DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

Deep learning-based forecasting of financial assets

PHILIP WIDEGREN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Deep learning-based forecasting of financial assets

PHILIP WIDEGREN

Degree Projects in Financial Mathematics (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics
KTH Royal Institute of Technology year 2017
Supervisors at Lynx Asset Management: Tobias Rydén and Martin Rehn
Supervisor at KTH: Jimmy Olsson
Examiner at KTH: Jimmy Olsson

TRITA-MAT-E 2017:27
ISRN-KTH/MAT/E--17/27--SE

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Prediktering av finansiella tillgångar med hjälp av djupa neuronnät

Sammanfattning

Djupa neuronnät har under det senaste årtiondet blivit ett väldigt användarbart verktyg för att lösa komplexa problem, tack vare förbättringar i träningsalgoritmer. Två områden där djupinlärning visat sig väldigt användbart är inom taligenkänning och maskinöversättning. Det finns relativt få artiklar där djupinlärning används inom finans men i de få som existerar finns det tydliga tecken på att djupinlärning skulle kunna appliceras framgångsrikt på finansiella problem.

Denna uppsats studerar prediktering av finansiella prisrörelser med framåtkopplade nätverk och rekurrenta nätverk. För de framåtkopplade nätverken kommer vi använda oss av djupa nätverk med färre neuroner per lager och mindre djupa nätverk med fler neuroner per lager. Förutom en jämförelse mellan framåtkopplade nätverk och rekurrenta nätverk kommer även en jämförelse mellan de djupa och mindre djupa framåtkopplade nätverken att göras. De rekurrenta nätverket består av ett rekurrent lager som sedan projicerar på ett framåtkopplande lager följt av ett outputlager. Nätverken är tränade med två olika uppsättningar av insignaler, ett mindre komplext och ett mer komplext.

Resultaten för jämförelsen mellan de olika framåtkopplade nätverken indikerar att det inte med säkerhet går att säga om man vill använda sig av ett djupare nätverk eller inte, då det beror på många olika faktorer som tex. variabeluppsättning. Resultaten för jämförelsen mellan de rekurrent nätverken och framåtkopplade nätverken indikerar att rekurrenta nätverk nödvändigtvis inte presterar bättre än framåtkopplade nätverk trots att finansiell data vanligtvis är tidsberoende. Det finns signifikanta resultat där den mer komplexa variabeluppsättningen presterar bättre än den mindre komplexa. Den högsta träffsäkerheten för att prediktera rätt tecken på nästkommande prisrörelse är 52.82% vilket är signifikant bättre än ett enkelt benchmark.

Deep learning-based forecasting of financial assets

Abstract

Deep learning and neural networks has recently become a powerful tool to solve complex problem due to improvements in training algorithms. Examples of successful application can be found in speech recognition and machine translation. There exist relative few finance articles were deep learning have been applied, but existing articles indicate that deep learning can be successfully applied to problems in finance.

This thesis studies forecasting of financial price movements using two types of neural networks, namely; feedforward and recurrent networks. For the feedforward neural networks we considered non-deep networks with more neurons and deep networks with fewer neurons. In addition to the comparison between feedforward and recurrent networks, a comparison between deep and non-deep networks will be made. The recurrent architecture consists of a recurrent layer mapping into a feedforward layer followed by an output layer. The networks are trained with two different feature setups, one less complex and one more complex.

The findings for non-deep vs. deep feedforward neural networks imply that there does not exist any general pattern whether deep or non-deep networks are preferable. The findings for recurrent neural networks vs. feedforward neural networks imply that recurrent neural networks do not necessarily outperform feedforward neural networks even though financial data in general are time-dependent. In some cases, adding batch normalization can improve the accuracy for the feedforward neural networks. This can be preferable instead of using more complex models, such as a recurrent neural networks. Moreover, there are significant differences in accuracies between using the two different feature setups. The highest accuracy for all networks are 52.82%, which is significantly better than the simple benchmark.

Acknowledgements

First of all I would like to thank my supervisors at Lynx Asset Management, Tobias Rydén and Martin Rehn, for their feedback, comments, and interest in the project. I would also like to express my gratitude to Lynx Asset Management for giving me the opportunity to write my thesis for them as well as providing me with the necessary data.

Moreover, I would like to show appreciation to my supervisor at KTH Royal Institute of Technology, Jimmy Olsson, for his invaluable comments and guidance in this thesis.

I would also like to thank all friends I have come to know during my five years of studies and for all interesting discussions we have had both in and outside the university.

Lastly, I would like to express my greatest gratitude to my girlfriend, Rebecka Tallberg, for all support she has given me during my studies, especially for her ability to light up the most challenging days.

Stockholm, May 2017

Philip Widegren

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Prediction of returns on financial assets is one of the most frequently discussed topics in finance. In practice it is almost impossible to predict the correct return and most research considers the case of predicting whether the market will go up or down during the following period. Moreover, prediction of the market is of great interest in order both to speculate about potential returns and to understand which factors that drive the market. According to the efficient-market hypothesis, prices on financial assets reflect all available information and therefore implies that it is impossible to consistently outperform the market [1].

To try to model the future returns one has to have an idea about which data and economic variables that influence the future prices. The research about which data and economic variables that influence the future returns is a research area in itself and there does not exist any theory specifying relevant parameters [2] [3]. Moreover, there exists evidence that future returns can be represented by historical returns [4].

Most financial data can be considered as time series, i.e. it has a time dependency. Therefore, methods where the time series structure is preserved is to prefer. The relationship between the data is very complex. Thus, machine learning algorithms have been of interest since the data have complex relationships and the amount of data has increased.

Assets across different markets can be considered to be correlated [4]. Therefore, specific markets tend to follow each other and a multivariate model can be preferable to capture those kind of scenarios.

A natural question as the interest in deep learning has grown in the last decade, due to some improvements in training, is if deep learning can be successfully applied to the prediction of returns on financial assets.

1.1 Research questions

In this section the related research questions will be introduced. The results will be presented in Chapter 5 and a deeper discussion will be presented in Chapter 6.

1. Is it possible that a deep feedforward neural network with fewer neurons in each hidden layer can outperform a non-deep neural network with many neurons in each hidden layer?

2. Is it possible that a recurrent neural network can outperform a feedforward neural network with the same feature setup?

1.2 Related work

There exists an extensive literature regarding prediction of price movements using machine learning. Methods like support vector machines [5] [6], hidden Markov models [7], and neural networks [8] [9] have been used.

An interesting survey that discusses different works in deep learning is “Deep Learning in Finance” by J. B. Heaton, N. G. Polson and J. H. Witte [10]. The article explores how deep learning has been used for problems in financial prediction and classification and suggests different ways to improve the methods. Moreover, the authors give an introduction to Long Short-Term Memory, recurrent neural networks, and suggest that the model can be used as a volatility model. The authors conclude that deep learning has the ability to detect patterns in the financial data that is invisible to the existing financial economic theory.

Another interesting article is “Using machine learning for medium frequency derivative portfolio trading” by A. Sharang and C. Rao. They use different machine learning methods, logistic regression, support vector machines, and feedforward neural networks, to construct a portfolio consisting of 5 year and 10 year US Treasury note futures. The mathematical formulation consists in predicting the weekly direction of the price movements using feature extraction from a deep belief network which is trained on technical indicators. Their result shows that it is possible to make profitable trades.

Moreover, Long Short-Term Memory neural network has been used in “Deep Learning Stock Volatility with Google Domestic Trends” by R. Xiong, E. P. Nichols and Y. Shen [11]. They model the volatility of S&P 500, using Google domestic trends as indicators of macroeconomic factors and the public atmosphere. Their Long Short-Term memory model outperformed methods like linear regression and GARCH benchmarks. As a consequence of their results, there exist strong evidence for increased prediction accuracy for predicting stock market behaviour using deep learning and neural network models.

Furthermore, Gilberto Batres-Estrada has applied deep learning in his master thesis “Deep Learning for Multivariate Financial Time Series” at KTH [8]. Estrada used deep learning to construct a portfolio of some given stocks. His deep learning model consists of a deep belief network mapping into a feedforward neural network. Moreover, he concluded that deep learning methods in finance are reliable and have good performance.

Also, there exist projects at the course Convolutional Neural Networks for Visual Recognition at Stanford considering different deep learning methods for stock trading. One interesting project is “Convolutional Networks for Stock Trading” by A. Siripurapu [12]. Siripurapu uses convolutional neural networks to predict the price movements for stock prices. In order to use the convolutional neural network he maps price information on 1 2D array that can be analysed with image recognition methods.

However, there does not exist any significant paper related to recurrent neural networks applied to financial forecasting of price movements. Recurrent neural networks have shown to be very powerful tools for speech recognition [13] and machine translation [14] where sequential data is considered. Therefore, it would be interesting to use recurrent neural networks for predicting price movements to investigate if the recurrent neural network is more powerful than the more classical feedforward neural network. In other words, can a recurrent neural network use the information in the sequence in a proper way to get use of the time-series that the usually feedforward neural networks not necessarily captures.

1.3 Scope and Limitations

This thesis focuses on comparing different neural network architectures and two different feature setups. The networks are trained with respect to the accuracy of predicting correct sign on the following price movement. The return of the model will not be considered as a qualitative variable to compare the models, just a measure if the model is tradable or not. The aim of the thesis is not to optimize the neural networks with respect to validation accuracy since training neural networks is very computationally expensive. Namely, the setup consists of many hyperparameters and therefore the networks will not be optimized with respect to validation accuracy since the problem is to computationally expensive and it is hard to construct a rigorous grid-search in larger dimensions.

1.4 Outline

This thesis has the following structure. Chapter 2 describes the necessary financial background needed to understand the difficulties behind forecasting the direction of price movements. Chapter 3 describes the mathematics behind neural networks, especially feedforward and recurrent neural networks, and different methods to train and regularize neural networks. In Chapter 4 the methodology is presented, namely different types of setups that will be considered in this thesis. In Chapter 5 the results will be introduced, with short comments. The results will be discussed in Chapter 6 followed up with conclusions and suggestions about future work in Chapter 7.

Chapter 2

Financial background

This chapter gives a short review of financial terminology and theory. It begins with describing what a short or long position means. Then, it follows up with a brief introduction to futures contracts and how they can be used, a description of the efficient market hypothesis and how it relates to this thesis, and lastly, a measure called Sharpe ratio is introduced in order to be able to compare different returns.

2.1 Long and short positions

A position is called *long* if we have a positive amount in the asset and a position is called *short* if we have a negative amount in the asset.

In theory it is often possible to be either long or short but in reality it can sometimes be hard to be short in certain kinds of assets, e.g. shares of small companies, since we then need to find an owner of the asset and borrow it from him to be able to go short in the asset. However, in this report we are only considering futures contracts and by construction one party has to be short and one has to be long, so this will not be a problem. Moreover, in this thesis we will always be either long or short in an asset.

2.2 Futures contracts

A *futures contract* is a standardized contract between between two parties to either buy or sell an asset at a certain date and predetermined price, thus the contract has one buying part and one selling part. The buying part is obligated to buy an asset at a certain time and the predetermined price while the selling part are obligated to sell this agreed asset at the given time and for the predetermined price. One important part of these contracts are that they are standardized thus they are traded on an exchange and the two parties must not necessarily know each other. Another important part of the contract is that it can have either physical or cash settlement, but the more technical parts will be described later. [15]

Futures contracts can be traded on various type of underlying assets, e.g. commodities such as gold, aluminum, wool, sugar and pork, and financial assets such as stock indices and currencies.

Moreover, there do also exist so called margins to decrease the counterparty and liquidity risks [15]. When entering a futures contract the buyer and the seller must deposit funds into a margin account. The specific amount that is needed when entering the contract is called initial margin, which is much smaller than the price of the total contract. Then after each trading day the margin account is adjusted to reflect the investor's gain or loss. If the deposits on the margin account decreases under a certain level then the holder gets a margin call. The margin call tells the investor that he needs to enter more deposits into the margin account or otherwise lose the contract. The futures contracts are suited to either hedge or speculate with since the required amount of funds needed to enter futures contracts is rather small compared to being long/short in the underlying asset. Also, it is common to standardize the returns of futures contracts by some type of value due to the leverage possibilities and since we actually do not buy/sell the asset when we entering the contract. In this report we are going to divide the return between two dates by a volatility measure based on historical data.

Furthermore, due to the fact that futures contracts are expiring on certain dates it is important to have some type of rolling scheme, i.e. when we should change our investment from a contract close to expiry to another contract not so close to expiry. The rolling scheme can be based on different interesting aspects but typically based on; the number of days to maturity or before first notice day and if the volume in the new contract is larger then the existing contract.

When trading futures contracts it is important to be aware of the fact that the returns of the futures contracts is not necessarily equal to the return of the underlying asset.

2.3 EMH

The *efficient-market hypothesis* (EMH) states that it is impossible to consistently beat the market since the prices reflect all relevant available information [16]. A consequence of EMH is that it is impossible to have a trading strategy that consistently outperforms the market over time [17]. There exist different types of efficiency, namely; weak, semi-strong-form, and strong efficiency.

2.3.1 Weak-form efficiency

The *weak-form efficiency* states that future prices cannot be predicted using historical prices [15]. An investment strategy or model cannot beat the market in the long term. Moreover, it will not help to use technical analysis to try to outperform the market but some fundamental analysis can be used. Furthermore, an implication of weak-form efficiency is that share prices do not have any serial dependencies or equivalently there does not exist any pattern in the asset prices. This implies that future stock prices are driven by information that cannot be extracted from the historical stock prices. Thus, since there does not exist any pattern in asset prices and future prices are not driven by historical prices we must have that asset prices follow a random walk.

This efficiency has been questioned and many studies have shown that the stock market is trending over some time periods, e.g. weeks or longer [18].

Moreover, there exist studies that indicate that future returns are reflected by the momentum, i.e. assets that have performed well over a given period in the past will continue to perform in the future well while assets that have performed poorly over a certain period in the past will continue to perform poorly in the future [19] [20].

2.3.2 Semi-strong-form

The *semi-strong-form efficiency* states that asset prices adjust to new publicly available information very fast such that it is not possible to get any returns by trading on that information. A consequence of semi-strong-form efficiency is that neither technical nor fundamental analysis can be used to beat the market in the long term run.

2.3.3 Strong-form efficiency

The *strong-form efficiency* states that asset prices reflect all available information, both public and private, and nobody can consistently beat the market in the long term by using an investment strategy or trading on information.

2.4 Sharpe ratio

Sharpe ratio is a measure that standardizes the returns of an asset compared to a benchmark in terms of their risks [15]. An advantage of this is that we can compare different returns even though they have different risks. Sharpe ratio is introduced as a measure to determine the trade-off between return and risks. The benchmark is often a more safe investment, for instance the risk-free rate or when investing in stocks on the U.S. equity market then the benchmark can be S&P500.

The formula for Sharpe ratio is given by

$$Sh = \frac{\mathbb{E}[R_a - R_b]}{\sqrt{\mathbb{V}ar[R_a - R_b]}},$$

where R_a is the return of the asset and R_b is the return of the benchmark asset.

In this thesis we will not have any benchmark asset, thus we will neglect R_b in the formula. The measure when neglecting the benchmark term is also called risk-adjusted return but in this thesis we will refer to it as Sharpe ratio since this is the most frequently used term for these kind of applications.

Chapter 3

Neural networks

This chapter is aimed to give the relevant information for the reader about neural networks. The neural networks considered in this report are feedforward neural networks and recurrent neural networks.

The chapter begins with describing the architecture of feedforward neural networks and recurrent neural networks. Then, it follows up with theory of the used loss function, different types of training methods and algorithms, and lastly, regularization methods.

3.1 Feedforward neural networks

Feedforward neural networks (FNN) are inspired by the neurons in the human brain, where the feedfoward neural networks connect a large number of computational units, called neurons, to solve a specific task [21]. The neurons are often divided into several layers and a neural network with many hidden layers is called a deep neural network. The architecture of a neural network is an input layer and an output layer where the connection between the input layer and output layer is given by an arbitrary amount of hidden layers. The output of each layer in the neural network is determined by an activation function f and the connections between each layer and the neurons in those layers are given by a weight matrix W.

For a neural network with two hidden layers, one output layer, input vector $\mathbf{x} \in \mathbb{R}^{m_0}$ and output vector $\mathbf{y} \in \mathbb{R}^{m_3}$ denoted $(2, \#neurons)$, the output of the network is given by

$$\begin{aligned}\mathbf{h}^{(1)} &= f^{(1)}(W^{(1)T}\mathbf{x} + \mathbf{b}^{(1)})\\ \mathbf{h}^{(2)} &= f^{(2)}(W^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})\\ \mathbf{y} &= f^{(3)}(W^{(3)T}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})\end{aligned}$$

where $\mathbf{h}^{(i)} \in \mathbb{R}^{m_i}$ denotes the output of the hidden layer i, $f^{(i)}$ denotes the activation function of layer i, $\mathbf{b}^{(i)} \in \mathbb{R}^{m_i}$ denotes the bias term in layer i and $W^{(i)} \in \mathbb{R}^{m_{i-1} \times m_i}$ is the the weight matrix in layer i. Notice that a neural network is a composition of functions.

Moreover, the universal approximation theorem [22] [23] states that a feedforward neural network with at least one hidden layer and linear output activation and any "squashing" activation, e.g. tanh or sigmoid, can approximate any function, provided that the network is given enough hidden units. Thus, neural networks are a very powerful tool in describing complex functions.

In figure 3.1 below we have a neural network with one input layer consisting of four inputs, three hidden layers where each layer consists of five hidden neurons and an output layer consisting of one neuron.

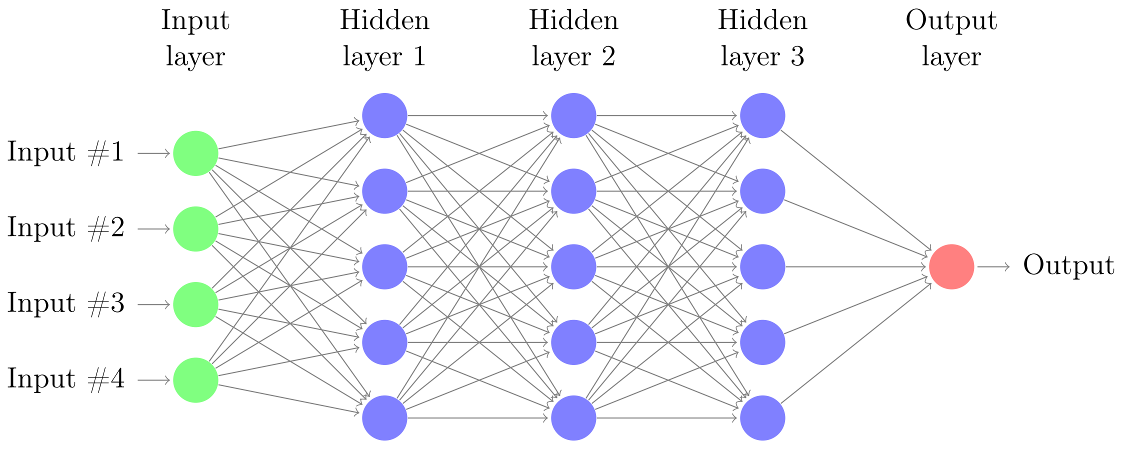

Figure 3.1: A fully connected feedforward network with an input layer consisting of 4 inputs, three hidden layers consisting of 5 neurons each, and an output layer consisting of 1 neuron.

3.2 Recurrent neural networks

A *recurrent neural network* (RNN) is a class of neural networks for processing sequential data [24]. They have been particularly successful applied to machine translation [14] and speech recognition [13]. The structure of the network is similar to the feedforward network but it allows connections between the hidden nodes with a time delay. Through these connections the network can learn from earlier events that are far away from each other in the data.

There exist different type of architectures of recurrent neural networks depending on the goal of using the network. The first architecture consists of non-sequence input and a sequence output, e.g. a picture as input and a description of the picture as output. The second architecture consists of a sequence input and non-sequence output, e.g. classification if a sentence is on English or not. The third architecture consists of a sequence input and a sequence output. This architecture can be divided into two parts where one consists of continuous prediction and the other consists of encoding/decoding.

Given an input sequence $(\mathbf{x}_1, ..., \mathbf{x}_T) \in \mathbb{R}^{m_0 \times T}$, the recurrent network takes a sequence of $(\mathbf{h}_1, ..., \mathbf{h}_T) \in \mathbb{R}^{m_1 \times T}$ hidden states, which are computed at time step t from the equation

$$\mathbf{h}_t = \tanh(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t)$$

where tanh is the recurrent activation function, $W_h \in \mathbb{R}^{m_1 \times m_1}$ is the recurrent weight matrix, and $W_x \in \mathbb{R}^{m_1 \times m_0}$ is the input-to-hidden weight matrix.

For a sequence input to a sequence output $(\mathbf{y}_1, ..., \mathbf{y}_T) \in \mathbb{R}^{m_2 \times T}$ the output from the network is given by

$$\mathbf{y}_t = f^{(1)}(W_y \mathbf{h}_t + \mathbf{b})$$

where $f^{(1)}$ is the output activation function, $W_y \in \mathbb{R}^{m_2 \times m_1}$ is the output weight matrix. Moreover, for a sequence input to a non-sequence output the output from the network is given by

$$\mathbf{y}_T = f^{(1)}(W_y \mathbf{h}_T + \mathbf{b})$$

where $f^{(1)}$ is the output activation function, $W_y \in \mathbb{R}^{m_2 \times m_1}$ is the output weight matrix.

In the figure below we have an illustration of a recurrent neural network with sequence input and sequence output.

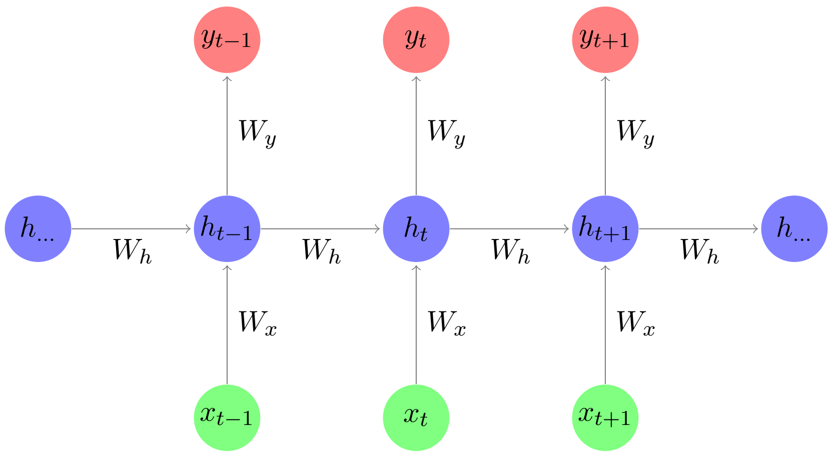

Figure 3.2: Recurrent neural network with sequence input to sequence output, input vector x, weight matrices W_x, W_h, and W_y.

3.3 Activation functions

The most popular activation function in research about neural networks is the function called rectifier, also known as rectified linear unit (ReLU) [25]. The rectifier is defined by

$$f(x) = \max(0, x).$$

One disadvantage of using the rectified linear unit is that if any features are negative then their corresponding weights may not be updated at all.

Another popular activation function is the leaky rectified linear unit (LReLU) [26]. The leaky rectified linear unit is defined by

$$f(x) = \max(0.01x, x).$$

This leaky rectified linear unit take care of the problem of not training weights with negative inputs since the function will give small linear output if the input is negative, i.e. $0.01x$, while it gives a larger output if the input is positive, i.e. x.

Another popular activation function is the scaled-tanh function [25]. The scaled tanh is defined by

$$f(x) = \beta \tanh(\alpha x).$$

One advantage of using the scaled-tanh function is that it has smooth derivative and is relative easy to calculate. One disadvantage of using the scaled-tanh function is that if we have large negative inputs then we will only get $-\beta$ as output while if we have large positive inputs then we will only get β as output.

Another popular activation function is the sigmoid function [25]. The sigmoid is defined by

$$f(x) = \frac{1}{1+e^{-x}}.$$

One advantage of using the sigmoid function is that it has a smooth derivative and are relative easy to calculate. One disadvantage for using the sigmoid is similar to the disadvantage for the tanh function. Another potential weakness of the sigmoid function is that it maps only on the interval $[0, 1]$ and this might introduce a potential bias in the network.

3.4 Cost/loss function

In machine learning we are often interested in a performance measure P, typically accuracy. To optimize the network with respect to this P we introduce a cost function $J(\theta)$ which is related to the performance measure in such a way that a decrease in the cost corresponds to an increase in the performance measure. Typically, when one can derive a Likelihood function it is often appropriate to use the negative log-likelihood as this cost function. [25]

Consider the binary classification problem where we have two different classes, i.e. $y_i = \{0, 1\}$ where 0 corresponds to the outcome of negative price movement and 1 corresponds to the outcome of positive or unchanged price movement. Then, the likelihood can be written as

$$L(\theta; x, y) = \prod_{i=1}^{n} \mathbb{P}(y_i = 1|\theta, x_i)^{y_i} \mathbb{P}(y_i = 0|\theta, x_i)^{(1-y_i)},$$

where n is the number of samples. Therefore, the log-likelihood is given by

$$l = \sum_{i=1}^{n} \left[y_i \log \mathbb{P}(y_i = 1|\theta, x_i) + (1 - y_i) \log \mathbb{P}(y_i = 0|\theta, x_i) \right]. \tag{3.1}$$

The log-likelihood in equation (3.1) may be appropriate for a setup consisting of predicting one market. A generalization of predicting multiple markets at the same time, i.e. multivariate predictions, is given below.

Consider the problem where we have m different markets and each market has two different classes each time step, i.e. $y_{i,j} = \{0, 1\}$. Then, the cost function is given by

$$J(\theta) = -\frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \left[y_{i,j} \log \mathbb{P}(y_{i,j} = 1|\theta, x_{i,j}) + (1 - y_{i,j}) \log \mathbb{P}(y_{i,j} = 0|\theta, x_{i,j}) \right],$$

where n is the number of observations. Notice that $y_{i,j}$ corresponds to the direction of the market on day i and market j.

The interpretation of this cost function in the multivariate case is that we are predicting a matrix instead of a vector, i.e. returns for multiple assets on multiple days. Therefore, a possible scalar cost function is to take the mean over each element in this matrix instead of the mean over each element in a vector, which would be the case in the univariate case. An average over each element corresponds to having equal weights of importance for classifying the m different markets.

A possible disadvantage of using a cost function that only maximizes the accuracy is that it does not necessarily mean that we maximize the return even though high accuracy is correlated with high returns.

3.5 Training neural networks

3.5.1 Batch and minibatch

For most optimization algorithms we need to calculate the gradient of the cost function with respect to the given parameters in order to know how we should update our parameters. This can be very expansive since it requires to evaluate the model on every data point in the entire dataset. Thus, it is common to divide the training data into several parts where each part is called a mini-batch. The main advantage of using mini-batches over using the entire dataset is that the calculation of the gradients will be more efficient since we have smaller datasets.

The size of the mini-batch has the following relationships [25];

- The relationship between the size of the mini-batches and accuracy of the gradient estimate is typically non-linear.
- The time to estimate the gradient may have a lower bound. Thus, using mini-batches smaller than a certain size might be unnecessary since we know that increases in the size of the mini-batches will increase the accuracy of the estimate.
- Using small sizes of the mini-batches can give a regularizing effect since the mini-batches add some type of noise to the learning process [27].

3.5.2 Gradient descent

There exists a multitude of different Gradient Descent algorithms where the aim is to minimize the cost function $J(\theta)$. The most used gradient descent algorithms are batch gradient descent, stochastic gradient descent and mini-batch gradient descent and all these three gradient descent algorithms are similar and only differ in how the training data are used [25].

In the batch gradient descent algorithm the gradient is calculated with respect to the parameter θ for the entire dataset. The updates in the gradient descent algorithm are done according to the following equation

$$\theta_{l+1} = \theta_l - \eta \nabla J(\theta_l),$$

where η is the learning rate. This method is often very slow since it requires calculations of the gradients for the entire dataset in every update.

In the stochastic gradient descent algorithm the gradient is calculated with respect to the parameter θ for only one datapoint in the dataset, i.e. x_i and y_i, and then this is repeated for all training points. The updates in the gradient descent algorithm are done according to the following equation

$$\theta_{l+1} = \theta_l - \eta \nabla J(\theta_l; x_i; y_i),$$

where η is the learning rate. Moreover, the stochastic gradient descent algorithm does just perform one update at a time while the batch gradient descent requires much more computations since it recomputes all the gradients for similar examples before updating each parameter. Thus, in general, the stochastic gradient descent algorithm is much faster. One disadvantage of the stochastic gradient descent is that the updates have higher variance since we only update with one training point at each time.

Moreover, the batch gradient descent does always converge to the closest minimum to the given parameter setup while the stochastic gradient descent may allow you to find a potentially better local minimum due to the variance. This may cause the stochastic gradient descent algorithm to have problems to converge to the exact minimum. However, it has been shown that when decreasing the learning rate, η, in stochastic gradient descent then this algorithm can achieve the same properties as using the batch gradient descent.

In the mini-batch gradient descent algorithm the gradient is calculated with respect to the parameter θ for a mini-batch. Assume that each mini-batch consists of n datapoints. Then, the updates in the mini-batch descent algorithm is done according to the following equation

$$\theta_{l+1} = \theta_l - \eta \nabla J(\theta_{l+1}; x_{i:i+n}; y_{i:i+n}),$$

where η is the learning rate. Moreover, the mini-batch gradient descent algorithm is more time efficient than the batch gradient descent and not necessarily much less time efficient than the stochastic gradient descent algorithm due to the computational lower bound. The mini-batch gradient descent algorithm provides a more accurate estimation of the gradient than the stochastic gradient descent and a less time consuming estimation of the gradient than the batch gradient descent. The mini-batch descent algorithm is the most frequently used algorithm when training a neural network.

As mentioned earlier it is appropriate to decrease to the learning rate over time when using stochastic gradient descent and mini-batch gradient descent due to the introduced noise in the estimation of the gradients. Two sufficient conditions in order to have convergence in the algorithms are

$$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \text{and}$$
$$\sum_{k=1}^{\infty} \eta_k^2 < \infty,$$

i.e. divergent first order sum of coefficient but finite second order sum of coefficient, where η_k the denotes the learning rate at iteration k [25].

One common used way to decrease the learning rate is

$$\eta_{k+1} = \eta_k \cdot \alpha,$$

where α corresponds to the decay factor. Another common used way to decrease the learning rate is to decay the learning until a iteration τ according to the following formula

$$\eta_k = (1-\alpha)\eta_0 + \alpha\eta_\tau,$$

where $\alpha = k/\tau$ and keep the learning rate constant after τ iterations. Notice that the exponential decaying above does not fulfill the two sufficient conditions above but it is a common way for decaying the learning rate. [25]

3.5.3 Momentum

The learning with a descent gradient method can sometimes be very slow [25]. Thus, a natural idea is to add a momentum method since the method of momentum is constructed to speedup the learning process, and especially for high curvatures, or noisy gradients [28]. The momentum term consists of past gradients that have been accumulated with an exponentially decaying moving average.

The momentum is updated according to the following formula

$$\begin{aligned} v_{l+1} &= \alpha v_l - \eta \nabla J(\theta_l), \\ \theta_{l+1} &= \theta_l + v_{l+1}, \end{aligned}$$

where v is often called the velocity vector and $\alpha \in [0,1)$ is the hyperparameter determining how quickly past gradients decay. Notice that if $\alpha = 0$ then the momentum term-vanishes and we are back to the original optimization algorithm without momentum.

3.5.4 Nesterov momentum

The Nesterov momentum is a momentum algorithm inspired by Nesterov's accelerated gradient method [29] [30]. The momentum is updated according to the following formula

$$\begin{aligned} v_{l+1} &= \alpha v_l - \eta \nabla J(\theta_l + \alpha v_l), \\ \theta_{l+1} &= \theta_l + v_{l+1}, \end{aligned}$$

where v is often called the velocity vector and $\alpha \in [0,1)$ is the hyperparameter determining how quickly past gradients decay.

The main difference of Nesterov momentum and the ordinary momentum described earlier is when the gradients are calculated. For the ordinary momentum the gradients are calculated before the velocity has been applied while for the Nesterov momentum the gradients are calculated after the velocity has been applied. Thus, a natural interpretation of the Nesterov momentum is that we want to find an appropriate direction after we have calculated our momentum since we then will be closer to our local optimum.

Moreover, there exists a simplified version of the Nesterov momentum based on the ordinary momentum described above [31]. Assume that $\Theta = \theta + \alpha v_{t-1}$. Then, the simplified Nesterov momentum is given by

$$\begin{aligned} v_{t+1} &= \alpha v_t - \eta \nabla J(\Theta_t), \\ \Theta_{t+1} &= \Theta_t + \alpha^2 v_t - (1+\alpha)\eta \nabla J(\Theta_t). \end{aligned}$$

Assuming that the initial velocity $v_1 = 0$ and the velocity at convergence $v_T \approx 0$, yields that the parameters Θ are equivalent to θ.

Notice that this Nesterov momentum is more similar to the ordinary momentum described earlier since it has just only a different linear combination of ordinary momentum described above.

Moreover, the implementation of Nesterov momentum to recurrent neural networks have increased their performance significantly [13].

3.5.5 RMSProp

The RMSProp is an unpublished optimization algorithm introduced by Geoffrey Hinton in his Coursera Class [32]. It is an algorithm with adaptive learning rate and is inspired by another algorithm called AdaGrad.

The main idea of RMSProp is to adapt the learning rate with respect to past gradients. The past gradients are accumulated with an exponentially weighted moving average instead of an ordinary sum as in the case of AdaGrad. Thus, it prevents the algorithm to have a too small learning before arriving to a local minima.

The RMSProp is updated according to the following formula

$$\begin{aligned} r_l &= \rho r_{l-1} + (1-\rho)\nabla J(\theta_l) \odot \nabla J(\theta_l), \\ \theta_{l+1} &= \theta_l - \frac{\eta}{\sqrt{r_l + \epsilon}} \nabla J(\theta_l), \end{aligned}$$

where $\odot$ denotes element-wise multiplication, ϵ is a small number to prevent numerical errors, and ρ is the hyperparameter determining how quickly past gradients decay. Notice that both r and $\nabla J(\theta)$ are vector valued and therefore the division is done element-wise.

3.6 Regularization

A well known problem in machine learning is how to develop a model that performs well on the test data and not only on the training data. A strategy that is explicitly used to reduce the test error are known as a regularization strategy. Developing effective regularization strategies have been one of the major research in deep learning during the last decade. [25]

3.6.1 Early stopping

A neural network does in general contain a lot of parameters, hence a natural problem to deal with is overfiting. The main idea behind early stopping is to stop the learning process when the validation loss increases while the training loss decreases. Alternatively, stop when validation accuracy decreases while the training accuracy increases.

Therefore, if we stop learning when the validation loss increases then our idea is that the test set has the same properties as the validation test. A consequence is that a model with early stopping has a better validation set error than a model without early stopping.

There exist different types of early stopping, namely;

- Learn until the validation loss increases, or alternatively for a classification problem, learn until the validation accuracy decreases.
- Learn until the validation loss has increased during the last n epochs, or alternatively for a classification problem, learn until the validation accuracy have decreased during the last n epochs.
- Learn for a fixed number of epochs.

A consequence of the simplicity and effectiveness of using early stopping have made it one of the most used regularization techniques in deep learning [25].

3.6.2 Dropout

A common technique to prevent the model from overfitting is called dropout [33]. The main idea behind dropout is to randomly drop units between different layers and thus consider a smaller sub-model instead. The dropout rate is commonly denoted p and is the probability of dropping the unit, i.e. a p close to 1 corresponds to it being likely to drop a unit while a p close to 0 corresponds to it being unlikely to drop a unit.

A consequence of using dropout is that with dropout we train the ensemble consisting of all subnetworks that can be formed of a neural network by removing units from the base network. An example is given below.

Consider a feedforward neural network with two input vectors, one hidden layer consisting two neurons and one output neuron then the full model can be represented by the figure below.

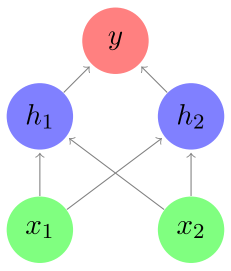

Figure 3.3: A feedforward neural network with two input vectors, one hidden layer consisting two neurons and one output neuron then the full model can be represented by the figure below.

Assuming that we have a dropout rate p larger than 0, then three possible sub-models consists of the three different sub-figures below where in the first model x_1 has been dropped, in the second model x_2 has been dropped and in the third the hidden neuron h_1 have been dropped. This illustration are given in Figure 3.4.

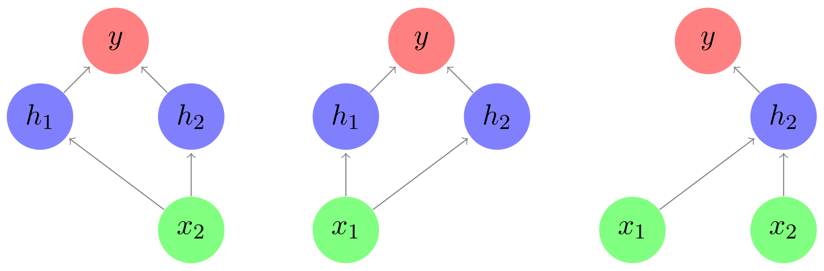

Figure 3.4: Possible sub-models after applying dropout to the model considered in figure 3.3.

3.6.3 Batch normalization

Batch normalization was introduced to speed up the training for neural networks [34], and especially for deep neural networks, since normalization before a layer speeds up the convergence even when features are not decorrelated [35].

The output for a network using batch normalization is given by

$$\begin{aligned}
\hat{\mathbf{s}}_1 &= \text{BatchNorm}_{\gamma,\beta}(W^{(1)T}\mathbf{x} + \mathbf{b}^{(1)}) \\
\mathbf{h}^{(1)} &= f^{(1)}(\hat{\mathbf{s}}_1) \\
\hat{\mathbf{s}}_2 &= \text{BatchNorm}_{\gamma,\beta}(W^{(2)T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) \\
\mathbf{h}^{(2)} &= f^{(2)}(\hat{\mathbf{s}}_2) \\
\mathbf{y} &= f^{(3)}(W^{(3)T}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})
\end{aligned}$$

where BatchNorm is the function that is the affine transformation of a vector to a vector with zero mean and unit variance, i.e. given the mini-batch $(x_1, ..., x_m)$ then the normalization is given by

$$\begin{aligned}
\mu_B &= \frac{1}{m}\sum_{i=1}^{m} x_i, \\
\sigma_B^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2, \\
\hat{x}_i &= \gamma\frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta
\end{aligned}$$

during the training where γ and β are parameters to learn. During training the mean and standard deviation are saved by an exponential moving average of each layer for the forward pass, i.e.

$$\begin{aligned}
\mu_{av} &= \alpha\mu_{av} + (1-\alpha)\mu_B \\
\sigma_{av}^2 &= \alpha\sigma_{av}^2 + (1-\alpha)\sigma_B^2.
\end{aligned}$$

Moreover, notice that the network bias term vanishes when batch normalization is used but the term β acts like a bias term.

3.6.4 Gradient constraint

A general problem for recurrent neural networks are exploding or vanishing gradients and one common way to handle the exploding gradients is to using gradient clipping

[36]. Gradient clipping can be considered to be a regularization method since when we exceed a certain threshold, then, the gradient will be re-scaled and we will not move optimally in the given direction.

The equation for gradient clipping is described by

$$G = \begin{cases} \frac{c}{||\nabla J(\theta)||}\nabla J(\theta) & \text{if } ||\nabla J(\theta)|| \geq c \\ \nabla J(\theta) & \text{otherwise} \end{cases}$$

where G is the scaled gradient used for updating θ, and c is the given threshold.

3.6.5 Ensemble methods

This is a technique used for reducing the generalization error [37]. The main idea is to train k different models separately and then let each model contribute with a prediction, e.g. if each model predicts probabilities then averaging these over all k different models is how ensemble will be used in this thesis. Another type of ensemble is to use the majority vote of each model, i.e. for the binary classification; if 5 of 9 models predict the class 1 then the majority vote would be to predict the class 1 instead of 0. Using multiple of models to predict an outcome will decrease the importance of good initializations for all the weight matrices since a single model not necessarily select the prediction of the ensemble.

Methods like this is called model averaging in machine learning and techniques like this are known as ensemble methods [25].

Chapter 4

Methodology

4.1 Data processing

Data from 03-06-1993 to 17-02-2017 was used to predict 12 different futures contracts, or markets. I am going to take the intersection of dates between the 12 markets since they can be closed on different days. The 12 markets are given in the table below.

Name	**Short Name**	**Asset Class**	**Data Type**
Brent		Commodity	Futures
Gold		Commodity	Futures
Soybeans		Commodity	Futures
EUR		FX	Futures
GBP		FX	Futures
JPY		FX	Futures
10ynote		Rates	Futures
Tbond		Rates	Futures
Bund		Short Rate	Futures
Short Sterling		Short Rate	Futures
FTSE 100		Stock indices	Futures
OMXS 30		Stock indices	Futures
S&P 500		Stock indices	Futures

Table 4.1: Assets used in this thesis

The used data in this report consists of High, Low, Open and Close, denoted H_t, L_t, O_t, and C_t respectively at time t, prices for a certain time horizon. The data was provided by Lynx Asset Management.

4.2 Filters

We are now going to introduce two different type of filters, namely; simple moving average and exponential moving average. In this report we are mainly going to use the exponential moving average due to its properties of fast changes and smooth changes in time.

4.2.1 SMA

Simple moving average with period m, denoted SMA(X, m), is an average over the last m data points of the time series X_t [38]. The formula for calculating the simple moving average is given by

$$\text{SMA}(X_t, m) = \frac{1}{m}\sum_{i=0}^{m-1} X_{t-i}.$$

Note that we need m data points in order to calculate the first value in the time series X_t. Moreover, when the first value have been calculated then we can update the time series according to the following formula

$$\text{SMA}(X_t, m) = \text{SMA}(X_{t-1}, m) + \frac{X_t}{m} - \frac{X_{t-m}}{m}.$$

4.2.2 EMA

Exponential moving average with parameter α, denoted $EMA(\alpha)$, is an infinitely long SMA of the time series X_t with exponential decay [38]. The formula for calculating the α exponential moving average is given by

$$\text{EMA}(X_t, \alpha) = \sum_{i=0}^{\infty} \alpha(1-\alpha)^i X_{t-i}.$$

Similarly as for the simple moving average we can calculate the exponential moving average recursively. The recursively formula is given by

$$\begin{aligned} \text{EMA}(X_1, \alpha) &= X_1 \\ \text{EMA}(X_t, \alpha) &= \alpha X_t + (1-\alpha)\text{EMA}(X_{t-1}, \alpha), \quad \text{for } t > 1. \end{aligned}$$

Moreover, it may be convenient to talk about the exponential moving average as a function of days instead of α. Thus, we can talk about the m day exponential moving average by using the relationship

$$\alpha = \frac{2}{1+m}.$$

4.3 Volatility estimators

In order have an appropriate volatility estimator to normalize time series we are going to introduce the Garman-Klass and Garman-Klass Yang-Zhang Extension volatility estimators. These methods are estimating the different volatility of the market with respect to more parameters than just the close prices.

4.3.1 Garman-Klass

The *Garman-Klass* volatility estimator was created in late 1980 [39]. It is an extension of an volatility estimator created by Parkinson in 1980 which was one of the first advanced volatility estimators based on high and low prices instead of closing prices. The Garman-Klass volatility estimator ignore overnight jumps but is based on high, low, close and opening prices. Consider the time series X_t given by

$$X_t = \frac{1}{2}\left(\ln\left(\frac{H_t}{L_t}\right)\right)^2 - (2\ln(2) - 1)\left(\ln\left(\frac{C_t}{O_t}\right)\right)^2$$

then the SMA version is given by

$$Z_t = \sqrt{F}\sqrt{\text{SMA}(X_t, m)}$$

and the EMA version is given by

$$Z_t = \sqrt{F}\sqrt{\text{EMA}\left(X_t, \frac{2}{1+m}\right)}.$$

4.3.2 Garman-Klass Yang-Zhang extension

The *Garman-Klass Yang-Zhang Extension* is a volatilty estimator created by Yang-Zhang and is adjusted for the jumps in the specific time series [39]. The Garman-Klass Yang-Zhang Extension exists in two different versions namely one SMA version and one EMA version. Consider the time series X_t given by

$$X_t = \left(\ln\left(\frac{O_t}{C_{t-1}}\right)\right)^2 + \frac{1}{2}\left(\ln\left(\frac{H_t}{L_t}\right)\right)^2 - (2\ln(2) - 1)\left(\ln\left(\frac{C_t}{O_t}\right)\right)^2$$

then the SMA version is given by

$$Z_t = \sqrt{F}\sqrt{\text{SMA}(X_t, m)}$$

and the EMA version is given by

$$Z_t = \sqrt{F}\sqrt{\text{EMA}\left(X_t, \frac{2}{1+m}\right)}.$$

4.4 Features

In this section we are going to introduce features that have been found in earlier articles considering prediction of the stock markets.

4.4.1 Momentum

The *momentum* feature is a simple feature that calculates the price difference between two different points. Moreover, the d day momentum is sometimes called lag-d differencing operator in time series analysis due to its property of decompose time series into one trend and one noise component [38]. The momentum is given by

$$\text{Momentum}(m) = X_t - X_{t-m}$$

where m is the length of the momentum.

4.4.2 Rising

The aim of the *rising* feature is to capture either a positive or negative trend. Let X_t be equal to 1 if the closing price at time t was larger than the closing price at time $t-1$. Then the rising feature is given by

$$\text{Rising}(m) = m \cdot \text{SMA}(X_t, m).$$

The main idea of the rising feature is that, if the last couple of days have closed in the same direction then the rising feature should be high while if the market behaves more like a random walk then the rising feature should be low.

4.4.3 Return vs Risk

The aim of the *Return vs Risk* (RvR) feature is to consider the trade-off between risk and returns of a specific asset. The idea of considering the trade-off between risk and returns was introduced by H. Markowitz in a 1952 essay [40] where he discuss the idea of finding a portfolio with lowest possible risk given a specific return. Moreover, in finance the returns are usually very small on a daily-basis and therefore we can estimate the variance by only the second order moment and neglect the mean. Thus, the feature is given by

$$\begin{aligned}
\hat{V}_t(m) &= \text{EMA}\left((C_t - C_{t-1})^2, \frac{2}{1+m}\right), \\
\text{RvR}(m) &= \frac{C_t - C_{t-1}}{\sqrt{\hat{V}_{t-1}(m)}}.
\end{aligned}$$

4.4.4 Stochastic K%

The aim of the *stochastic K%* feature is to get an indication about the current price relative to the highest and lowest prices during the last m days [41]. The stochastic K% is a feature to identify bearish or bullish markets, i.e. decreasing or increasing markets. The feature is given by

$$\text{StochK}(m)_t = \frac{C_t - \min\{L_t, ..., L_{t-m}\}}{\max\{H_t, ..., H_{t-m}\} - \min\{L_t, ..., L_{t-m}\}} \cdot 100$$

where C_t is the closing price at day t, L_t is the lowest price at day t and H_t is the highest price at day t. A common value for m is 14.

4.4.5 Stochastic D%

Stochastic D% is a feature of smoothing the stochastic K% [41]. The m_1 day stochastic D% is given by

$$\text{StochD}(m_1, m_2)_t = \text{SMA}(\text{StochK}(m_2), m_1),$$

where $\text{StochK}(m_2)$ is the stochastic K% with period m_2. A common parameter setup for (m_1, m_2) is $(3, 14)$

4.4.6 Stochastic slow D%

Stochastic slow D% is a feature of smoothing the stochastic D% [41]. The m_1 day stochastic slow D% is given by

$$\text{StochD}(m_1, m_2, m_3)_t = \text{SMA}(\text{StochD}(m_2, m_3), m_1)$$

where $\text{StochD}(m_2, m_3)$ is the stochastic D% feature with parameter (m_2, m_3). A common parameter setup for (m_1, m_2, m_3) is $(3, 3, 14)$.

4.4.7 Smoothed changes

The aim of the *smoothed changes* is to get a direction whether the current price is low or high relative to the last m days. The m day smoothed changes is given by

$$\text{SC}(m) = \left(\frac{C_t - \text{SMA}(C_t, m)}{\text{SMA}(C_t, m)} \right) \cdot 100.$$

4.4.8 Percentage price oscillator

The aim of the *percentage price oscillator* is to measure the difference between two moving averages. The percentage price oscillator measures the relative difference instead of the absolute difference [42]. The percentage price oscillator is given by

$$\text{PPO}(m_1, m_2) = \left(\frac{\text{SMA}(X_t, m_1) - \text{SMA}(X_t, m_2)}{\text{SMA}(X_t, m_2)} \right) \cdot 100,$$

where m_1 and m_2 determine the lengths of the simple moving averages, and $m_1 < m_2$ to have a short minus a long.

4.4.9 Relative strength index

The aim of the *relative strength index* (RSI) is to measure the speed and change of price movements. The RSI is bounded between 0 and 100. Moreover, RSI is a very popular momentum indicator due to its strength of indicating whether an asset is overbought, oversold, and have a general trend [43]. RSI is considered to be overbought when it is above 70 and oversold when it is less than 30.

Let Δ_t, U_t and D_t be equal to $C_t - C_{t-1}$, $\Delta_t \mathbb{1}(\Delta_t > 0)$ and $\Delta_t \mathbb{1}(\Delta_t < 0)$ respectively. Then, the m day RSI is given by

$$\begin{aligned} \text{RS} &= \frac{SMA(U, m)}{SMA(D, m)} \\ \text{RSI}(m) &= 100 - \frac{100}{1 + \text{RS}}. \end{aligned}$$

4.4.10 Williams R%

The aim of *Williams %R* is to get an intuition about how the current price relates to previous prices. The Williams %R is bounded between -100 and 0 where values between 0 and -20 indicate that the asset is overbought while values between -80 and -100 indicate that the asset is oversold [44]. The feature is given by

$$\text{WilliamsR}(m)_t = -\frac{\max\{H_t, ..., H_{t-m}\} - C_t}{\max\{H_t, ..., H_{t-m}\} - \min\{L_t, ..., L_{t-m}\}} \cdot 100$$

where C_t is the closing price at day t, L_t is the lowest price at day t and H_n is the highest price at day t. A common parameter for m is 14. Williams R% is a mirrored version of the Stochastic K% along the 0%-line, i.e. $\text{StochK}(m)_t - \text{WilliamsR}(m)_t = 1$.

4.4.11 Commodity channel index

The aim of the commodity channel index is to warn for extreme conditions or identify trends. The commodity channel index was in the beginning developed for identifying cyclic behaviour in the commodity market but can be successfully applied to other securities [45]. The commodity channel index measures the current price relative to historical average prices. The commodity channel index can identify when the asset is overbought or oversold. The feature is given by

$$\begin{aligned} X_t &= \frac{C_t + H_t + L_t}{3}, \\ \text{CCI}(m) &= \frac{X_t - \text{SMA}(X_t, m)}{0.015 \cdot \text{SMA}(|X_t - \text{SMA}(X_t, m)|, m)}, \end{aligned}$$

where X_t is the typical price at day t, and $|X_t|$ is the absolute value of the typical price at time t. Moreover, the term in the denominator is often called constant times the mean deviation.

4.5 Scaling features

An important property in machine learning to speed up the learning is to scale the training features by making them 0 mean and unit variance. This can only be done for the training data and then the features in the validation and training set are scaled with the same constants. This is described in the algorithm below.

Algorithm 4.1 Scaling training, validation, and test features

```
for all the features do
    $\hat{x}_{\texttt{train}}, \mu_{x_{\texttt{train}}}, \sigma_{x_{\texttt{train}}} \leftarrow x_{\texttt{train}}$
    $\hat{x}_{\texttt{val}} \leftarrow x_{\texttt{val}}, \mu_{x_{\texttt{train}}}, \sigma_{x_{\texttt{train}}}$
    $\hat{x}_{\texttt{test}} \leftarrow x_{\texttt{test}}, \mu_{x_{\texttt{train}}}, \sigma_{x_{\texttt{train}}}$
end for
```

4.6 Feature setup

In this section we are going to describe which features that were used when we are going to talk about a neural networks with less complex and more complex features. Moreover, all time series were risk-adjusted by the volatility estimator Garman-Klass Yang-Zhang Extension with $\sqrt{F} = 1$.

4.6.1 Less complex

For the less complex features we consider 9 different features for each market, namely;

- Close price minus a 10 day exponential moving average of close prices.
- Close price minus a 20 day exponential moving average of close prices.
- Close price minus a 60 day exponential moving average of close prices.
- 5 day exponential moving average for 10 day momentum of close prices.
- 5 day exponential moving average for 20 day momentum of close prices.
- 5 day exponential moving average for 60 day momentum of close prices.
- 5 day exponential moving average of close prices minus a 60 day exponential moving average of close prices.
- 20 day exponential moving average of close prices minus a 60 day exponential moving average of close prices.
- 40 day exponential moving average of close prices minus a 60 day exponential moving average of close prices.

4.6.2 More complex

For the more complex features we are consider 14 different features for each market, namely;

- 1 day momentum of close prices.
- 10 day momentum of close prices.
- 6 day exponential moving average of close prices minus 12 days exponential moving average.
- Rising with the parameter 10.
- Rising with the parameter 20.

- 20 day return vs risk.
- Stochastic K with the parameter 14.
- Stochastic D with the parameters 3 and 14.
- Stochastic D slow with the parameters 3, 3 and 14
- Relative strength index with the parameter 14.
- Williams R% with the parameter 14.
- Commodity channel index with the parameter 20.
- Percentage Price Oscillator with the parameters 5 and 10.
- Smoothed changes with the parameter 6.

4.7 Architectures

In this thesis we will consider two main architectures. The first architecture is a vanilla feedforward neural network as described in the previous chapter. The second architecture consists of a recurrent layer mapping into a feedforward neural network, this network will be referred to as recurrent neural network. All the networks will be constructed without a bias term since finance data is generally very noisy and a possible consequence of noisy data together with a bias term in a network will be that the network trains to either predict up or down during the whole period, i.e. the bias term dominating the prediction.

For the feedforward neural network we will consider different type of setups, namely; non-deep networks with many neurons vs. deep networks with less neurons, networks with batch normalization vs. networks without batch normalization, and networks with simple features vs. networks with complex features. The activation functions through all hidden layers in the feedforward network are scaled-tanh with $\beta = 1.7159$ and $\alpha = 2/3$, similar as used by Estrada [8].

The non-deep networks consist of 1 hidden layer with either 100, 200, or 400 neurons. The deep networks consist of 4 hidden layer with either 25, 50, or 100 neurons. The number of neurons will be kept constant through the hidden layers.

For the recurrent architecture we will consider one recurrent layer of the type sequence input and non-sequence output and then mapping into a feedforward neural network before the output layer. The recurrent layer will have either 5, 10, 20, 40, or 80 neurons. The feedforward network will consist of 1 hidden layer with 100 neurons. The activation functions for the hidden layers in the feedforward network are scaled-tanh with $\beta = 1.7159$ and $\alpha = 2/3$, similar as used by Estrada [8].

The network built on a recurrent layer will have two type of setups, namely; network with simple features vs. network with complex features.

The output layer in these two networks consists of 12 neurons with a sigmoid activation function. One reason why the scaled-tanh is used is because symmetric functions tends to speed up the learning process for a neural network [25]. Another reason why scaled-tanh was used instead of the more classical ReLU is because of zero inputs in the network may give a zero gradient through all training and a bias is introduced for positive inputs.

Moreover, another advantage of using scaled-tanh is due to the universal approximation theorem, which says that any function can be approximated if the hidden layers consists of "squashing" functions and enough of hidden layers.

4.8 Ensemble

In this thesis we will consider one type of ensemble. The ensemble computes the average output of k different models. For the feedforward neural networks the ensemble is done by 16 different models while for the recurrent architecture the ensemble is done by 8 different models. One reason why we have more models in our ensemble for the feedforward neural networks is due to computational efficiency since the recurrent neural network requires much more computations.

4.9 Prediction setup

The data is divided into three different parts, namely; training-, validation- and test set. The test set consists of 1500 values made on 5 different parts, namely; each model has a training set, validation set, and training set consisting of 3400, 300, and 300 data points respectively. Then training is done by rolling these data sets forward, e.g. the first model predicts the data points from $1 - 300$, the second model predicts the data points from $301 - 600$, and so fourth. Thus, the predictions of the 1500 are done in 5 different steps and the union of the predicted days corresponds to predicting 1500 consecutive days. Moreover, it is important to be careful when selecting the training, validation and test sets since we have time series. For instance, it is important that the training set consists of the first 3400 consecutive data points, then the validation set of the following 300 consecutive data points and lastly the test set of the last 300 consecutive data points to not introduce some type of knowledge about the future. A more brief description about the training of the model is given in the algorithm below.

Algorithm 4.2 Predict 1500 consecutive values with 5 different periods

```
for steps = 1 to 5 do
    Training data ← $X_{300\cdot(\texttt{steps}-1)+1}, \ldots, X_{300\cdot(\texttt{steps}-1)+3400}$
    Validation data ← $X_{300\cdot(\texttt{steps}-1)+3401}, \ldots, X_{300\cdot(\texttt{steps}-1)+3700}$
    Test data ← $X_{300\cdot(\texttt{steps}-1)+3701}, \ldots, X_{300\cdot(\texttt{steps}-1)+4000}$
    Initialize a new model
    Train the new model on the training data
    Predict the test data with the trained model
end for
```

The response variable is given by the relationship below

$$y_{t,j} = \mathbb{1}\left(O_{t+2,j} - O_{t+1,j} \geq 0\right)$$

where $O_{t,j}$ corresponds to the opening price at time t for market j. The reason why $O_{t+2} - O_{t+1}$ is considered instead of the more classical setup $C_{t+1} - C_t$ is because we have a multivariate setup and do not want to introduce some type of biased look-forward in our features, e.g. Asian markets closes before American markets and therefore American close prices should not help to predict Asian close prices.

4.10 Comparing with benchmark

The benchmark in this thesis will be the prediction $Y_{t+1} = Y_t$, i.e. take the same position as the label of the previous day. To determine if the corresponding network is better than this benchmark I will do a comparison with Accuracy and p-value.

The p-value will be constructed in the following way. From previous studies we know that returns between different days are nearly uncorrelated, but returns between different markets on the same day are correlated [4]. On each day we predict the label of 12 different markets. Thus, construct a vector with a 1 at place i if the network predicted more correct labels on day i than the benchmark, a 0 if the network predicted less correct labels on day i than the benchmark and remove the data point from the observation if the network and benchmark predicted correctly on the same number of markets at day i. Therefore, at each position in the vector we will have a Bernoulli distributed random variable and a sum of Bernoulli random variables are binomial distributed. However, we are interested in determine whethering it is more likely that our networks predict more correctly than the benchmark thus we want to show that $p > 0.5$. Therefore, the following test statistic can be constructed

$$T = \frac{\sum_{i=1}^{n} X_i - np}{\sqrt{npq}},$$

where n is the length of the vector and $p = q = 0.5$.

4.10.1 Combining p-values

To determine whether a specific model is good or not we have to combine multiple of p-values since each ensemble has 1 p-value. In order to combine different p-values we used Fisher's method [46] [47]. The Fisher's method uses the formula

$$T = -2\sum_{i=1}^{n} \log(p_i) \sim \chi^2(2n)$$

where p_i is the p-value for test i, since under the null hypothesis $p_i \sim \text{Unif}(0,1)$. For small p-values T tends to be large which suggests that the null hypothesis are not true for all tests.

4.11 Implementation

The results in this report have been produced by Python, Theano, and Lasagne. Theano is a library in Python developed for using deep learning [48]. The calculations in Theano are done on either CPU or GPU, depending on the user setup. Lasagne is an open-source package built on Theano to simplify for the user, e.g. different type of architectures have been implemented in Lasagne with Theano code such as recurrent layers and batch normalization.

There exist many other packages for implementing neural networks, e.g. Caffe , Keras, and TensorFlow, but the results in this report will not depend on which package that was used. Caffe and TensorFlow are similar to Theano and Keras is similar to Lasagne.

Moreover, I have parallelized each training model in the ensemble since each model in the ensemble can be trained independently of each other. The Python package multi-processing was used for parallelization.

Chapter 5

Results

5.1 Parameters

After some trial and error the feedforward neural networks are trained with RMSProp, Nesterov-momentum, dropout, and lastly with and without batch normalization. The parameters are given in the table below.

Method	Parameter	Value
RMSProp	η	$1 \cdot 10^{-3}$
RMSProp	ρ	0.95
Momentum	α	0.8
Dropout	p	0.5
Batch norm.	α	0.9
Ensemble	k	16
Epochs		40
Early Stopping		No

After some trial and error the recurrent neural networks are trained with RMSProp, Nesterov-momentum, dropout, exponential decaying the learning rate, gradient clipping, and early stopping with respect to validation accuracy. The parameters are given in the table below.

Method	Parameter	Value
RMSProp	η	$1 \cdot 10^{-3}$
RMSProp	ρ	0.95
Momentum	α	0.95
Decaying η	η_τ	$1 \cdot 10^{-5}$
Decaying η	τ	10
Gradient clipping	c	$1 \cdot 10^{-2}$
Dropout	p	0.5
Ensemble	k	8
Epochs		10
Early Stopping		Yes
Sequence length	T	300

5.2 Benchmark

The used benchmark corresponds to taking the same position at last day, i.e. if the label of yesterday was 0 then the label of today is 0 and if the label of yesterday was 1 then the label of today is 1. However, implementing this yields an accuracy of 49.50%. This benchmark will be used to compare the performance of the models.

Model	Accuracy	Sh
Benchmark	49.50%	-1.2394

5.3 Less complex features

In this section we will consider the less complex feature setup introduced in Section 4.6.1. The model architectures are feedforward network, either non-deep with many units or deep with less units, and a type of recurrent neural network with either 5, 10, 20, 40, or 80 hidden nodes and then connected to a feedforward network with 1 hidden layer and 100 units. Also, for the feedforward neural network we will consider a case of either using batch normalization or not.

For the feedforward neural network the tables will be organized as follows following; Each row in the table, except the last row, corresponds to the result of the ensemble with 16 different models with different initialization. The last row corresponds to an average model of the 5 rows above. Accuracy corresponds to the given test accuracy for the ensemble, Mean corresponds to the average test accuracy for these 16 models, Std corresponds to the average standard deviation of test accuracy within all the 16 models, p-value corresponds to the p-value of that the model predicts better than the benchmark, Reject corresponds to the p-value being less than or greater than 5%, and Sh corresponds to the Sharpe ratio of the ensemble.

For the recurrent architecture the table will be organized in the same way except that we instead use an ensemble of 8 different models with different initializations.

5.3.1 Feedforward neural network

5.3.1.1 Without batch normalization

In the Table 5.1 below we present the results for the non-deep feedforward networks without batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
50.78%	50.53%	$6.39 \cdot 10^{-3}$	$1.240 \cdot 10^{-1}$	No	0.4066
50.46%	50.44%	$6.70 \cdot 10^{-3}$	$1.398 \cdot 10^{-1}$	No	−0.0022
50.75%	50.51%	$6.43 \cdot 10^{-3}$	$1.607 \cdot 10^{-1}$	No	0.3524
50.60%	50.43%	$5.85 \cdot 10^{-3}$	$1.179 \cdot 10^{-1}$	No	0.0735
50.66%	50.52%	$6.72 \cdot 10^{-3}$	$1.534 \cdot 10^{-1}$	No	0.4938
50.65%	50.49%	$6.42 \cdot 10^{-3}$	$3.129 \cdot 10^{-2}$	Yes	0.2648

(a) (1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
50.58%	50.29%	$4.87 \cdot 10^{-3}$	$1.071 \cdot 10^{-1}$	No	0.2477
50.59%	50.26%	$7.02 \cdot 10^{-3}$	$1.063 \cdot 10^{-1}$	No	0.3675
50.43%	50.33%	$5.67 \cdot 10^{-3}$	$2.031 \cdot 10^{-1}$	No	0.1581
50.39%	50.33%	$7.47 \cdot 10^{-3}$	$2.271 \cdot 10^{-1}$	No	0.0637
50.49%	50.38%	$6.62 \cdot 10^{-3}$	$1.728 \cdot 10^{-1}$	No	0.2543
50.50%	50.32%	$6.33 \cdot 10^{-3}$	$4.543 \cdot 10^{-2}$	Yes	0.2183

(b) (1,200)

Accuracy	Mean	Std	p-value	Reject	Sh
50.16%	50.27%	$6.42 \cdot 10^{-3}$	$3.004 \cdot 10^{-1}$	No	0.0027
50.27%	50.35%	$6.67 \cdot 10^{-3}$	$1.603 \cdot 10^{-1}$	No	0.1046
50.20%	50.21%	$6.37 \cdot 10^{-3}$	$1.226 \cdot 10^{-1}$	No	0.0554
49.93%	50.17%	$6.73 \cdot 10^{-3}$	$3.506 \cdot 10^{-1}$	No	−0.1118
50.39%	50.28%	$5.91 \cdot 10^{-3}$	$8.765 \cdot 10^{-2}$	No	0.0307
50.19%	50.26%	$6.42 \cdot 10^{-3}$	$6.944 \cdot 10^{-2}$	No	0.0163

(c) (1,400)

Table 5.1: 1 hidden layer networks with less complex feature setup and without batch normalization

We observe that no individual model has a p-value less than 5%. However, we can see that the networks with 1 hidden layer and either 100 and 200 neurons have a combined p-value less than 5%. Another interesting property is that models with higher p-values seem to have lower accuracies. Moreover, the Sharpe ratios for the networks with less neurons seem to be higher, but the Sharpe ratio oscillating depending one the initialization.

In Table 5.2 we present the results for the deep feedforward networks without batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
50.78%	50.52%	$7.99 \cdot 10^{-3}$	$1.796 \cdot 10^{-1}$	No	0.6890
50.37%	50.34%	$7.27 \cdot 10^{-3}$	$1.228 \cdot 10^{-1}$	No	0.2741
50.61%	50.93%	$8.25 \cdot 10^{-3}$	$9.172 \cdot 10^{-2}$	No	0.4512
50.59%	50.41%	$6.10 \cdot 10^{-3}$	$3.290 \cdot 10^{-1}$	No	0.3916
50.48%	50.54%	$6.12 \cdot 10^{-3}$	$7.056 \cdot 10^{-2}$	No	0.4497
50.56%	50.44%	$7.15 \cdot 10^{-3}$	$2.990 \cdot 10^{-2}$	Yes	0.4511

(a) (4,25)

Accuracy	Mean	Std	p-value	Reject	Sh
50.84%	50.55%	$7.46 \cdot 10^{-3}$	$8.044 \cdot 10^{-1}$	No	0.5523
51.02%	50.71%	$9.30 \cdot 10^{-3}$	$3.873 \cdot 10^{-2}$	Yes	0.4043
50.62%	50.43%	$8.06 \cdot 10^{-3}$	$1.977 \cdot 10^{-1}$	No	0.4164
51.09%	50.77%	$9.79 \cdot 10^{-3}$	$9.494 \cdot 10^{-3}$	Yes	0.9568
50.60%	50.48%	$9.05 \cdot 10^{-3}$	$1.799 \cdot 10^{-1}$	No	0.4469
50.83%	50.59%	$8.73 \cdot 10^{-3}$	$1.103 \cdot 10^{-2}$	Yes	0.5553

(b) (4,50)

Accuracy	Mean	Std	p-value	Reject	Sh
51.36%	50.59%	$8.04 \cdot 10^{-3}$	$2.906 \cdot 10^{-2}$	Yes	0.6931
50.57%	50.49%	$9.42 \cdot 10^{-3}$	$2.349 \cdot 10^{-1}$	No	0.0492
50.59%	50.51%	$1.05 \cdot 10^{-2}$	$7.509 \cdot 10^{-2}$	No	0.1499
51.06%	50.65%	$9.69 \cdot 10^{-3}$	$9.272 \cdot 10^{-2}$	No	0.4674
50.95%	50.68%	$8.72 \cdot 10^{-3}$	$7.071 \cdot 10^{-2}$	No	0.1328
50.91%	50.58%	$9.28 \cdot 10^{-3}$	$4.967 \cdot 10^{-3}$	Yes	0.2985

(c) (4,100)

Table 5.2: 4 hidden layer networks with less complex feature setup and without batch normalization

We observe that it is not likely that individual models have p-values less than 5%. However, we can see that all the networks with 4 hidden layer have a combined p-value of less than 5%. Moreover, the Sharpe ratios for the networks with less neurons seem to be higher, but the Sharpe ratios oscillating depending on the initialization.

5.3.1.2 With batch normalization

In Table 5.3 we present the results for the non-deep feedforward networks with batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
51.15%	51.27%	$4.04 \cdot 10^{-3}$	$6.147 \cdot 10^{-2}$	No	0.7486
51.32%	51.28%	$4.49 \cdot 10^{-3}$	$3.073 \cdot 10^{-2}$	Yes	0.8844
51.18%	51.22%	$4.37 \cdot 10^{-3}$	$3.485 \cdot 10^{-2}$	Yes	0.9419
51.37%	51.25%	$4.19 \cdot 10^{-3}$	$2.719 \cdot 10^{-2}$	Yes	0.9942
51.36%	51.30%	$4.03 \cdot 10^{-3}$	$1.468 \cdot 10^{-2}$	Yes	0.8567
51.28%	51.26%	$4.22 \cdot 10^{-3}$	$1.294 \cdot 10^{-4}$	Yes	0.8852

(a) (1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
51.29%	51.21%	$4.23 \cdot 10^{-3}$	$4.152 \cdot 10^{-2}$	Yes	1.0287
51.26%	51.09%	$3.70 \cdot 10^{-3}$	$1.019 \cdot 10^{-1}$	No	0.9453
51.20%	51.21%	$4.89 \cdot 10^{-3}$	$2.401 \cdot 10^{-2}$	Yes	0.9325
51.14%	51.18%	$3.68 \cdot 10^{-3}$	$1.029 \cdot 10^{-1}$	No	0.8129
51.40%	51.20%	$4.76 \cdot 10^{-3}$	$2.556 \cdot 10^{-2}$	Yes	0.9357
51.26%	51.18%	$4.25 \cdot 10^{-3}$	$7.735 \cdot 10^{-4}$	Yes	0.9310

(b) (1,200)

Accuracy	Mean	Std	p-value	Reject	Sh
50.82%	50.77%	$4.91 \cdot 10^{-3}$	$2.521 \cdot 10^{-2}$	Yes	0.7608
50.79%	50.70%	$4.94 \cdot 10^{-3}$	$3.122 \cdot 10^{-2}$	Yes	0.6945
50.89%	50.82%	$4.65 \cdot 10^{-3}$	$4.408 \cdot 10^{-2}$	Yes	0.7558
50.86%	50.77%	$5.50 \cdot 10^{-3}$	$2.756 \cdot 10^{-2}$	Yes	0.6994
50.96%	50.90%	$4.84 \cdot 10^{-3}$	$5.827 \cdot 10^{-2}$	No	0.8952
50.86%	50.79%	$4.97 \cdot 10^{-3}$	$2.327 \cdot 10^{-4}$	Yes	0.7611

(c) (1,400)

Table 5.3: 1 hidden layer networks with less complex feature setup and with batch normalization

We can observe that 10 out of 15 the individual models have a p-value less than 5%. However, we can see that all the networks with 1 hidden layer have a combined p-value of less than 5%. Another interesting property is that networks with less neurons seem to have higher accuracies. Moreover, the Sharpe ratios for all the networks seem to be more stable now than before.

In Table 5.4 we present the results for the deep feedforward networks with batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
50.87%	50.83%	$4.70 \cdot 10^{-3}$	$4.776 \cdot 10^{-2}$	Yes	0.9383
51.02%	50.76%	$4.30 \cdot 10^{-3}$	$1.437 \cdot 10^{-2}$	Yes	0.9932
50.74%	50.67%	$4.00 \cdot 10^{-3}$	$2.719 \cdot 10^{-2}$	Yes	0.7218
51.02%	50.90%	$3.86 \cdot 10^{-3}$	$2.564 \cdot 10^{-2}$	Yes	1.0569
50.87%	50.79%	$4.13 \cdot 10^{-3}$	$2.426 \cdot 10^{-2}$	Yes	0.8158
50.90%	50.79%	$4.20 \cdot 10^{-3}$	$6.787 \cdot 10^{-5}$	Yes	0.9052

(a) (4,25)

Accuracy	Mean	Std	p-value	Reject	Sh
50.87%	50.83%	$4.70 \cdot 10^{-3}$	$4.776 \cdot 10^{-2}$	Yes	0.9383
51.02%	50.76%	$4.30 \cdot 10^{-3}$	$1.437 \cdot 10^{-2}$	Yes	0.9932
50.74%	50.67%	$4.00 \cdot 10^{-3}$	$2.719 \cdot 10^{-2}$	Yes	0.7218
51.02%	50.90%	$3.86 \cdot 10^{-3}$	$2.564 \cdot 10^{-2}$	Yes	1.0569
50.87%	50.79%	$4.13 \cdot 10^{-3}$	$2.426 \cdot 10^{-2}$	Yes	0.8158
50.90%	50.79%	$4.20 \cdot 10^{-3}$	$6.787 \cdot 10^{-5}$	Yes	0.9052

(b) (4,50)

Accuracy	Mean	Std	p-value	Reject	Sh
50.95%	50.89%	$4.61 \cdot 10^{-3}$	$6.875 \cdot 10^{-2}$	No	0.7412
51.01%	50.98%	$4.48 \cdot 10^{-3}$	$5.574 \cdot 10^{-2}$	No	0.7565
51.03%	50.96%	$4.08 \cdot 10^{-3}$	$4.256 \cdot 10^{-2}$	Yes	1.0311
50.98%	50.97%	$3.95 \cdot 10^{-3}$	$5.009 \cdot 10^{-2}$	No	0.7852
50.81%	50.92%	$3.94 \cdot 10^{-3}$	$7.038 \cdot 10^{-2}$	No	0.7582
50.96%	50.95%	$4.21 \cdot 10^{-3}$	$1.374 \cdot 10^{-3}$	Yes	0.8144

(c) (4,100)

Table 5.4: 4 hidden layer networks with less complex feature setup and with batch normalization

We can observe that 11 out of the 15 individual models have a p-value less than 5%. However, we can see that the networks with 1 hidden layer have a combined p-value of less than 5%. Another interesting property is that networks with less neurons seem to have higher accuracies. Moreover, the Sharpe ratios for all the networks seem to be more stable now than before.

5.3.2 Recurrent neural network

In Table 5.5 we present the results for the recurrent network architectures.

Accuracy	Mean	Std	p-value	Reject	Sh
50.23%	50.41%	$8.59 \cdot 10^{-3}$	$3.481 \cdot 10^{-1}$	No	0.3481
50.43%	50.35%	$9.00 \cdot 10^{-3}$	$2.121 \cdot 10^{-1}$	No	−0.3986
50.86%	50.57%	$9.41 \cdot 10^{-3}$	$9.037 \cdot 10^{-2}$	No	−0.5188
51.30%	50.66%	$1.15 \cdot 10^{-2}$	$3.727 \cdot 10^{-2}$	Yes	0.5175
50.76%	50.31%	$1.03 \cdot 10^{-2}$	$8.243 \cdot 10^{-2}$	No	−0.1810
50.72%	50.46%	$9.76 \cdot 10^{-3}$	$1.730 \cdot 10^{-2}$	Yes	−0.0587

(a) (5,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
51.37%	50.77%	$7.52 \cdot 10^{-3}$	$1.488 \cdot 10^{-2}$	Yes	0.3625
51.13%	50.59%	$9.22 \cdot 10^{-3}$	$4.779 \cdot 10^{-2}$	Yes	0.3142
51.08%	50.66%	$7.52 \cdot 10^{-3}$	$6.912 \cdot 10^{-3}$	Yes	0.1509
50.48%	50.33%	$9.47 \cdot 10^{-3}$	$2.626 \cdot 10^{-1}$	No	−0.6880
50.13%	50.20%	$1.00 \cdot 10^{-2}$	$1.318 \cdot 10^{-1}$	No	−0.7787
50.84%	50.51%	$8.75 \cdot 10^{-3}$	$5.491 \cdot 10^{-4}$	Yes	−0.1278

(b) (10,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
51.02%	50.50%	$8.87 \cdot 10^{-3}$	$5.876 \cdot 10^{-3}$	Yes	0.3778
50.76%	50.48%	$9.53 \cdot 10^{-3}$	$7.692 \cdot 10^{-3}$	Yes	−0.1623
50.94%	50.47%	$9.42 \cdot 10^{-3}$	$6.412 \cdot 10^{-3}$	Yes	0.1929
50.63%	50.45%	$6.77 \cdot 10^{-3}$	$8.145 \cdot 10^{-2}$	No	−0.0521
50.72%	50.33%	$6.99 \cdot 10^{-3}$	$7.446 \cdot 10^{-2}$	No	0.1679
50.81%	50.45%	$8.32 \cdot 10^{-3}$	$1.489 \cdot 10^{-5}$	Yes	0.1048

(c) (20,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
50.31%	50.53%	$7.43 \cdot 10^{-3}$	$1.338 \cdot 10^{-1}$	No	−0.0722
51.21%	50.53%	$8.86 \cdot 10^{-3}$	$1.174 \cdot 10^{-2}$	Yes	−0.1976
50.48%	50.62%	$6.66 \cdot 10^{-3}$	$1.801 \cdot 10^{-1}$	No	−0.4989
50.43%	50.49%	$8.41 \cdot 10^{-3}$	$9.927 \cdot 10^{-2}$	No	−0.4768
50.54%	50.50%	$8.09 \cdot 10^{-3}$	$3.381 \cdot 10^{-1}$	No	−0.2052
50.60%	50.53%	$7.89 \cdot 10^{-3}$	$1.028 \cdot 10^{-2}$	Yes	−0.2489

(d) (40,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
51.06%	50.74%	$8.22 \cdot 10^{-3}$	$4.035 \cdot 10^{-2}$	Yes	0.1092
50.58%	50.38%	$6.56 \cdot 10^{-3}$	$2.692 \cdot 10^{-2}$	Yes	−0.1617
50.44%	50.38%	$7.78 \cdot 10^{-3}$	$1.377 \cdot 10^{-1}$	No	−0.1111
51.12%	50.55%	$8.04 \cdot 10^{-3}$	$2.883 \cdot 10^{-3}$	Yes	0.2007
51.22%	50.53%	$7.61 \cdot 10^{-3}$	$5.674 \cdot 10^{-2}$	No	0.2430
50.89%	50.52%	$7.64 \cdot 10^{-3}$	$1.224 \cdot 10^{-4}$	Yes	0.0560

(e) (80,1,100)

Table 5.5: Recurrent architecture with less complex feature setup

We can observe that the p-values are oscillating depending on model structure. Another interesting property is that models that have a p-value less than 5% seem to have a higher Sharpe ratio. Moreover, we can observe that increases in accuracy do not in general result in increases in Sharpe ratio.

5.4 More complex features

In this section we will consider the more complex feature setup introduced in Section 4.6.2. The model architectures are feedforward network, either non-deep with many units or deep with less units. For the feedforward neural network we will consider a case of either using batch normalization or not.

The tables are organized similarly as in Section 5.3.

5.4.1 Feedforward NN

5.4.1.1 Without batch normalization

In Table 5.6 we present the results for the non-deep feedforward networks without batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
52.50%	52.15%	$4.75 \cdot 10^{-3}$	$3.247 \cdot 10^{-4}$	Yes	0.3365
52.18%	52.02%	$4.84 \cdot 10^{-3}$	$9.310 \cdot 10^{-5}$	Yes	0.0799
52.44%	52.17%	$4.71 \cdot 10^{-3}$	$3.186 \cdot 10^{-5}$	Yes	0.4077
52.33%	52.18%	$5.01 \cdot 10^{-3}$	$6.790 \cdot 10^{-5}$	Yes	0.2626
52.42%	52.11%	$4.55 \cdot 10^{-3}$	$5.485 \cdot 10^{-5}$	Yes	0.1754
52.37%	52.13%	$4.77 \cdot 10^{-3}$	$7.772 \cdot 10^{-16}$	Yes	0.2524

(a) (1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
52.25%	52.10%	$5.38 \cdot 10^{-3}$	$9.100 \cdot 10^{-5}$	Yes	0.6077
52.22%	51.95%	$5.19 \cdot 10^{-3}$	$5.110 \cdot 10^{-5}$	Yes	0.4529
51.97%	51.92%	$4.43 \cdot 10^{-3}$	$3.368 \cdot 10^{-4}$	Yes	0.1976
51.99%	51.90%	$5.58 \cdot 10^{-3}$	$1.854 \cdot 10^{-4}$	Yes	0.1782
52.07%	52.00%	$4.75 \cdot 10^{-3}$	$1.231 \cdot 10^{-4}$	Yes	0.2431
52.10%	51.98%	$5.07 \cdot 10^{-3}$	$6.550 \cdot 10^{-15}$	Yes	0.3359

(b) (1,200)

Accuracy	Mean	Std	p-value	Reject	Sh
52.07%	51.95%	$5.20 \cdot 10^{-3}$	$1.184 \cdot 10^{-4}$	Yes	0.3015
52.00%	51.88%	$5.50 \cdot 10^{-3}$	$1.304 \cdot 10^{-3}$	Yes	0.1137
51.78%	51.96%	$5.36 \cdot 10^{-3}$	$4.658 \cdot 10^{-4}$	Yes	0.2316
51.99%	51.91%	$4.98 \cdot 10^{-3}$	$9.310 \cdot 10^{-5}$	Yes	0.1703
51.99%	51.95%	$5.52 \cdot 10^{-3}$	$2.620 \cdot 10^{-4}$	Yes	−0.0259
51.97%	51.93%	$5.31 \cdot 10^{-3}$	$2.257 \cdot 10^{-13}$	Yes	0.1582

(c) (1,400)

Table 5.6: 1 hidden layer networks with complex feature setup and without batch normalization

We can observe that all the individual models have p-values less than 5%. Therefore, their combined p-values are less than 5%. Moreover, the accuracy is significantly higher than for the case with less complex features. We can also observe that the Sharpe ratio tends to be positive, but oscillating.

In Table 5.7 we present the results for the deep feedforward networks without batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
52.29%	52.37%	$3.92 \cdot 10^{-3}$	$2.853 \cdot 10^{-5}$	Yes	-0.1323
52.44%	52.29%	$4.38 \cdot 10^{-3}$	$3.026 \cdot 10^{-5}$	Yes	0.1490
52.52%	52.46%	$4.14 \cdot 10^{-3}$	$3.146 \cdot 10^{-5}$	Yes	0.0435
52.54%	52.42%	$4.24 \cdot 10^{-3}$	$1.092 \cdot 10^{-5}$	Yes	0.1544
52.33%	52.39%	$3.84 \cdot 10^{-3}$	$9.082 \cdot 10^{-5}$	Yes	0.1782
52.42%	52.39%	$4.10 \cdot 10^{-3}$	0	Yes	0.0786

(a) (4,25)

Accuracy	Mean	Std	p-value	Reject	Sh
52.41%	52.28%	$5.18 \cdot 10^{-3}$	$4.760 \cdot 10^{-5}$	Yes	0.0542
52.42%	52.32%	$4.25 \cdot 10^{-3}$	$2.966 \cdot 10^{-5}$	Yes	0.1571
52.34%	52.23%	$5.71 \cdot 10^{-3}$	$6.459 \cdot 10^{-6}$	Yes	0.1021
52.27%	52.16%	$4.80 \cdot 10^{-3}$	$4.169 \cdot 10^{-5}$	Yes	-0.0389
52.37%	52.35%	$5.16 \cdot 10^{-3}$	$2.780 \cdot 10^{-5}$	Yes	0.1755
52.36%	52.27%	$5.02 \cdot 10^{-3}$	0	Yes	0.0900

(b) (4,50)

Accuracy	Mean	Std	p-value	Reject	Sh
52.02%	51.95%	$5.82 \cdot 10^{-3}$	$1.071 \cdot 10^{-4}$	Yes	-0.1601
52.04%	52.05%	$5.09 \cdot 10^{-3}$	$6.552 \cdot 10^{-5}$	Yes	-0.0551
51.90%	51.98%	$4.75 \cdot 10^{-3}$	$1.171 \cdot 10^{-4}$	Yes	-0.2027
52.10%	51.94%	$5.30 \cdot 10^{-3}$	$1.014 \cdot 10^{-5}$	Yes	-0.0500
52.08%	52.08%	$5.09 \cdot 10^{-3}$	$1.242 \cdot 10^{-4}$	Yes	0.0086
52.03%	52.00%	$5.21 \cdot 10^{-3}$	$2.220 \cdot 10^{-16}$	Yes	-0.0917

(c) (4,100)

Table 5.7: 4 hidden layers networks with more complex feature setup and without batch normalization

We observe that all the individual models have p-values less than 5%. Therefore, their combined p-values are less than 5%. Moreover, the accuracy is significantly higher than for the case with less complex features. The accuracy seems to be higher for networks with less neurons. Also, the accuracy within a certain type of model structure seems to be stable. We can also observe that the Sharpe ratios tend to be positive, but oscillating.

5.4.1.2 With batch normalization

In Table 5.8 we present the results for the non-deep feedforward networks with batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
51.18%	51.05%	$4.17 \cdot 10^{-3}$	$3.195 \cdot 10^{-1}$	No	0.6915
51.13%	51.11%	$3.78 \cdot 10^{-3}$	$3.491 \cdot 10^{-1}$	No	0.8585
51.12%	51.13%	$4.50 \cdot 10^{-3}$	$3.699 \cdot 10^{-1}$	No	0.7930
51.02%	51.04%	$3.93 \cdot 10^{-3}$	$2.887 \cdot 10^{-1}$	No	0.7469
51.02%	51.10%	$4.12 \cdot 10^{-3}$	$3.490 \cdot 10^{-1}$	No	0.7539
51.09%	51.09%	$4.10 \cdot 10^{-3}$	$3.602 \cdot 10^{-1}$	No	0.7688

(a) (1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
51.02%	51.01%	$4.70 \cdot 10^{-3}$	$3.188 \cdot 10^{-1}$	No	0.7922
51.07%	50.98%	$4.07 \cdot 10^{-3}$	$2.609 \cdot 10^{-1}$	No	0.8710
50.97%	50.95%	$4.08 \cdot 10^{-3}$	$4.123 \cdot 10^{-1}$	No	0.8706
50.97%	51.15%	$3.92 \cdot 10^{-3}$	$3.393 \cdot 10^{-1}$	No	0.8845
50.73%	51.00%	$4.41 \cdot 10^{-3}$	$3.808 \cdot 10^{-1}$	No	0.5346
50.95%	50.99%	$4.23 \cdot 10^{-3}$	$3.703 \cdot 10^{-1}$	No	0.7906

(b) (1,200)

Accuracy	Mean	Std	p-value	Reject	Sh
51.11%	51.04%	$4.73 \cdot 10^{-3}$	$1.024 \cdot 10^{-1}$	No	0.7405
51.34%	50.98%	$5.63 \cdot 10^{-3}$	$9.289 \cdot 10^{-2}$	No	0.8546
50.87%	50.78%	$4.59 \cdot 10^{-3}$	$4.129 \cdot 10^{-1}$	No	0.6194
50.98%	50.08%	$4.47 \cdot 10^{-3}$	$1.084 \cdot 10^{-1}$	No	0.4521
51.13%	50.86%	$4.73 \cdot 10^{-3}$	$2.448 \cdot 10^{-1}$	No	0.5787
51.09%	50.89%	$4.83 \cdot 10^{-3}$	$4.952 \cdot 10^{-2}$	Yes	0.6491

(c) (1,400)

Table 5.8: 1 hidden layer networks with more complex feature setup and with batch normalization

We observe that all the individual models has p-values larger than 5%. The combined p-values are less than 5% for only the network with 400 neurons. Moreover, we can observe that the p-values are rather high which is not the case for the more complex feature setup without batch normalization. We can also observe that the Sharpe ratios tend to be higher than without batch normalization and tend to be stable.

In Table 5.9 we present the results for the deep feedforward networks with batch normalization.

Accuracy	Mean	Std	p-value	Reject	Sh
51.12%	50.94%	$5.60 \cdot 10^{-3}$	$3.952 \cdot 10^{-2}$	Yes	0.7787
50.88%	50.95%	$3.77 \cdot 10^{-3}$	$5.868 \cdot 10^{-2}$	No	0.7315
50.84%	51.05%	$5.39 \cdot 10^{-3}$	$4.639 \cdot 10^{-2}$	Yes	0.5977
50.89%	50.86%	$5.11 \cdot 10^{-3}$	$6.800 \cdot 10^{-2}$	No	0.7877
51.02%	50.96%	$5.86 \cdot 10^{-3}$	$5.951 \cdot 10^{-2}$	No	0.6003
50.95%	50.95%	$5.15 \cdot 10^{-3}$	$1.116 \cdot 10^{-3}$	Yes	0.6992

(a) (4,25)

Accuracy	Mean	Std	p-value	Reject	Sh
50.98%	51.04%	$4.92 \cdot 10^{-3}$	$1.411 \cdot 10^{-1}$	No	0.7448
51.08%	51.08%	$5.23 \cdot 10^{-3}$	$1.224 \cdot 10^{-1}$	No	0.9619
51.11%	51.03%	$4.39 \cdot 10^{-3}$	$1.480 \cdot 10^{-1}$	No	0.8287
51.07%	51.06%	$4.58 \cdot 10^{-3}$	$8.438 \cdot 10^{-2}$	No	0.7520
51.04%	51.03%	$4.49 \cdot 10^{-3}$	$1.545 \cdot 10^{-1}$	No	0.6505
51.05%	51.05%	$4.72 \cdot 10^{-3}$	$2.392 \cdot 10^{-2}$	Yes	0.7876

(b) (4,50)

Accuracy	Mean	Std	p-value	Reject	Sh
51.04%	51.03%	$4.56 \cdot 10^{-3}$	$1.818 \cdot 10^{-1}$	No	0.6899
51.20%	51.09%	$3.80 \cdot 10^{-3}$	$2.386 \cdot 10^{-1}$	No	0.8430
51.09%	51.12%	$4.28 \cdot 10^{-3}$	$1.683 \cdot 10^{-1}$	No	0.5987
51.04%	51.02%	$3.85 \cdot 10^{-3}$	$2.135 \cdot 10^{-1}$	No	0.8532
51.26%	51.22%	$4.45 \cdot 10^{-3}$	$3.404 \cdot 10^{-1}$	No	0.8396
51.14%	51.09%	$4.19 \cdot 10^{-3}$	$1.291 \cdot 10^{-1}$	No	0.7649

(c) (4,100)

Table 5.9: 4 hidden layers networks with more complex feature setup and with batch normalization

We observe that 13 out of 15 models have p-values larger than 5%. The combined p-values are less than 5% for the networks with 25 and 50 neurons. Moreover, we observe that the p-values are rather high which is not the case for the more complex feature setup without batch normalization, but the p-values tend to be smaller for the deep network relative to the non-deep. We can also observe that the Sharpe ratios tend to be higher than without batch normalization and tends to be stable.

5.4.2 Recurrent NN

In Table 5.10 we present the results for the recurrent network architectures.

Accuracy	Mean	Std	p-value	Reject	Sh
52.81%	52.48%	$3.93 \cdot 10^{-3}$	$4.094 \cdot 10^{-6}$	Yes	0.1476
52.82%	52.64%	$4.71 \cdot 10^{-3}$	$2.427 \cdot 10^{-6}$	Yes	0.3592
52.51%	52.31%	$3.77 \cdot 10^{-3}$	$1.752 \cdot 10^{-5}$	Yes	−0.0271
52.68%	52.40%	$7.63 \cdot 10^{-3}$	$1.080 \cdot 10^{-4}$	Yes	0.2335
52.28%	52.40%	$4.75 \cdot 10^{-3}$	$1.226 \cdot 10^{-5}$	Yes	0.1555
52.62%	52.45%	$4.96 \cdot 10^{-3}$	0	Yes	0.1737

(a) (5,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
52.69%	52.45%	$5.88 \cdot 10^{-3}$	$5.081 \cdot 10^{-7}$	Yes	0.5285
52.42%	52.49%	$5.16 \cdot 10^{-3}$	$9.446 \cdot 10^{-6}$	Yes	0.4368
52.57%	52.48%	$3.88 \cdot 10^{-3}$	$5.552 \cdot 10^{-5}$	Yes	0.2901
52.56%	52.35%	$6.23 \cdot 10^{-3}$	$3.685 \cdot 10^{-6}$	Yes	0.1880
52.52%	52.40%	$4.76 \cdot 10^{-3}$	$6.443 \cdot 10^{-6}$	Yes	0.1962
52.55%	52.44%	$5.18 \cdot 10^{-3}$	0	Yes	0.3279

(b) (10,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
52.42%	52.38%	$3.58 \cdot 10^{-3}$	$1.192 \cdot 10^{-5}$	Yes	0.0653
52.72%	52.45%	$4.44 \cdot 10^{-3}$	$9.532 \cdot 10^{-7}$	Yes	0.4078
52.59%	52.37%	$4.06 \cdot 10^{-3}$	$4.160 \cdot 10^{-6}$	Yes	0.3185
52.82%	52.24%	$7.37 \cdot 10^{-3}$	$1.095 \cdot 10^{-7}$	Yes	0.2556
52.50%	52.21%	$5.37 \cdot 10^{-3}$	$8.694 \cdot 10^{-7}$	Yes	0.2641
52.61%	52.33%	$4.96 \cdot 10^{-3}$	0	Yes	0.2623

(c) (20,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
52.57%	52.41%	$5.02 \cdot 10^{-3}$	$5.168 \cdot 10^{-7}$	Yes	0.3730
52.63%	52.03%	$6.75 \cdot 10^{-3}$	$2.557 \cdot 10^{-7}$	Yes	0.4308
52.64%	52.28%	$6.01 \cdot 10^{-3}$	$1.437 \cdot 10^{-5}$	Yes	0.2429
52.62%	52.33%	$5.32 \cdot 10^{-3}$	$1.344 \cdot 10^{-6}$	Yes	0.2642
52.27%	52.16%	$7.05 \cdot 10^{-3}$	$9.703 \cdot 10^{-7}$	Yes	0.3386
52.55%	52.24%	$6.03 \cdot 10^{-3}$	0	Yes	0.3299

(d) (40,1,100)

Accuracy	Mean	Std	p-value	Reject	Sh
52.35%	52.14%	$5.96 \cdot 10^{-3}$	$1.458 \cdot 10^{-5}$	Yes	0.0646
52.47%	52.05%	$9.41 \cdot 10^{-3}$	$3.467 \cdot 10^{-5}$	Yes	0.0935
52.76%	52.20%	$4.72 \cdot 10^{-3}$	$1.728 \cdot 10^{-5}$	Yes	0.0610
52.67%	52.04%	$6.12 \cdot 10^{-3}$	$3.537 \cdot 10^{-5}$	Yes	0.1209
52.29%	52.02%	$5.73 \cdot 10^{-3}$	$1.789 \cdot 10^{-4}$	Yes	−0.1761
52.51%	52.09%	$6.39 \cdot 10^{-3}$	0	Yes	0.0328

(e) (80,1,100)

Table 5.10: Recurrent architecture with more complex feature setup

We observe that all models has p-values larger than 5%, and in fact the p-values tend to be really small. Therefore, the combined p-values are less than 5% for all networks. Moreover, we can also observe that the Sharpe ratios tend to be positive even though they are not either large or stable.

5.5 Summary

We have a summary of the feedforward network in Table 5.11. The first 12 rows correspond to feedforward networks trained without batch normalization and the 12 rows below correspond to feedforward networks trained with batch normalization.

Feature	Network	Accuracy	Mean	Std	p-value	Reject	Sh
Simple	(1,100)	50.65%	50.49%	$6.42 \cdot 10^{-3}$	$3.129 \cdot 10^{-2}$	Yes	0.2648
Simple	(1,200)	50.50%	50.32%	$6.33 \cdot 10^{-3}$	$4.543 \cdot 10^{-2}$	Yes	0.2183
Simple	(1,400)	50.19%	50.26%	$6.42 \cdot 10^{-3}$	$6.944 \cdot 10^{-2}$	No	0.0163
Simple	(4,25)	50.56%	50.44%	$7.15 \cdot 10^{-3}$	$2.990 \cdot 10^{-2}$	Yes	0.4511
Simple	(4,50)	50.83%	50.59%	$8.73 \cdot 10^{-3}$	$1.103 \cdot 10^{-2}$	Yes	0.5553
Simple	(4,100)	50.91%	50.58%	$9.28 \cdot 10^{-3}$	$4.967 \cdot 10^{-3}$	Yes	0.2985
Complex	(1,100)	52.37%	52.13%	$4.77 \cdot 10^{-3}$	$7.772 \cdot 10^{-16}$	Yes	0.2524
Complex	(1,200)	52.10%	51.98%	$5.07 \cdot 10^{-3}$	$6.550 \cdot 10^{-15}$	Yes	0.3359
Complex	(1,400)	51.97%	51.93%	$5.31 \cdot 10^{-3}$	$2.257 \cdot 10^{-13}$	Yes	0.1582
Complex	(4,25)	52.42%	52.39%	$4.10 \cdot 10^{-3}$	0	Yes	0.0786
Complex	(4,50)	52.36%	52.27%	$5.02 \cdot 10^{-3}$	0	Yes	0.0900
Complex	(4,100)	52.03%	52.00%	$5.21 \cdot 10^{-3}$	$2.220 \cdot 10^{-16}$	Yes	−0.0917
Simple	(1,100)	51.28%	51.26%	$4.22 \cdot 10^{-3}$	$1.294 \cdot 10^{-4}$	Yes	0.8852
Simple	(1,200)	51.26%	51.18%	$4.25 \cdot 10^{-3}$	$7.735 \cdot 10^{-4}$	Yes	0.9310
Simple	(1,400)	50.86%	50.79%	$4.97 \cdot 10^{-3}$	$2.327 \cdot 10^{-4}$	Yes	0.7611
Simple	(4,25)	50.84%	50.81%	$4.95 \cdot 10^{-3}$	$1.182 \cdot 10^{-4}$	Yes	0.8068
Simple	(4,50)	50.90%	50.79%	$4.20 \cdot 10^{-3}$	$6.787 \cdot 10^{-5}$	Yes	0.9052
Simple	(4,100)	50.96%	50.95%	$4.21 \cdot 10^{-3}$	$1.374 \cdot 10^{-3}$	Yes	0.8144
Complex	(1,100)	51.09%	51.09%	$4.10 \cdot 10^{-3}$	$3.602 \cdot 10^{-1}$	No	0.7688
Complex	(1,200)	50.95%	50.99%	$4.23 \cdot 10^{-3}$	$3.703 \cdot 10^{-1}$	No	0.7906
Complex	(1,400)	51.09%	50.89%	$4.83 \cdot 10^{-3}$	$4.952 \cdot 10^{-2}$	Yes	0.6491
Complex	(4,25)	50.95%	50.95%	$5.15 \cdot 10^{-3}$	$1.116 \cdot 10^{-3}$	Yes	0.6992
Complex	(4,50)	51.05%	51.05%	$4.72 \cdot 10^{-3}$	$2.392 \cdot 10^{-2}$	Yes	0.7876
Complex	(4,100)	51.14%	51.09%	$4.19 \cdot 10^{-3}$	$1.291 \cdot 10^{-1}$	No	0.7649
	Benchmark	49.50%					−1.2394

Table 5.11: Summary of results for the feedforward networks. The first 12 rows correspond to feedforward networks trained without batch normalization and the 12 rows below correspond to feedforward networks trained with batch normalization.

We observe that the accuracies for the models with more complex feature setup are higher than for the models with less complex feature setup. Moreover, the Sharpe ratios seem to be higher for models with batch normalization. Also, notice that for the less complex feature setup the accuracies seem to increase when adding batch normalization while for the more complex feature setup the accuracies seem to decrease when adding batch normalization.

We have a summary of the recurrent architecture in Table 5.12.

Feature	Network	Accuracy	Mean	Std	p-value	Reject	Sh
Simple	(5,1,100)	50.72%	50.46%	$9.76 \cdot 10^{-3}$	$1.730 \cdot 10^{-2}$	Yes	-0.0587
Simple	(10,1,100)	50.84%	50.51%	$8.75 \cdot 10^{-3}$	$5.491 \cdot 10^{-4}$	Yes	-0.1278
Simple	(20,1,100)	50.81%	50.45%	$8.32 \cdot 10^{-3}$	$1.489 \cdot 10^{-5}$	Yes	0.1048
Simple	(40,1,100)	50.60%	50.53%	$7.89 \cdot 10^{-3}$	$1.028 \cdot 10^{-2}$	Yes	-0.2489
Simple	(80,1,100)	50.89%	50.52%	$7.64 \cdot 10^{-3}$	$1.224 \cdot 10^{-4}$	Yes	0.0560
Complex	(5,1,100)	52.62%	52.45%	$4.96 \cdot 10^{-3}$	0	Yes	0.1737
Complex	(10,1,100)	52.55%	52.44%	$5.18 \cdot 10^{-3}$	0	Yes	0.3279
Complex	(20,1,100)	52.61%	52.33%	$4.96 \cdot 10^{-3}$	0	Yes	0.2623
Complex	(40,1,100)	52.55%	52.24%	$6.03 \cdot 10^{-3}$	0	Yes	0.3299
Complex	(80,1,100)	52.51%	52.09%	$6.39 \cdot 10^{-3}$	0	Yes	0.0328
	Benchmark	49.50%					-1.2394

Table 5.12: Summary of results for the recurrent architecture.

We observe that the accuracies for the models with more complex feature setup are higher than for the models with less complex feature setup. Moreover, the Sharpe ratios seem to be higher for models with more complex features than models with less complex features. Also, notice that for the more complex feature setup the accuracies seem to be higher than for the feedforward neural networks with more complex features, and the accuracies for the less complex feature seem to be higher than for the feedforward neural networks without batch normalization, but not higher than for the feedforward neural networks with batch normalization.

Chapter 6

Discussion

In this section the results from Chapter 5 will be discussed. The discussion consider the following cases; deep vs. non-deep networks, recurrent vs. feedforward networks, difference between the less complex and more complex feature setup and relationship between accuracy and Sharpe ratio.

6.1 Deep vs. non-deep networks

We will start to investigate Table 5.11 to answer the question if it is preferable to use a deep network with fewer neurons compared to a non-deep network with more neurons or the other way around. The answer can be divided into the four following cases; less complex features without batch normalization, less complex features with batch normalization, more complex features without batch normalization, more complex features with batch normalization. To get more detailed answers we investigate the underlying models in Table 5.1 to 5.4 and 5.6 to 5.9.

6.1.1 Less complex features without batch normalization

For the less complex features without batch normalization we observe that the two best accuracies correspond to deep networks. The combined p-values are less than 5% for all deep networks and for 2 out of 3 models for non-deep networks. Therefore, we need to study Table 5.1 and 5.2 to investigate this further.

In Table 5.1 we observe that increases of neurons yield decreases in accuracy and in Table 5.2 increases of neurons yields increases in accuracy. Comparing networks with equally many neurons in Table 5.1 and 5.2 implies that (4,50) and (4,100) dominate (1,200) and (1,400) respectively. Moreover, we observe that (1,100) has a higher combined accuracy than (4,25), but when we observe at individual level we can not see any significant results that suggests that (1,100) outperforms (4,25). Thus, for this setup deep networks seems to slightly outperform non-deep networks.

If we instead focus on the p-values in Table 5.11, then we observe that deep networks tends to have smaller p-values. However, from Table 5.1 and 5.2 we observe that only 1 out of 15 models has a p-value under 10% for the non-deep networks and 8 out of

15 models for the deep networks. Thus, deep networks seem to outperform non-deep networks.

The deep network seem to slightly outperform the non-deep networks when taking both accuracies and p-values into consideration.

6.1.2 Less complex features with batch normalization

For the less complex features with batch normalization we observe that the two best accuracies correspond to non-deep networks. The combined p-value are less than 5% for all networks. Therefore, we need to study Table 5.3 and 5.4 to investigate this further.

In Table 5.3 we can see that increases of neurons give decreases in accuracy for non-deep networks and in Table 5.4 increases of neurons do not seem to affect the accuracies. In Table 5.3 we observe that 10 out of 15 networks have accuracies over 51% while only 6 out of 15 networks have accuracies over 51% in Table 5.4. Comparing networks with equally many neurons in table 5.3 and 5.4 shows that $(1, 100)$ and $(1, 200)$ dominate $(4, 25)$ and $(4, 50)$ respectively. Moreover, we observe that $(4, 100)$ have a higher combined accuracy than $(1, 400)$. Furthermore, we observe that deep and non-deep networks have accuracies larger than 50.90% in 4 out of 5 and 1 out of 5 networks respectively. Therefore, for smaller number of neurons for non-deep networks seem to outperform deep networks, but with larger number of neurons deeper networks seem to outperform non-deep networks. Thus, there does not exist a general pattern whether deep networks outperforms non-deep networks or the other way around.

If we instead focus on the p-values in Table 5.3 and 5.4, then both non-deep and deep networks have 11 out of 15 networks with p-values less than 5%. We observe that all networks for the two smallest deep networks have p-values less than 5% and the largest has only p-values less than 5% for 1 out of 5 networks. For the non-deep networks the p-values are less than 5% for either 3 or 4 out of 5 networks. However, comparing the different p-values does not give any significant answer whether to use deep or non-deep networks.

None of the networks seems to outperform the other when taking both accuracies and p-values into consideration.

6.1.3 More complex features without batch normalization

For the more complex features without batch normalization we observe that both the best and worst accuracies correspond to non-deep networks. The combined p-values are really small for both non-deep and deep networks. Moreover, we need to study Table 5.6 and 5.7 to investigate this further.

In Table 5.6 we observe that 10 out of 15 networks have an accuracy above 52% and in Table 5.7 we observe that 14 out of 15 networks have accuracies over 52%. Comparing these two tables we observe that the deep networks seem to slightly outperform the non-deep networks with respect to accuracy.

If we instead focus on the p-values in Table 5.6 and 5.7, then we observe that all models have small p-values but deep networks seem to have smaller p-values than non-deep networks. Thus, deep networks seem to slightly outperform non-deep networks with respect to p-values.

Deep networks seems to slightly outperform non-deep networks when taking both accuracies and p-values into consideration.

6.1.4 More complex features with batch normalization

For the more complex features with batch normalization we observe that both the best and worse accuracies correspond to deep networks. The combined p-values are less than 5% for 1 out of 3 non-deep networks and for 2 out of 3 deep networks. Therefore, we need to study Table 5.8 and 5.9 to investigate this further.

In Table 5.8 we observe that 10 out of 15 networks have higher accuracies than 51% and 11 out of 15 networks for the networks in Table 5.9. Comparing networks with equally many neurons does not give any significant results whether to use deep or non-deep networks. Therefore, none of the networks seem to outperform the other.

If we instead look at the p-values then we observe that none of non-deep networks and only 2 out of 15 of deep networks have a p-value less than 5%. The deep networks have in general smaller p-values than non-deep networks even though most of the p-values are larger than 5%. Thus, none of the networks seem to outperforms the other.

None of the networks seem to outperform the other when taking both accuracies and p-values into consideration.

6.1.5 Summary

Summarizing these four cases gives us that the deep networks seem to slightly outperform the non-deep networks in two cases and in the two other cases we cannot conclude if one of the networks outperforms the other. Therefore, the conclusion is that usage of deep or non-deep networks depends on the problem setup, e.g. features and network architecture.

6.2 RNN vs. FNN

We start to compare Table 5.11 and 5.12 to answer the question if recurrent neural network can learn more from the features due to its time-dependency. To answer this question we will start by comparing the two following cases; less complex features and more complex features. Moreover, investigation about the influence of batch normalization is also necessary.

6.2.1 Less complex features

For the less complex features a comparison between Table 5.11 and 5.12 suggests that recurrent neural networks have similar results as feedforward neural networks without batch normalization, and the feedforward neural networks with batch normalization seem to outperforms recurrent neural networks. However, to study this further we need to investigate the results in Table 5.1 to 5.5.

Comparing Table 5.1 and 5.2 with Table 5.3 and 5.4 we observe that batch normalization increases the accuracy significantly. Moreover, the networks seem to be stable and robust with respect to accuracies. In Table 5.5 we observe that the accuracies are not stable and robust with respect to accuracies. The greatest accuracies and many of the worst accuracies correspond to recurrent neural networks. Therefore, we cannot conclude whether feedforward neural networks or recurrent neural networks are preferable in this setup, but we can conclude that adding batch normalization increases the accuracies for the feedforward neural networks.

6.2.2 More complex features

For the more complex features a comparison between Table 5.11 and 5.12 suggests that recurrent neural networks achieve slightly better results than all the feedforward neural networks. However, to study this further we need to investigate the results in Table 5.6 to 5.10.

Comparing Table 5.6 and 5.7 with Table 5.8 and 5.9 we observe that batch normalization decreases the accuracy significantly. For both recurrent neural networks and feedforward neural networks the accuracies seem to be stable and robust with respect to the accuracies. The greatest accuracies correspond to recurrent neural networks and the worst accuracies to feedforward neural networks. For instance, it is rarely that feedforward neural networks achieve accuracies over 52.5%, 3 out of 30, but likely for recurrent neural networks, 18 out of 25. Therefore, we conclude that recurrent neural networks for the more complex feature setup seem to slightly outperform the feedforward neural networks.

6.2.3 Summary

Summarizing these two cases gives us that introducing recurrent neural networks does not necessarily increase the accuracy significantly.

For the less complex feature setup recurrent neural networks have an oscillating effect on the accuracy while the feedforward networks seem to be more stable and robust. Moreover, increases in accuracies can be done by adding batch normalization to the feedforward neural networks instead of using recurrent neural networks.

For the more complex feature setup both recurrent neural networks and feedforward neural networks are stable and robust with respect to accuracies. The recurrent neural networks seem to slightly improve the accuracy compared to the feedforward neural networks. Moreover, adding batch normalization to the feedforward neural networks decreases the accuracies significantly instead of increasing as the case for less complex features.

6.3 Feature setup

We start to compare Table 5.11 and 5.12 to answer the question if some feature setup seems to outperform another. In both Table 5.11 and 5.12 we observe that the more complex feature setup seems to outperform the less complex feature setup for all networks except for feedforward neural networks with batch normalization. For feedforward neural networks with batch normalization we cannot see any pattern whether the more complex feature setup are better than the less complex feature setup and vice versa. Therefore, in general it might be appropriate to select reasonable feature setups since features seem to increase the accuracies even though in some cases increases in accuracies can be done by training with another method, e.g. batch normalization.

The reason why the less complex feature setup improves the accuracies relative to the more complex feature setup for the feedforward neural network with batch normalization is because of large decreses in accuracies for the more complex features and small increases for the less complex features. One potential reason why batch normalization decreases the accuracies for the more complex feature setup can be that the more complex feature setup describes the output relative well and normalizing the input before each activation function may destroy the financial signals, e.g. destroy the intuition when an asset is overbought/oversold.

6.4 Sharpe ratio

We start to compare Table 5.11 and 5.12 to answer the question if high accuracies are strongly correlated with high Sharpe ratios. From Table 5.11 we observe that the networks with highest accuracies, more complex feature setup without batch normalization, do not have high or highest Sharpe ratios and not even necessarily positive Sharpe ratios even though the accuracies are above 52%. Similar results can be found in Table 5.12 where we have recurrent neural networks. To get more detailed answers we investigate the underlying models given in Table 5.1 to 5.10.

In Table 5.1 to 5.2, 5.5 to 5.7 and 5.10 we observe that the Sharpe ratios are very oscillating, for instance it can be both negative and positive for similar setups. In Table 5.3 to 5.4 and 5.8 to 5.9 we observe that the Sharpe ratios do not oscillate very much and seem to be more stable. Therefore, we conclude that for all networks without batch normalization the accuracies seems to not have any strong influence on the Sharpe ratio but when batch normalization are used then, the networks seem to have high and stable Sharpe ratio.

Chapter 7

Conclusion

In general deep networks do not outperform non-deep networks and vice versa. Deep networks can be preferable depending on feature setup and network architecture, e.g. feedforward networks without batch normalization. The non-deep networks do not significantly outperform the deep networks in any scenario but are equally good in some cases.

Recurrent neural networks do not necessarily outperform feedforward neural networks and batch normalization can improve feedforward neural networks. For the less complex feature setup it can be appropriate to keep the feedforward neural network and adding batch normalization instead of adding more complexity to the model, i.e. introducing recurrent neural networks. For the more complex feature setup it can be appropriate to introduce a more complex model such as recurrent neural networks to achieve more signals from the time-dependent data. Moreover, for the more complex feature setup batch normalization decreased the accuracies.

In general the more complex feature setup increased the accuracy significantly compared to the less complex feature setup. Introducing batch normalization to the feedforward neural networks have a positive effect on the less complex feature setup but negative for the more complex feature setup.

The Sharpe ratios seem to not be strongly correlated with high accuracies. For all networks without batch normalization, the Sharpe ratios oscillated and could be both negative and positive even for networks with accuracies above 52%. For all networks with batch normalization the Sharpe ratios became more stable and positive.

7.1 Future work

This thesis has given an introduction on how recurrent neural networks can be used to forecast price movements of futures contracts. There exist different interesting results to investigate further and I am going to introduce some of them here.

7.1.1 Different assets and asset classes

This thesis focuses on forecasting price movements of futures contracts. To generalize these results in financial forecasting it would be interesting to investigate if the networks perform similarly on different asset classes, e.g. stocks and bonds. The networks use features from all markets in this thesis and it would be interesting to introduce more assets to see if the networks can learn more from the correlation between markets.

7.1.2 Loss function with respect to return

This thesis focuses on optimizing the accuracy of forecasting price movements of futures contracts. In the discussion we observed that high accuracies does not imply good Sharpe ratios. In practice Sharpe ratio is of greater importance than accuracy since we want a model to be able to gain profits. Therefore, it would be interesting to introduce another type of cost function that takes the return into consideration when optimizing the networks, e.g. weighted cross-entropy loss where the weights depend on returns.

7.1.3 Stacked recurrent networks

This thesis focuses on introducing the recurrent neural network in forecasting price movements of futures contracts. Therefore, the recurrent part in the recurrent neural network consists of only 1 hidden recurrent layer. Then, a natural question is whether stacked recurrent layers can improve the results or is the data so noisy that financial time series cannot be considered as typical sequential data and the complexity of recurrent layers can be neglected.

7.1.4 Long Short-Term Memory

This thesis focuses on introducing the recurrent neural network in forecasting price movements of futures contracts. Therefore, the recurrent layer consists of the most vanilla recurrent layer. Long Short-Term Memory (LSTM) networks was introduced since vanilla recurrent layers can be difficult to train. Thus, can LSTM improve the results of vanilla recurrent neural networks given in this report.

7.1.5 Feature importance

In this thesis we consider two different feature setups, namely; the less complex and more complex feature setup. In the discussion we observed that the more complex feature setup seem to outperform the less complex feature setup. Therefore, it would be interesting to determine which of the more complex features improves the accuracy most, can the feature parameters be optimized to get a better result, and does it exist other complex features that can boost the accuracy.

Bibliography

[1] B. Malkiel, *A Random Walk Down Wall Street: Including a Life-cycle Guide to Personal Investing.* Norton, 1999.

[2] E. F. Fama, "Stock returns, real activity, inflation and money," *American Economic Review*, vol. 71, pp. 545–565, 1981.

[3] E. F. Fama, "Stock returns, expected returns, and real activity," *Journal of Finance*, vol. 45, pp. 1575–1617, 1990.

[4] H. Hult, F. Lindskog, O. Hammarlind, and C. J. Rehn, *Risk and Portfolio Analysis.* Springer New York, 2012.

[5] L. Yu, H. Chen, S. Wang, and K. K. Lai, "Evolving least squares support vector machines for stock market trend mining," *IEEE Transactions on Evolutionary Computation*, vol. 13, pp. 87–102, Feb 2009.

[6] T. V. Gestel, J. A. K. Suykens, D. emma Baestaens, A. Lambrechts, G. Lanckriet, B. Vandaele, B. D. Moor, and J. Vandewalle, "Financial time series prediction using least squares support vector machines within the evidence framework."

[7] A. Gupta and B. Dhingra, "Stock market prediction using hidden markov models," in *2012 Students Conference on Engineering and Systems*, pp. 1–4, March 2012.

[8] G. Batres-Estrada, "Deep learning for multivariate financial time series," Master's thesis, KTH Royal Institute of Technology, June 2015.

[9] Y. Li and W. Ma, "Applications of artificial neural networks in financial economics: A survey," in *Proceedings of the 2010 International Symposium on Computational Intelligence and Design - Volume 01*, ISCID '10, (Washington, DC, USA), pp. 211–214, IEEE Computer Society, 2010.

[10] J. B. Heaton, N. G. Polson, and J. H. Witte, "Deep learning in finance," *CoRR*, vol. abs/1602.06561, 2016.

[11] R. Xiong, E. P. Nichols, and Y. Shen, "Deep learning stock volatility with google domestic trends." `https://arxiv.org/abs/1512.04916`.

[12] A. Siripurapu, "Convolutional networks for stock trading." `http://cs231n.stanford.edu/reports/2015/pdfs/ashwin_final_paper.pdf`.

[13] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013.

[14] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[15] J. C. Hull, *Options, Futures, And Other Derivatives.* Pearson Education, 8 ed., 2011.

[16] E. F. Fama, "Efficient capital markets: A review of theory and empirical work," *Journal of Finance*, vol. 25, p. 383–417, 1970.

[17] B. G. Malkiel, "Reflections on the efficient market hypothesis: 30 years later," *The Financial Review*, vol. 40, no. 1, pp. 1–9, 2005.

[18] E. W. Saad, D. V. Prokhorov, and D. C. Wunsch, "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks," *IEEE Transactions on Neural Networks*, vol. 9, pp. 1456–1470, Nov 1998.

[19] N. Jegadeesh and S. Titman, "Returns to buying winners and selling losers: Implications for stock market efficiency," *Journal of Finance*, vol. 48, no. 1, pp. 65–91, 1993.

[20] N. Jegadeesh and S. Titman, "Profitability of momentum strategies: An evaluation of alternative explanations," Working Paper 7159, National Bureau of Economic Research, June 1999.

[21] M. Wahde, *Biologically Inspired Optimization Methods: An Introduction.* WIT Press, 2008.

[22] G. Cybenko, "Approximations by superpositions of sigmoidal functions," *Mathematics of Control, Signals, and Systems*, vol. 2, 1989.

[23] G. Cybenko, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, p. 251–257, 1991.

[24] H. G. Rumelhart, D. and R. Williams, "Learning representations byback-propagating errors.," *Nature*, vol. 323, p. 533–536, 1986.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[26] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, 2013.

[27] D. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," *Neural Networks*, vol. 16, no. 10, pp. 1429 – 1451, 2003.

[28] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, pp. 1–17, 1964.

[29] B. T. Polyak, "A method for unconstrained convex minimization problem with the rate of convergence o(1/k2)," *Doklady ANSSSR (translated as Soviet.Math.Docl.)*, vol. 269, pp. 543–547, 1983.

[30] I. Sutskever, *Training Recurrent Neural Networks.* PhD thesis, University of Toronto, 2013.

[31] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," *CoRR*, vol. abs/1212.0901, 2012.

[32] G. Hinton, "Neural networks for machine learning. lecture 6e.."

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[35] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Effiicient backprop," in *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, (London, UK, UK), pp. 9–50, Springer-Verlag, 1998.

[36] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012.

[37] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, Aug. 1996.

[38] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting.* Springer-Verlag New York, 2002.

[39] C. Bennett and M. A. Gil, "Measuring historical volatility."

[40] H. Markowitz, "Portfolio selection," *Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[41] "Stochastic oscillator." `http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:stochastic_oscillator_fast_slow_and_full`. Accessed: 2017-02-01.

[42] "Percentage price oscillator." `http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:price_oscillators_ppo`. Accessed: 2017-02-01.

[43] J. Wilder, *New Concepts in Technical Trading Systems.* Trend Research, 1978.

[44] "Williams %r." `http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:williams_r`. Accessed: 2017-02-01.

[45] "Commodity channel index (cci)." `http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:commodity_channel_index_cci`. Accessed: 2017-02-01.

[46] R. Fisher, *Statistical methods for research workers.* Edinburgh Oliver & Boyd, 1925.

[47] F. Mosteller and R. A. Fisher, "Questions and answers," *The American Statistician*, vol. 2, no. 5, pp. 30–31, 1948.

[48] "Theano framework." `http://deeplearning.net/software/theano/`. Accessed: 2017-02-01.

[49] C. W. J. Granger and O. Morgenstern, "Spectral analysis of new york stock market prices1," *Kyklos*, vol. 16, no. 1, pp. 1–27, 1963.

TRITA -MAT-E 2017:27
ISRN -KTH/MAT/E--17/27--SE