On the difficulty of training Recurrent Neural Networks
Additional material

1 Analytical analysis of the exploding and vanishing gradients problem

1.1 Linear model

$$\mathbf{x}_t = \mathbf{W}_{rec}\sigma(\mathbf{x}_{t-1}) + \mathbf{W}_{in}\mathbf{u}_t + \mathbf{b} \tag{1}$$

Let us consider the term $\mathbf{g}_k^T = \frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k}\frac{\partial^+ \mathbf{x}_k}{\partial \theta}$ for the linear version of the parametrization in equation (1) (i.e. set σ to the identity function) and assume t goes to infinity and $l = t - k$. We have that:

$$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \left(\mathbf{W}_{rec}^T\right)^l \tag{2}$$

By employing a generic *power iteration method* based proof we can show that, given certain conditions, $\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}\left(\mathbf{W}_{rec}^T\right)^l$ grows exponentially.

Proof Let $\mathbf{W}_{rec}$ have the eigenvalues $\lambda_1, .., \lambda_n$ with $|\lambda_1| > |\lambda_2| > .. > |\lambda_n|$ and the corresponding eigenvectors $\mathbf{q}_1, \mathbf{q}_2, .., \mathbf{q}_n$ which form a vector basis. We can now write the row vector $\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}$ into this basis:

$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} = \sum_{i=1}^N c_i \mathbf{q}_i^T$

If j is such that $c_j \neq 0$ and any $j' < j, c_{j'} = 0$, using the fact that $\mathbf{q}_i^T\left(\mathbf{W}_{rec}^T\right)^l = \lambda_i^l \mathbf{q}_i^T$ we have that

$$\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t}\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = c_j\lambda_j^l\mathbf{q}_j^T + \lambda_j^l\sum_{i=j+1}^{n} c_i\frac{\lambda_i^l}{\lambda_j^l}\mathbf{q}_i^T \approx c_j\lambda_j^l\mathbf{q}_j^T \tag{3}$$

We used the fact that $\left|{}^{\lambda_i}/_{\lambda_j}\right| < 1$ for $i > j$, which means that $\lim_{l\to\infty}\left|{}^{\lambda_i}/_{\lambda_j}\right|^l = 0$. If $|\lambda_j| > 1$, it follows that $\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k}$ grows exponentially fast with l, and it does so along the direction $\mathbf{q}_j$. □

The proof assumes $\mathbf{W}_{rec}$ is diagonalizable for simplicity, though using the Jordan normal form of $\mathbf{W}_{rec}$ one can extend this proof by considering not just

the eigenvector of largest eigenvalue but the whole subspace spanned by the eigenvectors sharing the same (largest) eigenvalue.

This result provides a necessary condition for gradients to grow, namely that the spectral radius (the absolute value of the largest eigenvalue) of $\mathbf{W}_{rec}$ must be larger than 1.

If $\mathbf{q}_j$ is not in the null space of $\frac{\partial^+ \mathbf{x}_k}{\partial \theta}$ the entire temporal component grows exponentially with l. This approach extends easily to the entire gradient. If we re-write it in terms of the eigen-decomposition of $\mathbf{W}$, we get:

$$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{j=1}^{n} \left(\sum_{i=k}^{t} c_j \lambda_j^{t-k} \mathbf{q}_j^T \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right) \tag{4}$$

We can now pick j and k such that $c_j \mathbf{q}_j^T \frac{\partial^+ \mathbf{x}_k}{\partial \theta}$ does not have 0 norm, while maximizing $|\lambda_j|$. If for the chosen j it holds that $|\lambda_j| > 1$ then $\lambda_j^{t-k} c_j \mathbf{q}_j^T \frac{\partial^+ \mathbf{x}_k}{\partial \theta}$ will dominate the sum and because this term grows exponentially fast to infinity with t, the same will happen to the sum.

2 Clipping threshold and regularization term

2.1 Clipping threshold

In Fig 1 we can see the overall norm during training as well as a zoom in on a portion. These plots show the exploding gradient problem at work on the temporal order task, when is learned by varying the length of the sequence between 50 to 200 steps (see section below).

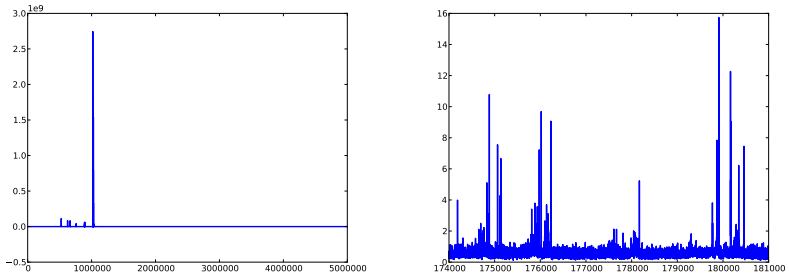

Figure 1: x-axis shows update index, while y-axis shows the norm of the gradients for a learning run on the temporal order task, where the cutoff threshold for clipping was set to 6. On the left we have the plot for the entire run, while on the right we have zoomed on a portion of the learning process

Setting the threshold for clipping should be done by examining plots as the one in Fig. 1. The average norm of the gradient can depend greatly on the

architecture and task you run. For example if the cost is computed as a sum versus a mean (over the minibatch or sequence length), it can make a big impact. Square error versus negative log likelihood also affects the average norm. It is crucial to protect the model against large steps (like the one of size 1e9 on the left plot). Doing this is quite trivial. However if we look on the zoomed version of the plot is not clear if smaller peaks are harmful. Is a step of norm 4 too large? Or 6 ?

Our intuition tells us that we should look at the average norm of the gradient, which in the plot above seems to be just below 1. If we set the value close to 1, the we are ensured that we will not take any large step, though with higher probability we will end up clipping gradients (the smaller peaks like the ones at 4 or 6.) and affecting the trajectory taken by MSGD. Note that while any clipped gradient still points in a descent direction for the current minibatch, the expected value of the clipped gradients is not pointing in the same direction as the gradient on the training set (i.e. by clipping we are artificially giving more weight to certain training examples). In some sense we would like clipping to happen as rarely as possible in order to not affect convergence. Further more a step of 4, while being somewhat large, might not even be harmful.

If we set the value further away from 1, we will allow larger steps, that, with a high probability (depending on how high the threshold is), might locally disrupt learning. Our intuition is that while these steps will not take as too far away from the valley we are in, they can slow down convergence, as MSGD might have to find the valley again. This effect is also attenuated by the learning rate (that scales the norm of the step). With lower learning rates we can allow the threshold to be larger, and hence interfere less with the true descent direction of MSGD, while with higher learning rate we might need smaller thresholds for stability.

It is not clear which is a better options. While clipping seems to interfere with learning the empirical distribution (by reweighing examples) we had not seen this effect playing a large role in practice. On the other hand large thresholds also lead to convergence, only making convergence slower at times.

We make another point, namely that of numerical stability. While clipping the gradient should always work, when dealing with only 32 bits, gradients can quickly grow enough for one not to be able to get an accurate norm (i.e. we have seen NaN creeping in our computations, albeit very rarely). For this reason whenever we get not numeric values in the gradients (or larger than a set threshold, which we fixed to 1e10 in our later experiments) we replace the gradient of the recurrent weights by $.02 * \mathbf{W}_{rec}$ and any other gradient by 0. This ensures that the largest singular value of $\mathbf{W}_{rec}$ decreases at this step, hopefully taking the model in a more reliable regime. In practice this events are extremely rare.

2.2 Regularization term

The regularization term is given by the following equation:

$$\begin{aligned}\frac{\partial^+\Omega}{\partial \mathbf{W}_{rec}} &= \sum_k \frac{\partial^+\Omega_k}{\partial \mathbf{W}_{rec}} \\ &= \sum_k \frac{\partial^+\left(\frac{\left\|\frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}}\mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_k))\right\|^2}{\left\|\frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}}\right\|^2}-1\right)^2}{\partial \mathbf{W}_{rec}}\end{aligned} \tag{5}$$

Given that we consider the "immediate" derivative, this term is not expensive to compute, as it does not need to be computed sequentially for each Ω_k. Each value is independent from the others and each derivative can be computed in parallel. More specifically, computing this regularization term for all Ω_k can be express in a matrix form. It only involves a matrix multiply and a few element-wise operations. To get the gradient we only need to traverse this graph in reverse, i.e. the gradient itself only involves two matrix multiplications and some element-wise operations for all Ω_k.

Special care should be taken for the term $\left\|\frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}}\right\|^2$ which can shrink to very small numbers resulting in a division by 0 or instability. For this reason we only consider the components Ω_k for whom $\left\|\frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}}\right\|^2$ is larger than some threshold, that we set and keep fixed to 1e-20.

In practice we have noticed that for complex task (like the polyphonic music prediction), the regularization term can hurt learning. It is possible that by regularizing the model we force it to pay less attention to short term components, which can harm the quality of the local minima one can find. We believe that an effective way of using this term should involve some kind of schedule that gradually decreases the weight of the regularization term in the total cost. We've found that a $\frac{1}{t}$ schedule works, though more investigation is needed.

3 Experimental setup

3.1 The pathological synthetic tasks

For all pathological tasks we look at the percent of correctly classified sequences. For regression tasks (like the addition task) we consider a sequence is correctly classified when the square error is smaller than 0.04. A run (i.e. training a model starting from randomly sampled weights) is considered successful if it makes less than 1% errors in classifying 10000 sequences (similar to Martens and Sutskever (2011)). For both training and testing we always generate data on the fly, i.e. every time we compute the test error we generate a new set of 10000 sequences (the same when we need to do a MSGD step).

3.1.1 Temporal order

For the temporal order the length of the sequence is given by T. We have a fixed set of two symbols $\{A, B\}$ and 4 distractor symbols $\{c, d, e, f\}$. The sequence entries are uniformly sampled from the distractor symbols everywhere except at two random positions, the first position sampled from $[\frac{T}{10}, \frac{2T}{10}]$, while the second

from $[\frac{4T}{10}, \frac{5T}{10}]$. The task is to predict the order in which the non-distractor symbols were provided, i.e. either $\{AA, AB, BA, BB\}$.

We used a gridsearch, where we tried the following values:

- number of hidden units N in 50,100
- learning rate lr in .01,.001
- threshold for gradient clipping μ in 6., 1., 4.
- regularization weight α in 4., 2., 1., .5
- maximal number of updates 100k, 5M

We did not explore any decreasing scheme for the learning rate or regularization weight. No momentum was used here or in any other experiments. The hyperparameters where set based on a few initial jobs with sequences of length 100 (i.e. we did not explore every combination for all possible lengths and all seeds). 100k updates where not sufficient for the models to converge on all possible lengths, and hence we increased this up to 5M. For 5M steps, in worse case scenario, the code had to run close to 48h. In the first experiments we used a larger minibatch of 100 and 1000 examples, which we decrease to 20 to reduce running times. We did not notice any significant instability due to the reduced minibatch size.

The final hyper-parameters used were:

- $N = 50$ (number hidden units)
- $lr = 0.01$ (learning rate)
- $\mu = 1.$ (threshold for gradient clipping)
- $\alpha = 4.$ (regularization term)
- maximal number of updates 5M

Note that other combinations also lead to convergence (for e.g. μ of 6. and smaller learning rate of .001), though we find this combination to converge faster, and work better in the regime of sequences of length 250 steps.

For generating the subplots of figure 7., we explored the following lengths T (i.e. we trained 5 models for each of this value, the 5 models using the same random seeds):

- for *sigmoid* initialization, considered T values are: 16, 20, 22, 50, 100, 150, 200 ,250
- for *basic tanh* initialization, considered T values are : 16, 20, 35, 45, 50, 55, 65, 75, 100, 150, 200, 250
- for *smart tanh* initialization, considered T values are : 16, 20, 50, 100, 150, 200, 250

The number of updates required for convergence greatly depends on the length of the sequence. Using both clipping and the regularization term, on average, for $T = 200$, the *sigmoid* initialization takes 685k updates, the *basic tanh* takes 476k updates, and *smart tanh* 434k updates. In comparison, if we fix $T = 20$ and the initialization to *smart tanh*, we have for MSGD 21k updates, for MSGD-C 11k updates and for MSGD-CR 2k updates.

For figure 7 we stopped an running experiment and consider it a success as soon as the number of misclassified sequences was less than 100 out of 10000.

When training a single model for random lengths from 50 to 200 steps, we decreased the regularization weight α to 2. All values of μ seem to work, though convergence seems faster with $\mu = 1$. All 20 sequences in a training minibatch had the same length T uniformly sampled from $[50, 200]$. The test set was split into 500 minibatches, each of a different length uniformly sampled from $[50, 100]$. Each test evaluation implied generating a new set of such 500 minibatches. For all tasks where we considered varying length sequences, we stopped learning and consider a sequence successful when no sequence was misclassified out of 10000 sequences of varying length. We then evaluate the model on 10000 sequences of length 50, 10000 sequences of length 100, 10000 sequences of length 150 and 10000 sequences of length 200. Additionally we explored T in $\{250, 500, 1000, 2000, 3000, 5000\}$. For the temporal order problem in all cases 0 sequences were misclassified.

3.1.2 Addition problem

The input consists of a sequence of random numbers, where two random positions (one in the beginning and one in the middle of the sequence) are marked. The model needs to predict the sum of the two random numbers after the entire sequence was seen. For each generated sequence we sample the length T' from $[T, \frac{11}{10}T]$, though for clarity we refer to T as the length of the sequence in the paper. The first position is sampled from $[1, \frac{T'}{10}]$, while the second position is sampled from $[\frac{T'}{10}, \frac{T'}{2}]$. These positions i, j are marked in a different input channel that is 0 everywhere except for the two sampled positions when it is 1. The model needs to predict the sum of the random numbers found at the sampled positions i, j divided by 2.

To address this problem we use a 50 hidden units model, using the *basic tanh* initialization. We explored the learning rates $\{.01, .001\}$ and $\alpha \in \{.5, 1., 2.\}$. We have tried $\mu \in \{6., 1.\}$. The final choice of hyper-parameters are $lr = .01$, $\alpha = .5$. We did not notice any significant difference in picking $\mu = 6$ or $\mu = 1$, though clipping was required (i.e. $\mu = \infty$ fails). $\mu = 1.$ seems to result in slightly faster convergence (on average 1.226M versus 1.299M updates).

We directly explored training the model on sequences of varying length T between 50 and 200, following the same procedure as for the temporal order task.

A single model manages to handle (within the permitted 1% error) lengths of 50,100, 150, 200, 250, 300, 400 and even 600. All the 5 seeds we've explored

T	50	100	150	200	250	300	350	400	600
error	.08%	.01%	.01%	0%	0%	.02%	.01%	.04%	.52%

Table 1: Addition task. Error as percentage of misclassified sequences out of 10000 for different values of T for the same trained model.

T	50	100	150	200	250	300	350	400	600
error	.55%	.04%	.04%	.04%	.08%	.26%	.28%	.73%	2.11%

Table 2: Multiplication task. Error as percentage of misclassified sequences out of 10000 for different values of T for the same trained model.

ended up in successfully trained model, which outperforms the results presented in Martens and Sutskever (2011) (using Hessian Free), where we see a decline in success rate as the length of the sequence gets closer to 200 steps. Hochreiter and Schmidhuber (1997) only considers sequences up to 100 steps. Jaeger (2012) also addresses this task with 100% success rate, though the solution does not seem to generalize well as it relies on very large output weights. In the table 3.1.2 you can see how the misclassification error behaves for different lengths for a trained model.

3.1.3 Multiplication problem

This task is similar to the problem above, just that the predicted value is the product of the random numbers instead of the sum. We used the same hyper-parameters as for the previous case (without validating them through a grid search), and obtained very similar results (table 3.1.3). We only explored the value of μ which seemed to reduce more the number of updates required than before (1.728M updates for $\mu = 1$ versus 3.418M updates for $\mu = 6$).

3.1.4 3-bit temporal order problem

Similar to the temporal order problem, except that we have 3 random positions, first sampled from $[\frac{T}{10}, \frac{2T}{10}]$, second from $[\frac{3T}{10}, \frac{4T}{10}]$ and last from $[\frac{6T}{10}, \frac{7T}{10}]$.

As before, we explored directly training the model on sequences of varying length. We explored the number of hidden units in $\{50, 100\}$, learning rate in $\{.01, .001\}$ and $\mu \in \{1., 6.\}$. We explored $\alpha \in \{2., 1.\}$. Best combination (with regard to convergence speed) was $lr = .01$, $\mu = 1.$, $\alpha = 2$. We chose 100 hidden units. Average number of updates to train was 569k. Table 3.1.4 describes a single trained model.

T	50	100	150	200	250	300	350	400	600
error	.0%	.01%	.0%	.06%	.01%	0%	2.75%	14.13%	24.81%

Table 3: 3-bit temporal order task. Error as percentage of misclassified sequences out of 10000 for different values of T for the same trained model.

T	50	100	150	200	250	300	350	400	600
error	.04%	.01%	.02%	.03%	.02%	0%	.03%	.02%	.04%

Table 4: Random permutation task. Error as percentage of misclassified sequences out of 10000 for different values of T for the same trained model.

Task	50 steps	100 steps	150 steps	200 steps
Original task	0%	0%	0%	0%
Extended task	0.844%	0.724 %	0.75%	0.77%

Table 5: Noiseless memorization problem. Average (over the 5 seeds) percentage of misclassified sequences out of 10000 for the different lengths considered and the two variations of the task

3.1.5 Random permutation problem

In this case we have a dictionary of 100 symbols. Except the first and last position which have the same value sampled from $\{1, 2\}$ the other entries are randomly picked from $[3, 100]$. The task is to do next symbol prediction, though the only predictable symbol is the last one.

We explored 50 and 100 hidden units, with a learning rate of either .01 and .001. $\alpha \in \{2., 1., .5\}$ and $\mu \in \{6., 1.\}$. We run experiments only on varying length sequences. The task proved harder to train. 2 out of 5 runs were successful for the final chosen hyper-parameters: 100 hidden units, with a learning rate of .001 and $\alpha = 1.$ and $\mu = 6$. The error for different lengths are listed in table 3.1.5 for one of the two successful run.

3.1.6 Noiseless memorization problem

For the noiseless memorization we are presented with a binary pattern of length 5, followed by T steps of constant value. After these T steps the model needs to generate the pattern seen initially. We also consider the extension of this problem from Martens and Sutskever (2011), where the pattern has length 10, and the symbol set has cardinality 5 instead of 2.

All runs on varying length sequences failed, so we trained a different model for the considered lengths (50, 100, 150, 200).

We explored $\alpha \in \{1, 2\}$, learning rate in $\{.01, .001\}$, number of hidden units in $\{50, 100\}$ and $\mu \in \{1., 6.\}$. We explored hyper-parameters on the original task (where the symbol set had cardinality 2). We chose to use $\alpha = 1$, learning rate 0.01, 100 hidden units and $\mu = 1$ for faster convergence.

We manage a 100% success rate (i.e. all 5 runs had under 1% misclassified sequences out of 10000) on these tasks for 5 different random seeds, though we train 5 models for each of the 4 possible lengths (50, 100, 150, 200).

3.2 Natural Tasks

We use *smart tanh* initialization in all our latest results, though preliminary experiments with *basic tanh* seem to perform equally well.

3.2.1 Polyphonic music prediction

We train our model, a sigmoid units RNN, on sequences of 200 steps. The cut-off coefficient threshold is the same in all cases, namely 8 [1].

In case of the Piano-midi.de dataset we use 300 hidden units and an initial learning rate of 1.0. For all natural tasks we halved the learning rate every time the error over an epoch increased instead of decreasing. For the regularized model we used a initial value for regularization coefficient α of 0.5, where α follows a $\frac{1}{2t}$ schedule, i.e. $\alpha_t = \frac{1}{2t}$ (where t measures the number of epochs). We found that keeping alpha constant as before results in worse performance for $\alpha \in \{0.5, 1., 2.\}$. Further more we speculate that for tasks where short term information is important one needs to use a decreasing schedule for the regularization weight.

For the Nottingham dataset we used 400 hidden units, with an initial $\alpha = 5$. that started decreasing with $1/max(1,t-10)$ after the first 10 epochs. μ was set to 8. The learning rate was set to 1. 100 of the 400 hidden units where leaky integration units (Bengio *et al.*, 2012) with their leaky factor randomly sampled from $[0.02, 0.2]$.

For MuseData we used 400 units. The learning rate was also decreased to 0.5. For the regularized model, the initial value for α was 0.1, and $\alpha_t = \frac{1}{2t}$. 100 of the 400 hidden units where leaky integration units, as used in Bengio *et al.* (2012). Their leaky-integration factor was randomly sampled in $[0.02, 0.2]$.

We have explored number of hidden units in the range $\{200, 300, 400\}$, learning rates of $\{2., 1., .5\}$, $\mu \in \{4, 8, 12\}$ and $\alpha \in \{1, 5, .5\}$. We tried keeping α constant, or start decreasing from epoch 0, epoch 10 or epoch 20. We use either $\frac{1}{t}$ or $\frac{1}{2t}$ schedule. Not all hyper-parameters where explored on all tasks. Most of the hyper-parameter selection was done on the Piano-midi.de task.

3.2.2 Language modelling

For the language modelling task we used a 500 sigmoidal hidden units model with no biases (Mikolov *et al.*, 2012). We use the normal distribution $\mathcal{N}(0, 0.1)$ to sample the weights. The model is trained over sequences of 200 steps, where the hidden state is carried over from one step to the next one.

We use a cut-off threshold of 45 (though we take the sum of the cost over the sequence length) in this case. For next character prediction we have a learning rate of 0.01 when using clipping with no regularization term, 0.05 when we add the regularization term and 0.001 when we do not use clipping. When

[1] Note this value changes considerably if one takes the sum over the sequence length versus the mean. We used the mean

predicting the 5th character in the future we use a learning rate of 0.05 with the regularization term and 0.1 without it.

Interestingly enough, the strategy for the regularization factor α when solving the next character prediction that performed best was to keep it constant to the value .01. For the modified task a $\frac{1}{t}$ schedule again seem to perform better, and we used an initial value for α of 0.05 with $\alpha_t = \alpha \frac{1}{1+\frac{t}{2800}}$, where t is the update index.

We have explored constant $\alpha \in \{6, 1, 0.5, 0.01, 0.001\}$ and $\frac{1}{1+\frac{max(0,t-t_0)}{\beta}}$ schedule with $\beta \in \{2800, 500\}$ and t_0 in $\{0, 28108\}$. The number of hidden units was kept constant to 500.

References

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2012). Advances in optimizing recurrent networks. Technical Report arXiv:1212.0901, U. Montreal.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, **9**(8), 1735–1780.

Jaeger, H. (2012). Long short-term memory in echo state networks: Details of a simulation study. Technical report, Jacobs University Bremen.

Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with Hessian-free optimization. In *Proc. ICML'2011*. ACM.

Mikolov, T., Sutskever, I., Deoras, A., Le, H.-S., Kombrink, S., and Cernocky, J. (2012). Subword language modeling with neural networks. preprint (http://www.fit.vutbr.cz/ imikolov/rnnlm/char.pdf).