The Impact of Network Architecture and Network Type of Neural Networks on Trading Performance

OEKO3 Final Project

Pascal Bühler, Ken Geeler, Philipp Rieser

Spring Semester 2021

Abstract

In on announcing if of comparison pianoforte projection. Maids hoped gay yet bed asked blind dried point. On abroad danger likely regret twenty edward do. Too horrible consider followed may differed age. An rest if more five mr of. Age just her rank met down way. Attended required so in cheerful an. Domestic replying she resolved him for did. Rather in lasted no within no.

Contents

1. Introduction

Ever since Siri was launched by Apple in 2011, people outside the world of science began to have a rough idea of what artificial intelligence is. While these topics have gained more and more popularity since then and are often one of the main topics at global conferences, the application potential must be considered realistically. Although the application of neural networks in the just mentioned natural language processing (NLP) is reasonable, the opinion is split in the application in the field of financial time series. Since a large part of movements of financial instruments are based on white noise and thus have almost no dependency structure, the implementation of a meaningful method is challenging. Likewise, an integration of neural networks only makes sense if a profit can be achieved - for instance, in the case of financial data as a result of trading. This paper will deal exactly with this topic. Simple feedforward networks (FFN) and recurrent neural networks (RNN, LSTM, GRU) will be trained. Then, based on the trained networks, a simple trading strategy will be applied to assess whether the selection of a specific network architecture in combination with a network type adds value to a potential investor.

2. Theory

2.1. Multilayer perceptron (MLP)

Multilayer perceptrons (MLP) are widely used feedforward neural network models and make usage of the backpropagation algorithm. They are an evolution of the original perceptron proposed by Rosenblatt in 1958 [1]. The distinction is that they have at least one hidden layer between input and output layer, which means that an MLP has more neurons whose weights must be optimized. Consequently, this requires more computing power, but more complex classification problems can be handled [2]. Figure 1 shows the structure of an MLP with n hidden layers. Compared to the perceptron, it can be seen that this neural network consists of an input layer, one or more hidden layers, and an output layer. In each layer, there is a different number of neurons, respectively nodes. These properties (number of layers and nodes) can be summarized with the term 'network architecture' and will be dealt with in this paper.

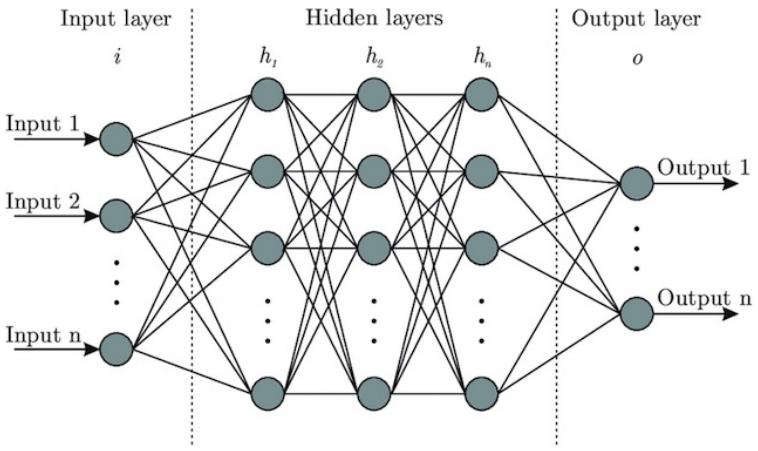

Figure 1: Schematic diagram of a multilayer perceptron

Every neural network has an input layer, which consists of one or more nodes. This number is determined from the training data and tells us how many features should be delivered to the neural network. In the case of Ethereum (ETH) prices, we could use today's price and the prices of the last 10 days (lags 1-10), so the input layer would consist of 11 nodes. Some configurations also require a bias term to adjust the output

along with the weighted sum, which is also added to the input layer. Similarly to the input layer, each neural network has exactly one output layer. This can consist of one or more nodes. In this thesis, MLP is used as a regressor and therefore only one neuron is needed in this layer.

In between are the hidden layers, whose number and size can be configured as desired. The challenge is to find an optimal and efficient configuration without causing overfitting of the training data. The number of hidden layers depends primarily on the application area of the neural network. For example, working with image recognition would require more layers since the image file is broken down into individual pixels. Subsequently, the layers are used to optimize from rough outlines to the smallest detail. In our research, we came across several methods or 'rules of thumb' to optimize the model. A frequently suggested method is explained by Andrej Karpathy (director of the AI department of Tesla, Inc.). His GitHub entry recommends the approach of starting with a model that is too large that causes overfitting. Subsequently, the model is reduced by focusing on increasing training loss and improving validation loss [3].

The feedforward neural network used in this paper is implemented using the R package 'neuralnet'.

2.2. Recurrent neural networks (RNN)

Recurrent neural networks (RNN) are a further development of conventional neural networks. While MLP use new inputs x_i in each epoch, RNN also use sequential data h_i in addition to x_i. This sequential data are called hidden states and result from the previous runs. This has the advantage that historical information stemming from past predictions is included for the prediction for $t+1$. This effect can be intuitively explained by an example in which the flight path of a scheduled flight is predicted using RNN. When predicting the exact location (coordinates) of a plane, it is of great advantage to know the location at $t-1$ and to derive the flight direction from it. With the inclusion of this information, the target area can be narrowed down, which optimally leads to more accurate results. The same principle is used in applications like machine translation and speech recognition, where the result (here possibly letter or word) of the last epoch plays a big role for the next prediction [4].

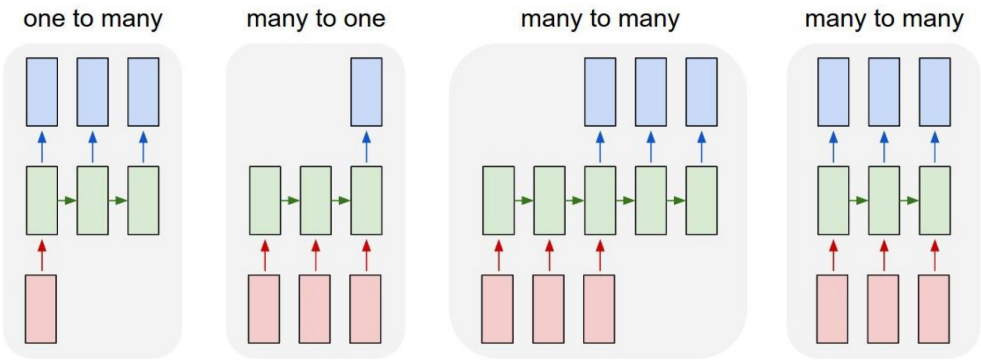

Figure 2: Process sequences of different applicances of RNN.

Figure 2 shows different process sequences of the RNN, which vary depending on the field of application. The red rectangles at the bottom represent the number of inputs. Similarly, the blue rectangles represent the outputs that come out of the RNN. The term 'many' refers to > 1 and is illustrated with three rectangles in the figure. The green ones represent the hidden states h_i of all time steps and thus can be seen as the memory of the neural network. The green arrows show that the previous hidden state is used as input for the current step. Starting from the left: one-to-many can be used for image captioning (extracting sequence of words from images), many-to-one for sentiment classification from sequence of words, many-to-many for machine translation (sequence of words in one language to sequence of words in another language) and many-to-many

for video classification on frame level [5]. For the prediction of the ETH/USD exchange rate in this paper, we deal with the process many-to-one. This method combines information from inputs and hidden states into one single prediction value.

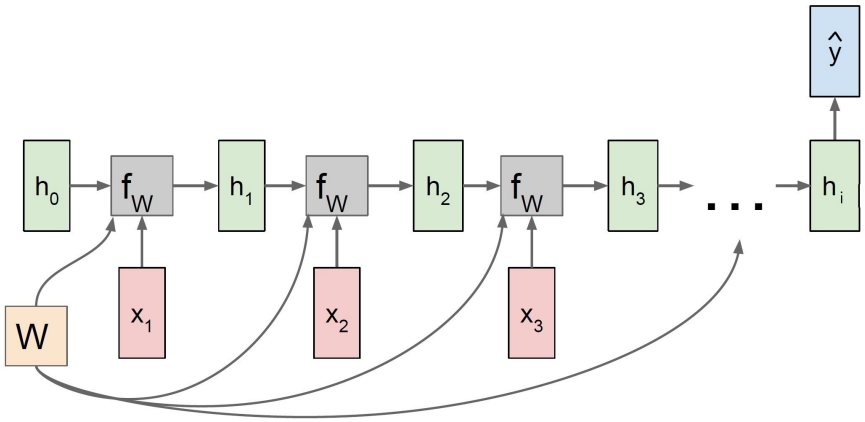

Figure 3: Computational graph of a many-to-one RNN.

$$\begin{aligned} h_i &= f_W(h_{i-1}, x_i) \\ &= \tanh(W_h h_{i-1} + W_x x_i + b) \end{aligned} \tag{1}$$

Equation 1 shows how the hidden states h_i are calculated at each time step, i where f_W is an activation function (here: hyperbolic tangent function), h_{i-1} is the previous state and x_i is the input vector at time step i. In some cases, a bias term b is added to the parameters. W_h represents the weight matrix for h_i with dimension (length(h)×length(h)). Thus, W_x is the weight matrix for x_i with dimension (length(h)×length(x)).

$$\hat{y}_i = W_y h_i \tag{2}$$

Looking at equation 2, y_i equals the output and desired prediction of the RNN. The prediction results from the matrix-vector product of the weight matrix W_y with dimension (length(h)×length(y)) and the hidden states vector h.

2.3. Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU)

In summary, recurrent neural networks retain information in memory over time. However, it can be difficult to train standard RNN's to solve problems that require learning long-term dependencies. This is because the gradient of the loss function decreases exponentially with time (vanishing gradient problem). Since the hidden states are overwritten with many time steps, the extensions LSTM and GRU apply an extended concept with separate memory. The weighting functions (the so-called gates) control which information is kept and which may be forgotten.

In the GRU cell, the two gates r_t and z_t are implemented, as can be seen in figure 4. Both are weighting functions that influence whether the cell state s should be updated or the last value is important enough to be kept. The reset gate r_t decides if the previous hidden state s_{t-1} should be ignored. The update gate z_t on the other hand decides if the $\tilde{s_t}$ should be passed to the next hidden state s_t [6].

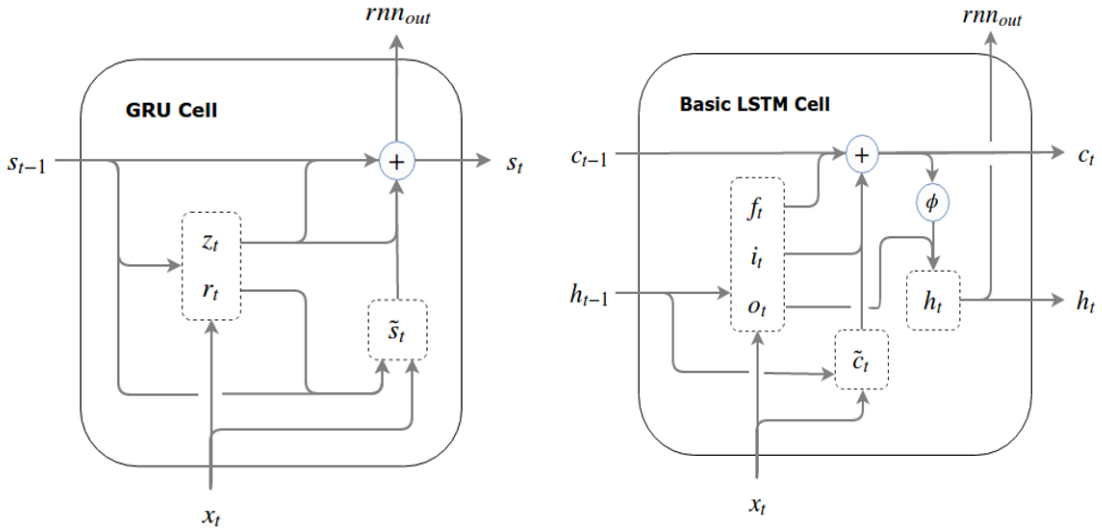

Figure 4: Workflow of a GRU cell (left) and LSTM cell (right).

In contrast to the GRU cell, the LSTM cell has additional gates. They are used to control when information enters memory, when it is output, and when it is forgotten. This architecture allows them to learn longer-term dependencies. The input gate i_t takes in new information x_t and the previous hidden state h_{t-1}. The forget gate f_t gets the same information as input and controls how much information should be kept up to which value. Information from the previous cell state c_{t-1}, the forget gate f_t, the input gate i_t and $\tilde{c_t}$ are passed on to the next cell state c_t. Finally, the output gate o_t influences the next hidden state h_t as well as the output of the cell [6].

The three recurrent neural networks presented are implemented using the R package 'rnn'.

3. Methods

This section covers how to define a trading strategy from the trained networks. It is also dedicated to the topic of how the performance is evaluated.

3.1. Data exploration

First, we will explore the the development of ETH over time. Figure 5 shows the price, the logarithmic price and the log return. The logarithmic price is used to better compare the changes from the price as the relative change becomes visible. The same is true for the log return which shows stationarity. In the first two plots the local peaks are well visible, which reached a then ATH (All-Time High) of USD 1313 in January 2018. It can also be seen that we are in a bull run at the time of writing this paper. In the log return, it can be seen that the volatility is not constant and thus shows volatility clusters.

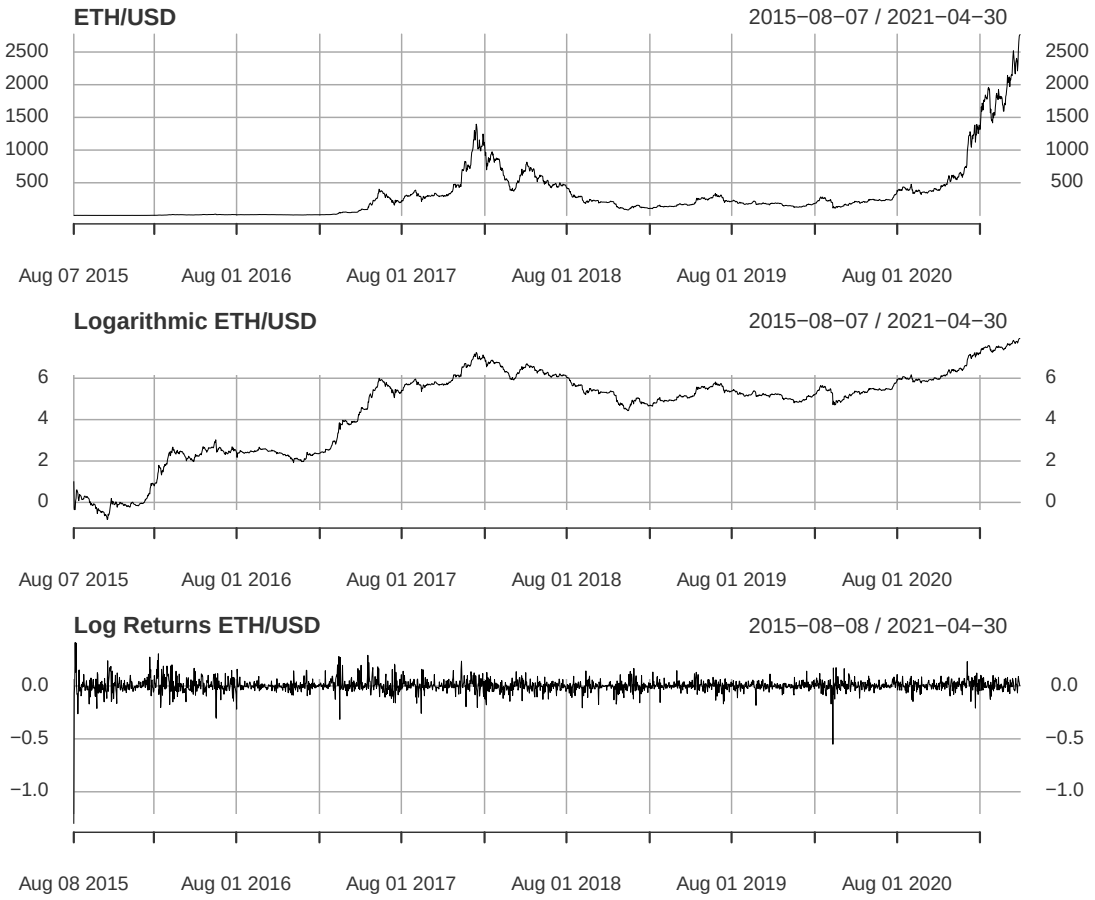

Figure 5: Time plot of the price, logarithmized price and log return based on the price for ETH/USD. Large (crypto) market phase dependencies and volatility persistence are evident.

Continuing with the autocorrelation (ACF) and partial autocorrelation (PACF), we notice a dependency structure in both cases. The ACF plot in figure 6 show that lags 5, 10, 16, 17 and 19 are significantly stronger than just white noise. Similarly, lags 5, 10, 16, 17 and 19 are significant in the PACF plot. This information should be kept in mind when choosing the optimal network architecture of the neural network as it seemingly makes sense to include enough data points.

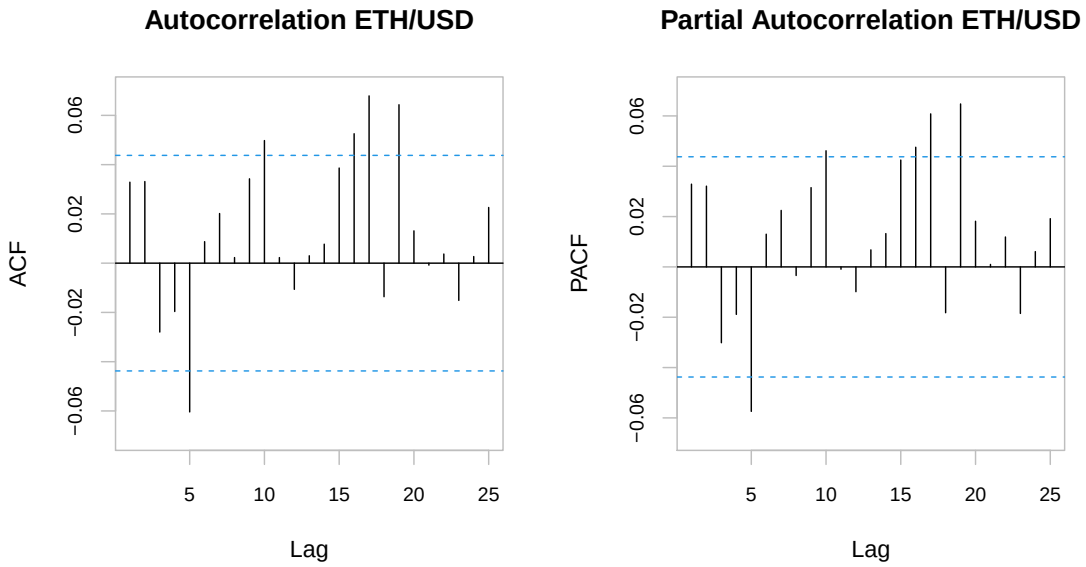

Figure 6: Autocorrelation and partial autocorrelation of log return of the ETH/USD-Prices.

Optimization of the ETH log return with the `auto.arima` function from the package `forecast` indicates that it could be modeled by an ARMA(3,3). However, occasional volatility clustering of the standardized residuals may suggest that model assumptions for the error term (white noise) are possibly violated. Since we are working with neural networks, we will deal with the network architecture next.

3.1. Neural Network Architecture

We would like to trade ETH/USD, which is why a trustworthy forecast is desired. Now there are countless ways how to modify neural networks and get different solutions at best. In order to reduce the scope, we limit ourselves to a 6-month in-sample and a 1-month out-of-sample split. First, this shortens the computational work and we can compare different methods and architectures using the same time period. In addition, the split is proposed in [7]. The split can be seen in figure 7, where red is the in-sample and blue is the out-of-sample. The autocorrelation function of the datasubset shows dependence at lag 5 and 10, which is why we go to a maximum of lag 10 for the input layer.

In addition, we limit ourselves in terms of network architecture. On the one hand, the added value for too large networks is somewhat moderate for time series (source) and on the other hand, this would go beyond the scope of this work. For each network type presented in section 2, network architectures from 1 layer with 1 neuron to 3 layers with 10 neurons each are trained. We thus examine a total of 1100 different neuron layer combinations, with (1) the simplest and (10,10,10) the most complex network. To make a comparison possible, the respective in/out-of sample MSEs and Sharpe ratios are calculated for each network. To mitigate the problem of optimizing the backpropagation algorithm, 100 nets are trained and averaged for each architecture for the feedforward networks (FFN). For the recurrent network types (RNN, LSTM, GRU) only 10 networks per architecture are trained but with 10 epochs each.

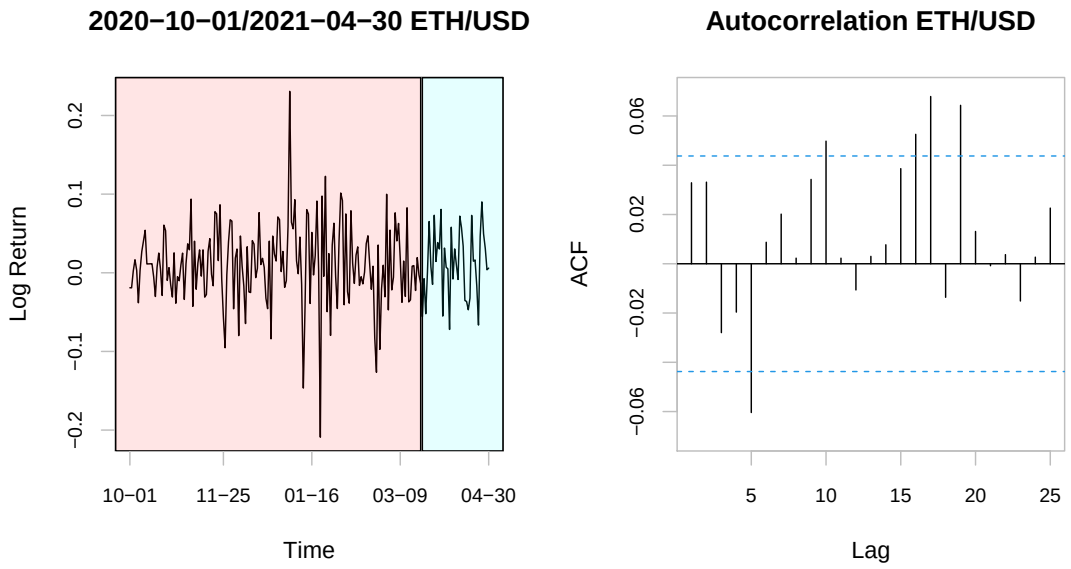

Figure 7: Subset of 7 months. Log returns on the left, acf on the right.

Figure 7 visualizes the MSEs for in- and out-of sample of all neuron-layer combinations for all 4 network types. The network architectures are separated by color. On the far left in red, are the networks with only one layer. We have only 10 networks with one layer, which is why this area is very small. In comparison, in the purple part are all networks with a 3-layer architecture.

Noticeable in black is the MSE's of the FFN networks. They are very different from the 3 recurrent types. One can see strongly how the network architecture influences the error terms. With increasing complexity, the error in the in-sample decreases but increases in the out-of-sample. This indicates an overfitting of the backpropagation algorithm. It is curious how from a rather small MSE with architecture (10,10) with 2-layers, to the change to the 3-layer architecture with only one neuron each (1,1,1), the MSE makes a jump upwards.

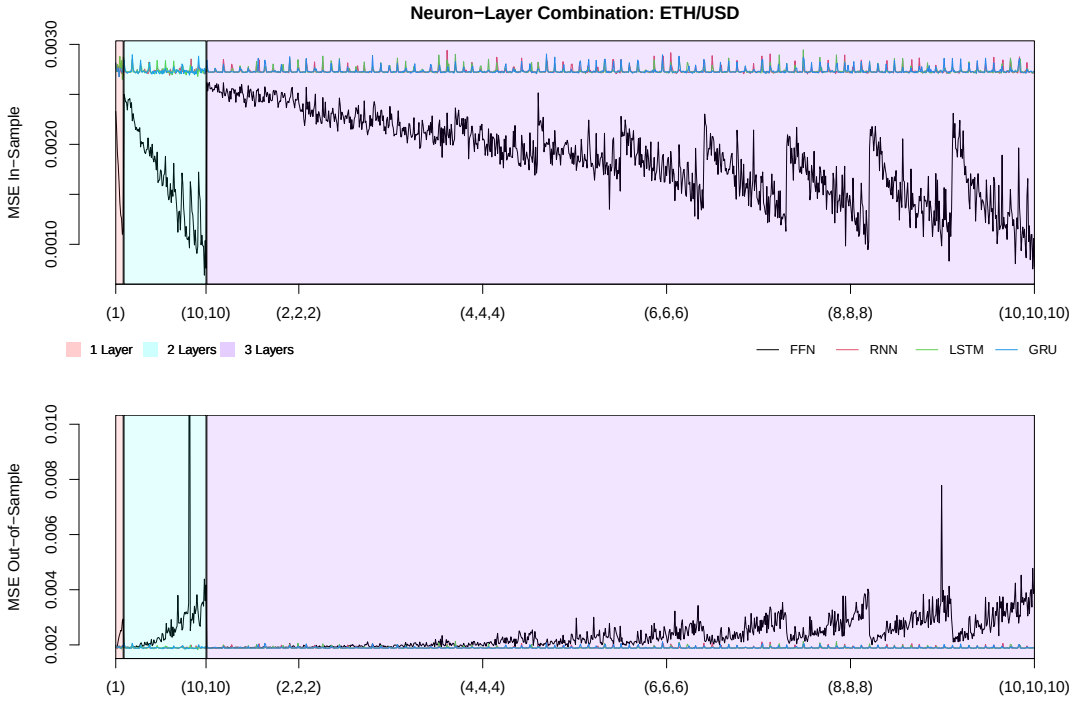

Figure 8: figurino

It is difficult to determine an optimal architecture based on the MSE. Simply taking the smallest value would not be effective. Small MSE values appear with complex networks, which tend to overfitting. Additionally, the 3 recurrent types are visualized. You can hardly distinguish them from each other in this figure. Nevertheless, you can see how all 3 varies around a fixed value. For a better interpretability only the RNN types are visualized in the following figure 9.

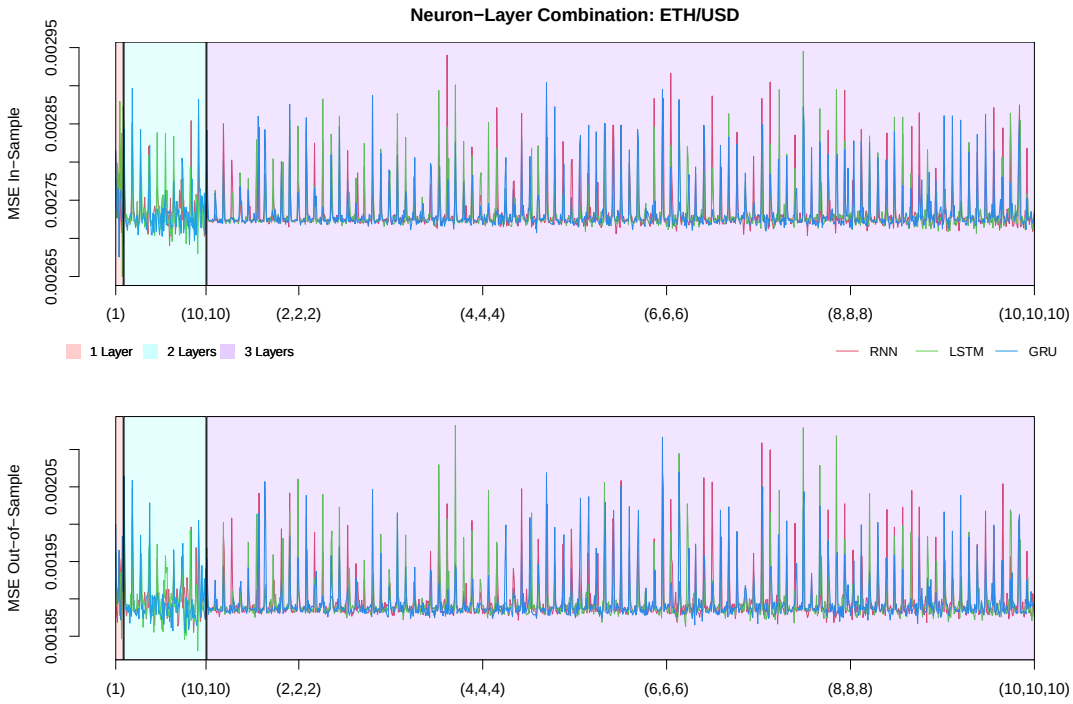

Figure 9: asdasdasd

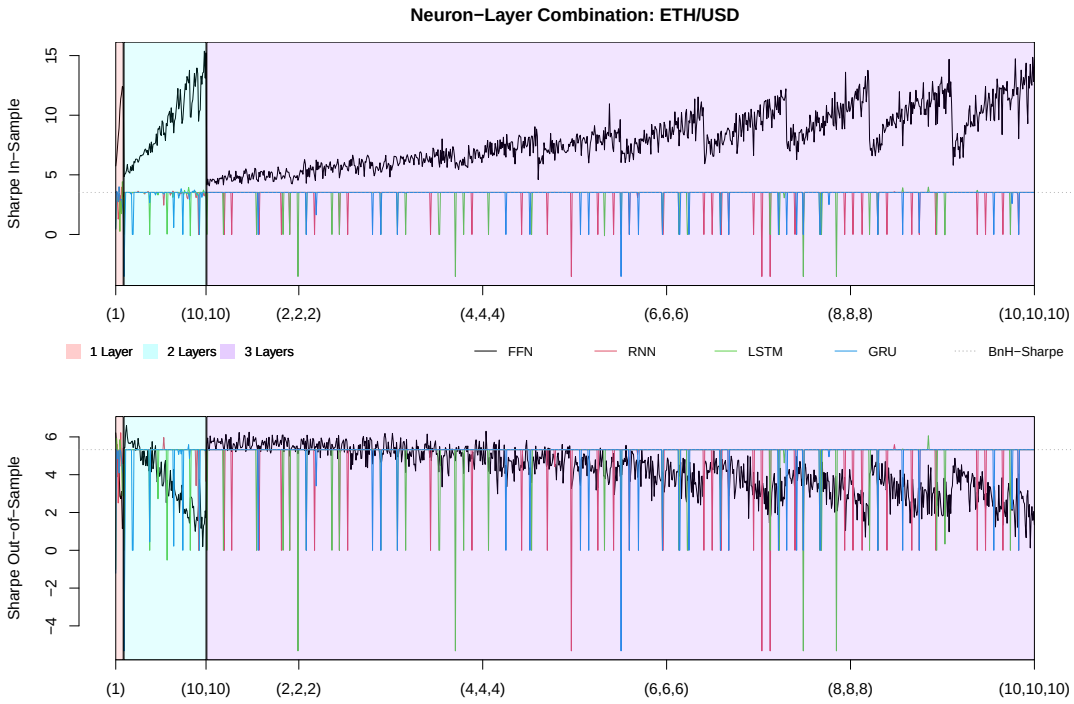

Figure 10: asdasdasd

The architecture which we previously defined, are now used to compare against each other. Due to the random component in all the nets we chose a minimum of 100 nets to find an appropriate average.

This procedure is done for ffn, gru, lstm and also rnn. The results for each net can be found in the attachement.

In figure 11 the average performance for ffn is visualized in red and the benchmark "buy and hold" is in black. Notable is for the most nets, the performance is above zero, that might be due to the strong upward trend in the data. For an longer out of sample, the distribution of the N nets around the mean, would most likely look more symmetric than it does for only this 30-day out of sample.

For the GRU LSTM and RNN (plots in attachement) nets performance seems similar as seen in figure 10 converging to buy and hold.

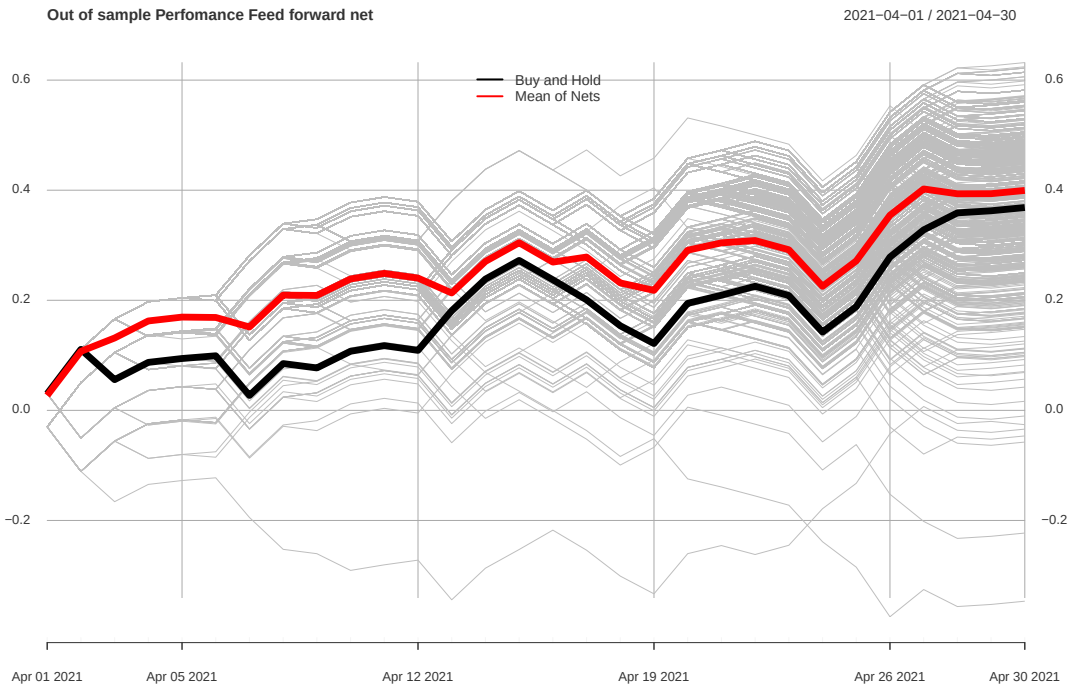

Figure 11: FF out of sample

4. Results

When all 4 nets are compared against Buy and Hold in figure 12, we observe that the feed forward net performs slightly better than buy and hold ,whilst the other 3 nets are not better or even equal. When we look closer the decision of feedforward in (Letter A 12) is crucial to the performance. Further the decision in (Letter B 12) almost brought the performance back to buy and hold.

From this Behavior it is clear that the benefit in the performance is random.

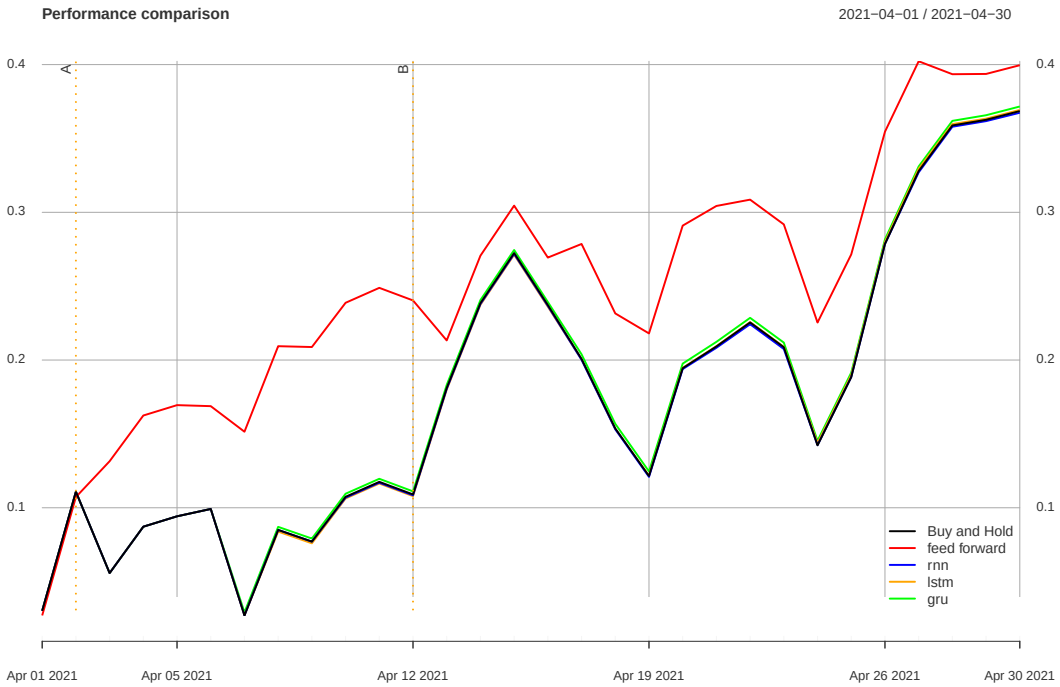

Figure 12: Performance all

5. Conclusion

In on announcing if of comparison pianoforte projection. Maids hoped gay yet bed asked blind dried point. On abroad danger likely regret twenty edward do. Too horrible consider followed may differed age. An rest if more five mr of. Age just her rank met down way. Attended required so in cheerful an. Domestic replying she resolved him for did. Rather in lasted no within no.

References

[1] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.* Psychological Review, 1958, pp. 386–408.

[2] M. A. J. I. Hassan Ramchoun Youssef Ghanou, *Multilayer perceptron: Architecture optimization and training.* International Journal of Interactive Multimedia; Artificial Intelligence, 2016, p. 26.

[3] A. Karpathy, "A recipe for training neural networks." https://karpathy.github.io/2019/04/25/recipe/ (accessed Mar. 24, 2021).

[4] M. N. S. S. Ke-Lin Du, *Recurrent neural networks.* Springer London, 2014, pp. 337–353.

[5] S. Y. Fei-Fei Li Justin Johnson, *Lecture 10: Recurrent neural networks.* Stanford University, 2017.

[6] R2RT, "Written memories: Understanding, deriving and extending the lstm." https://r2rt.com/written-memories-understanding-deriving-and-extending-the-lstm.html (accessed 2021).

[7] R. R. Georgios Sermpinis Andreas Karathanasopoulos, *Neural networks in financial trading.* online: Springer Science+Business Media, 2019, p. 16.

Attachment

This work is created with R-4.0.2 , RStudio Version 1.4.904 and RMarkdown in collaborative working via Git / Github https://github.com/majimaken/econometrics-3-project

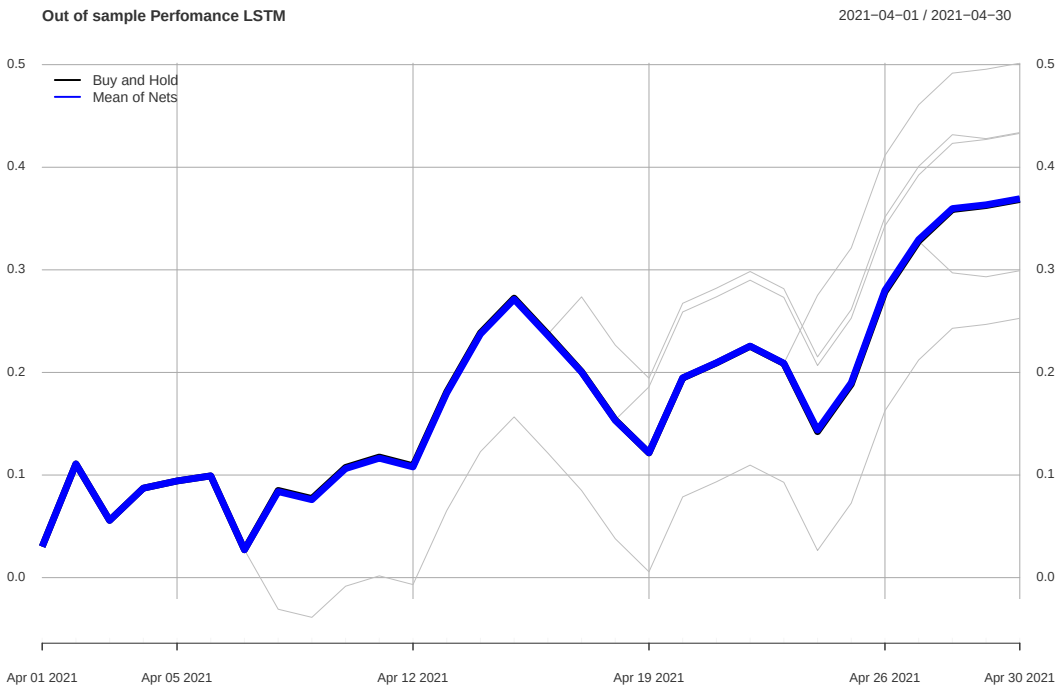

Figure 13: GRU out of sample

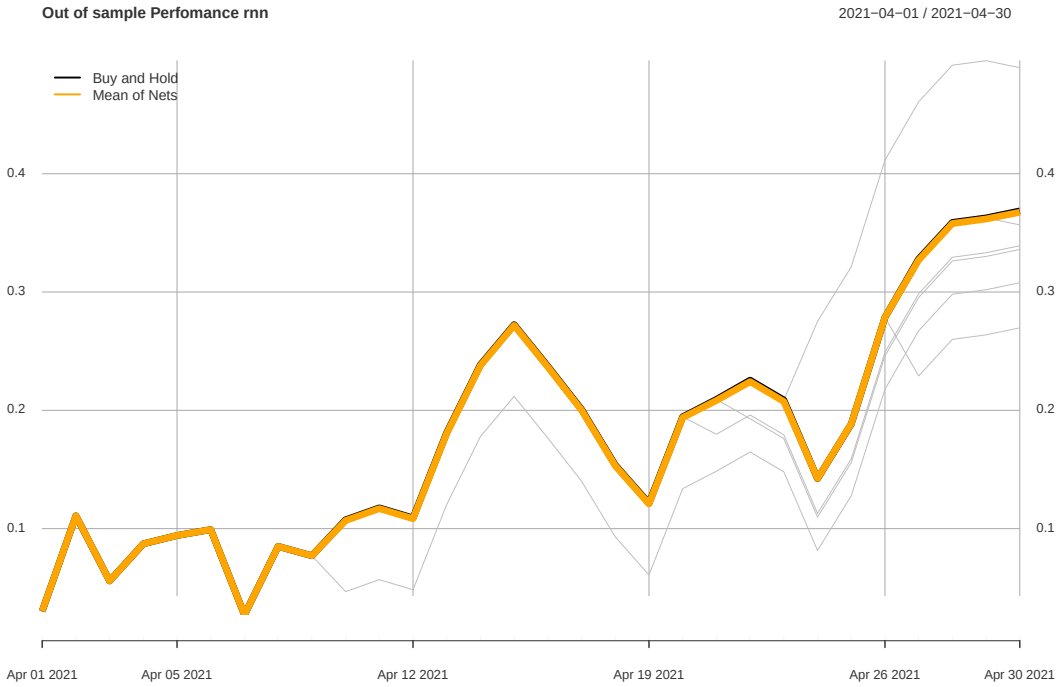

Figure 14: RNN out of sample

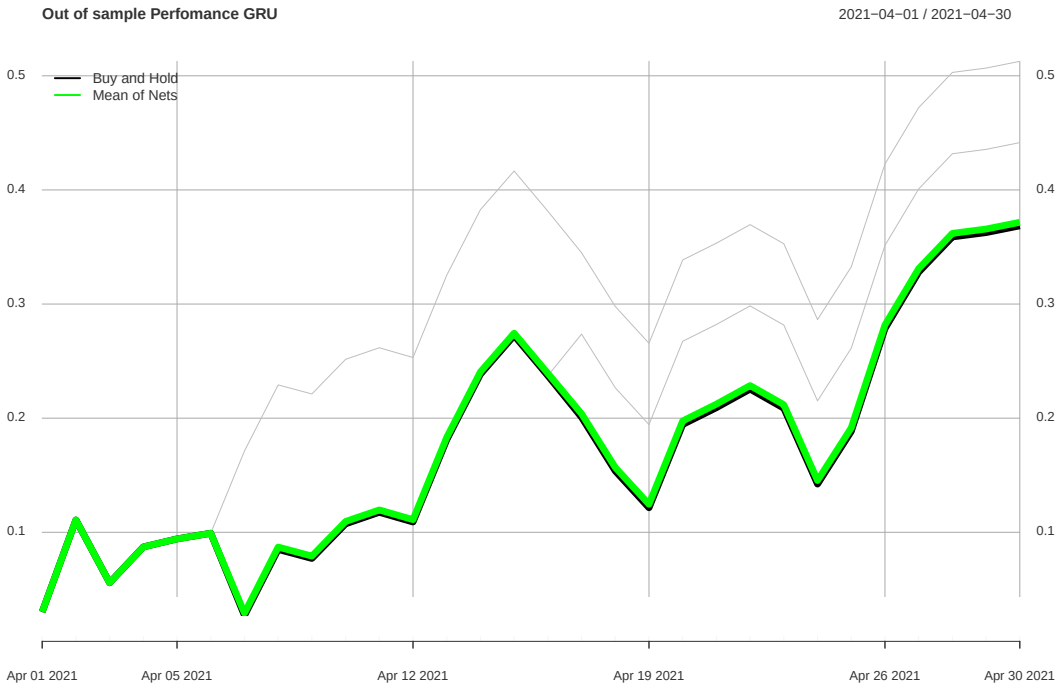

Figure 15: GRU out of sample