Master Thesis Econometrics and Management Science Quantitative Finance

Are deep learning models outperforming linear regression model in return prediction?

Date final version:
2022/01/19

Author & Student number:
Jeroen van Druten - 478791

Supervisor:
dr. S.O.R. Lonn

Second assessor:
prof. dr. M. van der Wel

Abstract

This study investigates whether deep learning models can improve sign prediction of returns over the linear regression model in the noisy and nonlinear financial data. We analyse the models on the S&P500 index for daily, weekly, and monthly intervals, where we compare the prediction and financial performance of the models. This study uses auto-machine learning for feature selection, hyperparameter optimisation, and model combination. We find that both kinds of models' overall prediction performance are good. However, the deep learning models cannot significantly outperform the linear regression model for any of the intervals in terms of prediction performance. In terms of financial performance, all the models achieved positive profit. However, there is no best performing model following the profitability across the intervals. The same holds when reviewing the annual Sharpe ratio. Here the linear regression model has a more stable annual Sharpe ratio. In contrast, the deep learning models fluctuate more around the annual Sharpe ratio of the S&P500 index itself.

Keywords: return prediction, time series forecasting, deep learning, generalised stacking, AutoML, feature selection, sign prediction, CNN, DMLP, LSTM, linear regression model

Abbreviations

AutoML Automated machine learning

CNN Convolutional neural network

CRSP Center for Research in Security Prices

DLM Deep learning model

DMLP Deep multilayer perceptron

FRED Federal Reserve Bank of St. Louis

HPO Hyperparameter optimisation

LSTM Long-short term memory network

MLP Multilayer perceptron

PnL Profit and loss

RNN Recurrent neural network

SP S&P500 index

WRDS Wharton Research Data Services

Contents

1 Introduction

1.1 Background

Various models are used to predict financial time series, from linear regression and autoregressive models to machine learning models. Balconi, Brusoni, and Orsenigo (2010) state that the linear regression model is set aside by researchers to propose using other models. These models either improve upon the linear regression model framework or in other directions. These models are chosen to overcome assumptions over the data from the linear regression model as linear relationships or the need to set relationships in advance for a linear regression model. The linear regression model follows

$$y = X\beta + \varepsilon, \tag{1}$$

with y a vector of the dependent variable, X a matrix of explanatory variables, β a vector of coefficients and ε a vector of errors. In order to estimate the linear regression model, we use the well-known ordinary least squares estimator, leaving us for the estimation of β

$$\hat{\beta} = (X'X)^{-1}X'y. \tag{2}$$

Although alternative models can work around the assumptions of a linear regression model, such as linear dependencies, it does not mean these models outperform the linear regression model in terms of modelling data or predicting future returns. This depends on the data. Furthermore, the drawback of these models compared to the linear regression model is that they become increasingly complex, leading to higher computational time. The high computational time can become problematic when it exceeds the predicting interval due to the prediction not being ready in time. Because of these properties, the linear regression model is used in multiple studies on financial predictions as a benchmark model to assess the gain in performance of other models (Elliot and Hsu (2017), Gencay (1996), Gencay (1998), Jang and Lee (2017)).

Finding patterns in financial price data is difficult because the financial price data is noisy and primarily nonlinear. Some studies suggest that stock returns are random walks, which causes future short-term returns to be unpredictable (Roll, At, and Roll (1970), Fama and MacBeth (1973), Fama (1965), Samuelson (1973)). Other studies find contrary evidence for the predictability of short-term returns. For example, Lo and MacKinlay

(1988) find that weekly returns do not follow random walks. Their study finds that they can reject the hypothesis of a random walk for weekly market prices. To test this hypothesis, they use a volatility-based specification test to test the presence of a significant drift parameter in the log-price process. They find that there is indeed a significant drift in the log-price process for weekly prices. Due to this finding, Lo and MacKinlay (1988) state that serial correlation is present in weekly price data. Furthermore, Fama (1995) states that technical analysis can extract information on market prices. Here technical analysis searches for patterns in the market price, whereas believers of technical analysis expect such patterns to repeat themselves. Therefore, researchers that use technical analysis can have meaningful insights when they consistently outperform random predictions with their analysis. To add on this, Hinich and Patterson (1985) find that daily market prices are not linear and that when modelled with a nonlinear regression model, the dependence between daily returns is higher than when using a linear regression model. This result indicates that there can be gains in predictability when modelling market prices nonlinear instead of linear. Because small increases in prediction performance in this field can be very lucrative, the search for a model which improves predictions is meaningfull.

Using deep learning models can be the solution to modelling unknown nonlinear relationships in financial return data. In contrast to regression models, deep learning models can capture multiple different kinds of relationships between input features and the response variable without being specified in advance. Therefore, deep learning models can find both linear and nonlinear relationships during training. Due to this characteristic, deep learning models and their underlying artificial neural network have become helpful in multiple research fields, such as image, speak and text recognition (LeCun, Bengio, and Hinton (2015)).

Deep learning, a branch of machine learning, uses an artificial neural network containing more than one hidden layer, which is called a deep neural network. The addition of layers allows a deep neural network to increase the complexity of relationships that the model can learn. This causes a deep neural network to find more precise patterns than their underlying artificial neural network.

Deep learning models learn from the input and output data in a training environment. First, the model gives weights to inputs, which are then updated using a penalty given to the error between the made prediction and the actual output. By repeating this process,

the model minimises the error. After this process, the model is used on the test dataset to test its performance.

Sezer, Gudelek, and Ozbayoglu (2020) review 140 papers to understand better the existing and well-performing deep learning models used in forecasting the return and prices of financial products. The reviewed studies by Sezer et al. (2020) use deep learning models for financial time series, including stock prices, index prices, bond prices and forex prices. Sezer et al. (2020) conclude that there are three most used and well-performing deep learning models, deep multilayer perceptron, convolutional neural network, and long short-term memory network.

The deep multilayer perceptron originates from its shallow model, the multilayer perceptron (Gardner and Dorling (1998)), also referred to as a 'vanilla' neural network. The deep multilayer perceptron contains multiple layers of neurons, which feed through information to link in- and output. The deep multilayer perceptron is mainly used for trend classification or identification of the buy and sell point. The convolutional neural network is the dominating deep learning model in image recognition. The model is originally made with a two-dimensional convolutional layer to work in two directions (horizontal and vertical), which is relevant for image processing (Rawat and Wang (2017)). However, we can transform the two-dimensional convolutional layer into a one-dimensional convolutional layer (Sezer and Ozbayoglu (2018)). In this way, the convolution is done in one direction (in time). Chen, Chen, Huang, Huang, and Chen (2016) show that convolutional neural network results can be used for financial data and are effective as a trading model. Lastly, the long short-term memory network is a deep learning model in the category of recurrent neural networks. The long short-term memory network is preferred because of its ability to capture both long- and short-term time-dependent components present in most financial data. The model is mainly used for predicting the next day value.

1.2 Objectives

This study investigates if deep learning models improve sign prediction of returns over the linear regression model. Deep learning models are generally used because they can learn multiple relationships without forehand specification. This property of deep learning models causes researchers to explore deep learning in multiple areas, such as financial time series forecasting. Sezer et al. (2020) state that deep learning models show promising

performance. However, a thorough comparison of the promising performance against the performance of earlier prediction models is lacking in the literature. Therefore, we investigate if these promising deep learning models outperform the linear regression models. This leaves us with the following research question: *'Do deep learning models improve sign prediction of returns over the linear regression model?'* And the underlying question: *'Does sign prediction of returns improve when increasing the time interval?'*.

Various deep learning models are available for financial time series forecasting. We are using the most used deep learning models in the financial field following the review of Sezer et al. (2020); deep multilayer perceptrons, convolutional neural networks and long short-term memory networks. The implementation of the deep learning models uses automated machine learning for feature selection, optimisation of the hyperparameters and the combination of the models. We consider automated machine learning methods to reduce the need for a well-embedded researcher. Since automated machine learning methods can perform optimisations, a well-embedded researcher in the area otherwise does. Without one of the other, optimisation tasks would take substantial time. This is due to the large sizes of both feature sets as hyperparameter spaces.

We use stacking to combine the predictions of the deep learning models. Stacking uses a meta-learner that learns to combine base models so that the combination of predictions maximises in-sample accuracy. It is surmised that this pattern holds in out-of-sample as well. When this holds, the stacked model achieves a higher overall out-of-sample accuracy than the deep learning models separately.

The return of the S&P500 index is used as financial time series to analyse and compare the performance of both kinds of models. We predict the sign of return for three time intervals, namely on daily, weekly and monthly basis. Multiple intervals are used to show possible influence in performance by the prediction interval. As features for the models, we include technical indicators, macro-economic variables and underlying stocks of the S&P500 index. Finally, we compare the performance of the models in terms of prediction and financial performance. To do this, we use the confusion matrix, accuracy, F1-score and the McNemar test as prediction measures, and profit and loss and annual Sharpe ratio as financial measures. Here we create a trading game to assess the financial performance.

The set-up of this paper is as follows. First, we go through the deep learning models;

their architecture, hyperparameter optimisation and stacking. Second, the time series and features are given together with the feature selection. Third, the prediction and financial measures are introduced. Finally, the results and conclusion are presented.

2 Deep learning models

This study uses three separate deep learning models (DLMs) and one model that combines the DLMs, the stacked model. The DLMs that are used are the deep multilayer perceptron (DMLP), convolutional neural network (CNN) and long short-term memory network (LSTM), which are the most used DLMs in finance following the review of Sezer et al. (2020). All the DLMs are used for two-class classification due to the sign prediction, either positive (zero or higher) or negative. The implementation of the DLMs goes via python 3.9.0, where the major included packages are: Tensorflow (Abadi et al. (2015)) and its underlying package Keras (Chollet et al. (2015)) for the DLMs and the stacked model, and Scikit-learn (Pedregosa et al. (2011)) for the prediction measures.

In the following of this section, we go through the concept of the DLMs. Whereafter, the tuning of the models via the choice of hyperparameters is given together with the data separation. Finally, we explain the model combination using stacking.

2.1 Deep multilayer perceptron architecture

DMLP is the first DLM we introduce for the study. DMLP is a multilayer perceptron (MLP) with multiple hidden layers. DMLP consists of three types of layers, containing input, hidden and output. As shown on the left side of Figure 1, the layers consist of nodes named neurons, which hold information about the time series. The concept that makes DMLP 'Deep' is that DMLP has multiple hidden layers instead of the single hidden layer of an MLP.

In the hidden layers, the neurons have the following terms; input (x), weight (w) and bias (b). The neurons are in contact with the preceding layer by lines called synapses. Through these synapses, the neuron gets signals from the preceding neurons. These signals contain the input of preceding neuron i, x_i, the weight assigned between the preceding neuron i and current neuron j, $w_{i,j}$, and a bias term between the preceding neuron i and current neuron j, b_i. The signals of all preceding neurons are input for a nonlinear

activation function

$$x_j = g\left(\sum_{i=1}^{n} x_i w_{i,j} + b_{i,j}\right), \tag{3}$$

where x_j is the output of the j^{th} current neuron, $g(\cdot)$ is the nonlinear activation function, and n is the number of neurons in the preceding layer. This results in a cumulative output for the current neuron. The process between the layers to get the information of the preceding neurons to the future neurons is visualised on the right side of Figure 1, where we have an example with five preceding neurons.

Passing through information from preceding neurons to future neurons is done through all layers starting at the input layer and is called forward passing. The left side of Figure 1 displays this process, where we use a DMLP with two hidden layers and five neurons per hidden layer. The number of hidden layers used in DMLP can vary. The more hidden layers, the more transformations are applied, making the model able to capture more complex relationships. However, it also increases the computation time.

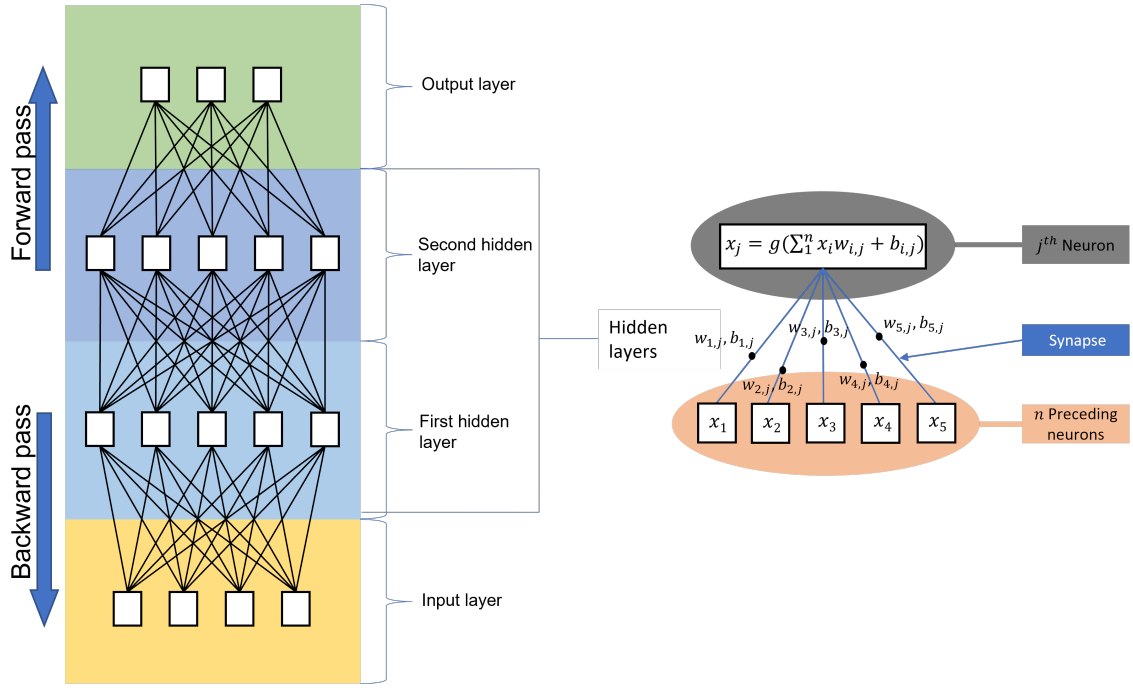


Figure 1: The process of DMLP. Here the left side shows the process for the forward and backward pass of DMLP with two hidden layers and five neurons per hidden layer. The right side presents the process between the layers to get from the preceding neurons to a particular future neuron.

DMLP can learn from the output data through backpropagation. Backpropagation is done by propagating the errors in the neurons in the output layer back to the preceding

layers. In this way, DMLP can update the neurons' terms (x, w and b). For the forward pass, the propagation is performed in the opposite direction. The backward and forward passes are performed sequentially until the error is minimised.

The benefits of the DMLP are that the model can construct any possible relationship between features and output as the number of hidden layers and neurons are sufficient. Here non of the relationships have to be set beforehand, making the model adaptive. A drawback of the model is the computation time, which increases significantly when the model grows in terms of layers and neurons. The list of hyperparameters of DMLP together with their search space is given in Section 2.4.

2.2 Convolutional neural network architecture

Compared to standard DMLP, CNN processes the data before applying DMLP itself. As a result, the input data is reduced in dimensionality, whereby information about the data is stored and used for the DMLP. A benefit of CNN is that preprocessing the data before DMLP results in a lower overall computational time. This is due to the decreased size of the input data, which implies fewer neurons in the input layer. This means that fewer parameters have to be tuned in the DMLP process. Furthermore, CNN holds the same benefits as the DMLP. A drawback of CNN is that preprocessing the data can cause valuable information of the input data to be left out.

CNN is constructed by three kinds of layers, namely convolutional, pooling, and fully connected. These layers form the framework of CNN. When there is more than one layer of a certain kind of layer, the shallow model becomes a deep neural network. We are using more than one convolutional and fully connected layer to ensure a deep neural network. To illustrate the process of CNN, we use a figurative univariate time series of returns with length n. However, for the sign prediction of return, we apply the models on a multivariate time series with a width of the number of features. The core process of the one-dimensional CNN does not change under the addition of columns. Figure 2 shows the process of CNN on the figurative time series.

The process begins in the first layer, the convolutional layer, where the first part of the preprocessing of the input data is performed. The preprocessing is done to reduce the dimensionality of the input data while storing all kinds of properties of the input data (in this case, the univariate time series). The convolutional layer gets its name from

the process called convolution. This process uses filters to create a set of vectors, where each vector contains specific properties of the input data. This is achieved by iterating through the input data. For a one-dimensional CNN, the iteration is only applied in one direction, vertically (in time).

The filters are vectors, which contain a certain number of convolution kernels. Here convolution kernels are matrices of learnable parameters, which assign weights according to the data. In the case of a one-dimensional CNN, these kernels are 1×1 matrices. Filters can consist of different amounts of kernels. Where every filter is able to capture different properties of the input data due to the unique set of kernels in a filter. There can be multiple filters in the convolution process with equal or different dimensions. In the example of a univariate time series as input, the filters are $m \times 1$, where m is the length.

As shown by Figure 2, the convolution process uses an algorithm that iterates through the univariate time series (for a multivariate time series, the same thing holds). Let the convolution use f filters. The algorithm then iterates the f filters through the time series, such that for the first iteration, a filter of $m \times 1$ goes over the first m rows of the time series. For the second iteration, the filter goes over the second row to $m + 1$ row and so on. By every iteration there is a dot product between filter i (f_i), containing a certain set of kernels (k_i), and the data points the algorithm goes over in that specific iteration (x_j). This dot product is then transformed using a nonlinear activation function ($g(\cdot)$)

$$y_{i,j} = g\left(\begin{bmatrix} k_{i,1} \\ \vdots \\ k_{i,m} \end{bmatrix} \cdot \begin{bmatrix} x_{j,1} \\ \vdots \\ x_{j,m} \end{bmatrix}\right), \tag{4}$$

where $y_{i,j}$ is the value of j^{th} iteration of the i^{th} filter. The resulting value $y_{i,j}$ is added to the i^{th} vector, with length $n - m - 1$ ($n - m - 1$ being the maximum iteration for an m sized filter). In this way, f vectors are constructed, which contain different properties of the input data.

After the convolution, pooling is applied to the f vectors. Pooling reduces the dimensions of the input data for the fully connected layer, which decreases the number of computations needed. The pooling of the vectors results in one final vector (p). This study uses max-pooling as a pooling technique, because max-pooling is the standard op-

tion in Keras for the pooling layer. Here max-pooling takes the maxima of the vectors y_i, for $i \in \{1, \ldots, f\}$. Such that the resulting vector p contains f values, where the values are

$$p_i = \text{Max}\left(\begin{bmatrix} y_{i,1} \\ \vdots \\ y_{i,n-x-1} \end{bmatrix}\right). \tag{5}$$

The resulting vector, p, is used as input for the fully connected layer. Here we use DMLP as explained in Section 2.1 for the fully connected layer. All hyperparameters used to make the CNN models together with their search space are given in Section 2.4.

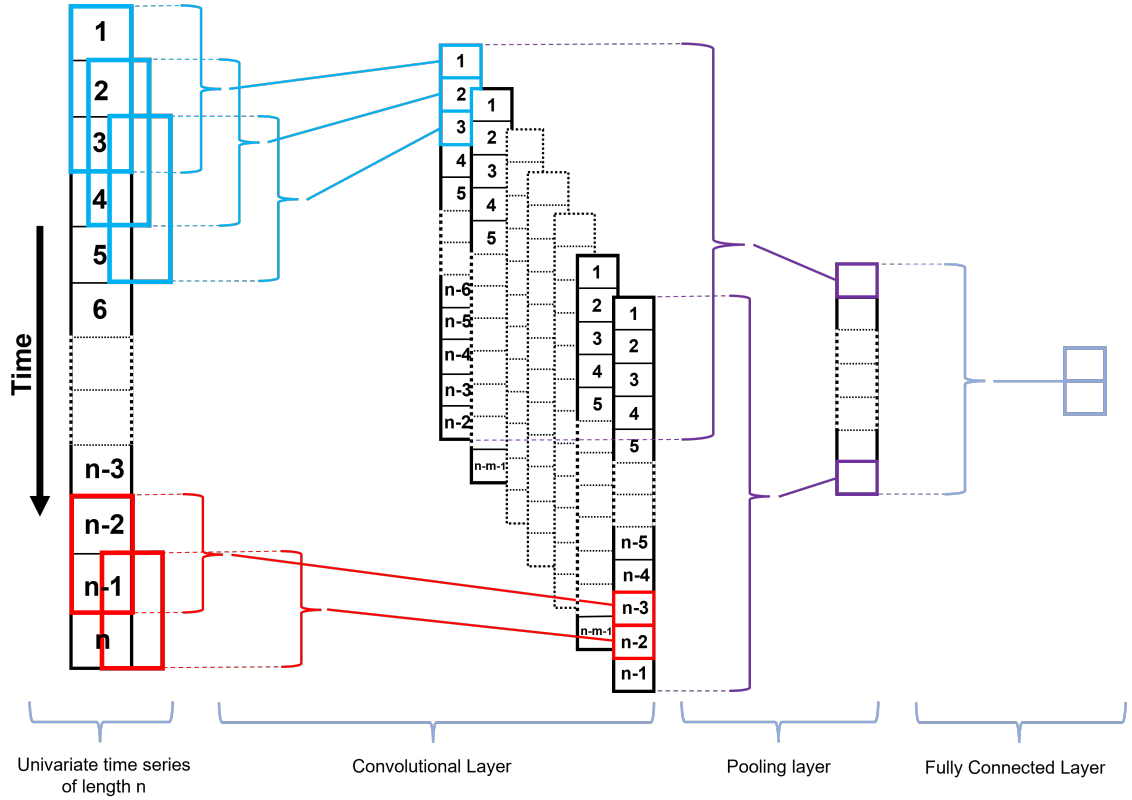


Figure 2: The process of CNN for a univariate time series of length n. For the convolutional layer we have two filters of lengths two and three.

2.3 Long short-term memory network architecture

The last DLM is LSTM, proposed by Hochreiter and Schmidhuber (1997), which falls under the category of recurrent neural networks (RNNs). RNNs are DLMs that can memorise short- and long-term dependencies throughout training. The memorising of these dependencies is beneficial for financial time series because both previous and new

information of the time series is used. This property is not present in DMLP and CNN. The drawback of basic RNNs is that the long-term dependencies tend to vanish over time due to the impact of short-term dependencies. Therefore, the gradient of long-term dependencies can completely vanish at the end of long time series (Hochreiter et al. (2001)). The vanishing of the gradient appears because the short- and long-term dependencies are stored in the same channel, which runs through the entire training dataset.

This drawback of basic RNNs is why LSTM is developed (Hochreiter and Schmidhuber (1997)). Here LSTM does not use one channel to pass through information like a basic RNN, but multiple channels. For a shallow LSTM, there are two channels. The first channel holds the cell state (C), which holds the relevant information of the time series at a certain point in time. This channel is also present in the RNN. The second channel is for the so-called hidden state (H), which keeps the long-term dependencies. Therefore, the long-term dependencies have their own channel, which ensures that their gradient is not vanished due to the impact of short-term dependencies. In this way, LSTM can use relevant information of both long- and short-term dependencies. Where the influence of the dependencies can be decreased when they are not relevant anymore.

Figure 3 illustrates the process of LSTM. LSTM uses gates to add and/or remove information from the cell state over time. In the process, the gates that feed new information to the cell state learn along with the time series. This is done using back and forward propagation. We have three different gates in LSTM, the forget, input and output gate. The forget gate updates the previous cell state (C_{t-1}) by multiplying values of the previous cell state pointwise with a vector consisting of weights calculated by the sigmoid activation function ($S(\cdot)$) with as input the updated previous hidden state (H_t^*)

$$C_t^* = C_{t-1} \circ S(H_t^*), \tag{6}$$

where C_t^* is the updated previous cell state and H_t^* is the updated previous hidden state. Here H_t^* is the pointwise addition of the previous hidden state (H_{t-1}) and the input at time t (X_t),

$$H_t^* = H_{t-1} + X_t. \tag{7}$$

$S(H_t^*)$ consists of weights between zero and one, which is used to update the importance of the previous cell state by new information. Here the sigmoid function transforms a value x to a value between zero and one as follows

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{8}$$

To further update the cell state, we add the information of the updated previous hidden state to the cell state through the input gate. Before adding the information of the updated previous hidden state, its data is transformed. First, the values of the updated previous hidden state are transformed by the tanh activation function ($T(\cdot)$) to values between minus one and one. For a value x, the tanh activation function transforms the value following

$$T(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{9}$$

The transformation is used to remove the change of exploding values, which can cause information of other values to be left out due to (seemingly) insignificance. This vector of standardised values is then pointwise multiplied by the vector of weights of the updated previous hidden state calculated by the sigmoid activation function

$$TH_t = T(H_t^*) \circ S(H_t^*), \tag{10}$$

where TH_t is the transformed current hidden state. TH_t is then pointwise added to the updated previous cell state

$$C_t = C_t^* + TH_t, \tag{11}$$

with C_t being the current cell state.

Finally, we compute the current hidden state through the output gate using information from the current cell state and the updated previous hidden state. To receive the current hidden state (H_t), the vector of weights of the current hidden state calculated by the sigmoid activation function is pointwise multiplied with the vector of values of the current cell state standardised using the tanh activation function,

$$H_t = S(H_t^*) \circ T(C_t). \tag{12}$$

This gives the current hidden state that is used for the next state.

This process goes through the entire dataset while training. This results in a vector of weights that is used in combination with values of the feature set to predict returns in the test dataset. The list of hyperparameters used for LSTM models together with the search space of each hyperparameter are given in Section 2.4.

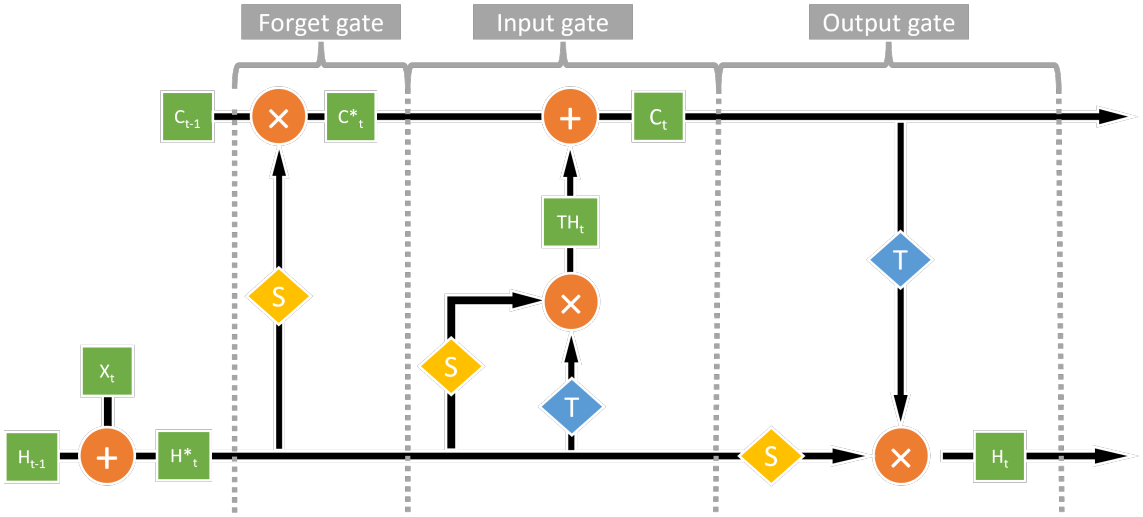


Figure 3: The process of LSTM, showing the forget, input and output gate. C_{t-1} is the previous cell state, H_{t-1} is the previous hidden state, X_t is the new information of the current state, H_t^* is the updated previous hidden state, C_t^* is the updated previous cell state, TH_t is the transformed current hidden state, C_t is the current cell state and H_t is the current hidden space. S and T denote the activation functions, sigmoid and tanh, respectively. $\times$ is the pointwise multiplication, and $+$ is the pointwise addition of two incoming vectors.

2.4 Model optimisation

Correctly tuning a DLM is needed to ensure a good performing model. Hyperparameters affect the architecture of the model and, thereby, the performance. By changing, for example, the number of hidden layers or the number of neurons in each layer, the performance of a specific DLM can change substantially.

In this section, we first go into the hyperparameter optimisation (HPO) in general, whereafter, we go into the HPO with automated machine learning (AutoML) and specifically Bayesian optimisation. Secondly, the hyperparameters that are present in the DLMs are displayed. Lastly, the separation of the dataset is presented.

2.4.1 Hyperparameter optimisation

Machine learning models can have many hyperparameters that have to be specified. In addition, DLMs usually have more hyperparameters than shallow networks because of the addition of layers. Due to the high number of hyperparameters finding the optimal set of hyperparameters is challenging. This is because a change in one hyperparameter can lead to a needed change in other hyperparameters to achieve higher performance. The possible combinations, therefore, become incredibly high. Hence, hyperparameter

optimisation requires an expert in the field where the model is used when choosing the hyperparameter set by hand (Elshawi, Maher, and Sakr (2019)). However, Elshawi et al. (2019) state that this can also be done by AutoML.

This study uses AutoML to optimise the hyperparameters of the DLMs. Data and an objective function are needed for AutoML. Therefore, a part of the training data is used to optimise of the hyperparameters. Here the assumption is that this data represents the entire data, making the DLM suitable for the training and testing dataset. An objective function is also needed to choose a hyperparameter set optimised for the correct problem, in our case, maximum accuracy. However, the objective function can also minimise loss. Due to the complexity of the problem, the form of the objective function is unknown and can be nonlinear and non-convex. Together with the high number of possible hyperparameters, the objective function can be tough to optimise.

2.4.2 Automated hyperparameter optimisation

Hutter, Kotthoff, and Vanschoren (2019) collect a series of papers on the automation of machine learning to give a comprehensive overview of the opportunities of AutoML. Elshawi et al. (2019) present AutoML techniques throughout the whole framework. AutoML can be used for data processing, feature making and selection, algorithm selection, hyperparameter optimisation and model combinations. There are a lot of automated hyperparameter optimisers created to best search for the optimal hyperparameter set. They have different beliefs on how to find the most optimal hyperparameter set. Two of the most used and straightforward hyperparameter optimisers are grid search and random search. These optimisers make a grid of combinations of all hyperparameters, also called samples. They then go through the grid and put samples into the objective function, which gives scores, where the best score is stored. They keep trying out samples until they converge or reach the iteration limit. The difference between these two optimisers is that grid search walks iteratively through the grid, whereas random search goes randomly through the grid, as the name suggests.

The characteristic that makes these two optimisers inefficient is that their methods do not store previous results, which causes them to search in the search space without having a direction towards the global optimum. This results in the methods being reasonably quick one time and slow the other time. Elshawi et al. (2019) find that Bayesian

optimisation is widely used for deep learning. This method contrasts the former HPOs in terms of the ability to keep track of previous results. Bayesian optimisation is a directed approach, where it uses previous results to choose which sample to evaluate next.

2.4.3 Bayesian optimisation

Bayesian optimisation is mainly used for complex objective functions, therefore, it is suggested for the hyperparameter optimisation of DLMs, which have complex objective functions. The random or grid search can be as good and probably quicker for less complex objective functions. However, having a direction for more complex objective functions speeds up the process and give a more consistent result.

As Bayesian optimisation suggests, the method is based on Bayesian beliefs. Bayesian optimisation makes use of a posterior. This posterior consists of the likelihood that a value occurs times the prior, where the prior gives a probability that is given on prior beliefs about an event. The prior of the Bayesian optimisation method is set by the method itself. The likelihood is updated every time the objective function evaluates a new sample. This way, we store more samples with their score from the objective function every time it is called. Therefore, the likelihood contains the probability of what score specific samples would have. The posterior now represents everything we know at a certain point about the objective function, making the posterior an approximation of the objective function.

The posterior is used as a surrogate for the objective function. This means that the posterior can function as a substitute for the objective function because it approximates the objective function. In this way, the objective function does not have to be called every time, which would have been very time-consuming. Instead, the less time-consuming posterior can be called, which contains all information available about the objective function at that point. So, the posterior finds the next sample to put into the objective function. In our case, the sample is a set of hyperparameters. This sample should be the optimal choice following the posterior. To get this optimal sample, we use a so-called acquisition function. An acquisition function leads the search for the optimal sample. Here the acquisition function assesses the performance of candidate samples using the posterior.

The acquisition function uses these results from the candidate samples to choose the

direction of the search. The optimal sample following the acquisition function is then used as input for the objective function. The objective function returns a score to update the likelihood and the posterior. The posterior gets more information on the optimal sample by doing this. This procedure of calling the objective function and updating the posterior is done until the optimisation convergence to a specific score or the maximum iteration is reached. Whereafter sample of hyperparameters corresponding with the optimal score is used for the DLM.

This procedure shows two characteristics. The first is not calling the objective function every time, reducing computation time. Secondly, the posterior contains beliefs on the data, which gives us a direction instead of searching randomly. This makes the method more consistent in time instead of non-directional methods.

For the implementation of the Bayesian optimisation method, BayesianOptimisation by Keras (Chollet et al. (2015)) is used[1]. We have to specify some components before the BayesianOptimisation tuner can be used. First, the domain of each hyperparameter should be specified to create the search space on which the optimisation takes place. The loss function is set to binary cross-entropy since our response variable, the sign of return, is a binary variable. Furthermore, an optimisation metric is needed. Here accuracy is used as a metric. The Adam optimisation algorithm is used as an optimisation algorithm. The Adam optimisation algorithm falls under stochastic gradient descent algorithms (Kingma and Ba (2014)). However, the Adam optimisation algorithm allows for a changing learning rate instead of a fixed learning rate like a classical stochastic gradient descent algorithm. In this way, the Adam optimisation algorithm is overall more efficient in training than other optimisation algorithms, leading to lower computation times. Lastly, the number of trails for finding the optimal hyperparameter set is set to 1000. We do this to get a high chance of finding the optimal hyperparameter set while still being reasonable in computational time. Because the batch size is a hyperparameter which is specified before running the BayesianOptimisation. We perform the entire BayesianOptimisation thirty times with thirty different batch sizes ranging from one to thirty.

A note to make on the results of the DLMs is that a specific hyperparameter set can give different in and out-of-sample performance when trained with the same batch size

[1]More information about the settings of the BayesianOptimisation tuner from Keras can be found on https://keras.io/api/keras_tuner/tuners/Bayesian/Bayesianoptimization-class (Consulted on 2021/12/20)

and the number of epochs. Here the number of epochs is the number of times that the entire training dataset is passed through a DLM while training and the batch size is the size of the subsets in which the entire training dataset is divided. The subsets are feed to the DLM sequentially in a particular epoch. Due to the change in performance for one hyperparameter set, the optimal set can perform well or not depending on the training performed. This occurs because the DLM learns randomly, which leads to different outcomes. To deal with this property, we train the DLM with the optimal hyperparameter 100 times. Here we evaluate every DLM model and save the best DLM model in terms of accuracy on the training dataset until that point. After the 100 separately trained DLM models are evaluated, the top-performing DLM model is used to predict the test dataset.

2.4.4 Hyperparameters and their domain

Before running the optimiser, one needs to specify the search space in which the optimiser should look for the optimal hyperparameter set. The search space contains all combinations of hyperparameters. Here the search space depends on the domains for each hyperparameter. In Table 1, the hyperparameters of the DLMs are presented together with their domain. Some hyperparameters such as weight initialisation and decay rate are specified by Keras (Chollet et al. (2015)) itself and, therefore, are not included in the list. For more information regarding all hyperparameters and settings of the models or layers, Keras provides an overview[2]. Table 1 includes hyperparameters that have been given a specific domain or are important to note.

For all DLMs, the number of layers of a specific kind range between two and five. We start at two layers to ensure a 'deep' model instead of a shallow neural network with one layer. Another aspect to note is that the terminology for neurons in Keras is computational units, short units. Therefore, the number of units have to be specified for each layer instead of the number of neurons. For CNN, it can be seen that the kernel size is set to one. The kernel size has to equal the dimension of the convolutional layer, which is, in our case, one-dimensional.

LSTM contains only one dense layer because it is used at the end to format the

[2]All hyperparameters and settings of the layers present in one or more of the models are provided by Keras. For the dense layer: https://keras.io/api/layers/core_layers/dense/ (Consulted on 2021/12/20). For the one-dimensional convolutional layer: https://keras.io/api/layers/convolution_layers/convolution1d/ (Consulted on 2021/12/20). For the LSTM layer: https://keras.io/api/layers/recurrent_layers/lstm/ (Consulted on 2021/12/20)

outcome of LSTM such that it is interpretable. Furthermore, both the activation and recurrent activation function of the LSTM layers are fixed due to the recommendation by Keras to use sigmoid and tanh, respectively.

Table 1: The hyperparameters included in the DLMs and their domains. 'in layer' means that these hyperparameters are set separately for each layer.

Hyperparameters of DMLP	**Search space**	**Hyperparameters of CNN**	**Search space**	**Hyperparameters of LSTM**	**Search space**
Number of dense layers	{2,...,5}	Number of convolutional layers	{2,...,5}	Number of LSTM layers	{2,...,5}
Number of units in layer	{1,...,200}, with steps of 10	Number of filters in layer	{2,...,200}, with steps of 10	Number of units in layer	{2,...,100}, with steps of 10
Activation function in layer	{elu, exponential, gelu, sigmoid, relu, selu, softmax, tanh}	Activation function in layer	{elu, exponential, gelu, sigmoid, relu, selu, softmax, tanh}	Activation function in layer	{tanh}
Learning rate	{0.01,0.001,0.0001}	Kernel size in layer	{1}	Recurrent activation function in layer	{sigmoid}
Batch size	{1,...,30}	Pooling layer	{max-pooling}	Number of dense layer	{1}
Number of epochs	{1,...,200}	Number of dense layers	{2,...,5}	Learning rate	{0.01,0.001, 0.0001}
		Learning rate	{0.01,0.001,0.0001}	Batch size	{1,...,30}
		Batch size	{1,...,30}	Number of epochs	{1,...,200}
		Number of epochs	{1,...,200}		

2.4.5 Data separation

For optimising a DLM, a training dataset is required. This training dataset is used for both hyperparameter optimisation and model training. Here training is done using the optimal hyperparameter set provided by the hyperparameter optimisation. An important point to keep in mind while training a DLM for hyperparameter optimisation and model training is over-or underfitting. Overfitting occurs when one trains the DLM to extensive. The DLM will then be set up too specific for the training dataset. This gives optimal performance for the training dataset, where the DLM has modelled the general information, details and noise in that data. However, it will underperform on the test data because the details and noise in the test data are different from that of the training data, which causes disturbance. On the other hand, we have underfitting, which addresses the opposite of overfitting. Here the DLM is trained insubstantially, causing no information to be modelled. This leads to weak performance on both training and test data.

DLMs should be trained carefully to avoid overfitting. The way data is used while training can help to overcome overfitting. For instance, one can choose to use a resampling technique to minimise the chances of overfitting. Here resampling is used to train and test the model on multiple subsets. Using multiple train and test parts within training makes the model less likely to overfit the training data since it calculates its performance on the test parts instead of the training parts. We prefer the more straightforward application of a validation dataset. However, both techniques can be applied.

The validation data is part of the training dataset. In this way, the model is trained on the training data and tested on the validation data. This ensures that the out-of-sample accuracy of the model is maximised instead of the in-sample accuracy. For DLMs, we include a validation part in the hyperparameter optimisation and model training. The first 30% is used as training data for the hyperparameter optimisation, and the following 20% is used as validation data. The model training is done on the first 50% of the data, whereas the following 20% is used to validate the model. No validation data is necessary for the linear and stacked model because those models do not encounter overfitting. Therefore, the first 70% is entirely used as training data. The remaining 30% is used for testing the out-of-sample performance of all the models. Generally, the training and testing dataset have a ratio of 70/30, 80/20 or 90/10. However, there is no real rule of thumb which to use. We have chosen a 70/30 ratio between training and testing because we have 5259 trading days, 1050 weeks and 262 months. This ensures enough training data with the 70/30 ratio for all intervals. Furthermore, the 70/30 ratio also leaves enough testing data for all intervals to show the out-of-sample performance.

2.5 Stacking

Using multiple models to predict the sign of returns leads to different performances between the models. We can select the best performing model and use this model to predict the future sign of return. Taking the overall best performing model out of the available models is a strategy named cross-validation (Stone (1977)). Another approach can be using the different models together to make a possible improvement upon the overall performance. By combining the models in a certain way, the thoughts about the data from all models can be assessed and used together, resulting in one model. This can be beneficial for the overall performance and can possibly outperform single models.

Combining models with the use of predictions on training data is called stacked generalisation or short stacking. Stacking, first explained by Wolpert (1992), uses a meta-learner to learn from the predictions of the base models to best combine the base models such that the performance is optimised. Stacking differentiates itself from forecast combinations because it uses training predictions of the base models and trains itself to optimal combine the models accordingly. This contrasts forecast combinations, which combines the forecasts without looking at the training predictions of the base models. Witten and Frank (2002) comment on stacking, where they mention that stacking should use models which are built differently. This is because the models then differentiate on beliefs of predicting, which creates a stronger model when combining the base models. We use the three deep learning models presented in this study as base models for stacking. Because these models differ in terms of modelling, stacking can be an interesting way to increase performance in sign predicting of returns against the single deep learning model. The DLMs have some characteristics, which are unique for only one of the DLMs, as stated in Section 2. This can cause a certain DLM to have different beliefs and, therefore, performance than other DLMs. If this holds, and a pattern can be found, it can be valuable to consider.

The linear regression model has not been included because the difference in performance between deep learning models and the linear regression model is of interest. Therefore, a combination of these models does not contribute to the main objective. We did include all models for stacking when modelling to test possible improvements. However, this decreased performance compared to using the three deep learning models. In the remainder of this section, the stacking procedure is presented, whereafter the meta-learner is explained in more detail.

Figure 4 helps to illustrate the entire process of the stacking for the three DLMs. The stacking process consists of four levels from the input to the final model. *Level 0* gets the training dataset as input consisting of the training and validation parts used to train the base models, the three DLMs. We do this in the same manner as when the DLMs are used separately. When these models are trained on the training dataset, the trained models predict the sign of return in the entire training dataset (*Level 1*). These predictions are then used as input for *Level 2*. Here the meta-learner uses these predictions and the actual values of the training dataset as input to learn how to combine the models

to optimise predictions for the training dataset. We use the predictions on the entire training dataset instead of only the validation data due to the limited validation data (especially for the monthly interval). Otherwise, the stacked model was not able to find patterns in the data causing underfitting. In *Level 3*, the meta-learner trained in *Level 2* makes predictions on the test dataset. Here the model created in *Level 3* is called the stacked model because of the stacking of the models.

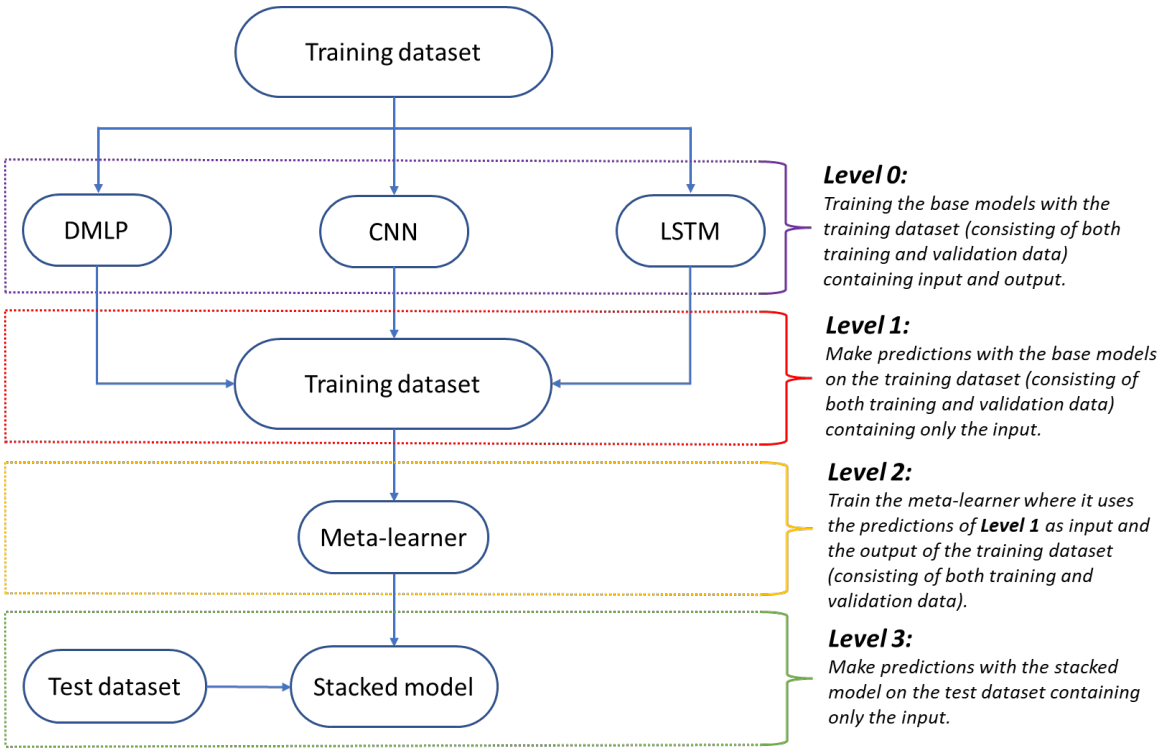


Figure 4: The process of stacking, showing the levels of the stacking model. Here the base models are DMLP, CNN and LSTM.

Meta in meta-learning refers to looking at one level above. For instance, meta-data is data about data, the number of rows or the name of columns. So, meta-learning is learning about learning (Vilalta and Drissi (2002)). Meta-learning in machine learning concerns a model, the meta-learner, which learns from the predictions made on the training dataset of two or more (base) models. Here the base models are three deep learning models. However, it can be any model for that matter. For the meta-learner, multiple kinds of models can be chosen. For instance, one can use a logistic regression, deep learning or machine learning model for classification tasks. A linear regression model can be used when the base models consist of regression models. We make use of a logistic regression as a meta-learner. Here the logistic regression gives probabilities to the prediction of the DLMs and then combines them using their probabilities.

The meta-learner uses the predictions of the base models as input variables and the actual values as the output, which the meta-learner should predict. In this way, the meta-learner learns to combine the predictions of the base models to optimise the classification problem in the training dataset. We then expect the pattern of combinations to be consistent for the test dataset, where the meta-learner should help increase classification accuracy.

3 Data

3.1 Time series

We use the S&P500 index (SP) as a time series to predict the sign of returns. We choose an index as a financial product for the study because of the available historical data and its importance and diversified portfolio. SP consists of nearly 500 top large-cap stocks in the United States. These stocks are either listed on the NASDAQ, NYSE or Cboe BZX Exchange. SP, one of the oldest indices (from 1957), comprises market-cap-weighted stocks. SP's prime stocks are Microsoft, Apple and Amazon (S&P Dow Jones indices (2021)).

We have retrieved the data of SP on the 1st of June 2021 from the Center for Research in Security Prices (CRSP) of the platform of the Wharton Research Data Service (2021) (WRDS). The data of SP contains daily close price data and daily close, value and equal-weighted returns. Here the daily close returns (r_d) are constructed from the daily close price

$$r_{d,t} = \frac{cp_t}{cp_{t-1}} - 1, \tag{13}$$

where cp_t is the current close price, cp_{t-1} is the previous close price and $t \in \{1, \ldots, n\}$ with n being the data length. The daily return, $r_{d,t}$, is used to construct both weekly and monthly SP returns because we are interested in the model performance on multiple intervals. The interval is set to five days for the weekly return to approximately equal one trading week. For the monthly return, the interval is set to four trading weeks of five days, which equals twenty days. This leaves us with a weekly and monthly return (r_w, r_m) constructed by taking the product of returns in that interval. After the construction of the returns, they are transformed into signs ($s_{i,t}$ for $i \in d, w, m$) for the sign prediction

application. This is constructed following

$$s_{i,t} = \begin{cases} 1 & \text{if } r_{i,t} \geq 0, \\ 0 & \text{if } otherwise. \end{cases} \tag{14}$$

SP data used ranges from the 1st of January 2000 until the 31st of December 2020. The first couple of days is used for feature construction. This is the case for constructing the technical indicators. For example, a simple twelve-day moving average can only start at day twelve because it does not have enough trading days to be calculated when starting prior. This results in an interval for the final dataset, which starts later. Table 2 shows the separation of the dataset following Section 2.4. Here the time periods are presented together with the number of data points.

Table 2: The data separation of the S&P500 index. Separated in training, validation, and test data used for hyperparameter optimisation and model training. Including the time period and number of data points in that time periods. HP stands for hyperparameter.

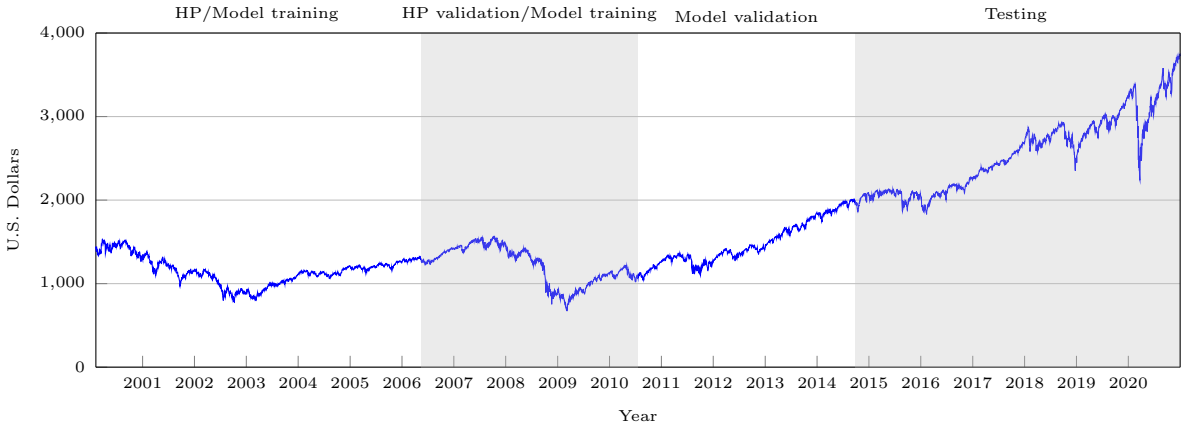
Data Section & period	**Time period & Data points**					
	Daily		Weekly		Monthly	
HP training 0-30%	08/02/2000 - 18/05/2006	1578	23/02/2000 - 23/05/2006	315	29/03/2000 - 14/06/2006	79
HP validation 30-50%	19/05/2006 - 23/07/2010	1052	31/05/2006 - 26/07/2010	210	13/07/2006 - 02/08/2010	52
Model training 0-50%	08/02/2000 - 23/07/2010	2630	23/02/2000 - 26/07/2010	525	29/03/2020 - 02/08/2010	131
Model validation 50-70%	26/07/2010 - 25/09/2014	1051	02/08/2010 - 25/09/2014	210	30/08/2010 - 18/09/2014	52
Model testing 70-100%	25/09/2014 - 31/12/2020	1578	02/10/2014 - 28/12/2020	315	16/10/2014 - 28/12/2020	79

Figure 5 presents the daily price and return of SP from 8th of February 2000 until 31st of December 2020, together with the time periods used for certain parts of the model making. There is an overall positive trend in the daily price. Furthermore, the daily return shows that SP had relatively high volatility in its first period from 2000 until 2002 in combination with a downwards movement in the price. This can be related to the Dot-com bubble followed by the September 11 attacks. The next volatile period, which also has a downward trend in the price, was during the financial crisis from 2007 until 2009. The last and recent volatile period was due to the COVID-19 recession, which dropped SP radically, whereafter SP recovered.

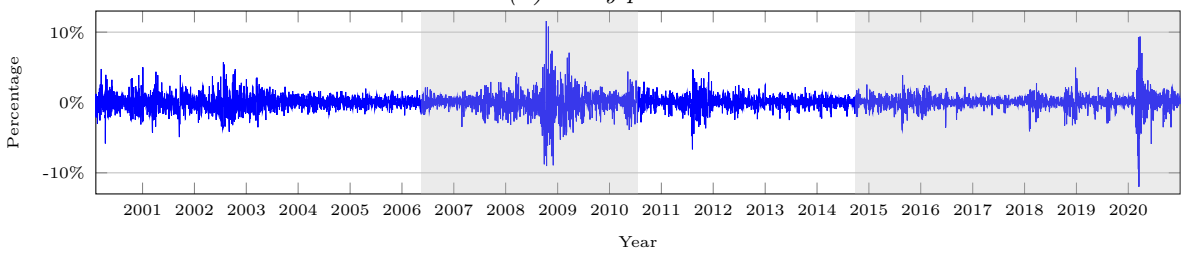
The pattern for larger intervals, weekly and monthly, remains the same as for the daily interval. However, overall the absolute returns become more stable when the interval is increased. The interval contains multiple days with high positive and negative returns,

which even each other out. Although, there are still a few large, primarily negative, returns for the larger interval. The return plots for all intervals are stated in Appendix A.1, Figure 9

For the daily returns, it can be seen that volatility clustering is appearing, where large absolute returns tend to follow each other up and equally for small absolute returns. For the daily interval, the observations from the returns display low persistence, where negative and positive returns occur sequentially. However, the SP has stronger persistence for a larger interval, as a monthly interval. This pattern is substantiated by the price of SP shown in Figure 5a, where positive trends are visible for multiple years sequentially. Furthermore, it can be seen that there is an overall negative trend in volatile periods most of the time.



(a) Daily price.



(b) Daily return.

Figure 5: The daily price and return of the S&P500 index in U.S. dollars. The data ranges from the 8th of February 2000 until the 31st of December 2020. Together with the time periods of training, validation, and testing data. HP stands for hyperparameter.

Table 3 shows the summary statistics of SP. Here an increase in all statistics is visible when increasing the interval. The annual Sharpe ratio over the entire period is equal to 0.24. This is a relatively low Sharpe ratio, due to the first period of SP, where SP has a low return and relatively high volatility. This results in an annualised average return of 4.73% with an annualised average volatility of 19.79%.

Table 3: The summary statistics of the return of the S&P500 index from the 8th of February 2000 until the 31st of December 2020. Including Mean, standard deviation (std), minimum (min) and maximum (max) for daily, weekly and monthly intervals. Here the weekly and monthly returns are constructed following Section 3.1. Bold font means that the mean does not significant differ from zero at with an alpha of 0.05.

Statistic	Daily return	Weekly return	Monthly return
mean	**0.0263%**	**0.134%**	**0.541%**
std	1.25%	2.58%	4.99%
min	-11.98%	-18.76%	-29.39%
max	11.58%	12.69%	16.86%

3.2 Features

As input data for the feature selection method, we construct a feature set, including technical indicators, macroeconomic variables, and SP's underlying stocks. Furthermore, the one-period lagged price, return (sign, actual, value and equal-weighted) from SP are also added to the feature. All these features are included in the feature set because they can have a relationship between the sign of return and its one-period lagged value. SP data is included due to the possible relationship between the sign of return and the one-period lagged values of SP price and return data. For example, Yang, Gong, and Yang (2017) find that one-period lagged price data can be a good predictor for the Chinese stock market index.

All the features are made using daily data, where the features of the weekly and monthly intervals are constructed from daily features. The features are normalised to remove the dominance of specific features in the set of features. The features set itself is then combined with the sign of return, where the feature set is one-period lagged. Instead of using all features, feature selection is applied.

3.2.1 Technical indicators

The first class of features we add to the feature set are technical indicators. Table 4 contains all the technical indicators. These technical indicators originate from Dai, Zhu, and Kang (2021) and Kara, Boyacioglu, and Baykan (2011). Dai et al. (2021) construct three new technical indicators: n-day moving average short-long, n-day moving median and n-day momentum. These technical indicators outperform existing moving averages and momentum features in their research. Kara et al. (2011) include technical indicators from experts in the field. Here we take their technical indicators constructed using close

price due to the unavailability of open, high, and low prices in WRDS. Therefore, we construct a total of eight technical indicators where they are constructed for multiple intervals, as can be seen in the **Range** column of Table 4.

Table 4: The technical indicators of both Kara et al. (2011) and Dai et al. (2021), containing the technical indicators name, formula and range.

Technical indicators	**Formulas**	**Range**
Simple n-day moving average (SMA_n)	$\frac{\sum_{i=0}^{n-1} cp_{t-i}}{n}$	$n = 1, 3, 6, 9, 12$
Weighted n-day moving average (WMA_n)	$\frac{\sum_{i=0}^{n-1}(n-i)cp_{t-i}}{\sum_{i=1}^{n} n}$	$n = 3, 6, 9, 12$
n-day moving average short-long($NMA_{s,l}$)	$\frac{\sum_{i=0}^{s-1} cr_{t-i}}{s} - \frac{\sum_{i=0}^{l-1} cr_{t-i}}{l}$	$s = 1, 2 \quad \& \quad l = 6, 9, 12$
New n-day moving median ($NMM_{s,l}$)	$Median\left[\{cr_t\}_{t-s-1}^{t}\right] - Median\left[\{cr_t\}_{t-l-1}^{t}\right]$	$s = 1, 2 \quad \& \quad l = 9, 12$
n-day Momentum (M_n)	$cp_t - cp_{t-n}$	$n = 1, 3, 6, 12$
New n-day momentum (NM_n)	$cr_t - cr_{t-n}$	$n = 1, 3, 6, 12$
Relative Strength Index (RSI_n)	$100 - \frac{100}{1+\frac{\sum_{i=0}^{n-1} up_{t-i}/n}{\sum_{i=0}^{n-1} dw_{t-i}/n}}$	$n = 1, 3, 6, 9, 12$
Moving average convergence divergence ($MACD_{s,l}$)	$ema_{s,t} - ema_{l,t}$, $ema_{i,t} = \alpha \times cp_t + (1-\alpha) * ema_{i,t-1}$	$s = 12 \quad \& \quad l = 24$

3.2.2 Macroeconomic variables

The next class of features are macroeconomic variables. We add macroeconomic variables because they can capture the characteristics of the economy. This is useful because SP contains a large set of stocks on U.S.-based markets, which should be affected by changes in the economy.

The macroeconomic variables are retrieved from the Federal Reserve Bank of St. Louis (2021) (FRED). Table 5 presents all macroeconomic variables. The covered macroeconomic subjects are treasury bills and maturity rates, foreign exchange rates, bank prime loan rate and the volatility index. These subjects help explain changes in economic expectations. Rapach, Wohar, and Rangvid (2005) find that macroeconomic variables can be used in predicting returns using a large set of macroeconomic variables. Macroeconomic variables such as inflation, gross domestic product, unemployment rate and interest rate are not considered because these variables are only updated monthly in the FRED economic data centre. Table 5 shows all the used macroeconomic variables. We also include their first differences to account for possible trends in the features.

Table 5: The macroeconomic variables, including the name and code.

Macroeconomic variables	Code
3-Month Treasury Bill: Secondary Market Rate	3MTB
6-Month Treasury Bill: Secondary Market Rate	6MTB
1-Year Treasury Constant Maturity Rate	1YTCMR
5-Year Treasury Constant Maturity Rate	5YTCMR
10-Year Treasury Constant Maturity Rate	10YTCMR
20-Year Treasury Constant Maturity Rate	20YTCMR
U.S. / U.K. Foreign Exchange Rate	US_UK
U.S. / EURO Foreign Exchange Rate	US_EUR
Bank Prime Loan Rate	BPLR
NASDAQ Composite Index	NASDAQ
Volatility Index S&P500	VIX

3.2.3 Underlying stocks

The last class of features are SP's underlying stocks. SP contains around 500 stocks of U.S. companies. Therefore, examining relationships between underlying stocks and SP can be valuable. For example, Niaki and Hoseinzade (2013) and Zhong and Enke (2017) also include data from stocks from SP in their feature set to predict the sign of the return of SP.

We take all underlying stocks in SP for at least the beginning of 2000 until the end of 2020. We choose these underlying stocks since they are included for the entire period we examine in this study. This is needed to make predictions with this class of features. The resulting underlying stocks are given in Appendix A.1, Table 17. The list consists of 204 stocks, where we use their daily prices and returns. Here the weekly and monthly prices and returns are constructed from the daily data. All prices and returns are one-period lagged to predict the sign of return.

4 Feature selection

The feature set contains 468 features from which we do not know for sure if they have a valuable relationship with the sign of return series. Therefore, using all features might be unnecessary, increases computation time, or negatively influences predictions. The negative influence on predictions when using all features can occur when having non-informative features in the input, disturbing the accuracy of the prediction. Therefore, we perform feature selection to select those features which contain the most information on the future sign of returns. For the feature selection, we use an AutoML method called

SelectKbest[3] provided by the python module, Scikit-learn (Pedregosa et al. (2011)). Feature selection methods focus on removing features non-informative (Kuhn and Johnson (2013)). Furthermore, Chandrashekar and Sahin (2014) state that feature selection can improve accuracy.

SelectKbest selects the k best features out of the feature set. The method uses the p-values from all features resulting from the ANOVA F-test, the default setting for ranking features for classification tasks. SelectKbest then transforms the p-values to a score used to rank the features. We used different values for k to evaluate which size of feature set would perform best. We find that increasing k higher than ten does not increase overall performance significantly. Although, increasing the number of features in the feature set significantly increased the computation time. Furthermore, a larger feature set would also make it more difficult to retrieve a well-optimised model in hyperparameter training and model training due to the increased dimensions. Supported by both drawbacks, we set k to ten and use ten features for each interval.

The feature selection method constructs different feature sets for all intervals, daily, weekly, and monthly. The feature sets are given in Appendix A.2, Table 18. The selection method is performed on the training data to ensure that no information from the 'unknown' test data is used. For the daily interval, different categories of features are selected by SelectKbest. Here the one-period lagged return of SP is present, with the sign, value-weighted return and the return itself. Furthermore, two technical indicators are used, simple one-day moving average and one-day momentum. The other features are one-period lagged returns of underlying stocks. For the weekly interval, the features selected are the one-period lagged prices of underlying stocks, together with two macroeconomic variables, the ten- and twenty-year treasury constant maturity rate. The selected features for the monthly interval consist of one-period lagged returns of underlying stocks and the ten-year treasury constant maturity rate. The one-period lagged prices, and returns of the underlying stocks are the majority of the selected features among all intervals. None of the underlying stocks is present more than ones among all intervals.

[3]More information regarding the settings of SelectKbest from Sklearn can be found on https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (Consulted on 2021/12/20)

5 Performance measures

In this study, the prediction of the sign of return is our main objective. The interest in the sign of return instead of the actual value is because one makes a profit when correctly predicted the sign. Here the actual value only adds to the amount of profit or loss. Furthermore, there is a performance difference between sign and point prediction models. Leung, Daouk, and Chen (2000) find by an empirical experiment that sign prediction (two-class classification/binary classification) models performed better than point (or level) prediction models when looking at the prediction of the direction of the stock market. This is due to the increasing prediction error when predicting the exact value instead of its sign. This is because it is harder to create a model that identifies future values exactly. Therefore, incorrect directional predictions are expected to be made more frequently when making point predictions. Leung et al. (2000) also find that sign prediction models got higher returns than point prediction models in a trading game. In line with these findings, Sezer et al. (2020) point out that most studies in their review use sign prediction instead of point prediction. Due to these properties, sign prediction is used throughout the study.

To be able to measure the sign prediction performance of the linear regression model, we need to transform predictions of the model. Here we transform the predictions into binary results (above or equal to 0.5 is positive, below 0.5 is negative). 0.5 is chosen as a threshold due to the positive returns receiving 1 and negative returns receiving 0. In this way, the threshold of 0.5 is the centre of both options. After transformation, the performance measures can be applied to the linear regression model, the DLMs and stacked model.

In this section, the performance measures used to evaluate the performance of the models on sign prediction are presented. We focus on both prediction and financial performance since models can perform differently in both areas. For example, a model can outperform other models in predicting performance while under-performing in terms of financial performance. This can occur when the correct predicted signs of return are small profits. Whereas a model with lower prediction performance, can correctly predict certain high returns, leading to an overall higher financial performance. In the following of this section, the prediction measures are presented first. Then, the trading game, together with its measures, are explained.

5.1 Prediction measures

The confusion matrix, presented in Table 6, is a matrix that contains all information about the made predictions. The confusion matrix captures the number of true-positive (TP), false-positive (FP), true-negative (TN) and false-negative (FN) predicted values. In this way, the confusion matrix reports every aspect of the predicting performance. In the result section (Section 6), the confusion matrix is presented with percentages instead of raw numbers to compare the confusion matrix between intervals more easily.

Table 6: The confusion matrix. TP is true positive, FP is false positive, TN is true negative, and FN is false negative.

		Prediction	
		Positive	Negative
Actual	Positive	*TP*	*FN*
	Negative	*FP*	*TN*

The prediction measure we use for classification is the classification accuracy (A), which calculates the percentage of correctly predicted signs of return

$$A_{m,i} = \frac{TP_{m,i} + TN_{m,i}}{TP_{m,i} + FP_{m,i} + TN_{m,i} + FN_{m,i}} \times 100\%, \tag{15}$$

where subscripts m and i specify the model used and the corresponding interval, respectively.

To assess the precision and robustness of a model, the F1-score can be used. The F1-score is calculated using the metrics precision and recall. Precision (P) assess the proportion of positive predictions which were correctly predicted

$$P_{m,i} = \frac{TP_{m,i}}{TP_{m,i} + FP_{m,i}}. \tag{16}$$

Recall (R) examines the robustness of the model, where it gives the proportion of correct positive predictions against all actual positive values (both correctly and incorrectly predicted)

$$R_{m,i} = \frac{TP_{m,i}}{TP_{m,i} + FN_{m,i}}. \tag{17}$$

The F1-score combines both the precision and recall

$$F1_{m,i} = \frac{2}{\frac{1}{P_{m,i}} + \frac{1}{R_{m,i}}}, \tag{18}$$

where the metric ranges between 0 and 1. In this way, a perfect model receives an F1-score of 1.

Additionally, we use the McNemar test, a nonparametric test for two related samples. The McNemar test is used to examine if one model performs significantly different than another model. It is recommended by (Dietterich (1998)), a widely used paper of statistical hypothesis tests for comparing classification models, due to the low chance of a type 1 error. Kim (2003) uses the McNemar test to examine if one of his machine learning models outperforms others in predicting financial prices. Using the McNemar test for examining performance differences can be done when combining it with accuracy. When the model has higher accuracy and a significant McNemar test statistic, the model significantly outperforms another model.

The McNemar test introduced by McNemar (1947) uses information stated in a contingency table. This table, presented in Table 7, contains four cells. Cells 1 and 4 contain the number of times both models predicted correct and incorrect, respectively. Cell 2 gives the number of times that model 1 predicts correctly while model 2 predicts incorrectly and vice versa in cell 3. Model 1 outperforms model 2 when cells 2 and 3 differ significantly, and the accuracy of model 1 exceeds that of model 2. The corresponding null hypothesis is that cells 2 and 3 do not differ significantly. Mathematically, the test statistic with continuity correction (-1 term) follows Equation 19, where a is equal to cell 2 and b to cell 3. The degree of freedom of the chi-squared distribution is 1.

$$\chi_1^2 \sim \frac{(|a-b|-1)^2}{a+b} \tag{19}$$

Table 7: The contingency table. Correct and incorrect represent that a specific model has predicted correctly or incorrectly, respectively.

		Model 1	
		Correct	incorrect
Model 2	Correct	*1*	*2*
	incorrect	*3*	*4*

5.2 Financial measures

A trading game is played with the models to show their performance in terms of financial performance. Multiple papers on return prediction include a trading game, where they use their models to make an autonomous trading strategy and test its performance. For example, Fleming, Kirby, and Ostdiek (2003), Deng, Bao, Kong, Ren, and Dai (2016), Dixon, Klabjan, and Bang (2017), Sezer and Ozbayoglu (2018) include a trading game in

their research on return or price prediction. The papers have different views on which real-world restrictions and features they should include. For instance, Fleming et al. (2003) and Deng et al. (2016) include transaction costs, and Sezer and Ozbayoglu (2018) include transaction costs and volume restrictions. However, Dixon et al. (2017) do not include any restrictions or features. Our trading game does not include real-world restrictions due to the available data, which does not allow for the restrictions to be calculated.

The trading game is done with the return data from the test dataset and captures the performance of the models as autonomous trading strategies. The autonomous trading strategy is able to go long and short. The autonomous trading strategy goes long, so buy, when predicting positive returns (where positive means zero or higher). Similar, the strategy goes short, so sell, when predicting negative return.

The profit made by the model is presented in percentages and in a plot of PnL over time. The PnL plots show the performance throughout the test dataset. In this way, we can assess the performance over time.

The calculation of profit is only done using percentages, where we start at 0% profit. To calculate the profit, the returns are multiplied by each other following

$$P_{m,i,T} = ((\prod_{t=1}^{T}(1 + r_{i,t} * S_{m,i,t})) - 1) \times 100\%, \tag{20}$$

where $P_{m,i,T}$ is the profit of model m for interval i at time T. Here only known returns and predictions are used to calculate $P_{m,i,T}$. $r_{i,t}$ is the real return for interval i on time t and $S_{m,i,t}$ being the predicted sign of the return of model m for interval i at time t. Here the sign is 1 when predicted positive (non-negative) and -1 when predicted negative.

The Sharpe ratio is used to extend the examination of financial performance. The Sharpe ratio is an extensively used metric to classify financial performance (Ledoit and Wolf (2008), Sharpe (1994)). The Sharpe ratio of a model m follows

$$SR_m = \frac{\mu_m - \mu_{\text{rf}}}{\sigma_m}, \tag{21}$$

where μ_m is the average return made by the model, μ_{rf} is the average risk-free rate calculated using the one-year treasury rate of each year, and σ_m the standard deviation of the returns made by the model. We make use of the annual Sharpe ratio of a model m, $SR_{\text{annual},m}$, which is SR_m multiplied by the square root of the number of trading moments in a year, n,

$$SR_{\text{annual},m} = SR_m \times \sqrt{n}. \tag{22}$$

In this study, n is 252 for the daily interval, n is 50 for the weekly interval and n is 13 for the monthly interval.

6 Results

This section presents the out-of-sample results and comparisons between the models and intervals. First, we assess the performance of the models in terms of prediction performance using the confusion matrix, accuracy, F1-score and the McNemar. Here the results are presented per type of model. Whereafter, they are compared. Second, the financial performance is assessed using profit and loss and the annual Sharpe ratio, which is done per interval, ending with an overall comparison of the financial performances.

6.1 Prediction performance

6.1.1 Linear regression model

We start by analysing the results of the linear regression model. The model uses the features selected by the feature selection method (Section 3.2). Table 8 shows the accuracy and the F1-score of the linear regression model, together with precision and recall, used to calculate the F1-score. The accuracy of the linear regression model rises with the size of the interval. The accuracy is always above the accuracy of a coin flip (50%) and increases over ten percentage points between the daily and monthly intervals.

The F1-score shows the same pattern as the accuracy. This means that the linear regression model becomes a better model when the interval increases. The model especially has a high recall rate for the weekly and monthly intervals. This observation suggests that the model is robust. However, with a high recall rate and a precision rate close to the accuracy, it shows that the model predicts mainly positive returns.

Table 8: The prediction measures of the linear regression model for the out-of-sample test dataset for the daily, weekly and monthly intervals.

Interval	Accuracy	F1	Precision	Recall
DAILY	55.1%	0.666	0.560	0.823
WEEKLY	63.5%	0.777	0.635	1
MONTHLY	67.1%	0.800	0.675	0.981

That the linear regression model mainly predicts positive returns is visible in table 9, which provides the confusion matrices in percentages of the linear regression model for the different intervals. The model predicts around 80% positive for the daily interval. At the same time, the model predicts towards 100% positive for the weekly and monthly interval. This pattern for the weekly and monthly model can be related to three points. First, it can be attributed to the percentage of positive returns compared to negative returns. This percentage is around ten percentage points higher for the weekly (63.5%) and monthly (66,3%) than for the daily (54.4%). Therefore, it can be that the linear regression model cannot achieve higher accuracy with more negative predictions than predicting (almost) only positive returns. Second, as the interval increases from daily to monthly, the number of data points in the training dataset decreases. This can affect the model's performance because less training data is available. Third, because the linear regression model is incapable of capturing nonlinear relationships correctly, (important) nonlinear relationships are not included. Here we expect DLMs to be able to capture these nonlinear relationships.

Table 9: The confusion matrices in percentages of the linear regression model for the out-of-sample test dataset for the daily, weekly and monthly intervals.

(a) Daily interval.

		Prediction	
		Positive	Negative
Actual	Positive	44.8%	9.6%
	Negative	35.2%	10.3%

(b) Weekly interval.

		Prediction	
		Positive	Negative
Actual	Positive	63.5%	0%
	Negative	36.5%	0%

(c) Monthly interval.

		Prediction	
		Positive	Positive
Actual	Positive	65.8%	1.3%
	Negative	31.6%	1.3%

6.1.2 Deep learning models

The results of the DLMs are presented together to get an overall view of their performance. We achieve the results using the DLMs with their best performing hyperparameter sets (in the training environment). These hyperparameter sets are given in Appendix A.2, Table 19. The hyperparameter sets differ between DLMs and the intervals. However, the number of a certain kind of layer (dense, LSTM, convolutional and fully connected layers) is two for each DLM, excluding the closing layer of the dense layers. When increasing the number of layers, the model's performance decreased. This can be due to the hyperparameter space and model becoming larger when increasing the number of layers. When the hyperparameter space becomes larger, hyperparameter optimisation becomes more difficult due to more possible hyperparameter sets. Furthermore, when the model itself becomes larger, training becomes more difficult since more parameters

have to be optimised while training.

Table 10 provides the prediction measures of the DLMs. From the results, there is no clear winning DLM for all intervals. However, the DMLP outperforms the other DLMs for the weekly and monthly intervals in terms of accuracy. The CNN performs marginally better than the other DLMs on the daily interval. Furthermore, all DLMs outperform the accuracy of a coin flip, where the accuracy of the DMLP model for the monthly interval is the highest at 70.9%.

For the DMLP and LSTM model, it holds that an increase in the interval increases the accuracy of the DLM. However, the CNN models do not follow such a pattern due to a lower accuracy for the monthly interval. There is a similar pattern in terms of the F1-score.

Table 10: The prediction measures of the DLM models for the out-of-sample test dataset for the daily, weekly and monthly intervals.

(a) DMLP

Interval	**Accuracy**	**F1**	**Precision**	**Recall**
DAILY	52.7%	0.528	0.578	0.487
WEEKLY	63.5%	0.777	0.635	1
MONTHLY	70.9%	0.803	0.734	0.887

(b) CNN

Interval	**Accuracy**	**F1**	**Precision**	**Recall**
DAILY	54.4%	0.641	0.561	0.747
WEEKLY	63.2%	0.768	0.640	0.96
MONTHLY	58.2%	0.703	0.672	0.736

(c) LSTM

Interval	**Accuracy**	**F1**	**Precision**	**Recall**
DAILY	53.7%	0.589	0.570	0.609
WEEKLY	54.0%	0.670	0.615	0.735
MONTHLY	60.8%	0.687	0.739	0.642

Table 11 shows the confusion matrices in percentages of all the DLMs and intervals. It shows that the DLMs are capable of predicting negative returns. The DLMs predict overall more positive than negative returns, which is in line with the percentage of positive returns. An exception is the DMLP model for the weekly interval. Here the model only predicts positive returns.

That the DMLP is predicting only positive can be due to the training section of DLMs. As explained in Section 2.4, the predictions made by a DLM with a specific hyperparameter set can be different for two training procedures due to the self-learning property of the model during training. Therefore, it can be that the DMLP model with the optimal hyperparameter set could have predicted better. However, a better predic-

tion is not found in the set of trainings that have been performed.

Table 11: The confusion matrices in percentages of the DLM models for the out-of-sample test dataset for the daily, weekly and monthly intervals.

(a) DMLP - daily interval.

		Prediction	
		Positive	Negative
Actual	Positive	26.5%	27.9%
	Negative	19.3%	26.2%

(b) DMLP - weekly interval.

		Prediction	
		Positive	Negative
Actual	Positive	63.5%	0%
	Negative	36.5%	0%

(c) DMLP - monthly interval.

		Prediction	
		Positive	Negative
Actual	Positive	59.5%	7.6%
	Negative	21.5%	11.4%

(d) CNN - daily interval.

		Prediction	
		Positive	Negative
Actual	Positive	40.7%	13.8%
	Negative	31.9%	13.7%

(e) CNN - weekly interval.

		Prediction	
		Positive	Negative
Actual	Positive	61.0%	2.5%
	Negative	34.3%	2.2%

(f) CNN - monthly interval.

		Prediction	
		Positive	Negative
Actual	Positive	49.4%	17.7%
	Negative	24.1%	8.9%

(g) LSTM - daily interval.

		Prediction	
		Positive	Negative
Actual	Positive	33.1%	21.3%
	Negative	25.0%	20.5%

(h) LSTM - weekly interval.

		Prediction	
		Positive	Negative
Actual	Positive	46.8%	16.9%
	Negative	29.0%	7.3%

(i) LSTM - monthly interval.

		Prediction	
		Positive	Negative
Actual	Positive	43.0%	24.1%
	Negative	15.2%	17.7%

6.1.3 Stacking

Table 12 presents the prediction measures for the stacked model. It shows that the model can achieve good accuracy. Here the accuracy increases with the size of the interval. Furthermore, the stacked model has a high recall rate, suggesting robust performance. Despite that, it can be seen that the precision rate is close to the accuracy. Together with the high recall rate, this shows that the stacked model predicted primarily positive returns.

Table 12: The prediction measures of the stacked model for the out-of-sample test dataset for the daily, weekly and monthly intervals.

Interval	**Accuracy**	**F1**	**Precision**	**Recall**
DAILY	54.9%	0.704	0.548	0.986
WEEKLY	62.9%	0.772	0.633	0.990
MONTHLY	67.1%	0.800	0.675	0.981

Table 13 shows the confusion matrices in percentages for the different intervals. It indeed shows that the stacked model predicts primarily positive returns. Less than 3% is predicted negative over all intervals. The reason why the stacked model is incapable of predicting more negative returns can be due to the meta-learner used in the stacked model. It can be that the relationships between the predictions of the base models and the sign of return are difficult to model for this meta-learner. Alternatively, it can be that there are no good relationships present to predict both negative and positive returns.

Table 13: The confusion matrices in percentages of the stacked model for the out-of-sample test dataset for the daily, weekly and monthly intervals.

(a) Daily interval.

		Prediction	
		Positive	Negative
Actual	Positive	53.7%	0.8%
	Negative	44.4%	1.2%

(b) Weekly interval.

		Prediction	
		Positive	Negative
Actual	Positive	62.9%	0.6%
	Negative	36.5%	0%

(c) Monthly interval.

		Prediction	
		Positive	Negative
Actual	Positive	65.8%	1.3%
	Negative	31.6%	1.3%

6.1.4 Comparison of prediction performance

We now compare the prediction performance of the models for each interval separately because it is impossible to compare the models between intervals due to the differences in datasets. The table including all McNemar test statistics is presented in Appendix A.2, Table 20.

For the daily interval, the linear regression model achieves the highest overall accuracy, with 55.1%. Nevertheless, the accuracy is only marginally higher than the stacked model (54.9%). The three DLMs achieve accuracy's between 52.7% and 54.4%. None of the models significantly outperforms the other models for the daily interval following the McNemar test statistics.

Moving on to the weekly interval, the results regarding the prediction performance show that the models perform closely in terms of prediction performance, except the LSTM model with substantially lower accuracy than the other models (54.0%). The top-performing models are the DMLP and linear regression models, with both 63.5%. Both models only predicted positive returns and therefore follow a buy and hold strategy. Following the McNemar test statistics, the LSTM is the only model that significantly underperforms compared to the other models for the weekly interval. For the other models, it holds that they do not differ significantly in terms of prediction performance.

For the monthly interval, the DMLP model achieves the highest accuracy of all models (70.9%). The linear and stacked models follow with an accuracy of 67.1%, where both follow the buy and hold strategy. The CNN and LSTM models perform the lowest in terms of prediction performance. Despite the differences in the prediction performance of the models, it is significant following the McNemar test statistics.

Overall the results show that DLMs are more capable of predicting both positive and negative returns. Whereas the linear and stacked models both mainly predicted positive returns. Furthermore, the stacked model is not capable of improving the performance of

the DLMs. A general note is that the overall prediction performance tends to increase when increasing the interval.

6.2 Financial performance

6.2.1 Daily interval

Following Figure 6, including the PnL plots, the profit of the models change over time, mainly with a positive trend. Furthermore, the models change position in terms of their profit. For instance, the LSTM model performs best until the end of 2017, ends with the lowest profit of 45.9%. The stacked model achieves the highest profit of 176.5% on the end date, which is a substantial difference. Furthermore, the linear regression model shows great profitability, coming close to the profit of the stacked model and outperforms all DLMs in means of profit. A general note is that the most profitable period is the COVID-19 pandemic, which caused high absolute returns (See Figure 9). So, when correctly predicted results in high positive returns.

Table 14: The total profit and annual Sharpe ratio of all models together with the long position of the S&P500 index for the out-of-sample test dataset with daily data.

Model	S&P500 index	LINEAR	DMLP	CNN	LSTM	STACKED
Profit	91.1%	140.5%	54.6%	101.6%	45.9%	176.5%
Annual Sharpe ratio	0.58	0.78	0.40	0.63	0.35	0.90

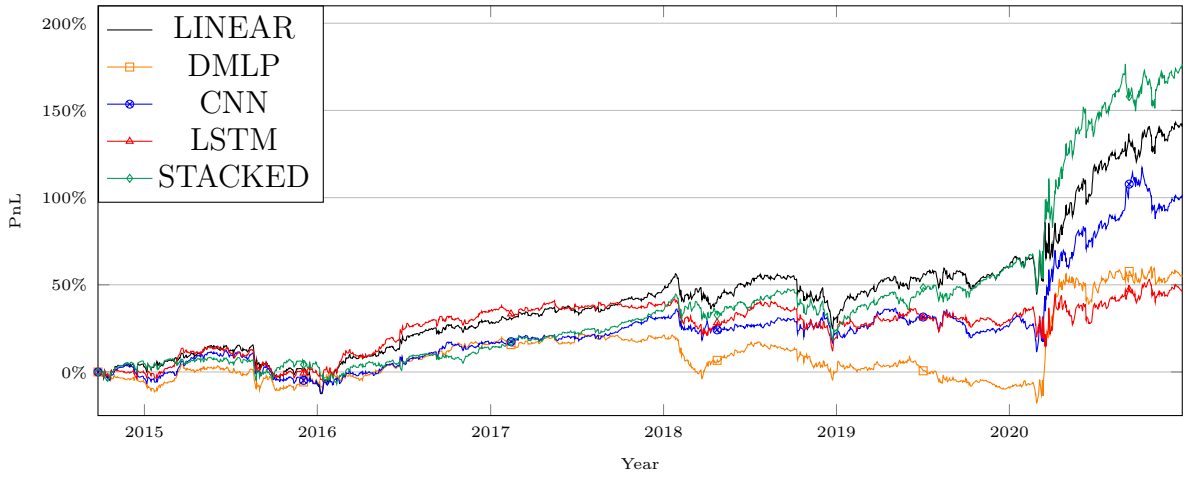


Figure 6: The PnL plots of all the models for the daily interval. The PnL is measured from the out-of-sample test dataset, ranging from October 2014 until December 2020. Here PnL is given in percentages.

For the daily interval, the results are not completely in line with the results found for the prediction performance of the models. Where the linear regression model outperforms the other models in terms of prediction performance, it is outperformed in terms

of financial performance by the stacked model. Also, the ranking of the DLMs differs between prediction and financial performance.

The second metric for financial performance, the annual Sharpe ratio, shows the same pattern as the profit of the models. Here only LSTM has an annual Sharpe ratio below that of the SP index. Whereas the other models improve upon the SP. Here the annual Sharpe ratio of the stacked model improves that of SP by 55%.

6.2.2 Weekly interval

As Figure 7 shows all models, except the LSTM model, were performing almost the same until the end of 2019. After this, only the CNN model correctly predicts the returns around the COVID-19 pandemic. This causes the CNN model to achieve the highest profit (153.8%), together with an annual Sharpe ratio exceeding that of SP by 45%. The outperforming CNN model in terms of financial performance can not be related to the prediction performance, where it is almost equal to most of the other models.

Table 15: The total profit and annual Sharpe ratio of all models together with the long position of the S&P500 index for the out-of-sample test dataset with weekly data.

Model	S&P500 index	LINEAR	DMLP	CNN	LSTM	STACKED
Profit	91.1%	93.0%	93.0%	153.8%	33.5%	88.7%
Annual Sharpe ratio	0.58	0.60	0.60	0.84	0.27	0.58

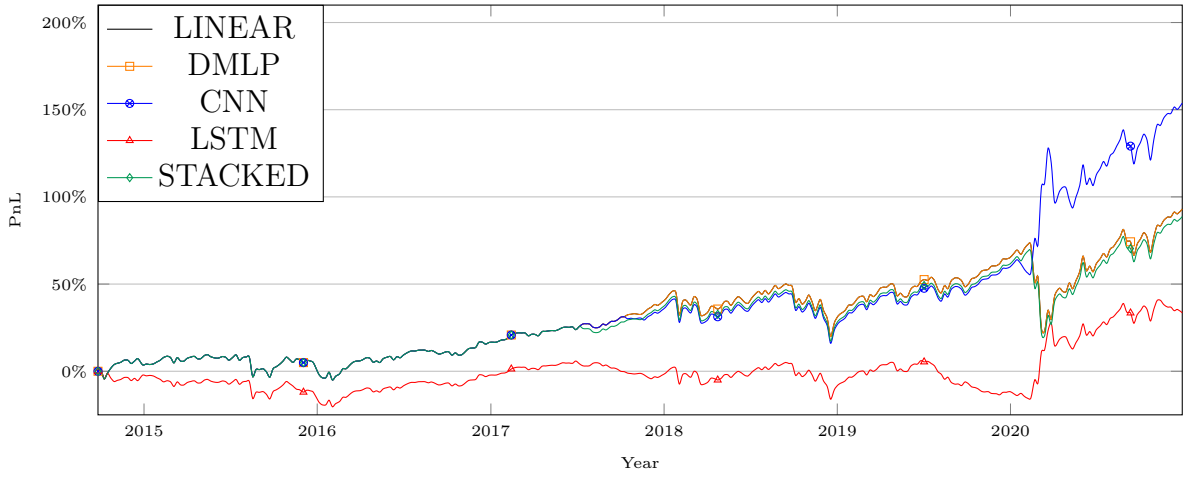


Figure 7: The PnL plots of all the models for the weekly interval. The PnL is measured from the out-of-sample test dataset, ranging from October 2014 until December 2020. The PnL plots of the linear and DMLP models overlap. Here PnL is given in percentages.

The weak performing model, the LSTM model, moves around 0% until the end of 2019, whereafter it predicts the returns around the COVID-19 pandemic correctly following the

CNN model. However, the LSTM model ends with the lowest profit of 33.5%, with the only annual Sharpe ratio under that of SP. The financial performance of the LSTM is in line with the prediction performance, where the LSTM model also showed the lowest score. Furthermore, both the linear and DMLP models perform the same as a buy and hold strategy. This is because these models only buy and do not sell. Due to the setup of this trading strategy, this results in a buy and hold strategy. Therefore, the profits and annual Sharpe ratios of the linear and DMLP models are almost equal to that of the long position of SP. They differ because the profit and annual Sharpe ratio of the long position of SP are calculated using daily return data instead of weekly.

6.2.3 Monthly interval

Figure 8 presents the PnL plots of the models of the monthly interval. The LSTM model outperforms the other models for the monthly interval at the end of the period. With a profit of 189.4%, the LSTM model achieves the highest overall profit among all intervals. The LSTM also achieves the highest overall annual Sharpe ratio of 0.95, which improves the annual Sharpe ratio of SP by 67%.

Table 16: The total profit and annual Sharpe ratio of all models together with the long position of the S&P500 index for the out-of-sample test dataset with monthly data.

Model	S&P500 index	LINEAR	DMLP	CNN	LSTM	STACKED
Profit	91.1%	93.0%	149.8%	28.8%	189.4%	93.0%
Annual Sharpe ratio	0.58	0.54	0.84	0.25	0.97	0.54

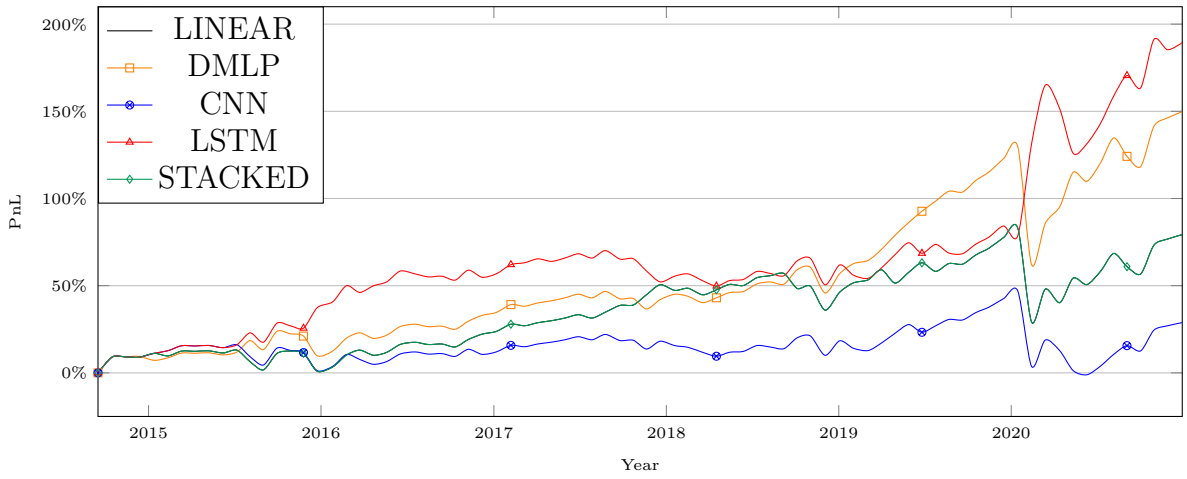


Figure 8: The PnL plots of all the models for the monthly interval. The PnL is measured from the out-of-sample test dataset, ranging from October 2014 until December 2020. The PnL plots of the linear and stacked models overlap. Here PnL is given in percentages.

Although the DMLP model got the highest accuracy for the monthly interval, it does not outperform the LSTM model in terms of financial performance at the end of the period. On the other hand, the CNN model, which has the lowest accuracy for the monthly interval also has the lowest profit for the monthly interval (28.8%), and an annual Sharpe ratio substantially lower than that of SP. Both the linear and stacked models follow a buy and hold strategy resulting in a profit of 93.0% and an annual Sharpe ratio of 0.53.

6.2.4 Comparison of financial performance

In terms of financial performance, no model outperforms the other models consistently for all intervals. The linear and stacked models are the most stable across the intervals, where both models do not have annual Sharpe ratios, which are substantially lower than that of SP. For the DLMs, the financial performance changes more between the intervals. Here the DLMs achieve annual Sharpe ratios, improving SP by more than 50%. However, the DLMs also achieve annual Sharpe ratios 50% below SP. As a result, the highest overall annual Sharpe ratio and profit are achieved by LSTM, and the lowest overall annual Sharpe ratio and profit are achieved by the CNN, both for the monthly interval.

When investigating the overall profit and annual Sharpe ratios achieved by the models, there is no better performing interval. For example, both the lowest and highest profits are achieved monthly. This can be due to returns increasing when the interval increases. This leads to higher absolute returns per trade, making a few trades able to positively and negatively impact the overall profit. Furthermore, the order of the models change with the intervals. For instance, the LSTM achieves the lowest profit for the daily and weekly intervals. However, it is the best performing model in terms of profit for the monthly interval.

The results also show that the volatility of the SP impacts the profit of the models. Here non-volatile periods follow small changes in financial performance due to small absolute returns. However, a volatile period such as the COVID-19 pandemic, shown in Figure 9a, has a large impact on the models in terms of their financial performance. This is due to the large absolute returns present in these periods. The PnL plots show that the COVID-19 pandemic indeed influenced the financial performance deeply. Therefore, having high accuracy alone does not ensure high profits because the high accuracy can

be based on correctly predicted returns, which were small in absolute value.

The differences in the size of the intervals result in different impacts of certain events. For example, where the COVID-19 pandemic is profitable for all models for the daily interval, increasing the interval decreases the number of profitable models in that period. This is due to the number of returns in that period. Here the models can adapt more easily in the daily interval than for the weekly and monthly intervals.

7 Conclusion

This study investigated if deep learning models can outperform the linear regression model in the sign prediction of returns. The research is done on the returns of the S&P500 index from the beginning of 2000 until the end of 2020. Here we predict returns on three-time intervals; daily, weekly and monthly. Three models were deployed for the deep learning models: the convolutional neural network, deep multilayer perceptron, and long short-term memory network. These are the most popular deep learning models for financial time series forecasting following Sezer et al. (2020). Furthermore, a stacked model is constructed to extend upon the deep learning models, which combines the models on their predictions using a meta-learner. For implementing the deep learning models, automated machine learning is applied for feature selection, optimisation of the hyperparameters and the combination of the models. The feature selection is done using a feature set containing price and return data of the S&P500 index, technical indicators, macroeconomic variables and underlying stocks of the S&P500 index. Finally, the performance is assessed by the prediction and financial performance. Here the metrics used are the confusion matrix, accuracy, F1-score, PnL, and the annual Sharpe ratio.

We found that the linear regression model performed best in terms of prediction power and performed well in terms of financial performance for the daily interval. The weekly and monthly intervals showed that there were deep learning models which had higher performance than the linear regression model in terms of prediction and financial performance. Despite that, there were also deep learning models which performed worse than the linear regression model. When using the McNemar test to compare the models, no model predicted significantly better than the other evaluated models. However, the out-of-sample predictions of the long short-term memory network for the weekly interval were significantly worse than those from the other models. The overall highest performing

model in terms of prediction performance was the deep multilayer perceptron for the monthly interval, with an accuracy of 70.9%.

In terms of financial performance, there were large differences in terms of profit and annual Sharpe ratios. All profits achieved by the models are positive. Despite that, there were annual Sharpe ratios obtained below the annual Sharpe ratio of the S&P500 index itself. The linear regression model was more stable across the intervals, where the deep learning models achieved annual Sharpe ratios, which were both 50% lower and higher than the S&P500 index itself. The overall highest performing model in terms of financial performance was the long short-term memory network for the monthly interval, with a profit of 189.4% and an annual Sharpe ratio of 0.95.

Answering the underlying research question: *'Does sign prediction of returns improve when increasing the time interval?'* We found that the overall prediction performance increased when increasing the interval. However, there was one exception for the convolutional neural network. For the financial performance, there is no overall increase or decrease when increasing the time interval. Answering the research question: *'Do deep learning models improve sign prediction of returns over the linear regression model?'* Deep learning models cannot consistently improve sign prediction of returns compared to the linear regression model in terms of prediction and financial performance. Thus, despite the theory suggesting that the deep learning models can find complex relationships, this did not lead to deep learning models significantly outperforming the linear regression model in terms of prediction performance. The performance of the deep learning models could be suboptimal due to four aspects.

First, It could be that the optimal hyperparameter set is not found. This can be due to a too large hyperparameter space. When the hyperparameter space gets larger, the computational time needed for the hyperparameter optimisation to find an optimal set increases substantially. Therefore, we set a maximum iteration for the hyperparameter optimisation. However, it can be that the optimal hyperparameter set was not yet found. This influences the entire performance of the deep learning model when training the model.

Second, the training of deep learning models could have led to suboptimal performance due to two aspects; the complexity of the model due to the hyperparameters and the changing performance every time the model is trained. The complexity of a deep learning

model, due to many layers or neurons in each layer, substantially increases the number of weights that must be tuned. Therefore, the model is less likely to achieve the optimal set of weights due to the maximum iteration. Regarding the changing performance between training, a deep learning model ends up with a different weighting scheme every time the model is trained due to the self-learning property during training. Therefore, it can be that the model's maximum performance was not yet achieved.

Third, the performance could be suboptimal due to the lack of useful features available. Although automated machine learning helps the feature selection, a feature set with useful features is necessary. When the feature set does not include useful features, it is not possible to select features and find relationships between the selected features and the response variable.

Last, the return data itself is vital in the performance of the deep learning models or any model for that matter. Unfortunately, financial data tends to be very noisy, making it harder to make prediction models perform well. Therefore, despite that the deep learning models can find more relationships than the linear regression model, this does not mean that the out-of-sample performance of a deep learning model has to be superior to the out-of-sample performance of the linear regression model.

A general note is that the study is limited on a computational aspect. The deep learning models require a certain level of computational power to ensure that the computation time does not explode. Due to limited computational power, we restricted our hyperparameter optimisation and model training to maintain reasonable computation times. This could have led to suboptimal results, explaining that the deep learning models do not outperform the linear regression model significantly. Furthermore, due to computational power, the input data used for the models were restricted to ten features. Including more features can benefit the performance since more relationships can be found. However, the added features can also affect the performance negatively, due to the addition of noise.

For further research upon this study, one can include more models for comparison. For example, forecasting models such as autoregressive (moving average) models (AR, ARMA) or the shallow versions of the used deep learning models as well as other machine learning models. Furthermore, the comparison of the models can be made more extensive by investigating their performance on other time series, for example, stocks, foreign exchanges, and commodities.

References

Margherita Balconi, Stefano Brusoni, and Luigi Orsenigo. In defence of the linear model: An essay. *Research policy*, 39(1):1–13, 2010.

Aaron Elliot and Cheng Hua Hsu. Time series prediction: Predicting stock price. *arXiv preprint arXiv:1710.05751*, 2017.

Ramazan Gencay. Non-linear prediction of security returns with moving average rules. *Journal of Forecasting*, 15(3):165–174, 1996.

Ramazan Gencay. The predictability of security returns with simple technical trading rules. *Journal of Empirical Finance*, 5(4):347–359, 1998.

Huisu Jang and Jaewook Lee. An empirical study on modeling and prediction of bitcoin prices with bayesian neural networks based on blockchain information. *Ieee Access*, 6: 5427–5437, 2017.

Richard Roll, T At, and Bob Roll. *Behavior of Interest Rates*, volume 465. Basic Books, 1970.

Eugene F Fama and James D MacBeth. Risk, return, and equilibrium: Empirical tests. *Journal of political economy*, 81(3):607–636, 1973.

Eugene F Fama. The behavior of stock-market prices. *The journal of Business*, 38(1): 34–105, 1965.

Paul A Samuelson. Proof that properly discounted present values of assets vibrate randomly. *The Bell Journal of Economics and Management Science*, pages 369–374, 1973.

Andrew W Lo and A Craig MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The review of financial studies*, 1(1):41–66, 1988.

Eugene F Fama. Random walks in stock market prices. *Financial analysts journal*, 51 (1):75–80, 1995.

Melvin J Hinich and Douglas M Patterson. Evidence of nonlinearity in daily stock returns. *Journal of Business & Economic Statistics*, 3(1):69–77, 1985.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Omer Berat Sezer, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu. Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181, 2020.

Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural computation*, 29(9):2352–2449, 2017.

Omer Berat Sezer and Ahmet Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 70:525–538, 2018.

Jou-Fan Chen, Wei-Lun Chen, Chun-Ping Huang, Szu-Hao Huang, and An-Pin Chen. Financial time-series data analysis using deep convolutional neural networks. In *2016 7th International conference on cloud computing and big data (CCBD)*, pages 87–92. IEEE, 2016.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

François Chollet et al. Keras. `https://keras.io`, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Radwa Elshawi, Mohamed Maher, and Sherif Sakr. Automated machine learning: State-of-the-art and open challenges. *arXiv preprint arXiv:1906.02287*, 2019.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

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

Mervyn Stone. Asymptotics for and against cross-validation. *Biometrika*, pages 29–35, 1977.

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

Ian H Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with Java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

S&P Dow Jones indices. S&P500 constituents, 11 2021. URL `https://www.spglobal.com/spdji/en/indices/equity/sp-500/#data`.

Wharton Research Data Service. CRSP Stocks, 06 2021. URL `https://wrds-www.wharton.upenn.edu/pages/get-data/center-research-security-prices-crsp/`.

Bing Yang, Zi-Jia Gong, and Wenqi Yang. Stock market index prediction using deep neural network ensemble. In *2017 36th chinese control conference (ccc)*, pages 3882–3887. IEEE, 2017.

Zhifeng Dai, Huan Zhu, and Jie Kang. New technical indicators and stock returns predictability. *International Review of Economics & Finance*, 71:127–142, 2021.

Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the istanbul stock exchange. *Expert systems with Applications*, 38(5):5311–5319, 2011.

Federal Reserve Bank of St. Louis. FRED economic data, 2021. URL `https://fred.stlouisfed.org/`.

David E Rapach, Mark E Wohar, and Jesper Rangvid. Macro variables and international stock return predictability. *International journal of forecasting*, 21(1):137–166, 2005.

Seyed Taghi Akhavan Niaki and Saeid Hoseinzade. Forecasting s&p 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International*, 9(1):1–9, 2013.

Xiao Zhong and David Enke. Forecasting daily stock market return using dimensionality reduction. *Expert Systems with Applications*, 67:126–139, 2017.

Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

Mark T Leung, Hazem Daouk, and An-Sing Chen. Forecasting stock indices: a comparison of classification and level estimation models. *International Journal of forecasting*, 16(2):173–190, 2000.

Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1-2):307–319, 2003.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

Jeff Fleming, Chris Kirby, and Barbara Ostdiek. The economic value of volatility timing using "realized" volatility. *Journal of Financial Economics*, 67(3):473–509, 2003.

Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

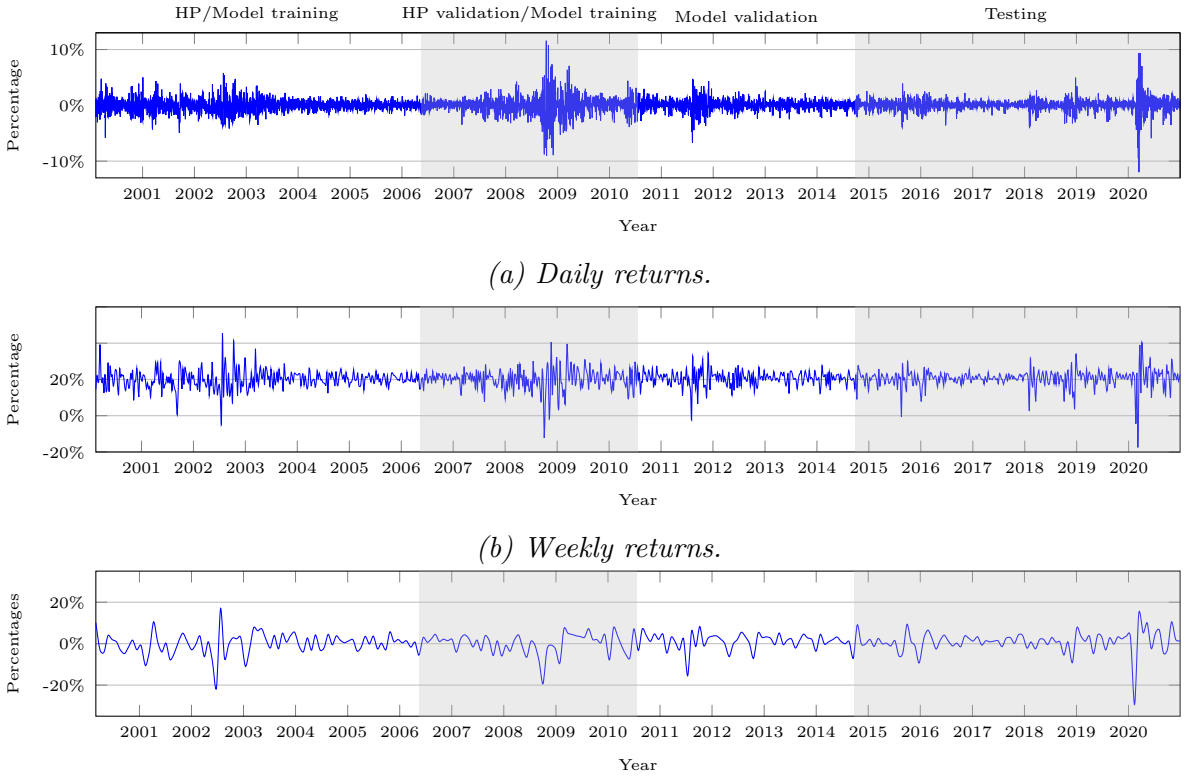
Matthew Dixon, Diego Klabjan, and Jin Hoon Bang. Classification-based financial markets prediction using deep neural networks. *Algorithmic Finance*, 6(3-4):67–77, 2017.

Oliver Ledoit and Michael Wolf. Robust performance hypothesis testing with the sharpe ratio. *Journal of Empirical Finance*, 15(5):850–859, 2008.

William F Sharpe. The sharpe ratio. *Journal of portfolio management*, 21(1):49–58, 1994.

8 Appendix

A.1 - Data



(a) Daily returns.

(b) Weekly returns.

(c) Monthly returns.

Figure 9: The return of the S&P500 index in U.S. dollars for the daily, weekly and monthly intervals. The data ranges from 8th of February 2000 to 31st of December 2020. Together with the time periods of training, validation, and testing data. HP stands for hyperparameter.

Table 17: The symbol and company name of the stocks present in the S&P500 index from at least the 1st of January 2000 until the 31th of December 2020. The list contains 204 underlying stocks.

Symbol	Company	Symbol	Company
ED	Consolidated Edison	AXP	American Express
GLW	Corning Inc.	BAC	Bank of America Corp
DHR	Danaher Corp.	CI	Cigna
DRI	Darden Restaurants	DUK	Duke Energy
D	Dominion Energy	LNC	Lincoln National
ETN	Eaton Corporation	TAP	Molson Coors Beverage Company
ES	Eversource Energy	NEE	NextEra Energy
FITB	Fifth Third Bancorp	DIS	The Walt Disney Company
FE	FirstEnergy Corp	WFC	Wells Fargo

Table 17: The symbol and company name of the stocks present in the S&P500 index from at least the 1st of January 2000 until the 31th of December 2020. The list contains 204 underlying stocks.

Symbol	Company	Symbol	Company
BEN	Franklin Resources	MMM	3M Company
FCX	Freeport-McMoRan Inc.	INTC	Intel Corp.
GE	General Electric	TGT	Target Corp.
HUM	Humana Inc.	TXT	Textron Inc.
HBAN	Huntington Bancshares	VFC	VF Corporation
K	Kellogg Co.	WBA	Walgreens Boots Alliance
KLAC	KLA Corporation	AIG	American International Group
LEG	Leggett & Platt	FDX	FedEx Corporation
L	Loews Corp.	PCAR	Paccar
MAR	Marriott International	ADP	Automatic Data Processing
MKC	McCormick & Co.	GWW	Grainger (W.W.) Inc.
MCK	McKesson Corp.	MAS	Masco Corp.
MCO	Moody's Corp	ADM	Archer-Daniels-Midland Co
MSI	Motorola Solutions Inc.	WMT	Walmart
NTRS	Northern Trust Corp.	SNA	Snap-on
OMC	Omnicom Group	SWK	Stanley Black & Decker
PAYX	Paychex Inc.	AAPL	Apple Inc.
PNW	Pinnacle West Capital	OXY	Occidental Petroleum
PPL	PPL Corp.	LB	L Brands Inc.
QCOM	Qualcomm	T	AT&T Inc.
ROK	Rockwell Automation Inc.	VZ	Verizon Communications
SPGI	S&P Global Inc.	LOW	Lowe's Cos.
SRE	Sempra Energy	PHM	PulteGroup
SBUX	Starbucks Corp.	HES	Hess Corporation
STT	State Street Corp.	LMT	Lockheed Martin Corp.
TROW	T. Rowe Price Group	HAS	Hasbro Inc.
TSN	Tyson Foods	BLL	Ball Corp
USB	U.S. Bancorp	APD	Air Products & Chemicals
VLO	Valero Energy	NUE	Nucor Corp.
VIAC	ViacomCBS	PKI	PerkinElmer
VNO	Vornado Realty Trust	NOC	Northrop Grumman
WM	Waste Management Inc.	CNP	CenterPoint Energy
WAT	Waters Corporation	TJX	TJX Companies Inc.
WRK	WestRock	DOV	Dover Corporation

Table 17: The symbol and company name of the stocks present in the S&P500 index from at least the 1st of January 2000 until the 31th of December 2020. The list contains 204 underlying stocks.

Symbol	Company	Symbol	Company
WY	Weyerhaeuser	PH	Parker-Hannifin
WHR	Whirlpool Corp.	ITW	Illinois Tool Works
MO	Altria Group Inc	GPS	Gap Inc.
AEP	American Electric Power	MDT	Medtronic plc
BA	Boeing Company	SYY	Sysco Corp.
BMY	Bristol-Myers Squibb	MMC	Marsh & McLennan
CPB	Campbell Soup	AVY	Avery Dennison Corp
CAT	Caterpillar Inc.	HD	Home Depot
CVX	Chevron Corp.	PNC	PNC Financial Services
KO	Coca-Cola Company	C	Citigroup Inc.
CL	Colgate-Palmolive	NKE	Nike, Inc.
COP	ConocoPhillips	ECL	Ecolab Inc.
CVS	CVS Health	GL	Globe Life Inc.
DE	Deere & Co.	NWL	Newell Brands
DTE	DTE Energy Co.	ORCL	Oracle Corp.
EIX	Edison Int'l	ADSK	Autodesk Inc.
ETR	Entergy Corp.	MRO	Marathon Oil Corp.
EXC	Exelon Corp.	AEE	Ameren Corp
XOM	Exxon Mobil Corp.	AMGN	Amgen Inc.
F	Ford Motor Company	LIN	Linde plc
GD	General Dynamics	IPG	Interpublic Group
HAL	Halliburton Co.	COST	Costco Wholesale Corp.
HIG	Hartford Financial Svc.Gp.	CSCO	Cisco Systems
HSY	The Hershey Company	EMN	Eastman Chemical
IBM	International Business Machines	KEY	KeyCorp
IP	International Paper	UNM	Unum Group
KMB	Kimberly-Clark	MSFT	Microsoft Corp.
KR	Kroger Co.	LUV	Southwest Airlines
MRK	Merck & Co.	UNH	UnitedHealth Group Inc.
NSC	Norfolk Southern Corp.	MU	Micron Technology
PEP	PepsiCo Inc.	BSX	Boston Scientific
PFE	Pfizer Inc.	AMAT	Applied Materials Inc.
PPG	PPG Industries	BK	The Bank of New York Mellon
PG	Procter & Gamble	ALL	Allstate Corp

Table 17: The symbol and company name of the stocks present in the S&P500 index from at least the 1st of January 2000 until the 31th of December 2020. The list contains 204 underlying stocks.

Symbol	**Company**	**Symbol**	**Company**
PEG	Public Service Enterprise Group	CMA	Comerica Inc.
SEE	Sealed Air	AON	Aon plc
SO	Southern Company	AZO	AutoZone Inc
UNP	Union Pacific Corp	ADBE	Adobe Inc.
XEL	Xcel Energy Inc	CAH	Cardinal Health Inc.
ABT	Abbott Laboratories	SCHW	Charles Schwab Corporation
HON	Honeywell Int'l Inc.	EFX	Equifax Inc.
SHW	Sherwin-Williams	APA	APA Corporation
CMI	Cummins Inc.	PGR	Progressive Corp.
EMR	Emerson Electric Company	YUM	Yum! Brands Inc
SLB	Schlumberger Ltd.	TFC	Truist Financial
CSX	CSX Corp.	CINF	Cincinnati Financial
CLX	The Clorox Company	COF	Capital One Financial
GIS	General Mills	RF	Regions Financial Corp.
NEM	Newmont Corporation	AES	AES Corp
MCD	McDonald's Corp.	CCL	Carnival Corp.
LLY	Lilly (Eli) & Co.	LUMN	Lumen Technologies
BAX	Baxter International Inc.	CMS	CMS Energy
BDX	Becton Dickinson	AFL	Aflac
JNJ	Johnson & Johnson	NTAP	NetApp
GPC	Genuine Parts	BBY	Best Buy Co. Inc.
HPQ	HP Inc.	VMC	Vulcan Materials
WMB	Williams Companies	ADI	Analog Devices, Inc.
JPM	JPMorgan Chase & Co.	XLNX	Xilinx
IFF	International Flavors & Fragrances	CTXS	Citrix Systems

A.2 - Results

Table 18: The feature sets for the daily, weekly and monthly intervals. Selected from the entire feature set by SelectKbest with k=10. The returns and prices of underlying stocks are one-period lagged returns and prices.

DAILY	WEEKLY	MONTHLY
Lagged sign of return	Consolidated Edison & Co. - Price	Archer-Daniels-Midland Co. - Return
Lagged S&P500 return	Kellogg Co. - Price	CVS Health - Return
Lagged value weighted return	Sempra Energy - Price	Cardinal Health Inc. - Return
SMA_1	Edison Int'l - Price	Norhtrop Grumman - Return
M_1	Southern Co. - Price	Nucor Corp. - Return
Exxon Mobil Corp. - Return	Cummins Inc. - Price	Newell Brands - Return
Emerson Electric Company - Return	Genuine Parts - Price	Lowe's Cos. - Return
Merck & Co - Return	Autozone Inc. - Price	Ecolab Inc. - Return
Cincinnati Financial - Return	YTCMR10	YUM! Brands Inc.- Return
American Express - Return	YTCMR20	YTCMR10

Table 19: The best performing hyperparameter sets of DMLP, LSTM and CNN for the daily, weekly and monthly intervals.

	Hyperparmeters of DMLP									
	Dense layers	Units dense layer 1	Activation dense layer 1	Units dense layer 2	Activation dense layer 2	Units dense layer 3	Activation dense layer 3	Learning rate	Batch size	Number of epochs
DAILY	3	171	softmax	181	tanh	1	exponential	0.0001	6	27
WEEKLY	3	31	relu	161	elu	1	sigmoid	0.01	12	4
MONTHLY	3	181	selu	71	relu	1	exponential	0.0001	15	110

	Hyperparmeters of LSTM									
	LSTM layers	Units LSTM layer 1	Activation LSTM layer 1	Recurrent activation LSTM layer 1	Units LSTM layer 2	Activation LSTM layer 2	Recurrent activation LSTM layer 2	Dense layers	Units dense layer 1	Learning rate
DAILY	2	82	tanh	sigmoid	92	tanh	sigmoid	1	1	0.001
WEEKLY	2	82	tanh	sigmoid	92	tanh	sigmoid	1	1	0.001
MONTHLY	2	2	tanh	sigmoid	92	tanh	sigmoid	1	1	0.01

	Batch size	Number of epochs
DAILY	5	151
WEEKLY	20	68
MONTHLY	11	101

	Hyperparmeters of CNN									
	Convolutional (conv) layers	Filters conv. layer 1	Kernel size conv. layer 1	Activation conv. layer 1	Filters conv. layer 2	Kernel size conv. layer 2	Activation conv. layer 2	Pooling layer	Fully Connected/ dense layers	Units dense layer 1
DAILY	2	112	1	tanh	62	1	gelu	max-pooling	3	91
WEEKLY	2	191	1	selu	162	1	relu	max-pooling	3	131
MONTHLY	2	82	1	gelu	182	1	tanh	max-pooling	3	101

	Activation dense layer 1	Units dense layer 2	Activation dense layer 2	Units dense layer 3	Activation dense layer 3	Learning rate	Batch size	Number of epochs
DAILY	elu	161	exponential	1	sigmoid	0.001	12	106
WEEKLY	elu	111	selu	1	selu	0.0001	3	200
MONTHLY	tanh	171	tanh	1	sigmoid	0.0001	10	143

Table 20: The McNemar test results for the out-of-sample test dataset for the daily, weekly and monthly intervals. Significant scores are presented in bold (at a 5% significant level). A score of zero means that either two models predict exactly the same, or their is a difference of 1 between cell 2 and 3, see Section 5.1.

(a) Daily interval.

	Linear	**DMLP**	**CNN**	**LSTM**	**Stacked**
Linear		2.402	0.496	1.341	0.032
DMLP	2.402		1.383	0.863	1.322
CNN	0.496	1.383		0.275	0.122
LSTM	1.341	0.863	0.275		0.510
Stacked	0.032	1.322	0.122	0.510	

(b) Weekly interval.

	Linear	**DMLP**	**CNN**	**LSTM**	**Stacked**
Linear		0	0	**11.066**	0.5
DMLP	0		0	**11.066**	0.5
CNN	0	0		**12.852**	0
LSTM	**11.066**	**11.066**	**12.852**		**9.851**
Stacked	0.5	0.5	0	**9.851**	

(c) Monthly interval.

	Linear	**DMLP**	**CNN**	**LSTM**	**Stacked**
Linear		0.267	1.895	0.563	0
DMLP	0.267		5.786	2.042	0.267
CNN	1.895	5.786		0.071	1.895
LSTM	0.563	2.042	0.071		0.563
Stacked	0	0.267	1.895	0.563	