
    $g, G_{*,n-k+1}^{(u)} \leftarrow Hgrad(h_k^{(u)}, u_k, g)$    // Compute $\frac{\partial L}{\partial u_k}$
end for
$\bar{\Sigma} \leftarrow diag(g \circ h_{k_2}^{(v)}), g \leftarrow \Sigma g$    // Compute $\frac{\partial L}{\partial \Sigma}$
$g^{(\hat{\sigma})} \leftarrow \frac{\partial diag(\Sigma)}{\partial \hat{\sigma}} \circ diag(\bar{\Sigma})$    // Compute $\frac{\partial L}{\partial \hat{\sigma}}$
for $k = n-m_2+1, ..., n$ do
    $g, G_{*,n-k+1}^{(v)} \leftarrow Hgrad(h_{k+1}^{(u)}, v_k, g)$    // Compute $\frac{\partial L}{\partial v_k}$
end for
$\frac{\partial L}{\partial U} \leftarrow G^{(u)}, \frac{\partial L}{\partial V} \leftarrow G^{(v)}, \frac{\partial L}{\partial \hat{\sigma}} \leftarrow g^{(\hat{\sigma})}, \frac{\partial L}{\partial h^{(t-1)}} \leftarrow g$
```

Algorithm 2
$\hat{h} = Hprod(h, u_k)$

```
Input: $h, u_k$
Output: $\hat{h} = \mathcal{H}_k(u_k)h$
// Compute $\hat{h} = (I - \frac{2u_k u_k^\top}{u_k^\top u_k})h$
$\alpha \leftarrow \frac{2}{\|u_k\|^2} u_k^\top h$
$\hat{h} \leftarrow h - \alpha u_k$
```

Algorithm 3
$\bar{h}, \bar{u}_k = Hgrad(h, u_k, g)$

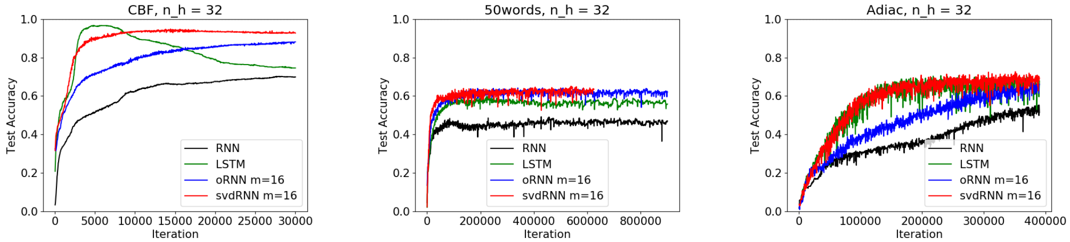
```
Input: $h, u_k, g = \frac{\partial L}{\partial \tilde{h}}$ where $\tilde{h} = \mathcal{H}_k(u_k)h$
Output: $\bar{h} = \frac{\partial L}{\partial h}, \bar{u}_k = \frac{\partial L}{\partial u_k}$
$\alpha = \frac{2}{\|u_k\|^2} u_k^\top h$
$\beta = \frac{2}{\|u_k\|^2} u_k^\top g$
$\bar{h} \leftarrow g - \beta u_k$
$\bar{u}_k \leftarrow -\alpha g - \beta h + \alpha\beta u_k$
```

C. More Experimental Details

C.1. Time Series Classification

In this experiment, we focus on the time series classification problem, where time series are fed into RNN sequentially, which then tries to predict the right class upon receiving the sequence end (Hüsken & Stagge, 2003). The dataset we choose is the largest public collection of class-labeled time-series with widely varying length, namely, the UCR time-series collection from (Chen et al., 2015). We use the training and testing sets directly from the UCR time series archive `http://www.cs.ucr.edu/~eamonn/time_series_data/`, and randomly choose 20% of the training set as validation data. We provide the statistical descriptions of the datasets and experimental results in Table 4.

In all experiments, we used hidden dimension $n_h = 32$, and chose total number of reflectors for oRNN and Spectral-RNN to be $m = 16$ (for Spectral-RNN $m_1 = m_2 = 8$). We choose proper depth t as well as input size n_i. Given sequence length L, since $tn_i = L$, we choose n_i to be the maximum divisor of L that satisfies $depth \leq \sqrt{L}$. To have a fair comparison of how the proposed principle itself influences the training procedure, we did not use dropout in any of these models. As illustrated in the optimization process in Figure 6, this resulted in some overfitting (see (a) CBF), but on the other hand it shows that Spectral-RNN is able to prevent overfitting. This supports our claim that since generalization is bounded by the spectral norm of the weights (Bartlett et al., 2017), Spectral-RNN will potentially generalize better than other schemes. This phenomenon is more drastic when the depth is large (e.g. ArrowHead(251 length) and FaceAll(131 length)), since regular RNN, and even LSTM, have no control over the spectral norms. Also note that there are substantially fewer parameters in oRNN and Spectral-RNN as compared to LSTM.



(a) (b) (c)

Figure 6. Performance comparisons of the RNN based models on three UCR datasets.

Datasets	Data Descriptions				Depth	RNN		LSTM		oRNN		Spectral-RNN	
	training/testing size		*length*	*#class*		*acc* (n_{param})		*acc* (n_{param})		*acc* (n_{param})		*acc* (n_{param})	
50words	450	455	270	50	27	0.492	(3058)	0.598	(7218)	0.642	(2426)	**0.651**	(2850)
Adiac	390	391	176	37	16	0.552	(2694)	0.706	(6950)	0.668	(2062)	**0.726**	(2486)
ArrowHead	36	175	251	3	251	0.509	(1219)	0.537	(4515)	0.669	(587)	**0.800**	(1011)
Beef	30	30	470	5	47	0.600	(1606)	0.700	(5766)	**0.733**	(974)	**0.733**	(1398)
BeetleFly	20	20	512	2	32	**0.950**	(1699)	0.850	(6435)	0.900	(1067)	**0.950**	(1491)
CBF	30	900	128	3	16	0.702	(1476)	**0.967**	(5444)	0.881	(844)	0.948	(1268)
Coffee	28	28	286	2	22	**1.000**	(1570)	**1.000**	(6018)	**1.000**	(938)	**1.000**	(1362)
Cricket X	390	390	300	12	20	0.310	(1997)	0.456	(6637)	0.495	(1365)	**0.500**	(1789)
DistalPhalanxOutlineCorrect	276	600	80	2	10	0.790	(1410)	0.798	(5378)	0.830	(778)	**0.840**	(1202)
DistalPhalanxTW	154	399	80	6	10	**0.815**	(1641)	0.795	(5609)	0.807	(1009)	**0.815**	(1433)
ECG200	100	100	96	2	12	**0.640**	(1410)	**0.640**	(5378)	**0.640**	(778)	**0.640**	(1202)
ECG5000	500	4500	140	5	14	0.941	(1606)	0.936	(5766)	0.940	(974)	**0.945**	(1398)
ECGFiveDays	23	861	136	2	17	0.947	(1443)	0.790	(5411)	**0.976**	(811)	0.948	(1235)
FaceAll	560	1690	131	14	131	0.549	(1615)	0.455	(4911)	**0.714**	(983)	**0.714**	(1407)
FaceFour	24	88	350	4	25	0.625	(1701)	0.477	(6245)	0.511	(1069)	**0.716**	(1493)
FacesUCR	200	2050	131	14	131	0.449	(1615)	0.629	(4911)	0.710	(983)	**0.727**	(1407)
Gun Point	50	150	150	2	15	0.947	(1507)	0.920	(5667)	0.953	(875)	**0.960**	(1299)
InsectWingbeatSound	220	1980	256	11	16	0.534	(1996)	0.515	(6732)	**0.598**	(1364)	0.586	(1788)
ItalyPowerDemand	67	1029	24	2	6	0.970	(1315)	0.969	(4899)	0.972	(683)	**0.973**	(1107)
Lighting2	60	61	637	2	49	**0.541**	(1570)	**0.541**	(6018)	**0.541**	(938)	**0.541**	(1362)
MiddlePhalanxOutlineCorrect	291	600	80	2	10	0.793	(1410)	0.783	(5378)	0.712	(778)	**0.820**	(1202)

Table 4. Test accuracy (number of parameters) on UCR datasets. For each dataset, we present the testing accuracy when reaching the smallest validation error. The highest precision is in bold, and lowest two are colored gray.