A Survey : Unitary Transformation in Recurrent Neural Networks

Yingru Li
yingruli@cuhk.edu.cn

February 16, 2019

Abstract

When the recurrent weight matrix in Recurrent Neural Networks (RNN) is ill-conditioned, optimization and training in RNN becomes difficult due to the well-known vanishing and exploding gradients problem, especially when trying to learn long-term dependencies. To circumvent this issue, most recently, several works have been proposed in top AI conferences, mainly exploiting norm-preserving property of unitary transformation.

Majority of the methods enforce either hard or soft unitary constraints on recurrent weight matrices by matrix decomposition and parametrization or by optimization on stiefel manifold. In these methods, there are always trade-offs between computational efficiency problem and the matrix searching space. Therefore, how to avoid over parametrization and how to find a simple but efficient representation of unitary matrix are also big problems in this line of works. In addition to making restriction on recurrent matrix of internal hidden state, there are other works embedding orthogonality in external associative memory which provide alternative solutions.

In this survey, we first introduce the preliminaries of recurrent neural networks and motivations of applying unitary transformation in recurrent neural networks. We then describe the proposed novel methods in these works and also the mathematical or physical foundations in details, as well as providing comparison among these works. Potential research directions will be discussed in the end.

Contents

1 Introduction

Deep neural networks have defined the state-of-the-art in a wide range of problems in computer vision, speech analysis, and natural language processing (Krizhevsky et al., 2012; Hinton et al., 2012; Tomáš, 2012). However, these models suffer from two key issues. First, they are over-parametrized; thus it takes a very long time for training and inference. Second, learning deep models is difficult because of the poor conditioning of the matrices that parametrize the model. These difficulties are especially problematic to recurrent neural networks. Indeed, the number of distinct parameters in RNNs grows as the square of the size of the hidden state conversely to convolutional networks which enjoy weight sharing. Moreover, poor conditioning of the recurrent matrices results in the gradients to explode or vanish exponentially fast along the time horizon. This problem prevents RNN from capturing long-term dependencies (Hochreiter & Schmidhuber, 1997; Bengio et al., 1994).

There exists an extensive body of literature addressing over-parametrization in neural networks. Le Cun et al. first studied the problem and proposed to remove unimportant weights in neural networks by exploiting the second order information. Several techniques which followed include low-rank decomposition (Denil et al., 2013), training a small network on the soft-targets predicted by a big pre-trained network (Ba & Caruana, 2014), low bit precision training (Courbariaux et al., 2014), hashing (Chen et al., 2015), etc. These techniques are primarily aimed at feed-forward fully connected networks and very few studies have focused on the particular case of recurrent networks (Arjovsky et al., 2016).

The problem of vanishing and exploding gradients has also received significant attention. Hochreiter & Schmidhuber (1997) proposed an effective gating mechanism in their seminal work on LSTMs. Later, this technique was adopted by other models such as the Gated Recurrent Units (GRU) (Chung et al., 2015) and the Highway networks (Srivastava et al., 2015) for recurrent and feed-forward neural networks respectively. Other popular strategies include gradient clipping (Pascanu et al., 2013), and orthogonal initialization of the recurrent weights (Le et al., 2015). More recently Arjovsky et al. (2016) proposed to use a unitary recurrent weight matrix. The use of norm preserving unitary maps prevent the gradients from exploding or vanishing, and thus help to capture long-term dependencies. The resulting model called unitary RNN (uRNN) is computationally efficient since it only explores a small subset of general unitary matrices. Unfortunately, since uRNNs can only span a reduced subset of unitary matrices their expressive power is limited (Wisdom et al., 2016). We denote this restricted capacity unitary RNN as RC uRNN. Full capacity unitary RNN (FC uRNN) (Wisdom et al., 2016) proposed to overcome this issue by parametrizing the recurrent matrix with a full dimensional unitary matrix, hence sacrificing computational efficiency. Indeed, FC uRNN requires a computationally expensive projection step which takes $\mathcal{O}(N^3)$ time (N being the size of the hidden state) at each step of the stochastic optimization to maintain the unitary constraint on the recurrent matrix. Mhammedi et al. (2016) in their

orthogonal RNN (oRNN) avoided the expensive projection step in FC uRNN by parametrizing the orthogonal matrices using Householder reflection vectors, it allows a fine-grained control over the number of parameters by choosing the number of Householder reflection vectors. When the number of Householder reflection vector approaches N this parametrization spans the full reflection set, which is one of the disconnected subset of the full unitary set. Jing et al. (2016) also presented a way of parametrizing unitary matrices which allows fine-grained control on the number of parameters. This work called as Efficient Unitary RNN (EURNN), exploits the continuity of unitary set to have a tunable parametrization ranging from a subset to the full unitary set. Jose et al. (2017) present a new model called Kronecker Recurrent Units (KRU), using Kronecker factored recurrent matrix which provide an elegant way to adjust the number of parameters to the problem at hand. This factorization allows finely modulating the number of parameters required to encode $N \times N$ matrices, from $\mathcal{O}(\log(N))$ when using factors of size 2×2, to $\mathcal{O}(N^2)$ parameters when using a single factor of the size of the matrix itself.

Although the idea of parametrizing recurrent weight matrices with strict unitary linear operator is appealing, it suffers from several issues: (1) Strict unitary constraints severely restrict the search space of the model, thus making the learning process unstable. (2) Strict unitary constraints make forgetting irrelevant information difficult. While this may not be an issue for problems with non-vanishing long term influence, it causes failure when dealing with real world problems that have vanishing long term influence. Henaff et al. (2016) have previously pointed out that the good performance of strict unitary models on certain synthetic problems is because it exploits the biases in these data-sets which favours a unitary recurrent map and these models may not generalize well to real world data-sets. More recently Vorontsov et al. (2017) have also studied this problem of unitary RNNs and the authors found out that relaxing the strict unitary constraint on the recurrent matrix to a soft unitary constraint improved the convergence speed as well as the generalization performance. Jose et al. (2017) tackle the vanishing and exploding gradient problem through a efficient soft unitary constraint (Jose & Fleuret, 2016; Henaff et al., 2016; Cisse et al., 2017; Vorontsov et al., 2017) by exploiting the properties of Kronecker matrices (Van Loan, 2000).

Besides, enforcing orthogonality in external associative memory (Jing et al., 2017) and constructing novel memory unit (Dangovski et al., 2017) to guarantee the norm-preserving property on hidden state get advantages from both associative unit and unitary transformation, which are indeed beneficial to resolve exploding and vanishing gradients problems. Dangovski et al. (2017) invent a novel memory unit called Rotational Unit of Memory (RUM) which has the advantages of both orthogonality and differentiability, avoiding the computational difficulty of optimization on parametrized matrices.

The survey is organized as follows, in section 2 we restate the formalism of RNN and detail the core problems and motivations for using Unitary(Orthogonal) matrix constraint. In section 3 we present different matrix parameterization ways in most recent works in this line of research. In section 4 we present

the issue of hard and soft unitary constraints. In section 5 we describe recent explorations on orthogonality in external RNN memory. In the end, we give conclusions and interesting future directions.

Table 1: Notations

D, N, M	Input, hidden and output dimensions;
$\mathbf{x}_t \in \mathbb{R}^D$ or $\mathbb{C}^D$, $\mathbf{h}_t \in \mathbb{C}^D$	Input and hidden state at time t;
$\mathbf{y}_t \in \mathbb{R}^M$ or $\mathbb{C}^M$, $\hat{\mathbf{y}}_t \in \mathbb{R}^M$ or $\mathbb{C}^M$	Prediction targets and RNN predictions at time t;
$\mathbf{U} \in \mathbb{C}^{N\times D}, \mathbf{W} \in \mathbb{C}^{N\times N}, \mathbf{V} \in \mathbb{C}^{M\times N}$	Input, recurrent and output weight matrices;
$\mathbf{b} \in \mathbb{R}^N$ or $\mathbb{C}^N, \mathbf{c} \in \mathbb{R}^M$ or $\mathbb{C}^M$	Hidden and output bias;
$\sigma(.), \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$	Non-linear activation function and the loss function.

2 Preliminaries and problems formalism

Table 1 summarizes some notations that we use in the paper. We consider the field to be complex rather than real numbers. We will reclaim the motivation of the choice of complex numbers later in this section. Consider a standard recurrent neural network (Elman, 1990). Given a sequence of T input vectors: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$, at a time step t RNN performs the following:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \tag{1}$$

$$\hat{\mathbf{y}}_t = \mathbf{V}\mathbf{h}_t + \mathbf{c}, \tag{2}$$

where $\hat{\mathbf{y}}_t$ is the predicted value at time step t.

2.1 Over parametrization and computational efficiency

The total number of parameters in a RNN is $c(DN + N^2 + N + M + MN)$, where c is 1 for real and 2 for complex parametrizations. As we can see, the number of parameters grows quadratically with the hidden dimension, *i.e.*, $\mathcal{O}(N^2)$. Moreover, it has a direct impact on the computational efficiency of RNNs because the evaluation of $\mathbf{W}\mathbf{h}_{t-1}$ takes $\mathcal{O}(N^2)$ time and it recursively depends on previous hidden states. However, other components $\mathbf{U}\mathbf{x}_t$ and $\mathbf{V}\mathbf{h}_t$ can usually be computed efficiently by a single matrix-matrix multiplication for each of the components. That is, $\mathbf{U}[\mathbf{x}_0, \ldots, \mathbf{x}_T]$ and $\mathbf{V}[\mathbf{h}_0, \ldots, \mathbf{h}_{T-1}]$ can be efficiently computing using modern BLAS libraries. So to summarize, if we can control the number of parameters in the recurrent matrix $\mathbf{W}$, then we can control the computational efficiency.

2.2 Poor conditioning implies gradients exploding or vanishing

The vanishing and exploding gradient problem refers to the decay or growth of the partial derivative of the loss $\mathcal{L}(.)$ with respect to the hidden state $\mathbf{h}_t$ *i.e.* $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$

as the number of time steps T grows (Arjovsky et al., 2016). By the application of the chain rule, the following can be shown:

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\| \leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \|\mathbf{W}\|^{T-t}. \tag{3}$$

From eq. (3), it is clear that if the absolute value of the eigenvalues of $\mathbf{W}$ deviates from 1 then $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ may explode or vanish exponentially fast with respect to $T-t$. So a strategy to prevent vanishing and exploding gradient is to control the spectrum of $\mathbf{W}$.

2.2.1 Analysis of vanishing and exploding gradients in RNN

Given a sequence of T input vectors: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$, let us consider the operation at the hidden layer t of a recurrent neural network:

$$\mathbf{z}_t = \mathbf{W}_t \mathbf{h}_{t-1} + \mathbf{U}_t \mathbf{x}_t + \mathbf{b} \tag{4}$$

$$\mathbf{h}_t = \sigma(\mathbf{z}_t) \tag{5}$$

By the chain rule,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \tag{6}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=T-1}^{t} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{h}_k} = \tag{7}$$

$$= \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=T-1}^{t} \frac{\partial \mathbf{h}_{k+1}}{\partial \mathbf{z}_{k+1}} \frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{h}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=T-1}^{t} \mathbf{J}_{k+1} \mathbf{W} \tag{8}$$

where σ is the non-linear activation function and $\mathbf{J}_{k+1} = diag(\sigma^{'}(\mathbf{z}_{k+1}))$ is the Jacobian matrix of the non-linear activation function.

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\| = \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \prod_{k=T-1}^{t} \mathbf{J}_{k+1} \mathbf{W} \right\| \tag{9}$$

$$\leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \prod_{k=t}^{T-1} \|\mathbf{J}_{k+1} \mathbf{W}\| \tag{10}$$

$$\leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \|\mathbf{W}\|^{T-t} \prod_{k=t}^{T-1} \|\mathbf{J}_{k+1}\| \tag{11}$$

From eq. (11) it is clear the norm of the gradient is exponentially dependent upon two factors along the time horizon:

- The norm of the Jacobian matrix of the non-linear activation function $\|\mathbf{J}_{k+1}\|$.

- The norm of the hidden to hidden weight matrix $\|\mathbf{W}\|$.

These two factors are causing the vanishing and exploding gradient problem.

Since the gradient of the standard non-linear activation functions such as tanh and ReLU are bounded between [0, 1], $\|\mathbf{J}_{k+1}\|$ does not contribute to the exploding gradient problem but it can still cause vanishing gradient problem.

For the special ReLU case, the norm $\|\mathbf{J_{k+1}}\| = \max_{j=1,\ldots,n} |\sigma^{'}(\mathbf{z}_{k+1})|$. Therefore, unless all the activations are killed at one layer, the maximum entry of $\mathbf{J_{k+1}}$ is 1, resulting in $\|\mathbf{J_{k+1}}\| = 1$ for all layer $k+1$. With ReLU nonlinearities, we thus have

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\| \leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \|\mathbf{W}\|^{T-t} \tag{12}$$

As we know the spectral norm is the largest singular value of matrix, if the largest singular value of the recurrent matrix $\mathbf{W}$ is much less than 1, they will cause gradient vanishing, $\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\| \to 0$ as $T - t$ grows up; while if the spectral norm of $\mathbf{W}$ is much larger than 1, the RHS of the bound is going to infinity, which is meaningless and can cause gradient explosion.

2.3 Unitary approach to solve gradient explosion or vanishing problem

A matrix $\mathbf{W}$ is orthogonal if

$$\mathbf{W}^T\mathbf{W} = \mathbf{W}\mathbf{W}^T = \mathbf{I}.$$

Orthogonal matrices have the property that they preserve norm, i.e. $\|\mathbf{Wh}\|_2 = \|\mathbf{h}\|_2$. Therefore, repeated iterative multiplication of a vector by orthogonal matrix leaves the norm of the vector unchanged.

Unitary matrices generalize orthogonal matrices to complex domain. A complex valued, norm preserving matrix, $\mathbf{U}$, is called *unitary* matrix, and is such that

$$\mathbf{U}^H\mathbf{U} = \mathbf{U}\mathbf{U}^H = \mathbf{I},$$

where $\mathbf{U}^H$ is the conjugate transpose of $\mathbf{U}$.

If the matrix $\mathbf{W}$ in eq. (12) is norm preserving (i.e. orthogonal or unitary), than we can rewrite eq. (12) as

$$\left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_t} \right\| \leq \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \|\mathbf{W}\|^{T-t} = \left\| \frac{\partial \mathcal{L}}{\partial \mathbf{h}_T} \right\| \tag{13}$$

Most notably, this result holds for a network of arbitrary depth and thus renders engineering tricks like gradient clipping unnecessary. This mathematical analysis was first given in Arjovsky et al. (2016).

Directly parameterizing the set of unitary matrices in such a way that gradient-based optimization can be applied is not straightforward because a

gradient step will typically yield a matrix that is not unitary, and projecting on the set of unitary matrices (e.g., by performing an eigendecomposition) generally costs $\mathcal{O}(N^3)$ computation when $\mathbf{U}$ is $N \times N$.

The most important feature of unitary and orthogonal matrices for our purpose is that they have eigenvalues λ_j with absolute value 1. The following lemma, proved in Hoffman & Kunze (1971), may shed light on a method which can be used to efficiently span a large set of unitary matrices.

Lemma 1. *A complex square matrix* $\mathbf{W}$ *is unitary if and only if it has an eigendecomposition of the form* $\mathbf{W} = \mathbf{V}\mathbf{D}\mathbf{V}^H$*, where* ^{H} *denotes the conjugate transpose. Here,* $\mathbf{V}, \mathbf{D} \in \mathbb{C}^{N \times N}$ *are complex matrices, where* $\mathbf{V}$ *is unitary, and* $\mathbf{D}$ *is a diagonal such that* $|\mathbf{D}_{j,j}| = 1$*. Furthermore,* $\mathbf{W}$ *is a real orthogonal matrix if and only if for every eigenvalue* $\mathbf{D}_{j,j} = \lambda_j$ *with eigenvector* v_j*, there is also a complex conjugate eigenvalue* $\lambda_k = \overline{\lambda_j}$ *with corresponding eigenvector* $v_k = \overline{v_j}$ *.*

Proof. For any eigenvalue λ_j of a unitary matrix $\mathbf{W}$ and its corresponding eigenvector v_j,

$$\mathbf{W}v_j = \lambda_j v_j. \tag{14}$$

As mentioned above, unitary matrix is able to preseve norm, i.e. $\|\mathbf{W}v_j\| = \|v_j\|$, and we have $|\mathbf{D}_{j,j}| = |\lambda_j| = 1$.

If $\mathbf{W}$ is a real orthogonal matrix, take conjugate in both side of the equation and we have

$$\overline{\mathbf{W}v_j} = \overline{\lambda_j v_j} \tag{15}$$

$$\mathbf{W}\overline{v_j} = \overline{\lambda_j}\overline{v_j}. \tag{16}$$

□

Writing $\lambda_j = e^{iw_j}$ with $w_j \in \mathbb{R}$, a naive method to learn a unitary matrix would be to fix a basis of eigenvectors $\mathbf{V} \in \mathbb{C}^{N \times N}$ and set

$$\mathbf{W} = \mathbf{V}\mathbf{D}\mathbf{V}^H, \tag{17}$$

where $\mathbf{D}$ is a diagonal such that $\mathbf{D}_{j,j} = \lambda_j$.

Lemma 1 informs us how to construct a real orthogonal matrix, $\mathbf{W}$. We must (i) ensure the columns of $\mathbf{V}$ come in complex conjugate pairs, $v_k = \overline{v_j}$, and (ii) tie weights $w_k = -w_j$ in order to achieve $e^{iw_j} = \overline{e^{iw_k}}$. Most neural network objective functions are differentiable with respect to the weight matrices, and consequently w_j may be learned by gradient descent.

Unfortunately the above approach has undesirable properties. Fixing $\mathbf{V}$ and learning w requires $\mathcal{O}\left(N^2\right)$ memory, which is unacceptable given that the number of learned parameters is $\mathcal{O}(N)$. Further note that calculating $\mathbf{V}u$ for an arbitrary vector u requires $\mathcal{O}(N^2)$ computation. Setting $\mathbf{V}$ to the identity would satisfy the conditions of the lemma, whilst reducing memory and computation requirements to $\mathcal{O}(N)$, however, $\mathbf{W}$ would remain diagonal, and have poor representation capacity.

2.4 Representation power of structure unitary matrices

In this section, we reclaim a theorem in (Wisdom et al., 2016) that can be used to determine when any particular unitary parametrization does not have capacity to represent all unitary matrices. Wisdom et al. (2016) first establish an upper bound on the number of real-valued parameters required to represent any $N \times N$ unitary matrix.

Lemma 2. *The set of all unitary matrices is a manifold of dimension N^2.*

Proof. The set of all unitary matrices is the well-known unitary Lie group $\mathbf{U}(N)$ (Gilmore, 2008, §3.4). A Lie group identifies group elements with points on a differentiable manifold (Gilmore, 2008, §2.2). The dimension of the manifold is equal to the dimension of the Lie algebra $\mathfrak{u}$, which is a vector space that is the tangent space at the identity element (Gilmore, 2008, §4.5). For $U(N)$, the Lie algebra consists of all skew-Hermitian matrices $\mathbf{A}$ (Gilmore, 2008, §5.4). A skew-Hermitian matrix is any $\mathbf{A} \in \mathbb{C}^{N \times N}$ such that $\mathbf{A} = -\mathbf{A}^H$, where $(\cdot)^H$ is the conjugate transpose. To determine the dimension of $\mathbf{U}(N)$, we can determine the dimension of $\mathfrak{u}$. Because of the skew-Hermitian constraint, the diagonal elements of $\mathbf{A}$ are purely imaginary, which corresponds to N real-valued parameters. Also, since $A_{i,j} = -A^*_{j,i}$, the upper and lower triangular parts of $\mathbf{A}$ are parameterized by $\frac{N(N-1)}{2}$ complex numbers, which corresponds to an additional $N^2 - N$ real parameters. Thus, $\mathbf{U}(N)$ is a manifold of dimension N^2. □

Theorem 1 (Limit of representation power (Wisdom et al., 2016))**.** *If a family of $N \times N$ unitary matrices is parameterized by P real-valued parameters for $P < N^2$, then it cannot contain all $N \times N$ unitary matrices.*

Proof. We consider a family of unitary matrices that is parameterized by P real-valued parameters through a smooth map $g : \mathcal{P}(P) \to \mathcal{U}(N^2)$ from the space of parameters $\mathcal{P}(P)$ to the space of all unitary matrices $\mathcal{U}(N^2)$. The space $\mathcal{P}(P)$ of parameters is considered as a P-dimensional manifold, while the space $\mathcal{U}(N^2)$ of all unitary matrices is an N^2-dimensional manifold according to lemma 2. Then, if $P < N^2$, Sard's theorem Sard (1942) implies that the image $g(\mathcal{P})$ of g is of measure zero in $\mathcal{U}(N^2)$, and in particular g is not onto. Since g is not onto, there must exist a unitary matrix $\mathbf{W} \in \mathcal{U}(N^2)$ for which there is no corresponding input $\mathbf{P} \in \mathcal{P}(P)$ such that $\mathbf{W} = g(\mathbf{P})$. Thus, if P is such that $P < N^2$, the manifold $\mathcal{P}(P)$ cannot represent all unitary matrices in $\mathcal{U}(N^2)$. □

2.5 Why complex field?

Although Arjovsky et al. (2016) and Wisdom et al. (2016) use complex valued networks with unitary constraints on the recurrent matrix, the motivations for such models are not clear. There is a simple but compelling reason for complex-valued recurrent networks.

The absolute value of the determinant of a unitary matrix is 1. Hence in the real space, the set of all unitary (orthogonal) matrices have a determinant of 1

or -1, *i.e.*, the set of all rotations and reflections respectively. Since the determinant is a continuous function, the unitary set in real space is disconnected. Consequently, with the real-valued networks we cannot span the full unitary set using the standard continuous optimization procedures. On the contrary, the unitary set is connected in the complex space as its determinants are the points on the unit circle and we do not have this issue.

As we mentioned in the introduction, Jing et al. (2016) uses this continuity of unitary space to have a tunable continuous parametrization ranging from subspace to full unitary space. Any continuous parametrization in real space can only span a subset of the full unitary set. For example, the Householder parametrization (Mhammedi et al., 2016) suffers from this issue.

2.6 Is strict unitary constraint always good?

Although the idea of parameterizing recurrent weight matrices with strict unitary linear operator is appealing, it suffers from several issues: (1) Strict unitary constraints severely restrict the search space of the model, thus making the learning process unstable. (2) Strict unitary constraints make forgetting irrelevant information difficult. While this may not be an issue for problems with non-vanishing long term influence, it causes failure when dealing with real world problems that have vanishing long term influence. Henaff et al. (2016) have previously pointed out that the good performance of strict unitary models on certain synthetic problems is because it exploits the biases in these data-sets which favours a unitary recurrent map and these models may not generalize well to real world data-sets. More recently Vorontsov et al. (2017) have also studied this problem of unitary RNNs and the authors found out that relaxing the strict unitary constraint on the recurrent matrix to a soft unitary constraint improved the convergence speed as well as the generalization performance. The latest work Jose et al. (2017) enforce soft unitary constraint on each factor of the Kronecker factored recurrent matrix, which is computational efficient and proved to maintain the approximately unitary property of the final product of all factors. This topic will be discussed in section 4 in details.

Methods	Constraint on the recurrent matrix	Time complexity of one online gradient step	Number of parameters in the recurrent matrix	Search space of the recurrent matrix
RC uRNN Arjovsky et al. (2016)	$\|W\| = 1$	$\mathcal{O}(TN\log(N))$	$\mathcal{O}(N)$	subspace of $\mathbf{U}(N)$ when $N > 7$
FC uRNN Wisdom et al. (2016)	$\|W\| = 1$	$\mathcal{O}(TN^2 + N^3)$	$\mathcal{O}(N^2)$	full space of $\mathbf{U}(N)$
Unitary RNN Hyland & Rätsch (2017)	$\|W\| = 1$	$\mathcal{O}(TN^2 + N^3)$	$\mathcal{O}(N^2)$	full space of $\mathbf{U}(N)$
oRNN (Mhammedi et al., 2016)	$\|W\| = 1$	$\mathcal{O}(TNK)$ where $K \leq N$	$\mathcal{O}(NK)$	tunable space of $\mathbf{O}(N)$ full space when $K = N$
EURNN(FFT) (Jing et al., 2016)	$\|W\| = 1$	$\mathcal{O}(TN\log N)$	$\mathcal{O}(N\log N)$	subspace of $\mathbf{U}(N)$
EURNN(Tunable) (Jing et al., 2016)	$\|W\| = 1$	$\mathcal{O}(TNL)$ where $L \leq N$	$\mathcal{O}(NL)$	tunable space of $\mathbf{U}(N)$ full space when $L = N$
soft oRNN Vorontsov et al. (2017)	$\forall\sigma_i \in \{\text{diag}(\mathbf{\Sigma})\}$ $\sigma_i \in [1-m, 1+m]$	$\mathcal{O}(TN^2 + N^3)$	$\mathcal{O}(N^2)$	approximate full space of $\mathbf{O}(N)$
KRU (Jose et al., 2017)	$\forall f \in \{0, \ldots, F-1\}$ $\min_{\mathbf{W}_f} \left\|\mathbf{W}_f^H\mathbf{W}_f - \mathbf{I}\right\|^2$	$\mathcal{O}(TN\log N)$	$\mathcal{O}(\log N)$	approximate subspace of $\mathbf{U}(N)$

Table 2: Table showing the time complexities associated with one stochastic gradient step (mini-batch size =1) for different methods, when the size of the hidden layer of an RNN is N and the input sequence has length T. For the oRNN method, K denotes how many householder reflections it uses in parametrization. For the tunable-style EURNN, L is an integer between 1 and N parameterizing the unitary matrix capacity. For the soft oRNN, $\mathbf{\Sigma}$ is the singular matrix corresponding to the SVD of recurrent matrix $\mathbf{W}$. For the KRU method, $\mathbf{W}_f$ is a kronecker factor matrix. $\mathbf{U}(N)$ and $\mathbf{O}(N)$ denote the topological space of unitary matrix and orthogonal matrix of size $N \times N$ repectively.

3 Matrix parametrization and search spaces

3.1 Unitary evolution RNN (Arjovsky et al., 2016)

As mentioned in section 2.3, directly parameterizing the set of unitary matrices using the property of eigen-decomposition is both computational and memory expensive. Unitary evolution RNN (uRNN) proposed to solve the vanishing and exploding gradients through a unitary recurrent matrix in a efficient way, which is for the form:

$$\mathbf{W} = \mathbf{D}_3\mathbf{R}_2\mathscr{F}^{-1}\mathbf{D}_2\Pi\mathbf{R}_1\mathscr{F}\mathbf{D}_1. \tag{18}$$

Where:

- $\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3$: Diagonal matrices whose diagonal entries are of the from $\mathbf{D}_{kk} = e^{i\theta_k}$, implies each matrix have N parameters, $(\theta_0, \ldots, \theta_{N-1})$. They all permit $\mathcal{O}(N)$ storage and $\mathcal{O}(N)$ computation.
- $\mathscr{F}$ and $\mathscr{F}^{-1}$: Fast Fourier operator and inverse fast Fourier operator respectively. They require no storage and $\mathcal{O}(N \log N)$ matrix vector multiplication using the Fast Fourier Transform algorithm.
- $\mathbf{R}_1, \mathbf{R}_2$: Householder reflections. $R = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^H}{\|\mathbf{v}\|^2}$, where $v \in \mathbb{C}^N$. They need $\mathcal{O}(N)$ storage and $\mathcal{O}(N)$ computation for matrix vector products.

The total number of parameters for this uRNN operator is $7N$ and the matrix vector can be done $Nlog(N)$ time. It is parameter efficient and fast but not flexible and suffers from the retention of noise and difficulty in optimization due its unitarity.

3.2 Full capacity unitary RNN (Wisdom et al., 2016)

According to Theorem 1, note that the parameterization (3.1) has $P = 7N$ real-valued parameters. If we solve for N in $7N < N^2$, we get $N > 7$. Thus, the parameterization (3.1) cannot represent all unitary matrices for dimension $N > 7$.

Full capacity unitary RNN (FC uRNN) does optimization on the full unitary set instead on a subset like uRNN. That is FC uRNN's recurrent matrix $\mathbf{W} \in U(N)$. There are several challenges in optimization over unitary manifold especially when combined with stochastic gradient method. The primary challenge being the optimization cost is $\mathcal{O}(N^3)$ per step.

3.2.1 Optimizing full-capacity unitary matrices on the Stiefel manifold

The limitations of restricted-capacity parameterizations could be solved directly by optimizing a full-capacity unitary matrix. Consider the Stiefel manifold of

all $N \times N$ complex-valued matrices whose columns are N orthonormal vectors in $\mathbb{C}^N$ (Tagare, 2011). Mathematically, the Stiefel manifold is defined as

$$\mathcal{V}_N(\mathbb{C}^N) = \left\{ \mathbf{W} \in \mathbb{C}^{N \times N} : \mathbf{W}^H \mathbf{W} = \mathbf{I}_N \right\}. \tag{19}$$

For any $\mathbf{W} \in \mathcal{V}_N(\mathbb{C}^N)$, any matrix $\mathbf{Z}$ in the tangent space $\mathcal{T}_{\mathbf{W}}\mathcal{V}_N(\mathbb{C}^N)$ of the Stiefel manifold satisfies $\mathbf{Z}^H\mathbf{W} + \mathbf{W}^H\mathbf{Z} = 0$ (Tagare, 2011). The Stiefel manifold becomes a Riemannian manifold when its tangent space is equipped with an inner product. Tagare (Tagare, 2011) suggests using the canonical inner product, given by

$$\langle \mathbf{Z}_1, \mathbf{Z}_2 \rangle_c = \mathrm{tr}\left(\mathbf{Z}_1^H (\mathbf{I} - \frac{1}{2}\mathbf{W}\mathbf{W}^H)\mathbf{Z}_2 \right). \tag{20}$$

Under this canonical inner product on the tangent space, the gradient in the Stiefel manifold of the loss function f with respect to the matrix $\mathbf{W}$ is $\mathbf{AW}$, where $\mathbf{A} = \mathbf{G}^H\mathbf{W} - \mathbf{W}^H\mathbf{G}$ is a skew-Hermitian matrix and $\mathbf{G}$ with $G_{i,j} = \frac{\partial f}{\partial W_{i,j}}$ is the usual gradient of the loss function f with respect to the matrix $\mathbf{W}$ (Tagare, 2011). Using these facts, Tagare (Tagare, 2011) suggests a descent curve along the Stiefel manifold at training iteration k given by the matrix product of the Cayley transformation of $\mathbf{A}^{(k)}$ with the current solution $\mathbf{W}^{(k)}$:

$$\mathbf{Y}^{(k)}(\lambda) = \left(\mathbf{I} + \frac{\lambda}{2}\mathbf{A}^{(k)} \right)^{-1} \left(\mathbf{I} - \frac{\lambda}{2}\mathbf{A}^{(k)} \right) \mathbf{W}^{(k)}, \tag{21}$$

where λ is a learning rate and $\mathbf{A}^{(k)} = \mathbf{G}^{(k)^H}\mathbf{W}^{(k)} - \mathbf{W}^{(k)^H}\mathbf{G}^{(k)}$. Gradient descent proceeds by performing updates $\mathbf{W}^{(k+1)} = \mathbf{Y}^{(k)}(\lambda)$. Tagare (Tagare, 2011) suggests an Armijo-Wolfe search along the curve to adapt λ, but such a procedure would be expensive for neural network optimization since it requires multiple evaluations of the forward model and gradients. Wisdom et al. (2016) found that simply using a fixed learning rate λ often works well. Also, RMSprop-style scaling of the gradient $\mathbf{G}^{(k)}$ by a running average of the previous gradients' norms (Tieleman & Hinton, 2012) before applying the multiplicative step eq. (21) can improve convergence. The only additional substantial computation required beyond the forward and backward passes of the network is the $N \times N$ matrix inverse in eq. (21).

3.3 Orthogonal RNN (Mhammedi et al., 2016)

Orthogonal RNN (oRNN) parametrizes the recurrent matrices using Householder reflections.

$$\mathbf{W} = \mathcal{H}_N(\mathbf{v}_N) ... \mathcal{H}_{N-K+1}(\mathbf{v}_{N-K+1}). \tag{22}$$

where

$$\mathcal{H}_K(\mathbf{v}_K) = \begin{pmatrix} \mathbf{I}_{N-K} & 0 \\ 0 & \mathbf{I}_K - 2\frac{\mathbf{v}_K \mathbf{v}_K^H}{\|\mathbf{v}_K\|^2} \end{pmatrix} \tag{23}$$

and

$$\mathcal{H}_1(\mathbf{v}) = \begin{pmatrix} \mathbf{I}_{N-1} & 0 \\ 0 & \mathbf{v} \in \{-1, 1\} \end{pmatrix} \tag{24}$$

where $\mathbf{v}_K \in \mathbb{R}^K$. The number of parameters in this parameterization is $\mathcal{O}(NK)$. When $N = K = 1$ and $v = 1$, it spans the rotation subset and when $v = -1$, it spans the full reflection subset.

The Householder transformation was shown to have a one-to-one relationship with the canonical coset decomposition of unitary matrices defined in group theory, which can be used to parametrize unitary operators in a very efficient manner.

This is appealing intuitively since multiplication of a vector by an orthogonal matrix preserves the length of that vector, and rotations and reflections exhaust the set of (real-valued) geometric operations that render invariant a vector's length.

3.4 Tunable efficient unitary RNN (EURNN) (Jing et al., 2016)

As mentioned in section 2.5, the determinant of real-valued rotation and reflection matrix are $+1$ and -1. Because the determinant is a continuous function, the unitary set in real space is disconnected. On the contrary, the unitary set is connected in the complex space as its determinants are the points on the unit circle. Therefore, instead of using real-valued rotation and reflection matrix, Jing et al. (2016) use the complex-valued rotation matrix which represent the basic operation in quantum and optical physics.

3.4.1 Fundamental principle from Optics

The beam splitter has played numerous roles in many aspects of optics. For example, in quantum information the beam splitter plays essential roles in teleportation, bell measurements, entanglement and in fundamental studies of the photon.

The lossless assumption of beam splitter lead to its unitary transformation behavior. Roughly speaking, the operation on photon in a 50/50 beam splitter can be represented as a unitary matrix:

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \tag{25}$$

The most general element of $U(2)$ can transfrom the input state with modes (k_1, k_2) into output state with modes (k_1', k_2'),

$$(k_1', k_2') = (k_1, k_2) \begin{pmatrix} e^{i\phi} \sin\theta & e^{i\phi} \cos\theta \\ \cos\theta & -\sin\theta \end{pmatrix} \tag{26}$$

This element can perform any transformation in $U(2)$ by choosing different parameters (ϕ, θ). And this element can be realized by two 50/50 beam splitters,

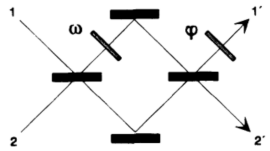

Figure 1: Experimental realization of a Mach-Zehnder interferometer using two 50/50 beam splitters, two mirrors and two phase shifters. (Picture from (Reck et al., 1994). Set ω here to 2θ.)

two mirrors and two phase shifters as shown in fig. 1,

$$\begin{pmatrix} e^{i\phi} & 0 \\ 0 & 1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \begin{pmatrix} e^{i2\theta} & 0 \\ 0 & 1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \tag{27}$$

$$= \begin{pmatrix} e^{i\phi}(e^{i\theta} - e^{-i\theta})/2 & ie^{i\phi}(e^{i\theta} + e^{-i\theta})/2 \\ i(e^{i\theta} + e^{-i\theta})/2 & -(e^{i\theta} - e^{-i\theta})/2 \end{pmatrix} \cdot e^{i\theta} \tag{28}$$

$$= \begin{pmatrix} e^{i\phi}\sin\theta & e^{i\phi}\cos\theta \\ \cos\theta & -\sin\theta \end{pmatrix} \cdot e^{i(\theta+\frac{\pi}{2})} \tag{29}$$

While the global phase cannot be observed, the experimental implementation of the general element in $U(2)$ is completed.

An arbitrary state can be represented as

$$\cos\theta \langle 0| + e^{i(\phi+\pi)} \sin\theta \langle 1| = (\cos\theta, -e^{i\phi}\sin\theta)$$

and it can be transformed to a state with single mode

$$e^{i\phi} \langle 1| = (0, e^{i\phi}) = (\cos\theta, -e^{i\phi}\sin\theta) \begin{pmatrix} e^{i\phi}\sin\theta & e^{i\phi}\cos\theta \\ \cos\theta & -\sin\theta \end{pmatrix} \tag{30}$$

Based on this observation, several efficient unitary matrix construction methods were proposed in physics society (Reck et al., 1994; Clements et al., 2016; Jing et al., 2016).

3.4.2 Unitary matrix parametrization

Any $N \times N$ unitary matrix $\mathbf{W}_N$ can be represented as a product of rotation matrices $\{\mathbf{R}_{i,j}\}$ and a diagonal matrix $\mathbf{D}$, such that $\mathbf{W}_N = \mathbf{D} \prod_{i=2}^{N} \prod_{j=1}^{i-1} \mathbf{R}_{i,j}$, where $\mathbf{R}_{i,j}$ is defined as the N-dimensional identity matrix with the elements R_{ii}, R_{ij}, R_{ji} and R_{jj} replaced as follows (Reck et al., 1994; Clements et al., 2016; Jing et al., 2016):

$$\begin{pmatrix} R_{ii} & R_{ij} \\ R_{ji} & R_{jj} \end{pmatrix} = \begin{pmatrix} e^{i\phi_{ij}}\sin\theta_{ij} & e^{i\phi_{ij}}\cos\theta_{ij} \\ \cos\theta_{ij} & -\sin\theta_{ij} \end{pmatrix}. \tag{31}$$

where θ_{ij} and ϕ_{ij} are unique parameters corresponding to $\mathbf{R}_{i,j}$. Each of these matrices is the most general element in $U(2)$, performing a $U(2)$ unitary transformation on a two-dimensional subspace of the N-dimensional Hilbert space and

leaving an $(N-2)$-dimensional subspace unchanged. In other words, a series of $U(2)$ rotations can be used to successively make all off-diagonal elements of the given $N \times N$ unitary matrix zero. This generalizes the familiar factorization of a 3D rotation matrix into 2D rotations parametrized by the three Euler angles. In intuition, this parametrization works like Gaussian elimination by finishing one column at a time. There are infinitely many alternative decomposition schemes as well; Figure 2 shows two that are particularly convenient to implement in software (and even in neuromorphic hardware (Shen et al., 2017)).

The unitary matrix $\mathbf{W}_N$ is multiplied from the right by a succession of unitary matrices $\mathbf{R}_{Nj}$ for $j = N-1, \cdots, 1$. Once all elements of the last row except the one on the diagonal are zero, this row will not be affected by later transformations. Since all transformations are unitary, the resulting matrix should be unitary which means the last column will then also contain only zeros except on the diagonal:

$$\mathbf{W}_N \mathbf{R}_{N,N-1} \mathbf{R}_{N,N-2} \cdots \mathbf{R}_{N,1} = \begin{pmatrix} \mathbf{W}_{N-1} & 0 \\ 0 & e^{iw_N} \end{pmatrix} \tag{32}$$

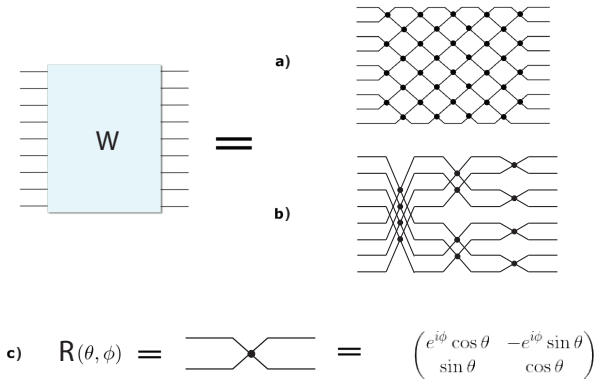

Figure 2: Figure adapted from (Jing et al., 2016): **Unitary matrix decomposition:** An arbitrary unitary matrix $\mathbf{W}$ can be decomposed (a) with the square decomposition method (Clements et al., 2016); or approximated (b) by the Fast Fourier Transformation(FFT) style decomposition method (Mathieu & LeCun, 2014). Each junction in the a) and b) graphs above represent the U(2) matrix as shown in c).

The effective dimensionality of the the matrix $\mathbf{W}$ is thus reduced to $N-1$. The same procedure can then be repeated $N-1$ times until the effective

dimension of $\mathbf{W}$ is reduced to 1, leaving with a diagonal matrix: [1]

$$\mathbf{W}_N\mathbf{R}_{N,N-1}\mathbf{R}_{N,N-2}\cdots\mathbf{R}_{i,j}\mathbf{R}_{i,j-1}\cdots\mathbf{R}_{3,1}\mathbf{R}_{2,1}=\mathbf{D}, \tag{33}$$

where $\mathbf{D}$ is a diagonal matrix whose diagonal elements are e^{iw_j}, from which the direct representation of $\mathbf{W}_N$ is [2]

$$\begin{aligned}\mathbf{W}_N &= \mathbf{D}\mathbf{R}_{2,1}^{-1}\mathbf{R}_{3,1}^{-1}\ldots\mathbf{R}_{N,N-2}^{-1}\mathbf{R}_{N,N-1}^{-1}\\ &= \mathbf{D}\mathbf{R}_{2,1}^{H}\mathbf{R}_{3,1}^{H}\ldots\mathbf{R}_{N,N-2}^{H}\mathbf{R}_{N,N-1}^{H}.\end{aligned} \tag{34}$$

This parametrization thus involves $N(N-1)/2$ different variables of θ_{ij}, $N(N-1)/2$ different variables of ϕ_{ij} and N different variables of w_i, combining to N^2 parameters in total and thus spans the entire unitary space. (Jing et al., 2016) then claims that full unitary space parametrization is not necessary by fixing a portion of our parameters and spanning only a subset of unitary space.

3.4.3 Tunable space implementation

Eq. (34) can be reordered and grouped with specific rotational matrices, as was shown in the optical community (Reck et al., 1994; Clements et al., 2016) in the context of universal multiport interferometers. For example (Clements et al., 2016), a unitary matrix can be decomposed as

$$\mathbf{W}_N \;=\; \mathbf{D}\mathbf{F}_A^{(1)}\mathbf{F}_B^{(2)}\ldots\mathbf{F}_{A/B}^{(L)}, \tag{35}$$

where the RHS ends with $\mathbf{F}_A$ if L is odd or with $\mathbf{F}_B$ if L is even and every

$$\mathbf{F}_A^{(l)} = \mathbf{R}_{1,2}^{H^{(l)}}\mathbf{R}_{3,4}^{H^{(l)}}\ldots\mathbf{R}_{2\lfloor\frac{N}{2}\rfloor-1,2\lfloor\frac{N}{2}\rfloor}^{H^{(l)}}$$

is a block diagonal matrix, with N angle parameters in total, and

$$\mathbf{F}_B^{(l)} = \mathbf{R}_{2,3}^{H^{(l)}}\mathbf{R}_{4,5}^{H^{(l)}}\ldots\mathbf{R}_{2\lceil\frac{N}{2}\rceil-2,2\lceil\frac{N}{2}\rceil-1}^{H^{(l)}}$$

with $N-1$ parameters, as is schematically shown in Fig. 2a. By choosing different values for L , $\mathbf{W}_N$ will span a different subspace of the unitary space. Specifically,when $L=N$, $\mathbf{W}_N$ will span the entire unitary space.

This is true because of two simple facts that 1) $\mathbf{R}_{i,j}\mathbf{R}_{i',j'}=\mathbf{R}_{i',j'}\mathbf{R}_{i,j}$ when i,j,i',j' are 4 distinct numbers. 2) for any diagonal complex matrix $\mathbf{D}$ and a rotation matrix $\mathbf{R}_{i,j}^H$ with parameter (ϕ_{ij},θ_{ij}), we can always find a $\mathbf{D}'$ and a rotation matrix $\mathbf{R}_{i,j}$ with parameter $(\phi'_{ij},\theta'_{ij})$, [3] such that $\mathbf{D}\mathbf{R}_{i,j}^H=\mathbf{R}_{i,j}\mathbf{D}'$.

[1] Note that Gaussian Elimination would make merely the upper triangle of a matrix vanish, requiring a subsequent series of rotations (complete Gauss-Jordan Elimination) to zero the lower triangle. We need no such subsequent series because since $\mathbf{W}$ is unitary: it is easy to show that if a unitary matrix is triangular, it must be diagonal.

[2] In physics society, R^H is always written as $R^\dagger$ and can be realized by reversing the linear optical elements.

[3] where $\phi'_{ij}=\omega_i-\omega_j$, $\omega'_i=\phi_{ij}+\omega_j$ and $\omega'_j=\omega_j$

(Jing et al., 2016) proposed an alternative way to organize the rotation matrices in an *FFT-style architecture*. Instead of using adjacent rotation matrices, each $\mathbf{F}$ here performs a certain distance pairwise rotations as shown in Fig. 2b:

$$\mathbf{W} = \mathbf{D}\mathbf{F}_1\mathbf{F}_2\mathbf{F}_3\mathbf{F}_4\cdots\mathbf{F}_{\log(N)}. \tag{36}$$

The rotation matrices in $\mathbf{F}_i$ are performed between pairs of coordinates

$$(p\cdot 2k+j, p\cdot(2k+1)+j) \tag{37}$$

where $p = N/2^i$, $k \in \{0, ..., 2^{i-1}\}$ and $j \in \{1, ..., p\}$. This requires only $\log(N)$ matrices, so there are a total of $N\log(N)/2$ rotational pairs. This is also the minimal number of rotations that can have all input coordinates interacting with each other, providing an approximation of arbitrary unitary matrices.

For instances, the 3 matrices for $N = 8$ case are:

$$\begin{pmatrix} e_1s_1 & & & & e_1c_1 & & & \\ & e_2s_2 & & & & e_2c_2 & & \\ & & e_3s_3 & & & & e_3c_3 & \\ & & & e_4s_4 & & & & e_4c_4 \\ c_1 & & & & -s_1 & & & \\ & c_2 & & & & -s_2 & & \\ & & c_3 & & & & -s_3 & \\ & & & c_4 & & & & -s_4 \end{pmatrix} \tag{38}$$

$$\begin{pmatrix} e_1s_1 & & e_1c_1 & & & & & \\ & e_2s_2 & & e_2c_2 & & & & \\ c_1 & & -s_1 & & & & & \\ & c_2 & & -s_2 & & & & \\ & & & & e_3s_3 & & e_3c_3 & \\ & & & & & e_4s_4 & & e_4c_4 \\ & & & & c_3 & & -s_3 & \\ & & & & & c_4 & & -s_4 \end{pmatrix} \tag{39}$$

$$\begin{pmatrix} e_1s_1 & e_1c_1 & & & & & & \\ c_1 & -s_1 & & & & & & \\ & & e_2s_2 & e_2c_2 & & & & \\ & & c_2 & -s_2 & & & & \\ & & & & e_3s_3 & e_3c_3 & & \\ & & & & c_3 & -s_3 & & \\ & & & & & & e_4s_4 & e_4c_4 \\ & & & & & & c_4 & -s_4 \end{pmatrix} \tag{40}$$

where $e_i = \exp(i\phi_i)$, $c_i = \cos\theta_i$, $s_i = \sin\theta_i$ and the variable with same subscript among these 3 matrices are distinct.

3.5 Kronecker recurrent units (KRU) (Jose et al., 2017)

Jose et al. (2017) consider parameterizing the recurrent matrix $\mathbf{W}$ as a Kronecker product of F matrices $\mathbf{W}_0, \ldots, \mathbf{W}_{F-1}$,

$$\mathbf{W} = \mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1} = \otimes_{f=0}^{F-1} \mathbf{W}_f. \tag{41}$$

Where each $\mathbf{W}_f \in \mathbb{C}^{P_f \times Q_f}$ and $\prod_{f=0}^{F-1} P_f = \prod_{f=0}^{F-1} Q_f = N$. $\mathbf{W}_f$'s are called as Kronecker factors.

To illustrate the Kronecker product of matrices, let us consider the simple case when $\forall_f \{P_f = Q_f = 2\}$. This implies $F = \log_2 N$. And $\mathbf{W}$ is recursevly defined as follows:

$$\mathbf{W} = \otimes_{f=0}^{\log_2 N-1} \mathbf{W}_f = \begin{bmatrix} \mathbf{w}_0(0,0) & \mathbf{w}_0(0,1) \\ \mathbf{w}_0(1,0) & \mathbf{w}_0(1,1) \end{bmatrix} \otimes_{f=1}^{\log_2 N-1} \mathbf{W}_f, \tag{42}$$

$$= \begin{bmatrix} \mathbf{w}_0(0,0)\mathbf{W}_1 & \mathbf{w}_0(0,1)\mathbf{W}_1 \\ \mathbf{w}_0(1,0)\mathbf{W}_1 & \mathbf{w}_0(1,1)\mathbf{W}_1 \end{bmatrix} \otimes_{f=2}^{\log_2 N-1} \mathbf{W}_f. \tag{43}$$

When $\forall_f \{p_f = q_f = 2\}$ the number of parameters is $8 \log_2 N$ and the time complexity of hidden state computation is $\mathcal{O}(N \log_2 N)$. When $\forall_f \{p_f = q_f = N\}$ then $F = 1$ and we will recover standard complex valued recurrent neural network. All Kronecker representations can be spanned in between by choosing the number of factors and the size of each factor. In other words, the number of Kronecker factors and the size of each factor give us fine-grained control over the number of parameters and hence over the computational efficiency. This strategy allows us to design models with the appropriate trade-off between computational budget and statistical performance.

The idea of using Kronecker factorization for approximating Fisher matrix in the context of natutal gradient methods have recently received much attention. The algorithm was originally presented in (Martens & Grosse, 2015) and was later extended to convolutional layers (Grosse & Martens, 2016), distributed second order optimization (Ba et al., 2016) and for deep reinforcement learning (Wu et al., 2017). However Kronecker matrices have not been well explored as learnable parameters except (Zhang et al., 2015) used it's spectral property for fast orthogonal projection and (Zhou et al., 2015) used it as a layer in convolutional neural networks.

3.5.1 Properties of Kronecker matrix (Van Loan, 2000)

Consider a matrix $\mathbf{W} \in \mathbb{C}^{N \times N}$ factorized as a Kronecker product of F matrices $\mathbf{W}_0, \ldots, \mathbf{W}_{F-1}$,

$$\mathbf{W} = \mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1} = \otimes_{i=0}^{F-1} \mathbf{W}_i. \tag{44}$$

Where each $\mathbf{W}_i \in \mathbb{C}^{P_i \times Q_i}$ respectively and $\prod_{i=0}^{F-1} P_i = \prod_{i=0}^{F-1} Q_i = N$. $\mathbf{W}_i$'s are called as Kronecker factors.

$$\text{If the factors } \mathbf{W}_i\text{'s are } \left\{\begin{matrix}\text{Nonsingular}\\ \text{Symmetric}\\ \text{Stochatsic}\\ \text{Orthogonal}\\ \text{Unitary}\\ \text{PSD}\\ \text{Toeplitz}\end{matrix}\right\} \text{ then } \mathbf{W} \text{ is } \left\{\begin{matrix}\text{Nonsingular}\\ \text{Symmetric}\\ \text{Stochatsic}\\ \text{Orthogonal}\\ \text{Unitary}\\ \text{PSD}\\ \text{Block Toeplitz}\end{matrix}\right\}$$

Theorem 2. *If* $\forall i \in 0, \ldots, F-1$, $\mathbf{W}_i$ *is unitary then* $\mathbf{W}$ *is also unitary.*

Proof.

$$\mathbf{W}^H\mathbf{W} = (\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1})^H(\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1}) \tag{45}$$

$$= (\mathbf{W}_0^H \otimes \cdots \otimes \mathbf{W}_{F-1}^H)(\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1}) \tag{46}$$

$$= \mathbf{W}_0^H\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1}^H\mathbf{W}_{F-1} = \mathbf{I}. \tag{47}$$

□

3.5.2 Product between a dense matrix and a Kronecker matrix

For simplicity here we use real number notations. Consider a dense matrix $\mathbf{X} \in \mathbb{R}^{M\times K}$ and a Kronecker factored matrix $\mathbf{W} \in \mathbb{R}^{N\times K}$. That is $\mathbf{W} = \otimes_{f=0}^{F-1}\mathbf{W}_f$, where each $\mathbf{W}_f \in \mathbb{R}^{P_f\times Q_f}$ respectively and $\prod_{f=0}^{F-1} P_f = N$ and $\prod_{f=0}^{F-1} Q_f = K$. Let us illustrate the matrix product $\mathbf{X}\mathbf{W}^T$ resulting in a matrix $\mathbf{Y} \in \mathbb{R}^{M\times N}$.

$$\mathbf{Y} = \mathbf{X}\mathbf{W}^T. \tag{48}$$

The computational complexity first expanding the Kronecker factored matrix and then computing the matrix product is $\mathcal{O}(MNK)$. This can be reduced by exploiting the following fact and recursive definition of Kronecker matrices.

$$\text{vec}(\mathbf{AXC}) = (\mathbf{C}^T \otimes \mathbf{A})\,\text{vec}(\mathbf{X}) \tag{49}$$

Applying this fact,

$$\mathbf{Y}^T = \mathbf{W}\mathbf{X}^T \tag{50}$$
$$= (\mathbf{W}_0 \otimes \cdots \otimes \mathbf{W}_{F-1}) (\mathbf{X}_{1,:}^T, \mathbf{X}_{2,:}^T, \cdots, \mathbf{X}_{M,:}^T) \tag{51}$$
$$= \left((\mathbf{W}_0 \otimes \underbrace{\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}}) \mathbf{X}_{i,:}^T \right)_{i=1:M} \tag{52}$$
$$= \left(\mathrm{vec} \left((\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}) \underbrace{\mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0) \mathbf{W}_0^T} \right) \right)_{i=1:M} \tag{53}$$
$$= \begin{pmatrix} (\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}) \mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0) \mathbf{W}_0^T(:,1) \\ (\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}) \mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0) \mathbf{W}_0^T(:,2) \\ \vdots \\ (\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}) \mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0) \mathbf{W}_0^T(:,P_0) \end{pmatrix}_{i=1:M} \tag{54}$$
$$= (\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}) (\mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0) \, \mathrm{vec}(\mathbf{W}_0^T))_{i=1:M} \tag{55}$$
$$= (\mathbf{W}_1 \otimes \cdots \otimes \mathbf{W}_{F-1}) (\mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0))_{i=1:M} \, \mathrm{vec}(\mathbf{W}_0^T) \tag{56}$$

where

$$\mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0) = \begin{pmatrix} \mathbf{X}_{i,1} & \mathbf{X}_{i,K/Q_0+1} & \cdots & \mathbf{X}_{i,(Q_0-1)K/Q_0+1} \\ \mathbf{X}_{i,2} & \mathbf{X}_{i,K/Q_0+2} & & \mathbf{X}_{i,(Q_0-1)K/Q_0+1} \\ \vdots & \vdots & & \vdots \\ \mathbf{X}_{i,K/Q_0} & \mathbf{X}_{i,2K/Q_0} & \cdots & \mathbf{X}_{i,K} \end{pmatrix} \tag{57}$$

is a K/Q_0 by Q_0 matrix. Define $\mathbf{Y_0}$ as

$$\mathbf{Y}_0^T = (\mathrm{Mat}(\mathbf{X}_{i,:}^T, K/Q_0, Q_0))_{i=1:M} \, \mathrm{vec}(\mathbf{W}_0^T) \tag{58}$$
$$= \mathbf{X}^T \odot \mathbf{W}_0^T \tag{59}$$

Please note that the binary operator $\odot$ is not the standard matrix multiplication operator but instead it denotes a strided matrix multiplication described above. Then we can define $\mathbf{Y}$ recursively:

$$\mathbf{Y}_0^T = \mathbf{X}^T \odot \mathbf{W}_0^T \tag{60}$$
$$\mathbf{Y}_f^T = \mathbf{Y}_{f-1}^T \odot \mathbf{W}_f^T. \tag{61}$$

and than we have

$$\mathbf{Y} = \mathbf{Y}_{F-1} \tag{62}$$

For examples when $N = K$ and $\forall_f \{P_f = Q_f = 2\}$, the matrix product can be computed in $\mathcal{O}(MN \log N)$ time instead of $\mathcal{O}(MN^2)$. This algorithm will cache all the intermediate outputs $(\mathbf{Y}_0, \ldots, \mathbf{Y}_{F-1})$ instead of just $\mathbf{Y}_{F-1}$. These intermediate outputs are then later to compute the gradients during the back-propagation. This cache if needed will save some computation during the back-propagation.

4 Hard or soft orthogonal and unitary constraint?

Poor conditioning results in vanishing or exploding gradients. Unfortunately, the standard solution which consists of optimization on the strict unitary set suffers from the retention of noise over time. Indeed, the small eigenvalues of the recurrent matrix can represent a truly vanishing long-term influence on the particular problem and in that sense, there can be good or bad vanishing gradients. Consequently, enforcing strict unitary constraint (forcing the network to never forget) can be a bad strategy. A simple solution to get the best of both worlds is to enforce unitary constraint approximately by using the following regularization on recurrent matrix:

$$\left\|\mathbf{W}^H\mathbf{W} - \mathbf{I}\right\|^2 \tag{63}$$

Henaff et al. (2016) and Vorontsov et al. (2017) have exploited this real-valued version of soft unitary constraints on standard RNN after identifying the problems with the strict unitary RNN models. Vorontsov et al. (2017) first use singular value decomposition to parametrize $\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ and then keep the bases $\mathbf{U}$ and $\mathbf{V}$ orthogonal via *geodesic gradient decent* on stifle manifold as the optimization method mentioned in 3.2.1 while parameterizing each singular value as

$$\sigma_i = 2m(\mathsf{sigmoid}(p_i) - 1/2) + 1, m \in [0, 1] \tag{64}$$

where m is a tunable hyper-parameter to control the margin of all singular values. As a consequence, $\sigma_i \in [1 - m, 1 + m]$ and $\mathbf{W}$ is softly restricted to be a orthogonal matrix.

However the computational complexity of naively applying this soft constraint is $\mathcal{O}(N^3)$. This is prohibitive for RNNs with large hidden state. Therefore, (Jose et al., 2017) considers a Kronecker factorization with such soft constraint,

$$\left\|\mathbf{W}_f^H\mathbf{W}_f - \mathbf{I}\right\|^2, \forall f \in \{0, \ldots, F-1\} \tag{65}$$

Please note that these constraints are enforced on each factor of the Kronecker factored recurrent matrix. This procedure is computationally very efficient since the size of each factor is typically small. It suffices to do so because if each of the Kronecker factors $\{\mathbf{W}_0, \ldots, \mathbf{W}_{F-1}\}$ are unitary then the full matrix $\mathbf{W}$ is unitary (Van Loan, 2000) and if each of the factors are approximately unitary then the full matrix is approximately unitary.

This type of regularizer has recently been exploited for real-valued models. Cisse et al. (2017) showed that enforcing approximate orthogonality constraint on the weight matrices make the network robust to adversarial samples as well as improve the learning speed. In metric learning (Jose & Fleuret, 2016) have shown that it better conditions the projection matrix thereby improving the robustness of stochastic gradient over a wide range of step sizes as well as the generalization performance.

In summary, although the idea of parametrizing recurrent weight matrices with strict unitary linear operator is appealing, it suffers from several issues:

(1) Strict unitary constraints severely restrict the search space of the model, thus making the learning process unstable. (2) Strict unitary constraints make forgetting irrelevant information difficult. While this may not be an issue for problems with non-vanishing long term influence, it causes failure when dealing with real world problems that have vanishing long term influence. Henaff et al. (2016) have previously pointed out that the good performance of strict unitary models on certain synthetic problems is because it exploits the biases in these data-sets which favours a unitary recurrent map and these models may not generalize well to real world data-sets. More recently Vorontsov et al. (2017) have also studied this problem of unitary RNNs and the authors found out that relaxing the strict unitary constraint on the recurrent matrix to a soft unitary constraint improved the convergence speed as well as the generalization performance. The latest work (Jose et al., 2017) enforces soft unitary constraint on each factor of the Kronecker factored recurrent matrix, which is computational efficient and proved to maintain the approximately unitary property of the final product of all factors.

5 Orthogonality in external associative memory

5.1 Long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997)

LSTM networks presented an elegant solution to the vanishing and exploding gradients through the introduction of gating mechanism. Apart from the standard hidden state in RNN, LSTM introduced one more state called cell state c_t. LSTM has three different gates whose functionality is described as follows:

- Forget gate ($\mathbf{W}_f, \mathbf{U}_f, \mathbf{b}_f$):Decides what information to keep and erase from the previous cell state.
- Input gate ($\mathbf{W}_i, \mathbf{U}_f, \mathbf{b}_i$): Decides what new information should be added to the cell state.
- Output gate ($\mathbf{W}_o, \mathbf{U}_o, \mathbf{b}_o$):Decides which information from the cell state is going to the output.

In addition to the gates, LSTM prepares candidates for the information from the input gate that might get added to the cell state through the action of input gate. Let's denote the parameters describing the function that prepares this candidate information as $\mathbf{W}_c, \mathbf{U}_c, \mathbf{b}_c$.

Given a sequence of T input vectors: $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{T-1}$, at a time step t LSTM performs the following:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}_{t-1} + \mathbf{U}_f \mathbf{x}_t + \mathbf{b}_f) \tag{66}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}_{t-1} + \mathbf{U}_i \mathbf{x}_t + \mathbf{b}_i) \tag{67}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}_{t-1} + \mathbf{U}_o \mathbf{x}_t + \mathbf{b}_o) \tag{68}$$
$$\hat{\mathbf{c}}_t = \tau(\mathbf{W}_c \mathbf{h}_{t-1} + \mathbf{U}_c \mathbf{x}_t + \mathbf{b}_c) \tag{69}$$
$$\mathbf{c}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t + \hat{\mathbf{c}}_t \odot \mathbf{i}_t \tag{70}$$
$$\mathbf{h}_t = \tau(\mathbf{c}_t) \odot \mathbf{o}_t \tag{71}$$

where $\sigma(.)$ and $\tau(.)$ are the point-wise sigmoid and tanh functions. $\odot$ indicates element-wise multiplication. The first three are gating operations and the 4th one prepares the candidate information. The 5th operation updates the cell-state and finally in the 6th operation the output gate decided what to go into the current hidden state.

5.2 Gated recurrent unit (GRU) (Chung et al., 2015)

A big step forward from LSTM is the Gated Recurrent Unit (Chung et al., 2015), which removed the extra memory state in LSTM. Specifically, the hidden state $\mathbf{h_t}$ in a GRU is updated as follows:

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z) \tag{72}$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \tag{73}$$
$$\hat{\mathbf{h}}_t = \tau(\mathbf{W}_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}_h \mathbf{x}_t + \mathbf{b}_h) \tag{74}$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \hat{\mathbf{h}}_t \tag{75}$$

- Remember gate $(\mathbf{W}_z, \mathbf{U}_z, \mathbf{b}_z)$: Decides what information to keep in the previous hidden state $\mathbf{h}_{t-1}$ and what to update with candidate current state $\hat{\mathbf{h}}_t$.
- Reset gate $(\mathbf{W}_r, \mathbf{U}_r, \mathbf{b}_r)$: Decides which part of previous hidden state could be updated.

5.3 Gated orthogonal recurrent unit (GORU) (Jing et al., 2017)

GORU simply enforces strict unitary constraint on hidden-to-hidden recurrent matrix $\mathbf{W}_h$ in GRU using the EURNN parameterization in section 3.4. To contrast GORU by GRU, we replace the notation of $\mathbf{W}_h$ by unitary matrix $\mathbf{W}$.

$$\mathbf{z}_t = \sigma(\mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z) \tag{76}$$
$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \tag{77}$$
$$\hat{\mathbf{h}}_t = \tau(\mathbf{W}(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{U}_h \mathbf{x}_t + \mathbf{b}_h) \tag{78}$$
$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \hat{\mathbf{h}}_t \tag{79}$$

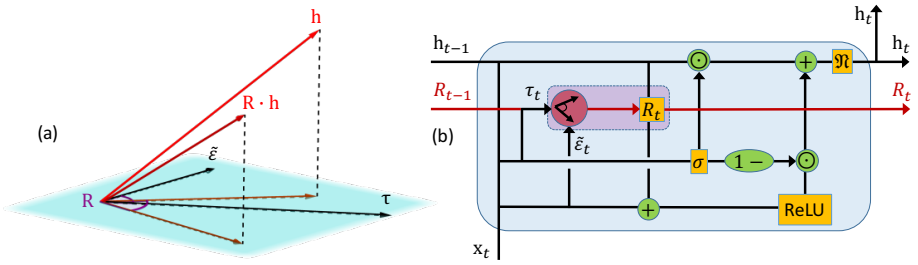

Figure 3: Figure adapted from (Dangovski et al., 2017): Rotation **is a universal differentiable operation that enables the advantages of the RUM architecture.** (a) The rotation $R(\mathbf{a}, \mathbf{b})$ in the plane defined by $\mathbf{a} = \tilde{\varepsilon}$ and $\mathbf{b} = \tau$ acts on the hidden state $\mathbf{h}$. (b) The RUM cell, in which Rotation encodes the kernel R. The matrix R_t acts on $\mathbf{h}_{t-1}$ and thus keeps the norm of the hidden state.

5.4 Rotational unit of memory (RUM) (Dangovski et al., 2017)

5.4.1 The operation: Rotation

The operation Rotation is an efficient encoder of an orthogonal operation, which acts as a unit of memory. Rotation computes an orthogonal operator $R(\mathbf{a}, \mathbf{b})$ in $\mathbb{R}^{N \times N}$ that represents the rotation between two non-collinear vectors $\mathbf{a}$ and $\mathbf{b}$ in the two-dimensional subspace $\text{span}(\mathbf{a}, \mathbf{b})$ of the Euclidean space $\mathbb{R}^N$ with distance $\|\cdot\|$. As a consequence, R can act as a kernel on a hidden state $\mathbf{h}$. More formally, R is a function

$$\text{Rotation}\colon \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}^{N \times N},$$

such that after ortho-normalizing $\mathbf{a}$ and $\mathbf{b}$ to

$$\mathbf{u}_a = \frac{\mathbf{a}}{\|\mathbf{a}\|} \quad \text{and} \quad \mathbf{u}_b = \frac{\mathbf{b} - (\mathbf{u}_a \mathbf{u}_a{}^T \mathbf{b})}{\|\mathbf{b} - (\mathbf{u}_a \mathbf{u}_a{}^T \mathbf{b})\|},$$

and get the rotation angle

$$\theta = \arccos \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\| \, \|\mathbf{b}\|},$$

then encode these information into the following matrix

$$R(\mathbf{a}, \mathbf{b}) = \left[\mathbf{I} - \mathbf{u}_a \mathbf{u}_a^T - \mathbf{u}_b \mathbf{u}_b^T\right] + (\mathbf{u}_a, \mathbf{u}_b)\, \tilde{R}(\theta)\, (\mathbf{u}_a, \mathbf{u}_b)^T. \tag{80}$$

Figure 3 (a) demonstrates the projection to the plane $\text{span}(\mathbf{a}, \mathbf{b})$ in the brackets of eq. (80). The mini-rotation in this space is $\tilde{R}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$. Hence, $\text{Rotation}(\mathbf{a}, \mathbf{b}) \equiv R(\mathbf{a}, \mathbf{b})$.

A practical advantage of Rotation is that it is both orthogonal and differentiable. On one hand, it is a composition of differentiable sub-operations, which enables learning via backpropagation. On the other hand, it preserves the norm of the hidden state, hence it can yield more stable gradients.

5.4.2 The RUM architecture

Recurrent Unit of Memory was the first application of Rotation to a recurrent cell. Figure 3 (b) is a sketch of the connections in the cell. RUM consists of an update gate $\mathbf{u} \in \mathbb{R}^N$ that has the same function as in GRU. Replacing the reset gate in GRU instead, the model learns a memory *target* variable $\tau \in \mathbb{R}^N$. RUM also learns to embed the input vector $\mathbf{x} \in \mathbb{R}^D$ into $\mathbb{R}^N$, yielding $\tilde{\varepsilon} \in \mathbb{R}^N$. Therefore, the rotation between the embedded input and the target is encoded in Rotation, which is accumulated to the associative memory unit $R_t \in \mathbb{R}^{N \times N}$ (originally initialized to the identity matrix). Here λ is a non-negative integer that is a hyper-parameter of the model. From here, the orthogonal R_t acts on the state $\mathbf{h}$ to produce an evolved hidden state $\tilde{\mathbf{h}}$. Finally RUM obtains the new hidden state via $\mathbf{u}$, just as in GRU. The RUM architecture are the following:

$$
\begin{aligned}
\left(\mathbf{u}'_t \quad \tau_t\right) &= \mathbf{W}_{u,\tau}\mathbf{h}_{t-1} + \mathbf{U}_{u,\tau}\mathbf{x}_t + \mathbf{b}_{u,\tau} && \text{initial update gate and memory target;}\\
\mathbf{u}_t &= \sigma(\mathbf{u}'_t) && \mathsf{sigmoid}\text{ activation of the update gate;}\\
\tilde{\varepsilon}_t &= \mathbf{U}_{\varepsilon}\mathbf{x}_t + \mathbf{b}_{\varepsilon} && \text{embedded input for Rotation;}\\
R_t &= (R_{t-1})^{\lambda}\mathsf{Rotation}(\tilde{\varepsilon}_t, \tau_t) && \text{rotational associative memory;}\\
\tilde{\mathbf{h}}_t &= \mathsf{ReLU}\left(\tilde{\varepsilon}_t + R_t\mathbf{h}_{t-1}\right) && \text{unbounded evolution of hidden state;}\\
\mathbf{h}'_t &= \mathbf{u}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{u}_t) \odot \tilde{\mathbf{h}}_t && \text{hidden state before time normalization } \mathfrak{N};\\
\mathbf{h_t} &= \eta\frac{\mathbf{h_t}}{\|\mathbf{h_t}\|} && \text{new hidden state, with norm } \eta.
\end{aligned}
$$

The kernel matrices have dimensions given by $\mathbf{U}_{u,\tau} \in \mathbb{R}^{2N \times D}$, $\mathbf{W}_{u,\tau} \in \mathbb{R}^{2N \times N}$ and $\mathbf{U}_{\varepsilon} \in \mathbb{R}^{2N \times D}$. The biases are variables $\mathbf{b}_{u,\tau} \in \mathbb{R}^{2N}$ and $\mathbf{b}_{\varepsilon} \in \mathbb{R}^N$. The norm η is a scalar hyper-parameter of the RUM model.

The orthogonal matrix $R(\tilde{\varepsilon}_t, \tau_t)$ conceptually takes the place of a kernel acting on the hidden state in GRU. This is the most efficient place to introduce an orthogonal operation, as the Gated Orthogonal Recurrent Unit (Jing et al., 2017) experiments suggest. The difference with the GORU cell is that GORU parameterizes and learns the kernel as an orthogonal matrix, while RUM does not parameterize the rotation R. Instead, RUM learns τ, which together with $\mathbf{x}$, determines R. The orthogonal matrix keeps the norm of the vectors, so the conventional tanh in gated mechanisms could be replaced by a ReLU activation.

While R is an orthogonal element of RUM, the norm of $\mathbf{h}'_t$ is not stable because of the ReLU activation. Therefore, Dangovski et al. (2017) suggests normalizing the hidden state $\mathbf{h_t}$ with norm η. This technique was named as *time normalization* because we usually feed mini-batches to the RNN during learning that have the shape (N_b, N_T), where N_b is the size of the batch and N_T is the length of the sequence that we feed in. Time normalization happens along the

sequence dimension as opposed to the batch dimension in batch normalization. Choosing appropriate η for the RUM model stabilizes learning and ensures the eigenvalues of the kernel matrices are bounded from above. This in turn means that the effect of exploding gradients could be controlled better with smaller η.

RUM is not a standard gated mechanism since it uses an update gate but does not have a reset gate. This issue is resolved by utilizing additional memory via the target vector τ. By feeding inputs to RUM, τ adapts to encode rotations, which align the hidden states in desired locations in $\mathbb{R}^N$, meanwhile without changing the norm of $\mathbf{h}$. The weight R_t seems like a input-dependent mechanism composing both reset and evolution operation over other conventional gated mechanisms, such as LSTM and GRU.

6 Conclusion

We have presented a bunch of new recurrent neural network models based on their core property that (approximately) preserves the norms. We analyze from both theoretical and computational perspectives. We show that the gradient vanishing and exploding problem in RNN can be solved perfectly when restrict the recurrent matrix to be orthogonal or unitary matrix. These parameterization methods suffers from different storage and computation consumption, which is a trade-off between the matrix search space and computational speed. Soft orthogonal constraints could improve convergence and generalization, and could also improve the computational efficiency. We see the norm-preserving property is also useful in the external memory like GORU and RUM architecture.

To further explore the methods efficiently parameterizing recurrent matrices with the help of unitary map with prior knowledge on specific classes of problems is a interesting direction, which needs to discover hidden structures of classes of recurrent matrices. Besides, the recurrent matrix in RNN is a square matrix, while in CNN or other deep neural architecture, there are many important transition wight matrix that is not square. Therefore, to explore the feasibility of restrict those matrix or transform to be orthogonal may also shed light to the gradient vanishing and other problems in CNN and capsule networks. The Capsule network in (Sabour et al., 2017; Anonymous, 2018) is most recently proposed to solve the problem of location correlations and transitional/rotational invariance in images. The orthogonal or unitary restriction might also empower this line of research. To explore the convex-value based architecture (Trabelsi et al., 2017) is also an interesting direction. While the complex value is widely used in signal processing and quantum information, the exploration on complex-value based neural networks might bridge the these area to deep learning society, which is significant and potential direction.

Acknowledgment

Great thanks to Ken Xingze Wang[4] for early discussion.

References

Anonymous. Matrix capsules with em routing. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HJWLfGWRb.

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

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pp. 2654–2662, 2014.

Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. 2016.

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

Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pp. 2285–2294, 2015.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pp. 2067–2075, 2015.

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

William R Clements, Peter C Humphreys, Benjamin J Metcalf, W Steven Kolthammer, and Ian A Walsmley. Optimal design for universal multiport interferometers. *Optica*, 3(12):1460–1465, 2016.

Matthieu Courbariaux, Jean-Pierre David, and Yoshua Bengio. Low precision storage for deep learning. *arXiv preprint arXiv:1412.7024*, 2014.

Rumen Dangovski, Li Jing, and Marin Soljacic. Rotational unit of memory. *arXiv preprint arXiv:1710.09537*, 2017.

[4] wxz@alum.mit.edu

Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pp. 2148–2156, 2013.

Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

R Gilmore. Lie groups, physics, and geometry: an introduction for physicists, engineers and chemists, cambridge univ. *Pr., Cambridge UK*, 2008.

Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

Mikael Henaff, Arthur Szlam, and Yann LeCun. Orthogonal rnns and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

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

Kennether Hoffman and Ray Kunze. Linear algebra, 2nd. *New J ersey: Prentice*, 1971.

Stephanie L Hyland and Gunnar Rätsch. Learning unitary operators with help from u(n). In *AAAI*, pp. 2050–2058, 2017.

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

Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljačić, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *arXiv preprint arXiv:1706.02761*, 2017.

Cijo Jose and François Fleuret. Scalable metric learning via weighted approximate rank component analysis. In *European Conference on Computer Vision*, pp. 875–890. Springer, 2016.

Cijo Jose, Moustpaha Cisse, and Francois Fleuret. Kronecker recurrent units. *arXiv preprint arXiv:1705.10142*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

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.

Yann Le Cun, John S Denker, and A Sara. Solla. 1989. optimal brain damage. NIPS.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pp. 2408–2417, 2015.

Michael Mathieu and Yann LeCun. Fast approximation of rotations and hessians matrices. *arXiv preprint arXiv:1404.7195*, 2014.

Zakaria Mhammedi, Andrew Hellicar, Ashfaqur Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. *arXiv preprint arXiv:1612.00188*, 2016.

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

Michael Reck, Anton Zeilinger, Herbert J Bernstein, and Philip Bertani. Experimental realization of any discrete unitary operator. *Physical Review Letters*, 73(1):58, 1994.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pp. 3857–3867, 2017.

Arthur Sard. The measure of the critical values of differentiable maps. *Bulletin of the American Mathematical Society*, 48(12):883–890, 1942.

Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature Photonics*, 2017.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

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

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Mikolov Tomáš. *Statistical language models based on neural networks*. PhD thesis, PhD thesis, Brno University of Technology, 2012.

Chiheb Trabelsi, Olexa Bilaniuk, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.

Charles F Van Loan. The ubiquitous kronecker product. *Journal of computational and applied mathematics*, 123(1):85–100, 2000.

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

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

Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Second-order optimization for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems*, pp. 5285–5294, 2017.

Xu Zhang, Felix X Yu, Ruiqi Guo, Sanjiv Kumar, Shengjin Wang, and Shi-Fu Chang. Fast orthogonal projection based on kronecker product. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2929–2937, 2015.

Shuchang Zhou, Jia-Nan Wu, Yuxin Wu, and Xinyu Zhou. Exploiting local structures with the kronecker layer in convolutional networks. *arXiv preprint arXiv:1512.09194*, 2015.