Efficient Orthogonal Parametrisation of Recurrent Neural Networks Using Householder Reflections

Zakaria Mhammedi†
Andrew Hellicar‡
Ashfaqur Rahman‡
James Bailey†

Abstract

Recurrent Neural Networks (RNNs) have been successfully used in many applications. However, the problem of learning long-term dependencies in sequences using these networks is still a major challenge. Recent methods have been suggested to solve this problem by constraining the transition matrix to be unitary during training, which ensures that its norm is exactly equal to one. These methods either have limited expressiveness or scale poorly with the size of the network when compared with the simple RNN case, especially when using stochastic gradient descent (SGD) with a small mini-batch size. Our contributions are as follows. We first show that constraining the transition matrix to be unitary is a special case of an orthogonal constraint. Therefore, it may not be necessary to work with complex-valued matrices. Then we present a new parametrisation of the transition matrix which allows efficient training of an RNN while ensuring that the matrix is always orthogonal. Using our approach, one SGD step can, in the worst case, be performed in time complexity $O(Tn^2)$, where $T$ and $n$ are the length of the input sequence and the size of the hidden layer respectively. This time complexity is the same as that of the simple RNN. Finally, we test our new parametrisation on problems with long-term dependencies. Our results show that the orthogonal constraint on the transition matrix applied through our parametrisation gives similar benefits to the unitary constraint, without the time complexity limitations.

1. Introduction

Recurrent Neural Networks (RNNs) have been successfully used in many applications involving time series. This is because RNNs are well suited for sequential data as they process inputs one element at a time and store relevant information in their hidden state. In practice, however, training simple RNNs can be challenging due to the problem of exploding and vanishing gradients (Hochreiter et al., 2001). It has been shown that exploding gradients can occur when the transition matrix of an RNN has a spectral norm larger than one (Glorot & Bengio, 2010). This results in an error surface, associated with some objective function, having very steep walls (Pascanu et al., 2013). On the other hand, when the spectral norm of the transition matrix is less than one, the information at one time step tend to vanish quickly after a few time steps. This makes it challenging to learn long-term dependencies in sequential data.

Different methods have been suggested to solve either the vanishing or exploding gradient problem. The LSTM has been specifically designed to help with the vanishing gradient (Hochreiter & Schmidhuber, 1997). This is achieved by using gate vectors which allow a linear flow of information through the hidden state. However, the LSTM does not directly address the exploding gradient problem. One approach to solving this issue is to clip the gradients (Mikolov, 2012) when their norm exceeds some threshold value. However, this adds an extra hyperparameter to the model. Furthermore, if exploding gradients can occur within some parameter search space, the associated error surface will still have steep walls. This can make training challenging even with gradient clipping.

Another way to approach this problem is to improve the shape of the error surface directly by making it smoother, which can be achieved by constraining the spectral norm of the transition matrix to be less than or equal to one. However, a value of exactly one is best for the vanishing gradient problem. A good choice of the activation function between hidden states is also crucial in this case. These ideas have been investigated in recent works. In particu-
lar, the unitary RNN (Arjovsky et al., 2015) uses a special parametrisation to constrain the transition matrix to be unitary, and hence, of norm one. This parametrisation and other similar ones (Hyland & Rätsch, 2016; Wisdom et al., 2016) have some advantages and drawbacks which we will discuss in more details in the next section.

The main contributions of this work are as follows:

- We first show that constraining the search space of the transition matrix of an RNN to the set of unitary matrices \( U(n) \) is equivalent to limiting the search space to a subset of \( O(2n)^8 \) of a new RNN with twice the hidden size. This suggests that it may not be necessary to work with complex matrices.

- We present a simple way to parametrise orthogonal transition matrices of RNNs using Householder matrices, and we derive the expressions of the back-propagated gradients with respect to the new parametrisation. This new parametrisation can also be used in other deep architectures.

- We develop algorithms to compute the back-propagated gradients efficiently. Using these algorithms, we show that the worst case time complexity of one gradient step is of the same order as that of the simple RNN.

2. Related Work

Throughout this work we will refer to elements of the following simple RNN (sRNN) architecture.

\[
\begin{align*}
    h_t &= \phi(Wh_{t-1} + Vx_t), \\
    o_t &= Yh_t,
\end{align*}
\]

where \( W \), \( V \), and \( Y \) are the hidden-to-hidden, input-to-hidden, and hidden-to-output weight matrices. \( h_{t-1} \) and \( h_t \) are the hidden vectors at time steps \( t - 1 \) and \( t \) respectively. Finally, \( \phi \) is a non-linear activation function. We have omitted the bias terms for simplicity.

Recent research explored how the initialisation of the transition matrix \( W \) influences training and the ability to learn long-term dependencies. In particular, initialisation with the identity or an orthogonal matrix can greatly improve performance (Le et al., 2015). In addition to these initialisation methods, one study also considered removing the non-linearity between the hidden-to-hidden connections (Henaff et al., 2016), i.e. the term \( Wh_{t-1} \) in Equation (1) is outside the activation function \( \phi \). This method showed good results when compared to the LSTM on pathological problems exhibiting long-term dependencies.

After training a model for a few iterations using gradient descent, nothing guarantees that the initial structures of the transition matrix will be held. In fact, its spectral norm can deviate from one, and exploding and vanishing gradients can be a problem again. It is possible to constrain the transition matrix to be orthogonal during training using special parametrisations (Arjovsky et al., 2015; Hyland & Rätsch, 2016), which ensure that its spectral norm is always equal to one. The unitary RNN (uRNN) (Arjovsky et al., 2015) is one example where the hidden matrix \( W \in \mathbb{C}^{n \times n} \) is the product of elementary matrices, consisting of reflection, diagonal, and Fourier transform matrices. When the size of hidden layer is equal to \( n \), the transition matrix has a total of only \( 7n \) parameters. Another advantage of this parametrisation is computational efficiency - the matrix-vector product \( Wv \), for some vector \( v \), can be calculated in time complexity \( O(n \log n) \). However, it has been shown that this parametrisation does not allow the transition matrix to span the full unitary group (Wisdom et al., 2016) when the size of the hidden layer is greater than 7. This may limit the expressiveness of the model.

Another interesting parametrisation (Hyland & Rätsch, 2016) has been suggested which takes advantage of the algebraic properties of the unitary group \( U(n) \). The idea is to use the corresponding matrix Lie algebra \( u(n) \) of skew hermitian matrices. In particular, the transition matrix can be written as \( W = \exp \left[ \sum_{i=1}^{n^2} \lambda_i T_i \right] \), where \( \exp \) is the exponential map and \( \{T_i\}_{i=1}^{n^2} \) are predefined \( n \times n \) matrices forming a bases of the Lie algebra \( u(n) \). The learning parameters are the weights \( \{\lambda_i\} \). The fact that the matrix Lie algebra \( u(n) \) is closed and connected ensures that the exponential mapping from \( u(n) \) to \( U(n) \) is surjective. Therefore, with this parametrisation the search space of the transition matrix spans the whole unitary group. This is one advantage over the original unitary parametrisation (Arjovsky et al., 2015). However, the cost of computing the matrix exponential to get \( W \) is \( O(n^3) \), where \( n \) is the dimension of the hidden state.

Another method (Wisdom et al., 2016) performs optimisation directly of the Stiefel manifold using the Cayley transformation. The corresponding model was called full-capacity unitary RNN. Using this approach, the transition matrix can span the full unitary group. However, this method involves a matrix inverse as well as matrix-matrix products which have time complexity \( O(n^3) \). This can be problematic in an online learning setting when the size of the hidden layer is large.

A more recent study (Vorontsov et al., 2017) investigated the effect of soft versus hard orthogonal constraints on the performance of RNNs. The soft constraint was applied by specifying an allowable range for the maximum singular value of the transition matrix. To this end, the transition
matrix was factorised as \( W = USV^T \), where \( U \) and \( V \) are orthogonal matrices and \( S \) is a diagonal matrix containing the singular values of \( W \). A soft orthogonal constraint consists, therefore, of specifying small allowable intervals around 1 for the diagonal elements of \( S \). Similarly to (Wisdom et al., 2016), the matrices \( U \) and \( V \) were updated at each training iteration using the Cayley transformation to ensure that they remain orthogonal. This transformation involves a matrix inverse.

All the methods discussed above, except for the original unitary RNN, involve a step that requires at least an \( O(n^4) \) time complexity. Furthermore, they all require the use of complex matrices. Table 1 summarises the time complexities of various methods, including our approach, for one online gradient step. In the next section, we show that imposing a unitary constraint on a transition matrix \( W \in \mathbb{C}^{n \times n} \) is equivalent to imposing a special orthogonal constraint on a new RNN with twice the hidden size. Furthermore, since the norm of orthogonal matrices is also always one, using the latter has the same theoretical benefits as using unitary matrices when it comes to the exploding gradient problem.

### 3. Complex unitary versus orthogonal

We can show that when the transition matrix \( W \in \mathbb{C}^{n \times n} \) of an RNN is unitary, there exists an equivalent representation of this RNN involving an orthogonal matrix \( \hat{W} \in \mathbb{R}^{2n \times 2n} \).

In fact, consider a complex unitary transition matrix \( W = A + iB \in \mathbb{C}^{n \times n} \), where \( A \) and \( B \) are now real-valued matrices in \( \mathbb{R}^{n \times n} \). We also define the following new variables

\[
\forall t, \hat{h}_t = \left[ \begin{array}{c} R(h_t) \\ \Im (h_t) \end{array} \right], \quad \hat{V} = \left[ \begin{array}{c} R(V) \\ \Im (V) \end{array} \right], \quad \hat{W} = \left[ \begin{array}{cc} A & -B \\ B & A \end{array} \right],
\]

where \( R \) and \( \Im \) denote the real and imaginary parts of a complex number. Note that \( \hat{h}_t \in \mathbb{R}^{2n} \), \( \hat{W} \in \mathbb{R}^{2n \times 2n} \), and \( \hat{V} \in \mathbb{R}^{2n \times n_u} \), where \( n_u \) is the dimension of the input vector \( x_t \) in Equation (1).

Assuming that the activation function \( \phi \) applies to the real and imaginary parts separately, it is easy to show that the update equation of the complex hidden state \( h_t \) of the unitary RNN can be written in the real space as follows

\[
\hat{h}_t = \phi(\hat{W}\hat{h}_{t-1} + \hat{V}x_t).
\]  

Even when the activation function \( \phi \) does not apply to the real and imaginary parts separately, it is still possible to find an equivalent representation in the real space. Consider the activation function proposed by (Arjovsky et al., 2015)

\[
\sigma_{\text{modReLU}}(z) = \begin{cases} (|z| + b)^{\frac{2}{|z|}} & \text{if } |z| + b > 0 \\ 0 & \text{otherwise} \end{cases}
\]  

where \( b \) is a bias vector. For a hidden state \( \hat{h} \in \mathbb{R}^{2n} \), the equivalent activation function in the real representation is

\[
\hat{\phi}(a) = \begin{cases} \sqrt{a_i^2 + a_k^2 + b_{k'}} & \text{if } a_i^2 + a_k^2 + b_{k'} > 0 \\ 0 & \text{otherwise} \end{cases}
\]

where \( k_i = ((i + n) \mod 2n) \) and \( k'_i = (i \mod n) \) for all \( i \in \{1, \ldots, 2n\} \). This activation is no longer applied to hidden units independently.

Now we will show that the matrix \( \hat{W} \) is orthogonal. By definition of a unitary matrix, we have \( WW^H = I \) where the \( H \) represents the conjugate transpose. This implies that \( A^2 + B^2 = I \) and \( BA - AB = 0 \). And since we have

\[
\hat{W}\hat{W}^T = \begin{bmatrix} A^2 + B^2 & AB - BA \\ BA - AB & B^2 + A^2 \end{bmatrix},
\]

it follows that \( \hat{W}\hat{W}^T = I \). Also note that \( \hat{W} \) has a special structure - it is a block-matrix.

The discussion above shows that using a complex, unitary transition matrix in \( \mathbb{C}^{n \times n} \) is equivalent to using an orthogonal matrix, belonging to a subset of \( \mathcal{O}(2n) \), in a new RNN with twice the hidden size. This is why in this work we focus mainly on parametrising orthogonal matrices.

### 4. Parametrisation of the transition matrix

Before discussing the details of our parametrisation, we first introduce a few notations. For \( n, k \in \mathbb{N} \) and \( 2 \leq k \leq n \), let \( H_k \) be the mapping from \( \mathbb{R}^k \) to \( \mathbb{R}^{n \times n} \) defined as

\[
\forall u \in \mathbb{R}^k, \quad H_k(u) = \begin{bmatrix} I_{n-k} & 0 \\ 0 & I_k - 2u'uu^T \end{bmatrix},
\]

where \( I_k \) denotes the \( k \)-dimensional identity matrix. For \( u \in \mathbb{R}^k \), \( H_k(u) \) is the Householder Matrix in \( \mathcal{O}(n) \) representing the reflection about the hyperplane orthogonal to the vector \( u \) and passing through the origin.

For the special case of \( k = 1 \), we define \( H_1 \) to be the mapping from \( \mathbb{R} \) to \( \mathbb{R}^{n \times n} \) defined as

\[
\forall u \in \mathbb{R}, \quad H_1(u) = \begin{bmatrix} I_{n-1} & 0 \\ 0 & u' \end{bmatrix},
\]

Note that, when \( u \in \{1, -1\} \), \( H_1(u) \) is orthogonal.

Finally, for \( n \in \mathbb{N} \) and \( 1 \leq k \leq n \), we define

\[
\mathcal{M}_k : \mathbb{R}^k \times \cdots \times \mathbb{R}^n \to \mathbb{R}^{n \times n} \quad (u_k, \ldots, u_n) \mapsto H_n(u_n) \cdots H_k(u_k).
\]

We propose to parametrise the transition matrix \( W \) of an RNN using the mappings \( \{\mathcal{M}_k\} \). When using \( m \) reflection
This parametrisation has the following advantages:

1. The parametrisation is smooth\(^3\), which is convenient for training with gradient descent. It is also flexible - a good trade-off between expressiveness and speed can be found by tuning the number of reflection vectors.

2. The time and space complexities involved in one gradient calculation are, in the worst case, the same as that of the sRNN with the same number of hidden units. This is discussed in the following subsections.

3. When \( m < n \), the matrix \( W \) is always orthogonal, as long as the reflection vectors are nonzero. For \( m = n \), the only additional requirement is that \( u_i \in \{-1, 1\} \).

4. When \( m = n \), the transition matrix can span the whole set of \( n \times n \) orthogonal matrices. In this case, the total number of parameters needed for \( W \) is \( n(n + 1)/2 \). This results in only \( n \) redundant parameters since the orthogonal set \( O(n) \) is a \( (n(n - 1))/2 \) manifold.

Note that \( H_1(u_1) \) is not the standard Householder reflection. Theorem 1 would not be valid if a standard Householder reflection was used instead of \( H_1 \). For example, in the two-dimensional case, the matrix

\[
\begin{pmatrix}
1 & 0 \\
0 & -1
\end{pmatrix}
\]

cannot be expressed as the product of exactly two standard Householder Reflections.

\(^3\)except on a subset of zero Lebesgue measure.

### Table 1.

| Methods                     | Constraint on the transition matrix | Time complexity of one online gradient step | Search space of the transition matrix |
|-----------------------------|-------------------------------------|--------------------------------------------|--------------------------------------|
| uRNN (Arjovsky et al., 2015)| \( \|W\| = 1 \)                     | \( O(Tn \log(n)) \) in the worst case      | A subset of \( U(n) \) when \( n > 7 \) |
| Full-capacity uRNN (Wisdom et al., 2016) | \( \|W\| = 1 \)                     | \( O(Tn^2 + n^3) \) in the worst case      | The full \( U(n) \) set                |
| Unitary RNN (Hyland & Rätsch, 2016) | \( \|W\| = 1 \)                     | \( O(Tn^2 + n^3) \) in the worst case      | The full \( O(n) \) set                |
| oRNN (Our approach)        | \( \|W\| = 1 \)                     | \( O(Tnm) \) in the worst case, where \( m \leq n \) when \( m = n \) | The full \( O(n) \) set                |

### 4.1. Back-propagation algorithm

Let \( u_i \in \mathbb{R}^i \). Let \( U = (u_n \ldots u_{n-m+1}) \in \mathbb{R}^{n \times m} \) be the parameter matrix constructed from the reflection vectors \( \{u_i\} \) as follows

\[
U_{ij} = \begin{cases} 
[u_n-j+1]_{i-j+1}, & \text{if } j \leq \min(i,m) \\
0, & \text{otherwise}
\end{cases}
\]

Let \( \mathcal{L} \) be a scalar loss function and \( C_t \triangleq W_{ht-1}, \) where \( W \) is constructed using the \( \{u_i\} \) vectors following Equation (8). In order to back-propagate the gradients through time, we need to compute the following partial derivatives

\[
\nabla C_t \triangleq \left[ \frac{\partial C_t}{\partial U} \right]^T \frac{\partial \mathcal{L}}{\partial C_t},
\]

\[
\nabla h_{t-1} \triangleq \left[ \frac{\partial C_t}{\partial h_{t-1}} \right]^T \frac{\partial \mathcal{L}}{\partial C_t},
\]

at each time step \( t \). Note that in Equation (10) \( h_{t-1} \) is taken as a constant with respect to \( U \). Furthermore, we have \( \frac{\partial C_t}{\partial u_i} = \sum_{t=2}^T \nabla C_t \) and \( \frac{\partial C_t}{\partial h_{t-1}} = \nabla h_{t-1} \). The gradient flow through the RNN at time step \( t \) is shown in Figure 1.

![Figure 1. Gradient flow through the RNN at time step \( t \). Note that we have \( \mathcal{L} = \sum_{t=1}^T \mathcal{L}_t(\alpha_t, \gamma_t) \), where \( \{\gamma_t\} \) are target outputs.](image)

Before describing the algorithm to compute the back-propagated gradients \( \nabla C_t \) and \( \nabla h_{t-1} \), we first derive their expressions as a function of \( U, h_{t-1} \) and \( \frac{\partial C_t}{\partial C_t} \) using the compact \( WY \) representation (Joffrain et al., 2006) of the product of Householder reflections.
Proposition 1. Let \( n, m \in \mathbb{N} \) such that \( m \leq n - 1 \). Let \( u_i \in \mathbb{R}^1 \) and \( U = (u_n, \ldots |u_{n-m+1}) \) the matrix defined in Equation (9). We have
\[
T \triangleq \text{striu}(UTU) + \frac{1}{2}\text{diag}(UTU),
\]
where \( \text{striu}(UTU) \) and \( \text{diag}(UTU) \) represent the strictly upper triangular part and the diagonal of the matrix \( UTU \), respectively. Equation (13) is the compact WY representation of the product of householder reflections.

For the particular case where \( m = n \), the RHS of Equation (13) should be replace by \((I - UT^{-1}UT)H_1(u_1)\), where \( H_1 \) is defined in (7) and \( U = (u_n, \ldots |u_2) \).

The following theorem, gives the expressions of the gradients of this algorithm.

Theorem 2. For \( m \leq n - 1 \), let \( U \in \mathbb{R}^{n \times m}, h \in \mathbb{R}^n \), and \( C = (I - UT^{-1}UT)h \), where \( T \) is defined in Equation (12). If \( \mathcal{L} \) is a scalar loss function which depends on \( C \), then the gradients \( \frac{\partial C}{\partial h} \) can be expressed as
\[
\frac{\partial C}{\partial h} = \left[ \frac{T \partial C}{\partial h} \right] + h, \quad \text{where} \quad T = \frac{\partial C}{\partial h} = \left( \frac{T \partial C}{\partial h} \right) + h.
\]

The proof of Equations (14) and (15) is provided in the supplementary material. Based on Theorem 2, Figure 2 displays a MATLAB code which performs one-step forward-propagation (FP) and back-propagation (BP) required to compute \( h_t, U_t, \) and \( h_{t-1} \). The required inputs for the FP and BP are, respectively, the tuples \((U, h_{t-1})\) and \((U, h_t, C_t)\).

4.2. Time and Space complexity

Note that the zero-initialisation at the beginning of the MATLAB code in Figure 2 is not necessary - only one preallocation is required. From this script, the flop count required at each time step \( t \) is \((13n+2)m^2; 6nm\) for the one-step FP and \((7n+2)m\) for the one-step BP. Note that the vector \( UU \) only needs to be calculated at one time step. This reduces the flop count at the remaining time steps to \((11n+3)m\). The fact that the matrix \( U \) has all zeros in its upper triangular part can be used to further reduce the total

\[ ^{\text{The time complexity involved in the initialisation of } \hat{h} \text{ and } h_{\text{bar}} \text{ was not taken into account.}} \]

# Summary

Table 2 summarises the flop counts for the FP and BP steps. Note that these flop counts are for the sRNN case with \( n \) hidden units, the global FP and BP have time complexities \( \approx 2n^2T \) and \( \approx 3n^2T \) for the one-step BP.

Note that if the values of the matrices \( \hat{f} \text{ and } \hat{C} \) are stored in the MATLAB code (see Figure 2), they are first stored during a "global" FP (i.e. using a single time step) and then used in the BP steps, the time complexity** for a global FP and BP using one input sequence of size \( T \) are, respectively, \( \approx n^2T \) and \( \approx 6n^2T \), when \( m \approx n \) and \( n \gg 1 \). In contrast with the sRNN case with \( n \) hidden units, the global FP and BP have time complexities \( \approx 2n^2T \) and \( \approx 3n^2T \). Hence, when \( m \approx n \), the FP and BP steps using our parametrisation require only about twice more flops than the sRNN case with the same number of hidden units.

Note, however, that storing the values of the matrices \( \hat{f} \text{ and } \hat{C} \) at all time step requires the storage of \( mnT \) values for one sequence of length \( T \), compared with \( nT \) when only the hidden states \{\( h_t \}_{t=1}^T \} \) are stored. When \( n \gg 1 \) this may not be practical. One solution to this problem is to generate the matrices \( \hat{f} \text{ and } \hat{C} \) locally at each BP step using \( U \) and \( h_{t-1} \). This results in a global BP complexity of \((11n + 5 - 2m)mT \). Table 2 summarises the flop counts for the FP and BP steps. Note that these flop counts are for

**We considered only the time complexity due to computations through the hidden-to-hidden connections of the network.
the case when \( m \leq n - 1 \). When \( m = n \), the complexity added due to the multiplication by \( H_1(u_1) \) is negligible.

4.3. Extension to the Unitary case

Although we decided to focus on the set of real-valued orthogonal matrices, for the reasons given in Section 3, our parametrisation can readily be modified to apply to the general unitary case. Namely, the mappings \( \{ H_k \}_{k=2}^{k=n} \) should be modified to go from \( \mathbb{C}^k \) to \( \mathbb{C}^{n \times n} \) and the transpose sign in Equation (6) should be replaced by the conjugate transpose. Furthermore, to allow the transition matrix to span the full set of \( n \times n \) unitary matrices, the mapping \( H_1 \) should be replaced by

\[
H_1 : \mathbb{R}^n \rightarrow \mathbb{C}^{n \times n} \\
\theta \mapsto \text{diag}(e^{i\theta_1}, \ldots, e^{i\theta_n}).
\]

With these new definitions of \( \{ H_k \}_{k=1}^{k=n} \), we have the following corollary

**Corollary 1.** Let \( \mathcal{M}_1 \) be the mapping defined as

\[
\mathcal{M}_1 : \mathbb{R}^n \times \mathbb{C}^2 \times \cdots \times \mathbb{C}^n \rightarrow \mathbb{C}^{n \times n} \\
(\theta, u_2, \ldots, u_n) \mapsto H_n(u_n) \cdots H_2(u_2) H_1(\theta).
\]

The image of \( \mathcal{M}_1 \) spans the full set of unitary matrices \( \mathbb{U}(n) \) and any point on its image is a unitary matrix.

The algorithm in Figure 2 can be modified accordingly for efficient BP through time in the unitary case.

5. Experiments

All RNN models were implemented using the python library *theano* (Theano Development Team, 2016). For efficiency, we implemented the one-step FP and BP algorithms described in Figure 2 using C code. We have tested the new parametrisation on five different datasets all having long-term dependencies. We call our parametrised network oRNN (for orthogonal RNN). We set its activation function to the leaky_ReLU defined as \( \phi(x) = \max(\tfrac{1}{2}x, x) \). To ensure that the transition matrix of the oRNN is always orthogonal, we set the scalar \( u_1 \) to -1 if \( u_1 \leq 0 \) and 1 otherwise after each gradient update. Note that the parameter matrix \( U \) in Equation (9) has all zeros in its upper triangular part. Therefore, after calculating the gradient of an objective function with respect to \( U \), the values in the upper triangular part are set to zero.

For all experiments, we used the *adam* method for stochastic gradient descent. We initialised all the parameters using uniform distributions similar to (Arjovsky et al., 2015). The biases of all models were set to zero, except for the forget bias of the LSTM, which we set to 5 to facilitate the learning of long-term dependencies (Koutník et al., 2014).

5.1. Sequence generation

In this experiment, we followed a similar setting to (Koutník et al., 2014) where we trained RNNs to encode song excerpts. We used the track *Manyrista* from album *Musica Deposita* by *Cuprum*. We extracted five consecutive excerpts around the beginning of the song, each having 800 data points and corresponding to 18ms with a 44.1Hz sampling frequency. We trained an sRNN, LSTM, and oRNN for 5000 epochs on each of the pieces with five random seeds. For each run, the lowest Normalised Mean Squared Error (NMSE) during the 5000 epochs was recorded. For each model, we tested three different hidden sizes. The total number of parameters corresponding to these hidden sizes was approximately equal to 250, 500, and 1000. For the oRNN, we set the number of reflection vectors to the hidden size for each case, so that the transition matrix is allowed to span the full orthogonal group. The results are shown in Figures 3 and 4. All the learning rates were set to \( 10^{-3} \). The orthogonal parametrisation outperformed the simple RNN and performed on average better than the LSTM.
Table 3. Results summary for the MNIST digit classification experiment and comparison with the uRNN results available in the literature. 'FC' and 'RC' stand for Full-Capacity and Restricted Capacity respectively. The best validation accuracy for the case of oRNN (m=128) was achieved with mini-batch size of 50 and learning rate $10^{-3}$. For the oRNN (m=256) a mini-batch size of 1 was best.

5.2. Adding Task

In this experiment, we followed a similar setting to (Arjovsky et al., 2015), where the goal of the RNN is to output the sum of two elements in the first dimension of a two-dimensional sequence. The location of the two elements to be summed are specified by the entries in the second dimension of the input sequence. In particular, the first dimension of every input sequence consists of random numbers between 0 and 1. The second dimension has all zeros except for two elements equal to 1. The first unit entry is located in the first half of the sequence, and the second one in the second half. We tested two different sequence lengths $T = 400, 800$. All models were trained to minimise the Mean Squared Error (MSE). The baseline MSE for this task is 0.167; for a model that always outputs one.

The oRNN was able to beat the baseline MSE in less than 5000 iterations for both lags and for two different random initialisation seeds. This is in line with the results of the unitary RNN (Arjovsky et al., 2015).

5.3. Pixel MNIST

In this experiment, we used the MNIST image dataset. We split the dataset into training (55000 instances), validation (5000 instances), and test sets (10000 instances). We trained two oRNNs with $(n,m) = (128, 16)$ and $(n,m) = (256, 32)$ respectively, where $n$ is the number of hidden units and $m$ the number of reflections, to minimise the cross-entropy. These networks had approximately 3.6K and 11K parameters respectively. We experimented with a mini-batch size of 50 with learning rate $10^{-3}$, and also
The total number of unique characters in the corpus was 49. The vocabulary size was 10K and any other words were replaced by the special token `<unk>`. The number of characters per instance (or line) in the training data ranged between 2 and 518 with an average of 118 char/line. We trained an oRNN and LSTM with hidden units 512 and 183 respectively, corresponding to a total of $\approx 180$K parameters, for 20 epochs. We set the number of reflections to 510 for the oRNN. The learning rate was set to 0.0001 for both models with a mini-batch size of 1.

Table 3 compares the test performance of our models against results available in the literature for unitary/orthogonal RNNs. Despite having fewer total number of parameters, our models performed better than three out the four models selected for comparison (all having $\geq 16K$ parameters). Figure 6 shows the convergence curves of the validation accuracy. The rate of convergence increased considerably with the number of parameters.

5.4. Penn Tree Bank

In this experiment, we tested the oRNN on the task of character level prediction using the Penn Tree Bank Corpus. The data was split into training (5017K characters), validation (393K characters), and test sets (442K characters). The total number of unique characters in the corpus was 49. The vocabulary size was 10K and any other words were replaced by the special token `<unk>`. The number of characters per instance (or line) in the training data ranged between 2 and 518 with an average of 118 char/line. We trained an oRNN and LSTM with hidden units 512 and 183 respectively, corresponding to a total of $\approx 180$K parameters, for 20 epochs. We set the number of reflections to 510 for the oRNN. The learning rate was set to 0.0001 for both models with a mini-batch size of 1.

Similar to (Pascanu et al., 2013) we considered two tasks: one where the model predicts one character ahead and the other where it predicts a character five steps ahead. It was suggested that solving the later task would require the learning of longer term correlations in the data rather than the shorter ones. Table 4 summarises the test results. The oRNN and LSTM performed similarly to each other on the one-step ahead prediction task. Whereas on the five-step ahead prediction task, the LSTM was better. Note that the performance of both models on this task was not too far from the state of the art result for RNNs which is 3.74 bpc (Pascanu et al., 2013).

On the one-step-ahead prediction task, our oRNN outperformed the results of (Vorontsov et al., 2017) which used both soft and hard orthogonality constraints on the transition matrix. Their RNN was trained on 99% of the data (sentences with $\leq 300$ characters) and had the same number of hidden units as the oRNN in our experiment. The lowest test cost achieved was 2.20(bpc) for the one-step-ahead prediction task.

5.5. Copying problem

We tested our model on the copy task (Gers et al., 2001) described in details in (Gers et al., 2001; Arjovsky et al., 2015). Using an oRNN with the leaky.ReLU activation function (Chernodub & Nowicki, 2016), which is a norm-preserving activation function. In order to explore whether the poor performance of the oRNN was only due to the activation function, we have tested the same activation as the uRNN (Arjovsky et al., 2015; Wisdom et al., 2016). However, we were able to achieve a comparable performance when using the OPLU activation function (Chernodub & Nowicki, 2016), which is a norm-preserving activation function. By using an oRNN with the leaky.ReLU activation function, we were able to achieve a comparable performance when using the OPLU activation function (Chernodub & Nowicki, 2016), which is a norm-preserving activation function. In order to explore whether the poor performance of the oRNN was only due to the activation function, we have tested the same activation as the uRNN (i.e. the real representation of modReLU defined in Equation (4)) on the oRNN. This did not improve the performance compared to the leaky.ReLU case suggesting that the block structure of the uRNN transition matrix (when expressed in the real space) may confer special benefits in some cases.

6. Discussion

In this work, we presented a new parametrisation of the transition matrix of a recurrent neural network using Householder reflections. This method allows an easy and computationally efficient way to enforce an orthogonal constraint on the transition matrix which then ensures that exploding gradients do not occur during training. Our method could also be applied to other deep neural architectures to enforce orthogonality between hidden layers.

It is important to note that our method is particularly advantageous for stochastic gradient descent when mini-batch size close to 1. In fact, if $B$ is the mini-batch size and $T$ is the average length of the input sequences, then a network with $n$ hidden units trained using other methods (Vorontsov et al., 2013).
that enforce orthogonality (see Section 2), would have time complexity $O(BTn^2 + n^3)$. Clearly when $BT \gg n$ this becomes $O(BTn^2)$, which is the same time complexity as that of the sRNN and oRNN (with $m = n$). In contrast with the case of fully connected deep forward networks with no weight sharing between layers ($\#$ layer $= L$), the time complexity using our method is $O(BLn^2 + Ln^3)$ whereas other methods discussed in this work (see Section 2) would have time complexity $O(BLn^2 + Ln^3)$. The latter methods are less efficient in this case since $B \gg n$ is less likely to be the case compared with $BT \gg n$ when using SGD.

From a performance point of view, further experiments should be performed to better understand the difference between the unitary versus orthogonal constraint.

**Acknowledgment**

The authors would like to acknowledge Department of State Growth Tasmania for partially funding this work through SenseT.

**References**

Arjovsky, Martin, Shah, Amar, and Bengio, Yoshua. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015.

Chernodub, Artem and Nowicki, Dimitri. Norm-preserving orthogonal permutation linear unit activation functions (oplus). *arXiv preprint arXiv:1604.02313*, 2016.

Gers, F. A., Eck, D., and Schmidhuber, J. Applying LSTM to time series predictable through time-window approaches. In Dorffner, Gerg (ed.), *Artificial Neural Networks – ICANN 2001 (Proceedings)*, pp. 669–676, Berlin, 2001. Springer.

Giles, Mike B. An extended collection of matrix derivative results for forward and reverse mode automatic differentiation. 2008.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pp. 249–256, 2010.

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

Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term Memory. 9(8):1735–1780, 1997.

Hochreiter, Sepp, Bengio, Yoshua, Frasconi, Paolo, and Schmidhuber, Jürgen. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Hyland, Stephanie L and Rätsch, Gunnar. Learning unitary operators with help from $u(n)$. *arXiv preprint arXiv:1607.04903*, 2016.

Jing, Li, Shen, Yichen, Dubcek, Tena, Peurifoy, John, Skirlo, Scott A., Tegmark, Max, and Soljacic, Marin. Tunable efficient unitary neural networks (EUNN) and their application to RNN. *CoRR*, abs/1612.05231, 2016. URL http://arxiv.org/abs/1612.05231.

Joffrain, Thierry, Low, Tze Meng, Quintana-Ortí, Enrique S, Geijn, Robert van de, and Zee, Field G Van. Accumulating householder transformations, revisited. *ACM Transactions on Mathematical Software (TOMS)*, 32(2): 169–179, 2006.

Koutník, Jan, Greff, Klaus, Gomez, Faustino J., and Schmidhuber, Jürgen. A clockwork RNN. *CoRR*, abs/1402.3511, 2014. URL http://arxiv.org/abs/1402.3511.

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.

Mezzadri, Francesco. How to generate random matrices from the classical compact groups. *arXiv preprint math-ph/0609050*, 2006.

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

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL http://arxiv.org/abs/1605.02688.

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

Wisdom, Scott, Powers, Thomas, Hershey, John R, Roux, Jonathan Le, and Atlas, Les. Full-capacity unitary recurrent neural networks. *arXiv preprint arXiv:1611.00035*, 2016.
**A. Proofs**

**Sketch of the proof for Theorem 1.** We need to show that for every $Q' \in O(n)$, there exits a tuple of vectors $(u_1, \ldots, u_n) \in \mathbb{R}^n$ such that $Q' = M_1(u_1, \ldots, u_n)$. Algorithm 1 shows how a QR decomposition can be performed using the matrices $\{H_k(u_k)\}_{k=1}^n$ while ensuring that the upper triangular matrix $R$ has positive diagonal elements. If we apply this algorithm to an orthogonal matrix $Q'$, we get a tuple $(u_1, \ldots, u_n)$ which satisfies

$$QR = H_n(u_n) \ldots H_1(u_1)R = Q'.$$

Note that the matrix $R$ must be orthogonal since $R = Q^TQ'$. Therefore, $R = I$, since the only upper triangular matrix with positive diagonal elements is the identity matrix. Hence, we have

$$M_1(u_1, \ldots, u_n) = H_n(u_n) \ldots H_1(u_1) = Q'.$$

**Algorithm 1** QR decomposition using the mappings $\{H_k\}$.

For a matrix $A \in \mathbb{R}^{n \times n}$, $\{B_{k,k}\}_{1 \leq k \leq n}$ denote its diagonal elements, and $B_{k,k,n} = [B_{k,k}, \ldots, B_{n,n}]^T \in \mathbb{R}^{n \times k+1}$.

**Require:** $A \in \mathbb{R}^{n \times n}$ is a full-rank matrix.

**Ensure:** $Q$ and $R$ where $Q = H_n(u_n) \ldots H_1(u_1)$ and $R$ is upper triangular with positive diagonal elements such that $A = QR$

$$R \leftarrow A$$

$Q \leftarrow I$ \{Initialise $Q$ to the identity matrix\}

for $k = 1$ to $n - 1$

if $R_{k,k} \neq \|R_{k:k,n}\|$ then

$$u_{n-k+1} = [0, \ldots, 0, 1]^T \in \mathbb{R}^{n-k+1}$$

else

$$u_{n-k+1} \leftarrow R_{k,k,n} - \|R_{k:k,n}\| [1, 0, \ldots, 0]^T$$

$$u_{n-k+1} \leftarrow u_{n-k+1} / \|u_{n-k+1}\|$$

end if

$$R \leftarrow H_{n-k+1}(u_{n-k+1})R$$

$$Q \leftarrow QH_{n-k+1}(u_{n-k+1})$$

end for

$u_1 = \text{sgn}(R_{n,n}) \in \mathbb{R}$

$R \leftarrow H_1(u_1)R$

$Q \leftarrow QH_1(u_1)$

where $dA$, $dB$, and $dC$ represent infinitesimal perturbations and

$$\mathcal{C} = \frac{\partial s}{\partial C}, \quad \mathcal{A} = \left[\frac{\partial C}{\partial A}\right]^T \frac{\partial s}{\partial C}, \quad \mathcal{B} = \left[\frac{\partial C}{\partial B}\right]^T \frac{\partial s}{\partial C}.$$

**Proof of Theorem 2.** Let $C = h - UT^{-1}U^T$ where $(U,h) \in \mathbb{R}^{n \times m} \times \mathbb{R}^n$ and $T = \text{striu}(UTU) + \frac{1}{2}\text{diag}(UTU)$. Notice that the matrix $T$ can be written using the Hadamard product as follows

$$T = B \circ (UTU),$$

where $B = \text{striu}(J_m) + \frac{1}{2}I_m$ and $J_m$ is the matrix of all ones.

Calculating the infinitesimal perturbations of $C$ gives

$$dC = (I - UT^{-1}U^T)dh - dUT^{-1}U^T h - UT^{-1} dU^T h + UT^{-1} dUT^{-1}U^T h.$$

Using Equation (16) we can write

$$dT = B \circ (dUTU + U^T dU).$$

By substituting this back into the expression of $dC$, multiplying the left and right-hand sides by $\mathcal{C}^T$, and applying the trace we get

$$\text{Tr}(\mathcal{C}^T dC) = \text{Tr}(\mathcal{C}^T (I - UT^{-1}U^T)dh)$$

$$- \text{Tr}(\mathcal{C}^T dUT^{-1}U^T h) - \text{Tr}(\mathcal{C}^T UT^{-1}dU^T h)$$

$$+ \text{Tr}(\mathcal{C}^T UT^{-1} (B \circ (dUTU + U^T dU)))T^{-1}U^T h).$$

Now using the identity $\text{Tr}(AB) = \text{Tr}(BA)$, where the second dimension of $A$ agrees with the first dimension of $B$, we can rearrange the expression of $\text{Tr}(\mathcal{C}^T dC)$ as follows

$$\text{Tr}(\mathcal{C}^T dC) = \text{Tr}(\mathcal{C}^T (I - UT^{-1}U^T)dh)$$

$$- \text{Tr}(T^{-1}U^T h \mathcal{C}^T dU) - \text{Tr}(h \mathcal{C}^T T^{-1}dUT)$$

$$+ \text{Tr}(T^{-1} U^T h \mathcal{C}^T UT^{-1} (B \circ (dUTU + U^T dU))).$$

To simplify the expression, we will use the short notations

$$\tilde{C} = T^{-1}U^T \mathcal{C},$$

$$\tilde{h} = T^{-1}U^T h,$$

$\text{Tr}(\mathcal{C}^T dC)$ becomes

$$\text{Tr}(\mathcal{C}^T dC) = \text{Tr}(\mathcal{C}^T - \tilde{C}T^{-1}U^T)dh)$$

$$- \text{Tr}(h \tilde{C}^T dU) - \text{Tr}(h \tilde{C}^T dUT)$$

$$+ \text{Tr}(h \tilde{C}T (B \circ (dUTU + U^T dU))).$$

**Lemma 1.** (Giles, 2008) Let $A$, $B$, and $C$ be real or complex matrices, such that $C = f(A,B)$ where $f$ is some differentiable mapping. Let $s$ be some scalar quantity which depends on $C$. Then we have the following identity

$$\text{Tr}(\mathcal{C}^T dC) = \text{Tr}(\mathcal{A}^T dA) + \text{Tr}(\mathcal{B}^T dB),$$

where $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{B}$ are matrices.
Orthogonal Parametrisation of Recurrent Neural Networks

Now using the two following identities of the trace
\[ \text{Tr}(A^T) = \text{Tr}(A), \]
\[ \text{Tr}(A(B \circ C)) = \text{Tr}((A \circ B^T)C)), \]
we can rewrite \( \text{Tr}(C^T \, dC) \) as follows
\[
\text{Tr}(C^T \, dC) = \text{Tr}((C^T - \hat{C}^T U^T) \, dh)
- \text{Tr}(\hat{h}C^T \, dU) - \text{Tr}(h\hat{C}^T dU^T)
+ \text{Tr}((\hat{h}\hat{C}^T \circ B^T)dU^T)
+ \text{Tr}((\hat{h}\hat{C}^T \circ B)U^T dU).
\]

By rearranging and taking the transpose of the third and fourth term of the right-hand side we obtain
\[
\text{Tr}(C^T \, dC) = \text{Tr}((C^T - \hat{C}^T U^T) \, dh)
- \text{Tr}(\hat{h}C^T \, dU) - \text{Tr}(\hat{C}h^T \, dU)
+ \text{Tr}((\hat{C}\hat{h}^T \circ B)U^T dU)
+ \text{Tr}((\hat{h}\hat{C}^T \circ B^T)U^T dU).
\]

Factorising by \( dU \) inside the \( \text{Tr} \) we get
\[
\text{Tr}(C^T \, dC) = \text{Tr}((C^T - \hat{C}^T U^T) \, dh)
- \text{Tr}(\hat{h}C^T \, dU) - \text{Tr}(\hat{C}h^T \, dU)
+ \text{Tr}((\hat{C}\hat{h}^T \circ B + (\hat{h}\hat{C}^T) \circ B^T)U^T \, dU).
\]

Using lemma 1 we conclude that
\[
\begin{align*}
\bar{U} &= U \left[ (\hat{h}\hat{C}^T) \circ B^T + (\hat{C}\hat{h}^T) \circ B \right] - \bar{C}h^T - \bar{h}\hat{C}^T, \\
\bar{h} &= C - U\hat{C}.
\end{align*}
\]

Sketch of the proof for Corollary 1. For any nonzero complex valued vector \( x \in \mathbb{C}^n \), if we chose \( u = x + e^{i\theta} \|x\| e_1 \) and \( H = -e^{-i\theta}(I - 2 \frac{x^H u}{\|u\|^2}) \), where \( x_1 = e^{i\theta}|x_1| \), we have (Mezzadri, 2006)
\[ Hx = \|x\|e_1 \] (17)

Using this fact, a similar approach to the one followed in the proof of Theorem 1 can be used here.