UNITARY EVOLUTION RECURRENT NEURAL NETWORKS

Martin Arjovsky *
Universidad de Buenos Aires
{marjovsky}@dc.uba.ar

Amar Shah*
Cambridge University
{as793}@cam.ac.uk

Yoshua Bengio
Universite de Montréal, CIFAR Senior Fellow
{yoshua.bengio}@gmail.com

ABSTRACT

Recurrent neural networks (RNNs) are notoriously difficult to train. When the eigenvalues of the hidden to hidden weight matrix deviate from absolute value 1, optimization becomes difficult due to the well studied issue of *vanishing* and *exploding* gradients, especially when trying to learn long-term dependencies. To circumvent this problem, we propose a new architecture that learns a unitary weight matrix, with eigenvalues of absolute value exactly 1. We construct an expressive unitary weight matrix by composing several structured matrices that act as building blocks with parameters to be learned. Optimization of this parameterization becomes feasible only when considering hidden states in the complex domain. We demonstrate the potential of this architecture by achieving state of the art in several hard tasks involving very long-term dependencies.

1 INTRODUCTION

Deep Neural Networks have shown increasingly good performance on a wide range of real life problems **??**. However, training very deep models is still a difficult task, and it becomes increasingly difficult as depth is increased. The main issue that surrounds the challenges of training deep networks is the *vanishing* and *exploding* gradients problems described in Bengio et al. (1994). The *exploding* gradient problem results in poor optimization, while the *vanishing* gradient problem results in learning effectively only the last layers of the network.

There has been support in recent literature for using orthogonal weight matrices to help optimization (Saxe et al., 2014) (Le et al., 2015).Orthogonal matrices have the property that they are norm preserving (i.e. $\|\mathbf{W}h\|_2 = \|h\|_2$). This means that you can multiply a vector with orthogonal matrices for as long as you want, without the norm of the vector blowing up or shrinking. In fact, let h_T and h_t be the hidden unit vectors for hidden layers T and t respectively, with $T >> t$ (for example with T the final layer). If C is the cost we are trying to minimize, then the *vanishing* and *exploding* gradient problems refer to the decay or growth of $\frac{\partial C}{\partial h_t}$ as the amount of layers increases (meaning $T \to \infty$). If

$$z_{t+1} = \mathbf{W}_t h_t + \mathbf{V}_t x_{t+1} \tag{1}$$

$$h_{t+1} = \sigma(z_{t+1}) \tag{2}$$

then it's easy to show by the chain rule

$$\frac{\partial C}{\partial h_t} = \frac{\partial C}{\partial h_T}\frac{\partial h_T}{\partial h_t} = \frac{\partial C}{\partial h_T}\prod_{k=t}^{T-1}\frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial C}{\partial h_T}\prod_{k=t}^{T-1}\mathbf{D}_{k+1}\mathbf{W}_k^T \tag{3}$$

Where $\mathbf{D}_{k+1} = diag(\sigma'(z_{k+1}))$ is the Jacobian matrix of the nonlinearity. Since the nonlinearity is evaluated pointwise, this Jacobian matrix is clearly diagonal.

*Indicates first authors. Ordering determined by coin flip.

In the following we will use the norm of matrix to mean the spectral radius (i. e. operator 2-norm) and the norm of a vector to mean L_2-norm. It's good to remember that with these norms, for any matrices $\mathbf{A}, \mathbf{B}$ and vector v we have by definition $\|\mathbf{A}v\| \leq \|\mathbf{A}\| \|v\|$ and $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \|\mathbf{B}\|$.

If the weight matrices $\mathbf{W}_k$ are norm preserving (i.e. orthogonal or unitary), then we can **prove**

$$\left\| \frac{\partial C}{\partial h_t} \right\| = \left\| \frac{\partial C}{\partial h_T} \prod_{k=t}^{T-1} \mathbf{D}_{k+1} \mathbf{W}_k^T \right\| \leq \left\| \frac{\partial C}{\partial h_T} \right\| \prod_{k=t}^{T-1} \|\mathbf{D}_{k+1} \mathbf{W}_k^T\| = \left\| \frac{\partial C}{\partial h_T} \right\| \prod_{k=t}^{T-1} \|\mathbf{D}_{k+1}\| \quad (4)$$

Since $\mathbf{D}_k$ is diagonal, then $\|\mathbf{D}_k\| = \max_{j=1,\dots,n} |\sigma'(h_k^{(j)})|$, with $h_k^{(j)}$ the j-th unit of the k-th hidden layer.

If the absolute value of the derivative σ' can be some number $\lambda > 1$, then this bound is useless, because it results on $\left\| \frac{\partial C}{\partial h_t} \right\| \leq \left\| \frac{\partial C}{\partial h_T} \right\| \lambda^{T-t}$ which grows *exponentially* with $T - t$. This means that we cannot effectively bound $\frac{\partial C}{\partial h_t}$ for deep networks, potentially resulting in *exploding* gradients.

Even worse would be the case where $|\sigma'| < \lambda < 1$, because we would be proving that $\frac{\partial C}{\partial h_t}$ goes exponentially to 0, resulting in certain vanishing gradients. With this argument, the rectified linear unit (ReLU) (Glorot et al., 2011) (Nair & Hinton, 2010) nonlinearity arises naturally. Unless all the activations are killed at one layer (which has never been reported), the maximum entry of $\mathbf{D}_k$ will be 1, leading to $\|\mathbf{D}_k\| = 1$ for all layers k. With this, we effectively prove that

$$\left\| \frac{\partial C}{\partial h_t} \right\| \leq \left\| \frac{\partial C}{\partial h_T} \right\| \prod_{k=t}^{T-1} \|\mathbf{D}_{k+1}\| = \left\| \frac{\partial C}{\partial h_T} \right\| \quad (5)$$

What is most important is the fact that this result is independent from the deep of the network or $T - t$. For the case of recurrent networks, it's easy to show that this results on the fact that $\frac{\partial C}{\partial \theta}$ can grow at most *additively* instead of exploding *multiplicatively*. Among other things, this implies that the use of gradient clipping (Pascanu et al., 2010) is no longer required.

To the best of our knowledge, this is the first time a deep neural network has been mathematically proven to have no exploding gradients.

2 UNITARY EVOLUTION RNNS

The most important feature of unitary and orthogonal matrices for our purpose is that they have eigenvalues λ_j with absolute value 1 (we therefore write $\lambda_j = e^{iw_j}$ 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

$$\mathbf{W} = \mathbf{V}\mathbf{D}\mathbf{V}^*$$

where $\mathbf{D}$ is a diagonal such that $\mathbf{D}_{j,j} = e^{iw_j}$. By restricting the choice of $\mathbf{V}$ and tying the learned w_js to create conjugates, $\mathbf{W}$ can be restricted to be a real matrix. This is evident from the following lemma:

Lemma 1. *A complex square matrix $\mathbf{W}$ is unitary if and only if it has an eigendecomposition of the form $\mathbf{W} = \mathbf{V}\mathbf{D}\mathbf{V}^*$. 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 an eigenvalue $\lambda_k = \overline{\lambda_j}$ with corresponding eigenvector $v_k = \overline{v_j}$*

Proof. See **??** □

To achieve a real orthogonal matrix $\mathbf{W}$ we require a fixed unitary matrix $\mathbf{V}$, whose columns come in complex conjugate pairs $v_k = \overline{v_j}$. Similarly, we should also tie $w_k = -w_j$ in order to achieve $e^{iw_j} = \overline{e^{iw_k}}$. $\mathbf{W}$ is subsequently real and orthogonal by the lemma. Since w_j are real numbers and if the cost is differentiable with respect to them, we can learn these weights by gradient descent.

However, this approach has one major problem. Mainly, the memory needed to store $\mathbf{V}$ is $\mathcal{O}\left(n^2\right)$. Also, if u is a vector, calculating $\mathbf{V}u$ has cost $\mathcal{O}\left(n^2\right)$. This would lead to a cost of $\mathcal{O}\left(n^2\right)$ for forward and backpropagation. However, we are only learning $\mathcal{O}(n)$ parameters, which is very inefficient.

If we didn't have $\mathbf{V}$ then we would just be learning a diagonal matrix, which likely isn't sufficient. However, since the product of unitary matrices is itself unitary, we can construct $\mathbf{W}$ by combining several simple structured unitary matrices that are efficient to store and to operate with. In **??** they do a similar construction to reduce the amount of parameters by more than an order of magnitude in an industrial sized network, while maintaining performance. This, combined with earlier work by **??** suggests that it is possible to create very expressive matrices with relatively little cost. The building blocks we use are as follows:

- $\mathbf{D}$ is a diagonal matrix with entries $\mathbf{D}_{j,j} = e^{iw_j}$ Since $\mathbf{D}$ is a diagonal with that have absolute value 1, it is clearly unitary. The weights $w_j \in \mathbb{R}$ are going to be learned by gradient descent.
- $\mathbf{R} = \mathbf{I} - 2\frac{vv^*}{\|v\|^2}$ is a reflection matrix by the complex vector $v \in \mathbb{C}^n$. This matrix is also learned, by doing gradient descent on the real and imaginary parts of v. The fact that this matrix is unitary is left as an exercise to the reader.
- $\mathbf{\Pi}$ a fixed random permutation. Since the transpose is the inverse permutation, the unitary condition follows.
- $\mathcal{F}$ and $\mathcal{F}^{-1}$ the Fourier and inverse Fourier transforms, which have long been showed to be unitary.

For the case of $\mathbf{D}, \mathbf{R}, \mathbf{\Pi}$, storing each of this matrices has cost $\mathcal{O}(n)$. For the diagonal, we only store the weights w_j, for the reflection only the vector v and for the permutation we save it as an array. Multiplying any of this matrices with a vector is also $\mathcal{O}(n)$. The Fourier transforms don't need to be stored, and multiplying a vector with them can be done in $\mathcal{O}(n \log n)$ via the Fast Fourier Transform. This means that the final cost of doing forward and backpropagation through the weight matrix is $\mathcal{O}(n \log n)$, with memory cost $\mathcal{O}(n)$ with $\mathcal{O}(n)$ parameters and hidden units.

One major advantage of this parameterization, is that the number of parameters, memory and computational cost increase linearly as a function of the hidden layer size. Therefore, we can potentially have inmense hidden layers, while this would be impossible in traditional RNNs. Especially for modelling long term dependencies, this is not a minor feature, because it increases the amount of information that can be carried from one time to another. Early work by Bengio et al. (1994) shows that having a big memory may be a crucial aspect of solving the difficulties in modeling long-term dependencies.

We call any architecture that uses a unitary hidden to hidden matrix a unitary evolution RNN (uRNN). After trying different combinations, the uRNN we settled on has the weight matrix

$$\mathbf{W} = \mathbf{D_3}\mathbf{R_2}\mathcal{F}^{-1}\mathbf{D}_2\mathbf{\Pi}\mathbf{R_1}\mathcal{F}\mathbf{D}_1$$

The main issue of most of this matrices (except the permutation) is that they are complex. However, they are parameterized with real weights, so if the final cost is real and is differentiable with respect to them, we can do gradient descent nonetheless. In the following section we specify the complete architecture of the network, and explain how the potential difficulties of using complex hidden units can be easilly bypassed.

3 ARCHITECTURE DETAILS

It is not enough to say how to parameterize a unitary weight matrix and get done with it. In this section, we describe the rest of the architecture, including the nonlinearity we used, how we go from real input to complex hidden units and from these units to the output. Finally, we display some implementation considerations, including memory optimization on GPUs.

3.1 COMPLEX HIDDEN UNITS

All the hidden units are represented internally with real numbers, as imaginary and complex parts. This way, it is far more easy to code, and we completely avoid the lack of support for complex numbers by most deep learning frameworks. One of the most important examples is when multiplying the weight matrix $\mathbf{W} = \mathbf{A} + i\mathbf{B}$ by the complex hidden vector $h = x + iy$ In fact, it is easy to show that $\mathbf{W}h = (\mathbf{A}x - \mathbf{B}y) + i(\mathbf{A}y + \mathbf{B}x)$, which leads to

$$\begin{pmatrix} Re(\mathbf{W}h) \\ Im(\mathbf{W}h) \end{pmatrix} = \begin{pmatrix} \mathbf{A} & -\mathbf{B} \\ \mathbf{B} & \mathbf{A} \end{pmatrix} \begin{pmatrix} Re(h) \\ Im(h) \end{pmatrix}$$

More generally, let $f : \mathbb{C}^n \to \mathbb{C}^n$ be any complex function and $z = x + iy$ any complex vector, it can be decompposed as $f(z) = u(x, y) + iv(x, y)$ where u, v are the real and imaginary parts of f. Therefore, we can consider the mapping as $f : \mathbb{R}^{2n} \to \mathbb{R}^{2n}$, and implement everything with real numbers. Doing this, any Deep Learning framework with automatic differentiation like Theano (Bergstra et al., 2010) is able to take care of computing the derivatives efficiently.

3.2 INPUT TO HIDDEN AND THE NONLINEARITY

As is the case with most recurrent networks, our uRNN follows the same hidden to hidden mapping as (1). For the input matrix we have $\mathbf{V} \in \mathbb{C}^{n \times n}$, and it's real and complex parts are learned as parameters. For completeness, the initial hidden state $h_0 \in \mathbb{C}^n$ is also learned

Sadly, it is not as easy to choose a good nonlinearity. As discussed in section 1, using a ReLU is the natural choice when having an orthogonal or unitary RNN. Previous work on complex RNNs usually have applied a nonlinearity on both the real and imaginary parts **??**. However, as we will show later, this usually results on poor performance. The interpetation of why this happens is that applying a nonlinearity on the real and imaginary parts brutally changes the phase of a complex number. Since unitary matrices can be interpreted as multiplying by a phase in each eigenvector direction, this phase might be important when recovering information about past memories. Therefore, we employ a variation of the ReLU that we deem modReLU, that is applied only on the absolute value, and is defined as follows.

$$\sigma : \mathbb{C} \to \mathbb{C}$$

$$\sigma(z) = \begin{cases} (|z| + b)\frac{z}{|z|} & \text{if } |z| + b \geq 0 \\ 0 & \text{if } |z| + b < 0 \end{cases}$$

Where $b \in \mathbb{R}$ is the bias of the nonlinearity, which is learned for each complex unit. It is important to note that $\sigma(z) = \text{ReLU}(|z| + b)\frac{z}{|z|}$

3.3 HIDDEN TO OUTPUT

For the output we have a matrix $\mathbf{U} \in \mathbb{R}^{2n_h \times n_o}$ where n_h is the hidden layer size and n_o is the output size. The output is calculated as

$$o_t = \mathbf{U} \begin{pmatrix} Re(h_t) \\ Im(h_t) \end{pmatrix} + b_o$$

with $b_o \in \mathbb{R}^{n_o}$ the output bias. Since this output is real, we can now use it to compare the cost in the classical way with any loss function (e.g. plug it into a softmax for classification).

3.4 INITIALIZATION

We experimented with several initialization strategies. However, because of the stability of the network, initialization doesn't seem to be extremely important. However, the following ones did seem to make optimization marginally easier, and give a few insights on the way the netwrok works.

- We initialize $\mathbf{V}$ and $\mathbf{U}$ (the input and output matrices) as in Glorot & Bengio (2010).
- The biases are initialized to 0. This way at the beginning, the nonlinearity is silent. This is because the input of the ReLU is always positive (it is an absolute value). Therefore, we start with a linear unitary mapping with additive contributions from the input, which is controled and seems to help early optimization (Le et al., 2015).
- The reflection vectors for $\mathbf{R}_1$ and $\mathbf{R}_2$ are initialized coordinate-wise from a uniform $\mathcal{U}[-1, 1]$. It's important to remark the fact that since the reflection matrix is independent from the scale of the vector, so any centered uniform will result in the same network.
- The diagonal weights w_j are sampled from a uniform $\mathcal{U}[-\pi, \pi]$. This way $\mathbf{D}_{j,j} = e^{iw_j}$ is sampled uniformly from the unit circle.
- We initialize h_0 with a uniform $\mathcal{U}[-\sqrt{\frac{3}{2n_h}}, \sqrt{\frac{3}{2n_h}}]$. This way $\mathbb{E}\left[\|h_0\|^2\right] = 1$. Since the norm of the hidden units is roughly preserved because of the unitary weight matrix and inputs are usually whitened to have norm 1, we have hidden states and inputs on the same order of magnitude, which seems to help optimization **??**.

REFERENCES

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEE Transactions on Neural Networks*, 5, 1994.

Bergstra, James, Breuleux, Olivier, Bastien, Frédéric, Lamblin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010.

Glorot, Xavier, Bordes, Antoine, and Bengio, Yoshua. Deep sparse rectifier neural networks. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

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

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. *International Conference on Machine Learning*, 2010.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. *International Conference on Machine Learning*, 2010.

Saxe, Andrew M., McLelland, James L., and Ganguli, Surya. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *International Conference in Learning Representations*, 2014.