Convergence Analysis of Initializing Recurrent Neural Network Weights under Unitary Group

Hoang Chu
MATH 179, Harvey Mudd College
Claremont, CA, 91711
hoachu@students.pitzer.edu

Abstract

A popular issue in training recurrent neural networks is the vanishing / exploding gradient problem. Other improved models like LSTM suffer from the limitation of complexity where such models stop being able to learn after a certain amount of time, and they're computationally expensive. I proved that the exploding and vanishing gradient problems in RNN can be further preventable when we restrict the weight matrices to be within the unitary group, and it's possible to do that restriction from gradient descent with a modified update step from experiments in topic modelling tasks.

1 Background and Related Work

The phenomena of vanishing and exploding gradients are closely tied to the eigenvalues of the recurrent weight matrix W_{rec} in a RNN. Pascanu et al. suggest, denoting λ_1 as the largest magnitude of the eigenvalues of W_{rec}, that $\lambda_1 < \frac{1}{\gamma}$ is a sufficient condition for vanishing gradients and that $\lambda_1 > \frac{1}{\gamma}$ is a necessary condition for exploding gradients, where γ is the upper bound on the magnitude of derivative σ' of the activation function used in the RNN. Letting $\boldsymbol{x}_t$ be the hidden state at time t, they observe that

$$\frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} = \prod_{t \geq i > k} \frac{\partial \boldsymbol{x}_i}{\partial \boldsymbol{x}_{i-1}} = \prod_{t \geq i > k} W_{\text{rec}}^T \operatorname{diag}\left(\sigma'\left(\boldsymbol{x}_{t-1}\right)\right)$$

such that by properties of the Frobenius norm,

$$\left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\| \leq \prod_{t \geq i > k} \left\| W_{\text{rec}}^T \right\| \left\| \operatorname{diag}\left(\sigma'\left(\boldsymbol{x}_{t-1}\right)\right) \right\| \leq \gamma^{t-k} \prod_{t \geq i > k} \left\| W_{\text{rec}}^T \right\|$$

They then claim that if λ_1 of W_{rec} (and hence W_{rec}^T) is smaller than $\frac{1}{\gamma}$,

$$\left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\| \leq \gamma^{t-k} \prod_{t \geq i > k} \left\| W_{\text{rec}}^T \right\| \leq \left(\gamma \lambda_1\right)^{t-k}$$

such that the left-hand-side tends to zero for large t as $\gamma \lambda_1 < 1$. However, this proof is fallacious as the authors assumed $\left\| W_{\text{rec}}^T \right\| \leq \lambda_1$ when in fact, $\left\| W_{\text{rec}}^T \right\| \geq \lambda_1$ as the Frobenius norm is bounded below by the operator norm which is then bounded below by λ_1. That said, their conclusions turn out to be correct and a revised proof will be provided in Section 2.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2 Theoretical Guarantees

In this section, we illustrate the relationship between the eigenvalues of W_{rec} and the phenomena of vanishing and exploding gradients. We will use the Frobenius norm $\|W\|_F = \sqrt{\text{trace}\,(W^\dagger W)} = \sqrt{\sum_{i,j} |W|_{ij}^2}$ and the operator norm $\|W\|_2 = \sup_{\|\boldsymbol{x}\|=1} \|W\boldsymbol{x}\|_2$ where $\|W\boldsymbol{x}\|_2$ is the standard vector 2-norm of $W\boldsymbol{x}$. In most cases, this distinction is irrelevant and the norm is denoted as $\|W\|$. The following lemmas will be useful.

Lemma 1. For a given diagonal matrix D and an arbitrary matrix A in $\mathbb{C}^{n\times n}$, we have

$$\left(\min_i |D_{ii}|\right) \|A\| = \underline{D}\|A\| \leq \|DA\| \leq \bar{D}\|A\| = \left(\max_i |D_{ii}|\right) \|A\|$$

where $\underline{D}$ and $\bar{D}$ are the minimum and maximum element-wise norms of D respectively. The proof is straightforward from the definitions of the Frobenius and operator norms.

Lemma 2. If $A \in \mathbb{C}^{n\times n}$ is given by

$$A = U_1 D_1 U_2 D_2 \dots U_n D_n$$

for unitary matrices $U_i \in \mathbb{C}^{n\times n}$ and diagonal matrices $D_i \in \mathbb{C}^{n\times n}$. Then,

$$\|I_{n\times n}\| \prod_{i=1}^{n} \underline{D}_i \leq \|A\| \leq \|I_{n\times n}\| \prod_{i=1}^{n} \bar{D}_i$$

where $I_{n\times n}$ is the identity matrix such that $\|I_{n\times n}\|_F = \sqrt{n}$ and $\|I_{n\times n}\|_2 = 1$.

Proof. Firstly, note that for any unitary matrix U, $\|AU\| = \|A\| = \|UA\|$ for an arbitrary matrix A. Thus,

$$\begin{aligned}
\left\| \prod_{i=1}^{n} U_i D_i \right\| &= \left\| U_1 D_1 \prod_{i=2}^{n} U_i D_i \right\| \\
&= \left\| D_1 \prod_{i=2}^{n} U_i D_i \right\| \\
&\leq \bar{D}_1 \left\| \prod_{i=2}^{n} U_i D_i \right\| \\
&\leq \cdots \\
&\leq \prod_{i=1}^{n} \bar{D}_i \|I_{n\times n}\|
\end{aligned}$$

where we have applied Lemma 1 in the third step. Following a similar argument produces a corresponding lower bound.

3 General Bounds for Diagonalizable Matrices

Although the below theorem only holds for diagonalizable W_{rec}, note that it is a representative analysis of the evolution of RNNs as diagonalizable matrices are dense in $\mathbb{C}^{n\times n}$ such that RNNs are highly likely to encounter diagonalizable matrices during their training. Theorem 1. Suppose $W_{\text{rec}} \in \mathbb{C}^{n\times n}$ (and thus $W_{\text{rec}}^\dagger$) is diagonalizable with smallest and largest eigenvalue norms $\underline{\lambda}$ and $\bar{\lambda}$. Furthermore, assume $\beta \leq |\sigma'(\boldsymbol{x})| \leq \gamma$ is bounded. Then,

$$(\underline{\lambda}\beta)^{t-k} \|I_{h\times h}\| \leq \left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\| \leq (\bar{\lambda}\gamma)^{t-k} \|I_{h\times h}\|$$

where h is the number of units in each hidden layer.

Proof. Since $W_{rec}^{\dagger}$ is diagonalizable, $W_{rec}^{\dagger} = U^{\dagger}DU$ for some unitary U and diagonal D. Thus, Equation 1 (with $W_{\text{rec}}^{\dagger}$ in place of W_{rec}) yields

$$\left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\| = \left\| \prod_{t \geq i > k} U^{\dagger} D U \operatorname{diag}\left(\sigma'\left(\boldsymbol{x}_{t-1}\right)\right) \right\|$$

after which an application of Lemma 2 gives the result.

Table 1 summarizes the implications of Theorem 1 in different regimes of $\lambda\beta$ and $\bar{\lambda}\gamma$. If $\underline{\lambda}\beta > 1$, $\lim_{t\to\infty} \left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\| = +\infty$ so the gradient explodes. For most activation functions, $\beta = 0$ so the lower bound is, in fact, not particularly edifying. On the other hand, the upper bound shows that if $\bar{\lambda}\gamma < 1$, $\lim_{t\to\infty} \left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\| = 0$ so the vanishing gradients problem occurs. Even if we wisely choose $\bar{\lambda}\gamma \geq 1$, there is still no guarantee that $\left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\|$ does not vanish (i.e. the previous statement is not a necessary condition for vanishing gradients). However, our following approach makes use of both these upper and lower bounds. To the best of our knowledge, this is the first occasion of a network with a proven lower bound on the norm of gradients between hidden layers.

Observe that if we require $\bar{\lambda}\gamma = \underline{\lambda}\beta$, $\left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\|$ is confined to be $(\bar{\lambda}\gamma)^{t-k} \left\| I_{h\times h} \right\|$. To make this expression independent of t and k such that gradients can neither vanish nor explode, we just need to ensure $\bar{\lambda}\gamma = 1$. Natural candidates for W_{rec} are then unitary matrices for which $\bar{\lambda} = \underline{\lambda} = 1$. Subsequently, we need $\beta = \gamma = 1$ - a specific example of such an activation function would be $f(x) = |x|$ but we surmise that the other functions in Table 2 could be effective too (desirable properties are in bold).

Low. / Up.	$\bar{\lambda}\gamma < 1$	$\bar{\lambda}\gamma = 1$	$\bar{\lambda}\gamma > 1$
$\underline{\lambda}\beta < 1$	0	0^*	$0/\infty^*$
$\underline{\lambda}\beta = 1$	–	Constant	∞^*
$\underline{\lambda}\beta > 1$	–	–	∞

σ/λ	$\underline{\lambda}$	λ
$\lvert x \rvert$	$\mathbf{1}$	$\mathbf{1}$
ReLU	0	$\mathbf{1}$
Leaky ReLU	$[\mathbf{0}, \mathbf{1}]$	$\mathbf{1}$

Table 1: $\lim_{t\to+\infty} \left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_k} \right\|$ in different regimes of $\lambda\beta$ Table 2: Eigenvalues of various activaand $\bar{\lambda}\gamma$.$*$ indicates the possibility of the limit occuring. tion functions.

4 Approach

In Arjovsky et al. [2015], the idea of a Unitary Evolution Recurrent Neural Network (uRNN) was introduced. In a bid to solve the vanishing and exploding gradient problems, the recurrent weight matrix is constrained to be unitary (or equivalently, assuming all entries are real, orthogonal. We take a similar approach to Wisdom et al. [2016], except that the orthogonality of the weight matrix will be preserved by an alternative update rule described in Plumbley [2004]:

$$\begin{aligned} B &\leftarrow B + \eta \left(\left(\nabla_{W_{\text{rec}}} J\right) W_{\text{rec}}^T - W_{\text{rec}} \left(\nabla_{W_{\text{rec}}} J\right)^T \right) \\ W &\leftarrow \exp(B) \end{aligned}$$

where η is the step size and exp is the matrix exponentiation defined by

$$\exp B = I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \dots$$

In our implementation, the exp function is computed efficiently via an eigenvalue decomposition. For an anti-symmetric matrix B (such as the case in the update above), by the complex spectral theorem, $B = U^{-1}DU$ for some unitary U and diagonal D. Then,

$$\begin{aligned}
\exp B &= I + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \dots \\
&= I + U^{-1}DU + \frac{U^{-1}D^2U}{2!} + \frac{U^{-1}D^3U}{3!} + \dots \\
&= U^{-1}\left(I + D + \frac{D^2}{2!} + \frac{D^3}{3!} + \dots\right)U \\
&= U^{-1}e^{D}U
\end{aligned}$$

where e^D is just the element-wise exponentiation for a diagonal matrix D. Thus, the overall time complexity of this matrix exponentiation is the time complexity of eigenvalue decompositions which can be performed efficiently for Hermitian matrices on CUDA (the anti-symmetric B can be transformed into a Hermitian matrix through multiplication by i after which the eigenvalues can be retrieved through multiplication by $-i$ while the eigenvectors remain the same).

Our implementation of the uRNN is built on top of the existing implementation of RNNs in PyTorch with the ReLU activation function. The major modifications are the introduction of the parametrization matrix B, whose gradient at each time step is manually updated as $\nabla_B J = \nabla_W J W^T - W\left(\nabla_W J\right)^T$. This allows for easy compatibility with optimizers with adaptive learning rates (such as Adam).

The overall architecture we will be using may be described as follows: the number sequence (resp. word sequence) is first fed through an embedding layer, then the resulting embeddings are fed into the RNN (resp. uRNN). The output from the RNN (resp. uRNN) is then passed through a linear projection layer after which dropout is applied. Finally, loss is calculated by taking the cross-entropy between the output of the linear projection layer and the ground truth encoded as a one-hot vector. During the validation and testing stages of the toy experiments, accuracy is also calculated based on the argmax of the final state.

The baseline we will be comparing against is a vanilla RNN. We will first test the conceptual validity of uRNN by running comparative experiments on various toy tasks. Subsequently, we will construct a language model using a uRNN, as well as other variants such as a unitary Gated Recurrent Unit (uGRU), based largely on the Gated Recurrent Units (GRU) introduced in Cho et al. [2014]. In particular, given the update rules:

$$\begin{aligned}
\boldsymbol{u}_t &= \sigma\left(W_u\boldsymbol{x}^{(t-1)} + U_u\boldsymbol{e}^{(t)} + \boldsymbol{b}_u\right) && \text{(update gate)} \\
\boldsymbol{r}_t &= \sigma\left(W_r\boldsymbol{x}^{(t-1)} + U_r\boldsymbol{e}^{(t)} + \boldsymbol{b}_r\right) \\
\tilde{\boldsymbol{x}}^{(t)} &= \tanh\left(W_h\boldsymbol{x}^{(t-1)} + U_h\boldsymbol{e}^{(t)} + \boldsymbol{b}_h\right) && \text{(reset gate)} \\
\boldsymbol{x}^{(t)} &= \left(1 - \boldsymbol{u}^{(t)}\right) \circ \boldsymbol{x}^{(t-1)} + \boldsymbol{u}^{(t)} \circ \tilde{\boldsymbol{x}}^{(t)} && \text{(generated hidden state)}
\end{aligned}$$

where $\boldsymbol{x}^{(t)}, \boldsymbol{e}^{(t)}$ denote the hidden state and word embeddings (input) respectively, only the W_h matrix is kept orthogonal for the uGRU.

5 Toy Experiments

6 Tasks and Setup

To test the practical validity of our approach, the following toy experiments were devised to test whether the model performs favourably on long inputs:

I. Given a sequence of N numbers from 0 to $M - 1$ inclusive, the task is to recall the first number of the sequence.

II. Given a sequence of N numbers from 0 to $N - 1$ inclusive, the task is to find the k-th largest number of the sequence. These tasks were tested against a vanilla RNN model and a uRNN model. Both setups used an Adam optimizer as well as a Dropout layer. All parameters used were identical

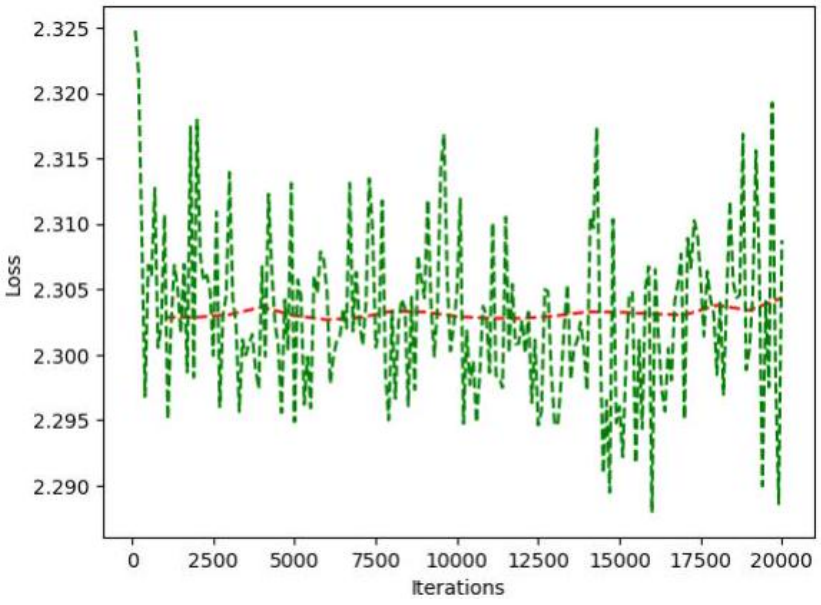
(see Table 8 so that we can get an unbiased comparison between both models. The datasets are randomly generated based on original code, with the same dataset used by both systems.

7 Results and Empirical Observations

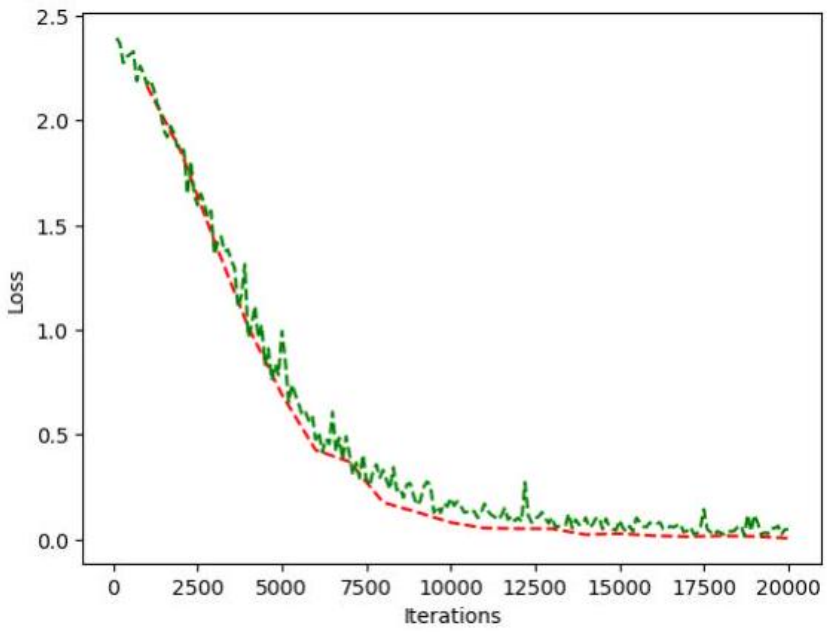
Generally, the uRNN system outperformed the vanilla RNN system by a large margin on long inputs (i.e. $N \geq 50$). This fits with our theoretical predictions that a larger number of time steps will have a correspondingly severe impact on the relative gradients. Despite each batch taking around 2-3 times longer (where the exponential operations were ran on CPU), the uRNN is often more efficient than the RNN in terms of training speed.

Loss / N	20	50	100	Acc. $(\%)/N$	20	50	100
RNN	2.33	2.33	2.33	RNN	10.22	10.56	0.936
uRNN	0.01*	0.01*	0.01*	uRNN	99.98*	100.00*	100.00*

Table 3: Final test losses and accuracy for experiment I. * indicates that training was stopped early due to loss threshold.



(a) Vanilla RNN, 20 epochs



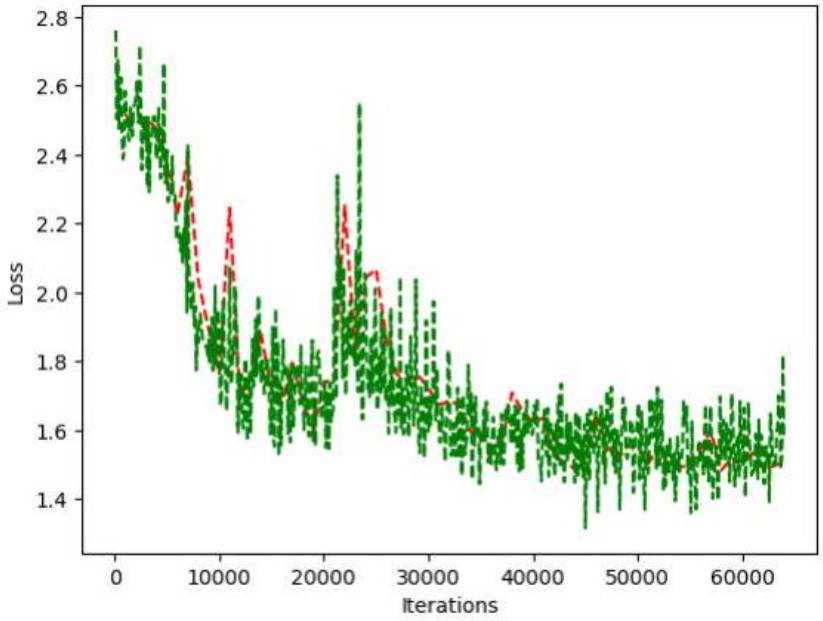
(b) uRNN, 20 epochs

Figure 1: Loss comparison for experiment I with $M = 10$ and $N = 100$. Test (green) and dev (red) losses indicated.

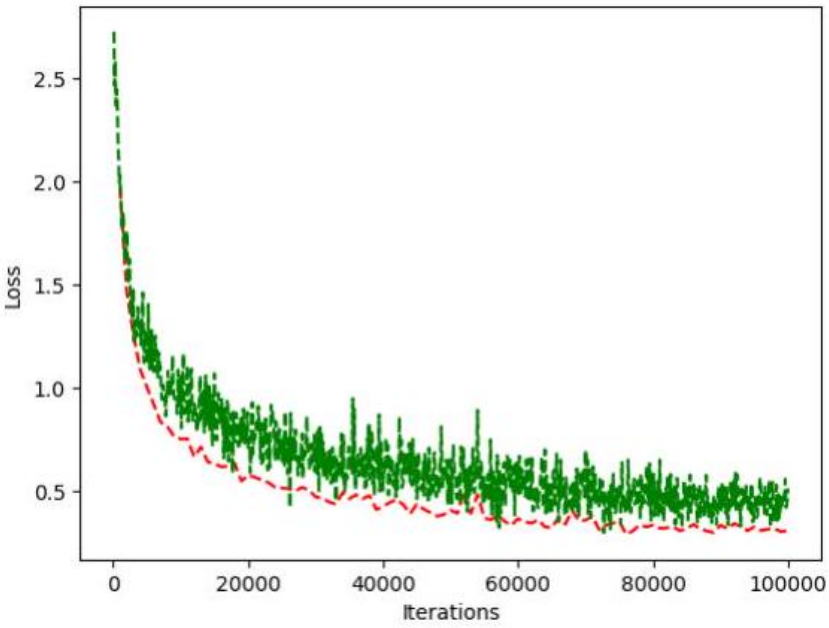
For experiment I, the vanilla RNN performed much worse than expected. From the accuracy levels, we may infer that it fails to solve the problem for even the smallest testing value of $N = 20$.

Loss $/N$	100	200	400	Acc. $(\%)/N$	100	200	400
RNN	1.52	2.12	1.20				
	0.31	0.21	0.10				
	uRNN	uRNN	98.12	34.15	15.23		

Table 4: Final test losses and accuracy for experiment II. Datapoints marked with $*$ indicates that training was stopped early due to loss threshold being reached.



(a) Vanilla RNN, run until completion



(b) uRNN, run until completion

Figure 2: Loss comparison for experiment II ($N = 100, k = 10$). Test (green) and dev (red) losses indicated.

For experiment II, the uRNN shows a clear advantage both in training speed and test results, even though it too starts to falter as N increases. Interestingly, in this case the RNN is able to make some significant progress towards solving the problem in the case of $N = 100$, whereas it made negligible progress in the previous toy problem (evident because accuracy hovers around 10%, which is the expected accuracy if the answers were random). The second problem is easier in the sense that the k-th largest number is expected to be large, hence it suffices for the RNN to simply keep track of the largeest numbers.

8 Language Modelling Experiment

9 Experimental Details

We adapted the word language model from the Pytorch Examples repository to run our uRNN model on the Penn Treebank dataset. The parameters used are summarized in Table 9 below, where the backpropagation length $(BPTT)$ refers to the maximum number of backward timesteps that gradients will reach.

Firstly, we experimented with a uRNN which relied on custom implementations of the activation functions stated in Table 2, After we determined that the uRNN indeed has its merits over the conventional RNN, we observed that its performance still strayed from the state-of-the-art and thus decided to adopt a more modern architecture. The ability of the uRNN to preserve "long-term" information immediately beckons an entity that decides the relevant pieces to store - leading us to turn to a Gated Recurrent Unit (GRU). We implemented a unitary version of a GRU (termed uGRU) which employs a unitary recurrent weight matrix while the update and reset matrices are unconstrained. The tanh activation function was adopted for compatibility with the Pytorch library.

Our baseline was a vanilla RNN for experiments with custom activation functions and a vanilla GRU for our revised experiment, with identical set-ups.

During the course of training, we realized that our gradients for W_{rec} sometimes exploded and hence decided to clip solely the gradient of W_{rec} to a maximum element-wise magnitude of 0.25, while leaving other parameters untouched.

10 Results of Custom Activations

Table 5 depicts the best validation and test perplexities attained by the RNN and uRNN. Surprisingly, the uRNN is able to outperform the RNN in two cases, despite a restricted search space of recurrent weight matrices.

Model / Activation	$\lvert x\rvert$	ReLU	Leaky ReLU ($\lambda = 0.5$)
RNN	$784.53/773.91$	$754.65/705.28$	$\mathbf{756.48/712.88}$
uRNN	$\mathbf{717.12/613.89}$	$\mathbf{720.88/666.93}$	$869.21/810.21$

Table 5: Best validation and test perplexities in language modelling.

Bounds / Activation	$\lvert x\rvert$	ReLU	Leaky ReLU ($\lambda = 0.5$)
Lower	$\sqrt{100} = 10$	0	$0.5^{30} = 9.3 \times 10^{-10}$
Upper	$\sqrt{100} = 10$	$\sqrt{200} = 14.14$	$\sqrt{200} = 14.14$

Table 6: Theoretical predictions of range of gradient norms for different activations. $|x|$ experiment was conducted with 100 hidden units while ReLU and Leaky ReLU experiments were conducted with 200 hidden units. BPTT and thus maximum sequence length were 30 for all experiments.

For the $|x|$ activation function, the gradients are ensured to be of constant magnitude (see Table 6 for the uRNN so it is likely able to exceed the performance of the RNN due to better training of parameters. This advantage is especially salient in this case as the non-monotone property of $|x|$ introduces further complexities in training. On the other hand, the lower bound $\left(0.5^{BPTT} = 9.3 \times 10^{-10}\right)$ imposed by the Leaky ReLU on gradient norms is virtually zero such that the uRNN is still prone to vanishing gradients. This, coupled with the weaker non-linearity of the Leaky ReLU activation, provides a possible explanation for its worse performance as the space of possible functions it can represent is severely restricted. Finally, the strong non-linearity of the ReLU function enables the uRNN to outperform the RNN despite a smaller search space, possibly because language modelling inherently prefers unitary matrices which preserve long-term relationships and because the uRNN gradients still tend to be larger despite the lack of a guaranteed lower bound.

Figure 4 on the next page depicts the Frobenius norms of the relative gradients between the last and first hidden layers $\left\| \frac{\partial \boldsymbol{x}_t}{\partial \boldsymbol{x}_1} \right\|_F$ across batches in epochs 7 and 100 for our uRNN and vanilla RNN. Overall, the gradient norms of both RNNs, with ReLU and Leaky ReLU activations, decayed with

the number of epochs but the gradient norms of the uRNN were generally two orders of magnitude larger than those of the vanilla RNN. The $|x|$ activation, on the other hand, indeed fixed the uRNN's gradient norm at a constant value.

For the uRNN which was run with ReLU in our experiment, the gradient norm was roughly level for the early epochs of training as illustrated in Fig. 4a. However, the gradient norm started to fluctuate around epoch 60 until it eventually decayed to a multitude of small spikes around epoch 100 as shown in Fig. 4b. A possible explanation for the existence of spikes on an otherwise flat plateau at zero norm is that towards the later stages of training, most hidden units were "dead" after the ReLU function, causing most gradients to vanish other than an occasional spike.

For the vanilla RNN, the gradient norms always fluctuated and were generally much smaller than the gradient norms in the uRNN in the same epoch, suggesting that it is more prone to vanishing gradients as seen from Figs. 4e and 4f. Similar observations were made for the Leaky ReLU function.

For the $|x|$ activation function, the gradient norms of the uRNN was level at the predicted value of 10 even at epoch 100 , as seen from Fig 4e, with a difference on the order of 10^{-5} probably due to numerical errors. Meanwhile, the gradients of the vanilla RNN continued to fluctuate. All-in-all, our empirical results were coherent with our theoretical predictions in Section 2 in all cases. Although the bounds were most meaningful in the case of $|x|$, we surmised that a Leaky ReLU with $\underline{\lambda} \geq 0.95$ could be effective too and thus plotted its gradient norms for a small subset of the training data in Fig 4f. As expected, the gradient norm indeed fell within the predicted range demarcated in green.

Finally, we generated random sentences using our trained uRNN through greedy decoding. Most sentences appear to flow smoothly at first glance but upon closer inspection, one realizes that though different sub-phrases appear logical in isolation, the sentences are still incoherent as a whole - an observation that holds for most language models. However, our model is, intriguingly, able to produce long sentences (Figure 3) which mimic the semantic structure of natural text, suggesting an ability in identifying long-term relationships in certain limited cases. this talk would the most important guidelines by now for our benefit capital which makes about N N of the key machines related a rate for customer democrats morning while the number of shared ones also with best united states now be counted following midnight night

Figure 3: Example sentence generated by uRNN

11 Results of uGRU

Our results for the uGRU experiments are summarized in Table 7. Notably, the uGRU was able to achieve a test perplexity below 100 despite its additional constraints.

The uGRU performed slightly worse than the GRU with $BPTT = 30$ but better with $BPTT = 150$. The fact the discrepancy in perplexity for $BPTT = 150$ outweighs that with $BPTT = 30$ bears credence to our hypothesis that the uGRU will be advantageous in tasks that require propagating gradients over long temporal periods.

Over the course of training, the uGRU was also remarkably only $2ms$ ($1 - 10\%$ depending on batch size) slower in training speed per batch due to our custom implementation of the otherwise costly matrix exponential operation in CUDA. This suggests that our approach is scalable and feasible in more complex set-ups.

Model / BPTT	35	150
GRU	$\mathbf{100.80/97.25}$	$108.33/105.34$
uGRU	$101.76/98.05$	$\mathbf{105.05/102.07}$

Table 7: Best validation and test perplexities in language modelling for GRU and uGRU.

12 Discussion

In this paper, we establish the first instance of a recurrent architecture with a proven lower bound on gradient norms that has been empirically verified. The restriction of recurrent weight matrices to unitary matrices not only solves the vanishing and exploding gradient problems but also leads

to improvements in both pathological synthetic tasks and the natural task of language modelling, despite a smaller search space. This suggests that tasks which depend on long-term contexts are fundamentally suited to be modelled by unitary recurrent networks.

Moreover, the advantages ascribed by our implementation only come at a negligible decrease in training speed, meaning that it can be readily deployed in practice.

Finally, we conjecture that the benefits of our unitary models stem from the superior training of parameters, facilitated by the bounded gradient norms. This can potentially be further leveraged by introducing more layers before the final recurrent layer or stacking multiple recurrent layers, which are examples of possible extensions.

13 Conclusion

In conclusion, the restriction of hidden weight matrices in RNNs (and its variants) to orthogonal matrices seems to solve the vanishing and exploding gradient problems to some extent: this provides improvements in both the toy cases as well as when applied to a language modelling task. This was both theoretically established (in section 2) and empirically verified by plots of gradient norms.

14 Future Work

In light of the theoretical and empirical bounds for the gradient norm of a uRNN which employs the $|x|$ activation function, we will implement a uRNN with this custom activation function and test it on the same experiments described above. Furthermore, to compensate for the reduced dimenstionality of a uRNN and to fully leverage the benefits in training due to the bounded gradient norms, we will introduce more layers before the final recurrent layer to create the best language model we can before the final deadline.

References

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL `http://arxiv.org/abs/1211.5063`.

M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. *CoRR*, abs/1511.06464, 2015. URL `http://arxiv.org/abs/1511.06464`.

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

M. D. Plumbley. Lie group methods for optimization with orthogonality constraints. In *Independent Component Analysis and Blind Signal Separation*, pages 1245–1252, 2004. DOI `10.1007/978-3-540-30110-3_157`.

S. Wisdom, T. Powers, J. R. Hershey, J. L. Roux, and L. E. Atlas. Full-capacity unitary recurrent neural networks. *CoRR*, abs/1611.00035, 2016. URL `http://arxiv.org/abs/1611.00035`.