Dealing with the Vanishing and Exploding Gradient Problems in Recurrent Neural Networks

Polina Kirichenko, Ekaterina Lobacheva
Advisor: Dmitry P. Vetrov

Department of Computer Science,
National Research University Higher School of Economics

Abstract

Recurrent Neural Networks (RNNs) are used in a wide range of machine learning problems where the input data is sequence-like: text classification, speech recognition, part-of-speech tagging and many others. However, simple RNNs are hard to train due to the vanishing and exploding gradients problem. It impedes optimization and prevents RNNs from learning long temporal dependencies. There is a number of suggested solutions for alleviating this issue including the most popular gated architectures – Long Short Term Memory (LSTM) [3] and Gated Recurrent Unit (GRU) [2]. However, these RNN modifications are complex and have much more parameters which slows down both training and applying RNNs. One of the recently proposed initialization tricks is to use normalized positive-definite matrix as the initial value for the recurrent matrix and Rectified Linear Unit (ReLU) as an activation function for the recurrent layer [9]. We try to reproduce the experiments from the paper [9] and further study the effect of positive-definiteness constraint on the recurrent matrix.

Index terms — Machine Learning, Recurrent Neural Network, Exploding and Vanishing Gradients.

Contents

1 Introduction

Neural Networks have recently shown state-of-the-art performance in a lot of machine learning tasks. Recurrent Neural Network (RNN) is Neural Network architecture designed for working with sequential data (like text or music), they have special temporal connections in their hidden layers as shown in the scheme 1. This recurrent layer is used for modeling memory and capturing long dependencies in input sequences. Consider RNN with one hidden layer, and let $x = (x_1, \dots, x_T)$ be a sequential input. The computation of the hidden state h_t at each step t goes as follows:

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_1)$$

W_{hh} will be further referred to as the recurrent matrix. We will see later that its norm significantly affects the learning progress. For tasks with sequential output (like text generation, tagging, *etc.*), y_t at each step is computed as:

$$y_t = g(W_{hy}h_t + b_2)$$

When having single target y for the whole sequence (in classification, regression), only the last hidden state h_T is used:

$$y = g(W_{hy}h_T + b_2)$$

Here, $f(x)$ and $g(x)$ are nonlinear activation functions.

RNNs are usually trained using Back Propagation Through Time algorithm (BPTT) [8]. When dealing with long sequences, vanilla RNNs are hard to train due to the *vanishing and exploding gradients* problem which we will closely look at in the next section. That is why, RNN modifications like Long Short Term Memory (LSTM) [3] or Gated Recurrent Unit (GRU) [2] are more often used in practice. Both LSTM and GRU have complex structure and much more parameters than in simple RNNs, so simpler ways of dealing with this problem are actively studied. In this work, we also address this problem and, in particular, how positive-definiteness restriction on the recurrent matrix affects the training progress.

We organize the rest of this report as follows: in section 2 we study the roots of the exploding and vanishing gradients problem, in section 3 we quickly go through some of the suggested solutions, further in section 4 we motivate ReLU nonlinearity for recurrent layer and normalized positive-definite initialization for recurrent matrix, in section 5 we introduce positive-definiteness constraint on the recurrent matrix and, finally, in section 6 we describe the experiments we conducted.

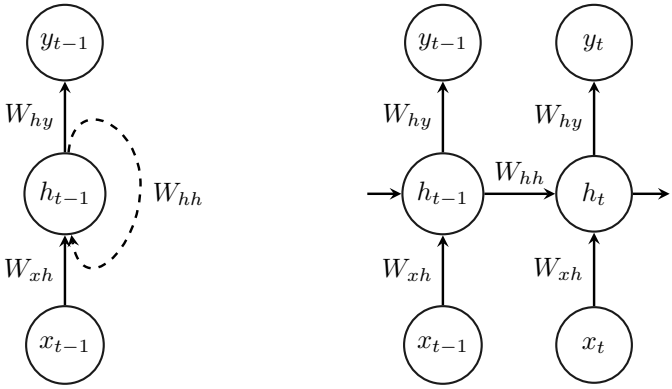


Figure 1: Folded (left) and unfolded (right) computation graph for vanilla Recurrent Neural Network with one hidden layer.

2 Exploding and vanishing gradients problem

Let us show where the problem of exploding and vanishing gradients arises. During BPTT, the errors are propagated back all along the sequence $x = (x_1, \dots, x_T)$. Consider two steps t and $k \ll t$. Then the gradient

of the hidden state h_t with respect to the hidden state h_k at the earlier step is:

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k}^{t-1} \frac{\partial h_{i+1}}{\partial h_i} = \prod_{i=k}^{t-1} W_{hh}^T \text{Diag}\left(f'(h_i)\right)$$

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq \prod_{i=k}^{t-1} \left\| W_{hh}^T \right\| \left\| \text{Diag}\left(f'(h_i)\right) \right\| \leq \left\| W_{hh}^T \right\|^{t-k} \max_{i=k,\ldots,t-1} \left\| \text{Diag}\left(f'(h_i)\right) \right\|^{t-k}$$

Here and elsewhere in this work we use ℓ_2-norm. For simplicity, consider $f(x) = x$ from which follows $\max_{i=k,\ldots,t-1} \left\| \text{Diag}\left(f'(h_i)\right) \right\| = 1$. Then if W_{hh} has eigenvalues $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, it is easy to prove [7] that:

- it is sufficient that $|\lambda_1| < 1$ for gradients to vanish
- it is necessary that $|\lambda_1| > 1$ for gradients to explode

Proof. Let $v_1, \ldots, v_n$ be the eigenvectors corresponding to the eigenvalues $\lambda_1, \ldots, \lambda_n$. They form a basis in $\mathbb{R}^n$, therefore:

$$\frac{\partial \mathcal{L}_t}{\partial h_t} = \sum_{i=1}^{n} c_i v_i \tag{1}$$

Using decomposition (1) and $\frac{\partial h_t}{\partial h_k} = \left(W_{hh}^T\right)^{t-k}$, we get:

$$\frac{\partial \mathcal{L}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} = \left(\sum_{i=1}^{n} c_i v_i \right)^T \left(W_{hh}^T\right)^{t-k} = \sum_{i=1}^{n} W_{hh}^{t-k} c_i v_i = \sum_{i=1}^{n} \lambda_i^{t-k} c_i v_i$$

Let c_j be the first non-zero coefficient in decomposition (1) (so $c_m = 0$ for $m < j$). Then

$$\frac{\partial \mathcal{L}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} = \sum_{i=j}^{n} \lambda_i^{t-k} c_i v_i = \lambda_j^{t-k} c_j v_j + \lambda_j^{t-k} \sum_{i=j+1}^{n} \left(\frac{\lambda_i}{\lambda_j} \right)^{t-k} c_i v_i \approx \lambda_j^{t-k} c_j v_j$$

Thus, from $|\lambda_1| < 1$ follows that $|\lambda_j| < 1$ and $\lambda_j^{t-k} c_j v_j$ exponentially tends to 0. Similarly, the gradient norm explodes when $|\lambda_j| > 1$ (so is $|\lambda_1|$). □

3 Existing solutions overview

There exists a number of suggested methods for overcoming the vanishing and exploding gradient problem. We can divide them in 3 categories, and we will mention a few examples of methods for each category:

1. Tackling both exploding and vanishing gradients:
   - Unitary parametrization of the recurrent matrix [1] – unitary matrices have the norm equal to 1.
2. Tackling exploding gradients:
   - Gradient clipping [7]: rescaling gradients whose norm exceeds the predefined threshold so that it would be equal to that threshold.
3. Tackling vanishing gradients:
   - Gated architectures like GRU [2], LSTM [3]: contain gates which control the information flow from, to and within the memory cell. They allow learning longer dependencies, but have 3 and 4 times more parameters respectively.

- Regularization forcing the error signal to preserve norm during BPTT [7].
- Initialization tricks assuming Rectified Linear Unit (ReLU) nonlinearity: initilization with identity matrix called IRNN [5], with normalizaed positive-definite matrix called npRNN [9].

We will further focus on initialization tricks in more detail.

4 ReLU and normalized positive-definite initialization

4.1 Motivation

Recall the bound on the gradient norm:

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq \left\| W_{hh}^T \right\|^{t-k} \max_{i=k,\dots,t-1} \left\| \text{Diag}\left(f'(h_i) \right) \right\|^{t-k}$$

Notice that besides the norm of the recurrent matrix W_{hh}, the norm of the activation function derivative is also involved. The most common nonlinearity choices are sigmoid, hyperbolic tangent and rectified linear unit (ReLU). It can be shown that $|\sigma'(x)| \leq \frac{1}{4}$, hence the gradients might be more prone to vanishing with this nonlinearity choice. Though for hyperbolic tangent $|\tanh'(x)| \leq 1$, its derivative is very close to 0 on the major part of the number line, which also potentially causes gradients to vanish. In the case when $f(x)$ is ReLU we have $f'(x) = 1$ for $x > 0$ and 0 otherwise. Consequently, among these three, ReLU seems to be a reasonable activation function choice. In both IRNN and npRNN initialization tricks, the importance of the choice of ReLU nonlinearity for recurrent layer is stressed and further initialization techniques are studied with this assumption.

In [5] identity matrix initialization was proposed with quite intuitive reasoning: in case of no input x_t and $f(x) = x$, hidden states h_t will infinitely stay the same.

The authors of [9] looked at RNN from a dynamical system perspective and provided the following motivation for the normalized positive-definite initialization. Assuming no input x_t and $b_1 = 0$, we can visualize what can happen to 2-dimensional hidden state vectors depending on the eigenvalues λ_1, λ_2 of W_{hh}. From the figure 2 we can see that h_t rotates when W_{hh} with complex eigenvalues is consistently applied to it. Applying the recurrent matrix with negative eigenvalue results in reflection across some direction. This yields that hidden state vectors are very likely to shrink to the origin if ReLU is applied after W_{hh} in these two cases. Thus, we may want to start with matrix W_{hh} which has real positive eigenvalues, or positive-definite matrix. As the norm of W_{hh} plays the key role in gradient problems predisposition, we normalize initial matrix so that it would have the norm 1. Finally, we obtain the following initialization procedure proposed in [9]:

$$\text{positive-definiteness} \begin{cases} R \sim \mathcal{N}(0,1), \quad R \in \mathbb{R}^{N \times N} \\ A = \frac{1}{N} R^T R \end{cases}$$

$$\text{normalization} \begin{cases} e = \max(\lambda(A + I_N)) \\ W_{hh} = \frac{A + I_N}{e} \end{cases}$$

where N is the number of neurons in recurrent layer and I_N is identity matrix of size N. In figure 3 we can see that with no input and $b_1 = 0$, after iteratively applying W_{hh} and ReLU, h_t will converge to a manifold corresponding to $\lambda_1 = 1$.

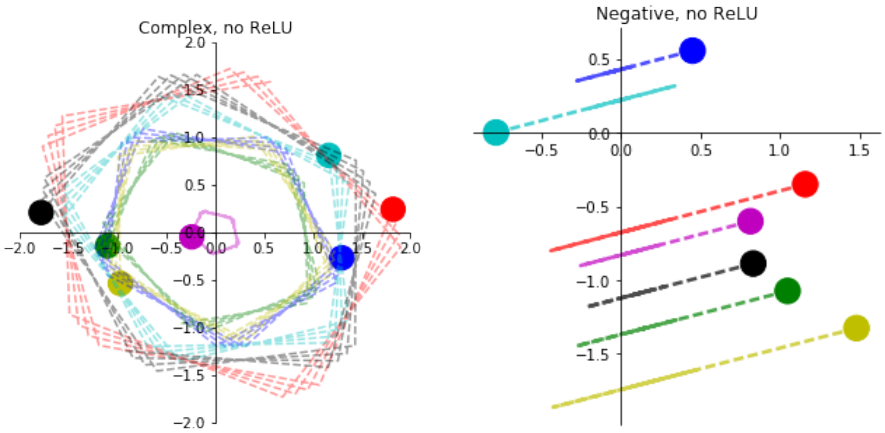


Figure 2: The points represent initial hidden states h_0 in 2-dimensional space, then their trajectories are plotted after W_{hh} acted on them multiple times. On the left, W_{hh} had normalized complex eigenvalues, so it was a rotation matrix. On the right, W_{hh} had a negative eingenvalue.

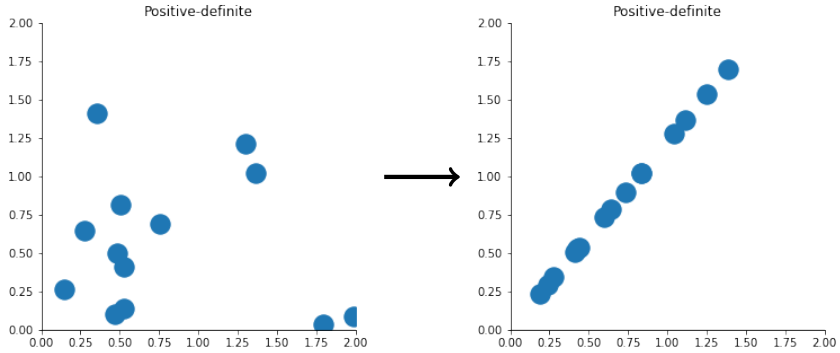


Figure 3: The points on the left represent initial hidden states h_0 in 2-dimensional space, the right picture shows where they converged to after W_{hh} and ReLU were consistently applied to them multiple times.

4.2 Reproducing experiments

The authors of [9] conducted a range of experiments and compared a few initialization strategies including the ones summarized in the table below:

RNN type	description
sRNN	random Gaussian initialization
IRNN	identity matrix initialization
nRNN	normalized random Gaussian initialization
npRNN	normalized-positive definite initialization

To clarify, nRNN is a Gaussian random matrix normalized by its eigenvalue with the highest norm.

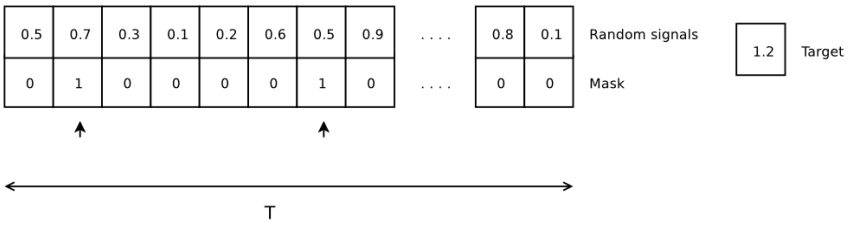


Figure 4: Addition problem illustration. Scheme is taken from [5]

First, the authors address the *addition problem*: input x_t is 2-dimensional vector, where $(x_t)_1 \sim \mathrm{U}[0;1]$ and $(x_t)_2 \in \{0,1\}$. $(x_t)_1$ are sampled independently for $t = 1, \dots T$ and $\sum_{t=1}^{T}(x_t)_2 = 2$, so only two vectors from the sequence have non-zero second coordinate. The target y for the input sequence is $\sum_{t=1}^{T}(x_t)_1 \cdot (x_t)_2$, or, putting it into words, the sum of the numbers in the first sequence after applying the mask sequence (see figure 4). The baseline solution here would be to always predict 1 as the expectation of the sum of two independent standard uniformly distributed variables. It is clear that the probability density function of the sum of two independent uniform variables ξ and η is:

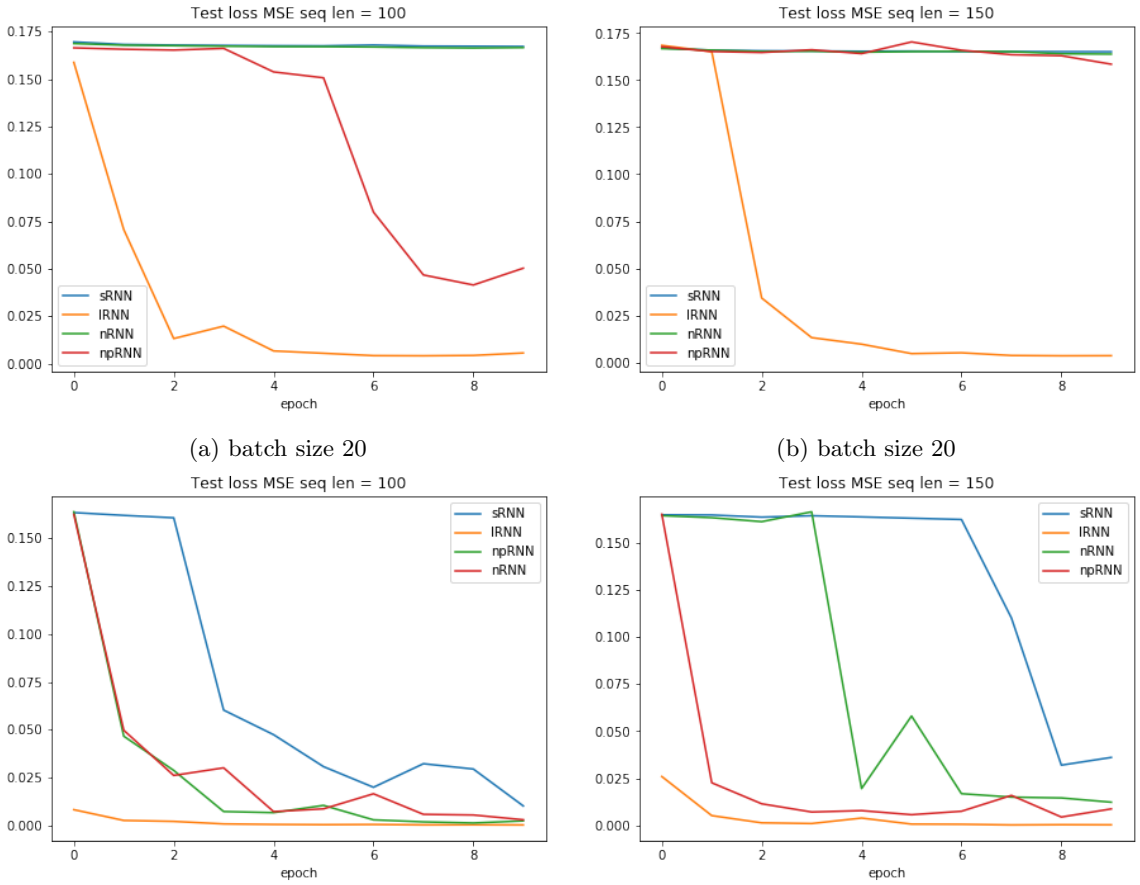
$$f_{\xi+\eta}(t) = \begin{cases} 0, & \text{if } t \notin [0,2] \\ t, & \text{if } t \in [0,1] \\ 2-t, & \text{if } t \in [1,2] \end{cases}$$

Consequently, baseline solution gives Mean Squared Error (MSE):

$$\mathbb{E}[\xi + \eta - 1]^2 = \mathbb{E}[\xi + \eta]^2 - 1 = 0.1(6) \approx 0.167$$

For the purpose of testing RNN's ability to learn long range dependencies, the first non-zero mask element is taken randomly within the first $T/10$ steps and the second – within the last $T/2$ steps. As well as the authors of the mentioned paper, we varied T from 150 to 400, also testing RNN's performance on shorter sequences.

According to the results presented in the paper, npRNN performed at par or better than any other initialization strategy for sequences of length ranging from 150 to 400. Unfortunately, we were not able to gain the same quality reproducing this experiment, though used the same RNNs' architectures and experiment protocol. Our results are shown in figure 5. IRNN significantly outperformed other initialization techniques which may be due to the fact that it is less likely to suffer from the vanishing gradients as we start from the matrix with all the eigenvalues equal to 1. Nevertheless, on longer sequences IRNNs are rather unstable as gradient norm sometimes become too large, and it complicates optimization. Also, we see that npRNN showed better performance than nRNN which could not be caused by different initial distribution of eigenvalues, so it was the motivation for studying the effect of positive-definiteness constraint on W_{hh}.



(a) batch size 20

(b) batch size 20

(c) batch size 1

(d) batch size 1

Figure 5: Test MSE for sequences of length 100 (left column) and 150 (right column). In the first row, batch size 20 was used during SGD, in the second row batch size 1 (as well as in [9]). With batch size 20, for sequences of length 150 and longer only IRNN was able to achieve a good score, whereas other RNNs were stuck in the baseline prediction. With batch size 1, IRNN still shows the best performance.

5 Positive-definiteness constraint

We used the following natural positive-definite parametrization for the recurrent matrix:

$$W_{hh} = X^T X$$

So, the gradients are taken with respect to X instead of W_{hh} during BPTT. We can use either IRNN or npRNN initialization, we test both in all the experiments. We will further denote positive-definite constrained RNNs with npRNN initialization by *posdef-npRNN* and positive-definite constrained RNNs with IRNN initialization by *posdef-IRNN*. When we compare posdef-npRNN to unconstrained npRNNs with initial recurrent matrix W_{hh}, the initial value for X is taken as follows:

$$W_{hh} = T^{-1}\Lambda T, \quad X = T^{-1}\sqrt{\Lambda}T$$

6 Experiments

We conduct experiments on toy problems similar to ones from npRNN paper: addition problem, MNIST classification and permuted MNIST classification, and additionally, on text generation task using Shakespeare dataset. For all the experiments we used vanilla RNN with one hidden layer.

6.1 Addition problem

The problem formulation can be found in section 4.2. Firstly, we tried to reproduce the experiment from npRNN paper comparing sRNN, nRNN, IRNN and npRNN initializations. Consistently with [9], we generated train set of 100000 examples and test set of 10000 examples of length ranging in {70, 100, 150, 200, 300, 400}. We used 100 hidden units, and trained RNN using SGD BPTT for 10 epochs with fixed learning rate 10^{-3} and gradient clip 10. Some results were shown in figure 5, more results with batch size 1 are presented in figure 6. From the plots, we see that IRNN converges faster that any other RNN, but it is absent from the figure with sequences of length 300, as the error and gradients often extremely explode for such long sequences. npRNN shows decent performance but converges slower and to worse optimum, it is also quite unstable on longer sequences.

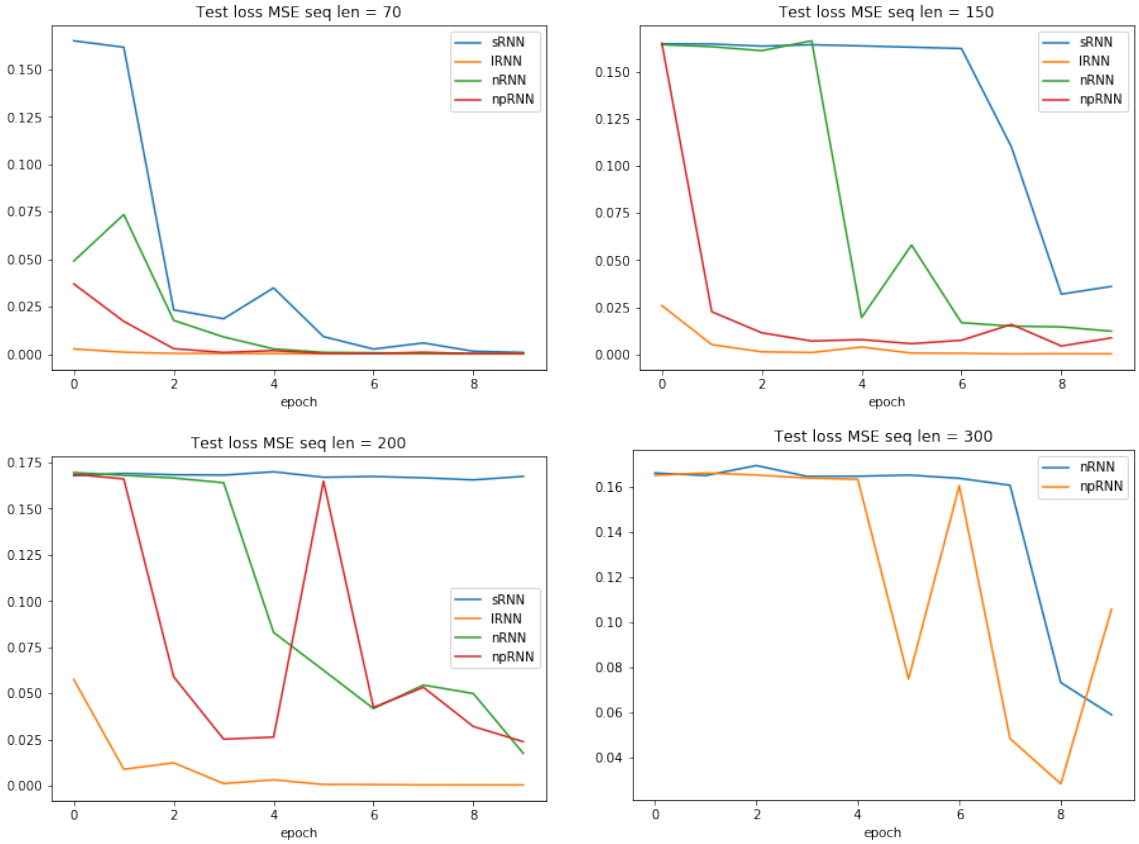


Figure 6: Test MSE for sequences of length 70, 150, 200 and 300. The experiment protocol is described above in detail.

For testing posdef-npRNN and posdef-IRNN performance, we needed to carry out grid search on learning rate and gradient clip parameters, as for these constrained RNNs gradients are of a different order of magnitude. For posdef-npRNN we ranged over learning rates $\{10^{-3}, 10^{-2}, 10^{-1}\}$ and over gradient clips

$\{10^{-1}, 10^{0}, 10^{1}\}$. For posdef-IRNN we used the grid $\{10^{-4}, 10^{-3}, 10^{-2}\}$ for learning rates (as they are more susceptible to gradient exploding) and the same grid for gradient clip. We performed this grid search for sequences of length 150 and batch size was set to 20 for computation speed. The results are shown in figures 7 and 8. We can conclude that the learning progress for both constrained and unconstrained RNNs significantly depends on hyperparameters, but constrained ones seem to be even more sensitive. The learning progress of npRNN and posdef-npRNN is comparable, the best score is achieved with learning rate 0.1 and gradient clip 0.1. posdef-IRNN often performed worse and less stably than unconstrained IRNN, whereas IRNN reaches the best MSE score with 2 sets of hyperparameters: learning rate 0.01 and gradient clip 1, learning rate 0.001 and gradient clip 10.

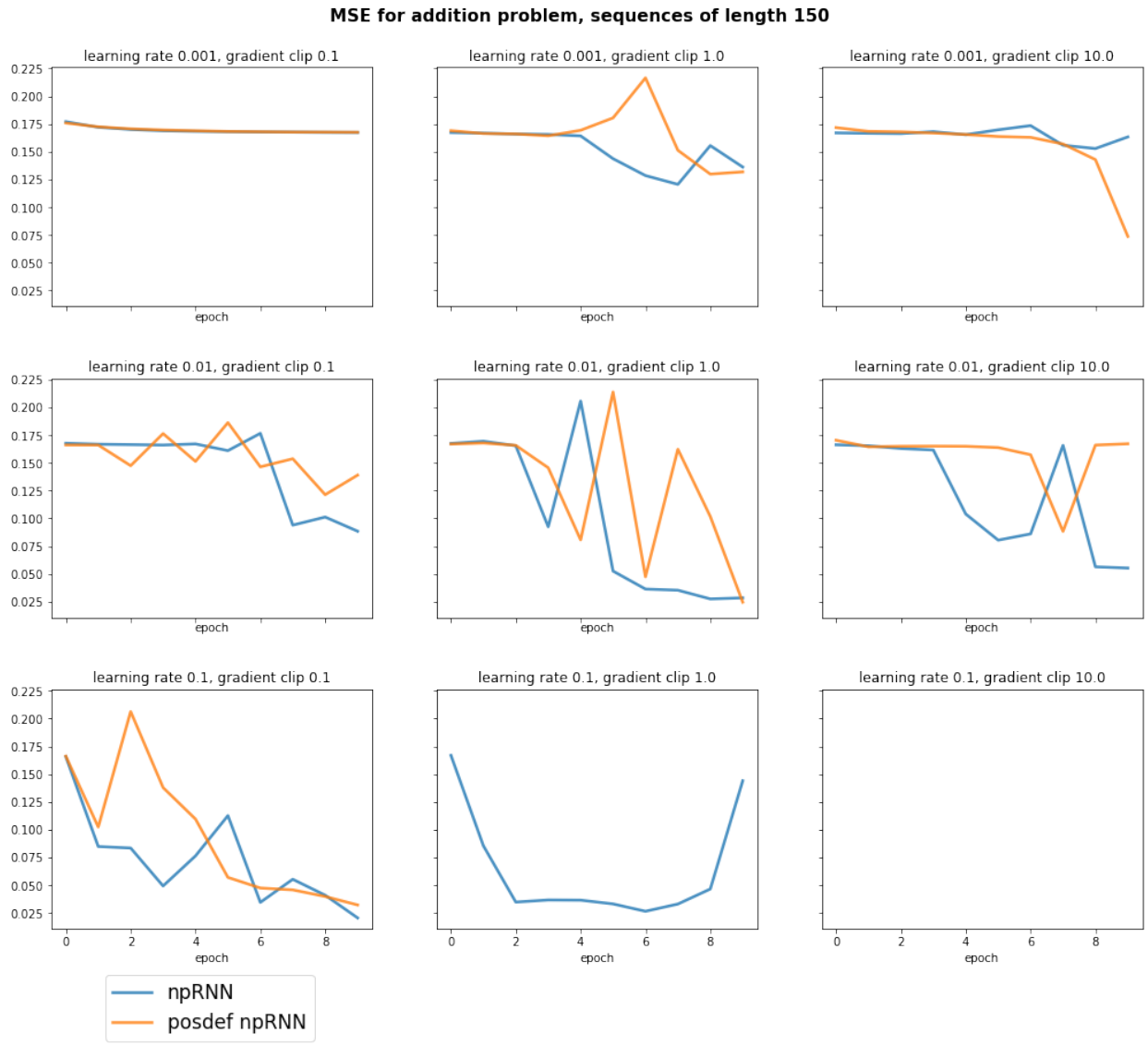


Figure 7: Grid search on learning rate and gradient clip parameters for npRNN and posdef-npRNN, we vary learning rate by row and gradient clip by column. Some plots are absent due to the gradient and error explosion.

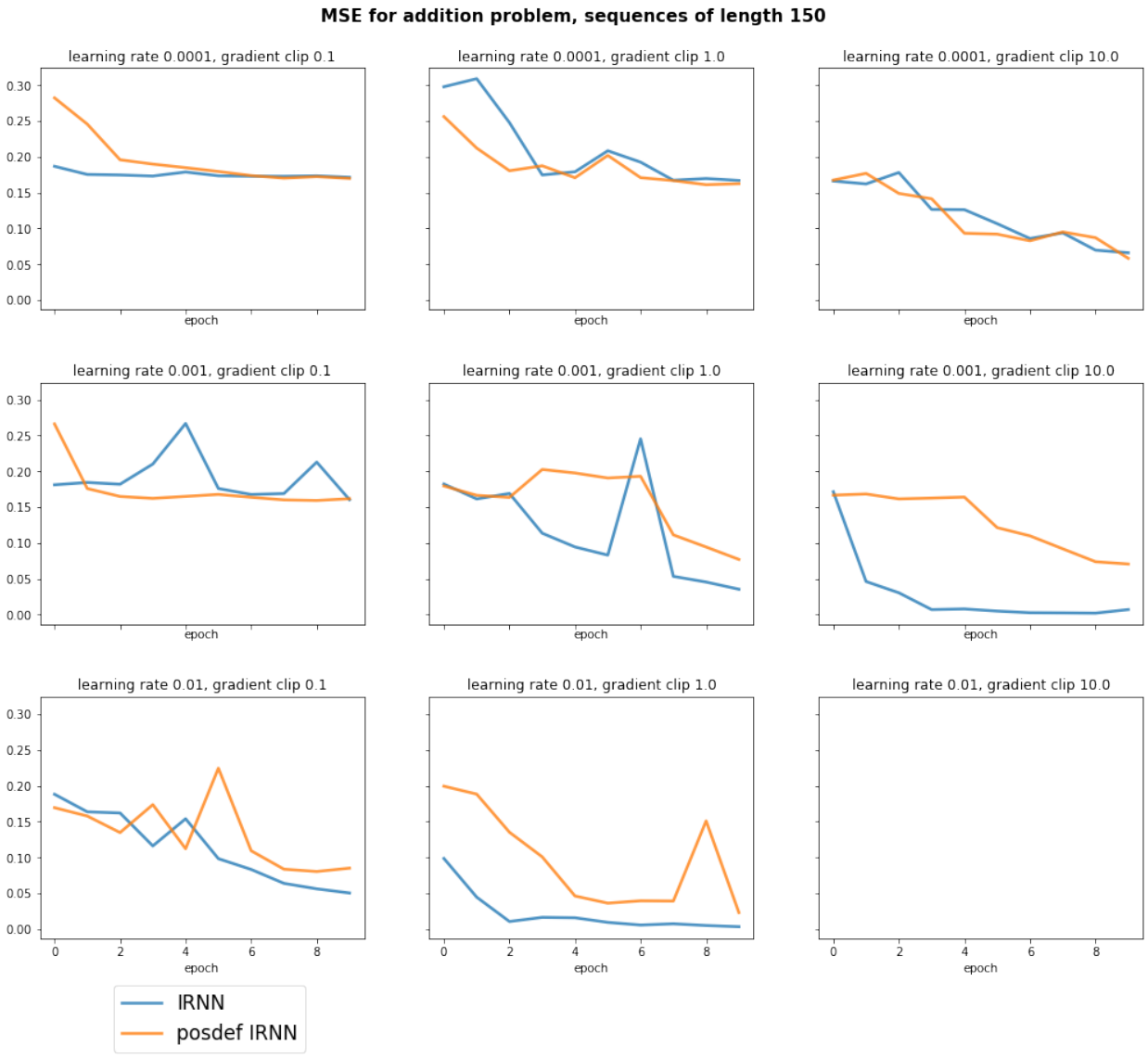


Figure 8: Grid search on learning rate and gradient clip parameters for IRNN and posdef-IRNN, we vary learning rate by row and gradient clip by column. Some plots are absent due to the gradient and error explosion.

We also plot the test error, gradient norm and maximum, median and minimum absolute eigenvalue at each iteration for IRNN, posdef-IRNN and posdef-npRNN (see figure 9). We used sequences of length 300, batch size 20, learning rate 10^{-3} and gradient clip 10. Note that for IRNN the test error starts to decrease at the time of the gradient explosion. We may see that posdef-IRNN aggravates exploding gradient problem, whereas posdef-npRNN suffers from the vanishing gradients (the gradient norm fluctuates between 10^{-1} and 10^{1} and the median eigenvalue stays somehere under 0.4), and both cannot achieve good performance.

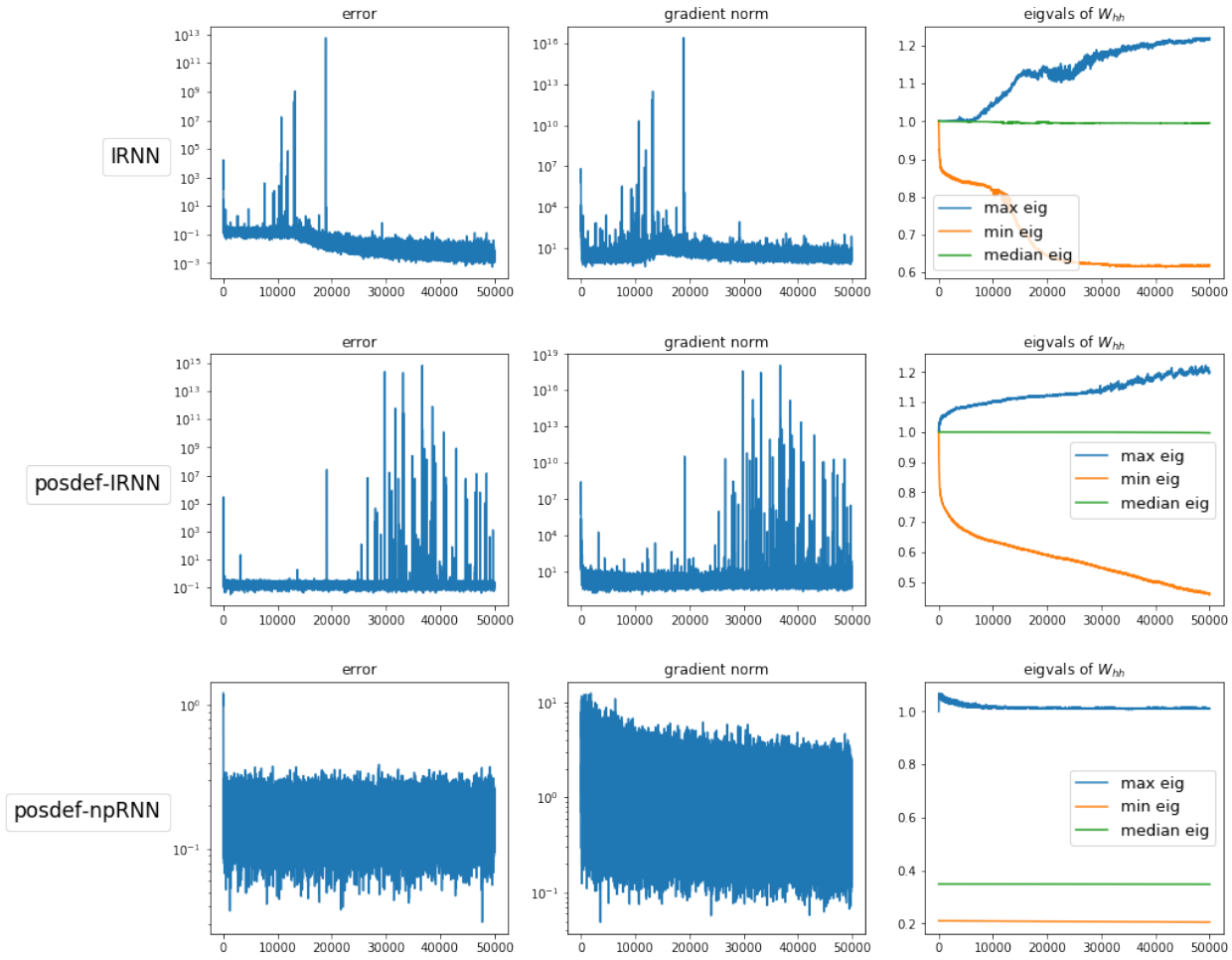


Figure 9: From left to right: test MSE, gradient norm (before clipping) and maximum, median and minimum eigenvalue norm after every iteration during training of IRNN, posdef-IRNN and posdef-npRNN.

6.2 MNIST classification

Another toy problem is pixel-by-pixel MNIST [6] classification. All the sequences have length 784, there are 60000 objects in train set and 10000 in test set. It was very hard to train RNNs with simple SGD for this task as it demanded subtle learning rate cooling, especially for posdef-RNNs with large gradients, so we train RNNs with 100 hidden units using Adam [4] for 100 epochs with initial learning rate 10^{-6}, batch size 16 and gradient clip 10. The objective function is crossentropy, but we also plot test accuracy, see figure 10.

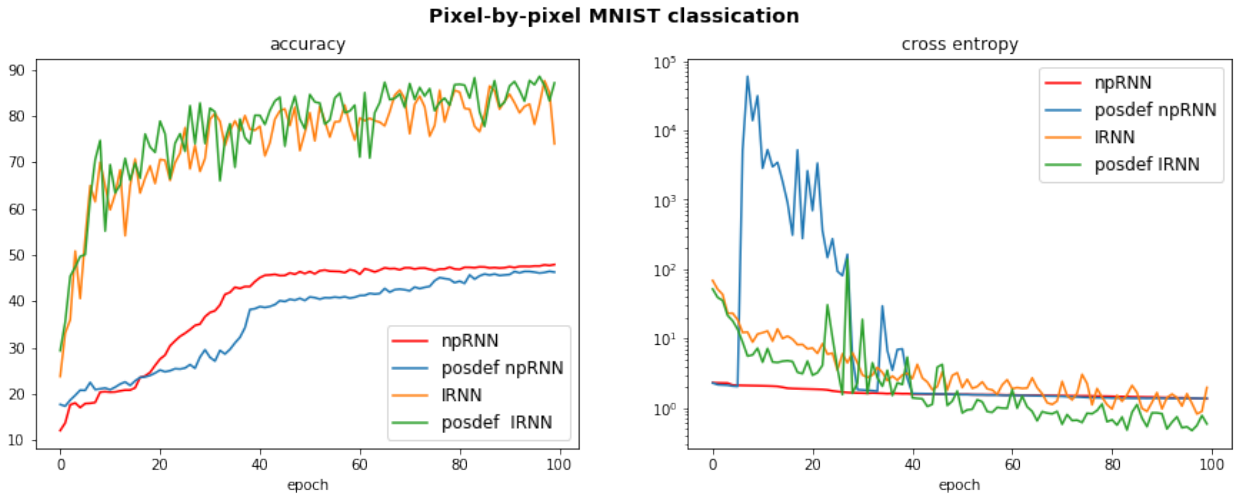


Figure 10: Test accuracy (left) and test crossentropy (right) after each epoch for MNIST classification.

We also apply a fixed permutation to all the sequences and test RNNs performance on such a transformed dataset which is considered to be a more challenging task. The results are plotted in figure 11.

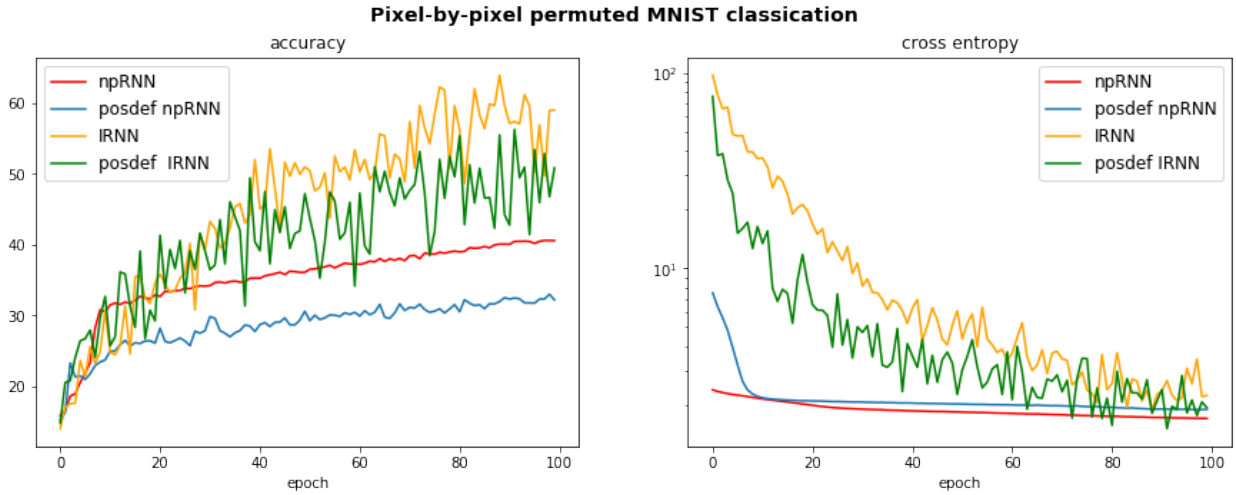


Figure 11: Test accuracy (left) and test crossentropy (right) after each epoch for permuted MNIST classification.

We can see that npRNN and posdef-npRNN performed considerably worse than IRNN counterparts. posdef-npRNN worked noticeably worse than unconstrained npRNN, while posdef-IRNN and IRNN have both worked comparably well – for pixel-by-pixel MNIST IRNN and posdef-IRNN reached test accuray around 87% and posdef-IRNN even achieved a lower crossentropy value; for permuted MNIST IRNN reached test accuracy around 60%. Nonetheless, both IRNN and posdef-IRNN were quite unstable during the learning.

6.3 Shakespeare text generation

Finally, we test RNNs on a more real-world task – text generation on Shakespeare dataset. We train charRNNs which means that the input is one-hot encoded symbol from the dictionary of size 65 (taken from the train set). RNNs have one hidden layer with 256 units, SGD optimizer is used with a fixed learning rate of 0.01 and batch size of 10, gradient clip parameter is set to 10. For one epoch we sample 1000 random subsequences from train text and test on 100 subsequences, in both cases subsequences are of length 100, and we conduct 70 such epochs. The objective function is cross-entropy. The result are presented in figure 12.

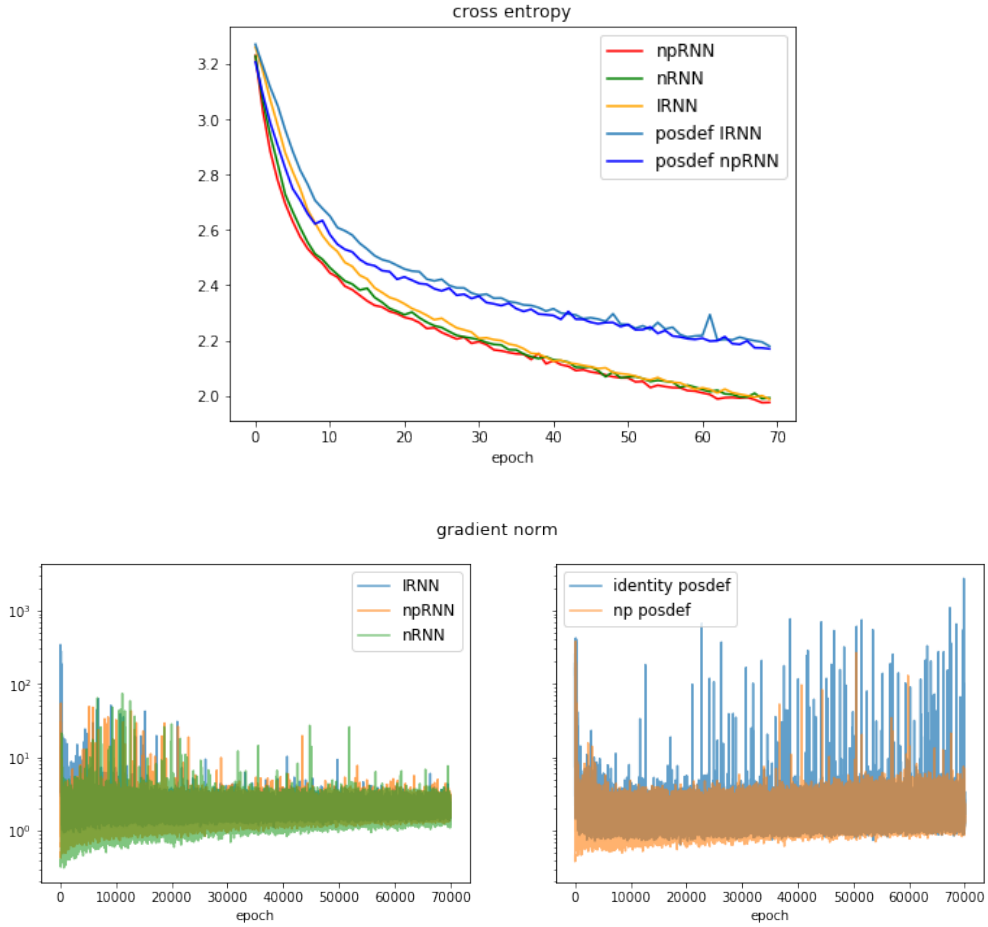


Figure 12: Test crossentropy after each epoch (top) and gradient norm after each iteration (left and right) for Shakespeare text generation.

Unconstrained RNNs converge to a better optimum comparing to posdef-RNNs. After showing quite poor performance on toy problems, npRNN converged faster than any other RNN: this can be explained by the fact that text generation does not require very long sequential dependencies.

7 Conclusion

Our results for toy problems with long temporal dependencies differ from those obtained in paper about npRNN, however in text generation task such initialization choice lead to good convergence. Positive-definiteness constraint on recurrent matrix leads to unstable gradients and gives no actual benefit in per-

formance for the experiments discussed above. It might be that the problem is in a wrong parametrization choice, so we would like to try different parametrization options and find a more stable procedure in the future work.

References

[1] Martın Arjovsky, Amar Shah, and Yoshua Bengio. "Unitary Evolution Recurrent Neural Networks". In: *CoRR* abs/1511.06464 (2015). URL: http://arxiv.org/abs/1511.06464.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2014). URL: http://arxiv.org/abs/1409.0473.

[3] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[4] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). URL: http://arxiv.org/abs/1412.6980.

[5] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units". In: *CoRR* abs/1504.00941 (2015). URL: http://arxiv.org/abs/1504.00941.

[6] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[7] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "Understanding the exploding gradient problem". In: *CoRR* abs/1211.5063 (2012). URL: http://arxiv.org/abs/1211.5063.

[8] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6. URL: http://dl.acm.org/citation.cfm?id=65669.104451.

[9] Sachin S. Talathi and Aniket Vartak. "Improving performance of recurrent neural network with relu nonlinearity". In: *CoRR* abs/1511.03771 (2015). URL: http://arxiv.org/abs/1511.03771.