UPPSALA
UNIVERSITET

Forecasting the Nasdaq-100 index using GRU and ARIMA

David Cederberg and Daniel Tanta

Bachelor's thesis in Statistics

Advisor
Martin Solberger

2022

Abstract

Today, there is an overwhelming amount of data that is being collected when it comes to financial markets. For forecasting stock indexes, many models rely only on historical values of the index itself. One such model is the ARIMA model. Over the last decades, machine learning models have challenged the classical time series models, such as ARIMA. The purpose of this thesis is to study the ability to make predictions based solely on the historical values of an index, by using a certain subset of machine learning models: a neural network in the form of a Gated Recurrent Unit (GRU). The GRU model's ability to predict a financial market is compared to the ability of a simple ARIMA model. The financial market that was chosen to make the comparison was the American stock index Nasdaq-100, i.e., an index of the 100 largest non-financial companies on NASDAQ. Our results indicate that GRU is unable to outperform ARIMA in predicting the Nasdaq-100 index. For the evaluation, multiple GRU models with various combinations of different hyperparameters were created. The accuracies of these models were then compared to the accuracy of an ARIMA model by applying a conventional forecast accuracy test, which showed that there were significant differences in the accuracy of the models, in favor of ARIMA.

Keywords: Neural network, GRU, ARIMA, stock market, forecasting, machine learning.

Table of contents

1. Introduction

The modern investor has access to an overwhelming amount of financial data at a very low cost. Companies like Renaissance Technologies and D.E. Shaw was successful in developing a pattern recognition model that could forecast short term price changes. As technology and computer power has advanced, many traders are now competing by constructing models that forecast more accurate price predictions. The competitive market has led to the development of different modern models that are challenging the classical time series models (Berk and Demarzo, 2020).

Ever since the introduction of machine learning many have tried using its different techniques to forecast future price of securities on the stock market. However, trying to use machine learning to forecast the market is a difficult task, especially when only using historical data. This is because markets have different characteristics compared to natural phenomena such as weather patterns. Despite it being a difficult task, it is not an impossible one (Chollet and Allaire, 2018).

The goal of this study is to evaluate the predictive ability of a Gated Recurrent Unit, GRU, model. The GRU model will be compared to another model that will use traditional statistical methods. The predictions will be made on the Nasdaq-100 index, an American stock exchange index. The GRU model is a modern machine learning method in the form of a Recurrent Neural Network, RNN. The other model is based on the literature of time series analysis in the form of an autoregressive integrated moving average, ARIMA, model. Both models will exclusively be using the same historical data from the Nasdaq-100 index for their forecasts.

There are previous studies which have tried to see if the machine learning models are able to beat the classical time series analysis models. One such study was made by Hewamalage et.al (2021) in which the authors discuss the competitiveness of RNN's, as a forecasting method. In the study the two RNN's, GRU and long-short-term memory, LSTM, models were compared to the established statistical model ARIMA. The comparison between the models were done on seasonal patterns. They concluded that the RNN's are good options for forecasting and that they can outperform established statistical models.

For the LSTM model, there is also quite extensive previous research that has been made to see if the model is able to make accurate predictions on the stock market. The results from these studies have generally suggested that LSTM shows promise for being a good tool for investments. For example, both studies by Roondiwala et.al (2017) and Shah et.al (2018) concluded similarly that the LSTM model performed well in making short term predictions.

There is also some research that has been made on GRU that shows that the GRU model's ability to forecast different stocks and stock markets is similar to the LSTM model. For example, a study by Gao et.al (2021) showed that both models performed efficiently when predicting stock price, and that neither of the models outperformed the other. However, when it comes to the comparison between the GRU models and established statistical methods performance on financial prediction, there seems to be a gap in the research.

The purpose of this study is thereby to evaluate how accurate a GRU model is in predicting the development of the Nasdaq-100 index, using a simple ARIMA model as a benchmark for the predictions.

Can a GRU model make more accurate predictions of the Nasdaq-100 index than a simple ARIMA model?

2. Background

In this section, the background theory is presented. The section begins with some definitions of the stock markets. Afterwards, core theory behind time series analysis and the components that make the ARIMA model are introduced. The last part of this section describes the background and theory behind neural networks, recurrent networks and, lastly, the GRU model.

2.1 Stock markets

Public companies trade their share on an organized market called a stock market or stock exchange. A stock is a share of the ownership in a company (Berk and Demarzo, 2020).

NASDAQ, an acronym for National Association of Securities Dealers Automated Quotations, is a global stock exchange and is the second largest in the world. Securities are financial instruments for example bonds, stocks and options. Meaning that the NASDAQ market includes more than just stocks. This study aims to predict the stock market index called Nasdaq-100. A stock market index measures a specific stock market. The Nasdaq-100 index "focusses on the 100 largest non-financial companies on the exchange" and is based on the market capitalization of each company (NASDAQ, 2022). Hundreds of funds track the Nasdaq-100 due to its broad sector coverage. Sectors such as technology, health care, industrial, telecommunication and more. The index is home to many branch-leading companies such as Apple, Intel, Microsoft, Google and more, making it one of the best indicators of global economic development (NASDAQ, 2022).

2.2 Stochastic processes

A sequence of random variables is in statistics known as a stochastic process. It is often used when modeling an observed time series, since it can be explained as the realization of a stochastic process. Let $\{Y_t\} = Y_1, Y_2, \ldots, Y_T$ be a sequence of random variables (a stochastic process) over time points $t = 1,2,\ldots,T$. Important to note is that an observed time series only has one observation per variable in the stochastic process. In other words, there is only one observation for each of the variables Y_1, Y_2 and so on. Therefore, to make robust inference on a time series, there are, in general, some specific assumptions that need to be fulfilled. The most important is the assumption of stationarity. A stationary process means that the probabilities associated with the variables in the stochastic process are constant over time.

There are two types of stationarity, strict stationarity and weak stationarity. For a process to be strictly stationary, then the joint distribution of $Y_{t_1}, Y_{t_2}, \ldots, Y_{t_n}$ needs to be the same as the joint distribution of $Y_{t_1-k}, Y_{t_2-k}, \ldots, Y_{t_n-k}$ for all time points t and all number of lags k. For a process to be weakly stationary, also called covariance stationary, then the following conditions need to be fulfilled:

$$E(Y_t) = E(Y_{t-k}) = \mu, \text{ for all t}, k \ ,$$

$$Cov(Y_t, Y_{t-k}) = E\{Y_t - \mu\}\{Y_{t-k} - \mu\} = \gamma(k), \text{for all } t, k \ .$$

That is, that the mean is constant for all variables in the stochastic process, and that the covariance between Y_t and Y_{t-k} (the autocovariance) only depends on the time difference k and not on the time point t (i.e., for any given k, $\gamma(k)$ is constant for all t). Note then that this implies that also the contemporary variances for the variables in the stochastic process are constant:

$$Var(Y_t) = E\{Y_t - \mu\}\{Y_t - \mu\} = \gamma(0), \text{for all } t \ .$$

In this thesis, when the term stationarity is used, it will always refer to the weaker form of stationarity (Cryer and Chan, 2008).

A common stationary process is the moving average, MA, process. The general MA process of order q can be formulated as an MA(q):

$$Y_t = e_t - \theta_1 e_{t-1} - \theta_2 e_{t-2} - \ldots - \theta_q e_{t-q} \ ,$$

where θ_j $(j = 1,2, \ldots, q)$ are parameters, q is the number of parameters and e_m $(m = t, t-1, \ldots, t-q)$ are unobserved white noise processes. A white noise process is a sequence of identically distributed, zero-mean, independent random variables. In the MA-equation the θ values represent the weights and e_t the variables which the different weights are applied to (Cryer and Chan, 2008).

Another common stationary process is the autoregressive, AR, process. The formula for the general AR process of order p can be formulated as an AR(p):

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + e_t \ ,$$

where ϕ_b $(b = 1,2,\ldots,p)$ are autoregressive parameters, p is the number of autoregressive parameters and e_t is a white noise process. This means that the current value of Y_t represents a linear combination of the past values of itself plus an added e_t which represents everything new in the time series that is not explained by the previous Y_t values (Cryer and Chan, 2008).

A process can also be a mixture of an AR and MA process of order p and q, respectively. In these cases, the process is called ARMA(p,q) and can be written as follows:

$$Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + e_t - \theta_1 e_{t-1} - \theta_2 e_{t-2} - \ldots - \theta_q e_{t-q} \ .$$

Note that an ARMA$(p,0)$ is just an ordinary AR process and that an ARMA$(0,q)$ is an ordinary MA process (Cryer and Chan, 2008).

2.3 ARIMA processes

A common process (or model) in times series analysis is the ARIMA process. It is a generalization of the ARMA process. The difference is that the ARIMA model can be used for non-stationary time series. This is possible due to the model's differencing ability. The general ARIMA model can be formulated as ARIMA(p,d,q):

$$\phi(B)\nabla^d Y_t = \theta(B) e_t \ , \qquad (1)$$

where ∇^d is the difference operator,

$$\nabla^d = Y_t - Y_{t-d} \ ,$$

and $\phi(B)$ and $\theta(B)$ are lag polynomials,

$$\phi(B) \ = \ 1 \ - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p \ ,$$

$$\theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \ldots - \theta_q B^q \ .$$

Here the p stands for the number of autoregressive lags and thereby represents the AR part of the model. The d stands for the number of differences that is needed for the model to be a stationary ARMA process. The q represents the MA part of the model. In most cases the ARIMA(p,d,q) process will be integrated of order one or order zero. If it is integrated of order zero, then that means that it is an ARMA (p,q) process (Ordóñez et.al, 2019; Mills, 2019). B is the lag operator which operates on a time series index and shifts time back (Asteriou and Hall, 2016; Cryer and Chan, 2008):

$$B^c Y_t = Y_{t-c} \ (c = 1,2,3, \ldots) \ .$$

Again, the ARIMA model is used for cases where the time series is non-stationary. To still be able to make parameter estimation in these cases, a transformation of the time series is required to make it stationary. This is where the I in ARIMA comes to use which represents the d and of which order the process is integrated. If $d = 1$, then the process is said to have a "unit root" and is integrated of the first order. The number of unit roots (the order of integration) depends on how many times the process needs to be differentiated to make it stationary.

A particular interesting process is the random walk, which is a special case of the ARIMA-process,

$$Y_t = Y_{t-1} + e_t \ .$$

The e_t is in this case the only decider for how much the time series moves from the previous step to the next one. The random walk is an ARIMA (1,1,0), where $\phi = 1$ (Mills, 2019).

2.4 Unit root tests

There are several tests to identify if the data contains a unit root. Two well-known tests are the Dickey-Fuller test and KPSS test. Both test for a unit root, the difference is their null hypothesis. The KPSS test assumes no unit root in the null hypothesis, as supposed to the Dickey-Fuller test, that assumes a unit root in the null hypothesis. However, setting the null

hypothesis that a unit root exists can lead to more differencing than necessary (Hyndman and Khandakar, 2008). In this thesis, KPSS will be used to test and decide the order of integration.

The KPSS test can be performed based on the following model (Kwiatkowski et.al, 1992):

$$Y_t = r_t + \varepsilon_t \,,$$

where ε_t is a stationary error, and r_t is a random walk (i.e., containing a unit root):

$$r_t \; = r_{t-1} + u_t \,,$$

where u_t is a process that is independently and identically distributed over time with mean zero and variance σ_u^2. Note that if $\sigma_u^2 = 0$, then r_t turns into a deterministic component, and Y_t becomes stationary. The test is then performed based on the following null hypothesis and alternative hypothesis:

$$H_0\colon\ \sigma_u^2 = 0, \text{ the process is stationary,}$$
$$H_1\colon \sigma_u^2 > 0, \text{ the process is not stationary.}$$

The test is executed by performing a separate regression. Let e_t be the residuals from the regression of Y_t on an intercept only, and define the partial sum $S_t = \sum_{i=1}^{t} e_i$. The KPSS test is the following Lagrange Multiplier test statistic:

$$\text{KPSS} = T^{-2} \frac{\sum_{t=1}^{T} S_t^2}{\hat{\sigma}_S^2} \,,$$

where $\hat{\sigma}_S^2$ is a consistent estimator of the so called "long-run" variance $\sigma_S^2 = \lim_{T\to\infty} T^{-1} E(S_t^2)$ that arises due to that the model error term ε_t (that relates to the regression residual) is assumed to be stationary, but not necessarily independent over time. A consistent estimator of σ_S^2 is:

$$\hat{\sigma}_S^2 = T^{-1} \sum_{t=1}^{T} e_t^2 + 2T^{-1} \sum_{s=1}^{l} w(s,l) \sum_{t=s+1}^{T} e_t e_{t-s} \,,$$

where $w(s,l)$ is a weight function that corresponds to the spectral window that is used. In our case the Bartlett window is used,

$$w(s,l) = 1 - \frac{s}{(l+1)},$$

where l is as a lag truncation parameter such that $l \to \infty$ as $T \to \infty$.

As $T \to \infty$, the test statistic tends in distribution to a non-standard distribution. The critical values can be found in Table 1.

Table 1. Critical values for the KPSS-test.

Critical level	Critical value
0.10	0.347
0.05	0.463
0.01	0.739

2.5 Order selection of an ARIMA

Recall that an ARIMA has orders p (order of the AR-part), q (order of the MA-part) and d (order of integration); see Equation (1). Before values for p and q are selected, d must be chosen. To select the order of integration for an ARIMA model, it is standard practice to use unit root tests, such as the KPSS test in Section 2.4. To select the remaining orders of the model (p and q), it is common to choose the numbers that minimize the Akaike information criterion,

$$AIC = -2log(L) + 2k,$$

where L is the likelihood of the data (see Section 2.6), $k = p + q + 1$ if the model has an intercept, otherwise $k = p + q$. The number 2 serves as a penalty function for complex models with too many parameters. This is to ensure parsimonious models (Chryer and Chan, 2014).

2.6 Estimating the parameters of an ARIMA

For given numbers of the orders p, d and q, the rest of the parameters can be estimated. After d is identified the parameters can be estimated as for a normal ARMA model. There are several approaches to estimate the parameters. Method of moments, least squares and maximum likelihood estimation (MLE) are common methods. MLE finds the value of the parameter that maximizes the likelihood function given the observed data (Hyndman and Athanasopoulos, 2018).

Consider the following ARMA(1,0) model:

$$Y_t = \phi Y_{t-1} + e_t \,,$$

i.e., an AR(1). The likelihood for this model is given by:

$$L(\phi, \mu, \sigma_e^2) = (2\pi\sigma_e^2)^{-\frac{T}{2}}(1-\phi^2)^{1/2} \exp\left[-\frac{1}{2\sigma_e^2} S(\phi, \mu)\right],$$

where σ_e^2 is the variance of e_t, and $S(\phi, \mu)$ is called the unconditional sum-of-squares function:

$$S(\phi, \mu) = \sum_{t=2}^{T}[(Y_t - \mu) - \phi(Y_{t-1} - \mu)]^2 + (1-\phi^2)(Y_1 - \mu) \,,$$

where μ is the unconditional mean of Y_t (which is zero in this particular case). The likelihood for an ARMA(p,q) is developed in a similar fashion (Cryer and Chan, 2008).

2.7 Forecasting time series

In this thesis, the purpose of these models is to forecast future values for Y_t based on the history of the time series $(Y_1, Y_2, \ldots, Y_t)$. These future values are denoted as Y_{t+h}, where h is the forecast horizon. For an ARMA process, Y_{t+h} can be given by:

$$Y_{t+h} = \begin{cases} \phi^h Y_t - \phi^{h-1}\theta e_t + e_{t+h} + \sum_{j=0}^{h-2} \phi^j(\phi - \theta) e_{t+(h-1)-j} & if\ h > 1 \,, \\ \phi Y_t - \theta e_t + e_{t+1} & if\ h = 1 \,. \end{cases}$$

In this thesis, one-step-ahead forecasts, i.e., $h = 1$, is used. A method of forecasting is by taking conditional expectation, which can be applied for $h = 1$ in the following matter:

$$E[(Y_{t+1}|I_t)] = E(\phi Y_t|I_t) - E(\theta e_t|I_t) + E(e_{t+1}|I_t) = \phi y_t - \theta e_t \,,$$

where I_t is all the information up until time point t $(Y_1, Y_2, \ldots, Y_t)$ and y_t is the actual value of the Nasdaq-100 index at time point t. The forecast calculations will be denoted as $\hat{Y}_{t+h}$ where the right-hand side of the conditional expectation will be replaced by consistent estimates. For

example, for $h = 1$, we have that $\hat{Y}_{t+1} = y_t - \hat{\theta}\hat{e}_t$, where $\hat{e}_t$ can be found from recursion (Hyndman and Athanasopoulos, 2018).

2.8 Neural networks

The initial idea for neural networks came from trying to imitate the human brain, similar to how flying came from watching birds. However, neural networks have over time come to differ a lot from how the brain works, in the same way that planes differ from birds. The neural network was introduced in the early 40's, but has become more and more used in recent years with technology advancing and data collection becoming much easier. Neural networks consist of neurons that work together by sending different signals to each other depending on the input they receive (Géron, 2018).

Training neural networks is done using different hyperparameters that together form the network. Training refers to the process where the model learns from data to make accurate predictions. Hyperparameters are parameters whose values are used to control the learning process. Figure 1 shows a typical neural network and all its components as well as how everything is connected. In Figure 1, it is shown that the first thing that happens is that input X which possibly is a vector of n values, $\boldsymbol{X} = (X_1 \dots X_n)^{\mathrm{T}}$, goes into the layers. Input X could in theory be any kind of data but in the case of time series analysis it can be considered as the observations from the time series used to make predictions.[1]

[1] Here $(\bullet)^{\mathrm{T}}$ denotes the matrix transpose.

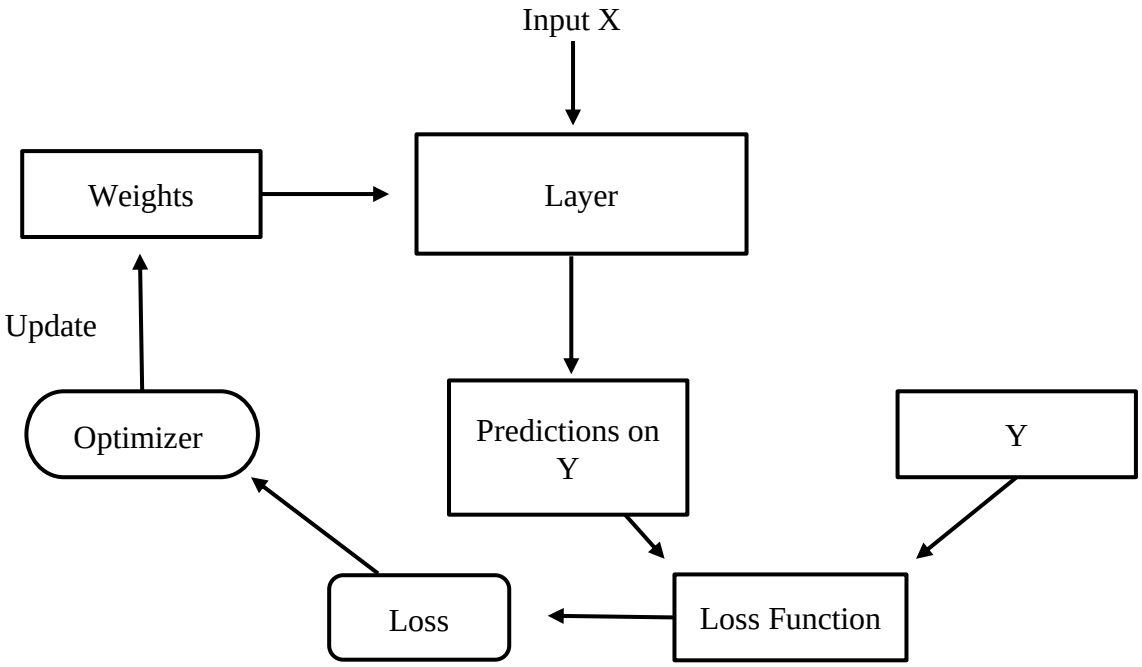

Figure 1. Neural network and its components and their relationship.

The layers are what, when combined, makes up the network. Different types of layers are appropriate for different input formats and different types of data processing. When dealing with time series or sequence data, the input should be processed by a recurrent layer such as a LSTM or GRU layer (Chollet and Allaire, 2018).

There is no clear answer when it comes to choosing the number of layers. One can produce decent results for many problems with just one or two layers. One layer can model very complex functions in the case that it has enough neurons. However, adding more layers can in some cases improve the model's ability to generalize to new data. When dealing with very complex problems, such as speech recognition, multiple layers are required, sometimes by the hundreds. It is however recommended to start off with just one or two layers since these kinds of models are known to yield good results (Géron, 2018).

It is within the layer that the calculations are done. The calculations are done by what is called *neurons,* or *units,* which are what the layers are built by. A neuron activates its output when a certain number of inputs are active. As with the layers, there is no exact answer to the question of how many neurons that should be used in an RNN. The calculations are done by multiplying each input with some weights as the linear combination:

$$u = w_1X_1 + w_2X_2 + \ldots + w_nX_n + b\,,$$

$$\hat{Y} = f(u) = f(w_1X_1 + w_2X_2 + \ldots + w_nX_n + b)\,,$$

where w_j $(j = 1, 2, \ldots, n)$ are weights, X_j are the inputs and f is an *activation function* which in turn produces an output or prediction $\hat{Y}$ and b is the bias for the estimations (Michelucci, 2018; Géron, 2018).

There are two activation functions that are used in this thesis: the *sigmoid function* and the *hyperbolic tangent function*. The sigmoid function is used on the input where the function takes on a value between 0 and 1. If the sigmoid function is 1, then the input is kept in its entirety, and if it is 0, then it is not used at all (Fathi and Maleki Shoja, 2018). The sigmoid function is denoted by σ and is given by:

$$\sigma(u) = \frac{1}{1+e^u}\,.$$

The hyperbolic tangent function is similar to the sigmoid function, but it can take on a value between -1 and 1. The hyperbolic tangent is denoted by $tanh$ and is given by (Michelucci, 2018):

$$tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}\,.$$

Inside the neurons is where the calculations are performed using input X. The result is the prediction $\hat{Y}$, as seen in Figure 1. This prediction is then compared to the actual values, Y, using a *loss function*. The loss function can be described as the quantity that is minimized during training and is used as a measure of success of the predictions. There are several available loss functions, and it is important to choose the right one for the task at hand, otherwise the network might end up doing undesirable things. A commonly used loss function in cases with time series analysis is the quadratic loss function, relating to the mean squared error, MSE:

$$MSE = \frac{1}{R}\sum_{t=1}^{R}(Y_t - \hat{Y}_t)^2\,,$$

where R is the number of observations in the training data and is defined in Section 3. From the loss function a loss score is in turn produced which goes into the *optimizer*, see Figure 1. An optimizer is used to minimize the loss function and thereby determines how the network learns. The optimizer uses the loss function to update the different weights in the network for better predictions. Weights refers to the parameters inside the neurons that transform the data (Chollet and Allaire, 2018).

There are many different optimizers that can be used. *Adam* is a type of optimizing algorithm that has been proven too often outperform other optimizers when using it on multi-layered neural networks. One of the advantages of using *Adam* as optimizer is that it does not require a stationary object, which is a good thing in this case since the closing prices on a stock market over time is likely to be a non-stationary process (Kingma and Ba, 2014). In short, the weights are updated by the optimizer with the aim to make better predictions and minimize the loss function.

This entire training process can also be called an *epoch*. Epochs is the number of iterations, or how many times the model is trained. One epoch is one cycle through all the training data (Michelucci, 2018). In other words, one epoch is when the training data goes through all the parts seen in Figure 1. There does not seem to be a clear answer on the ideal number of epochs. However, the number of epochs is usually determined by its effect on the loss function.

Predicting the future using neural networks is done using what is called a *Recurrent Neural Network*, RNN. RNN's can be used to analyze time series data and predict how the time series will develop. The RNN's have many uses, such as analyzing data from a stock and telling whether you should buy or sell the stock, or improving the safety of self-driving cars by predicting the trajectory of the surroundings. RNN's are quite similar in design to a normal neural network, but with an added connection that is pointing backwards. This means that each neuron receives input Y_t which is the most recent value, as well as its own output from the previous timestep, which is denoted as h_{t-1}, and called a hidden state. This can be described as the memory that the network has from the information from the previous timestep (Géron, 2018).

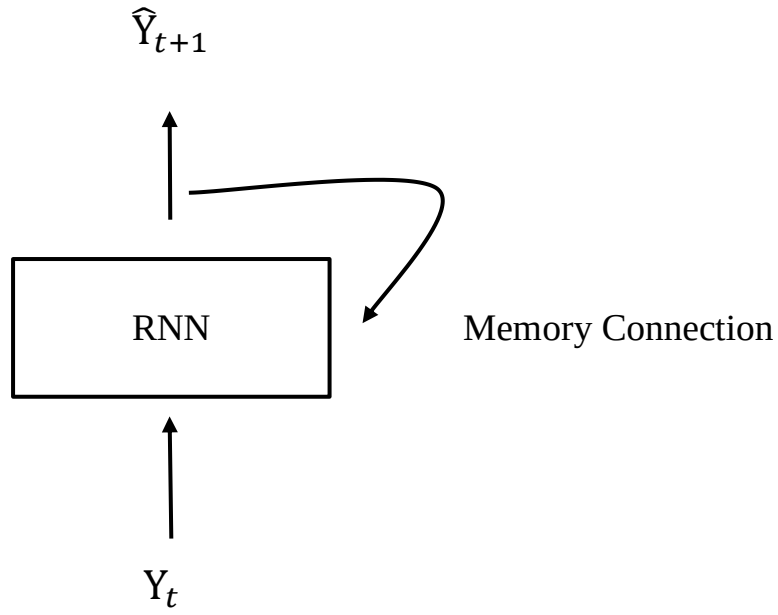

Figure 2. A Recurrent Neural Network.

A simple version of an RNN with only one neuron can be illustrated as in Figure 2, which shows how a recurrent neuron sends back its own output to itself. At every time step in an RNN each neuron receives information from the previous time step as well as the input. Figure 3 visualizes a recurrent neuron over time.

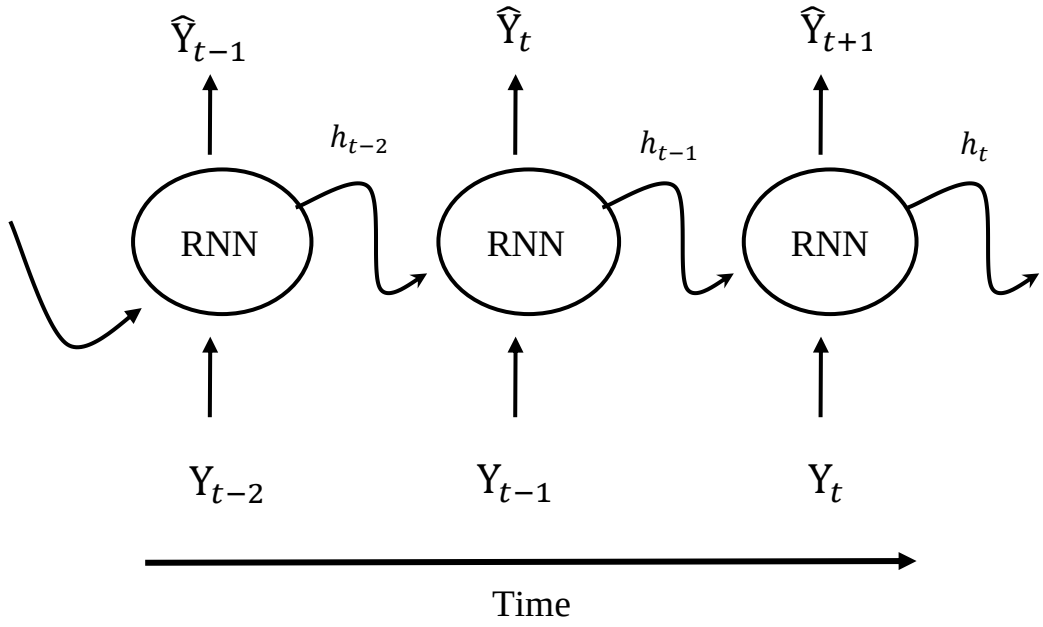

Figure 3. Recurrent Neural Network through time.

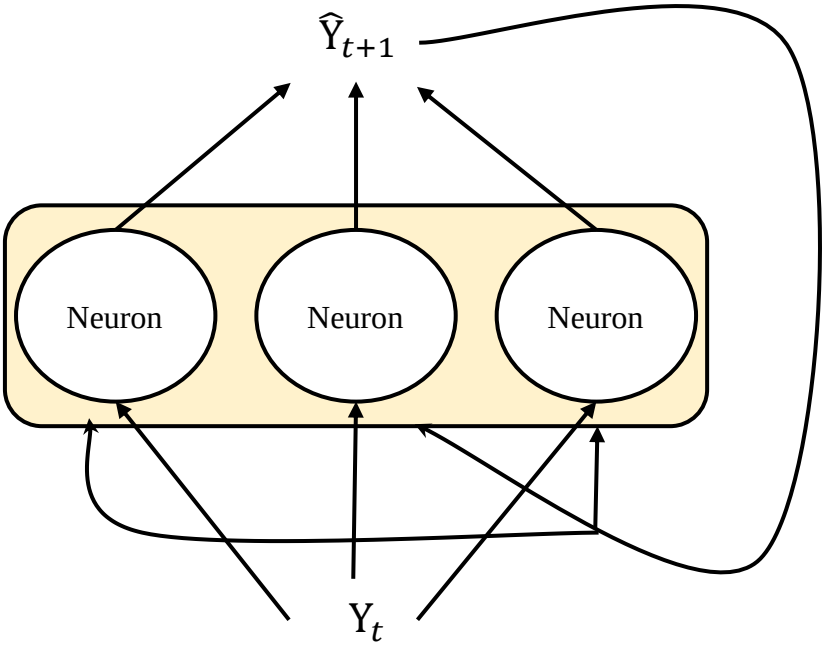

Figure 4. A layer of recurrent neurons.

With multiple neurons, a layer can be created as seen in Figure 4. Each neuron is affected by two different set of weights that affects how the neuron handles the input it receives. One weight affects the input Y_t and another set of weights affects the hidden state from previous time point, which for a layer can be defined as the vector:

$$\boldsymbol{h}_{t-1} = \begin{pmatrix} h_{t-1,1} \\ \vdots \\ h_{t-1,P} \end{pmatrix},$$

where P is the number of neurons in the previous layer. The weights for each recurrent neuron $(i = 1,2,\dots,P)$ can be denoted as a row vector $\boldsymbol{w_i} = \begin{pmatrix} w_{i,1} & \dots & w_{i,P} \end{pmatrix}$ for $\boldsymbol{h}_{t-1}$. Additionally, we let the scalar v denote the weight for the input Y_t. If we then consider all the neurons in a layer, the weights can be put into two weight matrices $\boldsymbol{W}$ and $\boldsymbol{V}$, that can be written as:

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{w_1} \\ \vdots \\ \boldsymbol{w_K} \end{pmatrix} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,P} \\ \vdots & \ddots & \vdots \\ w_{K,1} & \cdots & w_{K,P} \end{pmatrix},$$

$$\boldsymbol{V} = \begin{pmatrix} v_1 & \cdots & v_K \end{pmatrix}^{\mathrm{T}},$$

where K is the number of neurons in the current layer.

A problem with a simple RNN is that when more and more layers are added, the network becomes difficult to train. This is known as the Vanishing Gradient Problem. Another issue is the long-term memory problem. When the simple RNN trains over long sequences, the first input gradually fades away. Information gets lost after each timestep and in the end no trace of the first input is left. To counter this problem various types of long-term memory model solutions have been introduced. Two popular solutions are the LSTM model and the GRU model (Géron, 2018; Chollet and Allaire, 2018).

GRU is a type of RNN that has become popular in recent years. This model was first introduced by Cho et al. (2014). The GRU model was developed as an extension of the LSTM model, with the advantage of not requiring as much computational burden. Studies have shown that GRU models get similar performances, in terms of prediction errors, as LSTM but with less computations, making them easier to train without any clear disadvantage compared to LSTM (Géron, 2018; Wang et al., 2020).

GRU has two different gates: a reset gate and an update gate. The reset gate is where the short-term memory of the network is. The reset gate, $\boldsymbol{r}_t$, looks as follows:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r * \boldsymbol{h}_{t-1} + \boldsymbol{V}_r * Y_t + \boldsymbol{b}_r)\,, \tag{2}$$

where σ represents the logistic sigmoid function (see above) and $\boldsymbol{b}_r$ is the bias vector for the reset gate (Fathi and Maleki Shoja, 2018). The weight matrices $\boldsymbol{W}_r$ and $\boldsymbol{V}_r$ are what decides how the information that goes into the GRU model is processed. For mathematical clarity, we can decompose $\boldsymbol{r}_t$ as follows:

$$\boldsymbol{r}_t = (r_{1,t} \quad \cdots \quad r_{t,K})^{\mathrm{T}} = \sigma\begin{bmatrix} \boldsymbol{w}_{r,1}\boldsymbol{h}_{t-1} + v_{r,1}Y_t + b_{r,1} \\ \vdots \\ \boldsymbol{w}_{r,K}\boldsymbol{h}_{t-1} + v_{r,K}Y_t + b_{r,K} \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{w}_{r,1}\boldsymbol{h}_{t-1} + v_{r,1}Y_t + b_{r,1}) \\ \vdots \\ \sigma(\boldsymbol{w}_{r,K}\boldsymbol{h}_{t-1} + v_{r,K}Y_t + b_{r,K}) \end{bmatrix}.$$

The update gate is where the long-term memory is and looks as follows:

$$\boldsymbol{z}_t = \sigma(\boldsymbol{W}_z * \boldsymbol{h}_{t-1} + \boldsymbol{V}_z * Y_t + \boldsymbol{b}_z)\,.$$

The $\boldsymbol{W}_z$ and $\boldsymbol{V}_z$ represent weight matrices that are learned for the update gate, and $\boldsymbol{b}_z$ is the bias vector for the update gate. Again, σ represents the logistic sigmoid function. The update gate uses the same matrix algebra that is applied on the reset gate; see decomposition in relation to Equation (2).

Now, with the help of the two gates, the GRU process can be described. The process follows two steps. First the candidate hidden state is generated, which is denoted $\widehat{\boldsymbol{h}}_t$. The candidate hidden state takes both the hidden state from the previous time point, $\boldsymbol{h}_{t-1}$, and the input, Y_t, into account, and also uses the output from the reset gate, $\boldsymbol{r}_t$. This then leads to the following function for the candidate hidden state:

$$\widehat{\boldsymbol{h}}_t = tanh(\boldsymbol{W}_{\widehat{h}}\boldsymbol{h}_{t-1} \odot \boldsymbol{r}_t + \boldsymbol{V}_{\widehat{h}}Y_t + \boldsymbol{b}_{\widehat{h}}) \,,$$

where $tanh$ is the hyperbolic tangent function (see above) and $\odot$ is the element-wise multiplication, known as the Hadamard product. The $\boldsymbol{W}_{\widehat{h}}$ and $\boldsymbol{V}_{\widehat{h}}$ represent weight matrices and $\boldsymbol{b}_{\widehat{h}}$ is the bias vector for the candidate hidden state. In this case we get the following decomposition:

$$\widehat{\boldsymbol{h}}_t = \begin{pmatrix} \hat{h}_{t,1} & \cdots & \hat{h}_{t,K} \end{pmatrix}^{\mathrm{T}} = tanh\begin{bmatrix} \boldsymbol{w}_{\widehat{h},1}\boldsymbol{h}_{t-1,1}r_{t,1} + v_{\widehat{h},1}Y_t + b_{\widehat{h},1} \\ \vdots \\ \boldsymbol{w}_{\widehat{h},K}\boldsymbol{h}_{t-1,K}r_{t,K} + v_{\widehat{h},K}Y_t + b_{\widehat{h},K} \end{bmatrix}.$$

With the candidate hidden state, the hidden state, $\boldsymbol{h}_t$, can then be calculated. It is also here in the hidden state that the update gate is used. The function for the current hidden state looks as follows:

$$\boldsymbol{h}_t = \boldsymbol{z}_t \odot \widehat{\boldsymbol{h}}_t + (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} \,,$$

$$\boldsymbol{h}_t = \begin{pmatrix} h_{t,1} & \cdots & h_{t,K} \end{pmatrix}^{\mathrm{T}} = \begin{bmatrix} z_{t,1} * \hat{h}_{t,1} + (1 - z_{t,1}) * h_{t-1,1} \\ \vdots \\ z_{t,K} * \hat{h}_{t,K} + (1 - z_{t,K}) * h_{t-1,K} \end{bmatrix}.$$

To the get the output from the GRU process, the current hidden state is put in an output layer which only has one neuron:

$$\hat{Y}_{t+1} = \boldsymbol{w}_{\hat{Y}}\boldsymbol{h}_t + b_{\hat{Y}}\,, \tag{3}$$

$$\hat{Y}_{t+1} = (w_{\hat{Y},1} \quad \cdots \quad w_{\hat{Y},K})\begin{pmatrix} h_{t,1} \\ \vdots \\ h_{t,K} \end{pmatrix} + b_{\hat{Y}} = w_{\hat{Y},1}h_{t,1} + \cdots + w_{\hat{Y},K}h_{t,K} + b_{\hat{Y}}\,.$$

It is by using the current hidden state that the new prediction is generated for timepoint $t + 1$, $\hat{Y}_{t+1}$. (Cho et al. 2014; Hewamalage et.al, 2021). In Figure 5, the GRU process for one layer is visualized, where multiplications and addition are indicated by circles.

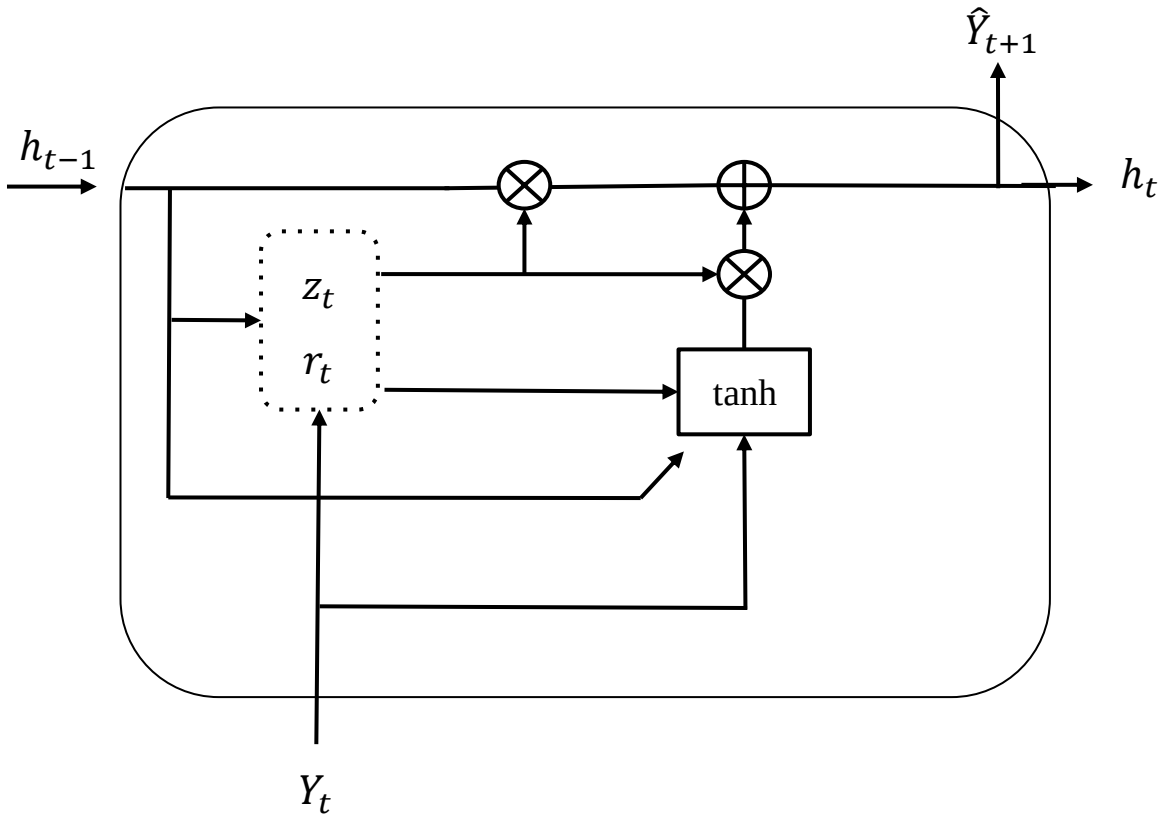

Figure 5. Illustration of the GRU process.

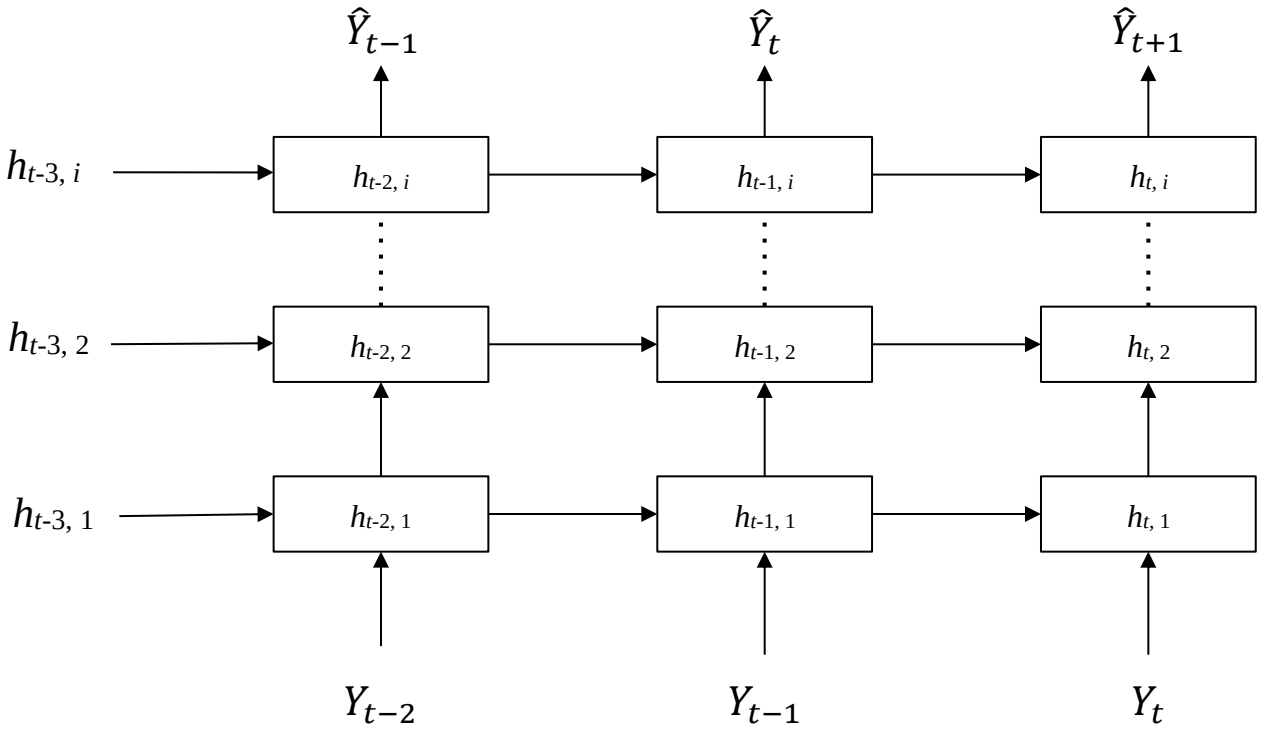

Figure 6. Illustration of a multi layered GRU process.

For multiple layers the GRU process is repeated by sending the output from the GRU layers into a new layer. The new layer is identical to the previous layer, the only difference is that the input it receives has changed through the first layer. This can be seen in Figure 6, which shows the process of the GRU with multiple layers, where i denotes number of layers (Hewamalage et.al, 2021). The last layer will always be an output layer with one neuron as seen in Equation (3).

The RNN is built by using many hyperparameters that need to be adjusted to receive the optimal results. For the GRU model the hyperparameters that are used are layers, neurons, optimizer, loss function and epochs (see above). Besides these hyperparameters there are some more parts that go into the GRU model. Those are the training, validation and test data and regularization (Géron, 2018).

When training a neural network, it is normal to split up the data into a training set, a validation set and a test set. The model is trained on the training set and then evaluated on the validation set. Once the model is completely trained it is then used and evaluated on the test data. Using

validation data is a form of fighting overfitting. Overfitting is when a model can't make good predictions on new data (Chollet and Allaire, 2018).

Another part of the GRU model that is important is the regularization. Regularization is a way of fighting overfitting. There are several regularization techniques and one of the most effective and common techniques is dropouts. When adding dropouts to a layer, several output features will randomly "drop out", meaning they get set to zero, during training. The dropout rate is the fraction of the features that are dropped out and is usually between 0.2 and 0.5 (Chollet and Allaire, 2018).

3. Data

The data used is based on the historical adjusted closing price of the Nasdaq-100 index, which has been collected from the website *Yahoo! Finance*. The adjusted closing price is what the models will be using, since this measurement adjusts for stock splits and dividend distributions. The adjusted closing price is also what the models will predict (Yahoo Finance, 2022).[2]

The collected data has its starting point at 2017-01-03 and stretches to 2021-12-31. This gives a total of 1,259 observations, since the stock market is closed on weekends and holidays. For the GRU models, the data is split up into training data, validation data and test data. The training data is what the model is trained on or learns from, it contains data from 2017-01-03 to 2020-10-19. The validation has 239 observations, from 2020-10-20 to 2021-09-30. The test data is what the models will be forecasting on, which is the last three months of 2021, that is, from 2021-10-01 to 21-12-31, which gives us 64 one-day-ahead predictions.

In the case of the ARIMA model, the data is split up into estimation data and test data. The estimation data is used for identification of the model order and estimation of the model parameters.

Both models will forecast one day ahead using an expanding estimation window. An expanding window means that the model is fitted using all available historic data in order to make a forecast, where the window size grows as more observations are used. This means that for every day the model progresses it gains one more observation. Let R be the initial number of observations (initial window size) used to estimate the models and then forecast Y_{t+h}. That is, in terms of window sizes, the first prediction is made on the timepoint $R + h$, i.e., on Y_{R+h}. In our case this is all observations before the last three months of the data, which means that $R = 1{,}195$. When forecasting for the next time point $R + h + 1$, the model will use $R + 1$ observations to make this prediction, and so on. When making the final prediction at time point T, the model will be using $T - h$ number of observations. Let P be the number of observations in the test dataset used for forecast evaluation. The size of P is $P = T - R + 1 - h = T - R$. In our case we have that $T = 1{,}259$ and $R = 1{,}195$, and therefore that $P = 1{,}259 - 1{,}195 = 64$.

[2] *Yahoo! Finance* is an open source that is free and available for everyone to use and collect data from.

For future reference, let the predictions for a given model be denoted as:

$$\hat{Y}_{R+h}, \hat{Y}_{R+1+h}, \dots, \hat{Y}_T \ ,$$

where, in our case, $h = 1$.

In Figure 7 the Nasdaq-100 index for the estimation window is shown. The red line separates the forecast evaluation set (i.e., test data), on the right side, from the rest of the data.

Figure 7. The Nasdaq-100 Index.

4. Fitting the models

In this section we go through how our models are fitted to data, by explaining how different parameters are estimated or chosen trough rule of thumb and trial and error.

4.1 Specification of the ARIMA models

As stated in Section 3, the gathered data was split up into estimation data and test data. The expanding estimation data was used to create the models for the 64 different one-step-ahead forecasts. All the parameters are re-estimated for each forecast. The models are specified by using the *forecast* package in the statistical software R.[3]

In this package the function *auto.arima* is used to fit the ARIMA models. This function fits the most suitable model for a univariate time series by first using the KPSS test to decide the order of integration and then minimizing AIC for various sets of AR and MA orders, where the parameters (coefficients) are estimated by MLE (see Section 2). After the model has been specified, the function *forecast* is used to make predictions by estimating the conditional expectation (see Section 2) (Hyndman and Khandakar, 2008; Hyndman and Athanasopoulos, 2018).

For the KPSS test, the *auto.arima* function tests the data for the null hypothesis of no unit root on a 5 percent significance level. If the test is significant (i.e., if it rejects the null hypothesis and concludes that there is a unit root), then the test is performed again but on the differenced data. This procedure continues until the test is insignificant (i.e., concluding that there is no unit root at that level). Once d is decided, the *auto.arima* function uses AIC to select the final model, whereby the parameters of the associated ARMA models are estimated by MLE for a pre-specified set of different values of the AR and MA orders p and q. Here, a restriction is set that the sum of p and q cannot exceed 5.

4.2 GRU specification

In this section, the specification of the GRU model is presented. Keras application is used to build the GRU model. Keras is a deep learning framework that can train almost any deep

[3] The package can be found and downloaded from CRAN where all the functions are detailed: https://cran.r-project.org/web/packages/forecast/forecast.pdf.

learning model. Keras is used by more than 150 000 users, such as researchers and engineers, startups and large companies such as Google, Netflix, Uber and more (Chollet and Allaire, 2018). Since Keras is written in the programing language Python, the software Google Collaboratory will be used to build the model.[4]

As previously stated, there are no exact answers to the specification of many of the hyperparameters. Because of this, the specification of the different hyperparameters will be based on a mixture of previous research, rule of thumb as well as trial and error (see Section 2.8). We will create 8 different GRU models, where the three hyperparameters layers, neurons and dropout all will be used with 2 different values each.

To decide the number of layers, there are as stated in Section 2.8 no exact rules. However, in many cases it is enough with one or two layers. Therefore, the results for the GRU models will be produced by using 2 GRU layers and then increasing it to 3 GRU layers to see if more layers improve the results. All the models include an output layer that has only one neuron, but what is interesting is the number of GRU layers. Different numbers of neurons will also be used, based on earlier research as well as trial and error, to see which number of neurons that generate the most accurate predictions. We use 50 neurons per layer to get the first results and then we change to 100 neurons per layer to see how this affects the results. As described in Section 2, the dropout is recommended to be between 0.2 and 0.5. For this thesis, the dropout rate will be set to 0.2 in 4 of the models. For the other 4 models there will be no dropout, to see how this affects the results. Our 8 models can be summarized as follows:

GRU 1: 2 layers, 50 neurons per layer and no dropout.
GRU 2: 2 layers, 100 neurons per layer and no dropout.
GRU 3: 3 layers, 50 neurons per layer and no dropout.
GRU 4: 3 layers, 100 neurons per layer and no dropout.
GRU 5: 3 layers, 50 neurons per layer and 0.2 dropout.
GRU 6: 3 layers, 100 neurons per layer and 0.2 dropout.
GRU 7: 2 layers, 50 neurons per layer and 0.2 dropout.
GRU 8: 2 layers, 100 neurons per layer and 0.2 dropout.

[4] More information on Google Collaboratory can be found at: https://colab.research.google.com/?utm_source=scs-index.

For all GRU models, the *Adam* optimizer is used. *Adam* was chosen as optimizer because it has proven to be a good optimizer when (i) using multi-layered neural networks, which a GRU model is, and (ii) modelling non-stationary time series data, which would typically be the case for stock market indexes. With this in mind, *Adam* is a suitable optimizer. When it comes to the loss function for the GRU models, we use MSE.

When training the model, 50 epochs are used to minimize the MSE. This is because the loss function does not get any lower after 50 epochs. Figure 8 shows how the MSE loss value decreases over 100 epochs. Even fewer epochs could have been used without any negative effect on the loss value. This is because the loss does not seem to decrease in any significant way after the first 10 epochs. An optimal choice of number of epochs is a mixture of a low loss and a low number of epochs to minimize the computational burden.

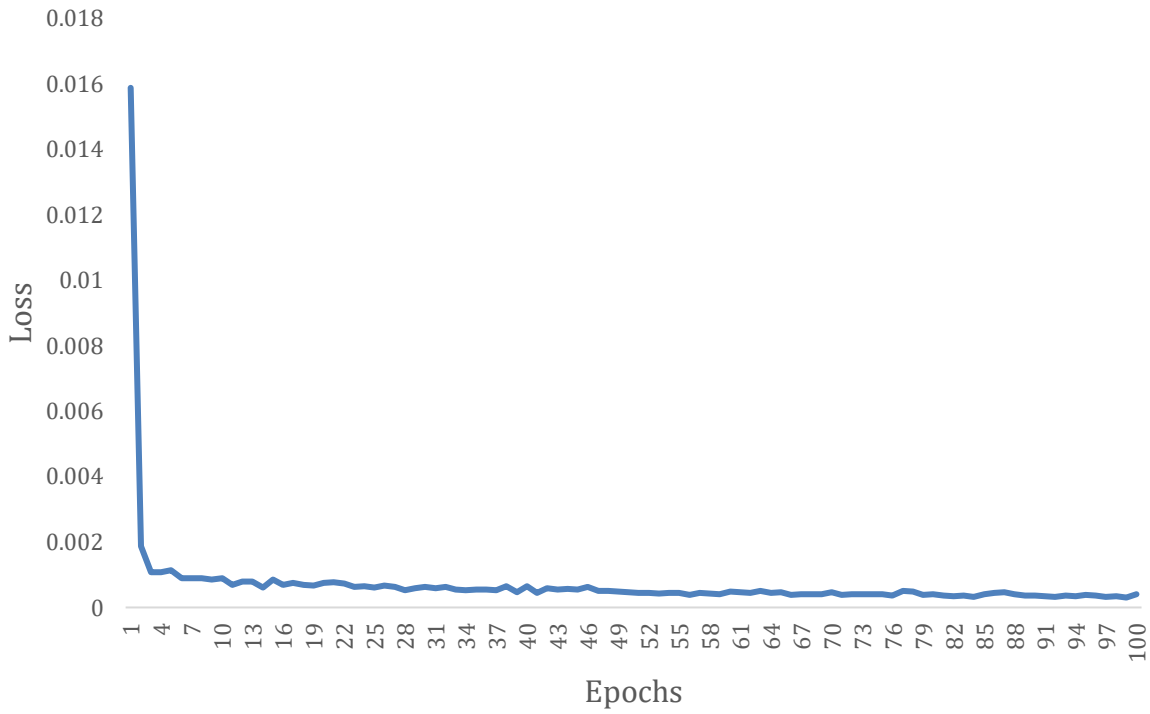

Figure 8. MSE loss function for the training data over 100 epochs.

5. Forecast evaluation

This section will present how the forecasts will be evaluated. We present our choice of error measurements, and also go through the Diebold-Mariano test, which can be used to indicate if there is a significant difference in the accuracy between two models.

Once the models have been created and results from them have been produced, they can be compared using an error measurement to see which of them performs the best. For evaluation of the models, the root mean squared error, RMSE, is used. RMSE is one of the most common error measurements in statistics. It works well in this case since both models have errors in the same units, meaning that the output from the predictions does not need to be standardized. However, it is important to consider that the difference between the prediction and the actual value can be both positive and negative, which this measurement does by squaring the error (Reich et al., 2016).

Consider the sequence of one-step-ahead forecasts from a given model as outlined in Section 3, $\hat{Y}_{t+1}(t = R, R+1, \dots, T-1)$, where $R = 1{,}195$ and $P = 64$ are the sizes of the initial estimation window and the evaluation window, respectively. The sequence of forecast errors can be defined as:

$$\epsilon_{t+1} = Y_{t+1} - \hat{Y}_{t+1}\ (t = R, R+1, \dots, T-1)\ .$$

The RMSE for this sequence is:

$$RMSE(\epsilon_t) = \sqrt{MSE(\epsilon_t)} = \sqrt{\frac{1}{P}\sum_{S=R+1}^{T} \epsilon_s^2}\ .$$

Note that, since $h(c) = \sqrt{c}$ is a monotone function of an input c, it makes no difference if we rank forecast precision by MSE or RMSE.

To test whether MSE (or RMSE) from two competing models are significantly different from one another, the test by Diebold and Mariano (1995) can be used. Suppose that two time series models that produce one-step-ahead forecasts over the evaluation window, $t = R+1, R+2, \dots, T$, and let $\epsilon_{1,t}$ and $\epsilon_{2,t}$ be the forecast errors for these forecasts.

Define the difference $d_t = g(\epsilon_{1,t}) - g(\epsilon_{2,t})$ $(t = R+1, R+2, \dots, T)$, where $g(\cdot)$ is some loss function. If we assign the quadratic loss function, so that $d_t = \epsilon_{1,t}^2 - \epsilon_{2,t}^2$, then the time series mean of d_t is the difference in MSE, $\bar{d} = P^{-1}\sum_{t=R+1}^{T}(\epsilon_{1,t}^2 - \epsilon_{2,t}^2) = MSE\big(\epsilon_{1,t}\big) - MSE(\epsilon_{2,t})$. If the models have the same predictive accuracy in terms of MSE, then $E(d_t) = 0$. Therefore, testing for a difference in predictive accuracy can be done using the following null and alternative hypotheses:

$$H_0: E(d_t) = 0\,,$$
$$H_1: E(d_t) \neq 0\,.$$

If d_t is a covariance stationary process, then we have that $\bar{d} \xrightarrow{p} E(d_t)$, as $P, T \to \infty$, by the law of large numbers. Therefore, the following statistic to test the above hypotheses can be applied:

$$DM = \frac{\bar{d}}{\sigma(\bar{d})}\,,$$

where $\sigma(\bar{d})$ is the (constant) standard deviation of $\bar{d}$. From the central limit theorem, the DM tends in distribution to a standard normal distribution as $P, T \to \infty$.

Note that $\bar{d}$ is an aggregate of a stochastic process (since the mean involves a sum over time), and that the variance $\sigma^2(\bar{d})$ entails possibly non-zero autocovariances. Due to this dependence, the variance of the mean (the variance of the sum) is *not* equal to a sum of variances,

$$\sigma^2\big(\bar{d}\big) = Var(P^{-1}\textstyle\sum_{t=R+1}^{T} d_t) \neq P^{-1}\sum_{t=R+1}^{T} Var(d_t)\,.$$

Instead, it can be shown that, for large P, we have that

$$\sqrt{P}\left(\bar{d} - E\big(\bar{d}\big)\right) \xrightarrow{d} N\big(0, \textstyle\sum_{j=-\infty}^{\infty}\gamma_d(j)\big)\,,$$

where $\sum_{j=-\infty}^{\infty}\gamma_d(j)$ is the "long-run" variance,

$$\textstyle\sum_{j=-\infty}^{\infty}\gamma_d(j) = \gamma_d(0) + 2\sum_{j=1}^{\infty}\gamma_d(j)\,,$$

where $\gamma_d(0) = Var(d_t)$ is the contemporary variance and $\gamma_d(j) = Cov(d_t, d_{t-j})$ is the autocovariance at lag j (compare with definitions in Section 2).

We can estimate $\sigma^2(\bar{d})$ consistently by a weighted sum of the available sample autocovariances,

$$\hat{\sigma}^2(\bar{d}) = \sum_{j=-(P-1)}^{P-1} K\hat{\gamma}_d(j) ,$$

where K is some truncation weight (that is either 1 or 0) and

$$\hat{\gamma}_d(j) = \frac{1}{P}\sum_{t=|j|+1}^{P}(d_t - \bar{d})(d_{t-|j|} - \bar{d}) .$$

The test will be performed at a five percent significance level, meaning that the null hypothesis will be rejected if the p-value of the test is lower than 0.05 or

$$|DM| > z_{\alpha/2} ,$$

where $z_{\alpha/2}$ is the critical value for a normal distribution for a given significance level, α. The test is included in the *forecast* package (see footnote 2). The following adjusted test statistic can be used for small samples:

$$DM^* = \left[\frac{P+1-2h+P^{-1}h(h-1)}{P}\right]^{1/2} DM ,$$

where h is the prediction horizon, in our case $h = 1$. The adjusted test statistic uses the critical values from a t-distrubution with $P - 1$ degrees of freedom. This means that the null is rejected if (Harvey et al., 1997):

$$|DM^*| > t_{a/2,(P-1)} .$$

6. Results and analysis

In this section, the results are presented using time series graphs and tables. The predictions and prediction errors will be presented first, and lastly the result from the Diebold-Mariano test.

In Figure 9 it seems like the one-step-ahead forecasts from the ARIMA model follows the Nasdaq-100 index very well. The predictions are especially accurate for the middle part of the graph where it at times even can be difficult to differentiate the ARIMA predictions from the Nasdaq-100 index. It is only in the beginning and the end of the period that there seems to be a bit of a difference between the ARIMA predictions and the actual values. This is more clearly visualized in Figure 10, which shows the difference between the actual values for Nasdaq-100 and the predictions made by the ARIMA model, i.e., the forecast errors.

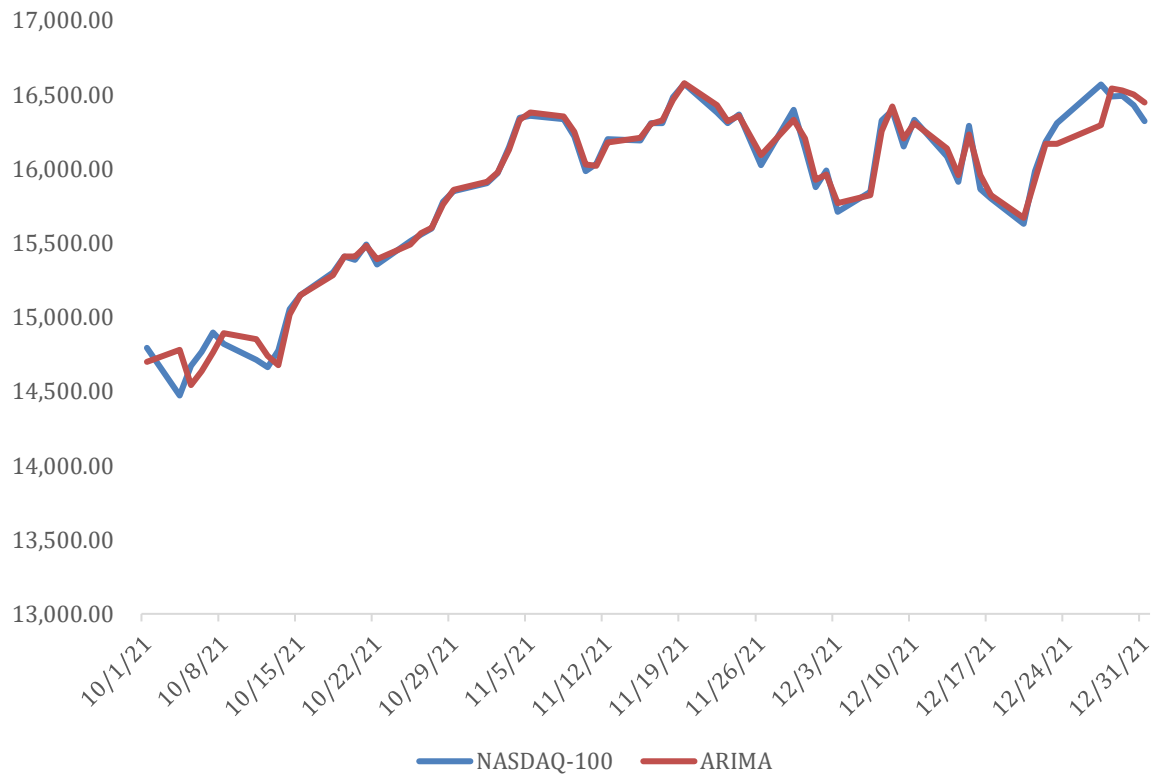

Figure 9. The Nasaq-100 index and ARIMA predictions.

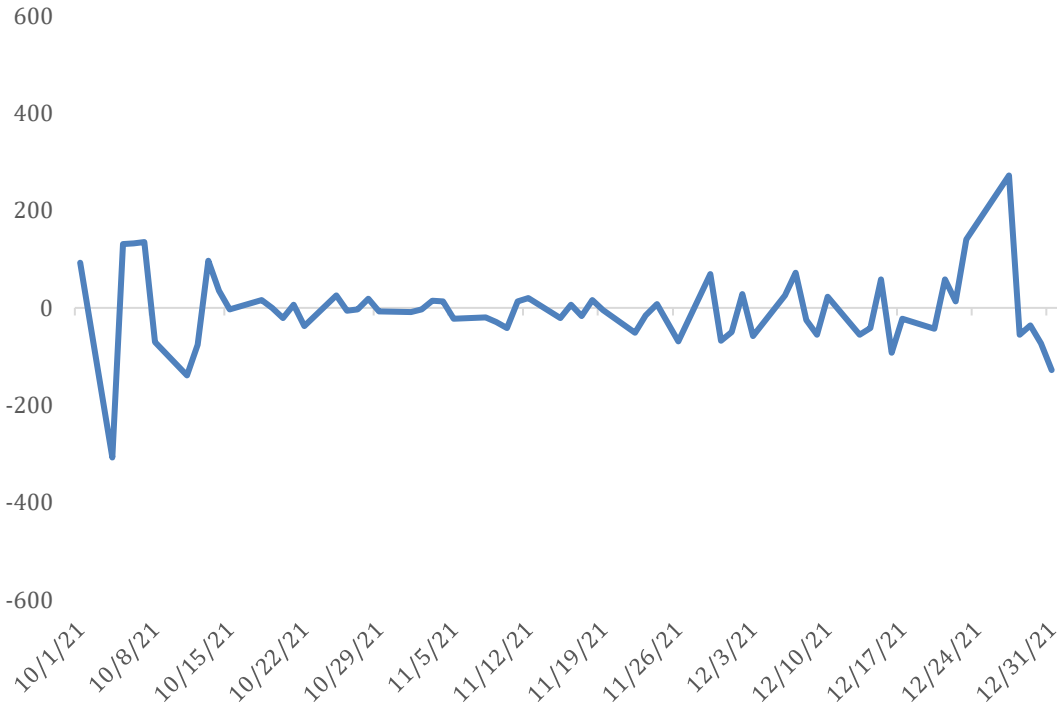

Figure 10. Difference between the actual values and the ARIMA predictions.

In Figure 11 the most accurate of our GRU models is presented, which was GRU 2. The figure shows that the prediction seems to be almost constantly a lagged version of the actual values of the Nasdaq-100 index. The prediction that has been made looks to always be one or two days behind the actual values. This means that the model rarely is way off the actual values, but only at a few times does it predict the correct value. Because of this, the prediction error can become quite high (see Figure 12), especially if compared to the ARIMA model, for which the predictions are closer to the correct values almost constantly throughout the time period. The predictions and prediction errors for the remaining GRU models are shown in the Appendix.

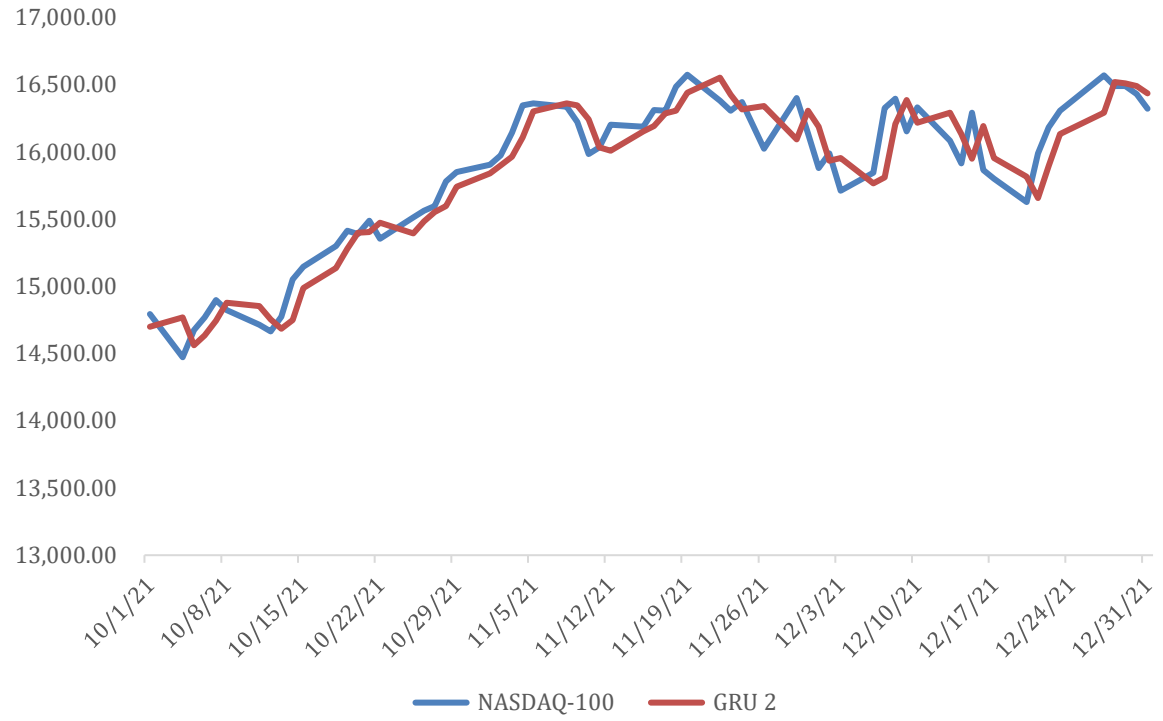

Figure 11. The Nasdaq-100 index and GRU predictions and with 2 layers, 100 neurons per layer and without dropout (GRU 2).

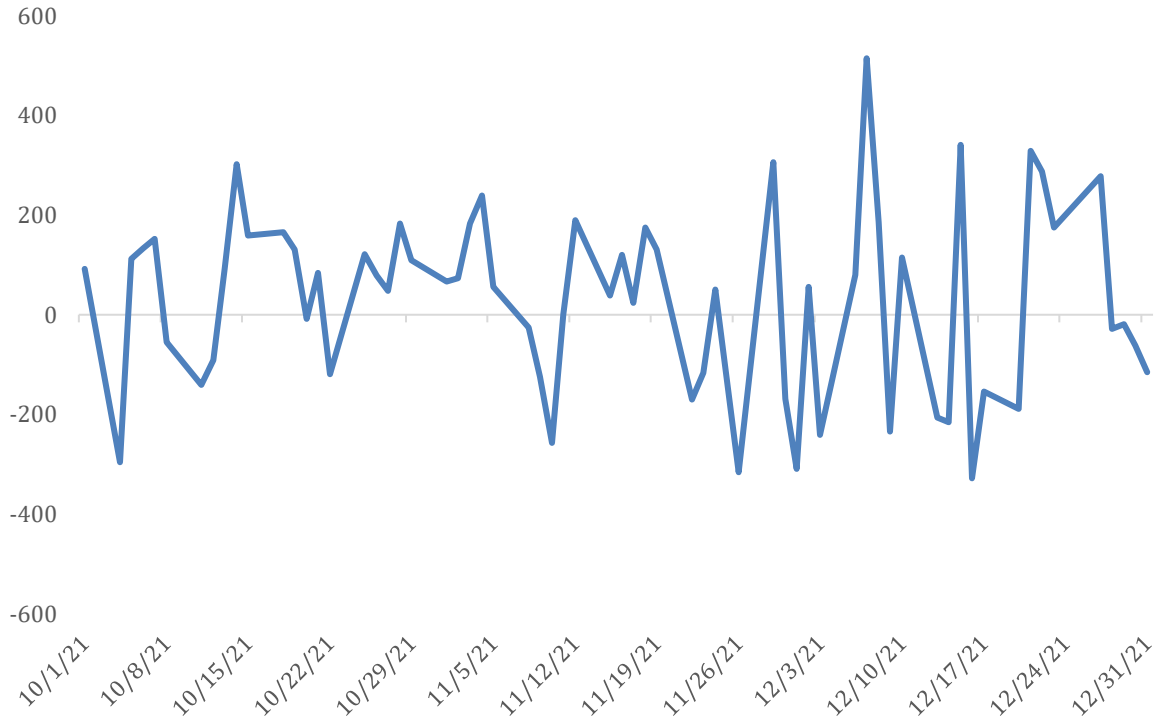

Figure 12. Difference between actual values and the GRU 2 model predictions.

Table 2. RMSE for the forecasts.

Model	RMSE
ARIMA	77.31
GRU 1	191.37
GRU 2	186.07
GRU 3	196.81
GRU 4	200.17
GRU 5	194.83
GRU 6	189.67
GRU 7	207.26
GRU 8	197.89

Table 2 shows the RMSE for all our models. The ARIMA model has the lowest RMSE values while all the GRU models have a RMSE value that is approximately between two and three times as high. For the GRU models, GRU 2 is the model that performs the best followed by the GRU 6 model. The GRU 2 model had 2 layers, 100 neurons per layer and no dropout. The GRU 6 model had 3 layers, 100 neurons per layer and 0.2 in dropout. The GRU model that performed the worst, by some margin, was the GRU 7 model.

By comparing the graphs for all the GRU models that have been created (See Figure 11 and corresponding figures in the Appendix), they all seem to follow a similar pattern: a lagged version of the Nasdaq-100 index rather than a prediction of future values. This could mean that GRU models are bad at predicting a stock market index. It could also mean that an error was made in the coding or setup of the models.

The Diebold-Mariano test (see Section 5) was performed to investigate if there is a significant difference in prediction accuracy for each of the forecasts from the 8 GRU models compared to forecasts from the ARIMA model. The results are shown in Table 3. The results indicate that the RMSE of the forecast from the ARIMA model is significantly smaller than the RMSE of the forecast from each of the GRU models. This was expected, since there are quite big differences in RMSE of the forecasts from the GRU models compared to the forecasts from the ARIMA model.

Table 3. Results from Diebold-Mariano tests.

Model comparison	Tests statistic (DM)	P-value
ARIMA and GRU 1	-5.80	< 0.001
ARIMA and GRU 2	-5.30	< 0.001
ARIMA and GRU 3	-5.90	< 0.001
ARIMA and GRU 4	-5.17	< 0.001
ARIMA and GRU 5	-5.28	< 0.001
ARIMA and GRU 6	-4.96	< 0.001
ARIMA and GRU 7	-4.92	< 0.001
ARIMA and GRU 8	-4.93	< 0.001

Based on these results, a GRU model does not seem to be a very good tool for predicting the development of the Nasdaq-100 index. This is because not any of the GRU models that were created was able to make predictions that outperformed the simple ARIMA model. Instead, GRU looks to be quite a bit worse based on the RMSE for the different models.

Even if these results point towards GRU not being a very good predictor in this case, it is important to keep in mind that it might be possible to create a similar GRU model that is more accurate. The hyperparameters that are used for the GRU models can be combined in several ways, and there are no exact rules for choosing the number of layers, the number of neurons and the rate of dropout. This means that the settings used for the GRU models in this thesis are not guaranteed to be the best ones. Even though some combinations were tested based on both rule of thumb as well as trial and error, there are many more combinations that were not tested. Additionally, the sizes for the different windows used for training, validation and test data can be varied in many ways, which may impact the results. Lastly, it is important to consider the forecast horizon, $h = 1$, and that different values for h may yield different results.

The results however pointed towards that changing the hyperparameters in this case had a quite low effect on the accuracy for the prediction. While both of the two most accurate GRU models had 100 neurons per layer, other models with 50 neurons differed in such a small way that it is difficult to draw any real conclusions from this. The same goes for both the number of layers and the dropout which both differed between the most accurate and second most accurate GRU

models. This further points towards the fact that it is difficult to draw conclusions for which number of hyperparameters that can produce the most accurate results.

7. Conclusions

In the beginning of this thesis the research question that was stated was: *Can a GRU model make more accurate predictions of the Nasdaq-100 index than a simple ARIMA model?* Based on the results of our research, this does not seem to be the case. Instead, the ARIMA clearly outperformed all of the different GRU models created in terms of RMSE. These results were strengthened by the Diebold-Mariano test that indicated that the differences in RMSE were significant on a 5 percent significance level.

This does, however, not mean that it is certain that an GRU model will always be less accurate than an ARIMA model. It is still possible that different combinations of hyperparameters would yield better results than our predictions. Additionally, it is possible that a GRU model could outperform an ARIMA if the sizes of the training, validation and forecast windows would be changed. For further research it would therefore be interesting to try more variations of the hyperparameters, different forecast horizons, h, and various sizes for the respective time series windows.

References

Asteriou, D., Hall, S. G. (2016). *Applied econometrics* (3rd ed.). Macmillan Education, London.

Berk, J., DeMarzo, P. (2019). *Corporate finance* (5th ed.). Pearson Education, Harlow.

Chai, T., Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? *Geoscientific Model Development*, 7, pp. 1525–1534.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., Bahdanau, D., Bengio, Y. (2014). *Learning phrase representations using RNN encoder–decoder for statistical machine translation*. Retrieved from: https://doi.org/10.48550/arXiv.1406.1078.

Chollet, F., Allaire, J. J. (2018). *Deep Learning with R* (1st ed.). Manning Publications Co., New York.

Cryer, J. D., Chan, K. (2008). *Time series analysis: With applications in R* (2nd ed.). Springer, New York.

Diebold, F. X., Mariano, R. S. (1995). Comparing predictive accuracy. *Journal of Business & Economic Statistics*, 13, pp. 134–144.

Fathi, E., Maleki Shoja, B. (2018). Deep neural networks for natural language processing. *Handbook of Statistics*, 38, pp. 229–316.

Gao, Y., Wang, R., Zhou, E. (2021). Stock prediction based on optimized LSTM and GRU Models. *Scientific Programming*, 2021.
Retrieved from: https://doi.org/10.1155/2021/4055281.

Géron, A. (2018). *Neural networks and deep learning* (1st ed.). O'reilly Media Inc. Retrieved from: https://go.exlibris.link/qkD8tqgm.

Harvey, D., Leybourne, S., Newbold, P. (1997). Testing the equality of prediction mean squared errors. *International Journal of Forecasting*, 13, pp. 281–291.

Hewamalage, H., Bergmeir, C., Bandara, K. (2021). Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37, pp. 388–427.

Hyndman, R. J., Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2nd ed.). OTexts, Melbourne.

Hyndman, R. J., Khandakar, Y. (2008). Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, pp. 1–22.

Kingma, D. P., Ba, J. (2014). *Adam: A method for stochastic optimization.* International Conference for Learning Representations, San Diego, California, USA.

Kwiatkowski, D., Phillips, P.C., Schmidt, P., Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, 54, pp. 159–178.

Michelucci, U. (2018). *Applied deep learning: A case-based approach to understanding deep neural networks* (1st ed). Apress, Berkley.

Mills, T. M. (2019). *Applied time series analysis: a practical guide to modeling and forecasting.* Elsevier Inc, London.

Nasdaq. (2022, April 5). *Nasdaq-100 Index*. Retrieved from: https://www.nasdaq.com/nasdaq-100

Ordóñez, C., Sánches Lasheras, F., Roca-Pardiñas, J., de Cos Juez, F. (2019). A hybrid ARIMA–SVM model for the study of the remaining useful life of aircraft engines. *Journal of Computational and Applied Mathematics*, 346, pp. 184–191.

Reich, N. G., Lessler, J., Sakrejda, K., Lauer, S. A., Iamsirithaworn, S., Cummings, D. A. T. (2016). Case study in evaluating time series prediction models using the relative mean absolute error. *The American Statistician*, 60, pp. 285–292.

Roondiwala, M., Patel, H., Varma, S. (2017). Predicting stock prices using LSTM. *International Journal of Science and Research,* 6, pp. 1754–1756.

Shah, D., Campbell, W., Zulkernine, F. H. (2018, Dec 10-13). *A comparative study of LSTM and DNN for stock market forecasting.* IEEE International Conference on Big Data, Seattle, Washington, USA.

Wang, R., Li, C., Fu, W., Tang, G. (2020). *Deep learning method based on Gated Recurrent Unit and Variational Mode Decomposition for short-term wind power interval prediction.* IEEE Transactions on Neural Networks and Learning Systems, 31, pp. 3814–3827.

Yahoo!finance. (2022, April 6). *Nasdaq-100*. Retrieved from: https://finance.yahoo.com/quote/%5ENDX/history?p=%5ENDX.

Appendix: Predictions and prediction errors

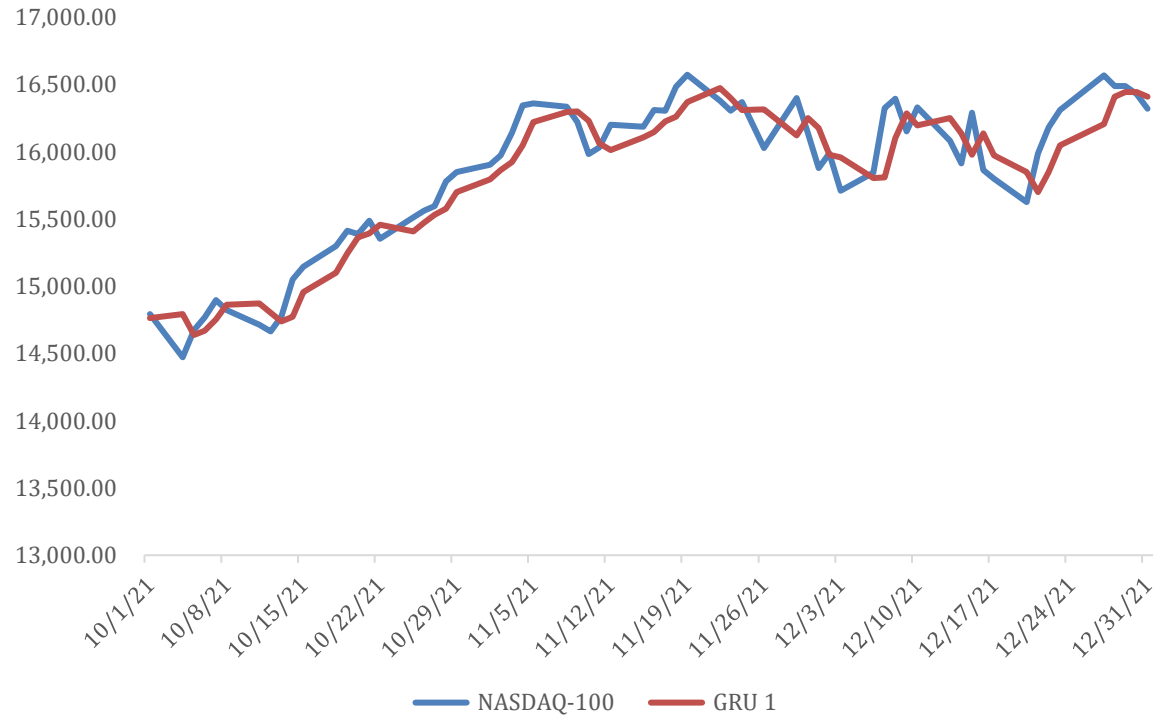

Figure A1. The Nasdaq-100 index and GRU predictions with 2 layers, 50 neurons per layer and no dropout (GRU 1).

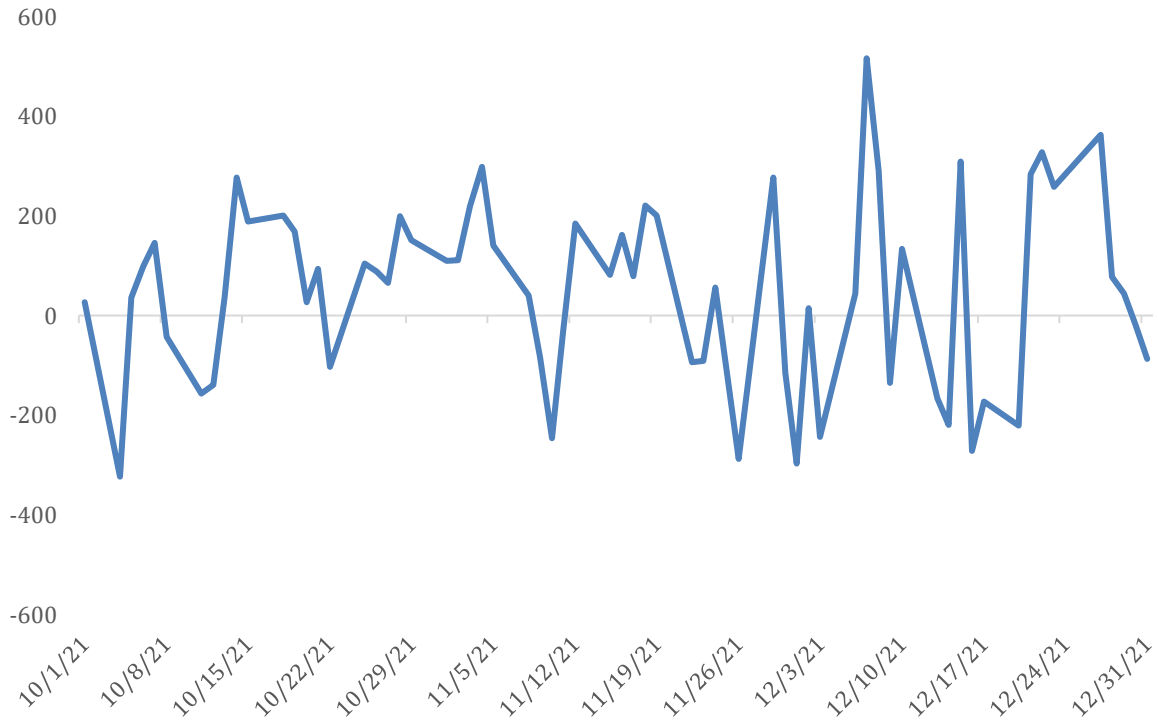

Figure A2. Difference between the actual values and the GRU 1 model predictions.

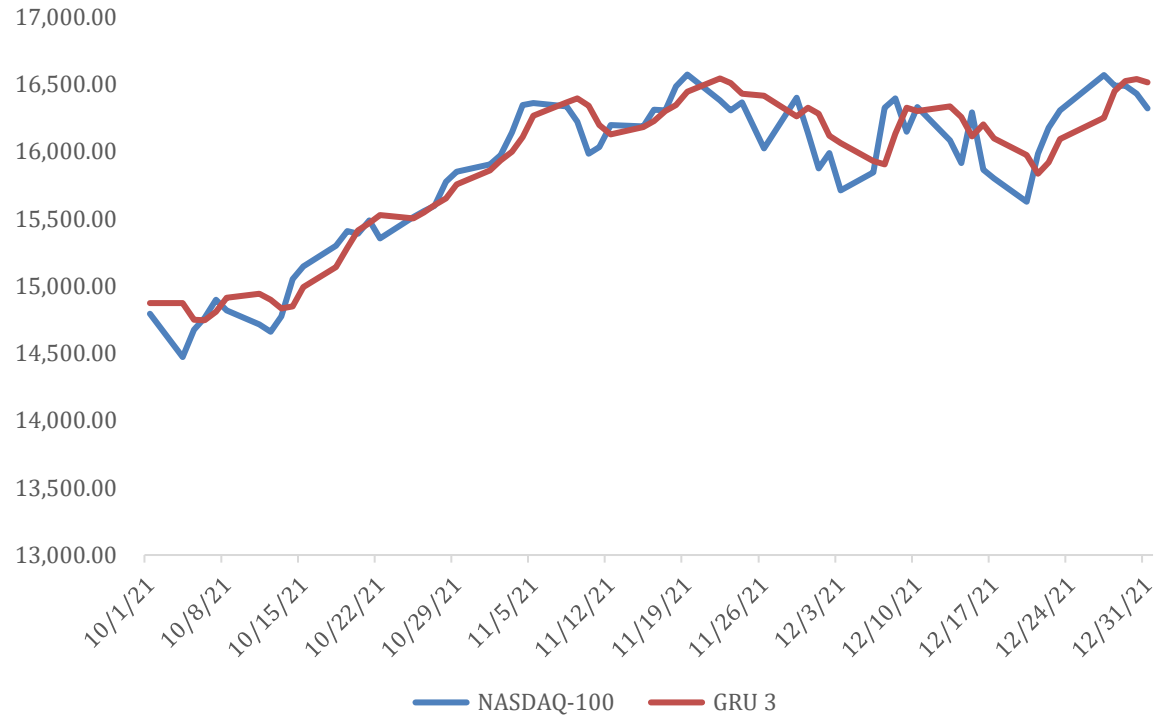

Figure A3. The Nasdaq-100 index and GRU predictions with 3 layers, 50 neurons per layer and without dropout (GRU 3).

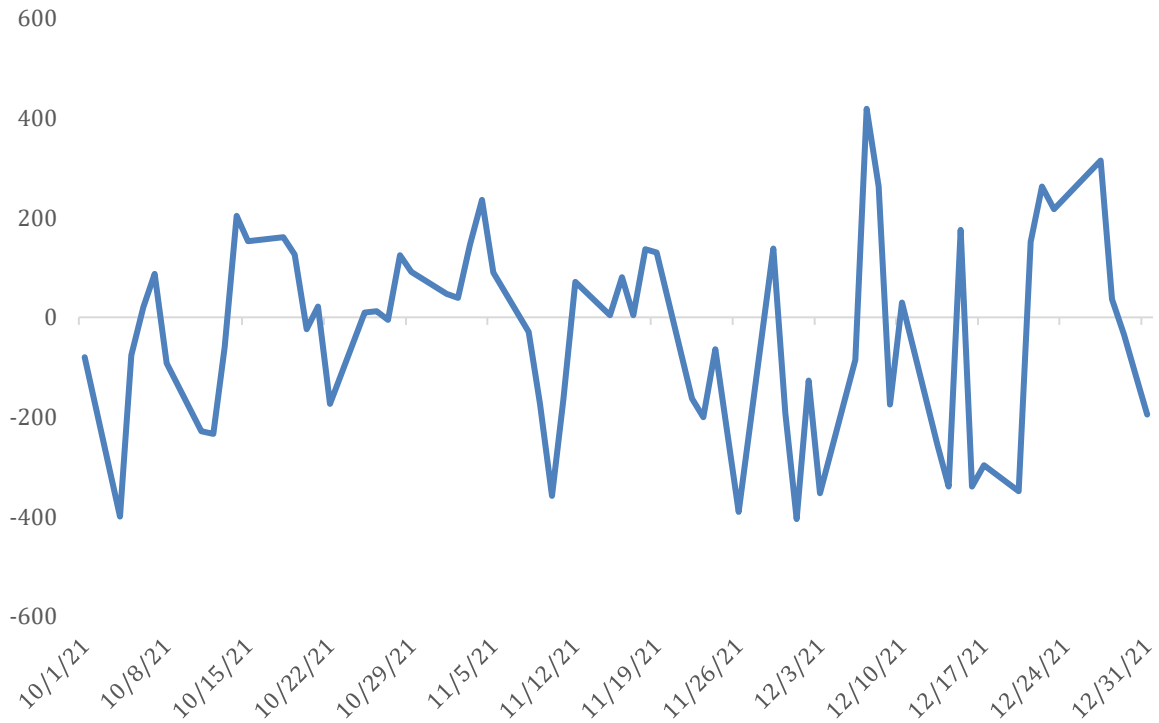

Figure A4. Difference between the actual values and the GRU 3 model predictions.

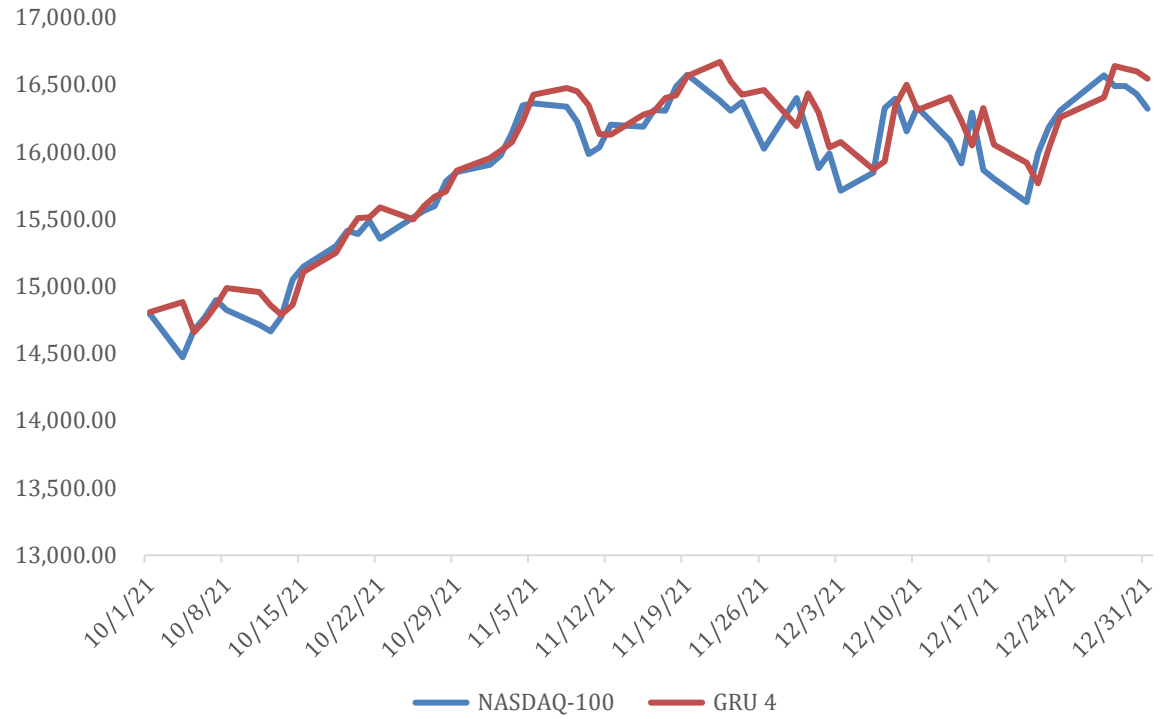

Figure A5. The Nasdaq-100 index and GRU predictions with 3 layers, 100 neurons per layer and without dropout (GRU 4).

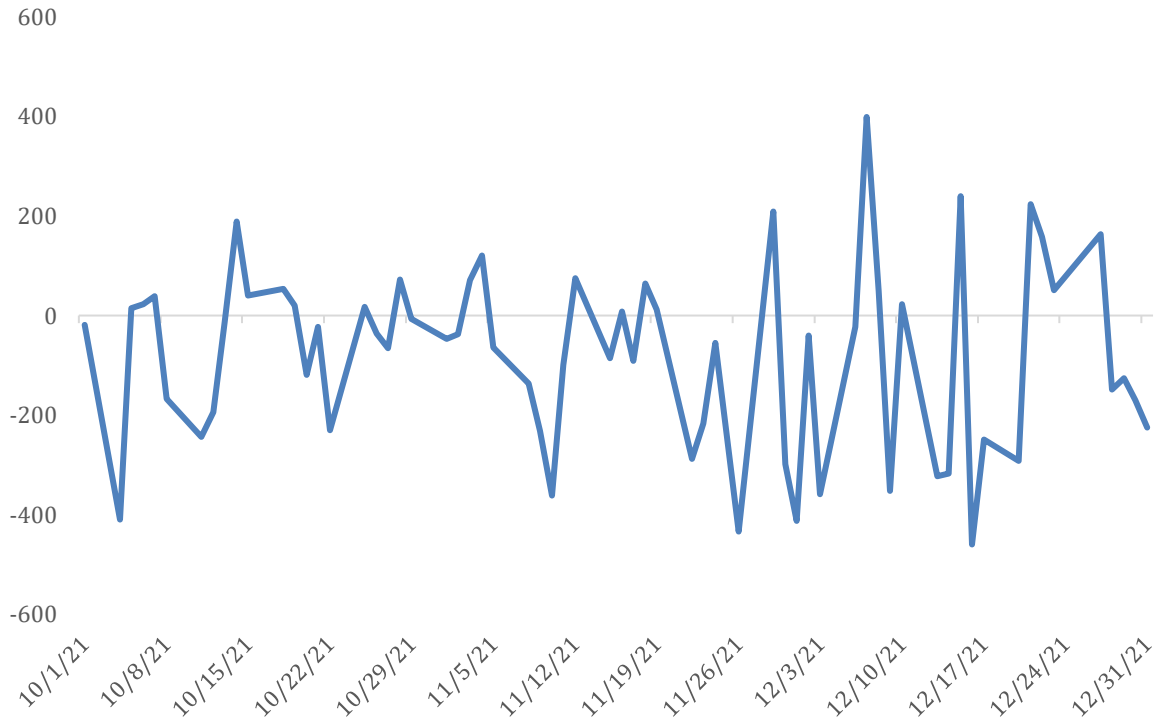

Figure A6. Difference between the actual values and the GRU 4 model predictions.

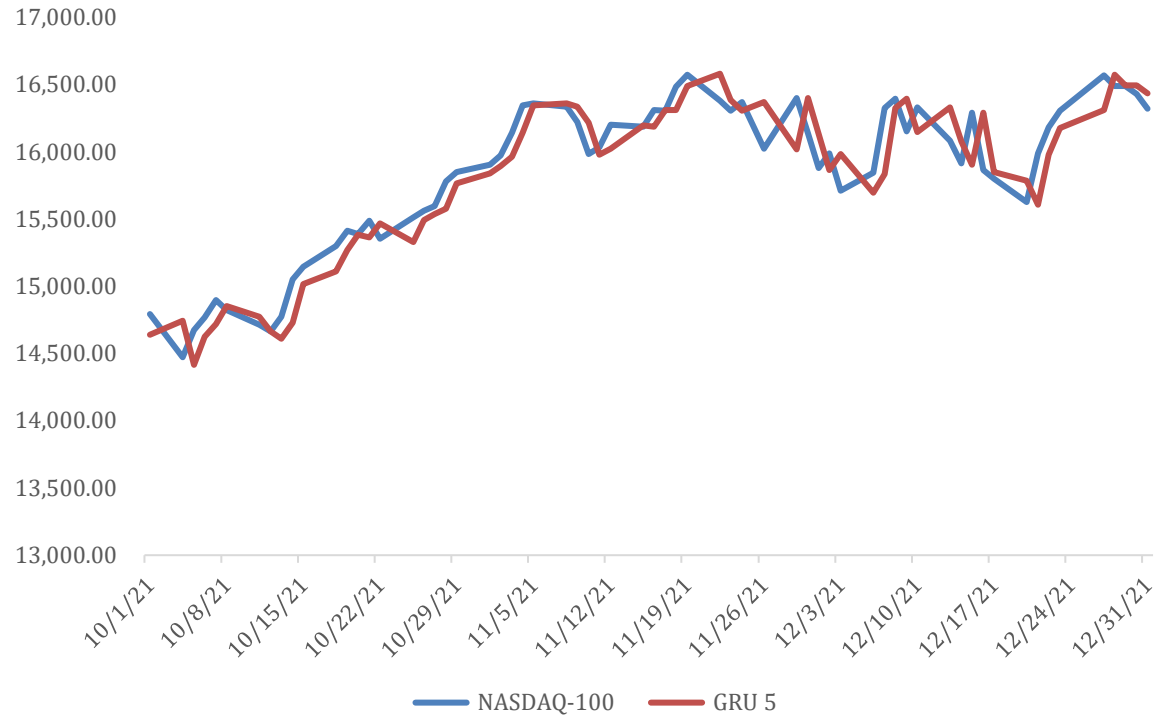

Figure A7. The Nasdaq-100 index and GRU predictions with 3 layers, 50 neurons per layer and 0.2 in dropout (GRU 5).

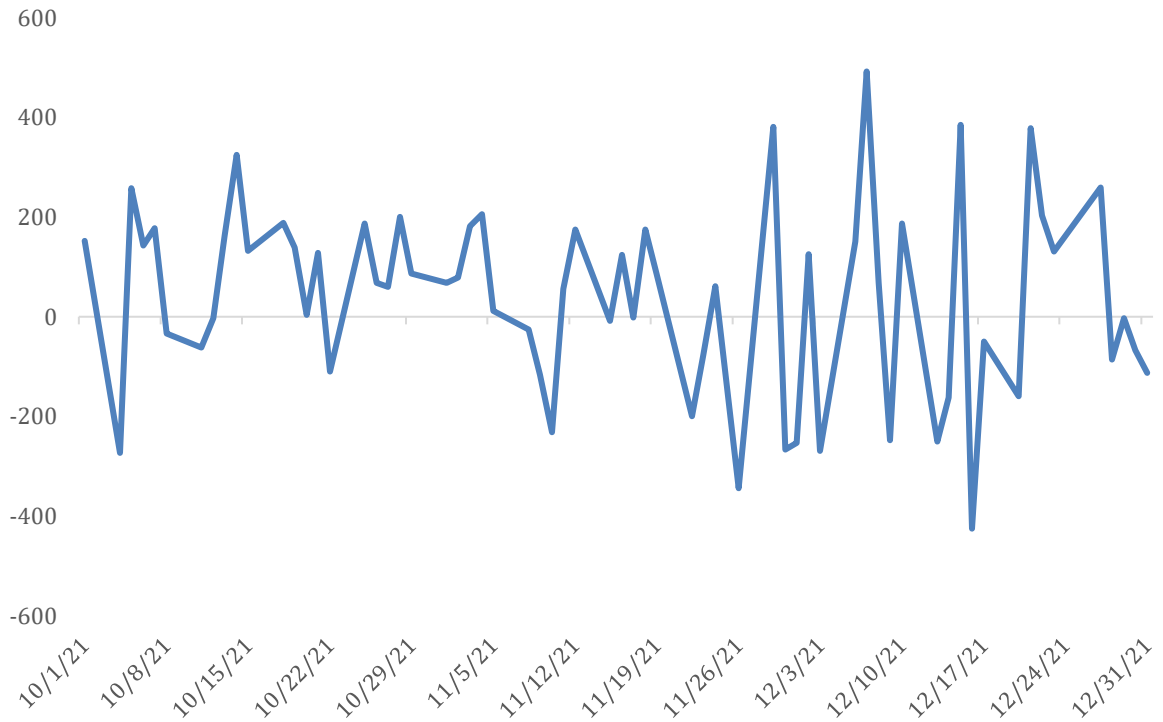

Figure A8. Difference between the actual values and the GRU 5 model predictions.

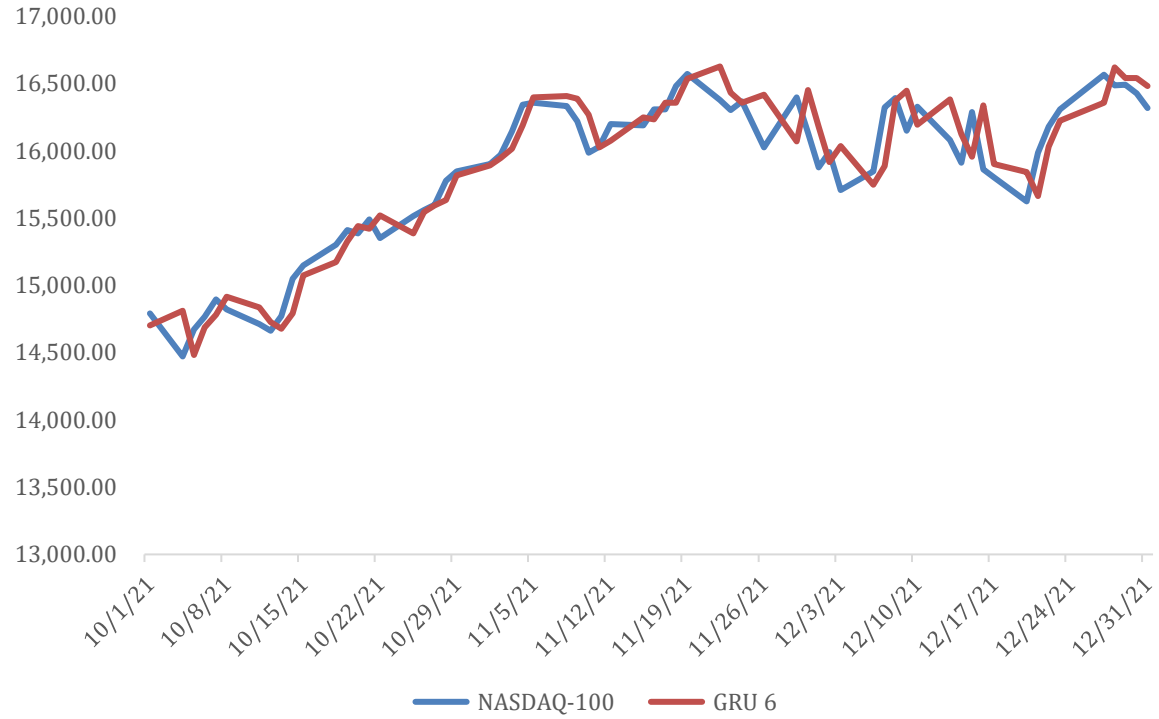

Figure A9. The Nasdaq-100 index and GRU predictions with 3 layers, 100 neurons per layer and 0.2 in dropout (GRU 6).

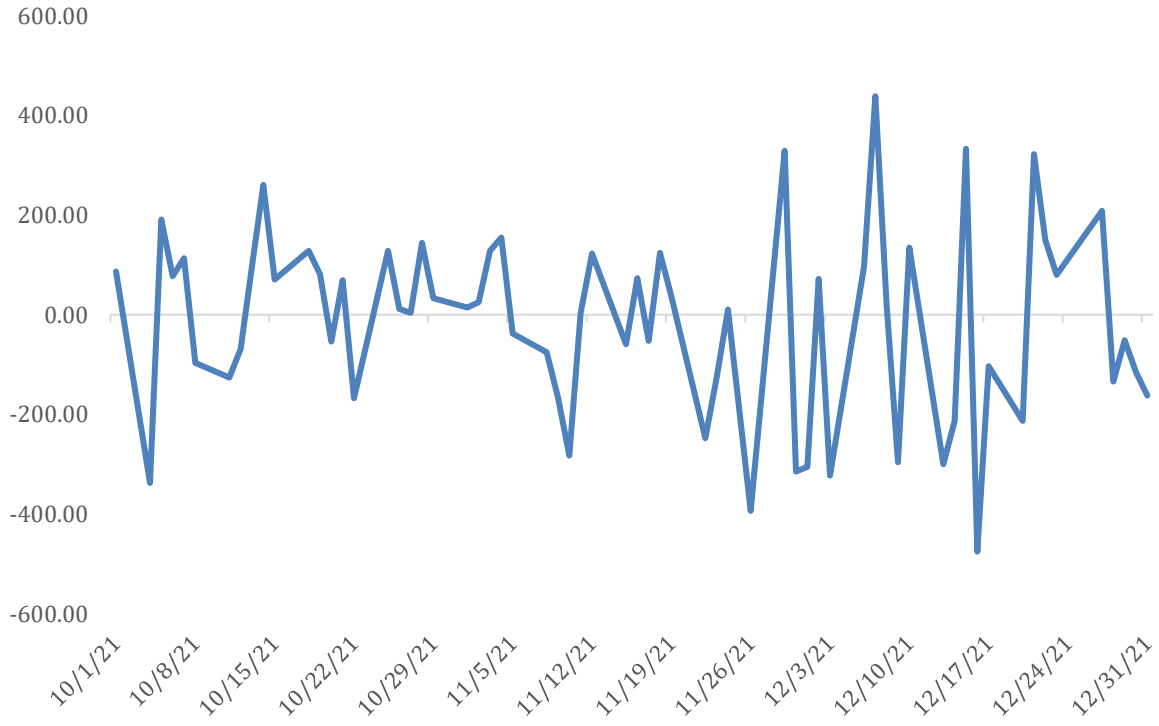

Figure A10. Difference between the actual values and the GRU 6 model predictions.

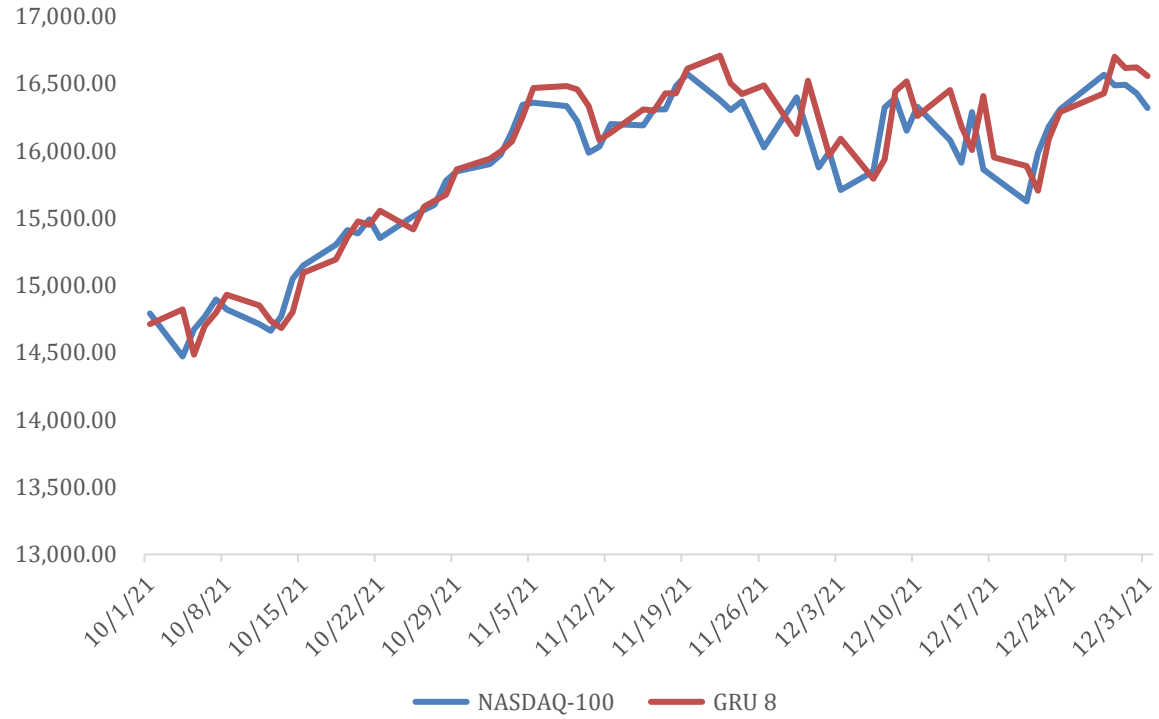

Figure A11. The Nasdaq-100 index and GRU predictions with 2 layers, 50 neurons per layer and 0.2 in dropout (GRU 7).

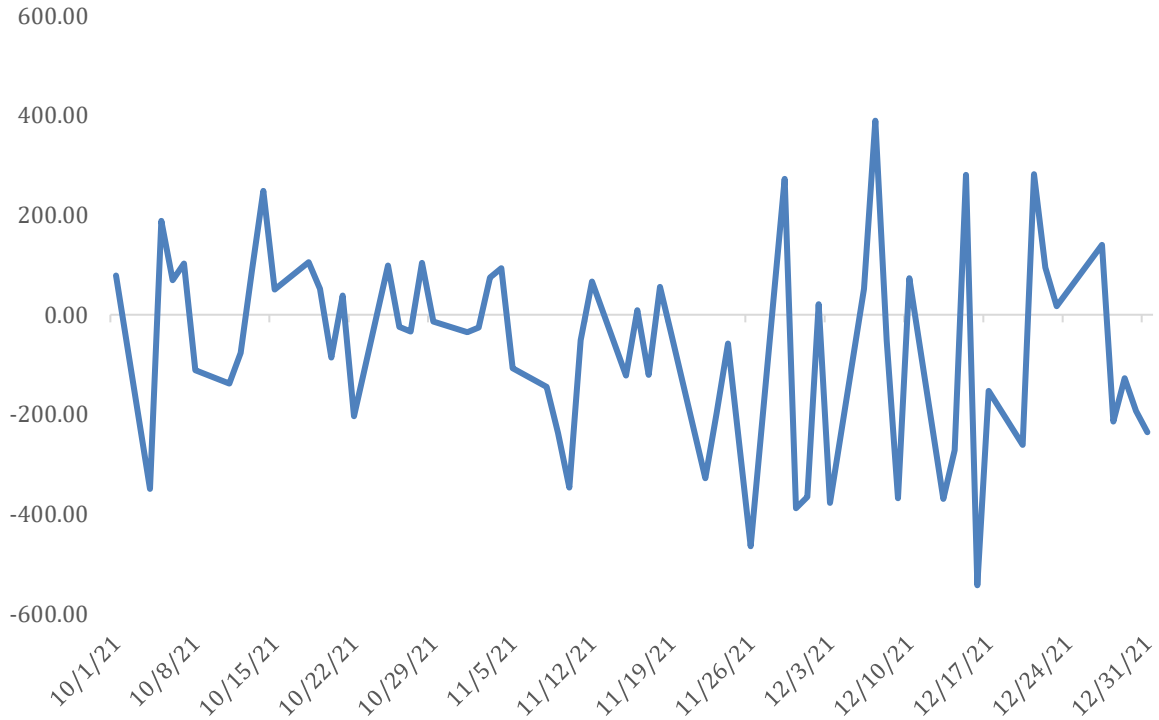

Figure A12. Difference between the actual values and the GRU 7 model predictions.

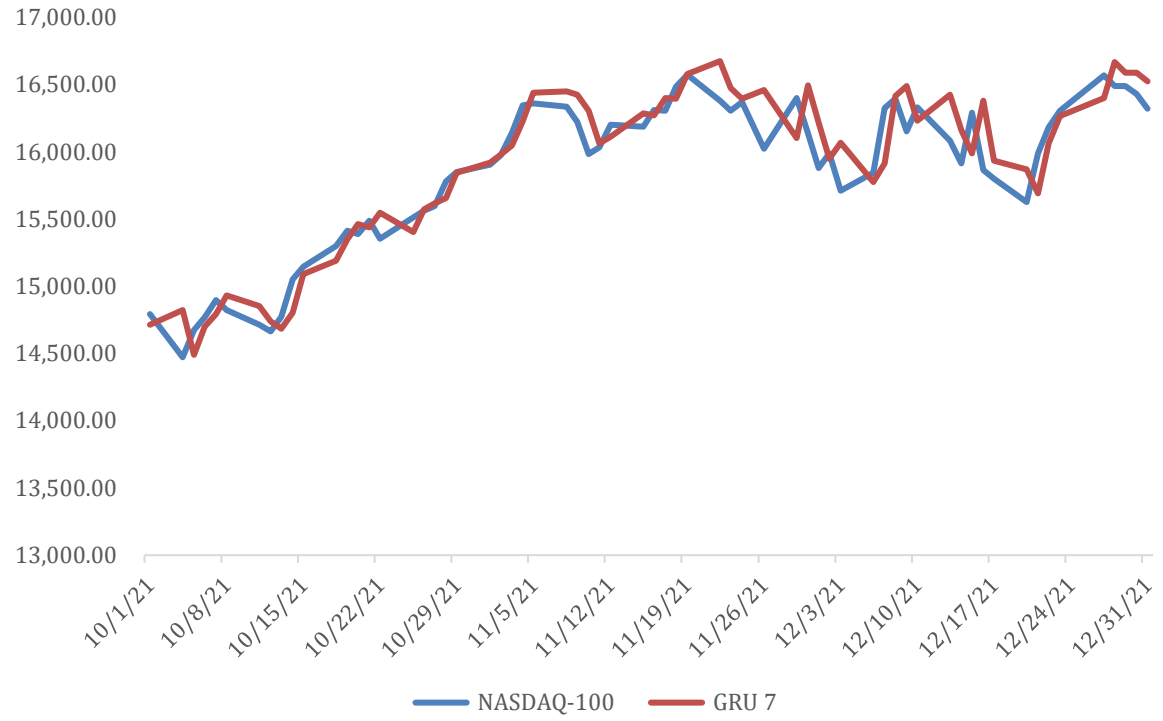

Figure A13. The Nasdaq-100 index and GRU predictions with 2 layers, 100 neurons per layer and 0.2 in dropout (GRU 8).

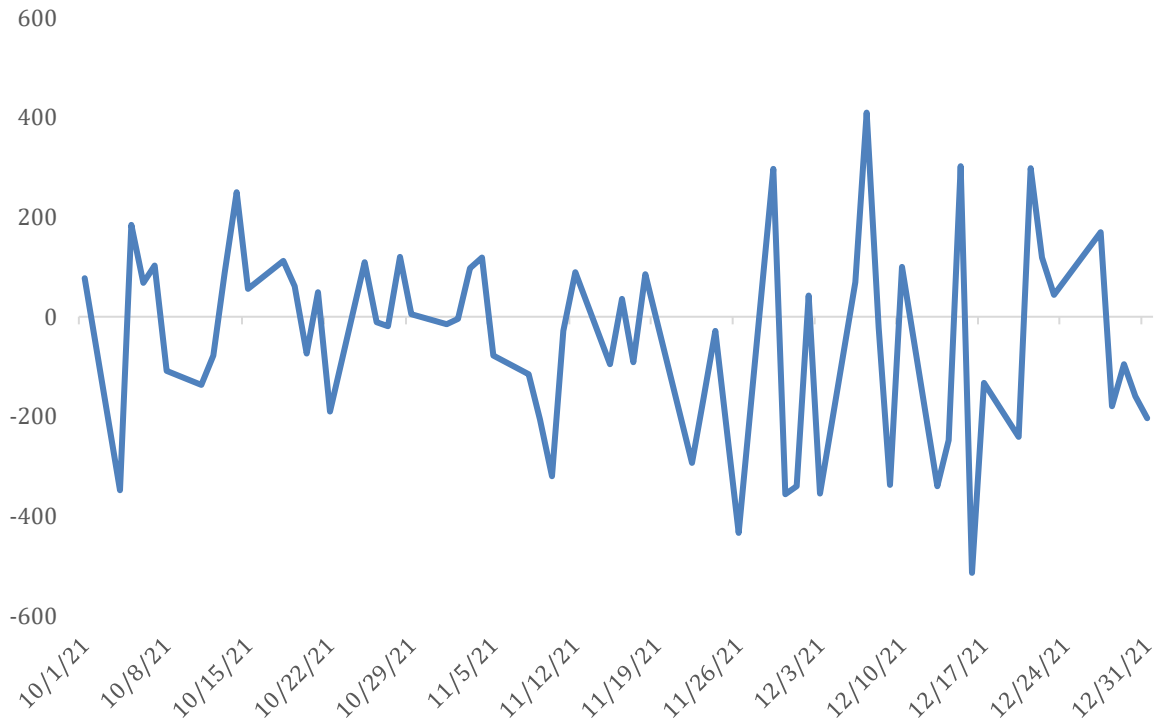

Figure A14. Difference between the actual values and the GRU 8 model predictions.