# DLNN Assignment 3 Report

Vince Hasse, Amin Moradi, Orson Peters, and Martijn Swenne

Leiden University

## Introduction

Neural networks have shown great potential for static data, especially in supervised methods. We have seen how Deep Neural Networks can extract information from high-dimensional datasets. Although previous methods can only work if sample points are independent of each other and the target value can only be inferred from one single input data point. This means end-to-end NNs have limitation in times-series and data streams i.e NLP and Signal Processing. Recurrent Neural Networks are designed to extract knowledge from sequential data where the output value is dependent on not just the previous value, but a weighted combination of the whole series. Throughout the years, there has been powerful implementations of RNN like LSTM[2] and GRU[1] which we will explore more. In this assignment, we will implement and adapt three Network (LSTM, GRU, RNN) on the problem of Adding Integers and explain their behaviour both in the training process and on the accuracy on the dataset.

## 1 Adding Integers with Recurrent Networks

§ For the task of solving addition problems the numbers first have to be parsed to be presentable to the RNN. Two randomly chosen numbers are generated and parsed to a string with a + sign in between. Then the string is padded with spaces to the maximum number of chars, which is 2 times the number of digits plus 1 to account for the + sign. The result of the addition is also stored as a string, and padded with spaces to the maximum number of chars, which is the number of digits plus 1 to account for the carry. To test the learning capabilities, the addition string can also be reversed, while the answer should be the same.

For each section below, the experiments are executed with an LSTM network, a SimpleRNN network and a GRU network as model. The model will be trained for 200 iterations using 50.000 training samples, of which 5% is used for validation, using a batch size of 128. Once the model is trained, we will be testing the RNN based on the 1 million possible addition problems of two numbers with a maximum of 3 digits. The errors are measured using the Mean Squared Error (MSE) and the Mean Absolute Error (MAE). Also will each section below test the addition in normal and reversed order of digits.

### 1.1 Normal integers

If we look at Figure 1, we can see that the SimpleRNN got a high accuracy rate very quickly, whereas the LSTM took a decent amount of epochs before reaching a high accuracy. For GRU we reach a high accurcay very fast for the reversed integers, but not for the normal integers. The same thing can be seen in Figure 2.

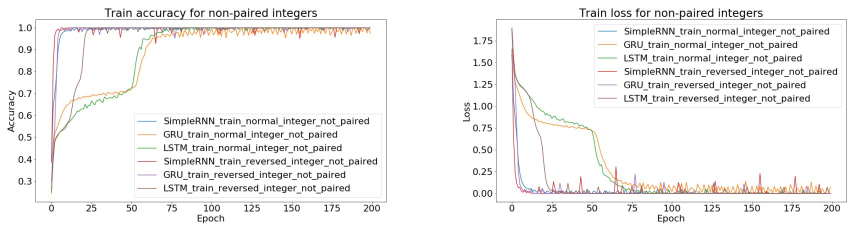

Fig. 1: Training accuracy and loss over all models trained on normal and reversed non-paired integers.

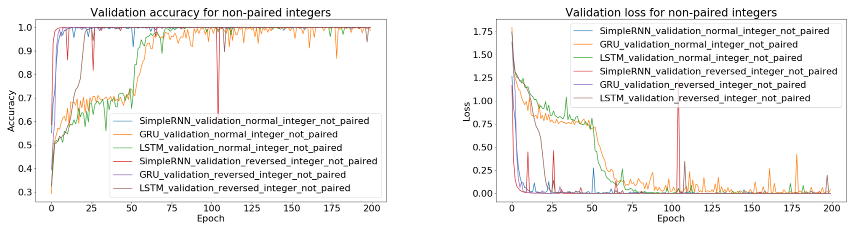

Fig. 2: Validation accuracy and loss over all models trained on normal and reversed non-paired integers.

### 1.2 Binary numbers

The results of the training on binary numbers can be seen in Figure 3 and the validation results can be seen in Figure 4. Again we see that the simpleRNN very easily obtains a high accuracy with only a few epochs, while GRU and LSTM are struggling more. After around a 100 iterations they start to converge to a high accuracy. This definitely shows binary addition is much harder to learn for LSTM and GRU.

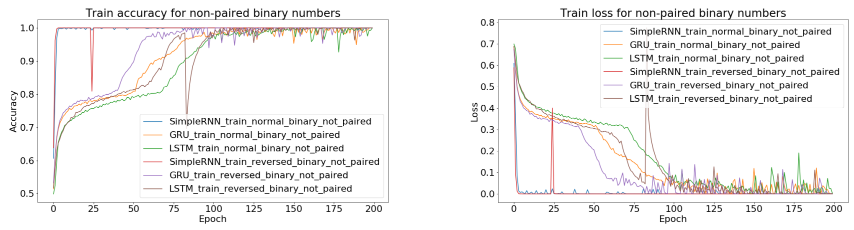

Fig. 3: Training accuracy and loss over all models trained on normal and reversed non-paired integers.

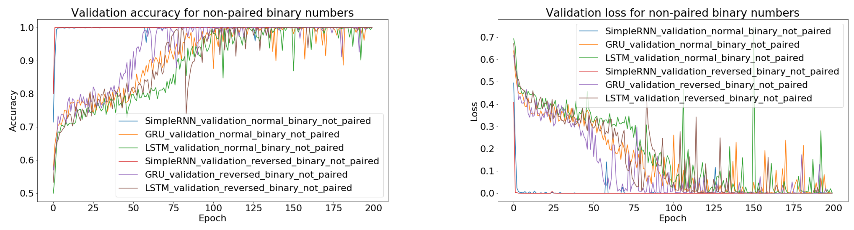

Fig. 4: Validation accuracy and loss over all models trained on normal and reversed non-paired binary numbers.

## 1.3 Paired digits

The training results on paired integers digits can be seen in Figure 5. We can see that simpleRNN again achieves a high accuracy very quickly, but now GRU and LSTM also achieve this very quickly on the reversed paired integers. This is however not the case for the normal paired integers. Again show the validation results in Figure 6 the same concluded above.

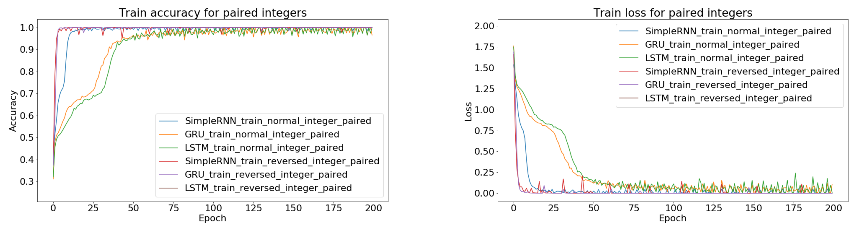

Fig. 5: Training accuracy and loss over all models trained on normal and reversed paired binary numbers.

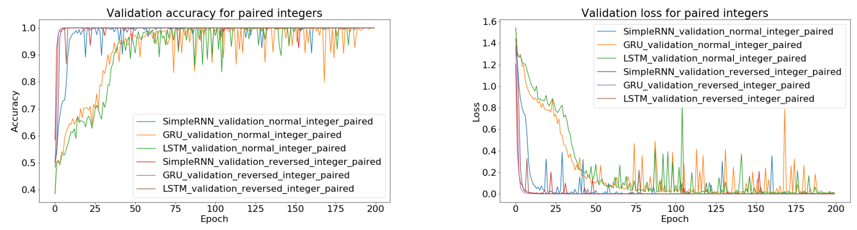

Fig. 6: Validation accuracy and loss over all models trained on normal and reversed paired integers.

## 1.4 Binary paired digits

The last results, on paired binary numbers, can be seen in Figure 7 and Figure 8. Here we see that all RNNs achieve a high accuracy very quickly. But the simpleRNN on normal paired binary numbers sometimes spikes in accuracy loss.

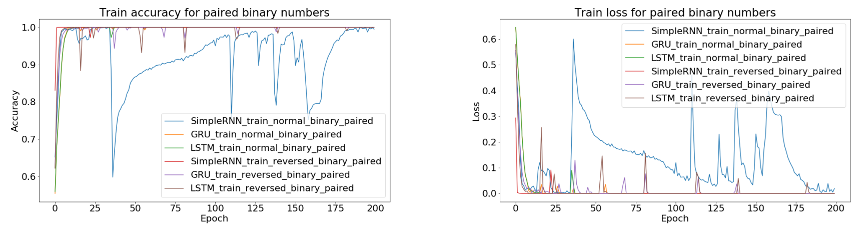

Fig. 7: Training accuracy and loss over all models trained on normal and reversed paired binary numbers.

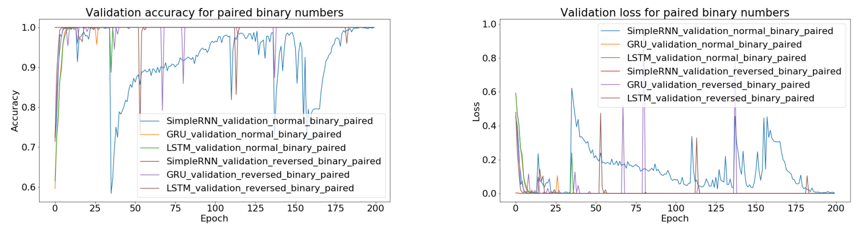

Fig. 8: Validation accuracy and loss over all models trained on normal and reversed paired binary numbers.

## 2 Conclusion

All results on the total 1 million data samples can be found in Table 1. We can see that all settings achieve a 99% accuracy, with exception of GRU on normal non-paired integers. The training data in the subsections of Section 1 show that simpleRNN is probably the best RNN for this specific problem, as it achieves a high accuracy very fast on all of our problems. But as seen in the results of the 1 million data samples is every RNN suitable for solving these types of problems, where each RNN achieves a $>$ 99% accuracy, except for GRU. We think this might be caused by a bad randomness seed, and by training it again it would succeed in achieving a $>$ 99% accuracy. The spikes seen in the plots might be caused by the RNN mispredicting one digit, but if that is the leftmost digit, it will have a huge impact on the accuracy. Throughout the training process we observed consistency in accuracy and differences in convergence rates between the different RNNs.

| | | | | **Training error** | | | **Full dataset error** | | |
|---|---|---|---|---|---|---|---|---|---|
| **Network** | **Reversed** | **Binary** | **Paired** | *Accuracy* | *MSE* | *MAE* | *Accuracy* | *MSE* | *MAE* |
| *LSTM* | *False* | *False* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 12.197 | 0.014 |
| | | | *True* | 0.999 | 0.07 | 0.07 | 0.998 | 0.497 | 0.019 |
| | | *True* | *False* | 1.000 | 0.0342 | 0.0021 | 0.999 | 0.295 | 0.008 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| | *True* | *False* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 1.923 | 0.003 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.000 | 0.000 | 0.000 |
| | | *True* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 0.005 | 0.000 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.000 | 0.000 | 0.000 |
| *GRU* | *False* | *False* | *False* | 0.957 | 637.513 | 1.375 | 0.951 | 802.848 | 1.622 |
| | | | *True* | 0.994 | 7.168 | 0.123 | 0.992 | 6.892 | 0.139 |
| | | *True* | *False* | 1.000 | 0.015 | 0.001 | 1.000 | 0.108 | 0.005 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| | *True* | *False* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 0.002 | 0.000 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.000 | 0.000 | 0.000 |
| | | *True* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 0.003 | 0.000 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 |
| *SimpleRNN* | *False* | *False* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 0.030 | 0.000 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.000 | 7.659 | 0.014 |
| | | *True* | *False* | 0.996 | 396.518 | 0.880 | 0.995 | 395.0.19 | 0.924 |
| | | | *True* | 0.995 | 77.325 | 0.569 | 0.994 | 135.897 | 0.662 |
| | *True* | *False* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 0.001 | 0.000 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.000 | 0.002 | 0.000 |
| | | *True* | *False* | 1.0 | 0.0 | 0.0 | 1.000 | 0.000 | 0.000 |
| | | | *True* | 1.0 | 0.0 | 0.0 | 1.000 | 0.000 | 0.000 |

Table 1: Accuracy, MSE and MAE results on the Training set and the full dataset for every possible setting.

## References

1. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
2. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)