Deep learning and Neural Networks

Assignment 3

Sjoerd Hermes
Anish Kisoentewari
Thomas Lieber

Universiteit Leiden
Dr. W.J. Kowalczyk
24-05-2020

Contents

1 Introduction

Deep neural networks (DNNs) are powerful machine learning models, however they can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensions (Sutskever et al., 2014)[4]. This poses a problem for tasks that deal with sequences whose lengths are not known a-priori. Luckily recurrent neural networks (RNNs) have the ability to use their internal state (memory) to process variable length sequence of inputs. In this report we aimed to use this ability of RNN's to solve simple addition tasks of variable length integers (for example '12+142' or '40+1'). The integers ranged from 0 to 999. The report is divided into 4 sub tasks. Every task we make use of the following RNN's: LSTM, SimpleRNN and GRU, all of which ran for 200 iterations without early stopping, thus overfitting seems likely to happen. Also in every task we measure the MAE, MSE and the 'true' accuracy: the percentage of correctly calculated sums on the complete set of 1 million (or 1048576 in the case of bit representation) addition problems. For every task, we use a normal and reversed sequence as input. The first task uses normal decimal representation (for example '321+123'), the second tasks uses bit representation of the digits. The third task uses pairs of digits (for example the sequence '123+345' (7 characters) is represented by a sequence of 3 pairs: (1,3), (2,4), (3,5). Lastly, the fourth task does the same as the third but then for binary pairs. For each task, we plotted the error measures to see which performed best for the specific task.

1.1 Expectations

We expected that reversed input leads to better performance. We base this on the study of Sutskever, Vinyals and Le (2014)[4] They found that a LSTM network was better at translating sentences when the sentences were reversed. They speculate that this could be caused by creating short term dependencies in the data set. We could not find other sources that tested this directly, but given that this speculation is true we assume that reversing the input also increases SimpleRNN and GRU accuracy. Furthermore, we expected that LSTM and GRU would have comparable performance, and the simpleRNN the worst. We base this hypothesis on the experiment of Chung, Gulcehre, Cho and Bengio (2014)[1]. They experimented with recurrent units and compared LSTM, GRU and traditional tanh units. The advanced units like GRU and LSTM outperformed the traditional units, while GRU and LSTM had comparable performance. Further, we do not expect big differences in performance for binary compared to decimal representation, as they both contain the same information. However, we do think that binary might be faster, as the input shape is smaller and therefore might require less computational effort. Lastly, we expect that the performance for paired digits increases accuracy, because this breaks down a problem like '123+456' to 3 smaller addition problems. This should make it easier for the network to learn. We expect similar results for the binary pairs.

2 Task 1: digit representation

Zaremba and Sutskever (2015) [3] found that training LSTMs with a reversed input, but the same output, had improved results when compared to keeping the input in the original format. They expect this is due to the introduction of many short term dependencies that make it easier for the LSTM to learn to make correct predictions. When looking at our own results, we can see this effect happening: for the SimpleRNN, LSTM and GRU with reversed digits input, the accuracy is higher, and the mae and mse are lower as compared to the SimpleRNN, LSTM and GRU without reversed digits input. The best performing model here is the reversed GRU and reversed LSTM, although the reversed GRU is more stable, which can be seen from the lower amount of, and smaller spikes in figure 1. The reversed SimpleRNN is slightly less accurate than both these models. Although the GRU is seems to be the best model here, it is slightly less fast at increasing accuracy than the other two models.

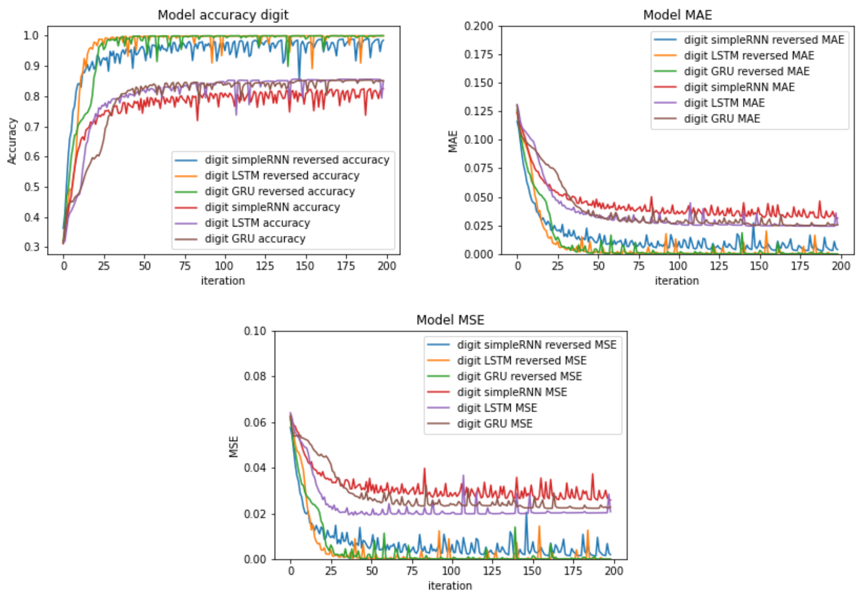

Figure 1: Accuracy, Mean Absolute Error and Mean Squared Error for the various configurations of the digit representation.

Table 1: True Accuracies, MAE and MSE on all examples

Model	**True Accuracy**	**MAE**	**MSE**
LSTM reversed	0.9995	2.0842e-04	6.4519e-05
SimpleRNN reversed	0.9850	0.0040	0.0019
GRU reversed	0.9997	1.2126e-04	3.7483e-05
LSTM plain	0.8515	0.0254	0.0207
SimpleRNN plain	0.8247	0.0314	0.0259
GRU plain	0.8492	0.0258	0.0228

Table 1 allows us to get a clear overview of how accurate the models really are. Just like in the plots of figure 1, the three RNNs with reversed inpput preform best on the 1 milion examples, with the GRU taking the crown. Finally, in terms of computing time, the GRU network takes the first place for both plain and reversed input. The time per iteration between the SimpleRNN and the LSTM networks are about the same. Both are about 3 seconds per iteration slower than the GRU. This is probably due to the GRU not having the cell state which the LSTM has. In the literature, GRUs are known to outpreform LSTMs and SimpleRNNs in terms of speed, see for example Khandelwal et al., (2017) [2]. The average time per iteration is around 24 for the GRU and 27 for the other two RNNs.

3 Task 2: bit representation

Post converting the decimal numbers to binary digits, two new intricacies are of importance. The first intricacy is that bit notation allows only for zero's and one's. Thus the network can only produce exactly the right digit, or exactly the wrong one. There is no middle ground (unlike decimal notation). This explains the greater difference in reversed and non reversed models compared to decimal notation. The second intricacy is that positioning in the string (and network) weighs more now compared to decimal notation (10 digits instead of 3 for 1 number). For the non-reversed networks, the first digit (from left to right) weighs the most. The hidden state of the non-reversed network computes the digit for this position last, thus allowing for a lower error compared to computation of the digit belonging to the first position. The plain model uses previous information on computation of the most important bit positions, while the reversed model uses the information for the least important bit position. In conjunction with the vanishing gradient problem, we can state the plain network optimizes better for the more important digit positions. This is the reason for the overall apparent better performance in terms of accuracy and error on the plain digit data compared to the reversed digit data.

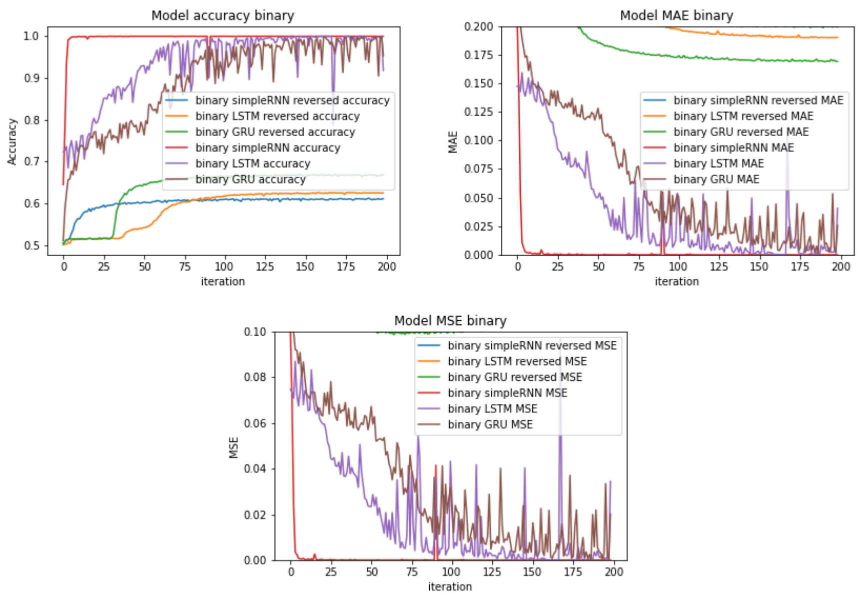

Figure 2: Accuracy, Mean Absolute Error and Mean Squared Error for the various configurations of the bit representation.

Table 2: True Accuracies, MAE and MSE on all examples

Model	**True Accuracy**	**MAE**	**MSE**
LSTM reversed	0.5170	0.2458	0.1248
SimpleRNN reversed	0.6149	0.1971	0.1117
GRU reversed	0.6437	0.1814	0.1110
LSTM plain	0.9182	0.0407	0.0344
SimpleRNN plain	1.0000	3.3829e-05	1.6331e-05
GRU plain	0.9537	0.0256	0.0201

When taking a look at figure 2 we can see this effect in action. In fact, the effect is quite strong: after 200 iterations, the SimpleRNN, LSTM and GRU networks have an accuracy of almost 1 using the plain model, whereas the the SimpleRNN and LSTM networks using the reversed order have an accuracy of about 0.6, and the GRU of about 0.65 (marginally better). The SimpleRNN stands out in the graphs, reaching the maximum accuracy in only a handful of iterations. Overfitting seems to be the expected culprit, but it could also be that the adjustments in the LSTM and GRU made to combat the short-term memory of the SimpleRNN backfire. This could perhaps be because less important digits weigh heavier in the hidden state of the network for the LSTM and GRU compared to the SimpleRNN, thus causing these more complicated networks to require more iterations to approach the same accuracy as the SimpleRNN.

Note from table 2 that the SimpleRNN has a true accuracy of 1.0000, which could imply that over all the examples, the model had no wrong predictions. This seems a bit unlikely, but we can expect such a high true accuracy from the model accuracy plot in figure 2, as the line for the plain SimpleRNN goes to 1 very quickly without any jumps or spikes (except for two small spikes if you look very closely) like the GRU and LSTM networks have. Furthermore, the model can have up to 52 wrong predictions, as the true accuracy is rounded up to 4 decimals, meaning that any value of 0.99995 or higher gets rounded up to 1. As $\frac{52}{1048576} \approx 0.00004959106$ and 1 - 0.00004959106 = 0.99995040894, the true accuracy would still be rounded up to 1. Since 52 is the largest integer value that can be divided by 1048576 without exceeding 0.00005, the maximum number of wrong predictions this model could have made is 52. Regarding computing time, we observe something akin to task 1 happening: the GRU network is the fastest per iteration, about 27 seconds, whereas the other to lag a little behind. In this case though, the SimpleRNN lags behind quite a bit, by about 18 seconds per iteration compared to the GRU and 15 seconds per iteration seconds behind the LSTM. This holds for both the reversed and the plain inputs.

4 Task 3: pairs of digits representation

In this task we used pairs of digits as representation. We used the same procedure as the first task but before compiling the model we converted the strings of addition problems to pairs (so '123+456' was converted to the string '142536'.

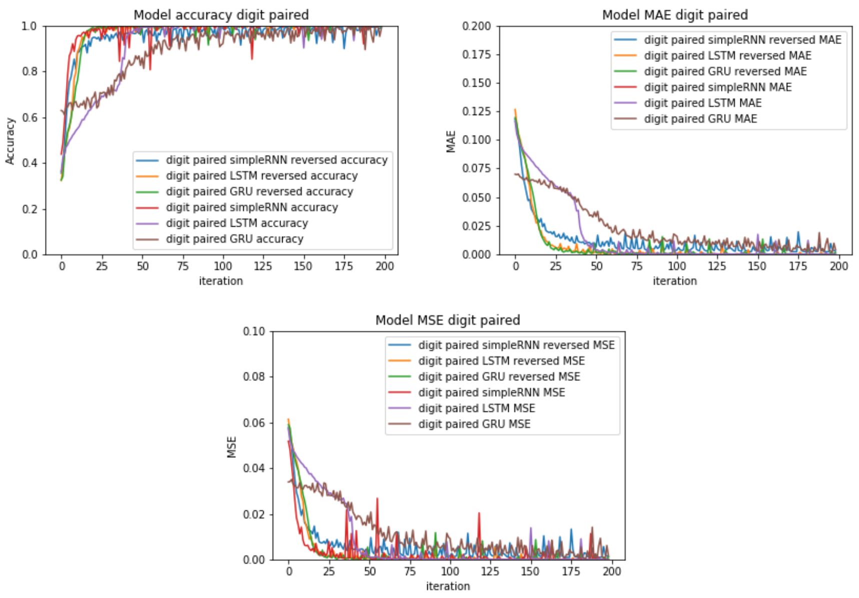

Figure 3: Accuracy, Mean Absolute Error and Mean Squared Error for the various configurations of the pairs of digits representation.

Table 3: True Accuracies, MAE and MSE on all examples

Model	**True Accuracy**	**MAE**	**MSE**
LSTM reversed	0.9998	9.7592e-05	2.4773e-05
SimpleRNN reversed	0.9911	0.0027	0.0012
GRU reversed	0.9927	0.0015	0.0010
LSTM plain	0.9998	1.2099e-04	3.3803e-05
SimpleRNN plain	0.9995	2.4056e-04	6.7868e-05
GRU plain	0.9868	0.0042	0.0017

The accuracy of the models all approached 1 after 200 iterations, see figure 3. Most models increased accuracy within 25 iterations, however non-reversed GRU and non-reversed LSTM took longer and were more gradual. This means that for these models a reversed representation of the input should be preferred. All models bounced around 0.9 − 1 accuracy around after around 1 iterations. This bouncing around is especially visible in the MSE plot. This might be caused by overfitting or a high learning rate. This bouncing phenomenon was especially visible in the non-reversed SimpleRNN, although this was also the model with the fastest convergence to as is visible in figure 3. There was no clear 'best' model, although the LSTM reversed seemed to be the fastest to converged together with the other reversed models. What also is noticeable is that both the plain and reversed models now perform similar. This was not the case for the decimal and bit notation models. Intuitively this makes sense as the difference between pairs selected from different positions in the string does not differ all that much when reversing the pairs. The difference is now smaller compared to reversing an entire string on its own. In table 3 we can see that all models have a very high true accuracy. With pairs of digits as inputs, we see that just like the accuracy of the RNNs, the computing time is very similar as well: all RNNs take around 27-28 seconds per iteration, for both reversed and plain inputs.

5 Task 4: pairs of bits representation

The final task combines the previous two sections on bit representation and pair representation. Thus we end up with pairs of binary digits to add. Similarly to the bit representation and the pair representation models, the plain models with binary pair input data preform better compared to the reversed models. This can again be explained due to the vanishing gradient problem in conjunction with increased importance of position within the string for binary data.

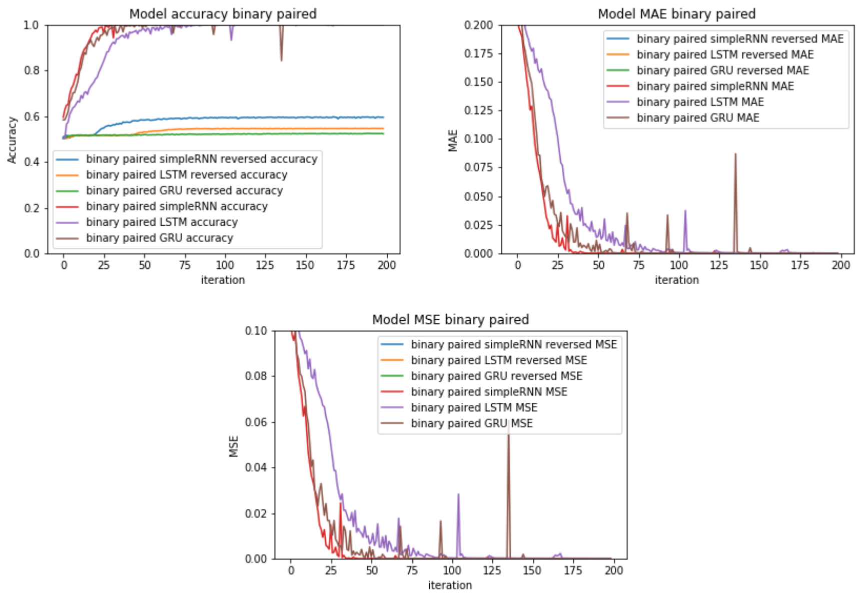

Figure 4: Accuracy, Mean Absolute Error and Mean Squared Error for the various configurations of the binary paired representation.

Table 4: True Accuracies, MAE and MSE on all examples

Model	**True Accuracy**	**MAE**	**MSE**
LSTM reversed	0.5451	0.2297	0.14345
SimpleRNN reversed	0.5941	0.2079	0.1171
GRU reversed	0.5224	0.2421	0.1347
LSTM plain	0.9997	2.0525e-04	1.3235e-04
SimpleRNN plain	1.0000	6.3132e-06	3.3528e-06
GRU plain	1.0000	1.2779e-05	5.9795e-06

However, contrary to task 3, introducing pairs to binary data does not make the reverse and non-reverse models similar in performance, see figure 4. Apparently because of the larger number of pairs and larger differences between the weights of these pairs according to position, the same phenomenon occurs as in task 2 for bit notation. This phenomenon apparently outweighs the increase in similarity pairwise addition brings between plain and reversed pairs. The introduction of pairs to bit notation does however significantly decrease the number of iterations needed for fitting for the plain models, just like in the previous section. Thus we can explicitly detect similar trends from both the bit notation and the pair representation in this combination of both. Just like we saw in task 2, the pairs of bits representation also has ”perfect” prediction, see table 4. However, we suspect that just like in task 2, there might be a few errors present. For both the reversed and plain inputs, the computing time of the GRU is very similar to that if the LSTM network, both are around 30 seconds per iteration, with a slightly bigger difference for the reversed input where the GRU is about 2-3 seconds faster per iteration than the LSTM network. However, both are a lot faster than the SimpleRNN, by about 15 seconds per iteration.

6 Conclusion

During this assignment, various representations of the input-output relation of recurrent neural networks with an addition task have been tried: decimal, bit, pairs of decimals and pairs of bits. The inputs were tried in both plain and reverse order, and for various types of neural networks: SimpleRNN, LSTM and GRU. In the end, clear differences became apparent, like expected. Using a decimal representation, reversing the input gave better results for all three types of RNN than the plain input, with no large spikes indicating instability. The SimpleRNN seemed to preform slightly worse for both the reversed and the plain input. When using bit representation we saw something very different: the reversed input made all networks preform worse. Not all networks preformed equally bad with the reversed input though, the GRU preformed a bit better than the other two. Even though the plain input made the network preform better than the reversed input, the performance is extremely unstable with this plain input, except for the SimpleRNN with reversed input, whereas the reverse input is very stable with no spikes at all. When switching to paired inputs, for the decimal representation the differences between reverse and plain inputs start to disappear after about 90 iterations, and remains accurate from then on, and all models can be considered to be stable, except for maybe the SimpleRNN with plain input. Overfitting seems to be the reason for this. The same can not be said for the paired bit representation: There are very large differences between the reversed and the plain input, which do not seem to fade after even 200 iterations. In this case, the non reverse inputs are again (like with non-paired bit representation) very accurate, where the LSTM and GRU seem to be a more unstable than the SimpleRNN. The GRU with plain input in particular seems to have a very unstable performance The SimpleRNN also preforms slightly better with the reversed input than the LSTM and GRU with reverse input. Concerning the computing time it seems that GRUs preform the best in these tasks, by a small margin when digit representation is used and by a large margin when bit representation is used. To conclude, it seems that the representation of the input matters a lot for the performance of the RNNs. Overall, it seems that a LSTM and a GRU network preform better than a SimpleRNN network, but SimpleRNN networks seem to produce the most stable results. Therefore, there is no clear winner and the "best" type of RNN depends on the wishes of the researcher, as well as the representation of the input.

References

[1] Gulcehre C. Cho K. Bengio Y. Chung, J. Empirical evaluation of gated recurrent neural networks on sequence modeling. 2014.

[2] Lecouteux B. Besacier L. Khandelwal, S. Comparing gru and lstm for automatic speech recognition. 2017.

[3] Zaremba W. Sutskever, I. Learning to execute. 2015.

[4] Vinyals O. Le Q. V. Sutskever, I. Sequence to sequence learning with neural networks. *Advances in neural information processing systems.*, pages 3104–3112, 2014.