Skolkovo Institute of Science and Technology

Numerical Linear Algebra

Stabilizing Gradients for Deep Neural Networks via Efficient SVD Parameterization

Kirill Shcherbakov, Egor Sevriugov, Mikhail Vasilkovsky, Egor Konyagin, Ekaterina Vorobyeva, Leonid Litovchenko

33^{rd} group

Professor
Ivan Oseledets

supervised by
Aleksandr Katrutsa

Moscow 2019

Contents

1 Introduction

Deep neural networks have achieved great success in various fields, including computer vision, speech recognition, natural language processing, etc. Despite their tremendous capacity to fit complex functions, optimizing deep neural networks remains a contemporary challenge. Two main obstacles are vanishing and exploding gradients [4], that become particularly problematic in Recurrent Neural Networks (RNNs) since the transition matrix is identical at each layer, and any slight change to it is amplified through recurrent layers.

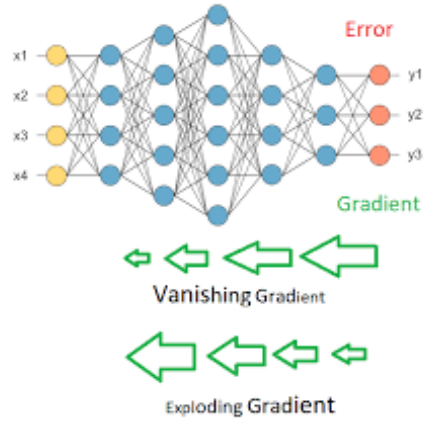


Figure 1: Vanishing and Exploding gradients

To show it, consider simple example. Suppose we have diagonalizable weight matrix W:

$$W = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} = S \begin{bmatrix} \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 & 0 \\ 0 & 0 & \lambda_3 & 0 & 0 \\ 0 & 0 & 0 & \lambda_4 & 0 \\ 0 & 0 & 0 & 0 & \lambda_5 \end{bmatrix} S^{-1} \tag{1}$$

During training of RNN weight matrix is multiplied by itself several times (N) we get:

$$W^N = S \begin{bmatrix} \lambda_1^N & 0 & 0 & 0 & 0 \\ 0 & \lambda_2^N & 0 & 0 & 0 \\ 0 & 0 & \lambda_3^N & 0 & 0 \\ 0 & 0 & 0 & \lambda_4^N & 0 \\ 0 & 0 & 0 & 0 & \lambda_5^N \end{bmatrix} S^{-1} = S\Lambda^N S^{-1} \tag{2}$$

If $|\lambda_i| < 1$ then $|\lambda_i^N| \to 0$ for $N \to \infty$. In computer arithmetics we can find finite N for which $|\lambda_i| = 0$. On the other hand if $|\lambda_i| > 1$ then $|\lambda_i^N| \to \infty$ for $N \to \infty$. That results in losing information during addition or multiplication

with small numbers. Simple example: $h_j = W_h h_{j-1} + W_x x_j$, where h — some hidden state. As you can see if j is really high and x_j is small then $W_h h_{j-1} = W^j h_0 + ... \to \infty$ and in computer arithmetics $h_j = W_h h_{j-1} + W_x x_j = W_h h_{j-1}$ — losing information.

In our project, we explore an efficient parametrization of weight matrices that arise in a deep neural network, thus allowing to stabilize the gradients that arise in its training, while retaining the desired expressive power of the network. In more detail we are using tools that are common in numerical linear algebra, namely Householder reflectors, for representing the orthogonal matrices that arise in the SVD. The SVD parametrization allows us to retain the desired expressive power of the network, while enabling us to explicitly track and control singular values.

2 Problem statement and existing solutions

Numerous approaches have been proposed to address the vanishing and exploding gradient problem. Let us briefly review and discuss their advantages and disadvantages.

Long short-term memory (LSTM) [5]

The core concept of LSTM's (Fig. 1) is the cell state, and it's various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. You can think of it as the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make it's way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information get's added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

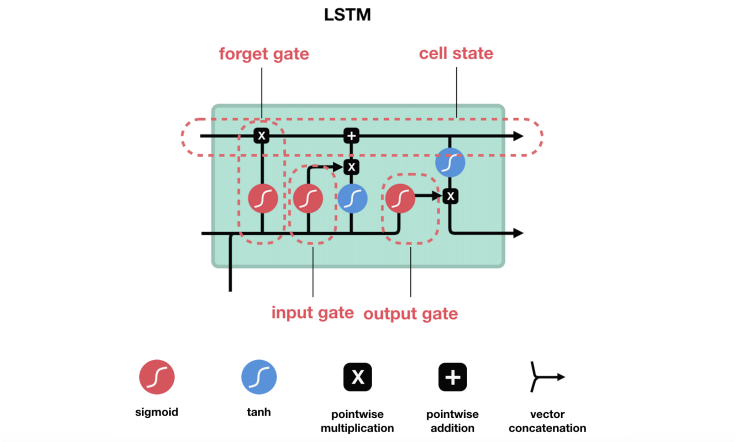


Figure 2: Idea of LSTM method

Recurrent neural networks using LSTM units partially solve the vanishing gradient problem, because LSTM units allow gradients to also flow unchanged. However, LSTM networks can still suffer from the exploding gradient problem.

Residual neural network (ResNet) [2]

Residual neural networks build on constructs known from pyramidal cells in the cerebral cortex this by utilizing skip connections, or shortcuts to jump over some layers.

One motivation for skipping over layers (Fig. 2) is to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights.

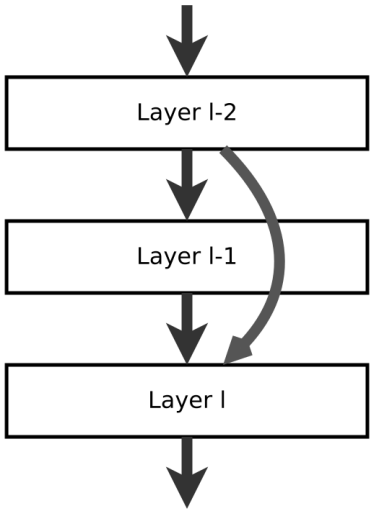


Figure 3: Idea of ResNets

Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. Towards the end of training, when all layers are expanded, it stays closer to the manifold and thus learns faster. A neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold, and necessitates extra training data to recover.

Identity recurrent neural network (IRNN) [6]

An identity recurrent neural network (IRNN) is a vanilla recurrent neural network (as opposed to e.g. LSTMs) whose recurrent weight matrices are initialized with the identity orthogonal matrix, the biases are initialized to zero, and the hidden units (or neurons) use the rectified linear unit (ReLU).

An IRNN can be trained more easily using gradient descent (as opposed to a vanilla RNN that is not an IRNN), given that it behaves similarly to an LSTM-based RNN, that is, an IRNN does not suffer (much) from the vanishing gradient problem.

The IRNN achieves a performance similar to LSTM-based RNNs in certain tasks, including the adding problem (a standard problem that is used to examine the power of recurrent models in learning long-term dependencies). In terms of architecture, the vanilla RNNs are much simpler than LSTM-based RNNs, so this is an advantage.

However, there is no guarantee that the transition matrix is close to orthogonal after a few iterations.

Unitary RNN (uRNN) [1]

The unitary RNN (uRNN) algorithm parameterizes the transition matrix with reflection, diagonal and Fourier transform matrices. By construction, uRNN ensures that the transition matrix is unitary at all times. Although this algorithm performs well on several small tasks, and it was showed that uRNN only covers a subset of possible unitary matrices and thus detracts from the expressive power of RNN.

Orthogonal RNN (oRNN) [3]

An improvement over uRNN, the orthogonal RNN (oRNN) was proposed. oRNN uses products of Householder reflectors to represent an orthogonal transition matrix, which is rich enough to span the entire space of orthogonal matrices. Meanwhile, and it was empirically demonstrated that the strong constraint of orthogonality limits the model's expressivity, thereby hindering its performance.

Hence, an idea to parameterize the transition matrix by its SVD was proposed $W = U\Sigma V^T$ (factorized RNN) and restrict Σ to be in a range close to 1; however, the orthogonal matrices U and V are updated by geodesic gradient descent using the Cayley transform, thereby resulting in time complexity cubic in the number of hidden nodes which is prohibitive for large scale problems.

3 spectralRNN overview

In this project we consider the article [8] and the proposed method of RNN model modification to avoid vanishing/exploding gradient problem. What is more the proposed method is claimed to solve the gradient vanishing/exploding problem without hurting expressive power of the model. Article authors are trying to solve this problem using recurrent neural networks, showing that each layer is not necessarily near identity, but being close to orthogonality suffices to get a similar result. SVD parametrization allows to develop an efficient way to constrain the weight matrix to be a tight frame (22), consequently be able to reduce the sensitivity of the network to adversarial examples.

Below we provide brief overview of the theoretical results obtained in considered article. The results we consider apply to square transition matrices W. The article also provides the generalization of the method for the rectangular matrices which we will omit in the overview.

The considered method modifies the following equations parameters computation of the RNN. Given a hidden state vector from the previous step $h^{(t-1)} \in R^n$ and input $x^{(t-1)} \in R^{n_i}$, RNN computes the next hidden state $h^{(t)}$ and output vector $\hat{y}(t) \in R^{n_y}$ as:

$$h^{(t)} = \sigma(Wh^{(t-1)} + Mx^{(t-1)} + b)$$

$$\hat{y}^{(t)} = Yh^{(t)}$$

Representation of SVD decomposition via Householder reflections

The SVD of the transition matrix $W \in R^{n\times n}$ of an RNN is given by $W = U\Sigma V^\top$, where Σ is the diagonal matrix of singular values, and $U, V \in R^{n\times n}$ are orthogonal matrices, i.e., $U^\top U = UU^\top = I$ and $V^\top V = VV^\top = I$(21). The SVD parametrization should be kept during the training process in order to maintain ability to regularize the singular values. But keeping the parametrization in a standard way results in tripling the amount of memory needed as well as increasing the complexity of Wh multiplication.

So the authors propose to store U and V as a composition of Householder reflectors. Given a vector $u \in R^k$, $k \leq n$, Householder reflector $H_k^n(u)$ is defined as:

$$H_k^n = \begin{cases} \begin{pmatrix} I_{n-k} & \\ & I_k - 2\frac{uu^\top}{||u||^2} \end{pmatrix}, & u \neq 0 \\ I_n & \text{otherwise} \end{cases}$$

Householder reflector is an orthogonal matrix. To store a Householder reflector it is enough to store vector u rather than the full matrix.

In the considered paper authors refer to the following theorems.

Theorem 1 *Any real matrix $M \in R^{n\times n}$ can be represented as*

$$M = H_n(u_n)\dots H_1(u_1)\,\mathrm{diag}(\sigma)H_1(v_1)\dots H_n(u_n),$$

where $u_k, v_k \in R^k$, $\sigma \in R^n$.

Theorem 2 *Any orthogonal matrix $M \in R^{n\times n}$ can be represented as*

$$M = H_n(u_n)\dots H_{k_1}(u_{k_1})\,\mathrm{diag}(\sigma)H_{k_2}(v_{k_2})\dots H_n(u_n),$$

where $u_k, v_k \in R^k$, $\sigma \in R^n$, if $k_1 + k_2 \leq n + 2$.

Theorem 2 indicates that if the total number of reflectors is greater than n:

$$(n - k_1 + 1) + (n - k_2 + 1) \geq n,$$

then the parameterization covers all orthogonal matrices.

Singular values regularization

In Spectral-RNN the transition matrix $W \in R^{n\times n}$ is parametrized using $m_1 + m_2$ Householder reflectors:

$$W = H_{k_1}(u_{k_1})\,\mathrm{diag}(\sigma)H_{k_2}(v_{k_2})\dots H_n(u_n),$$
$$\text{where } k_1 = n - m_1 + 1, \quad k_2 = n - m_2 + 1.$$

The selection of parameters m_1, m_2 affect the model expressiveness and the memory consumption. For example as it is stated in Theorem 1 $m_1 = m_2 = n$ choice is giving the same expressive power as the vanilla RNN. And the consequent result of Theorem 2:

$$m_1 + m_2 \geq n, \quad \sigma = 1$$

represent the oRNN.

To avoid the vanishing/exploding gradient problem the singular values in parametrization should be regularized. The authors of the article propose the following parametrization of σ:

$$\sigma_i = 2r(f(\hat{\sigma}_i) - 0.5) + \sigma^*, i = [n]$$

where f is the sigmoid function and $\hat{\sigma}_i$ is updated from u_i, v_i via stochastic gradient descent. The above allows to constrain σ_i to be within the following range:

$$\sigma_i \in [\sigma^* - r, \sigma^* + r].$$

In practice, σ^* is usually set to 1 and $r << 1$.

Complexity and memory

The authors show that for $m_1 = m_2 = n$ the memory consumption of spectralRNN is of the same order as the memory consumption of vanillaRNN. The general formula for the number of parameters of the spectralRNN is $(n_y + n_i + m_1 + m_2 + 2)n - \frac{m_1^2+m_2^2-m_1-m_2}{2}$.

The storage of transition matrix W via Householder matrices allows also to compute the operation Wh efficiently. Authors of the considered article compute the complexities of operations needed for the training of spectralRNN and show that the computational cost of spectralRNN is of the same order as RNN.

4 Project goals and workflow

We consider two main goals for our project: first is to confirm or deny the superiocity of spectralRNN compared to other recurrent models, second is to study the behaviour of spectralRNN dependency on the choosen parameters m_1, m_2.

NOTE: The results were obtained using magma library as a backend for efficient SVD and other matrix operations. The training process was conducted on nvidia Tesla V100 GPU on Google Cloud Platform. The installation process required build from a scratch, which was successfully done.

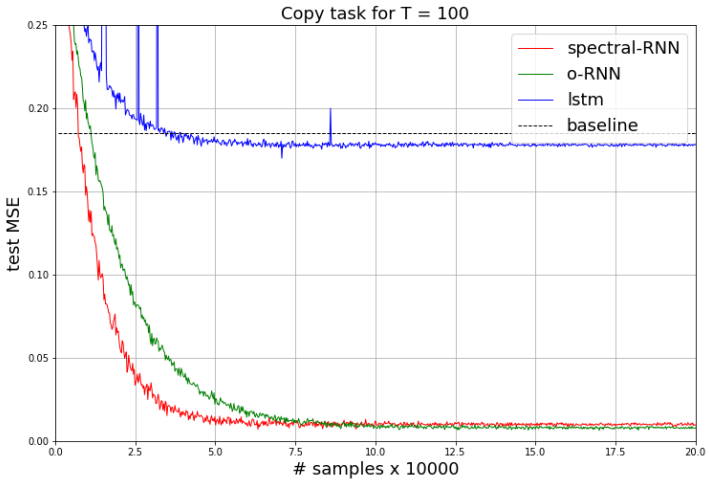
Related code can be found on our github repository:
https://github.com/waytobehigh/nla_proj.

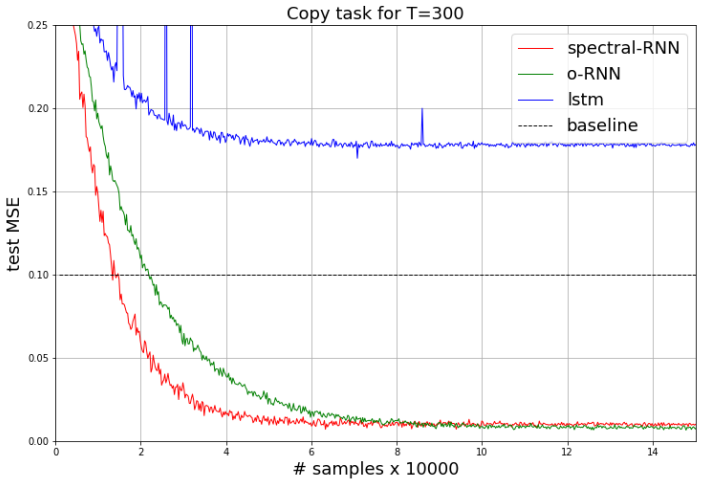
4.1 Models comparison

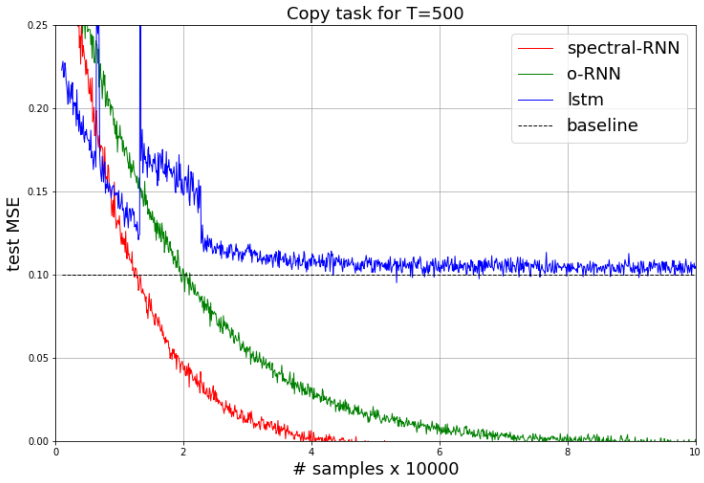
To compare the models performance we consider three problems: copying task, addition task and permute-MNIST. Authors of the article perform use these datasets to compare performance of LSTM, IRNN, oRNN, uRNN, factorized RNN, RC uRNN, FC uRNN and spectralRNN. In our work we will compare the perfomance of oRNN, LSTM and spectralRNN only, as the there is no open code for the other models that can be easily adopted and evaluated.

4.1.1 Copy task

Copy task: Let $A = a_i{}_{i=0}^{9}$ be the alphabet. The input data sequence $x \in A^{T+20}$ where T is the time lag. $x_{1:10}$ are sampled uniformly from $\{a_i\}_{i=0}^{7}$ and x_{T+10} is set to a_9. The rest of x_i are set to a_8. The network is asked to output $x_{1:10}$ after seeing a_9. That is to copy $x_{1:10}$ from the beginning to the end with time lag T. A baseline strategy is to predict a_8 for $T + 10$ entries and randomly sample from $a_i{}_{i=1}^{7}$ for the last 10 digits.
Results obtained for different values of T are presented below.


Copy task for T = 100
spectral-RNN
o-RNN
lstm
baseline
test MSE
samples x 10000


Copy task for T=300
spectral-RNN
o-RNN
lstm
baseline
test MSE
samples x 10000

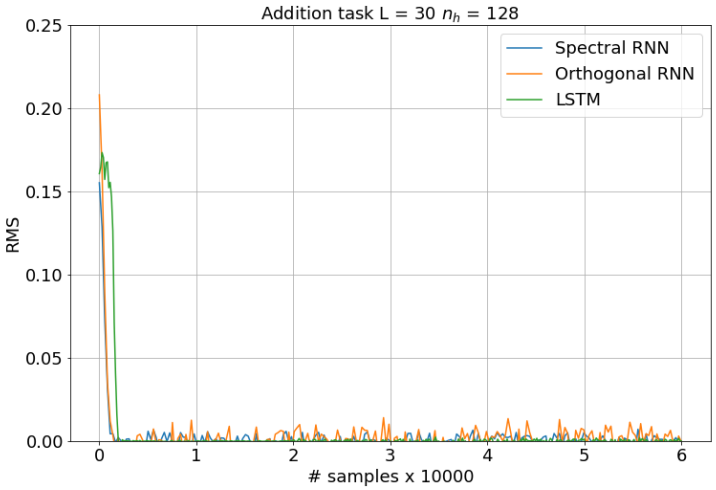


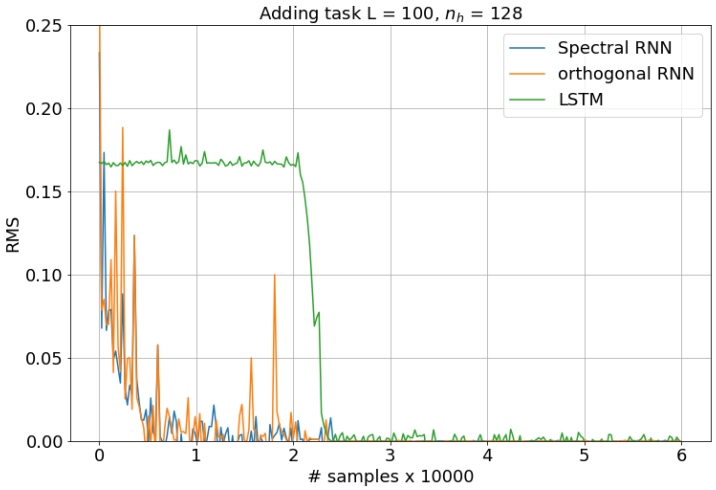
As it can be seen, Spectral-RNN consistently outperforms all other models. LSTM model is not able to beat the baseline when the time lag is large. In fact, the test MSE for LSTM is very close to the baseline, indicating that they may not utilize any useful information throughout the larger time lag.

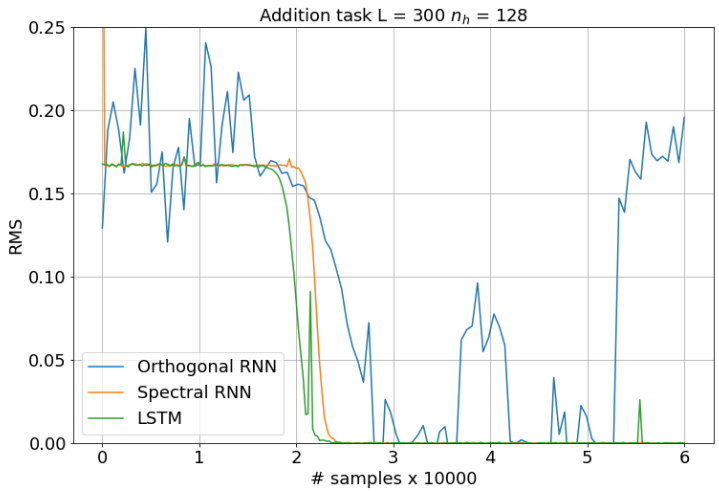
4.1.2 Addition task

The Addition task requires the network to remember two marked numbers in a long sequence and reduce their sum. Each input data includes two sequences: top sequence whose values are sampled uniformly from $[0, 1]$ and bottom sequence which is a binary sequence with only two ones. Such architecture is called Seq2One.

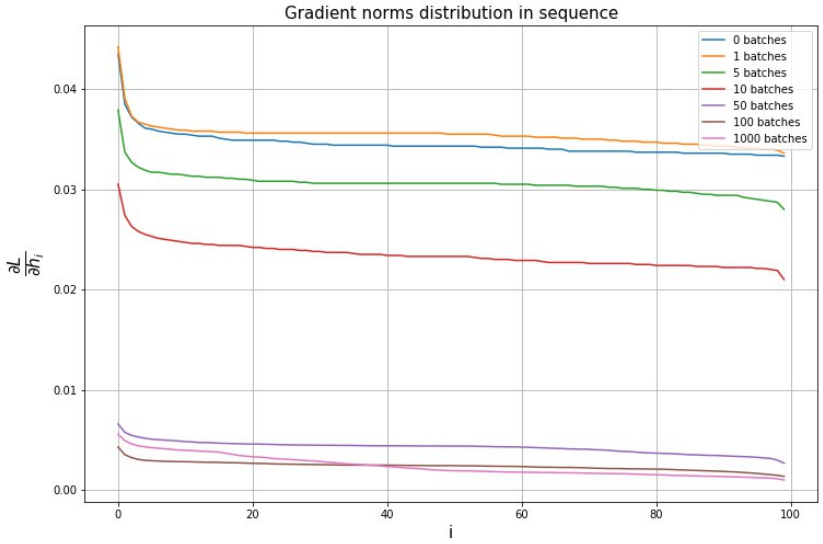
Results of the experiments conducted are presented bellow. L parameter here is a sequence length. As it can be seen from the plots, spectralRNN shows faster convergence than LSTM and more stable behaviour, compared to the orthogonal RNN.


Addition task L = 30 n_h = 128
Spectral RNN
Orthogonal RNN
LSTM
RMS
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
samples x 10000


Adding task L = 100, n_h = 128
Spectral RNN
orthogonal RNN
LSTM
RMS
0.25
0.20
0.15
0.10
0.05
0.00
0
1
2
3
4
5
6
samples x 10000



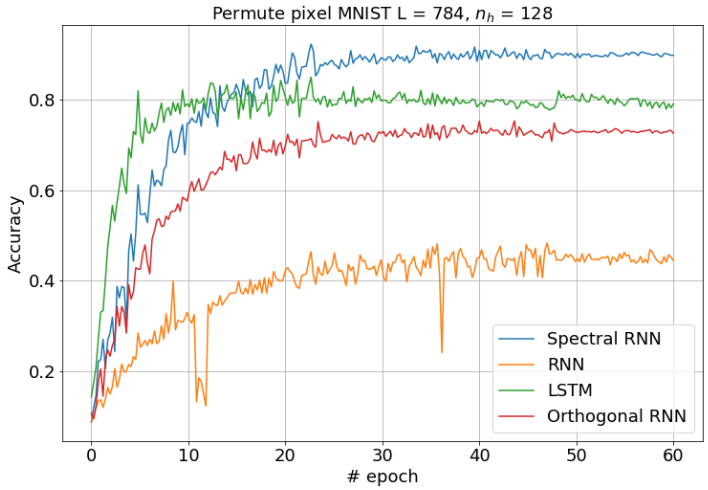
We have also checked the gradients of the model on the $L = 100$,



According to the plot, that the norm of the gradient derived by the earliest tokens is almost equal to one derived by the latest, which implies gradient stability. The same situation happens starting from the very first batches to the later epochs, but the norm in particular decreases. It can be connected to convergence to a local minimum or a saddle point, which usually leads to smaller gradient steps in case of smooth functions.

4.1.3 Permute-MNIST

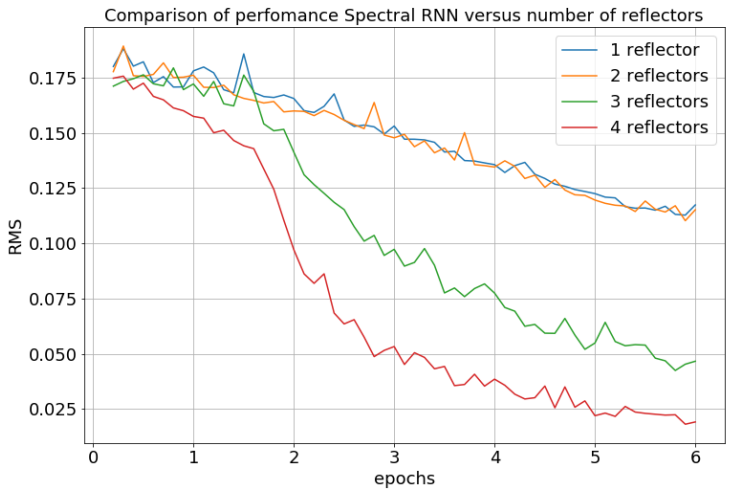
This task is a multiclass classification on the MNIST digits task based on this paper[7]. The images are first reshaped from (n, n) matrices into $(n^2, 1)$ vectors and then permuted. Then they are fed into the neural networks and benchmarks are conducted. The results are shown below.



As it can be seen, spectralRNN's behaviour is almost similar to the one of the LSTM despite somewhat worse performance of the latter.

4.2 Parameters variation

spectralRNN model has global parameters m_1, m_2. These are the numbers of the Householder reflectors. We aim to analyze the performance of the spectralRNN dependent on these parameters. We conducted our comparison on the Addition Task data.



The obtained results prove theoretical assumption: the model expressiveness grows as the number of Householder reflections increases.

5 Team contribution

All members of the team contributed equally. Areas of responsibility of the each team member are listed bellow:

- Kirill Shcherbakov - TF2 code implementation, add task and permute mnist performance evaluation, grid search over learning rate, decay rate, batch size
- Egor Sevriugov - TF2 code implementation, add task and permute mnist performance evaluation, expressiveness experiments
- Mikhail Vasilkovsky - team coordination, stability experiments, gradient stability analysis, pitching
- Egor Konyagin - initial code adaptation and launching issues (cuda and magma implementation), copy task performance evaluation, pitching
- Ekaterina Vorobyeva - team coordination, theory analysis, report, presentation
- Leonid Litovchenko - report, results interpretation, related papers analysis, data generation

6 References

References

[1] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[3] K. Helfrich, D. Willmott, and Q. Ye. Orthogonal recurrent neural networks with scaled cayley transform, 2017.

[4] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[6] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units, 2015.

[7] N. Masse, G. Grant, and D. Freedman. Alleviating catastrophic forgetting using context-dependent gating and synaptic stabilization. *Proceedings of the National Academy of Sciences*, 115, 02 2018.

[8] J. Zhang, Q. Lei, and I. S. Dhillon. Stabilizing gradients for deep neural networks via efficient svd parameterization, 2018.