Stockholms universitet

Performance of Feedforward Neural Network on Exchange-Traded Fund Invesco QQQ.

Prestanda av Feedforward Neuralt Nätverk på Exchange-Traded Fund Invesco QQQ.

Adonis Pelagia

Handledare:
Examinator:
Inlämningsdatum:

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

Abstract

The pursuit of consistently increasing the precision in predicting equity prices has increased over time along with advancements in computational technologies. Institutions, fund managers, and stock managers are actively seeking higher returns from financial markets with the help of algorithmic models and cutting-edge technology. Forecasting equity prices is a considerable challenge due to several factors influencing pricing fluctuations, such as news dynamics, earnings reports, political events, and various other elements that introduce volatility and unpredictability into the equity market landscape.

The thesis is centered on assessing the effectiveness of a feedforward neural network model in predicting the price movements of an Exchange-Traded Fund (ETF), specifically focusing on Invesco QQQ. Given the general perception that ETFs present lower price fluctuations compared to individual stocks, the research seeks to investigate the model's ability to capture and predict the shifts within the Invesco QQQ ETF. The feedforward neural network incorporates fundamental deep learning principles such as early stopping, rectified linear unit (ReLU), and the Adam optimizer. This thesis comprises 713 observations, and while the results are satisfactory, there is a recognition that further studies employing more complex models are necessary for a comprehensive examination of ETF price forecasting performance.

Sammanfattning

I detta arbete ligger fokuset på att utvärdera effektiviteten hos en feedforward neuralt nätverk modell när det gäller att förutsäga priset för en börsnoterad fond (ETF), med specifikt fokus på Invesco QQQ. Med tanke på att ETF:er uppvisar lägre prisfluktuationer jämfört med enskilda aktier, undersöker arbetet modellens förmåga att fånga och förutsäga prisförändringar inom Invesco QQQ ETF. Feedforward neuralt nätverk använder sig av grundläggande principer inom djupinlärning, såsom early stopping, rectified linear unit (ReLU) och Adam-optimizer. Studien omfattas av 713 observationer och resultaten är tillfredsställande. Det erkänns att ytterligare studier med mer komplexa modeller krävs för en omfattande undersökning av prestandan för att förutsäga ETF-priser.

Contents

1 Introduction

Over years, the field of artificial intelligence has surged in innovation as well as transformative technologies. Deep learning being a subfield of machine learning inspired by neuroscience and the function of the human brain [7, p. 13] , has emerged as a powerful paradigm reshaping the landscape of computational intelligence. The recent rise in the success of deep learning can be attributed to several factors such as increased computational power, big data, improved algorithms and architectures and several more of which the combination of these factors has led to a magnificent improvement in deep learning.

Researchers and practitioners strive for continuous innovation, new architectures, algorithms, and further applications. The application of machine learning and artificial intelligence has increased significantly the past few years in the financial world as financial institutions and investors seek for greater returns with less risk. Algorithms have been used for stock price forecasting also known as algorithmic trading.

1.1 Aim and Research Question

The main focus of this thesis is to investigate the performance of a feedforward neural network in predicting the price movements of Exchange-Traded Funds (ETFs). The past few years there has been plenty of research papers investigating the performance of various machine learning algorithms in predicting the price movement of stocks but it is not common in predicting the price movements of ETFs. Throughout this thesis, the ETF Invesco QQQ (QQQ) has been selected as the focus for the feedforward neural network, aiming to predict the adjusted closing price for the next day.

This thesis seeks to answer the fundamental question of whether feedforward neural network can effectively capture the distinctive patterns and behaviors exhibited by ETFs.

Specifically, the thesis aim to answer the following question:

- How effective is the feedforward neural network to predict the price movement of Invesco QQQ Trust for the next day?

2 Background

The following section aims to overview the essential mathematical and computer science fundamentals. There is a clear connection between mathematical principles and computational techniques which has served markedly on the development and innovation of artificial intelligence of which it is the focus of this thesis paper.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is a sub-discipline of computer science that focuses on creating machines and systems capable of performing tasks that typically require human intelligence. Some of those tasks would be analyzing data, recognizing patterns as well as decision making problems. AI has a wide range of applications and it encompasses a wide range of approaches, including machine learning and deep learning which are discussed in the following subsections.

2.2 Machine Learning

Machine Learning (ML) is a sub-discipline of AI and it focuses on developing algorithms and systems that learn and are able to make predictions/decisions by deriving patterns from data. The main idea of machine learning is that systems are able to improve their performance and adapt to any new data provided without the need of human intervention. One known example of a simple machine learning algorithm is naive Bayes which is able to distinguish spam and non-spam emails [7, p. 3].

2.3 Deep Learning

Transitioning to deep-learning, which will be the central emphasis of this thesis, is a sub-discipline of machine learning. The transition from AI to ML and Deep Learning is simply illustrated in Figure 2, providing a visual representation of the connection among these concepts. Deep learning has incredibly involved over the past few years and it is an exciting new technology that for many is considered relatively new. However, the fundamental concepts of deep learning existed already in the 1940s and the fact that it has been associated with various names over time it has undergone fluctuations in its popularity [7, p. 12]. As mentioned in the introduction, deep learning is an approach to artificial intelligence inspired by the field of neuroscience and just like humans learn from experience, machines aim to improve their performance by learning and recognizing patterns from given data. Deep learning relies on neural networks, which get their name from neurons, a term used in neuroscience [9]. In figure 1 a simple neural network consist of several layers (could be single), including the input layer, an output layer and the layers in between called hidden layers. Nodes from separate layers are connected but not necessarily to every node and that depends on the network architecture. Those connections are called edges and they are represented with weights, simply defining the strength of the connection. Each node receives an input information from several nodes and with the assistance of its activation function, it computes its activation value. If the activation value exceeds the given threshold, then the node is activated and passes the information onto the next one, otherwise it does not.

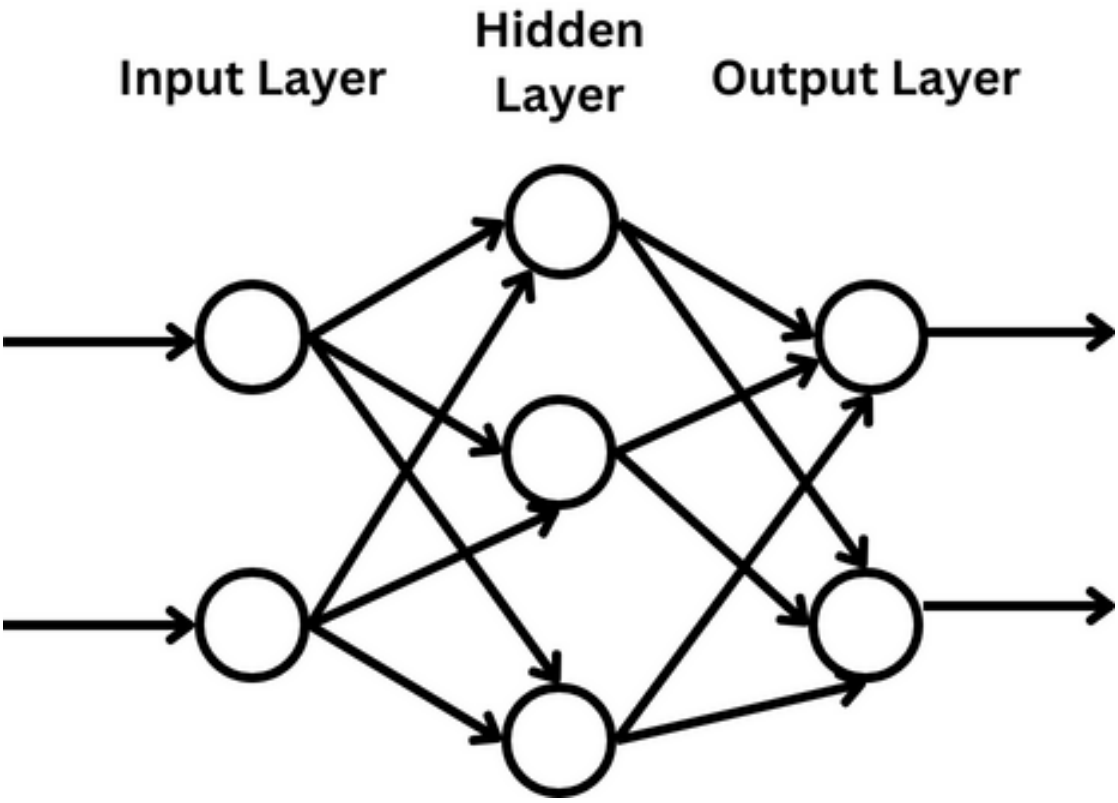

Figure 1: A fully connected feedforward network with an input layer consisting of two neurons. Thereafter, there is one hidden layer with three neurons and an output layer with two neurons.

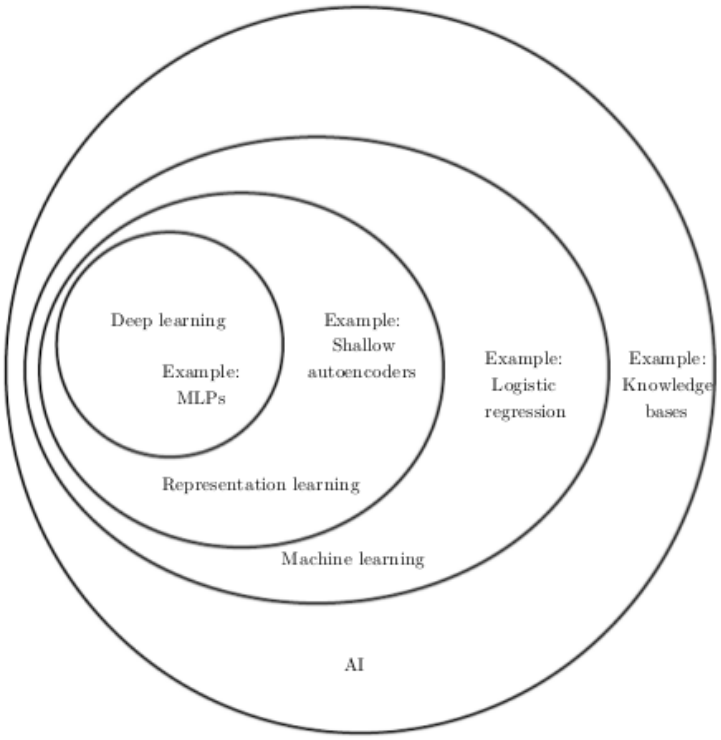

Figure 2: Venn diagram visualising the connection between AI, Machine Learning and Deep Learning.
Source: [7, p. 9]

2.4 Regression

Mathematically regression aims to predict or in other words find a correlation between two or more variables given an input. The algorithm's objective is to give a function as output $f : \mathbb{R}^n \to \mathbb{R}$. A machine learning algorithm that is commonly used for solving regression problems is linear regression. The aim of the algorithm is to produce a linear function that maps the input vector $x \in \mathbb{R}^n$ to the output scalar value $y \in \mathbb{R}^n$. Mathematically let us denote $\hat{y}$ to be the output value and the output can be defined as:

$$\hat{y} = w^\top x, \tag{1}$$

where $x \in \mathbb{R}^n$ is a vector of parameters. The symbol w is a vector of weights associated with each feature in the input vector x. The weights are important as it determines the strength and direction of the impact each feature has on the predicted output. A vector with smaller weight has less impact on the network compared to an input vector with higher weight.

2.5 Bias and Variance

In supervised learning, two fundamental concepts are often referenced: bias and variance. These are statistical metrics that are used to assess model's performance and error in its ability to generalize previously unseen data.

To begin with bias of an estimator is defined as:

$$bias(\hat{\theta}_m) = \mathbb{E}(\hat{\theta}_m) - \theta, \tag{2}$$

where the $\hat{\theta}_m$ represents the estimate of a parameter obtained from a particular model m, therefore it is written with the subscript m. The expectation is over the training data and θ is the true underlying value of θ used to define the data-generating distribution [7, p. 121] .

Thereafter, the variance of an estimator is

$$Var(\hat{\theta}),$$

where the random variable is the training set.

Variance essentially indicates how fluctuating the model's predictions are, with respect to their expected value [15].

Both of the two statistical variables are essential when dealing with supervised learning as it help us identify how far off the predicted output value is from the true value (bias) as well as how fluctuating the predictions are from the true value (variance).

2.6 Under-fitting and Over-fitting

Keeping the generalization error and error gap small is a common challenge in machine learning as it is quite delicate to maintain balance between under-fitting and over-fitting.

On one hand, over-fitting would fit complex patterns from the training set with a very small training error but the model would fail to recognize global patterns in the unseen data, implying a large test error [18].

On the other hand, an under-fit model will have a high training error because it is not complex enough to fit the data from the training set. The model often fails to recognize patterns from the training set and this leads to poor performance on recognizing global patterns from the test set, implying high test error [18].

Ultimately, the goal is to find a balance between over-fitting and under-fitting, finding the correct balance where the model can achieve optimal generalization performance [18]. In the plot shown in Figure 3 the correct balance where the model achieves optimal generalization performance is drawn with a dotted line. At that point the generalization gap between the validation error and train error is neither too small (for underfitting) nor too big (for overfitting).

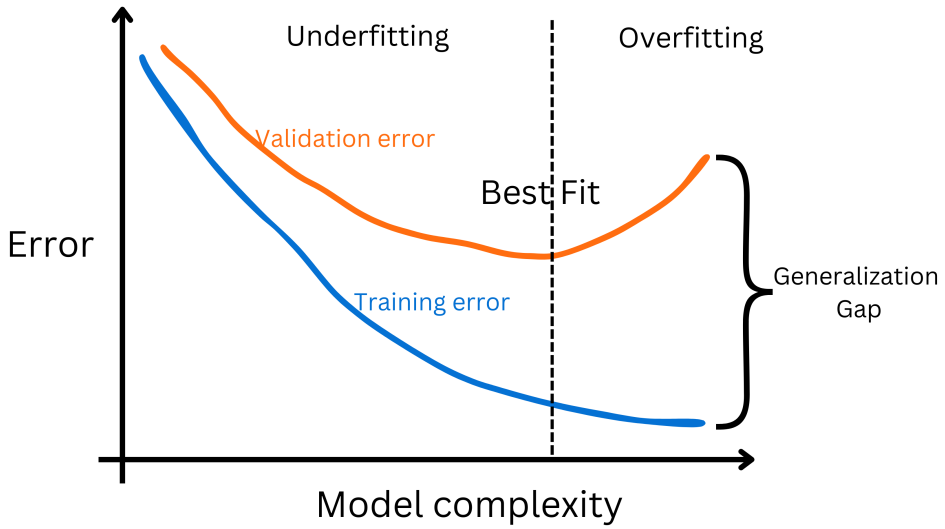

Figure 3: Visualisation of generalization gap between validation error (orange) and train error (blue). Regions with large gap imply over-fitting, whereas those with smaller gap implies under-fit.

2.7 Mean Squared Error

Machine learning often discusses the tradeoff between bias and variance. The bias-variance tradeoff is a crucial concept that revolves around finding the right balance between model simplicity and flexibility.

The tradeoff emerges as a delicate compromise: a model with high bias tends to be too rigid, overlooking the complexity of the underlying patterns in the data, while a model with high variance adapts too closely to the training set, capturing noise and making it less adaptable to new, unseen data.

Moving on and discussing in terms of choosing estimators, it is a common issue. An increasing complexity could lead to decreased bias and increased variance, therefore a common way to find the optimal complexity and negotiate this tradeoff is by using a technique called **cross-validation**. It is

used to evaluate the generalization ability of predictive models and reduce the risk of over-fitting. By systematically dividing the data into training and testing subsets multiple times, it provides a robust way to assess how well a model will perform on unseen data, thereby helping to identify and prevent over-fitting [4].

In addition, one could use the mean squared error as an evaluation metric of the different estimators in order to compare and choose the right estimators for the model. The MSE is given by

$$MSE = \mathbb{E}\Big[(\hat{\theta}_m) - \theta)^2\Big] = Bias(\hat{\theta}_m)^2 + Var(\hat{\theta}_m), \tag{3}$$

which measures the overall squared deviation between the estimator and the true value of the parameter θ.

Moreover, in supervised learning, MSE serves as a mathematical measure to assess the model's performance. This is referred to as the objective function, quantifying the difference between the model's predicted outcomes and the actual values within the training data. The objective function is defined as:

$$J(\theta; X, y), \tag{4}$$

where θ represents the parameter vector that the model aims to learn during training. In the case of this thesis, θ is a parameter vector with the updated weights associated with each node. X represents the matrix, where each row corresponds to a data point, and each column signifies a distinct node. Lastly, y is a vector containing the output values.

2.8 Generalization

When designing models, a fundamental goal is to develop robust models that perform effectively with previously unseen data. Therefore the concept of generalization characterizes the model's capability to perform effectively on the test set. A measure that is used is called generalization error and it is defined as the expected value of the prediction error. Naturally the goal is to decrease the training error as small as possible while

attempting to keep the error gap between the test and training set small. There are various techniques that are used to enhance the performance of the model by reducing the generalization error and they are referred to as regularization techniques.

2.9 Regularization

Regularization is any supplementary technique that aims at making the model generalize better, in other words reduce the error on previously unseen data [10]. Regularization techniques are used in order to reduce over-fitting.

It involves penalizing the model complexity by adding a parameter penalty $\Omega(\theta)$ to the objective function J.

Regularized objective function by $\tilde{J}$ is defined as:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta), \tag{5}$$

where $\alpha \in [0, \infty)$ is a hyper-parameter that weights the relative contribution of the penalty term, Ω relative to the standard objective function J [7, p. 223]. When $\alpha = 0$, the regularization term has no effect, and the model only tries to minimize the objective function. However, as α increases, the penalty on the model's parameters is stronger, resulting to simpler coefficients that prevent the model from overfitting. Therefore, α is vital in controlling the trade-off in the model and finding a good generalization performance.

There are several regularization techniques that are widely used, including L1 regularization, L2 regularization, data augmentation, early stopping, and dropout, which are available to mitigate over-fitting in machine learning models. The selection of an appropriate technique depends on the characteristics of the data that each model is trained on.

Early Stopping

When training a neural network, the aim is to reduce the training set error to the smallest error possible. However it has been shown that as the error decreases, there reaches a point where it eventually starts to degrade, resulting in increases error on the unseen data. (See Figure 4) Generalization error is evaluated through the use of a validation set, measuring the average error observed on distinct instances within that particular validation set over multiple epochs. What is meant by the term epoch is how many times the learning model will run through the entire dataset in order to update the parameters [5]. In the context of early stopping, the model assesses the generalization error over multiple epochs, as the model will run through the entire dataset and update its parameters. For example, one epoch implies that the learning algorithm has run through the whole dataset once and it has updated the parameters once (weights and biases).

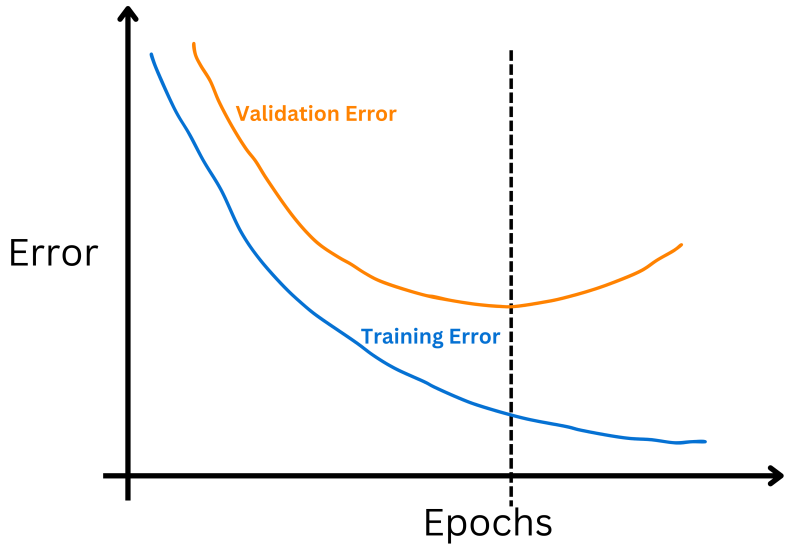

Figure 4: Early stopping is implemented when reaching the dashed line, signaling the interruption of training as the model exhibits no further improvement beyond this point.

Early stopping proceeds by first dividing the data into a training set and a

validation set, commonly done in a 2-to-1 ratio. Thereafter the model is exclusively trained on the training set, refining its parameters over several epochs. Over multiple epochs, the model's performance is evaluated on the validation set and as soon as the model's performance on the validation set is worse than the last step checked, the training process is stopped. The model uses weights from the last check as the resulting outcome [16].

Moving on, it is important to have a stopping criterion in order to halt the training of the model. There are several conditions that one could be using as stopping criteria but in this section two possibilities is discussed. The choice of stopping criterion depends on the specific characteristics of the data-set, and the behavior of the learning algorithm during training.
A common stopping criteria that is set on this investigation is a threshold equal to 0.001 that specifies the minimum improvement required to consider the model's performance as better. Training is halted if the improvement falls below this threshold.

In this study the early stopping technique is employed in the feedforward neural network as a form of regularization. The main reason of employing early stopping over other regularization techniques lies in the dynamic nature of an ETF prices, which are influenced by various market conditions over time. Early stopping is suitable regularization technique for this thesis as it enables the model to adapt and be flexible during training. Other techniques impose fixed penalties which may limit the flexibility in the face of the dynamic environment of ETF pricing.

2.10 Hidden Units

Choosing in advance the right type of hidden unit in the hidden layers can be quite hard but one can make a choice by intuition and trial and error. Most hidden units can be described as accepting a vector of inputs x, computing an affine transformation $z = W^\top x + b$, and then applying an element wise nonlinear function g(z) [7, p. 187].
However there exists a standard choice of activation functions i.e., Recti-

fied linear units, logistic sigmoid or hyperbolic tangent.

2.11 Activation Function

Rectified Linear Units (ReLU) uses the activation function $g(z) = max\{0, z\}$. Half of the rectified linear unit's domain outputs zero and it does so when $x < 0$. Otherwise it produces an outcome of a linear function when $x \geq 0$ [2].
It implies that the activation function takes an input x and if the output is positive, the output is equal to x. Otherwise, the result is 0.

In this report, ReLU is employed as an activation function. The main reason behind this choice is because ReLU produces a function of non-linearity which is suitable for the non-linearity problem of this report.

2.12 Architecture Design

The architecture design of neural networks require thoughtful consideration due to its non-linearity. It is vital to optimize the performance of the model by choosing the right depth (number of layers in the neural network) and the right width (number of neurons in each layer). This process involves organizing units into multiple layers that form a chain structure, where each layer's output serves as the input of the upcoming layer. The architecture can be defined by functions representing the transformations happening at each layer, where the first layer is defined as:

$$h^{(1)} = g^{(1)}(W^{(1)\top}x + b^{(1)}),$$

the second layer:

$$h^{(2)} = g^{(2)}(W^{(2)\top}h^{(1)} + b^{(2)})$$

and so on [7, p. 191].

In the case of this thesis and taking into consideration the small dataset, architecture design of the model consists of 5 layers; input layer, three

hidden layer and output layer. The input layer consists of nine neurons as it has been determined from the Auto-correlation function shown in Figure 6. Additionally the three hidden layers consists of 8 neurons, 4 neurons and 4 neurons respectively. Finally the output layer constists of a single neuron which is the ETF's next day adjusted closing price.

2.13 Optimization

Optimization algorithms are used for training of deep models and updating model parameters in order to minimize the loss function. The loss function can be written as an average over the data-generating distribution p_{data}

$$J^*(\theta) = \mathbb{E}_{(x,y)\sim p_{data}} L(f(x;\theta), y), \tag{6}$$

where L is the per-example loss function, $f(x;\theta)$ is the predicted output when the input is x [**?**] [7, p. 268]. Per-example refers to the loss that is actually computed for every data point in the dataset. Lastly $\hat{p}_{data}$ is the empirical distribution, which represents the distribution of the observed data.

Batch and Mini-batch Algorithms

Most algorithms use more than one sample training examples, referred as mini-batch, but do not use all training examples. Mini-batch is defined as the use of a subset of training samples where the batch size is more than one sample and less than the whole size of the training set. These are called mini-batch stochastic methods. Approaches employing the entire training set are referred to as batch gradient methods, whereas the batch size is equivalent to the whole training set [5]. In this study, the mini-batch stochastic method is utilized, whereas the use of mini-batches is employed for more efficient optimization of the model.

In the current thesis a mini-batch of size 8 and a number of epochs that is run through is 100. The reasoning is because the number of observations makes a small dataset, as the training set consists of approximately 572

datapoints. In this case, the training dataset is divided into 71 batches, each consisting of 38 datapoints. This implies that the learning model will be updating its weights and biases after each batch (8 datapoints). As defined in an earlier section, one epoch will take into consideration 71 batches. Since the number of epochs is 100, then it runs through the whole dataset 100 times, giving a total of 7100 mini-batches for the whole training of the model.

Stochastic Gradient Descent (SGD)

Stochasting Gradient Descent is an optimization algorithm that is most used in machine learning and deep learning. The objective is to minimize the loss function during the training process by adjusting the model's parameters. Gradient Descent (SG) is an algorithm that is used for updating the parameters of a model by computing the gradient on the whole training dataset. However when dealing with large datasets, SG faces challenges as it may require high computational memory or computation may be too slow. Then SGD is a variation of GD and its core, as the word "stochastic" reveals, is to update models parameters based on the gradient of the loss function computed on only one data set per iteration, rather than the entire training dataset. The process of SGD starts by selecting a proper learning rate ϵ and initial parameter values of the model θ. Thereafter, for every iteration the model's parameters are updated as followed:

$$\theta_{i+1} = \theta_i - \epsilon \times \nabla_\theta J(\theta; x^j; y^j) \tag{7}$$

The process is repeated until the loss function is optimized, in other words reaches a local minima [17].

Additionally, a variation that is used in this thesis is a mini-batch gradient descent, whereas the gradient is computed on a mini-batch, consisting of n training datapoints. The idea and steps are common to SGD with the only difference being that the training dataset is split into smaller subsets (mini-batches) and the gradient is calculated and parameters are updated for every mini-batch [17]. Steps are identical to ones mentioned in the

paragraph above, but the updating of parameters (θ) is defined as:

$$\theta_{i+1} = \theta_i - \epsilon \times \nabla_\theta J(\theta; x^{j:j+n}; y^{j:j+n}) \tag{8}$$

Two conditions which are related to the choice of the learning rate are:

$$\Sigma_{k=1}^{\infty} \epsilon_k = \infty, \tag{9}$$

and

$$\Sigma_{k=1}^{\infty} \epsilon_k^2 = \infty, \tag{10}$$

where ϵ_k is the learning over k iterations [7, p. 287].

The parameter ϵ, referred as the learning rate, is crucial when considering SGD algorithm. It essentially determines the size of the step taken during the optimization process. Consequently, too low learning rate can result to very slow convergence, even causing the algorithm to be stuck in local minima. Additionally, a too large learning rate can cause wild fluctuations in the loss function. Thereafter as described above, condition (0.9) ensures that the learning rate is large enough in order to evaluate all parameters effectively. Condition (0.10) ensures that the learning rate is not decreasing too fast so that convergence is guaranteed. A crucial property relating SGD and mini-batch is that the computation time for every update does not grow with the number of training points [7, p. 287]. One reason behind that is because every update is not determined by the total number of training points, while it is determined by the size of each subset (mini-batch). Additionally the property can be tested with empirical observations. It is possible to time the training process of a model with different datasets. If the time per update remains constant or it is growing significantly slow as the dataset size increases, it confirms the property relating SGD and mini-batch.

Momentum

Momentum is another important technique used in optimization algorithms in order to boost the convergence of the process but simultaneously

ensure no slow convergence or oscillations in the loss function. SGD as already mentioned updates the parameters after each iteration k. At every iteration, denoted as k, the model's parameters are updated with the help of computed gradient of the loss function at that specific moment (at iteration k). This indicates that parameters are updated after each iteration k, implying that previous steps are not considered when searching for the next iteration's solution [11].

Since SGD does not consider the history of the steps it results to two issues: Firstly a non-convex loss function can have many local minima which is problematic as there is not guarantee that the first local minima found is also the global minima. Consequently the gradient of the loss function is minimal which produces no weight updates. Secondly, it can occur that gradient descent is noisy which causes several oscillations. This is also problematic as a larger number of oscillations is needed until convergence is reached [11].

The way this technique works is by calculating a weighted average of all the previous gradients and utilize the average to update the model weights [12]. It results to smoother steps taken during gradient descent since previous gradients are all taken into consideration.
Therefore the updated equation where SGD is applied with momentum is defined as:

$$\theta_i = \theta_{i-1} - \epsilon \times b_i, \tag{11}$$

where b_i is the momentum term, which is the modified step direction [12]. The modified step direction implies that the optimization process takes into consideration both the current gradient as well as the momentum from previous steps. This enhances the process by building momentum towards directions where the gradient is pointing, resulting to smoother and potentially faster convergence.

Adam

Adam, short for Adaptive Moment Estimation, is probably the most common optimization algorithm used for training models. The algorithm is an extension of the Stochastic Gradient Descent and it combines ideas from momentum, Root Mean Square Propagation (RMSprop) and AdaGrad algorithm [8].

An overview of the Adam algorithm according to Ajagekar [3] is broken down into 5 parts, as the algorithm is used in this study for optimizing the feedforward network.

Firstly, Adam initializes two moment variables, noted as $s = 0$ (estimating momentum) and $r = 0$ (estimating RMSprop). In addition, there is the initialization of time step $t = 0$.

Thereafter, the algorithm includes the following hyper-parameters:
- α, being the learning rate,
- β_1, which is the exponential decay rate for the first moment estimate (default set to 0.9).
- β_2, which is the exponential decay rate for the second moment estimate (default set to 0.999).
- ϵ, a small constant (10^{-8}) to prevent division by zero.
Same values are applied for this thesis.

The Adam algorithm keeps running and updating the hyper-parameters until the stopping criterion is met. Assuming there is a sampling of a mini-batch of k samples, then the algorithm computes the gradient $g_t = \frac{1}{k}\Sigma_{i=1}^{k}\nabla J_i(\theta_t)$ at each iteration t on the current mini batch. The formula indicates that g_t is an average of all the gradients of each individual sample contained in the mini-batch.

The algorithm updates the first moment variables s and r:

$$s_t = \beta_1 \cdot s_{t-1} + (1 - \beta_1) \cdot g_t$$

$$r_t = \beta_2 \cdot r_{t-1} + (1 - \beta_2) \cdot g_t^2$$

.
One thing that is important is the bias correction as the initialized variables are biased towards zero, therefore Adam applies bias correction to reduce this. The estimations are calculated:

$$\hat{s}_t = \frac{s_t}{1 - \beta_1^t}$$

$$\hat{r}_t = \frac{r_t}{1 - \beta_2^t}$$

The estimators shown are used for correcting the bias as the number of iterations t increases. This is done by scaling with a factor of $\frac{1}{\beta_1^t}$ and $\frac{1}{\beta_2^t}$ respectively. It ensures a proper adjustment during the first and second moment estimation, especially during the initial stages. It is essential at the initial stages because as the number of iterations increases, the term β_1^t and β_2^t decreases, resulting to a correction factor equal to 1.

The model updates parameters θ based on the corrected estimates:

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{s}_t}{\sqrt{\hat{r}_t} + \epsilon}$$

[3]

Adam algorithm is used in this thesis with the parameters: $\alpha = 0.001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ and $\epsilon = 1e - 08$. As already mentioned, Adam algorithm takes the benefits from SGD, momentum, RMSprop and Ada-Grad algorithms which makes it appropriate for this thesis. Since the aim of investigation evolves around price forecasting, its ability to adjust its

parameters automatically based on historical gradients, makes it suitable for this problem. Thereafter, time-series analysis can be challenging and Adam algorithm's momentum terms take care of those challenges of finding local in an efficient manner.

3 Method

In this section, a clarity on the chosen data-set and the reasoning behind its selection will be given. The feedforward neural network model is designed to forecast the adjusted closing price one day ahead, a task that involves predicting a continuous numerical value and is thus framing it as a regression problem.
Only one experiment will be undertaken in this study, with an inclusion of predictor variables.

3.1 Programming

The study is using Python and a range of libraries to help with the create of the model as well as the analysis of the performance.

To begin with, Python programming language version $3.11.6$ is used for implementing the different stages of the study. Thereafter, the Scikit-learn library is used. It is a robust machine learning library and it is mainly used for data pre-processing and to evaluate the performance of the model. Pandas library is employed for efficient handling and pre-processing of the data-set. Its main use is to read the data-set and ease the process of organizing the data by cleaning and transforming the raw data into a desired format. NumPy is an essential package for scientific computing, playing a crucial role in numerical operations and array manipulations. The library Matplotlib is quite popular and it is used for data visualization. It is utilized to create clear and insightful visualizations. Further TensorFlow is an open-source machine learning library. Its use is mainly employed for building and training neural network models. Last, but not least,

Statsmodels, a Python library, is utilized in this study to generate the auto-correlation function on the data-set.

3.2 Data Collection

The data-set is retrieved from Yahoo Finance and it is from the period 2019-01-01 to 2021-10-31. The data-set comprises 713 rows of data, indicating a total of 713 data points corresponding to an equal amount of trading days. The initial data-set includes columns Date, Open, Low, High, Close, Adj Close, and Volume. Each column reveals information on the price movement of QQQ. The Open and Close columns reveal the opening and closing prices of QQQ for the respective trading days. Thereafter the Low and High columns shows the highest and lowest prices of QQQ for the respective trading days. Volume shows the total amount of shares that have been traded during the specific trading date. However, in a subsequent step, the columns Close and Volume have been filtered out for simplicity.

This study focuses on a supervised learning problem, therefore the process of splitting the data-set into training, validation and test set are not done randomly. Instead, it employs a time-based selection, where older dates are used as training set (80%) and leave the most recent 20% of the data-points for the validation set (10%) and test set (10%).

3.3 Data Formatting

As previously mentioned, the data-set encompasses two columns, with each column revealing distinct information corresponding to each trading day.

To begin with, the Date column reveals the date of the trading day. Following this, the next column is called "Adj Close". It is chosen over the "Close" column because it represents the adjusted closing price of the Exchange-Traded Fund (ETF). This adjustment takes into account vari-

ous corporate actions, including stock splits, dividends, and other events that might influence the ETF's price. Opting for the "Adj Close" column ensures a more accurate depiction of the ETF's true value over time.

In this study, the variable under consideration as the response is the next day's Adjusted Close price, while the factors influencing this response (predictor variables) include the current day's Adjusted Close prices and the Date. The following can be expressed mathematically as:

$$\text{Adj Close}_{t+1} = f(\text{Adj Close}_t), \text{with } t\text{=Date} \tag{12}$$

To start the process and before incorporating the data-set into the model, it is essential to investigate the stationarity of the data.
The data shows to be extremely non-stationary. Therefore in order to make the prices closer to stationary, the first difference times series is calculated.

The first difference time series for the Adjusted Close Price variable at time t is calculated as:

$$\triangle C_t = C_t - C_{t-1} \tag{13}$$

4 Results and Discussion

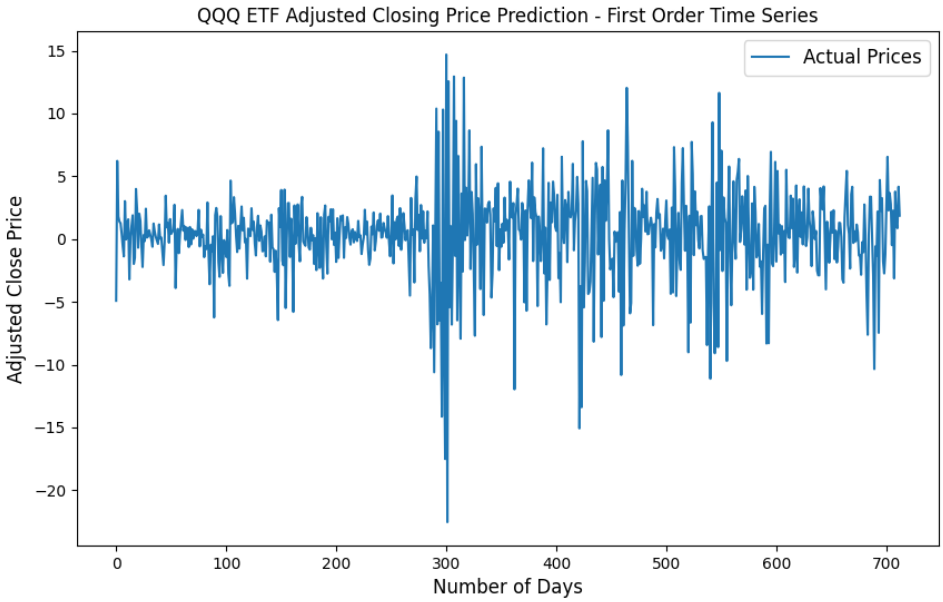

Figure 5: First difference time series of the data-set, visualising an approximately constant mean but not as much in variance.

Time series analysis is a vital technique used to analyse and interpret data points over time. A time series achieves stationarity when its key statistical characteristics such as mean, variance, and auto-correlation remain constant throughout time. Achieving stationarity is crucial for several compelling reasons, notably to extract meaningful statistics such as mean, variance, and auto-correlations with other variables. These statistical measures become dependable indicators of future trends only when assessed within the framework of a stationary series [13].

Figure 5 illustrates the first difference time series of the data, showing how this technique aids in achieving stationarity. It can be observed that approximately the mean and variance are more-or-less constant, which makes it significantly easier to use its characteristics for forecasting. While commonly employed for this purpose, it comes with at a cost of data loss.

This introduces specific limitations and challenges when forecasting time series of such a nature.

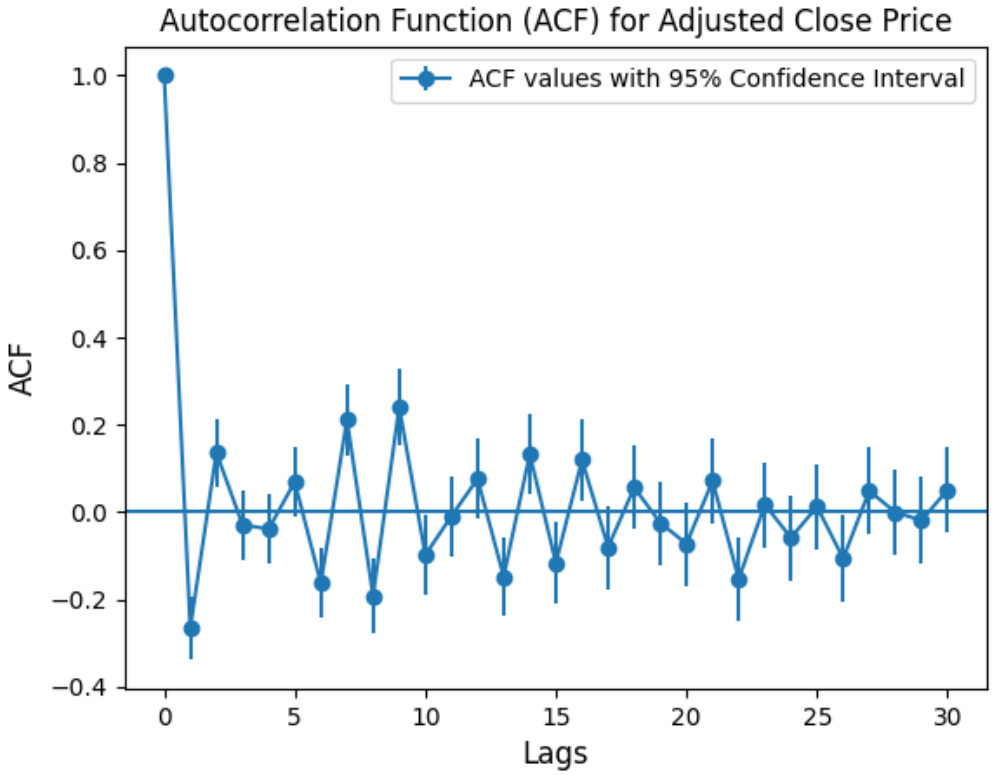

Figure 6: Auto-correlation function with vertical bars indicating the standard error. Strong correlation up to lag 9.

Additionally, following the creation of the first difference time series plot, the analysis proceeds to generate and examine the auto-correlation function (ACF) plot.

Mathematically it is defined as:

$$ACF(k) = \frac{Cov(Y_t, Y_{t-k})}{\sqrt{Var(Y_t) \cdot Var(Y_{t-k})}}, \tag{14}$$

The ACF plot serves as a statistical tool to analyse the correlation between

the first difference series and its lagged values at different time intervals. Analyzing the ACF is crucial for identifying potential temporal dependencies and plays a key role in selecting the appropriate time series models. It enhances our understanding of how past observations interrelate with future ones in the transformed time series.

This plot is valuable in determining the number of lags (number of time steps) to include in the model. Further, at lag 0, a perfect correlation is expected, as it reflects the correlation between the time series and itself at the same time point. Thereafter, the goal is to identify the model order by examining spikes in the lags, indicating close correlations between variables. It's evident that for $k > 10$ there is no significant correlation, gradually approaching 0. However, distinct spike is observed at lag $k = 9$.

In the context of time series analysis, negative lags are often less relevant, representing correlations with values preceding the current time point. Consequently, they are not given as much consideration.

In conclusion, based on insights from the ACF plot, the chosen model order, implying the number of included lags, is determined to be $k = 9$.

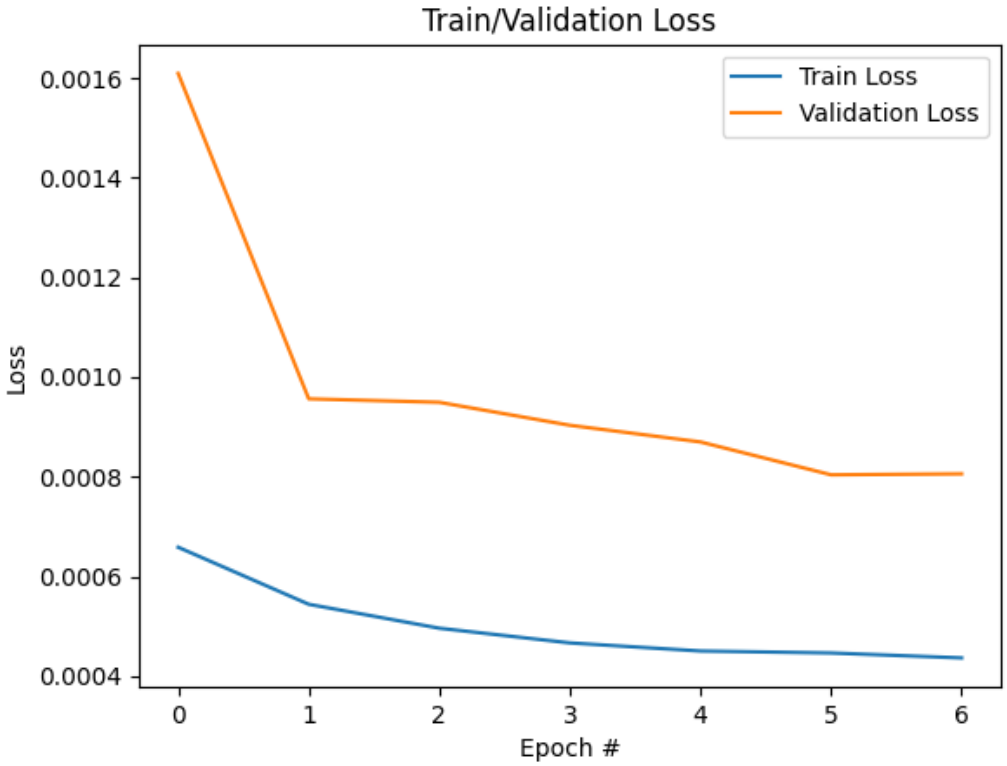

Figure 7: Training and validation loss over epochs with early stopping: Monitoring the learning process as the model trains to prevent over-fitting and achieve optimal performance. At epoch 5 to 6 the validation loss is flat, implying the model is no longer improving.

Figure 7 displays the training and validation loss curves, providing a visual representation of the loss resulted during the training and validation phases. The learning curve derived from the training data-set illustrates the model's learning performance, showing how effectively it is adapting to the provided data. Similarly, the learning curve generated from the validation data-set provides insight into the model's generalization capabilities, offering a visualization of its performance on unseen data [6]. These curves serve as valuable tools for assessing and understanding the training and generalization behavior of the model throughout the learning process.

As observed in Figure 7, the training loss curve tends towards 0 as the number of epochs increases. This reflects the model's effective learning of

the training data. This outcome is anticipated, particularly given the relatively small number of data points used for training, allowing the model to accurately predict target values for the training examples.

However, the elevated validation loss observed at the beginning is a result of the model's parameters being randomly initialized. This initial randomness can cause predictions to deviate significantly from the actual target values, resulting in a higher validation loss. As the number epochs progresses, the SGD algorithm updates the model's parameters based on the gradients of the loss function with respect to these parameters. Each iteration involves processing a random subset (mini-batch) of the training data, making the optimization process stochastic. Finally, the model moves toward a minimum of the loss function and the validation loss gradually decreases. It finally reaches plateaus at Epoch $5 - 6$ implying that the model is no longer improving.

A well-fitted model's learning curve typically exhibits a high validation loss initially, which decreases as more training examples are added. Eventually, the curve plateaus, suggesting that additional training examples do not significantly enhance the model's performance on unseen data.

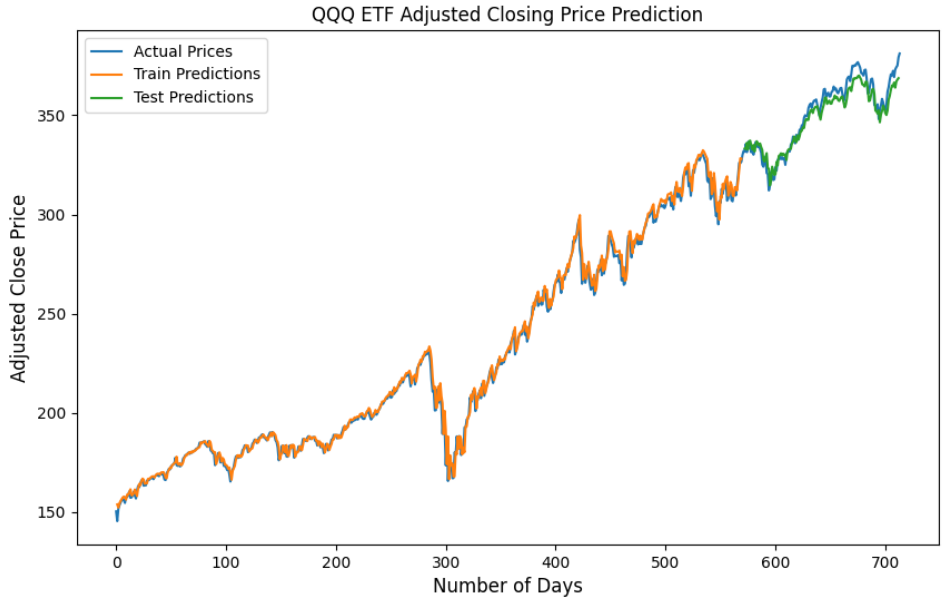

Figure 8: QQQ ETF's adjusted closing prices are showing in blue line, whereas the train predictions (orange) and test predictions (green) are visualised on top. Model uses actual values from the dataset to make predictions.

Figure 8 provides a visualization of the model's performance on the QQQ data-set. On one hand the blue line represents the actual adjusted closing prices of QQQ. On the other hand, the orange line illustrates the plot generated from the training set, comprising 80% of the data points, while the green line depicts test predictions based on the remaining 20% of the data points.

The aim of the model is to predict the ETF's next day adjusted closing price for just one day and not an extended period time in future. Therefore, the model is using the actual values from the dataset to predict the next day's adjusted closing priced instead of using the predicted values for making further predictions. Therefore as it can be seen in Figure 8, the simple feedforward neural network has demonstrated substantial success, delivering close predictions of the actual adjusted closing prices for QQQ.

5 Conclusion

In conclusion, this study has investigated into the performance of a feedforward neural network for predicting the adjusted close price of the QQQ ETF. The study involved the formulation, training, and evaluation of a neural network model using historical data.

The results obtained from the model conducted are promising. The neural network demonstrated relatively decent predictive capabilities, capturing more general patterns and relationships within the QQQ ETF data. The training process allowed the model to learn from historical trends and make relatively accurate predictions on the unseen test data.

Even though the performance is encouraging, it is essential to acknowledge certain limitations and clarifications. Model's complexity, such as hyper-parameters, architecture, and data-pre-processing, all play crucial roles in the neural network's effectiveness. Further research could explore optimizing these aspects, for a better a capture of long-term dependencies and more accurate predictions. In addition, the model is designed to predict only a single day in the future, reducing the risk of error accumulation over time. Figure 8 should not mislead the reader to think that the model can accurately predict the adjusted closing price of the ETF over an extended period of time. For such purposes, a different model should be designed, so that the predicted values are used for making further predictions.

The study aims on learning and applying machine learning techniques to financial forecasting. The feedforward neural network did a good job predicting the QQQ ETF's closing prices. This shows that these models can understand to some extend complicated patterns in financial data.

As we move forward, continued exploration of advanced neural network architectures, and fine-tuning of model parameters will likely perform bet-

ter. The findings of this thesis could be an inspiration for future research studies in the domain of financial forecasting using machine learning techniques.

Bibliography

[1] Invesco qqq trust (qqq). Retrieved from Yahoo! Finance. Accessed on 6 April 2022.

[2] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[3] Akash Ajagekar. Adam, 2021. Accessed on 12 December 2023.

[4] Daniel Berrar. Cross-validation, 2019. Accessed on 20 December 2023.

[5] Jason Brownlee. What is the difference between a batch and an epoch in a neural network. *Machine Learning Mastery*, 20, 2018.

[6] Jason Brownlee. How to use learning curves to diagnose machine learning model performance, 2019. Accessed on 12 December 2023.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, Massachusetts; London, 2016.

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] Anders Krogh. What are artificial neural networks? *Nature Biotechnology*, 26(2):195–197, 2008. Accessed on 11 October 2023.

[10] Jiří Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy. *arXiv preprint arXiv:1710.10686*, 2017.

[11] Thomas Lee, Greta Gasswint, and Elizabeth Henning. Momentum. `https://optimization.cbe.cornell.edu/index.php?title=Momentum`, 2021. Accessed on 14 December 2023.

[12] Kevin Patrick Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[13] Robert Nau. Stationarity and differencing of time series data. `https://people.duke.edu/~rnau/411diff.htm`, 2020. Accessed on 03 January 2024.

[14] Adonis Pelagia. Feed-forward neural network. `https://github.com/AdonisPe/feed-forward-neural-network`, 2024. Accessed on 15 January 2024.

[15] Aditya Prasad. The bias-variance trade-off: Explanation and demo, 2019. Accessed on 08 November 2023.

[16] Lutz Prechelt. *Early Stopping—But When?*, pages 55–69. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[18] Ivan Zakharchuk. Generalization, overfitting, and underfitting in supervised learning. Medium, 2021. Accessed on 02 November 2023.

Kandidatuppsats 2023:24
Datalogi
Januari 2024

www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm