UNIVERSITY OF GOTHENBURG
SCHOOL OF BUSINESS, ECONOMICS AND LAW

Financial Economics

Predicting Asset Prices with Machine Learning

Bachelor thesis 15 credits

Authors: Adam Eklund & Valter Trollius

Supervisor: Marcin Zamojski

Spring term 2020

Abstract

This study examines whether machine learning techniques such as neural networks contain predictability when modeling asset prices and if they can improve on asset pricing prediction compared to traditional OLS-regressions. This is analyzed through measuring and comparing the out-of-sample R^2 to find each models' predictive power. Furthermore, we establish the loss metrics of root mean squared error and mean bias error to assess model strength. A sample of Swedish stocks ranging over a 40-year period is considered the dataset.

We provide an analysis of various models to find indications of which models perform better from an economic viewpoint. Although we do not test for statistical significance, as forecasting returns infrequently exert this, the economic gains can prove relevant. We find that several neural networks outperform linear OLS regression in terms of out-of-sample R^2. We believe that this might not be enough information to profitably transact upon as a considerable number of factors such as transaction costs are still unaccounted for. Our conclusion is therefore that further studying is required to fully allow for all factors to be considered.

Keywords: Machine learning, neural networks, OLS regression, asset pricing, financial forecasting, out-of-sample, predictability.

JEL Classifications: C45, G12, G17

Acknowledgement

We would like to thank our supervisor Marcin for the continuous support and encouragement during the writing stage as well as aiding us in learning of the concept of machine learning. We also want to thank the students that have provided us with useful hints and guidance through the period of constructing the thesis.

TABLE OF CONTENT

1 Introduction 1
1.1 Problem definition 2
1.2 Purpose of the thesis 2
1.3 Research Questions 3
1.4 Thesis Structure 3
2 Literature review 4
2.1 The aim of research within the field 4
2.2 Existing research 5
3 Method 8
3.1 Machine learning 8
3.2 Neural Networks 9
3.2.1 Architectures 10
3.2.2 Hyperparameters 12
3.2.3 Overfitting 12
3.2.4 Regularization 13
3.2.5 Data split 13
3.3 Comparison of models 15
4 Data 16
4.1 Valuation ratios 16
4.2 Technical variables 17
4.3 The final dataset 18
5 Result & Discussion 19
5.1 Base run 20
5.2 Impact of dropout on model performance 21
5.3 Impact of number of lags 22
5.4 Impact of changing lags and dropout 24
5.5 Final Analysis 25
5.6 Limitations and further extensions 26
6 Conclusion 28
7 References 29
8 Appendix 31
8.1 The Data preperation code 31
8.2 Building the different Machine Learning Models 33

1 INTRODUCTION

During the past decades, more advanced forecasting strategies have allowed for slight economically and statistically significant out-of-sample gains (Rapach and Zhou, 2013). Forecasting future returns on assets demands that the used methods are capable of modeling complex patterns which can be difficult for traditional linear models. To predict future events, data sets are analyzed with specialized models to determine a likely outcome. Timmermann (2018) discusses the difficulties of forecasting asset prices, where sufficient market efficiency leads to a low signal-to-noise ratio. This in turn contributes to low predictive power of the models.

An area that can be used for prediction that has been researched in more detail in the past decade is machine learning. The use of machine learning has spread throughout numerous areas with its capabilities of pattern recognition and quick sorting of big data. Machine learning applied on financial asset pricing models is something that is relatively new, albeit successful in that it provides models that better predicts returns and have lower degree of error (Gu, Kelly and Xiu, 2019b). Machine learning can supposedly improve on this using complex structures and nonlinear conditional estimations of the factors (Gu, Kelly and Xiu, 2019a).

We contribute to the field by performing analysis of machine learning methods in new environments which further strengthens the assumptions that machine learning methods improve predictability of stock returns (Gu et al., 2019b). Our study also provides more evidence to deepen the empirical understanding of asset pricing. We evaluate the current literature to base our subsequent methodology on which enhances the comparability between our study and existing studies.

Our findings are that several of the examined machine learning methods called neural networks achieves predictability (positive out-of-sample R^2) when predicting returns. We also conclude that several of the examined neural networks outperform linear OLS regression for all considered out-of-sample horizons. The prominent feature of this study is that it uses Swedish-listed stocks as the dataset which serves to further expand and deepen the field.

1.1 PROBLEM DEFINITION

Asset pricing is notably difficult as it varies with unforeseeable events. The measurement of an assets price is a fundamental financial problem since it requires prediction because the price is a conditional expectation of a future realized excess return (Gu et al., 2019b). To predict stock returns, it is required to have adequate information of the asset's characteristics as well as knowledge of various theoretical aspects that covers the topic. Linear models face difficulty when the data follows complex or nonlinear patterns. This poses an even larger problem when the number of variables becomes extremely high (high-dimensional). In recent literature, machine learning has been utilized to improve on this with neural networks that allows for more complex modeling (Gu et al., 2019b).

Predicting stock prices is hard because of high competition among the stakeholders which leads to market efficiency (Timmermann, 2008). The different variables that explain the rate of return of stocks are not only huge in numbers but are also difficult to measure because of the physical, psychological, rational and irrational aspects of the stock market. It is therefore of interest to evaluate the models applied by machine learning to determine whether there is any benefit in using these when predicting stock returns. Studies of Gu et al. (2019b), Gu et al. (2019a) and Chen et al. (2020) have been conducted on U.S. markets to determine the comparative strength of machine learning models and which factors that are significant. There is, however, a lack of studies done in a similar manner to that of Gu et al (2019b) and Gu et al (2019a) done outside of the U.S.

1.2 PURPOSE OF THE THESIS

In this thesis we analyze whether the field of machine learning can be applied to financial asset pricing and if the predictive abilities of nonlinear models can be an improvement over other methods such as linear OLS-regression. The reason of our study is to further add research to the field to increase understanding of asset pricing. We will proceed in a similar manner to how previous studies such as Gu et al. (2019a) and Gu et al. (2019b) have been performed with the exception that we focus on the Swedish stock market. To achieve better predictive abilities of the models, we use neural networks from the field of machine learning.

1.3 RESEARCH QUESTIONS

The aim of the study is to examine whether machine learning techniques such as neural networks can improve upon traditional linear OLS regression models. The research questions that are set up are hence:

(I) ***Is it possible to achieve predictability on stock prices through the use of neural networks?***

(II) ***Can neural networks outperform linear regression when predicting stock prices?***

The research questions will be answered by calculating the out-of-sample R^2 for the neural network models and if a positive sign is obtained then predictability is achieved. This predictability will infer that it is possible to earn excess economic gains compared to the benchmark, which in this case is the historical mean return for the used dataset. Measurement metrics of mean bias error and root mean squared error will used to dissect the comparison among neural networks and linear OLS regressions.

1.4 THESIS STRUCTURE

The remainder of the thesis is structured in distinctive sections. In section 2, we analyze the current literature in the field of machine learning applied on asset pricing. In section 3, we present key concepts of machine learning and, in detail, describe how we proceeded with the comparison between the machine learning method of neural networks and linear OLS regression. In section 4, we present the data, how it was handled and an illustration of the explanatory variables. The fifth section covers the found results of our study as well as an analysis of the impact of the explanatory variables and the hyperparameters. The sixth and final section comprises of our thoughts and conclusions of the study.

2 LITERATURE REVIEW

The two strands of asset pricing and machine learning have their own respective fields of studies. The literature that combines the two is relatively new and there is therefore a limit to the amount of published literature that exists. In this section we present the combined strand of machine learning applied on asset pricing, their supposed problems and incorporate relevant articles to provide a foundation for which later sections builds upon.

Generally, forecasting problems are difficult to handle as the number of possible scenarios often are very large. In financial theory this is likely even harder due to the competitive nature of asset prices (Timmermann, 2018). Campbell and Thompson (2008) discuss the use of out-of-sample measurements for the R^2-statistic for financial problems. It is shown that by using restrictions on the predictive regressions, the out-of-sample R^2 can be improved which could lead to investors profiting by using timing strategies. Meanwhile, Timmermann (2008) discerns that any return predictability found is expected to deteriorate quickly.

2.1 THE AIM OF RESEARCH WITHIN THE FIELD

The main reason for the research of machine learning applied on asset pricing is to deepen the understanding of asset pricing and to provide researchers and investors with tools to improve navigation on financial markets. The general results from literature within the field is that machine learning methods such as regression trees and neural networks enhance modeling capabilities due to the incorporation of nonlinear functions and more efficient computing.

There are two branches that needs to be distinguished in the field of machine learning on asset pricing. One uses predictive modeling in which forecasting techniques are used to find patterns and infer future outcome while the other uses contemporaneous modeling in which data mining tools are used to gain insight of past events. To conjoin the areas of machine learning and asset pricing researchers have chosen models and methods that they believe will address the issue at hand. There exists no consensus of how the different techniques and models of machine learning should be used in the asset pricing spectrum. It has, however, been determined that machine learning is particularly useful when assessing asset pricing due to the problem being predictive by nature (Gu et al., 2019b). The predictive branch, which is emphasized in this thesis, is concerned with what models and techniques to use to achieve models that perform better compared to existing ones.

One of the core issues with modeling asset prices is the ability to handle model complexity and the associated risk of overfitting (Gu et al., 2019b). The aim is to produce models that generalize well and thus research focus on what tools to use to mitigate overfitting. Additionally, some research focus on finding what variables explain return and to what degree. This is done to give researchers and investors a better understanding of which stock characteristics are the most important. The results within this area have improved due to the implementation of machine learning techniques. This is because of the computational power needed being substantial due to the high number of variables as well as the ability to model when the variables are highly correlated (Gu et al., 2019b).

2.2 EXISTING RESEARCH

Compared to other machine learning fields such as medicine and image recognition, there exists relatively few studies on the topic of asset pricing. Gu et al. (2019a) discuss how to improve models by incorporating an unsupervised neural network that aims to reduce dimensions and condition asset returns on the information of the stock characteristics. This is done to estimate the exposure factors have towards stock characteristics. The authors criticize the use of the linearity assumption when modeling asset prices and instead discuss how nonlinear models are more flexible and therefore provide better mapping of the asset prices. The authors conclude that their nonlinear autoencoder model outperforms linear Fama and French and Principal Component Analysis (PCA) models in terms of Sharpe ratio.

Similarly, Gu et al. (2019b) analyzes the effect of using machine learning methods to measure risk premia. The authors perform a comparative analysis among various methods to evaluate differences and performance. An analysis of what variables are most dominant in explaining return is also run. The authors present a benchmark for the accuracy of measuring risk premia in the market and in individual stocks and this is ranked by high out-of-sample R^2. Gu et al. (2019b) demonstrates that machine learning forecasts can be used by investors to extract economic gains. This is established by acquiring a higher annualized out-of-sample Sharpe ratio compared to a buy-and-hold strategy.

Meanwhile, Chen et al. (2020) proposes a general nonlinear conditioning model called a long short-term memory recurrent neural network to be used for modeling asset prices with a set of stock characteristics and a sizeable number of macroeconomic variables. The authors claim that the crucial innovation of the article is the usage of the condition of no-arbitrage

integrated into the neural network algorithm which supposedly improves risk premium indication and explanation of individual stock returns.

The three articles mentioned above share similar features. The data used in Gu et al. (2019a) and Gu et al. (2019b) is identical and uses U.S. listed firms from 1957 to 2016. The data in Chen et al. (2020) is U.S. listed firms from 1967 to 2016. In the work of Chen et al. (2020), the study of Gu et al. (2019b) is referenced many times and thus it is likely to believe that they share similarities. What differs slightly is the explanatory variables where Gu et al. (2019a) and Gu et al. (2019b) uses 94 stock characteristics and 8 macroeconomic variables while Chen et al. (2020) uses 46 stock characteristics and considerably larger set of 178 macroeconomic variables. A technical detail that differentiates Gu et al. (2019a) and Gu et al. (2019b) from Chen et al. (2020) is the structure of the neural network where Gu et al. (2019a) and Gu et al. (2019b) choose to focus on the more basic "feedforward neural network" while Chen et al. (2020) opts for the slightly more advanced "long short-term memory recurrent neural network".

Potential risks with the current literature of machine learning on asset pricing is that since the topic is relatively new, there is limited number of available models and methods that are proven to work. There is also a limited number of authors on the subject which could induce a narrow point of view and group thinking. Transaction costs are not mentioned in the studies of Gu et al. which could affect the credibility of their assumptions that machine learning improves economic gains. While it might be appropriate to neglect transaction costs due to the central aim of the studies being to provide understanding of asset pricing rather than proving economic gains, it can still be of importance when, for instance, comparing market timing Sharpe ratio gains.

The next field that we analyze is the theory of financial forecasting. The literature by Timmerman (2018) describes important features that differentiates economic and financial forecasting. The article demonstrates three features that needs to be considered when discussing financial forecasting. First, the competitiveness and the market efficiency leads to a low "signal-to-noise" ratio in many financial forecasting problems. This is particularly problematic when predicting asset prices, as opposed to standard macroeconomic prediction problems. The presence of weak predictors and parameter estimation errors are therefore crucial to examine when dealing with financial forecasting. Second, model instability is important in financial forecasting as the magnitude of the outcome is stronger in finance than

in other areas (Timmermann, 2018). According to Timmermann, this is because of the high competition between asset managers and investors trying to exploit mispricing of assets. The third features discussed by Timmermann are issues of overfitting and data mining. The problem is to find a truly independent dataset on which to test the forecasting performance on.

Furthermore, Timmermann (2018) discusses limitations and challenges in the area of financial forecasting such as data-limitations, weak predictors, persistent predictors, model instability and data mining. From the perspective of variable selection, inclusion of predictors in regressions of return is considered to hold uncertainty that is not likely to be resolved by model selection methods (Timmermann, 2018). Persistent predictors are a problem due to some valuation ratios such as the dividend yield being highly persistent and correlated with unexpected shocks to returns. The limitation of model instability is due to the predictors changing during the timeframe of the gathered data. Asset returns depend on prices which in turn reacts to expectations of future payoff. These limitations further explain why financial forecasting is both challenging and fascinating. According to Timmermann (2018), the financial payoff is what motivates researchers to uncover even small increases in predictive power of models as this potentially yields large economic gains.

3 METHOD

Under this section we present key concepts of machine learning and relate to econometric terminology to ease the understanding for those with a financial background. We present machine learning concepts in a way that it can be understood without prior knowledge of the subject. We will provide information of what methods and models we use as well as an explanation of the comparison among them. Finally, the data and the data treatment will be covered.

As discussed under the previous section, it is important to make the distinction between predictive or contemporaneous models. In this research we construct a predictive model due to the aim being to determine how much a machine learning model will influence the prediction of stock prices. By looking at data points from the past and finding patterns and key trends within the fixed variables, we expect that the model will give an estimate to how well the machine learning method can perform.

We proceed in a similar manner to prior literature of Gu et al. (2019a) and Gu et al. (2019b) and hence our choice of method will also overlap and be influenced by these studies. By applying machine learning on complex nonlinear models, we expect to receive a higher coefficient of determination, R^2, and smaller mean bias errors and root squared mean prediction errors for predicting future stock returns compared to the linear model. The machine learning models will be constructed in Python. The package that is used for the neural network is called Tensorflow. By using the high-level neural network API called Keras on top of Tensorflow, it is possible to create the sequential model with its layers, which will be discussed in detail further ahead. The package Tensorflow with the API Keras needs input data that has been structured, reshaped and split by certain requirements.

3.1 MACHINE LEARNING

Machine learning uses computationally efficient algorithms to perform tasks without the need for outside intervention. The concept is related to econometrics with a few key differences. While econometrics is interested in finding causal correlations, machine learning instead focus on finding the optimized fit for a model to induce predictions (Chen et al., 2020). Both methods, however, utilize regression as the main tool to produce results. Machine learning methods use training to learn from data sets. The models used in our study utilizes a training

method called supervised learning. When the model uses supervised learning, it will form inferred assumptions based on input and output data. There is also unsupervised training, in which the model will learn to infer assumptions of patterns only based on input data (Gu et al., 2019a). There are certain models of neural networks that utilize unsupervised learning, but this will not be considered further in this study.

3.2 NEURAL NETWORKS

The foundation of a neural network is the neuron. The neuron can be interpreted as a univariate regression with a coefficient and an intercept. The intercept is known as the neurons bias and the coefficient is called a weight. The neuron alters the information it receives as input and yields this altered information as the output (Gu et al., 2019b).

$$z_n = \sum_{i=1}^{n} (l_n * w_n) + b_n$$

In the equation, z_n is neuron n, $\sum_{i=1}^{n}(l_n * w_n)$ is the summation of the neurons in the previous layer times the coefficient (weight) that connects it to neuron n. Finally, the bias b_n for neuron n is added.

The neurons reside within layers. There are three distinctive layers in a neural network which are the input, hidden and output layers respectively. The layer can be represented as a multivariate linear regression of the neurons where multiple variables are considered when modeling return. Initially, input data is entered into the network via the input layer. In most networks no computations are executed here, the input layer will only deliver the information to the first hidden layer. Finally, an activation function, such as ReLU, Sigmoid or Gaussian, is applied in each neuron to possibly add non-linearity to the process (Gu et al., 2019b). The activation function normalizes each neuron between a range of values, usually between 0 and 1. The hidden layers vary by number and can generally be viewed as mathematical functions that receives input from the prior layers and generates a new number through the activation function (Gu et al., 2019b).

The specific activation function we emphasize is called the rectifier, or ReLU. The prominent feature of ReLU is the threshold value that determines whether the neurons gets activated or

not. Because of the threshold value, only a certain combination of neurons are activated in the output layer. This has advantages of decreasing the dependency of single neurons within the network which increases the speed of the model as well as decreases the problem of vanishing gradients.

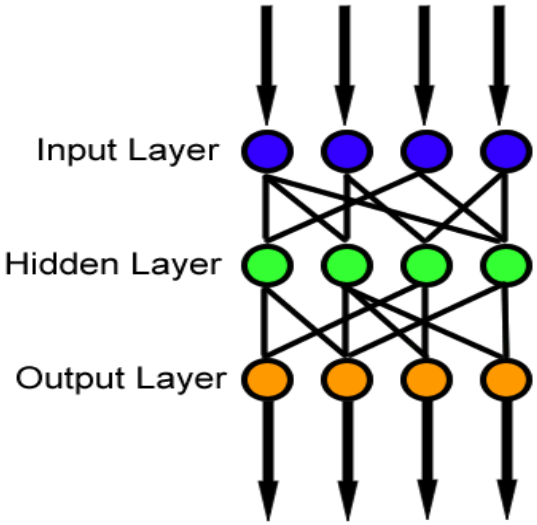
A common learning technique that is initialized in the event of the process reaching the wrong conclusion is backpropagation (Gu et al., 2019b). This follows the basic econometric concept of optimizing a criterion function. Backpropagation will shift the coefficients and biases slightly to increase its chances of being correct. This is usually done through gradient descent which computes the derivative of the coefficient's loss function at each step. If the error term is not at a local minimum the error term will be lowered by taking a step in direction where the slope is the steepest. The size of the gradient descent is called the learning rate and it is selected as a hyperparameter by the input. The learning rate is further optimized by empirical testing to determine which model optimizes the loss function the quickest (Gu et al., 2019b). The calculations are executed starting from the output layer to then proceeds into the hidden layer(s) and finally to the input layer. Problems that arise with the use of backpropagation are vanishing gradients and exploding gradients. Backpropagation calculates the partial derivative of the value received from the activation function, which proves troublesome when the activation function values approach either the minimum or the maximum. When this is the case, the partial derivative becomes very small or very large. If multiple neurons partial derivatives of the activation function take on extreme values, early layers will not receive correct tuning. Solutions to these problems include changing the architecture or the activation function.

3.2.1 Architectures

The structures of neural networks are denoted as architectures. The most common type of neural network is a feedforward neural network. This is also the simplest form of neural network in that it follows a one-way flow of information and does not contain any loops or cycles (Gu et al., 2019b). The information flows from the input layer to the hidden layer(s) and subsequently to the output layer. The different layers are connected through various coefficients (weights) which are random at first but will later variate as the model is refined through backpropagation. The neurons bias functions similarly to the coefficient in that it is refined over time to yield preferable results. The difference is that while the coefficients functions as parameters that are multiplied by the input, the bias is as explained earlier, an

intercept. As time goes (or as training proceeds) the coefficients will be tuned to improve the predictions made by the model (Gu et al., 2019b). This is the basics of how a feedforward neural network is being optimized. The architecture will form a passage in which layers in front will be a linear or nonlinear function of the layers in the back. When the neural network contains more than one hidden layer it is considered deep which allows for more complex modeling.

Figure 1: Feedforward neural network.



The figure shows a feedforward neural network with 1 hidden layer (non-deep),4 input neurons, 4 hidden neurons and 4 output neurons, connected by coefficients.

A broader, more complex group of architectures are recurrent neural networks. This group is more adept compared to feedforward neural networks in drawing conclusions of past events due to a temporal dimension (Chen et al., 2020). This dimension is integrated by having a feedback loop that attaches to the hidden layers and is often analogously considered a memory. Because this network can draw inferences built on past data it excels in modeling patterns of sequential data. A subgroup to the recurrent neural network is the long short-term memory neural network (LSTM). The LSTM neural network contains gates which restricts what values of the neurons can pass and therefore modifies the flow of information. According to Chen et al. (2020), LSTM are among the most successful machine learning methods for modeling sequences of data and thus ideal for predicting stock returns.

3.2.2 Hyperparameters

The optimal value for certain parameters in the model is not known beforehand. These are called hyperparameters and are to be determined prior to running the models. Hyperparameters define the architecture and are manually adjusted to find the optimal value (Gu et al., 2019b). The optimal value for the hyperparameters are in most cases the one that optimizes the measurement metrics, however, the speed of the convergence of the model can also be of interest. The hyperparameters that we determine iteratively are the number of hidden layers, the number of neurons in the hidden layers, the optimizer algorithm, the learning rate of the gradient descent, the out-of-sample horizon, the dropout rate, the batch normalization and the number of lagged days on the explanatory variables.

The number of hidden layers effect on model performance is often debated and recent studies has found ambiguous results (Gu et al., 2019b). On one hand more layers lead to a level of higher prediction with fewer regressors which improves results. This, however, comes with a higher need for computational capacity and due to the exponential nature of neural networks, it quickly becomes tough to accomplish. The primary distinction to make is between one and multiple hidden layers as mentioned previously. The exact number of hidden layers that we aim to use will be decided based on empirical testing on which models perform better and converges to the minimum the fastest. Another similar issue that arises is how many neurons should be within each hidden layer. The number of neurons depend on the type of layer (Gu et al., 2019b). In the input layer the number of neurons should equal the number of dependable variables in the data. The output layer should have as many neurons as there are outputs. The challenging part is to decide how many neurons should be in the hidden layers. As with the number of layers, the most common approach to evaluate how many neurons to use include in the hidden layers is to systematically test what number of neurons perform better (Gu et al., 2019b). The optimizer algorithm defines the model's training attributes (Gu et al., 2019b). Through empirical testing we conclude that the preferred optimizer algorithm for us is the Stochastic Gradient Descent, or SGD, which is the most commonly used optimizer in neural networks (Gu et al., 2019b).

3.2.3 Overfitting

When predicting asset prices, it is critical to understand the concepts of bias and variance. The trade-off between the two shows the risk of having a too simple model (high bias) or a too complex model (high variance). Bias and variance are a sign of the model over- or

underfitting. The most recurrent problem within machine learning is overfitting as the issues at hand often contain complex structures which leads to high variance (Gu et al., 2019b). This phenomenon occurs when the model puts too much emphasis on noise, or randomness, due to high complexity in the underlying data. The main symptom of overfitting is when the model performs adequately on the training data but performs considerably worse on the test data. To circumvent this issue, one must either alter the data, alter the architecture or add regularization (Ying, 2019). To reduce the problem of overfitting and further improve predictability we introduce regularization techniques. Generally, regularization comes in the form of penalizing certain unwanted characteristics in parameters or protect against outliers. This is done to ensure that the model disregards extreme values that only fit the training data.

3.2.4 Regularization

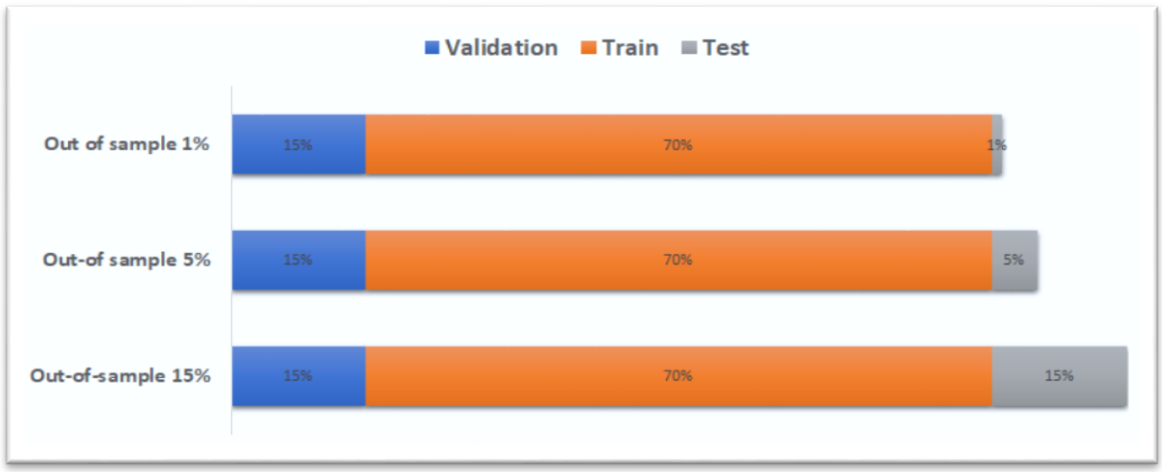
To reduce the problem of overfitting and improve speed and performance of the model we impose regularization techniques (Gu et al., 2019b). The regularization techniques we implement are early stopping, batch normalization and dropout. The dropout rate defines a percentage value of random neurons that will be ignored during the training stage. These neurons impact will be removed for the duration of one iteration and the layers will hence contain a different number of active neurons at each stage. This solves the issue of co-adaptation which occurs when neurons become too reliant on the input values they receive from neurons in prior layers (Srivastava, Krizhevsky, Hinton and Sutskever, 2014). The model becomes less prone to the risk of specific weights of individual neurons having too much importance. Early stopping is a technique that stops the process once the validation error increases from an iteration to the next. Batch normalization normalizes the number received from the activation function which makes the model converge to the early stopping faster and relieves the issues of vanishing and exploding gradients. Batch normalization also synergizes with dropout because it contains noise and thus less information will be dropped by the dropout function (Gu et al., 2019b).

3.2.5 Data split

The data containing both the explanatory and the dependent variables must be split in three distinctive parts. This is done to measure out-of-sample performance of the model as well as to prevent overfitting (Xu and Goodacre, 2018). The three sets are the training dataset, the validation dataset and the test dataset. The allocation is 15% as validation set, 70% as training

set and either 1%, 5% or 15% as test set. The test set varies to test predictive power of the models on different test set sizes (out-of-sample horizons).

Figure 2: Splitting the dataset in different out-of-sample sizes



This figure shows how the data set is split and how different out-of-sample horizons effects the data-splitting. The table demonstrates that the out-of-sample horizon is the size of the test data set.

The training dataset is the actual dataset that is used to train the program and the aim for this is to make the model learn what input variables will induce a certain output variable. The validation dataset is separate from the training dataset and is used to validate the model. The validation process is defined by training the model to improve hyperparameters and predictability independently from the training dataset (Xu and Goodacre, 2018). The major difference between the training and the validation processes is that the validation does not alter the coefficients or the biases of the neural connections. It instead aims to provide an unbiased control that any increase in accuracy over the training data will have the similar effect when used on data that has not yet been shown to the model. This aids in protecting the model from overfitting. The third and final separate dataset is the test dataset. The objective of the test dataset is to provide data that can be used to measure the models out-of-sample performance. This is done at the end of the process to fully reflect what the model has learned. The major difference between the training/validation data and the test data is that the test data is not used for any tuning or training of the model. All datasets should also be following the same probability distribution (Xu and Goodacre, 2018).

3.3 COMPARISON OF MODELS

To give a just estimation of how the neural networks perform, we analyze and compare them with a more simplistic model. The model of comparison is the standard linear OLS regression which follows standard OLS regression rules. We expect that the linear OLS model will perform poorly when used to predict future stock returns, which is in line with previous results in Gu et al. (2019b). This is likely due to the OLS becoming unstable when the number of predictors in the model increases. The neural networks that we model differ with regards to their hyperparameters which affects the out-of-sample predictive performance.

To evaluate model performance, two loss functions are measured on an in-sample and out-of-sample basis. The coefficient of determination, R^2, is measured on an out-of-sample basis on test data for the neural networks and on an out-of-sample basis on training data for the OLS-function as comparison. This is done to determine whether the model exhibits predictability. Campbell and Thompson (2008) discuss the use of out-of-sample measurements for the R^2-statistic for financial problems. Improving out-of-sample R^2 is shown to lead to investors profiting by using timing strategies. If a positive out-of-sample R^2 is found to be consistent through time, then it can be concluded that the model outperforms the random walk (Gu et al., 2019b). Any return predictability found however, is expected to deteriorate quickly (Timmermann, 2008).

It is of high relevance for the neural networks to balance bias and variance in order to receive a model that does not over- or underfit. The first loss function used is the root mean squared error, RMSE, which is a commonly used evaluation of the quality of the estimator. The RMSE is the root of the average of the squared differences between the predicted value and mean value. It encompasses both by how far off the average data point is from the actual value and by how much the data points are spread out from each other. The second measure to evaluate model performance is the mean bias error, MBE. MBE is used to capture the average bias in the model. A positive MBE indicates that the predicted values are larger than the observations and vice versa. The bias is important to determine to find the optimal structure of the neural network. The optimal neural network will hence minimize both RMSE and MBE while having a positive out-of-sample R^2 (Gu et al., 2019b). The measurement metrics are calculated as follows:

$$R^2_{oos} = \frac{\sum_{t=1}^{T}(y_i - \hat{y}_i)^2}{\sum_{t=1}^{T}(y_i - \bar{y}_i)^2} \quad MBE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \quad RMSE = \sqrt{(\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2)}$$

4 DATA

To receive as accurate models as possible we use the broad Swedish index together with a wide time span. The data used is 40 years of individual stock returns gathered from the Swedish House of Finance with a period ranging from January 1979 to January 2019. There are 491 stocks in total that were all listed on the Swedish exchanges. Explanatory variables in the form of stock characteristics are also part of the dataset. These are collected from the same source and are chosen based on what previous studies have found to be the dominant factors.

The descriptive variable in the dataset is the logarithmic value of the daily return. The explanatory variables are lagged values of the predictors. The predictors chosen for our model is categorized in two groups which are technical analysis variables and valuation ratios. This stems from the conclusion made by (Gu et al., 2019a) that the most successful predictors are price trends, liquidity, volatility and financial ratios.

4.1 VALUATION RATIOS

The valuation ratios that we apply as explanatory variables in the dataset are the market and the book value rankings. These rankings will determine the relative size of companies and also give signals of increases or decreases in the ranking between companies. A company's market value, also known as market capitalization, is the total value of all the traded shares on the market. The book value is defined as a company's tangible assets minus its liabilities. The purpose of these ratios is to assist the neural network in finding relationships between increases or decreases in the rankings and the stock return. The last valuation ratio is the dividend yield. There have been many studies of the relationship between the dividend yield and the stock return. Timmerman and Pesaran (1994) built a model that explained monthly S&P 500 returns with lagged values of the dividend yield and showed that the variable is significant for predicting stock returns. We consider these predictors in our dataset due to previous research showing their predictive power (Gu et al. 2019a).

4.2 TECHNICAL VARIABLES

Apart from the valuation ratios, we also implement technical analysis variables and lags of the logarithmic daily return in the dataset. We implement technical variables due to previous literature showcasing their statistical significance when forecasting returns (Gu et al., 2019b) In our dataset we include the Moving Average Convergence Divergence (MACD) which is a metric used to find trends in momentum between the exponential moving average (EMA) with a period of 26 days with the EMA of 12 days. The MACD is calculated by subtracting the fast EMA with the slow EMA. The next technical metric we include is the relative strength index (RSI) which determines whether a stock is overbought or oversold compared to a rolling average of the last 14 days. The metric ranges from 0 to 100 where a number in excess of 70 indicates an overbought stock and a number lower than 30 indicates an oversold stock. These technical indicators have shown to work as predictors and could potentially generate excess return for investors (Chong and Ng, 2008). Furthermore, a lagged variable of Simple Moving Average (SMA) is implemented, which comprises of the average returns over a certain period. The dataset also contains the SMA-crossover variable, which is defined as a slow rolling window of 100 days and a fast rolling window of 20 days. The final technical variable we add to the dataset is the bid-ask spread. The bid-ask-spread is the difference between the ask price and the bid price and is an indicator of supply and demand of an asset. Research have shown that higher spreads yield higher returns and is therefore interesting to add to our model (Amihud & Mendelson ,1986)

4.3 THE FINAL DATASET

Table 1 demonstrates the numerical summary of the predictors that are included in the dataset. Winsorization is conducted on each of the variables to reduce the effect of outliers. This is done by choosing a certain threshold value of the distribution, in which the excess gets replaced by a normalized percentile. In table 1, all variables range from 0 and 1 with the exception of the return variable, which is only winsorized but not normalized. This is because we do not want smaller standard deviations to suppress the effect of outliers in the case of daily returns.

Table 1: Predictor summary

Variable	count	mean	std	min	0,25	0,5	0,75	max
Return	961319	1,000	0,022	0,885	0,990	1,000	1,010	1,128
Market value	961319	0,342	0,218	0,000	0,166	0,324	0,486	1,000
Book value	961319	0,341	0,219	0,000	0,162	0,323	0,487	1,000
Dividend yield	961319	0,030	0,071	0,000	0,005	0,010	0,026	1,000
SMA crossover	961319	0,272	0,445	0,000	0,000	0,000	1,000	1,000
Volatility	961319	0,025	0,013	0,000	0,017	0,022	0,029	0,655
RSI	961319	0,523	0,124	0,000	0,442	0,523	0,607	1,000
MACD	961319	0,045	0,012	0,000	0,045	0,045	0,046	1,000
Bid- Ask-Spread	961319	0,062	0,128	0,000	0,009	0,023	0,055	1,000

The table shows the summary statistics of the predictors implemented in the dataset after winsorization and normalization.

5 RESULT & DISCUSSION

The result from the evaluation of each neural network after tuning hyperparameters and constructing different architectures will be presented in this section. The following tables will demonstrate the in-sample and out-of-sample metrics of the MBE, RMSE and the out-of-sample R^2. To demonstrate how the out-of-sample horizon changes the return predictability, results with three different horizons of 1%, 5% and 15% of the test set will be shown. A comparison between the OLS regression will also be demonstrated to showcase the relative strengths or weaknesses of different neural networks.

To achieve a relatively well-performing neural network it is crucial to examine the results of different setups and architectures. This section demonstrates ten different architectures that we denote NN1 to NN10, ranging from shallow (non-deep) to very deep in ascending order. The architectures are presented in table 2 and are defined as the number of neurons times the layers. For instance, the first neural network, NN1, is made up of 32 neurons in the input layer and 16 neurons in the hidden layer. The output neuron is always 1 due there being only one output and is therefore not shown in the table. By using the same architectures throughout the result section, we draw conclusions of correlation between the changes of the hyperparameters and the performance metrics.

Table 2: The Neural Network Architectures

Network	**Neurons in input layer**	**Hidden Layers (Neurons x Layers)**	***Type***	***Optimizer***	***Batchsize***
NN1	32	16x1	Shallow	SGD	500
NN2	64	32x1	Shallow	SGD	500
NN3	128	64x1	Shallow	SGD	500
NN4	256	128x1	Shallow	SGD	500
NN5	512	256x1	Shallow	SGD	500
NN6	128	64x2	Deep	SGD	500
NN7	128	64x2, 32x2, 16x2	Deep	SGD	500
NN8	256	128x3, 64x2	Deep	SGD	500
NN9	256	128x3, 64x3, 32x3	Deep	SGD	500
NN10	512	256x4, 128x4, 64x4, 32x4, 16x4, 8x4, 4x4, 2x4	Deep	SGD	500

The table shows the architectures that are being tested in all subsequent runs. Each network has their own structure with regards to how many neurons exist in the input and hidden layers as well as how many hidden layers there are.

5.1 BASE RUN

This run will be used as our benchmark for tuning and analyzing the best setup for the neural networks. The predictors in this setup are 20 days of lagged variables, a dropout rate of 0.5 together with batch normalization on each layer. The result is presented in the table below where the performance metrics are displayed for the OLS and the neural networks. The networks out-of-sample performance metrics MBE, RMSE and R^2 were better than for the OLS. Looking at the different out-of-sample horizon of 1%, 5% and 15% an interesting finding can be noticed. When the out-of-sample horizon increases the deep neural networks appears less stable which indicates that it might overfit. By looking at the out-of-sample horizon of 1%, the best neural network is NN8 which shows the lowest mean bias error and the highest out-of-sample R^2. This is in line with results from previous studies of Gu et al. (2019a), Gu et al. (2019b) and Chen et al (2020), that demonstrates a higher out-of-sample R^2 for the neural networks and regression trees compared to the linear OLS regression. However, when the out-of-sample horizon increases to 15% NN8 turns from a bias of -0.089 to 0.425 and receives the lowest out-of-sample R^2 of all the networks. The effect occurs for all of the deep neural networks, showing that when the out-of-sample horizon increases, the shallow neural networks perform better. The best-performing model was NN5 with an out-of-sample R^2 of 0.276. This can be explained by studies of Gu et al. (2019b), where deeper networks usually are more prone to overfitting as the data is likely to contain much noise. Shallower networks are less affected by noise and could be the reasoning to their relative outperformance.

The out-of-sample R^2 might be statistically insignificant but could prove economically highly relevant since our benchmark is random walk. (Timmerman, 2018) mentioned as a counter to the difficulties with establishing return predictability, that even small amounts of return predictability have the potential of translating into significant economic gains. This is not to say that it would be a profitable to trade based on these results, since the model does not account for transaction costs and still has a relatively poor performance. However, the setup presented for the base run indicates that the neural network out-of-sample predictions are stronger than linear regression (OLS) as is expected from studies of Gu et al. (2019b).

Table 3: The Base Run

		In-Sample		*Out-of-sample 1%*			*Out-of-sample 5%*			*Out-of-sample 15%*		
	Model	**Bias**	**RMSE**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**
	OLS	**0,000**	**22,004**	-1,698	16,983	-7,563	-0,444	16,653	-3,847	**0,058**	17,684	-3,135
Shallow	NN1	0,031	22,056	-1,487	16,915	0,558	-0,300	16,623	-0,181	0,190	17,657	0,030
	NN2	0,068	22,055	-1,492	16,911	0,987	-0,315	16,622	-0,136	0,159	17,656	0,079
	NN3	0,077	22,055	-1,532	16,914	0,705	-0,357	16,622	-0,159	**0,125**	17,656	0,133
	NN4	0,036	22,053	-1,507	16,919	0,095	-0,349	16,624	-0,385	0,132	17,656	0,045
	NN5	0,081	22,053	-1,546	16,914	0,597	-0,364	16,622	-0,046	0,130	**17,654**	**0,276**
Deep	NN6	0,164	22,056	-1,403	16,904	1,838	-0,233	16,620	0,098	0,250	17,657	-0,010
	NN7	0,127	22,058	-1,431	16,908	1,334	-0,221	16,619	0,254	0,297	17,658	-0,187
	NN8	0,290	22,059	**-1,292**	**16,896**	**2,748**	**-0,089**	**16,618**	**0,435**	0,425	17,660	-0,414
	NN9	0,150	22,058	-1,408	16,906	1,573	-0,199	16,619	0,292	0,319	17,659	-0,230
	NN10	0,035	22,058	-1,525	16,916	0,374	-0,314	16,620	0,083	0,205	17,657	-0,042

The table shows the setup for the first test where the different networks are compared by the metrics of MBE, RMSE and OOS-R2 on an in-sample and an out-of-sample basis. The best performing network in the out-of-sample size 15% is NN5 with a value of 0.276. All values are multiplied by 1000 for easier interpretation.

5.2 IMPACT OF DROPOUT ON MODEL PERFORMANCE

The purpose of this run is to determine the impact of decreasing the dropout rate relative to the performance of the model. The dropout rate is therefore lowered to 0.2 instead of 0.5 while holding other parameters fixed. The result in table 4 shows that the R^2 of the best performing network in any of the out-of-sample horizons do not improve over the previous run. The best model in this run is NN1 with an out-of-sample R^2 of 0.178. We determine that the deep neural networks NN6, NN7 and NN8 receives a lower mean bias error for the 15% out-of-sample horizon compared to the base run. They also receive a positive out-of-sample R^2 for the out-of-sample horizon of 15%.

Table 4: Decreasing the dropout

		In-Sample		*Out-of-sample 1%*			*Out-of-sample 5%*			*Out-of-sample 15%*		
	Model	**Bias**	**RMSE**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**
	OLS	**0,000**	**22,004**	-1,698	16,983	-7,563	-0,444	16,653	-3,847	0,058	17,684	-3,135
Shallow	NN1	0,017	22,056	-1,546	16,919	0,077	-0,370	16,624	-0,383	0,110	**17,655**	0,178
Shallow	NN2	0,019	22,054	-1,566	16,918	0,182	-0,381	16,623	-0,244	0,102	17,655	0,147
Shallow	NN3	0,028	22,052	-1,545	16,918	0,208	-0,370	16,624	-0,303	0,116	17,655	0,158
Shallow	NN4	0,005	22,050	-1,652	16,928	-0,953	-0,449	16,625	-0,515	**0,060**	17,656	0,131
Shallow	NN5	0,011	22,050	-1,636	16,923	-0,411	-0,412	16,624	-0,340	0,113	**17,655**	**0,149**
Deep	NN6	0,029	22,054	-1,559	16,920	-0,115	-0,391	16,625	-0,409	0,102	17,656	0,120
Deep	NN7	0,079	22,057	-1,480	16,912	0,921	-0,274	16,620	0,098	0,234	17,657	0,005
Deep	NN8	0,105	22,055	-1,468	16,910	1,063	-0,291	16,622	-0,125	0,190	17,656	0,085
Deep	NN9	0,113	22,058	-1,451	16,910	1,163	-0,253	16,620	0,153	0,254	17,658	-0,107
Deep	NN10	0,396	22,062	**-1,161**	**16,888**	**3,739**	**0,051**	**16,618**	**0,418**	0,570	17,665	-0,967

The table shows the comparison of the networks when the dropout rate has been lowered from 0.5 to 0.2.

Even though the performance of best performing network does not increase, we notice that the overall performance, for both the deep and non-deep networks, increase. This shows that changing the dropout rate is a feasible strategy to apply to impact the result.

5.3 IMPACT OF NUMBER OF LAGS

In this run, we analyze the impact of varying the number of lags. The dropout rate will be held constant at 0.5. In this run we notice an increase in the out-of-sample R^2 for the OLS and a decrease in out-of-sample R^2 for the neural networks when lowering the number of lags. As we reduce the complexity of our dataset by reducing lagged variables from 20 to 10, we notice an increase of the measurement's metrics for the OLS and a decrease of the measurement metrics for the neural networks. However, since there are still 10 lagged days of variables, it might be too complex of a setup for the OLS to receive a positive out-of-sample R^2 like the neural networks does. Comparing the out-of-sample horizon of 15% out-of-sample R^2 from the previous run with table 5 we see that the out-of-sample R^2 of the OLS have increased from -3.135 to -1.201.

Table 5: Decreasing the number of lags

		In-Sample		*Out-of-sample 1%*			*Out-of-sample 5%*			*Out-of-sample 15%*		
	Model	**Bias**	**RMSE**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**
	OLS	**0,000**	**22,142**	-1,651	17,146	-5,752	-0,475	16,788	-1,920	0,019	17,759	-1,201
Shallow	NN1	0,031	22,178	-1,441	17,095	0,167	-0,345	16,776	-0,444	0,117	17,749	-0,078
	NN2	0,058	22,178	-1,443	17,093	0,389	-0,358	16,775	-0,356	0,106	17,748	0,079
	NN3	0,095	22,177	-1,440	17,091	0,641	-0,340	16,773	-0,118	0,128	17,747	0,122
	NN4	0,054	22,174	-1,526	17,096	0,110	-0,398	16,772	-0,064	**0,089**	**17,745**	**0,317**
	NN5	0,066	22,175	-1,388	17,090	0,773	-0,245	16,774	-0,281	0,249	17,749	-0,096
Deep	NN6	0,185	22,179	-1,326	17,084	1,510	-0,238	16,773	-0,123	0,222	17,749	-0,071
	NN7	0,123	22,181	-1,376	17,087	1,165	-0,234	16,770	0,250	0,276	17,750	-0,157
	NN8	0,323	22,182	-1,170	**17,072**	**2,911**	**-0,038**	16,769	0,391	0,463	17,753	-0,584
	NN9	0,252	22,182	**-1,240**	17,076	2,366	-0,094	**16,768**	**0,403**	0,417	17,752	-0,479
	NN10	0,035	22,181	-1,456	17,093	0,354	-0,314	16,771	0,083	0,195	17,749	-0,040

The table shows the comparison of the networks when the amount of lagged days on the explanatory variables have decreased from 20 to 10. All values are multiplied by 1000 for easier interpretation.

By increasing the number of lagged days from 20 to 30, we receive a stronger indication of the previous statement that by increasing complexity, the OLS performs even worse relative to the neural networks. We can again conclude that the shallow neural networks outperform the deep neural networks. Looking at table 6 we notice that all the shallow networks receive a positive out-of-sample R^2 for an out-of-sample horizon of 15%, while the deep networks receive negative numbers. We can once again observe that NN4 performs relatively well compared to the other neural networks and the OLS.

Table 6: Increasing the number of lags

		In-Sample		*Out-of-sample 1%*			*Out-of-sample 5%*			*Out-of-sample 15%*		
	Model	**Bias**	**RMSE**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**
	OLS	**0,000**	**21,926**	-1,795	16,844	-7,783	-0,452	16,556	-5,396	**0,078**	17,632	-5,616
Shallow	NN1	0,024	21,996	-1,679	16,776	0,403	-0,373	16,512	-0,076	0,149	17,582	0,064
Shallow	NN2	0,086	21,995	-1,666	16,775	0,453	-0,379	16,514	-0,251	0,118	17,582	0,075
Shallow	NN3	0,099	21,995	-1,647	16,771	0,943	-0,370	16,514	-0,296	0,120	17,583	0,007
Shallow	NN4	0,120	21,993	-1,659	16,772	0,818	-0,370	16,513	-0,147	0,125	**17,581**	**0,134**
Shallow	NN5	0,134	21,993	-1,596	16,766	1,552	0,287	16,513	-0,106	0,223	17,582	0,079
Deep	NN6	0,213	21,997	-1,577	16,763	1,900	-0,283	16,511	0,124	0,221	17,583	-0,086
Deep	NN7	0,097	21,998	-1,631	16,770	1,088	-0,302	16,510	0,229	0,233	17,584	-0,101
Deep	NN8	0,099	21,998	-1,696	16,778	1,051	-0,235	16,531	0,230	0,354	17,586	-0,275
Deep	NN9	0,177	21,998	**-1,542**	**16,762**	**2,021**	**-0,214**	**16,509**	**0,387**	0,322	17,585	-0,269
Deep	NN10	0,040	21,998	-1,677	16,775	0,486	-0,349	16,510	0,109	0,186	17,583	-0,043

The table shows the comparison of the networks when the amount of lagged days on the explanatory variables have increased from 20 to 30. All values are multiplied by 1000 for easier interpretation.

In the two runs of increasing and decreasing the number of lagged variables, we notice that the performance of the deep networks does not change much, while the performance for the shallow networks improves substantially. We will further discuss this founding in the final analysis.

5.4 IMPACT OF CHANGING LAGS AND DROPOUT

In this run, we analyze the effect of changing both the number of lags and the dropout rate. By decreasing the number of lags by 10 and the dropout rate to 0.2 we observe that the shallow networks receives the lowest bias recorded so far as well as the highest out-of-sample R^2 for an out-of-sample horizon of 15%. We observe that the best performing network in this run is the NN4 which has the architecture of one input layer of 256 neurons and one hidden layer with 128 neurons. This neural network has improved its performance for each of the previous runs made and is considered the best-performing setup for this study's dataset. This run demonstrates the highest out-of-sample R^2 for an out-of-sample horizon of 15% of all the runs. We can conclude that by changing the number of lagged variables together with hyperparameter tuning, it is possible to further strengthen a neural network, however, this requires a lot of testing which becomes impractical. With these specifications of the architecture, the dropout rate, the number of lags together with batch normalization, we

receive a model that have outperforming predicting power over the linear regression and could with further optimization be valuable to investors.

Table 7: Decrease in dropout rate and number of lags

		In-Sample		*Out-of-sample 1%*			*Out-of-sample 5%*			*Out-of-sample 15%*		
	Model	**Bias**	**RMSE**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**	**Bias**	**RMSE**	**OOS-R2**
	OLS	**0,000**	**22,142**	-1,651	17,146	-5,752	-0,475	16,788	-1,920	0,019	17,759	-1,201
Shallow	NN1	0,018	22,177	-1,427	17,095	0,149	-0,315	16,775	-0,389	0,156	17,749	-0,102
	NN2	0,016	22,177	-1,511	17,097	-0,089	-0,406	16,775	-0,333	0,069	17,747	0,156
	NN3	0,017	22,174	-1,533	17,102	-0,631	-0,446	16,777	-0,670	**0,009**	17,747	0,094
	NN4	0,020	22,169	-1,542	17,096	0,064	-0,407	16,774	-0,215	0,079	**17,744**	**0,476**
	NN5	-0,006	22,169	-1,648	17,101	-0,491	-0,488	16,776	-0,438	0,011	17,745	0,366
Deep	NN6	0,077	22,176	-1,439	17,094	0,277	-0,328	16,775	-0,365	0,138	17,748	0,046
	NN7	0,059	22,179	-1,460	17,093	0,466	-0,326	16,772	0,016	0,177	17,748	0,008
	NN8	0,090	22,178	-1,427	17,092	0,516	-0,332	16,774	-0,225	0,136	17,748	0,075
	NN9	0,144	22,181	-1,358	17,084	1,419	-0,226	16,770	0,215	0,275	17,750	-0,185
	NN10	0,434	22,186	**-1,052**	**17,065**	**3,699**	**0,090**	**16,770**	**0,255**	0,599	17,759	-1,174

The table shows the comparison of the networks when the dropout rate has decreased from 0.5 to 0.2 and the amount of lagged days on the explanatory variables have decreased from 20 to 10. All values are multiplied by 1000 for easier interpretation.

5.5 FINAL ANALYSIS

Our study is less complex in terms of the number of days and variables compared to the studies of Gu et al. (2019b) and Gu et al. (2019a). However, we receive results that are similar to these studies. Even with a smaller number of predictors the R^2 for the OLS receives negative numbers and shows a negative correlation between number of predictors and the out-of-sample R^2. The base run with 20 lagged variables as predictors demonstrates an out-of-sample R^2 that is -3.135 for the OLS with an out-of-sample horizon of 15%. When the number of lagged variables decrease to 10 an out-of-sample R^2 of -1.2 is received for the OLS.

With many parameters to estimate, the efficiency of the OLS regression show the weakness that we were expecting and therefore produces forecasts that are highly unstable out of sample. Our neural networks, however, demonstrates an increase to the out-of-sample R^2 when the number of lagged variables increases. As our comparison for the different runs are analyzed, we can also draw the conclusion that shallow learning outperforms deep learning. As Gu et al. (2019b) mention in their research this differs from the typical conclusion in other

fields such as computer vision and bioinformatics and that this is likely due to the absence of large datasets and a low signal-to noise ratio.

Our findings are based on ten different architectures with different number of hidden layers and number of neurons. The "universal approximation" as stated by (Gu et al., 2019a) is that a model with a single hidden layer is most efficient, but recent literature have shown that this might not always be the case. Deeper neural networks can often achieve the same accuracy with substantially fewer parameters. In our case this was demonstrated by comparing model performance when varying the number of lagged days between 10 and 30. When the comparison was run, the deeper networks performance did not significantly change, however they were stable across the different complexities. The shallow networks performed better and by also decreasing the dropout we reached our best neural network with a relatively high out-of-sample R^2.

5.6 LIMITATIONS AND FURTHER EXTENSIONS

There are several limitations that affect the results of this study. Available data of stock returns and explanatory variables are limited in timeframe and quantity. When handling the dataset, we need to remove certain code with missing or invalid information. This further reduces the amount of data while also creating gaps in the dataset that have to be accounted for. While there is a plethora of machine learning methods available for image- and speech recognition and medicinal purposes, there is a relatively short supply of available methods and models that focus on the combined field of machine learning applied on asset pricing. Previous literature of Gu et al. (2019b) discerns what machine learning cannot be used for. While machine learning is adept at providing measurements of asset pricing, it does not provide any information of the economic mechanisms that in reality are what dictates the price of an asset. It also does not provide information of equilibrium of any kind.

A crucial limitation to this study is that no transaction costs are accounted for. This means that even if predictability and subsequent profitability over random walk is achieved, it is likely that in a real-world scenario, the predictability and the excess profitability vanishes in part or completely. Chen et al. (2020) demonstrates this through the implementation of a no-arbitrage condition being part of the neural network. Also, other financial obstacles such as liquidity might pose a problem when testing the models in a real-world scenario. Our sample

size will be smaller than that of Gu et al. and there will be a considerable reduction in the number of predicting variables due to constraints in time and computational power.

To improve results and relevance, the study can be further optimized through a variety of methods. These methods have not been implemented due to limitations of time and knowledge within the field as well as some being impractical. First, a different architecture, such as the LSTM can be used to improve predictability as observed by Chen et al. (2020). This would likely have further deepened the result but was cut short due to time constraints. There are many stock characteristics that explain returns and in this study a few of the most prominent have been considered. To further deepen the result and possibly receive a model with higher predictability, more characteristics could have been analyzed.

In addition, this study does not analyze the statistical significance of the explanatory variables or the received loss function metrics. Financial forecasting problems often deals with very small out-of-sample R^2 values which could imply that statistical significance is difficult to achieve. The values could, however, be highly relevant to investors and speculators by improving upon timing or trading strategies, as showcased by Campbell and Thompson (2008). The examined literature often protrudes into studying portfolios in addition to individual stocks. This enhances the measurability of return predictability for comparing traditional investing strategies such as buy-and-hold and machine learning models (Gu et al., 2019b).

To increase the predictive ability of the neural networks in this study, further optimization of the hyperparameters can be done. The number of permutations of the hyperparameters are quickly increasing as the number of hyperparameters increase and for this reason it is impractical to build a network with every possible combination of parameters. A larger dataset is likely to increase predictability due to the network having more data to train on. To further increase comparability, more linear models with different penalization techniques such as LASSO and elastic net as well as other machine learning techniques such as regression trees could have been implemented.

6 CONCLUSION

To begin, predicting asset prices is a very tough objective. Even though the field has been studied thoroughly since the inception of the traditional financial theories, the exists no consensus of how asset prices fluctuate. This is not to say that the field is not relevant to study since even a small increase in asset pricing accuracy can yield enormous economic gains to their respective stakeholder.

Many authors within the field agree that machine learning has a powerful predictive ability and can generate value to investors and researchers in the area of asset pricing. After having built models of neural networks and having them put to the test against linear OLS regression, we conclude that neural networks can improve on predictability of stock returns. The measurement metrics are, however, very small and would therefore unlikely be of any use to investors or speculators. Even if a substantial predictability could be found, it is also unlikely to hold, as discussed by Timmermann (2008). The findings of our study show that shallow networks generally outperform deep networks and that the best performing network has one input layer of 256 neurons and one hidden layer with 128 neurons.

We also conclude that neural networks outperform linear OLS regression when predicting stock prices. Our results indicate that OLS regression is too simple of a model to fully account for all the complex features of a stock market, where factors from a variety of fields, such as psychology, traditional finance and macroeconomics mix together. Even the complex structures of a neural network struggle to model this complexity as other issues such as biases arise. The overarching conclusion is therefore that further studying is required to fully allow for all factors to be considered.

7 REFERENCES

Gu, S. and Kelly, B. and Xiu, D. (2019b). Empirical Asset Pricing via Machine Learning, Chicago Booth Research Paper No. 18-04; 31st Australasian Finance and Banking Conference 2018; Yale ICF Working Paper No. 2018-09, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3159577

Gu, S. and Kelly, B. and Xiu, D. (2019a), Autoencoder Asset Pricing Models, Yale ICF Working Paper No. 2019-04; Chicago Booth Research Paper No. 19-24, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3335536

Timmermann, A. (2018), Forecasting methods in Finance, CEPR Discussion Paper No. DP12692, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3122334

Chen, L. and Pelger, M. and Zhu, J. (2020), Deep Learning in Asset Pricing, SSRN Electronic Journal. https://ssrn.com/abstract=3350138

Srivastava, N. and Krizhevsky, A. and Hinton, G. and Sutskever, I. (2014), Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, http://jmlr.org/papers/v15/srivastava14a.html

Xu, Goodacre (2018), On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning, Journal of analysis and testing, https://doi.org/10.1007/s41664-018-0068-2

Timmerman, A. and Pesaran, H. M. (1994), Forecasting stock returns an examination of stock market trading in the presence of transaction costs. Journal of forecasting, 13(4), pp. 335-367, https://onlinelibrary.wiley.com/doi/abs/10.1002/for.3980130402

Chong, T. and Ng, W-K. (2008), Technical analysis and the London stock exchange: Testing the MACD and RSI rules using the FT30. Applied Economics letters, 15, 1111-1114. https://www.researchgate.net/publication/23546661_Technical_analysis_and_the_London_stock_exchange_Testing_the_MACD_and_RSI_rules_using_the_FT30

Ying, X. (2019), An overview of overfitting and its solutions, Journal of physics, IOP Publishing Ltd, https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022

Timmermann, A. (2008), Elusive return predictability, International journal of forecasting, Elsevier, vol. 24(1), pages 1-18, https://doi.org/10.1016/j.ijforecast.2007.07.008

Campbell, J. Y. and Thompson, S. B. (2008), Predicting Excess Stock Returns Out of Sample: Can Anything Beat the Historical Average?, The Review of Financial Studies, Volume 21, Issue 4, July 2008, Pages 1509–1531, https://doi.org/10.1093/rfs/hhm055

Rapach, D and Zhou, G. (2013), Forecasting Stock Returns, Handbook of economic forecasting, edition 1, volume 2, chapter 0, pages 328-383, Elsevier. https://doi.org/10.1016/B978-0-444-53683-9.00006-2

Amihud, Y. and Mendelson, H. (1986). Asset pricing and the bid-ask spread, Journal of Financial Economics 17(2), Pages 223–249. https://www.sciencedirect.com/science/article/pii/0304405X86900656

Figure 1. https://en.wikipedia.org/wiki/Feedforward_neural_network

8 APPENDICES

8.1 THE DATA PREPARATION CODE

```
import matplotlib.pyplot as plt
import pandas as pd
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow import keras
import numpy as np


#Structuring the necessary variables for the model
df_stock = pd.read_csv("./Big_data.csv",sep=';', parse_dates=['day'])

#Creating each div & bid-ask-spread
df_stock["divyield"] = df_stock['dividendyeld'] / df_stock["lastad"]
df_stock["bidaskspread"] = df_stock['askad'] - df_stock["bidad"]
df_stock.dropna(inplace=True)

#Creating a column to the final_dataset with y-variable = log(daily return)
final_datasets=[]

#Choosing predicting horizon, winsorization and number of lags in the dataset
predict_horizon = 1
max_lag_returns= 20
winsorize_returns=0.001

#Adding the return-variable
dft=pd.pivot_table(df_stock, index='day', columns='ticker', values='lastad')
dft=np.log(dft)-np.log(dft.shift(1))
df_returns=dft
dftmelt=pd.melt(df_returns.reset_index(), id_vars=['day'], value_name='return')
v_name='return'
l,h=dftmelt[v_name].quantile([winsorize_returns,1-winsorize_returns]).values.tolist()
dftmelt.loc[dftmelt[v_name]<l,v_name]=l
dftmelt.loc[dftmelt[v_name]>h,v_name]=h
final_datasets.append(dftmelt)
for i in range(0,max_lag_returns):
    final_datasets.append(pd.melt(df_returns.shift(i+predict_horizon).reset_index(), id_vars=['day'], value_name
='return_lag{}'.format(i+predict_horizon)))

def add_variable_final_dataset(variable_name, lags, shifter, scale=True, winsorise=0.0):

    #Create the pivottable and calculation dependent on which type of variable
    if shifter == False:
        if variable_name == "marketvalue":
            dft =pd.pivot_table(df_stock, index='ticker', columns='day', values='marketvalue')
        elif variable_name == "bookvalue":
            dft =pd.pivot_table(df_stock, index='ticker', columns='day', values='bookvalue')
        dft = dft.rank()
```

```
        #dft=pd.pivot_table(df_stock, index='day', columns='ticker', values=variable_name).apply(np.log)
        dftmelt=pd.melt(dft.reset_index(), id_vars=['ticker'], value_name='log_{}'.format(variable_name)).dropna()
    else:
        if variable_name == "SMA50":
            dft = pd.pivot_table(df_stock, index='day', columns='ticker', values="lastad")
            dft = dft.rolling(window=20).mean() < dft.rolling(window=100).mean()
            dft.astype(int)
            dftmelt=pd.melt(dft.reset_index(), id_vars=['day'], value_name='{}'.format(variable_name)).dropna()
        elif variable_name == 'MACD':
            dft = pd.pivot_table(df_stock, index='day', columns='ticker', values="lastad")
            dft = dft.ewm(span=12,adjust=False).mean() - dft.ewm(span=26,adjust=False).mean()
            dftmelt=pd.melt(dft.reset_index(), id_vars=['day'], value_name='{}'.format(variable_name)).dropna()
        elif variable_name == 'volatility':
            dft=pd.pivot_table(df_stock, index='day', columns='ticker', values='lastad')
            dft=np.log(dft)-np.log(dft.shift(1))
            dft*=100
            dft=np.power(dft,2)
            dft=dft.ewm(alpha=0.05).mean()
            dft=np.sqrt(dft)
            dftmelt=pd.melt(dft.reset_index(), id_vars=['day'], value_name='{}'.format(variable_name)).dropna()
        elif variable_name == 'RSI':
            dft=pd.pivot_table(df_stock, index='day', columns='ticker', values='lastad')
            dft=np.log(dft)-np.log(dft.shift(1))
            rsi_period = 14
            dft = 100 - (100/(1+abs(dft.mask(dft<0,0).ewm(com = rsi_period-1,min_periods=rsi_period).mean()/dft.
mask(dft>0,0).ewm(com = rsi_period-1,min_periods=rsi_period).mean())))
            dftmelt=pd.melt(dft.reset_index(), id_vars=['day'], value_name='{}'.format(variable_name)).dropna()
        else:
            dft=pd.pivot_table(df_stock, index='day', columns='ticker', values=variable_name)
            dftmelt=pd.melt(dft.reset_index(), id_vars=['day'], value_name=variable_name).dropna()
#           dft=np.log(dft)-np.log(dft.shift(1))

    #Transform
    v_name=dftmelt.columns[-1]
    if winsorise>0.0:
        l,h=dftmelt[v_name].quantile([winsorise,1-winsorise]).values.tolist()
        dftmelt.loc[dftmelt[v_name]<l,v_name]=l
        dftmelt.loc[dftmelt[v_name]>h,v_name]=h

    if scale==True:
        scaler = MinMaxScaler(feature_range = (0,1))
        dftmelt[v_name]=scaler.fit_transform(dftmelt[[v_name]])

    dft=pd.pivot_table(dftmelt, index='day', columns='ticker', values=v_name)

    #Add lags to final data
    for i in range(0,lags):
```

```
        final_datasets.append(pd.melt(dft.shift(i+predict_horizon).reset_index(), id_vars=['day'], value_name='{}_l
ag{}'.format(variable_name,i+predict_horizon)))


#We use the function to add log variables and lags for 5 days
max_lag=20
add_variable_final_dataset('marketvalue',max_lag,False, scale=True, winsorise=0.001)
add_variable_final_dataset('bookvalue',max_lag, False, scale=True, winsorise=0.001)
add_variable_final_dataset('divyield',max_lag, True, scale=True, winsorise=0.001)
add_variable_final_dataset('SMA50',max_lag,True, scale=True, winsorise=0.001)
add_variable_final_dataset('volatility',max_lag,True, scale=True, winsorise=0.001)
add_variable_final_dataset('RSI',max_lag,True, scale=True, winsorise=0.001)
add_variable_final_dataset('MACD',max_lag,True, scale=True, winsorise=0.001)
add_variable_final_dataset("bidaskspread",max_lag,True, scale=True, winsorise=0.01)
final_dataset=final_datasets[0]
for df in final_datasets[1:]:
    final_dataset=pd.merge(
        final_dataset,
        df,
        how='left',
        left_on=['day', 'ticker'],
        right_on=['day', 'ticker']
    )
final_dataset.dropna(inplace=True)
final_dataset.head().T


#Save to a CSV-file
final_dataset.to_csv('d:/temp/final_dataset.csv')
```

8.2 BUILDING THE DIFFERENT MACHINE LEARNING MODELS

```
import matplotlib.pyplot as plt
import pandas as pd
import os
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow import keras
import numpy as np
import os


gpus = tf.config.experimental.list_physical_devices('GPU')
# Currently, memory growth needs to be the same across GPUs
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
```

```
#Read the csv-file we created erlier
final_dataset=pd.read_csv('./final_dataset.csv')
final_dataset=final_dataset.iloc[:,1:]
final_dataset.dropna(inplace=True)
final_dataset.iloc[:,2:23]+=1


#determine the size of the splits and the out of sample horizon
splits=[0.15,0.75,0.15]
out_of_sample_horizon = 60


#Split the data according to the pre-determined splits
splits=np.cumsum(splits)
dates=sorted(final_dataset.day.unique())
ix_validation=int(len(dates)*splits[0])
date_validation_to_split_on=dates[ix_validation]

ix_train_test_start=int(len(dates)*splits[1])

for ix in range(ix_train_test_start,len(dates)-out_of_sample_horizon):
    date_to_split_on=dates[ix]
    final_oos_date=dates[ix+out_of_sample_horizon]
    df_val=final_dataset.query('day <= @date_validation_to_split_on')
    df_train=final_dataset.query('day > @date_validation_to_split_on and  day <= @date_to_split_on')
    df_test=final_dataset.query('day > @date_to_split_on and day <= @final_oos_date')

    #df_valid=final_dataset.query('day > @date_to_split_on and <= @final_oos_date')
    # we want to split the data into 70% training and 15% validation and 15% testing
    #Fitting... #Help with validation and how it should be done?

    break


#create our x_train and y_train values of the dataset
def create_x_and_y(df):
    x_train=df.iloc[:,3:]
    y_train=df.iloc[:,2:3]
    return x_train, y_train


#declare the training-set, test-set & validation-set
x_train, y_train = map(np.array, create_x_and_y(df_train))
x_test, y_test = map(np.array, create_x_and_y(df_test))
x_val, y_val = map(np.array, create_x_and_y(df_val))


#Reshape so that the structure fit the model
def reshape_x(x_train):
    return np.reshape(x_train,(x_train.shape[0],1,x_train.shape[1]))
```

```
np.random.seed(1)
ix=sorted(np.random.randint(0,x_train.shape[0],100000).tolist())
x_train=x_train[ix,:]
y_train=y_train[ix,:]


reshape_x(x_train).shape


def get_ols(x_train, y_train):
    xx_train=np.hstack((np.ones((x_train.shape[0],1)), x_train))
    bb=np.dot(np.linalg.inv(np.dot(xx_train.T, xx_train)), np.dot(xx_train.T, y_train))
    bias=np.dot(xx_train,bb)-y_train
    return np.mean(bias), np.sqrt(np.mean(np.power(bias,2)))
get_ols(x_train,y_train)


from IPython.display import clear_output
class PlotLearning(keras.callbacks.Callback):
    def __init__(self, plot_every=1, logs={}, start_ploting=5):
        super(PlotLearning, self).__init__()
        self.df = pd.DataFrame()
        self.plot_every = plot_every
        self.start_ploting = start_ploting



    def on_epoch_end(self, epoch, logs={}):
        self.df=self.df.append(logs, ignore_index=True)
        if len(self.df)>self.start_ploting:
            if len(self.df) % self.plot_every == 0:
                measures=[i for i in self.df.columns if 'val' not in i]
                measures_val=["val_"+i for i in measures]
                f=plt.figure(figsize=(7*len(measures), 4))
                for i,m,mv in zip(range(len(measures)),measures,measures_val):
    #                 if m=='loss':
    #                     continue
                    ax=plt.subplot(1,len(measures),i+1)
                    self.df[[m,mv]].iloc[self.start_ploting:,:].plot(ax=ax).plot(ax=ax)
                plt.tight_layout()
                clear_output(wait=True)
                plt.show()


from time import time


class TerminateOnBaseline(keras.callbacks.Callback):
    """Callback that terminates training when either acc or val_acc reaches a specified baseline
    """
    def __init__(self, monitor='acc', baseline=0.9, mode='min'):
```

```
        super(TerminateOnBaseline, self).__init__()
        self.mode = mode
        self.monitor = monitor
        self.baseline = baseline

    def on_epoch_end(self, epoch, logs=None):
        logs = logs or {}
        acc = logs.get(self.monitor)
        if acc is not None:
            if self.mode=='max':
                if acc >= self.baseline:
                    print('Epoch %d: Reached baseline, terminating training' % (epoch))
                    self.model.stop_training = True
            if self.mode=='min':
#                 clear_output(wait=True)
#                 display([acc, self.baseline, acc/self.baseline])
                if acc <= self.baseline:
                    print('Epoch %d: Reached baseline, terminating training' % (epoch))
                    self.model.stop_training = True
class TerminateOnTime(keras.callbacks.Callback):
    """Callback that terminates training when either acc or val_acc reaches a specified baseline
    """
    def __init__(self, max_time=120):
        super(TerminateOnTime, self).__init__()
        self.t0 = time()
        self.max_time = max_time

    def on_epoch_end(self, epoch, logs=None):
        if time()-self.t0 >= self.max_time:
            print('Epoch %d: Reached max time, terminating training' % (epoch))
            self.model.stop_training = True

class ClearDisplay(keras.callbacks.Callback):
    """Callback that terminates training when either acc or val_acc reaches a specified baseline
    """
    def on_epoch_end(self, epoch, logs=None):
        clear_output(wait=True)


def fit_model(x_train, y_train, x_val, y_val, batch_size, optimizer, epochs, dropout, neural_network):
    clear_output(wait=True)
    np.random.seed(1)
    tf.random.set_seed(1)
    regressior = keras.Sequential()

    if neural_network == 1:
        regressior.add(keras.layers.Dense(units = 32, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_ini
tializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
```

```
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [16]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 2:
        regressior.add(keras.layers.Dense(units = 64, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_ini
tializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [32]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 3:
        regressior.add(keras.layers.Dense(units = 128, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [64]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 4:
        regressior.add(keras.layers.Dense(units = 256, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [128]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 5:
        regressior.add(keras.layers.Dense(units = 512, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [256]:
```

```
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 6:
        regressior.add(keras.layers.Dense(units = 128, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [64, 64]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 7:
        regressior.add(keras.layers.Dense(units = 128, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [64, 64, 32, 32, 16, 16]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 8:
        regressior.add(keras.layers.Dense(units = 256, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [128, 128, 64, 64]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 9:
        regressior.add(keras.layers.Dense(units = 256, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [128, 128, 128, 64, 64, 64, 32, 32, 32]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
```

```
            regressior.add(keras.layers.Dropout(dropout))

    if neural_network == 10:
        regressior.add(keras.layers.Dense(units = 512, input_shape=(1, x_train.shape[1]), use_bias=True, kernel_i
nitializer='glorot_uniform', bias_initializer='zeros', activation = 'relu'))
        regressior.add(keras.layers.BatchNormalization())
        regressior.add(keras.layers.Dropout(dropout))
        for i in [256, 256, 256, 256, 128, 128, 128, 128, 64, 64, 64, 64, 32, 32, 32, 32, 16, 16, 16, 16, 8, 8, 8, 8, 4, 4
, 4, 4, 2, 2, 2, 2]:
            regressior.add(keras.layers.Dense(units = i, use_bias=True, kernel_initializer='glorot_uniform', bias_init
ializer='zeros', activation = 'relu'))
            regressior.add(keras.layers.BatchNormalization())
            regressior.add(keras.layers.Dropout(dropout))



    regressior.add(keras.layers.Dense(units = 1, use_bias=True, kernel_initializer='glorot_uniform', bias_initializ
er='zeros', activation = 'linear',) )
    regressior.compile(optimizer=optimizer, loss = 'mse', metrics=[
        keras.metrics.MeanSquaredError(name='0MSE'),
#         keras.metrics.KLDivergence(name='1KLD'),
#         keras.metrics.MeanAbsolutePercentageError(name='2MAPE'),
#         keras.metrics.MeanAbsoluteError(name='3MAD'),
    ])

    callbacks=[
#         PlotLearning(plot_every=2, start_ploting=2),
        TerminateOnTime(max_time=3600),
        keras.callbacks.EarlyStopping(monitor='val_loss', patience=100, min_delta=1e-12, restore_best_weights=
True),
        keras.callbacks.TensorBoard(
            log_dir=os.path.join(os.getcwd(),'logs','NN7'.format(optimizer,batch_size)),
            histogram_freq=1,
            write_graph=False,
            write_images=True,
            update_freq='epoch',
            profile_batch=0
        ),
        ClearDisplay()
    ]
    regressior.fit(
        reshape_x(x_train), reshape_x(y_train),
        validation_data=(reshape_x(x_val),reshape_x(y_val)),
        batch_size=batch_size,
        epochs=epochs, callbacks=callbacks ,verbose=1)
    return regressior


def get_R2_OOS(x_test, y_test, y_train, res):
```

```
    yhat_test = res.predict(reshape_x(x_test)) #predictions of returns in the test set
    SSE = np.sum(np.power(reshape_x(y_test)-yhat_test,2)) #sum(returns in the test set - predictions of returns i
n the test set)^2
    #R2_OOS = 1 - bias_test/ sum of (returns in the test set - mean of returns in training)^2
    SST = np.sum(np.power(y_test - np.mean(y_train),2))
    R2_OOS = 1 - (SSE / SST)
    return R2_OOS


def get_nn(x_train, y_train, res):
    yhat=res.predict(reshape_x(x_train))
    bias=yhat-reshape_x(y_train)
    return np.mean(bias), np.sqrt(np.mean(np.power(bias,2)))


def get_ols_test(x_train, y_train, x_test, y_test):
    xx_train=np.hstack((np.ones((x_train.shape[0],1)), x_train))
    bb=np.dot(np.linalg.inv(np.dot(xx_train.T, xx_train)), np.dot(xx_train.T, y_train))
    bias=np.dot(np.hstack((np.ones((x_test.shape[0],1)), x_test)),bb)-y_test

    SSE = np.sum(np.power(bias,2)) #sum(returns in the test set - predictions of returns in the test set)^2
    #R2_OOS = 1 - bias_test/ sum of (returns in the test set - mean of returns in training)^2
    SST = np.sum(np.power(y_test - np.mean(y_train),2))
    R2_OOS = 1 - (SSE / SST)

    return np.mean(bias), np.sqrt(np.mean(np.power(bias,2))), R2_OOS


def print_result(res):
    for s, x, y in [

        ('Train', x_train, y_train),
        ('Val', x_val, y_val),
        ('Test', x_test, y_test)
    ]:
        print('='*80)
        print(s)
        print('OLS')
        print(get_ols(x, y))
        print('OLS vs out-of-sample')
        print(get_ols_test(x, y, x_test, y_test))
        print('NN')
        print(get_nn(x, y, res))
    print("R2-OOS = {}".format(get_R2_OOS(x_test, y_test, y_train, res)))
#res = keras.models.load_model('./Neural network 1.h5')
#print_result(res)
%%time
neural_network = 1
```

```
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.2, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 2
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.2, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 3
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 4
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 5
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 6
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 7
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)

%%time
neural_network = 8
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)


%%time
neural_network = 9
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)
```

```
%%time
neural_network = 10
res=fit_model(x_train, y_train, x_val, y_val, 500, 'sgd', 20000, 0.5, neural_network)
res.save('Neural Network {}.h5'.format(neural_network))
print_result(res)
```