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

The use of neural networks and their utility in financial time series is a frequently studied topic. Plain vanilla feedforward neural networks (FFN), recurrent neural networks (RNN), gated recurrent unit (GRU), and long short-term memory (LSTM) are applied and compared. The impact of the choice of a specific network type and network architecture (number of layers and neurons) on trading performance is investigated. Using a fixed in-sample and out-of-sample, all possible combinations between the simplest (one layer and one neuron) and the most complex network (three layers and ten neurons) are trained and their trading performance is quantified and compared to the buy-and-hold benchmark. The results of the recurrent networks are disappointing regarding financial time series, however, FFN's are worth pursuing and further investigating.

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, 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 the 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.

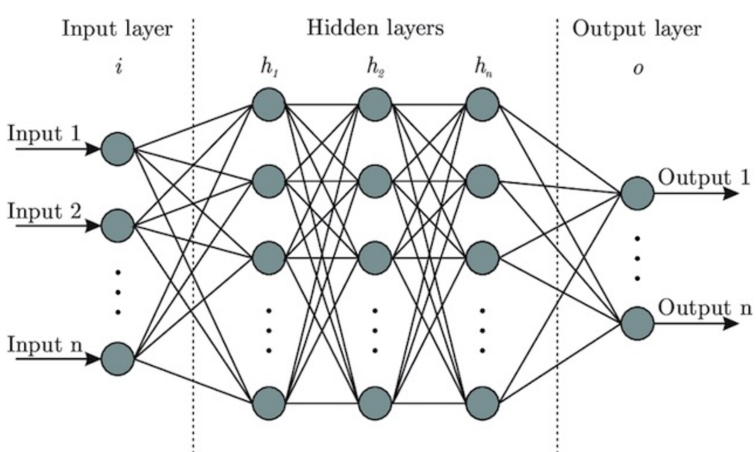

Figure 1: Schematic diagram of a multilayer perceptron

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 10 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 uses new inputs x_i in each epoch, RNN also uses sequential data h_i in addition to x_i. These 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 in the next prediction [4].

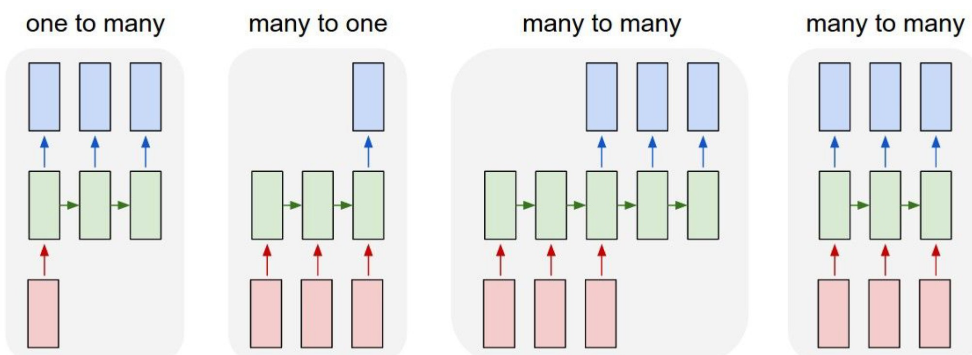

Figure 2: Process sequences of different applicances of RNN.

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 a 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.

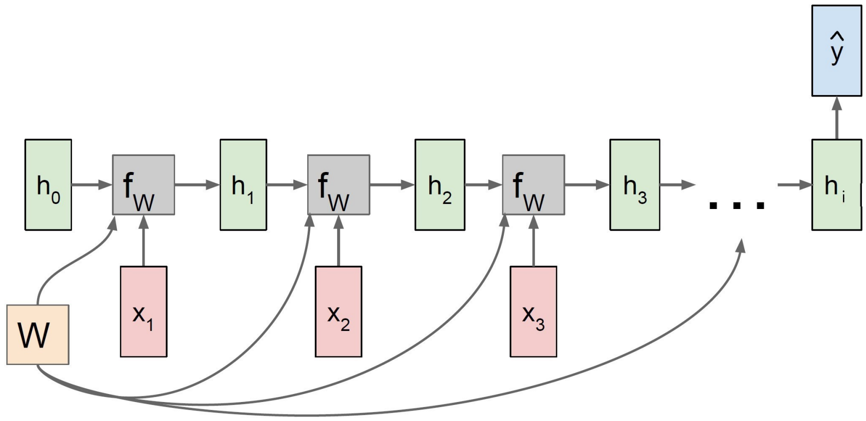

Figure 3: Computational graph of a many-to-one RNN.

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

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

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].

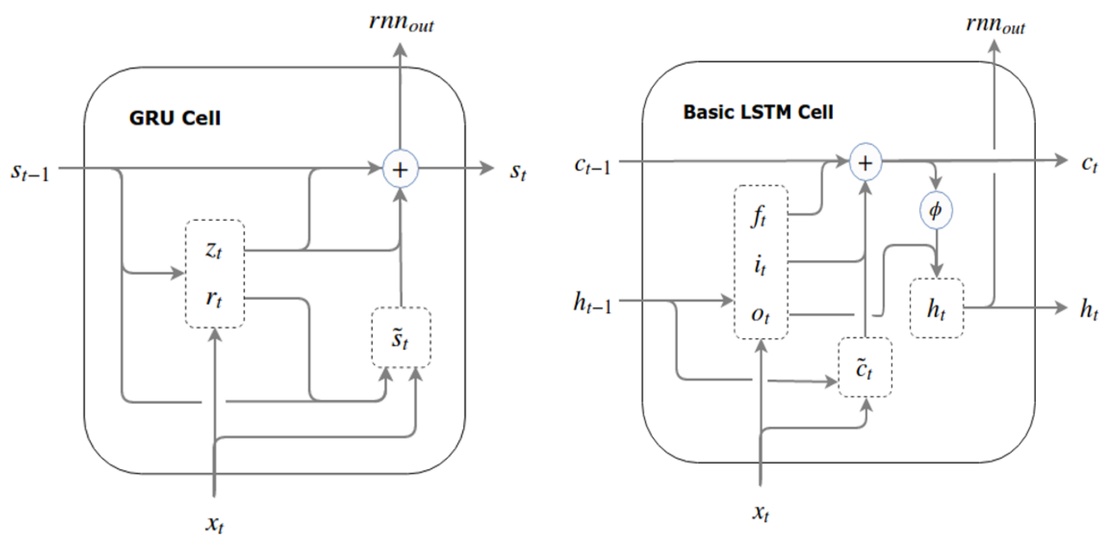

Figure 4: Workflow of a GRU cell (left) and LSTM cell (right).

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 performance is evaluated. The application is tested and evaluated using the time series of the Ether price (ETH). Ether is the cryptocurrency of the blockchain technology Ethereum.

3.1. Data Exploration

First, we will explore 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.

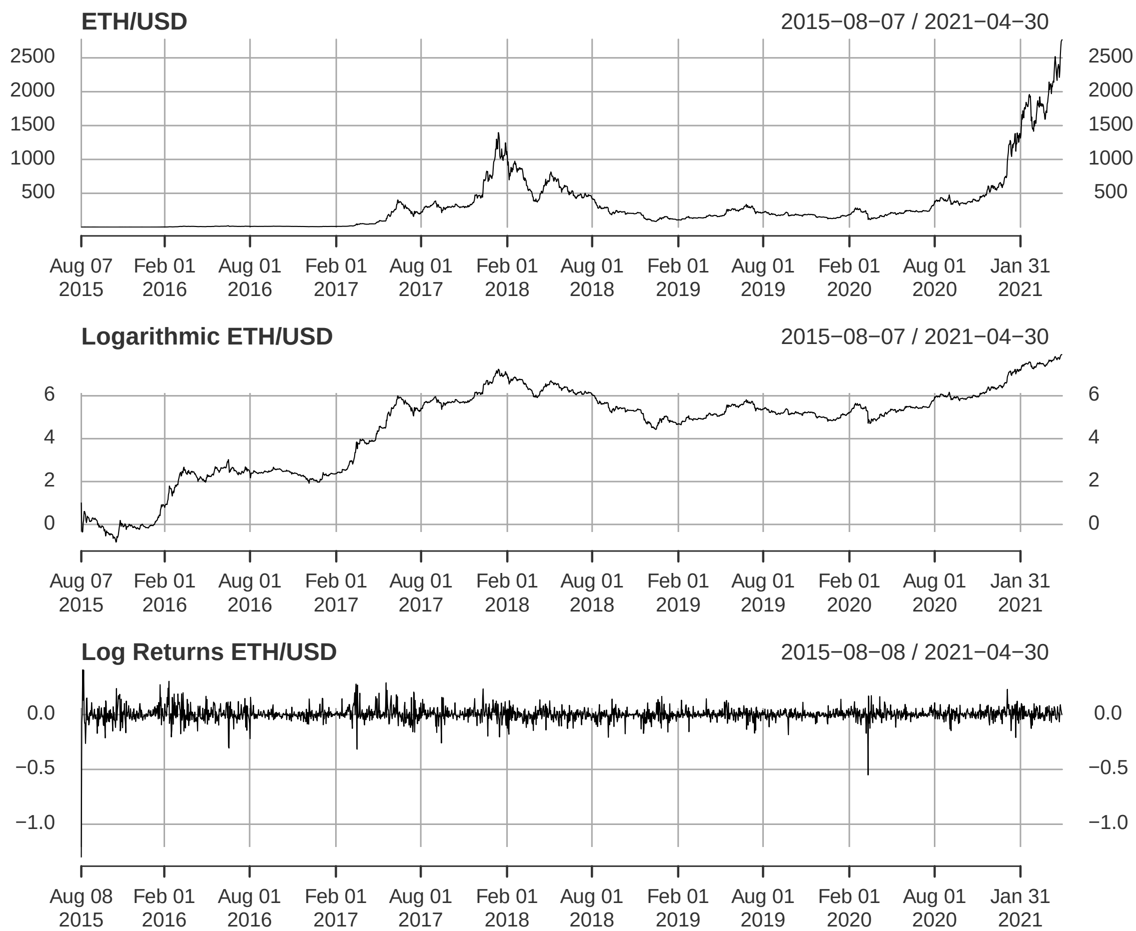

Figure 5: Time plot of the price, logarithmized price and log return based on the price for ETH/USD. Large (crypto) market phase dependencies and volatility persistence are evident.

Continuing with the autocorrelation (ACF) and partial autocorrelation (PACF), we notice a dependency structure in both cases. The ACF plot in figure 6 shows 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.

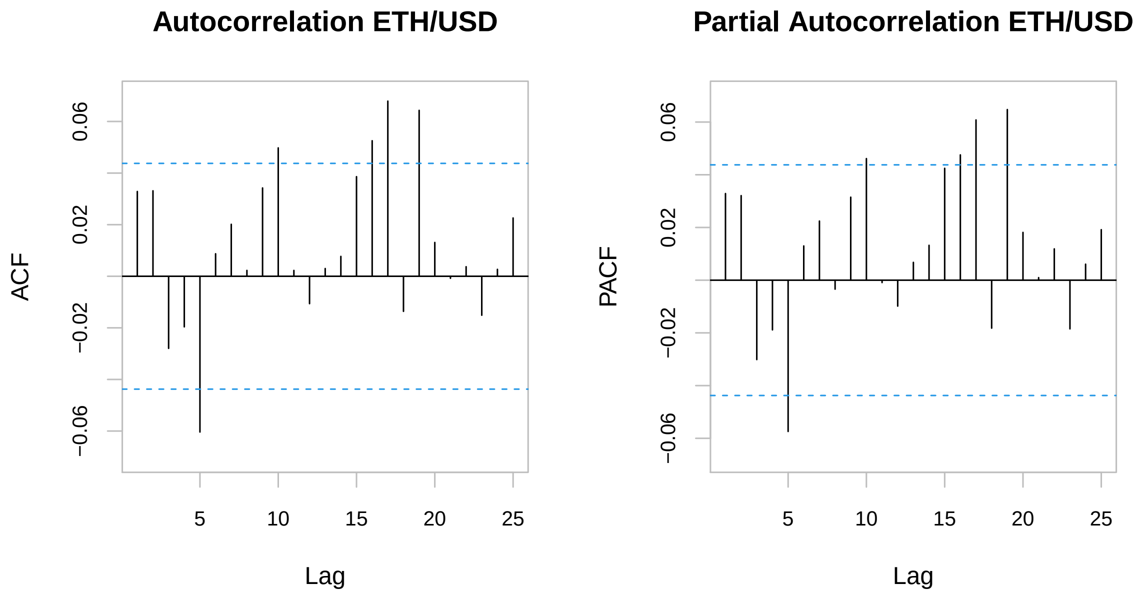

Figure 6: Autocorrelation and partial autocorrelation of log return of the ETH/USD-Prices.

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 assumption 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

The goal is to trade ETH/USD, which is why a trustworthy forecast is desired. 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, as G. Sermpinis and A. Karathanasopoulos proposed [7]. This shortens the computational work and we can compare different methods and architectures using the same time period. The split can be seen in figure 7, where red is the in-sample and blue represents the out-of-sample. The autocorrelation function of the data subset shows dependence at lags 5 and 10, which is why we go to a maximum of 10 lags 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 MSE's 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.

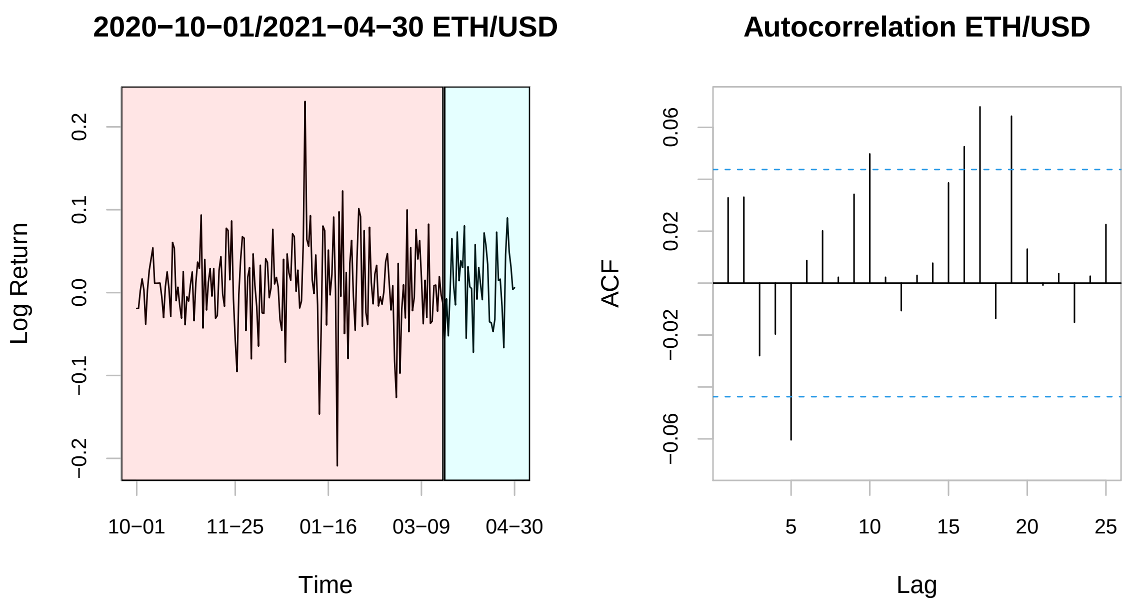

Figure 7: Subset of 7 months. Log returns on the left, acf on the right.

Figure 7 visualizes the MSE's 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 are the MSE's of the FFN. They are very different from the 3 recurrent types. We can observe 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 overfitting of the backpropagation algorithm. It is remarkable, that the in-sample MSE with the architecture (10,10) results in a much smaller error than a very simple model with three layers such as (1,1,1). The out-of-sample MSE improves drastically by using an architecture of (1,1,1). In this case, the addition of another layer adds value despite the small number of neurons.

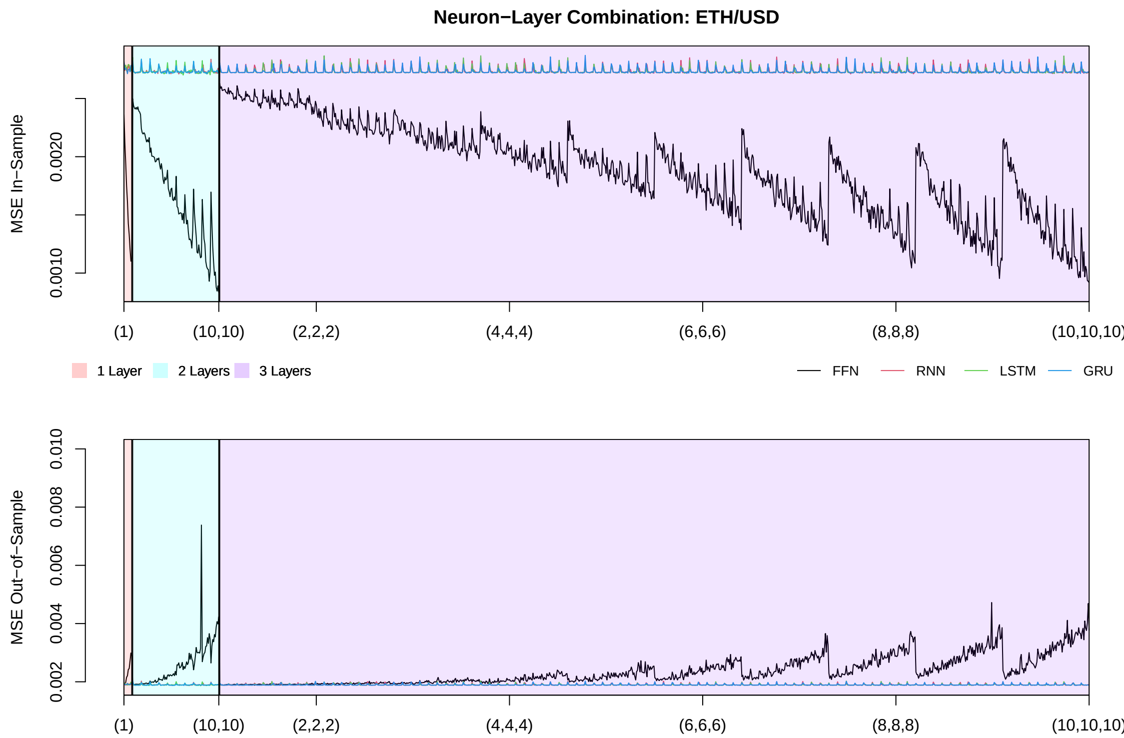

Figure 8: In and out-of sample MSE of all neural networks. For FFN, 100 networks were averaged in each case. For the recurrent ones, only 10 were averaged.

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

Here you can see the structure of the recurrent networks a little better. There is no pattern visible as observed in the FFN, the MSE's have no inverse correlation between in- and out-of-sample. There is a certain value that is not undercut. The MSE does not exceed a certain value. The spikes that occur uniformly are striking. An examination of the results showed that the spikes appear when the last layer of architecture contains just one or two neurons. In the attachment in figure 13, these combinations have been removed, therefore the spikes vanished. Keep in mind, however, that the differences between apparent convergence and a spike are not very large. Furthermore, only 10 networks per combination were trained. This structure could possibly look weakened if one would increase the number of networks.

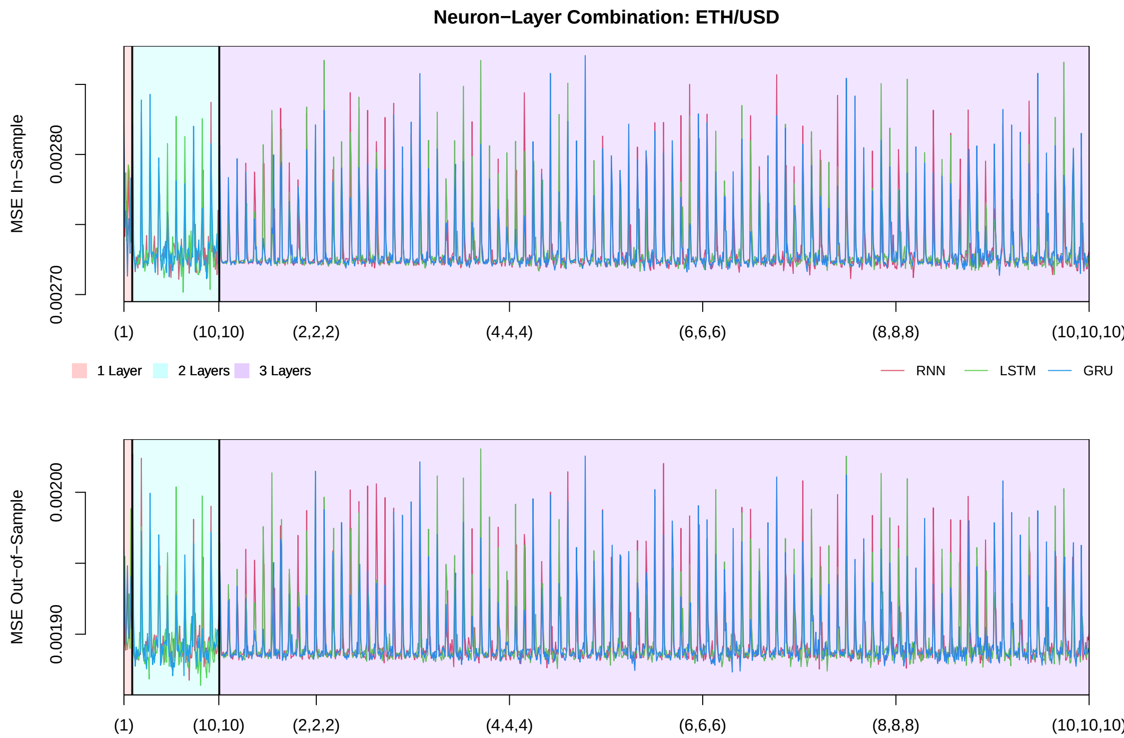

Figure 9: In and out-of sample MSE of the 3 different recurrent network types.

Nevertheless, we cannot use this visualization to determine an optimal network architecture based on the MSE. Regardless of the complexity, one achieves approximately the same error.

Here in figure 10, the Sharpe ratios are visualized. In the case of recurrent networks, one can again see this strange repeating pattern, which can be corrected by possible removal of some network architectures (see above). The dotted line shows the buy-and-hold (BnH) Sharpe. The trading signals generated by the recurrent networks (daily signum-trading) seem to be only as good as BnH. Looking closely, there are a few who beat the benchmark, but most likely the results are random.

As for the MSE's, the structure looks very similar to the Sharpe Ratios for the FFN. The rather small in-sample MSE's in figure 8 of the FFN, lead to high in-sample Sharpe ratios, but to rather low out-of-sample Sharpe ratios. This can also be explained by a too complex network, which leads to overfitting. Nevertheless, there are some architectures that show better out-of-sample Sharpe's than BnH and are not in the realm of overfitting. The most optimal out-of-sample Sharpe for FFN can be found with 2 layers, 1 neuron in the first and 4 neurons in the second layer (black dot).

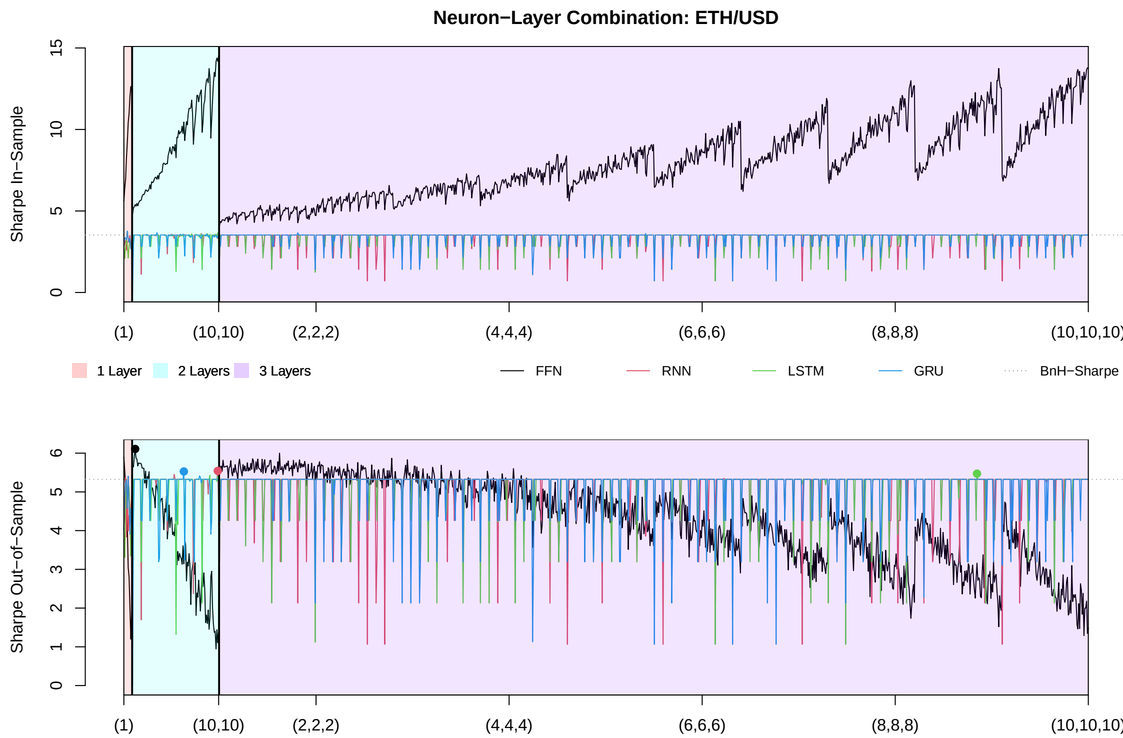

Figure 10: In and out-of sample Sharpe ratios of all neural networks.

For the recurrent network types, it is not so easy to find an optimal network. Regardless of the network architecture, they all perform equally well as BnH. This whole investigation gives us an added value for FFN but not for the recurrent types. We find strange structures for certain network architectures (1 or 2 neurons in the last layer) but cannot determine a clear network architecture. For further procedure, we determine the highest out-of-sample Sharpe ratios as the optimal network (indicated with the colored dots in each case). Keep in mind, however, that the optimization is only for a number of 10 networks. With a larger number, individual well-performing networks would not stand out as much.

3.1. Trading Strategy

The trading strategy is based on the predicted log returns for $t+1$. If the neural network used predicts a positive return, we stay in the market (signal = 1). If the forecast is negative, we enter a short position (signal = -1).

For this trading strategy we ignore the potential transaction costs. It is also assumed that long and short positions in ETH are possible.

3.2. Trading Performance

Resulting from the previous procedure, we pick the network architecture with the highest out-of-sample Sharpe and repeat this for every network type. The following models are selected for further analysis:

- FFN (1, 4)
- RNN (7) 30 epochs
- LSTM (10,9) 30 epochs
- GRU (6, 10) 30 epochs

These architectures are used to train and test a model on the data from figure 7. Due to the random component in all the neural networks, we compute 100 networks each (10000 for FFN) and calculate the average for every time step.

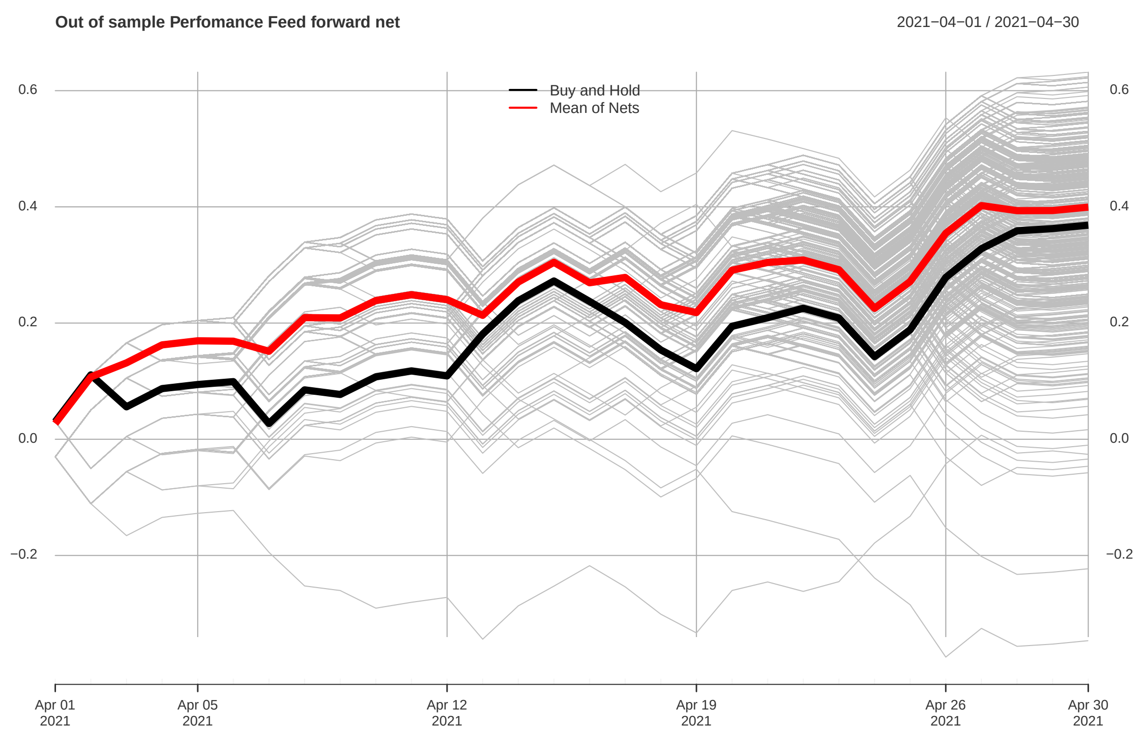

Figure 11: FF out of sample

In figure 11, the performances of all optimized FFN are plotted. The red line represents the average of all optimized FFN, which performs better than the buy-and-hold in black. In addition, it is worth noting that most realizations of the FFN have a performance better than zero. This is most likely due to the upward trend of the ETH. For a longer out-of-sample, the distribution of all the optimized FFN would most likely look more symmetric and scatter more evenly around the average than it does for only this 30-day out of sample. This procedure is repeated for FNN, GRU, LSTM and RNN. The results for each network type can be found in the attachement. For the GRU, LSTM and RNN nets performance seems similar as seen in figure 10, converging to buy-and-hold.

4. Results

The performances of all four models are compared with a buy-and-hold strategy, which can be seen in figure 12. We observe that the FFN performs slightly better than our benchmark, while the other three clearly do not meet the benchmark. Looking more closely, it can be seen that one single decision (letter A in figure 12) of the FFN leads to the overperformance. If this event did not occur, the performance would also be worse than the benchmark. Further, a second decision (letter B in figure 12) worsens the performance and brings it back to a level similar to buy-and-hold. These two events appear to have occurred coincidentally. To test whether FFN systematically beats the buy-and-hold, it would be necessary to extend the time window and test different in- and out-of-samples and compare their performance. These findings strongly suggest that the individual decisions, and thus the performance of the examined neural networks, is determined by chance.

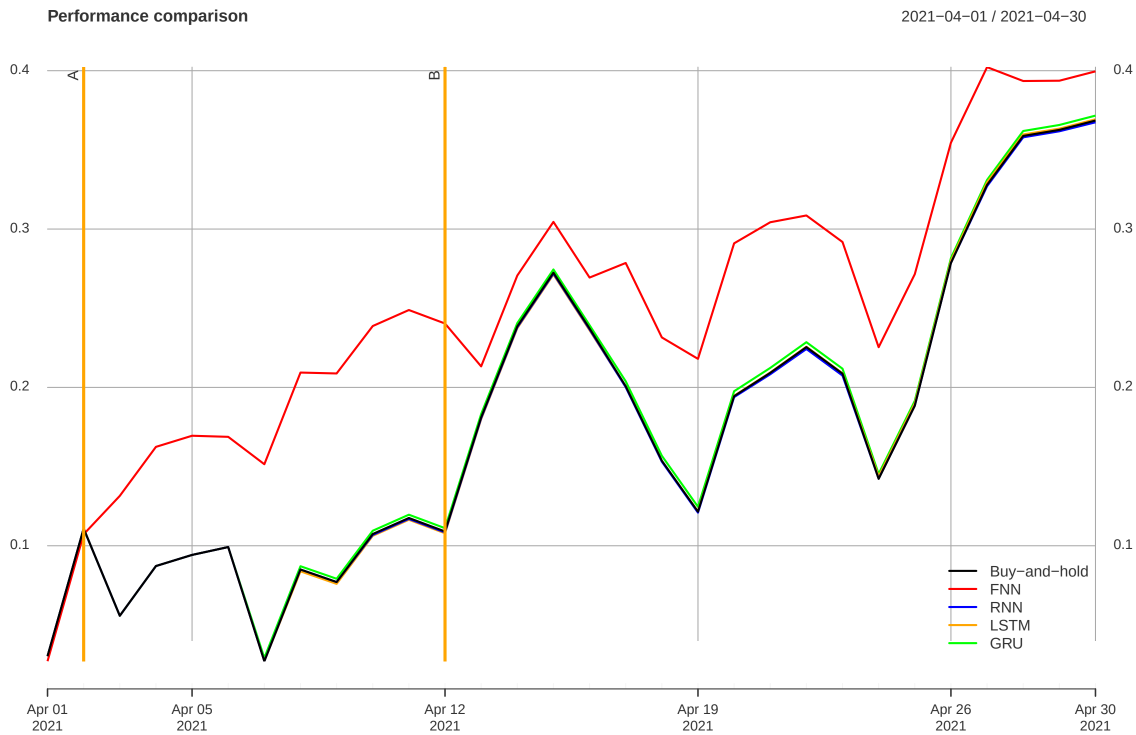

Figure 12: Performance all

5. Conclusion

5.1. Summary

The application of neural networks to financial time series is a newer topic. We used and compared feedforward neural networks (FFN), recurrent neural networks (RNN), gated recurrent unit (GRU), and long short-term memory (LSTM). The effect of network type and network architecture on trading performance was analyzed. All possible combinations between the simplest (one layer and one neuron) and the most complex network (three layers and ten neurons) were trained 100 times each and their average trading output was evaluated.

5.2. Conclusion and Outlook

When comparing the network architectures, it can be observed that with FFN the overfitting increases as the number of neurons increase. The same phenomenon cannot be observed to the same extent for RNN, GRU and LSTM. Presumably, there is a numerical problem with the optimization algorithm in the three types of RNN.

Looking at the trading performance, it can be seen that the cumulative returns converge to buy-and-hold in the most optimal case. Although the performance of FFN turns out to be better than the three competitors, this is most likely a random event and will be different for other time frames.

The advantage of recurrent neural networks is that they receive information from the last step. While this can be a great advantage in many applications, there is no benefit to our method. Since the lagged log-returns are used as input, in principle historical information is also available to the FFN. Furthermore, the usefulness of long-term memory is also very limited, as it is not suitable for detecting effects in noisy data.

Based on the points mentioned, we see the following opportunities to further develop this Thesis:

- Instead of the R Package 'rnn' used, alternatives could be used, such as 'keras.' This could possibly help to explain why the RNN, GRU, and LSTM optimizations behaved so strangely.
- The behavior of the neural networks was only considered for a certain time period. These 6+1 months do not prevent enough information about an asset class and should be evaluated for more different time periods.
- Lagged log-returns were always used as input in all cases. It could be examined in a further step whether adding more explanatory variables would provide added value. In particular, known indicators of technical analysis, such as s smoothing moving average or the relative strength index (RSI).

In recapitulation of the results, we conclude that the recurrent forms RNN, GRU and LSTM bring no further benefit in forecasting of financial time series. In particular, this is the case since FFN is also fed with historical log returns. On this basis, we believe that the potential of FFN could be further explored with the aforementioned suggestions.

6. 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.

7. 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

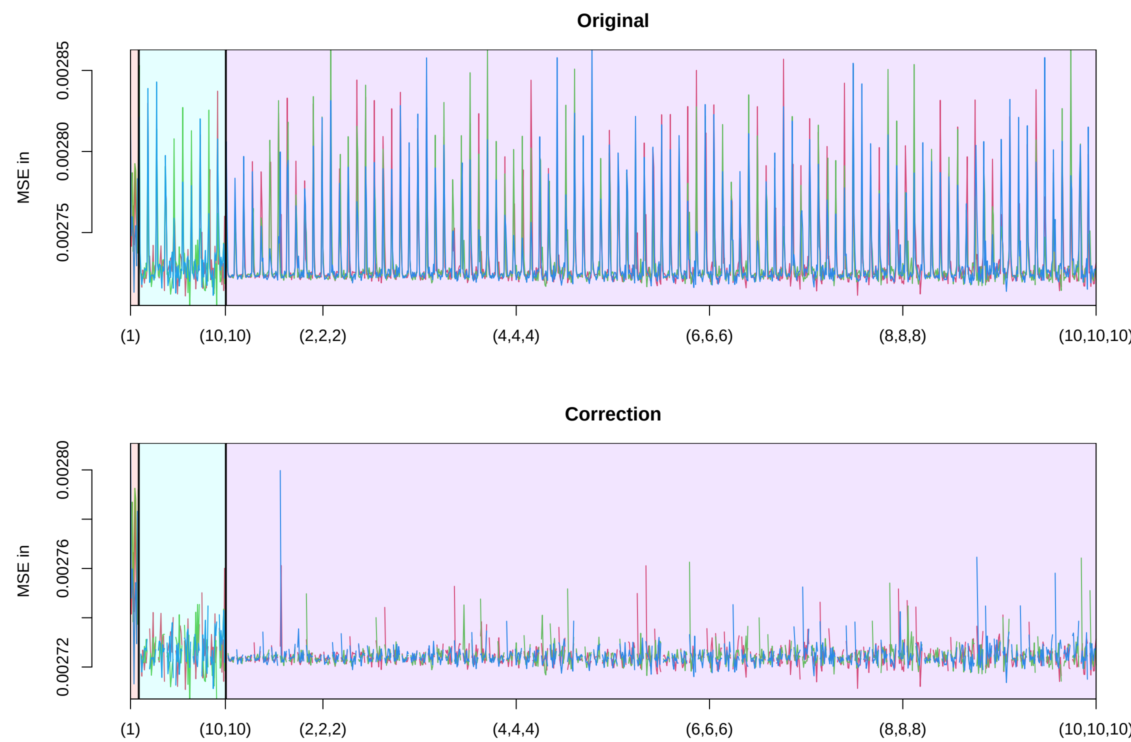

Figure 13: The upper plot shows the original values. In the lower plot, the values with 1 or 2 neurons in the last layer have been removed.

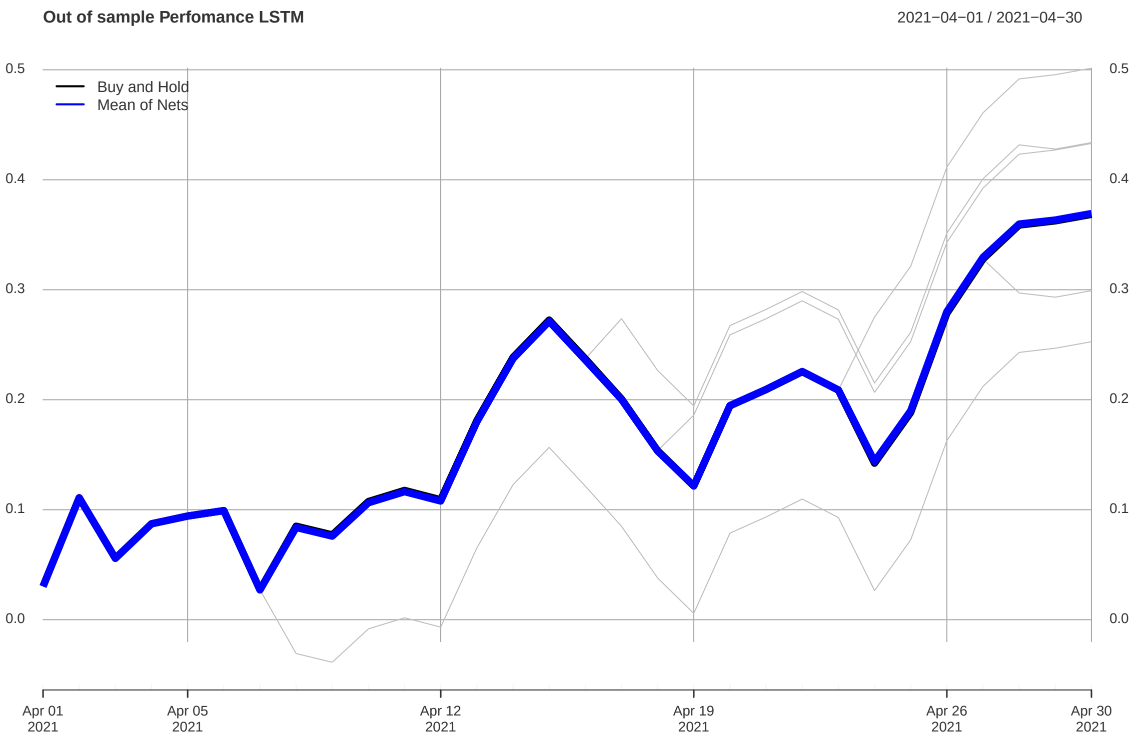

Figure 14: GRU out of sample

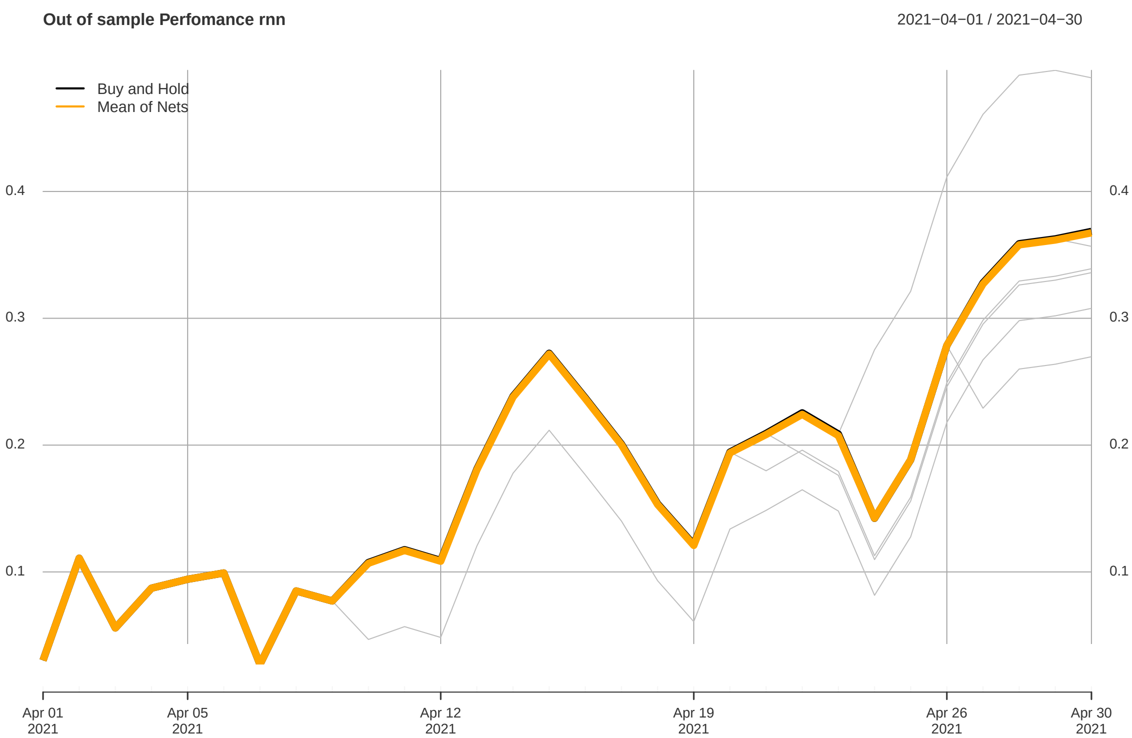

Figure 15: RNN out of sample

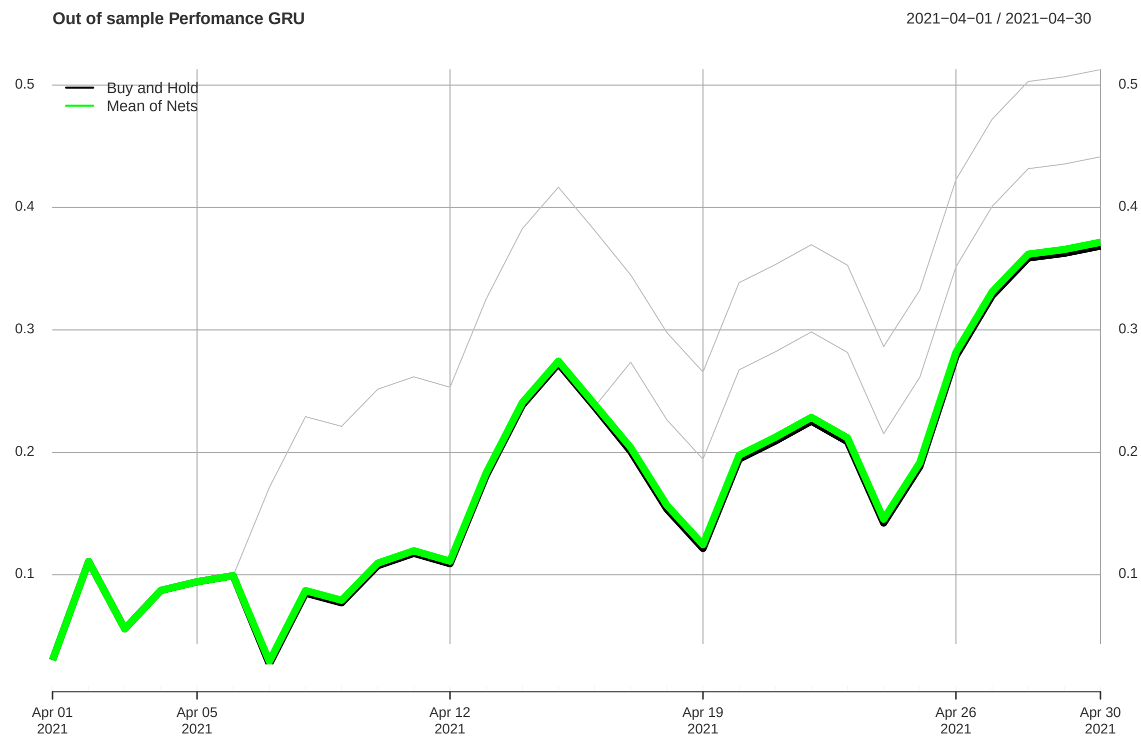

Figure 16: GRU out of sample