Improving the Performance of Unitary Recurrent Neural Networks and Their Application in Real-life Tasks

Ivan Ivanov
University of Cambridge
ivan@vankata.tech

Li Jing
Massachusetts Institute of Technology
ljing@mit.edu

Rumen Dangovski
Massachusetts Institute of Technology
rumenrd@mit.edu

ABSTRACT

During a prolonged time execution, deep recurrent neural networks suffer from the so-called *long-term dependency problem* due to their recurrent connection. Although Long Short-Term Memory (LSTM) networks provide a temporary solution to this problem, they have inferior long-term memory capabilities which limit their applications. We use a recent approach for a recurrent neural network model implementing a unitary matrix in its recurrent connection to deal with long-term dependencies, without affecting its memory abilities. The model is capable of high technical results, but due to insufficient implementation does not achieve the expected performance. We optimize the implementation and architecture of the model, achieving time performance up to 5 times better than the original implementation. Additionally, we apply our improved model to three common real-life problems: the automatic text understanding task, the speech recognition task, and cryptoanalysis, and outperform the widely used LSTM model.

KEYWORDS

neural network, recurrent, LSTM, unitary matrix, automatic text understaning, speech recognition, cryptography, vigenère

ACM Reference Format:
Ivan Ivanov, Li Jing, and Rumen Dangovski. 2018. Improving the Performance of Unitary Recurrent Neural Networks and Their Application in Real-life Tasks. In *19th International Conference on Computer Systems and Technologies (CompSysTech'18), September 13–14, 2018, Ruse, Bulgaria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3274005.3274027

1 INTRODUCTION

Artificial neural networks are robust artificial intelligence devices capable of solving complex problems in computer vision, speech recognition, and natural language processing. They present one of the greatest and most important paradigms in programming [12]. Conventionally, a computer receives direct instructions what operations to perform, and complex problems are being decomposed into basic operations. In contrast, the approach of artificial neural networks requires the computer to *learn* how to solve a problem after being presented observational data as input. The notion of neural networks was coined due to their resemblance of the human nervous system. Their learning ability resides in the weighted connections between the different layers in their structure. After initialization, these weights are altered during the process of learning and generally improve the performance of the network.

In the general case, to achieve sufficiently high accuracy on a particular task, a neural network must undergo many training iterations. For complex tasks such as speech recognition and natural language processing, the network should furthermore have a large number of weighted connections and ability to access past data. To provide the network with a *memory* ability, we introduce the recurrent connection which is a cyclic connection between the inside layers (Figure 1).

If used for a prolonged period of time, the first and last hidden layers further diverge from one another which makes their connection unstable and prevents the network from accurately connecting pieces of previous information, impairing its learning ability. This is known as the *long-term dependency problem*, also referred to in literature as the *exploding or vanishing gradient problem* [9]. Common attempts, aiming to solve the problem, have decreased the computational time of the recurrent neural network, but imposed serious limitations on its memory ability.

An approach proposed by Jing et al. in 2016 manages to avoid the problem, without affecting the memory ability, by keeping the recurrent connection weight matrix in a unitary state [11]. However, the implementation of Jing et al. does not meet the theoretically expected performance due to low degree of parallelism. This paper proposes changes to the algorithms by Jing et al. that provide an equivalent computation efficiency, but are highly parallelizable. Section 2 presents current common approaches dealing with the long-term dependencies problem, and explains in greater detail the unitary RNN model we are optimizing. In Section 3, we elaborate on the performed optimizations. In Section 4, we provide standard benchmarks of our improved implementation. In Section 5, we apply our improved implementation to common problems from the real world such as the automatic text understanding task, the speech recognition task, and the cryptoanalysis task, and analyze the achieved results. We publish our code under the GNU General Public License v3.0 on GitHub, accessible here.

2 BACKGROUND

In this section, we discuss the previous research in the field of recurrent neural networks (RNNs), presenting the gated and the unitary model.

2.1 The Artificial Neural Network

Artificial neural networks are organized in layers: the input, hidden, and output layer which respectively accept input data, process it, and store the produced result. For convenience, we present the interlayer connections in the form of matrices whose elements are the weight values. In a standard model of a network, input data is first presented to the input layer, then subjected to linear combination with the interlayer weight matrices, and at the end the final result is stored in the output layer. In each neuron cell, the pieces of data are combined with a certain *bias* value, stored in the cell, also undergoing adjustments, and an additional non-linear *activation* function is applied to them to avoid fitting to the input data. During *training*, the result produced by the neural network is compared to the expected one and the weight values are adjusted based on the discrepancy, so that the next time when the network is presented with similar data as input, it would produce

CompSysTech'18, September 13–14, 2018, Ruse, Bulgaria

ACM ISBN 978-1-4503-6425-6/18/09…$15.00
https://doi.org/10.1145/3274005.3274027

a more accurate result. This series of steps is repeated until the network achieves the desired accuracy.

2.2 The Concept of Recurrence

The recurrent neural network (RNN) is a specific type of artificial neural network described by a recurrent connection present in each of its hidden layers [4]. This connection allows for some of the output of the network to be conserved and then used as input during the next iteration. In this way, the functioning of the network becomes dependent on the previously processed data which creates the memory ability. This approach is especially useful when processing sequential data with dependencies, such as text and speech, as it makes the network able to consider these in its working process [8]. Typical use cases for this type of network include speech recognition and prediction, text analysis and translation. An unrolled model of a recurrent neural network is presented on Figure 1.

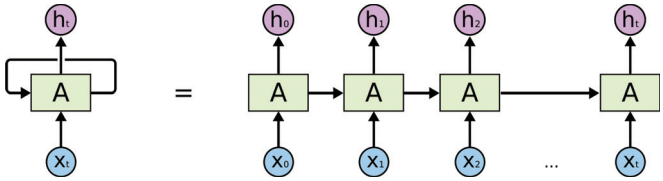


Figure 1: A recurrent neural network (RNN) model and its representation in conventional means through time [13]. The input layer is marked with x, the output layer is marked with h, the hidden layer (only one in this case) is marked with A, and the moment in time is marked by t (Olah, 2015).

2.3 Backpropagation and Gradient Descent

One of most widely used models for training of a neural network is the *gradient descent* algorithm [12]. It aims to reduce the discrepancy between the produced and the expected answer, also defined as the *cost function*, by adjusting the specific weights of the connections and the biases in the neuron cells. To compute the exact change in each of these properties, it uses the *backpropagation* function that determines how much each connection, or bias value has participated in the final cost. The generalized formula for computing the gradient of the hidden layer at time t, producing result $h^{(t)}$, is presented on 1. Using the chain rule it has been simplified to the product the hidden layer matrix, W, and the diagonal matrix with the derivatives of the non-linear activation function, $D^{(t)}$.

$$\frac{\partial C}{\partial h^{(t)}} = \frac{\partial C}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial h^{(t)}} = \frac{\partial C}{\partial h^{(T)}} \prod_{k=t}^{T-1} \frac{\partial h^{(k+1)}}{\partial h^{(k)}} = \frac{\partial C}{\partial h^{(T)}} W^k \prod_{k=t}^{T-1} D^{(k)} \quad (1)$$

2.4 The Long-Term Dependencies Problem and Long Short Term Memory (LSTM) Networks

During prolonged time execution, the unrolled RNN model consists of a significantly big number of hidden layers. Then the hidden layer matrix, W, gets raised to a sufficiently high power and may cause the data vector go to infinity, or converge to zero, making the hidden layer either start learning chaotically, or stop learning at all [3]. This is known as the *vanishing and exploding* gradient problem, or also referred to as *long-term dependencies* in the paper. The Long Short-Term Memory (LSTM) architecture is one of the current best solutions to this problem, first proposed by Hochreiter et al. [10]. The approach introduces three additional structures in the RNN cells, called *gates*. They utilize the sigmoid function to filter the amount of data flowing through the cell and cut some of the connections between the hidden layers. However, this method also limits the memory ability of the network, which in turn lowers its accuracy on important computational tasks [13]. Therefore, LSTM networks present only a partial solution to the long-term dependencies problem because by trading computational intensity they lower their working accuracy.

2.5 The Concept of the Unitary Matrix

A RNN model dealing with the problem that keeps its recurrent connection weight matrix in a unitary state is first proposed by Arjovsky et al. [2]. The approach uses the norm-preserving property of unitary matrices to prevent the data vector from *exploding*, or *vanishing*. The main challenge faced by this RNN model becomes finding an efficient method to retain unity throughout the training process. Arjovsky et al. proposes a $O(N)$ parameter representation of the hidden layer unitary matrix that spans an insufficient part of the $O(N^2)$-dimenstional space of $N \times N$ unitary matrices. In attempt to resolve this, Wisdom et al. proposes a full-space coverage method, but at the expense of N-dimenstional matrix multiplication resulting in computational complexity of $O(N^3)$ [16].

A mediate solution expanding the operational space of the recurrent connection weight matrix with minor increase in the complexity is proposed by Jing et al. [11]. Jing et al.'s RNN model uses two decompositions of the unitary recurrent connection weight matrix into 2×2 rotation matrices arranged in block diagonal matrices. Even though Jing et al.'s model uses full-space coverage and operation optimization, its implementation is not efficiently parallelized. As a result, it does not match the theoretically predicted high performance. In the current research, our goal is to propose step-wise refinements and enrichments of the implementation towards the theoretical efficiency of model.

3 OPTIMIZATIONS

In the process of improving the implementation, we focus on the following aspects: parallelizing serial procedures, replacing memory and time consuming operations with lighter alternatives, and expanding the range of acceptable input parameters.

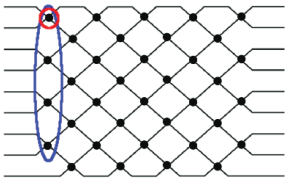
3.1 Implementing Parallelization

The idea of parallelization finds increasing use in neural network algorithms because the neurons in a single layer of the network operate independently of one another. In our approach, we implement our parallelization scheme through the open-source TensorFlow library, developed especially for devices with multiple threads such as GPUs [1]. Its main data type is the tensor data structure which in the context of TensorFlow can be viewed as a multidimensional array.

In the original implementation, the most time-consuming part is the construction of the block diagonal matrices, that build up the unitary matrix, and their efficient update (Figure 2 and Figure 3). Two important operational optimizations of the original algorithm, requires the rotation matrices and the data vector to be permuted based on the hyperparameters of the RNN cell. A downside of this approach is that the generation of the necessary permutations for this operation is implemented via interdependent `for` loops that cannot be parallelized efficiently.

For efficient parallelization of the generation of the permutations necessary for shuffling the vectors in the simple net representation, as presented on Figure 2, we propose a method replacing the hardly parallelizable `for` loops with reshaping, reversing, and transposing tensor operations that can be separated in independent parts and executed simultaneously.

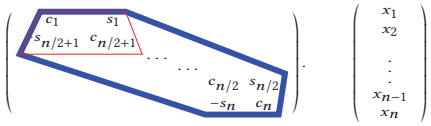
For the data vector permutation generation in the simple net decomposition model, we reshape the initial array into two columns, reverse the columns, and reshape the array back into a single row (2). For the rotation matrices permutation generation, we reshape the initial array into two rows, transpose the array, and reshape it back into a single row (3).



(a)

$$\begin{pmatrix} x_1 c_1 + x_2 s_1 \\ x_2 c_{n/2+1} - x_1 s_{n/2+1} \\ \vdots \\ x_{n-1} c_{n/2} + x_n s_{n/2} \\ x_n c_n - x_{n-1} s_n \end{pmatrix}$$

(c)



(b)

$$\begin{aligned} Fx &= v_1 * x_1 + v_2 * x_2 \\ v_1 &= (c_1, c_{n/2+1}, \ldots, c_{n/2}, c_n) \\ v_2 &= (s_1, -s_{n/2+1}, \ldots, s_{n/2}, s_n) \\ x_1 &= (x_1, x_2, \ldots, x_{n-1}, x_n) \\ x_2 &= (x_2, x_1, \ldots, x_n, x_{2n-1}) \end{aligned}$$

(d)

Figure 2: The simple net decomposition model of the unitary matrix [11]. Each point of intersection in (a) is a 2×2 rotation matrix. The rotation matrices are grouped in columns and each column presents the block diagonal matrix in (b). The multiplication process of this matrix with a data vector is illustrated on (b). The result of the multiplication is presented on (c). The mathematical formula for the optimized multiplication together with the four vector structures are given in (d). Based on the result (c) and the vector structures (d) we can determine the general form of the required permutations.

For the generation of the data vector permutation in the lightweight decomposition, we reshape the initial array into two columns whose elements are arrays of length $x = N/2^{s+1}$, depending on the column number s. Then, we reverse these columns and reshape the array back to its original shape. For the generation of the second permutation, we reshape the initial array into 2^s rows depending on the column number s. Then, we transpose the array and reshape it back to its original shape.

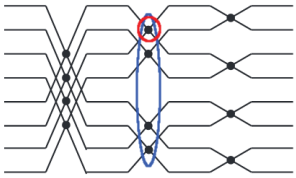
$$\begin{aligned} &[a_0, a_1, \ldots, a_{2k-2}, a_{2k-1}] \mapsto \\ \mapsto &\begin{bmatrix} a_0 & a_1 \\ \vdots & \vdots \\ a_{2k-2} & a_{2k-1} \end{bmatrix} \mapsto \begin{bmatrix} a_1 & a_0 \\ \vdots & \vdots \\ a_{2k-1} & a_{2k-2} \end{bmatrix} \mapsto \\ &\mapsto [a_1, a_0, \ldots, a_{2k-1}, a_{2k-2}] \end{aligned} \tag{2}$$

$$\begin{aligned} &[a_0, a_1, \ldots, a_{2k-2}, a_{2k-1}] \mapsto \\ \mapsto &\begin{bmatrix} a_0 & \cdots & a_{k-1} \\ a_k & \cdots & a_{2k-1} \end{bmatrix} \mapsto \begin{bmatrix} a_0 & a_k \\ \vdots & \vdots \\ a_{k-1} & a_{2k-2} \end{bmatrix} \mapsto \\ &\mapsto [a_0, a_k, \ldots, a_{k-1}, a_{2k-1}] \end{aligned} \tag{3}$$

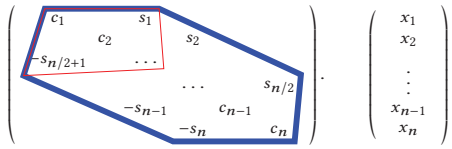
With this, we have transformed the inefficient serial generation of permutations, necessary for the approach, into highly parallelizable TensorFlow operations and have accomplished our goal of implementing a parallelization architecture. Furthermore, we can optimize the implementation by directly applying these transformation operations to the data vectors, skipping the generation of permutations and them permuting the data vectors. In this way we omit the TensorFlow *gather* function which is very time and memory consuming as it allocates additional memory on the GPU for shuffling the vectors.

3.2 Expanding Hyperparameter Range

Another disadvantage of the original implementation of the unitary model is that it has significant limitations on its range of hyperparameters and is not defined for odd hidden layer size and depth in the simple net decomposition, and for a hidden layer size that is not a power of two in the lightweight decomposition. We adapt the simple net decomposition for odd sizes by introducing control values in additional rows and columns that reduce the model to one with even dimensions. In the lightweight decomposition model, we apply Genz et al.'s method for generating random orthogonal matrices to serve any hidden layer size [6]. It consists of adding an extra column (block diagonal matrix) to the decomposition model for the additional rotation



(a)



(b)

$$\begin{pmatrix} x_1 c_1 + x_3 s_1 \\ x_2 c_2 + x_4 s_2 \\ \vdots \\ x_{n-1} c_{n-1} - x_{n-3} s_{n-1} \\ x_n c_n - x_{n-2} s_n \end{pmatrix}$$

(c)

$$\begin{aligned} Fx &= v_1 * x_1 + v_2 * x_2 \\ v_1 &= (c_1, c_2, \ldots, c_{n-1}, c_n) \\ v_2 &= (s_1, s_2, \ldots, -s_{n-1}, -s_n) \\ x_1 &= (x_1, x_2, \ldots, x_{n-1}, x_n) \\ x_2 &= (x_3, x_4, \ldots, x_{n-3}, x_{n-2}) \end{aligned}$$

(d)

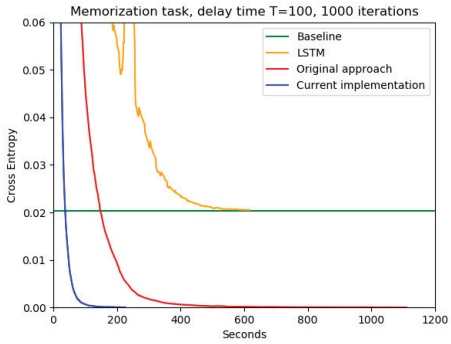
Figure 3: The lightweight decomposition model of the unitary matrix [11]. Each point of intersection in (a) is a 2×2 rotation matrix. The rotation matrices are again grouped in columns and each column presents the block diagonal matrix in (b). The multiplication process of this matrix with a data vector is illustrated on (b). The result of the multiplication is presented on (c). The mathematical formula for the optimized multiplication together with the four vector structures are given in (d). Based on the result (c) and the vector structures (d) we can determine the general form of the required permutations.

matrices and filling its empty cells with control values of 0 and 1. Then,

upon multiplication the control values are added to the rotation matrices values in the other block diagonal matrices and produce the final product. With these improvements, we have not changed the complexity of the RNN, but have made it more configurable and flexible in the general use case.

4 BENCHMARK

In this section, we test our implementation against the original Jing et al.'s model on the standard memorization benchmark task. It is defined by [10], [2], and [8], and tests the basic ability of the RNN to recall information presented T steps earlier in time. For better representation of the learning behavior, we draw a baseline marking the *memoryless* strategy.



Lower is better

Figure 4: Graph of the cross-entropy distribution of the cost vs. training iterations of the two unitary implementations and the LSTM model on the memorization task performed with a decay rate of 0.5 and learning rate of 0.001.

Both unitary implementations express sufficient learning behavior beating the baseline. Our improved implementation outperforms the original one by a factor of 5 in terms of execution time (Figure 4). In particular cases during the process of testing, the time performance improvement reached up to 12 times better than the original implementation. In these, the original implementation used a large portion of the GPU *cache* memory, probably due to the *gather* function, which was absent in our implementation. Compared with the LSTM model, our improved unitary implementation performs better both in terms of time and accuracy, as the LSTM network barely reaches the baseline and is not able to beat it.

An additional benchmarking test was performed on the well-known MNIST task, but as it showed similar results to the memorization task, it is not included in this paper.

5 TESTS ON PROBLEMS FROM REAL LIFE

This section evaluates the new implementation with several tasks from the real world assessing the network's appliance in reality. Due to content and time constraints, this paper presents only the most significant tests and graphics, conducted before the date of submission. The full test data and the most recent results are published in the GitHub Wiki of the code repository, accessible here.

5.1 Reading Comprehension: The bAbI Dataset

As a proof of concept for the performance of the improved model on the automatic text understanding task, we run our enhanced implementation on the bAbI dataset provided by Facebook [15]. It consists of 150 words used in 20 different tasks for automatic text understanding and reasoning each testing different capabilities of the RNN model.

The model we are testing is derived from a *Keras* sample implementation of an LSTM network [5]. Its architecture is presented on Figure 5. The results

$$Sentence \xrightarrow[\text{dropout}]{\text{embedding}} Processed \searrow$$
$$Question \xrightarrow[\text{dropout}]{\text{embedding}} RNN1 \rightarrow RNN2 \xrightarrow[\text{dense}]{\text{dropout}} Answer$$

Figure 5: The architecture of the Keras recurrent model

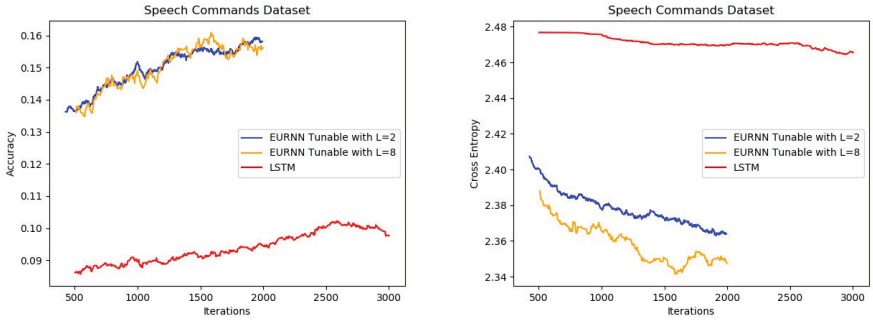
of the approach are the following: Both RNN cells achieve sufficiently high

Task	Unitary	LSTM
1 - Single Supporting Fact	74.6%	**100%**
2 - Two Supporting Facts	39.7%	**42.6%**
3 - Three Supporting Facts	**41.4%**	24.8%
4 - Two Arg. Relations	**100%**	76.9%
5 - Three Arg. Relations	**96.0%**	93.6%
6 - Yes/No Questions	82.5%	**83.2%**
7 - Counting	86.0%	**90.6%**
8 - Lists/Sets	**77.0%**	76.9%
9 - Simple Negation	81.2%	**84.7%**
10 - Indefinite Knowledge	**72.8%**	50.8%
11 - Basic Coreference	86.6%	**96.8%**
12 - Conjunction	86.4%	**99.4%**
13 - Compound Coreference	94.3%	**97.6%**
14 - Time Reasoning	**48.1%**	44.1%
15 - Basic Deduction	59.9%	**60.1%**
16 - Basic Induction	**48.9%**	47.0%
17 - Positional Reasoning	**71.1%**	52.0%
18 - Size Reasoning	90.7%	**91.9%**
19 - Path Finding	**15.6%**	12.6%
20 - Agents Motivations	98.2%	98.2%
Mean Performance	**72.6%**	71.2%

Table 1: Results of the LSTM and the unitary model using Keras' implementation on the bAbI dataset. The LSTM layer in the LSTM network has hidden layer size of 50 with 10000 recurrent parameters. The unitary layer in the unitary recurrent network uses the simple net representation architecture with width 2 and hidden layer size of 200 with 199 recurrent parameters. Both networks were trained on the standard 10000 training samples for 40 epochs.

accuracy on the tests (around 80-90%). Their results are comparable with the unitary cell performing better on more complex tasks such as the **Two and Three Supporting Facts**. On them, it keeps a relatively unchanging accuracy of around 40%, while the LSTM cell drops exponentially with the increase of the number of facts. On the most difficult task **Path Finding**, the unitary approach achieves accuracy with 3% greater than the LSTM approach, again presenting its capabilities of solving more complex problems than the LSTM approach.

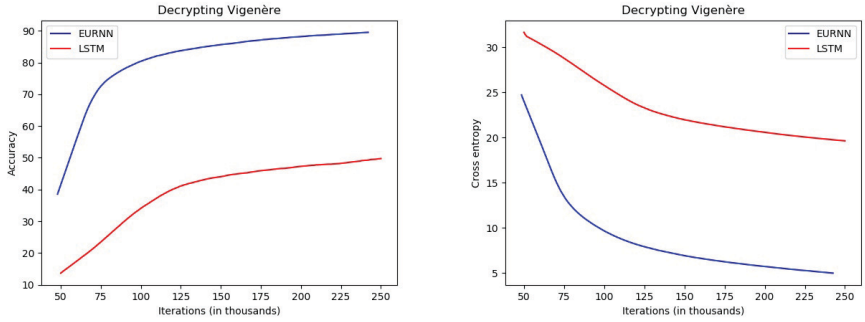
Additionally, we have tested another model on this task, implementing a LSTM and unitary RNN cell inside the Differential Neural Computer (DNC) device, but it showed around 20% lower accuracy on almost all tests and therefore is not included in this paper.



Higher is better **Lower is better**

(a) **(b)**

Figure 6: Results of two variations of the unitary implementation (EURNN) and the LSTM model on the speech recognition task performed with a decay rate of 0.9 and learning rate of 0.0001. The unitary configuration used hidden layer size of 40 and the complex domain for weights. The LSTM configuration used hidden layer size of 64.



Higher is better **Lower is better**

(a) **(b)**

Figure 7: Results of the unitary implementation (EURNN) and the LSTM model on decrypting Vigenère, performed with a learning rate of 0.0005. The unitary configuration used hidden layer size of 128, depth of 8, and the complex domain for weights. The LSTM configuration used hidden layer size of 40 with the same number of parameters.

5.2 Speech Recognition

The second real-life test we are performing is speech recognition via the newly released *Google Speech Commands Dataset* consisting of 65,000 WAVE audio files of people saying thirty different words [14]. The model we are using consists of a single RNN hidden layer that reads a spectrogram of the sound and returns the index of the word that has been said. The results from the conducted tests are presented on Figure 6. The results demonstrate that the unitary model achieves higher accuracy for less iterations. In this case, it even does not fall below on a single instance, while the LSTM cell seems to have reached its maximum potential. However, the low mean accuracy of both cells shows that this RNN configuration is not suitable for speech recognition in the general run and a more complex structure should be used.

5.3 Cryptoanalysis

As a third application in real life, we assess the model on cryptoanalysis, and more precisely on deciphering the *Vigenère* cipher. Neural networks are increasingly used for such tasks because of the non-linearity that they employ and which stays in the basis of many cryptographic ciphers. Additionally, neural networks are able to *learn* and model functions which also makes them able to encrypt, or decrypt messages. We have based our model on Sam Greydanus' implementation which uses stacked RNN cells to break the cipher [7]. The results from the test are presented on Figure 7.

The results show that the unitary model surpasses the LSTM network on the task. It achieves an accuracy of 92% for the training iterations while the LSTM hardly manages to reach 50%. This means that the unitary model will have much better application on real-life cryptographic tasks compared to the LSTM one. However, it is important to mention that despite its higher accuracy, the unitary model took much more time and memory resources for its work than the LSTM. This outlines a possible limitation for its use in devices with lower computational power as it will function ineffectively on them.

Additionally, cryptoanalysis tests were conducted on the Autokey cipher, but they showed similar results and are not included in this paper.

6 CONCLUSION

We presented the optimizations conducted in the implementation of a novel recurrent network model and provided results of its application to three real-life tasks: of automatic text understanding, of speech recognition, and of cryptoanalysis. Based on the results on the standard benchmarks, our implementation improves over the time performance of the original by at least 20%, reaching its best of 15 times better, reduces the runtime memory usage, and retains the accuracy of Jing et al.'s model. Based on the real-life task results, the improved implementation achieves maximum mean accuracy of around 75%, beating the LSTM model, on the automatic text understanding task. On the speech recognition task, the unitary model performs sufficiently better than the LSTM model, but both RNN models struggle to achieve an accuracy over 20% which is quite low for the use case, and presents that they are ineffective solutions for the problem. Last, on the cryptoanalysis test, the unitary model manages to reach an accuracy of around 92%, two times larger than the result of the LSTM cell. This shows it performs better than the LSTM on these types of tasks, but it requires much more time and memory resources than conventional networks which becomes one of its limitations. The conducted tests show that the unitary model can successfully be introduced to more real-life problems and replace some of the currently used RNN models.

As future work we plan on improving the structure of the lightweight decomposition model avoiding the increase in computational time, further decreasing time and memory consumption by using lighter operations, and applying the concept of the unitary matrix in Recurrent Highway Neural Networks known for their higher capabilities in dealing with more complex tasks.

7 ACKNOWLEDGMENTS

The current research was initiated during the summer research program *Research Science Institute* hosted in the *Massachusetts Institute of Technology*. It was performed in the Department of Physics in cooperation with Prof. Marin Soljačic's Deep Learning research group and was supported with their computational resources. Additional tests were run on an NVIDIA DGX cluster owned by SAP Labs Bulgaria.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, and Craig Citro et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.
[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. 2016. Unitary evolution recurrent neural networks. (2016), 1120–1128.
[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.
[4] Denny Britz. 2015. Recurrent Neural Networks Tutorial, Part 1 âĂŞ Introduction to RNNs. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/.
[5] François Chollet et al. 2015. Keras. https://github.com/keras-team/keras.
[6] Alan Genz. 1998. Methods for generating random orthogonal matrices. *Monte Carlo and Quasi-Monte Carlo Methods* (1998), 199–213.
[7] Sam Greydanus. 2017. Learning the Enigma with Recurrent Neural Networks. *arXiv preprint arXiv:1708.07576* (2017).
[8] Mikael Henaff, Arthur Szlam, and Yann LeCun. 2016. Orthogonal RNNs and long-memory tasks. *arXiv preprint arXiv:1602.06662* (2016).
[9] Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen Netzen. *Diploma, Technische Universität München* 91 (1991).
[10] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[11] Li Jing, Yichen Shen, Tena Dubček, John Peurifoy, Scott Skirlo, Max Tegmark, and Marin Soljačić. 2016. Tunable Efficient Unitary Neural Networks (EUNN) and their application to RNN. *arXiv preprint arXiv:1612.05231* (2016).
[12] Michael A. Nielsen. 2017. *Neural Networks and Deep Learning*. Determination Press.
[13] Christopher Olah. 2015. Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[14] Pete Warden. 2017. âĂIJLaunching the speech commands dataset. *Google Research Blog* (2017).
[15] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698* (2015).
[16] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. 2016. Full-capacity unitary recurrent neural networks. (2016), 4880–4888.