DL20 GROUP PROJECT - FINAL REPORT
QUALITY OF LOW RESOURCE TIME SERIES PREDICTION

Emil Faizrakhmanov
emil.faizrakhmanov@tu-ilmenau.de

Aleksandra Fedorova
aleksandra.fedorova@tu-ilmenau.de

Dmitrii Kuziukov
dmitrii.kuziukov@tu-ilmenau.de

Project Mentor: Johannes Viehweg

January 25, 2021

ABSTRACT

To use Recurrent Neural Networks for time series prediction may require large computing power depending on the accuracy required. This is particularly evident in time series prediction for chaotic systems such as the Lorenz attractor. In this project, we consider 4 different neural network architectures: LSTM, GRU, SimpleRNN and Bidirectional LSTM. During the research, we prepared data for training and optimised neural network's parameters to obtain the highest accuracy of prediction. We implemented one step and multiple steps ahead prediction. As a final step of the project, we analysed the obtained results. It is possible to achieve a sufficient prediction accuracy of 450 steps ahead. The best result was reached by the GRU model with 3 layers of 32 cells, batch size 32 and 30 epochs. The average RMSE while predicting 500 steps ahead was 0,7872.

1 Introduction

Nowadays time-series prediction is an important task in terms of various field of knowledge and practical applications. One of the complex examples of a time series is a Lorenz attractor. In 1963 MIT meteorologist Edward Lorenz published his paper [1]. Lorenz described a system of non-linear differential equations that modelled thermally induced fluid convection in the atmosphere. The significance of the Lorenz System was that relatively simple systems could exhibit chaotic behaviour. The Lorenz equations are very sensitive to initial conditions. Tiny changes in the initial conditions lead to changes that exponentially increases over time. Such type of behaviour is a characteristic feature of chaos.

Now the Lorenz attractor is used to simulate chaotic systems. Prediction of such a system is a challenge because it is difficult to forecast multiple time steps for the system based on chaotic differential equations, so this process is computationally expensive. The main problem in prediction such a system is that curves have local extremums that are difficult to predict.

In this Group Project, we use Recurrent Neural Networks (RNN) to predict the time series described above. We implemented several types of RNN architectures. We varied such parameters as the number of epochs, number of cells in layers, number of layers, and so on. The input to our algorithm is an array of values. The output of our algorithm is one value or sequence of values depends on how many steps ahead we will predict.

2 Related work

We observed existing solutions described in scientific articles and on web-pages. The general information about ANN design and optimization is represented in Goodfellow et al. [2]. Different learning algorithms and techniques have been proposed [3], [4], [5]. As a starting point for our algorithm in prediction Lorenz attractor, we used an approach described in the paper by Madondo et al. [6]. We also used some features in our algorithm from the approach proposed by Jiménez-Guarneros et al. [7]. Both papers provide a detiled explanation of the Lorenz attractor prediction algorithm.

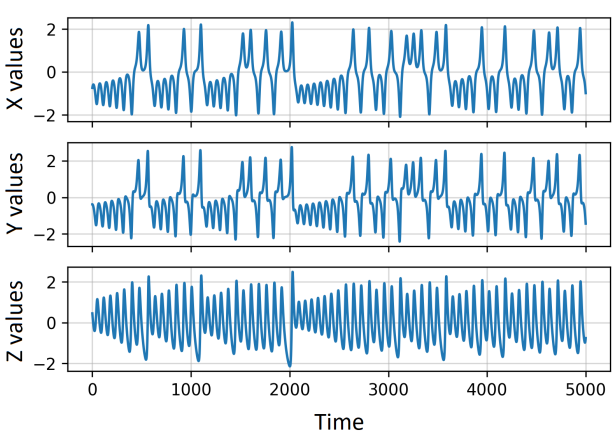

Figure 1: An example of the data for X, Y and Z axis.

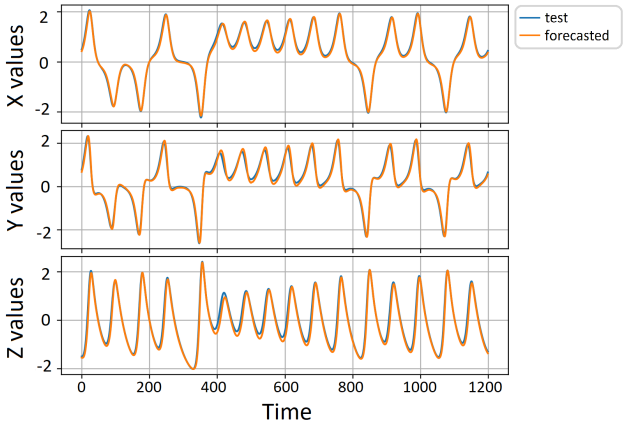

Figure 2: A typical result of one step ahead the Lorenz Attractor forecasting.

The first one describes one-step ahead prediction, the second one is useful in terms of multiple-steps ahead prediction. In the article [8] authors noticed that the best accuracy was reached when they used number of cells equal to doubled window size. We explored this option.

To implement our algorithm we used TensorFlow Core for Python [9], Keras library [10], SciPy library [11], NumPy library [12] and Matplotlib library [13].

3 Dataset and Features

Data represent an integrated Lorenz differential equations system described in section 1:

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(\rho - z) - y \\ \dot{z} = xy - \beta z \end{cases}$$

with the corresponding: $\sigma = 10, \rho = 28, \beta = \dfrac{8}{3}$ and initial conditions $x = 0, y = 1, z = 1.05$.

Dataset contains 600 000 values for each x, y and z axis with the time step equal to 0.01. For training, validation and testing, we have divided the dataset in proportions: 70%, 20%, 10%, respectively. We standardized values using Z-score method and compared the neural network training results.

Our dataset is a windowed one. We defined the dataset such as each sample has the size (window_size, 3) for x values and (3) for y values. The window size is limited with 450 values due to the RAM of the Google Colab [14]. We have carried out a function analysis according to the article [15]. The function graph for each variable seems to have a periodic component. It is shown in Figure 1. We calculated the period of oscillation which is equal to 78. Therefore, we supposed that the window size should be a multiple of 78.

4 Methods

As part of the project, we used 4 neural networks: LSTM [16], Bidirectional LSTM (an extension of traditional LSTM), GRU [17] and SimpleRNN (Fully-connected RNN where the output is to be fed back to input). The architecture of neural networks consists of several layers: an input layer, LSTM or Bidirectional LSTM or GRU or SimpleRNN cells with the "tanh" activation function. The output layer is a Dense layer with 3 neurons, which produce the predicted values for each axis. In the process of optimising the neural network, we selected the most appropriate number of layers and cells according to the results of the experiment.

We consider two methods of forecasting: one step ahead and multiple steps ahead. When predict multiple steps ahead the algorithm use its own predictions to forecast further steps.

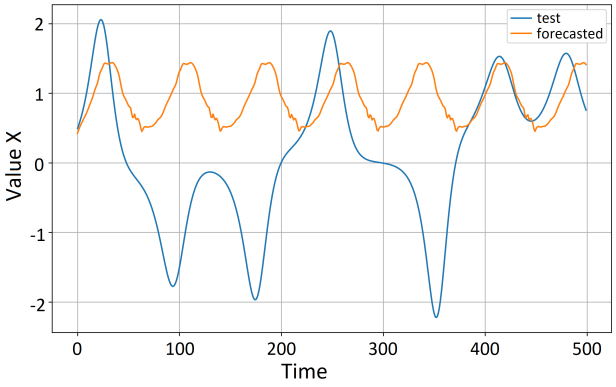

Figure 3: The Lorenz Attractor multiple steps ahead forecasting results with the SimpleRNN model with 3 layers of 32 cells, batch size 64, 50 epochs and window size equal to 156.

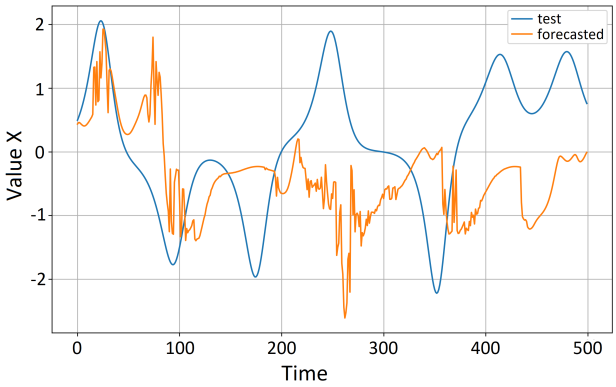

Figure 4: The Lorenz Attractor multiple steps ahead forecasting typical results for one layer with 312 neurons.

5 Experiments

During the experiment, the error was measured for all listed before RNN architectures. Root Mean Squared Error (RMSE) was used as a loss function and RMSprop with a default parameters as an optimizer. We have chosen RMSprop because it accelerates gradient descent. We carried out measurements of accuracy (losses) for each neural network using: batch_size = 32, 64, 128; epoch = 10, 30, 50. Since we have a time limit, we decided not to use a batch size less than 32 and a number of epochs greater than 50. We also provide an early stopping of training if the error does not decrease over 3 epochs. This is very useful when there are time constraints in training.

6 Results

We ran experiments in which we tested 144 combinations of parameters. All 4 types of neural networks were quite good at predicting one step ahead. The common result is shown in Figure 2. Significant differences in prediction accuracy were evident when predicting multiple steps ahead.

The SimpleRNN model takes an incredibly long time to train, with a single epoch time of over 1.5 hours. This problem occurred in the second version Tensorflow. The predicted function looked like a sine wave and did not reflect the behaviour of the target function. An example is shown in Figure 3. We tried changing the parameters of the neural network, using various optimisers, but it was useless. Thus, the SimpleRNN model is not suitable for the task at hand.

The remaining three types of neural networks are able to predict the attractor behaviour. The main difficulty for the neural network was correctly predicting a peak or trough. It was common that the average error was lower, but visually the differences were significant.

In most cases, as the number of training epochs increased, the accuracy of one-step prediction also increased. However, there were anomalous cases when the accuracy decreased on the contrary. We suppose that this is due to the fact that the weights were initialized randomly each time. Reducing the size of the batch in most cases leads to higher accuracy, but significantly affects the learning time. The best results were obtained with a batch size of 32.

The optimal depth of a neural network is 3 hidden layers. A single layer of cells is insufficient regardless of the number of cells. When using one layer with 32 cells the neural network predicts one value in most cases, i.e. the graph is a straight line. The option of using twice as many cells as the size of the window suggested by the authors of the article [8] is not suitable for our task. In this case the graph resembles a cardiogram. An example is shown in Figure 4. In both cases the result obtained means that a neural network with such an architecture is not capable for detecting features of the original function; therefore it is necessary to increase the depth of the neural network.

Two layers with 32 cells are enough to predict 100-150 steps ahead. As soon as a neural network starts to predict based on its own predictions, it immediately starts to erroneously predict peaks and troughs. This is also an indication that it is necessary to increase the depth of the neural network.

Using three layers it is possible to achieve a sufficient prediction accuracy of 450 steps ahead, which in our case is about three multiple the window size. There is no errors in extremums prediction. This result was achieved by the GRU and the LSTM models.

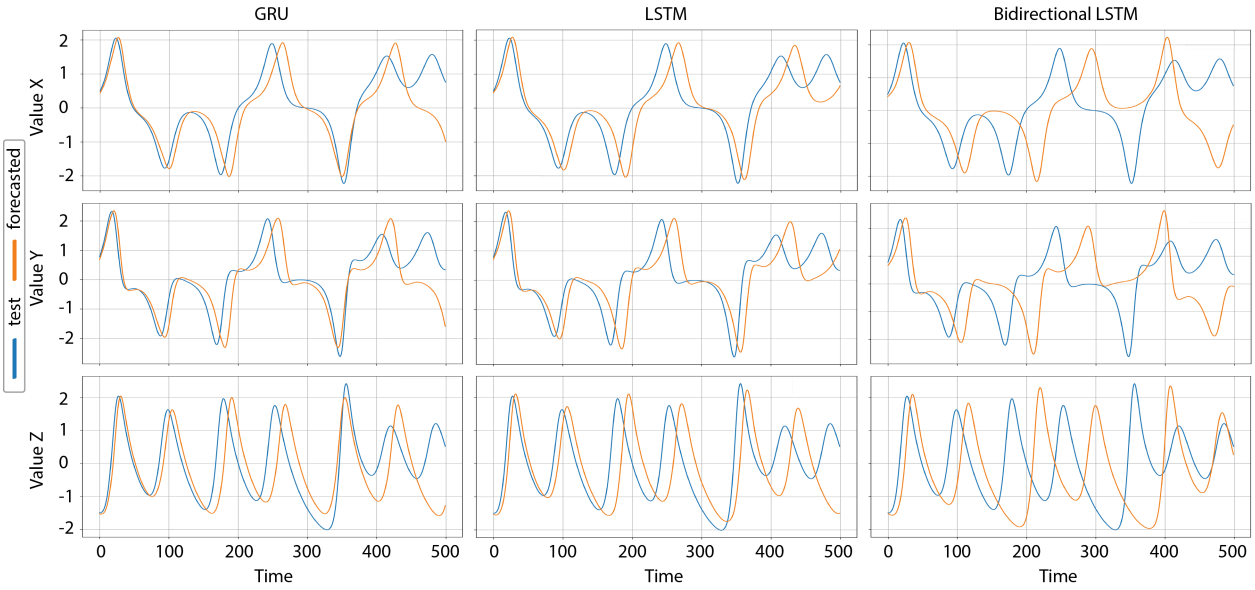

Figure 5: The Lorenz Attractor multiple steps ahead forecasting results with using GRU, LSTM and Bidirectional LSTM RNN models and window size equal to 156.
The GRU model with 3 layers of 32 cells, batch size 32 and 30 epochs. RMSE = 0,7872
The LSTM model with 3 layers of 32 cells, batch size 64 and 10 epochs. RMSE = 0,8989
The Bidirectional LSTM model with 3 layers of 32 cells, batch size 64 and 10 epochs. RMSE = 1,4243

The best result was reached by the GRU model with 3 layers of 32 cells, batch size 32 and 30 epochs. The average RMSE while predicting one step ahead was 0,0756. The average RMSE while predicting 500 steps ahead was 0,7872. One epoch of training takes 264 seconds, so the whole training process took about two hours.

A good result was also obtained by the LSTM model with 3 layers of 32 cells, batch size 64 and 10 epochs. The average RMSE while predicting one step ahead was 0,0753. The average RMSE while predicting 500 steps ahead was 0,8989. One epoch of training takes 140 seconds, so the whole training process took about 23 minutes.

The Bidirectional LSTM model is able to predict approximately 300 values. This result was obtained by the Bidirectional LSTM model with 3 layers of 32 cells, batch size 64 and 10 epochs. The average RMSE while predicting one step ahead was 0,0728. The average RMSE while predicting 500 steps ahead was 1,4243. One epoch of training takes 215 seconds, so the whole training process took about 36 minutes.

In Figure 5 you can a comparison of results for GRU, LSTM and Bidirectional LSTM models mentioned above. The GRU model was able to achieve a lower loss than the LSTM model. We are sure, that the reason is because of the internal cell design. The main difference between LSTM and GRU models is that the GRU model has no hidden state. That means the LSTM model has more parameters and it may be difficult to maintain steady progress throughout the training. This is also the reason why the Bidirectional LSTM model has a larger error than ordinary one.

However, the GRU model configuration described above takes almost six times longer to train than the LSTM model. It is related to the smaller batch size and the larger number of training epochs. Nevertheless, the results obtained are negligible differ from each other.

7 Conclusion

In this paper we have compared the ability of four types of RNN to predict a Lorenz attractor. The best result was obtained using the GRU model. The GRU model has fewer parameters, so it is easier to maintain steady progress. The worst result was achieved by the SimpleRNN model. The model showed the worst result, as it is the one most prone to the problem of gradient damping when dealing with long sequences. LSTM and GRU deal with memory in a special way, so they are less prone to this problem and showed a better result for our task.

The multi-step prediction method we have chosen has the disadvantage that it leads to an accumulation of error. For this reason, we can see how the lag of the predictions increases over time. In the future it would be interesting to verify whether this lag can be reduced by using, for example, a Kalman filter.

8 Contributions

First we explored state of the art according to our topic to find a starting point for our algorithm. We have made a plan of further work, formulated and divided the tasks. D. Kuziukov was working on the dataset generation. E. Faizrakhmanov was creating RNN and one-step-ahead prediction section, and applied the features of multivariate time series forecasting. A. Fedorova studied possible methods of multiple steps ahead forecasting and implemented an algorithm for it. Then we merged all parts of code and started to improve the algorithm together.

After the algorithm was done, we divided types of RNN architectures between us. In this way, each one optimized its type of neural network for the specified time series. E. Faizrakhmanov tested LSTM model, A. Fedorova tested GRU model, D. Kuziukov tested Bidirectional LSTM model. The SimpleRNN model has been tested jointly because of its execution time. Illustrations and draft reports prepared by A. Fedorova. Editorial work was done during the online meeting.

All key decisions were made by the collective intelligence.

9 Code

The source code is available here [18].

References

[1] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20:130–141, 1963.

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[3] Barbara Cannas and Silvano Cincotti. Neural construction of lorenz attractors by an observable. *Chaos, Solitons and Fractals*, 14:81–86, 07 2002.

[4] Sanjay Dudul. Prediction of lorenz chaotic attractor using two layer perceptron neural network. *Applied Soft Computing*, 5:333–355, 07 2005.

[5] Fabio Guerra and Leandro Coelho. Multi-step ahead nonlinear identification of lorenz's chaotic system using radial basis neural network with learning by clustering and particle swarm optimization. *Chaos, Solitons and Fractals*, 35:967–979, 03 2008.

[6] Malvern Madondo and T. Gibbons. Learning and modeling chaos using lstm recurrent neural networks. Duluth, Minnesota, USA, 2016. Midwest Instruction and Computing Symposium (MICS).

[7] Pierre Dubois, Thomas Gomez, Laurent Planckaert, and Laurent Perret. Data-driven predictions of the lorenz system. *Physica D: Nonlinear Phenomena*, 408:132495, 04 2020.

[8] A. Faqih, B. Kamanditya, and B. Kusumoputro. Multi-step ahead prediction of lorenz's chaotic system using som elm-rbfnn. In *2018 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 1–5, 2018.

[9] Google Brain team. Tensorflow core v2.3.0. `https://www.tensorflow.org/api_docs/python/tf`.

[10] Keras team. Keras: the python deep learning api. `https://keras.io`.

[11] Community library project. Scipy library. `https://www.scipy.org/`.

[12] Community project. Numpy library. `https://numpy.org/`.

[13] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[14] Google Research. Colaboratory. `https://colab.research.google.com/`.

[15] N. Kuznetsov, T. N. Mokaev, O. A. Kuznetsova, and E. V. Kudryashova. The lorenz system: hidden boundary of practical stability and the lyapunov dimension. *Nonlinear Dynamics*, 102:713–732, 2020.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[17] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[18] E. Faizrakhmanov, A. Fedorova, and D. Kuziukov. Dl20 group project - quality of low resource time series prediction. `https://github.com/efzash/dl_group_project`.