UiO : Department of Informatics
University of Oslo

Predicting Stock Markets with Neural Networks

A Comparative Study

Torkil Aamodt

Master's Thesis Spring 2015

Predicting Stock Markets with Neural Networks

Torkil Aamodt

May 4, 2015

Abstract

Correctly predicting price movements in stock markets carries obvious economical benefits. The task is traditionally solved by analyzing the underlying company, or the historical price development of the company's stock. A third option that is undergoing active research is to create a predictive model of the stock using machine learning. This thesis follows the latter approach, in which a machine learning algorithm is presented with historical stock data. The algorithm uses this information to train a model that is expected to infer future prices given recent price information.

Machine learning is a large field within computer science, and is under constant development. Breakthroughs in a family of machine learning models known as artificial neural networks have spiked an increased interest in these models, including applying them for financial prediction. With a plethora of models available, selecting between them is difficult, especially considering the constant flow of emerging models and learning techniques.

This study compares a selection of artificial neural networks when applied for stock market price prediction. The networks are selected to be relevant to the problem, and aim at covering recent advances in the field of artificial neural networks. The networks considered include: Feed forward neural networks, echo state networks, conditional restricted Boltzmann machines, time-delay neural networks and convolutional neural networks. These models are also compared to another type of machine learning algorithm, known as support vector machines. The models are trained on daily stock exchange data, to make short-term predictions for one day and two days ahead. Performance is evaluated in the context of following the models directly in a financial strategy, trading every prediction they make. Additional performance measures are also considered, to make the comparison as informed as possible.

Possibly due to the noisy nature of stock data, the results are slightly inconsistent between different data sets. If performance is averaged across data sets, the feed forward network generates most profit during the three year test period: 23.13% and 30.43% for single-step and double-step prediction, respectively. Convolutional networks get close to the feed forward in terms of profitability, but are found unreliable due to their unreasonable bias towards predicting positive price changes. The support vector machine delivered average profits of 17.28% for single-step and 11.30% for double-step. Low profits or large deviations were observed for the other models.

Preface

The efforts associated with writing this thesis have challenged me in several aspects. Finding my way through the jungle of neural network research and applying the models demanded my full attention. Furthermore, writing a thesis is an endeavor by itself. There are a number of people that have contributed to the completion of this thesis.

I would like to thank my supervisor at BEKK, Jørgen Braseth. Our discussions surrounding direction of the research, as well as his genuine comments regarding my progress, have kept me focused and motivated throughout the work. I would also like to thank Professor Jim Tørresen, my supervisor at the university. His feedback on the thesis and continuous support has been tremendously helpful to me during the course of my work.

Professor Herman Ruge Jervell initially functioned as my university supervisor, and I am grateful for his efforts and advices. A discussion with associate professor Dag Einar Sommervoll enriched my perspective on the task, which led to the inclusion of information regarding the day of week in the data samples.

Finally, I thank my family and friends, who have been nothing but supportive and understanding of my preoccupied state. I especially thank my sister Yngvild, who generously spent her own free time to read through and comment the entire thesis.

Contents

Acronyms

ANN Artificial Neural Network. 2–4, 11–16, 18–20, 39, 40, 53, 80, 83

ARIMA Autoregressive Integrated Moving Average. 4

BAC Bank of America Corporation. xiii, 40, 47–49, 61, 63, 64, 66, 68, 70, 72, 74, 77, 80, 83–85

BPTT Backpropagation Through Time. 24, 25, 54

CD Contrastive Divergence. 34, 36

CNN Convolutional Neural Network. viii, xiii, xv, 16, 23, 28, 29, 40, 41, 43, 56, 57, 72–81, 83, 84

convRBM Convolutional Restricted Boltzmann Machine. 41

CRBM Conditional Restricted Boltzmann Machine. viii, xiii, xv, 3, 4, 12, 16, 35, 40, 42–44, 55, 56, 68–70, 75–81, 83

DBN Deep Belief Network. viii, 4, 16, 36, 37, 80

ESN Echo State Network. vii, viii, xiii, xv, 4, 16, 25, 26, 40, 43, 44, 54, 55, 57, 66–68, 75–79, 81, 83

FFNN Feed Forward Neural Network. viii, xiii, xv, 3, 4, 13, 16, 18, 23, 24, 26–29, 32, 34, 40, 43, 54, 63–66, 75–81, 83, 84

GPU Graphics Processing Unit. 22

GSPC S&P 500. xiii, 40, 43, 46–48, 54, 61, 63, 64, 68, 70, 72, 74, 75, 77

HMM Hidden Markov Model. 3

IXIC NASDAQ Composite. xiii, 40, 46–48, 61, 63, 64, 68, 70, 72, 74, 77, 78

kNN k-Nearest Neighbors. 12–14

LSM Liquid State Machine. 25

MNIST Mixed National Institute of Standards and Technology database. 37

MSE Mean Squared Error. viii, 45, 60, 61, 63, 64, 66, 68, 70, 72, 74, 79

MSFT Microsoft Corporation. xiii, 40, 47–49, 59–74, 78, 84

N/A Not Applicable. 47

NASDAQ National Association of Securities Dealers Automated Quotations. 8, 46–48

NYSE New York Stock Exchange. 8, 46–48, 50

PBR Petróleo Brasileiro S.A. - Petrobras. xiii, 40, 47, 50, 53, 61, 63, 64, 66, 68, 70, 72, 74, 78, 80, 81, 83, 84

PoE Product of Experts. 34

RBM Restricted Boltzmann Machine. viii, xiii, 3, 12, 16, 32–36, 42, 52, 54, 55, 80

RNN Recurrent Neural Network. vii, 3, 4, 12, 16, 24, 25, 54

SVM Support Vector Machine. viii, 2–4, 11, 13, 14, 16, 29–32

SVR Support Vector Regression. viii, xiii, xv, 32, 40, 43, 44, 53, 54, 60–64, 66, 68, 74–84

TDNN Time-Delay Neural Network. vii, viii, xiii, xv, 3, 4, 16, 26, 27, 40, 43, 56, 70–72, 75–79, 81, 83

XOM Exxon Mobile Corporation. xiii, 40, 47, 50, 51, 61, 63, 64, 68, 70, 72, 74, 79, 84

XOR Exclusive OR. 2

List of Figures

List of Tables

Chapter 1

Introduction

This chapter puts the study in context. The motivation behind the research is given in section 1.1, before the problem itself is described in section 1.2. Similar or related research is presented in section 1.3. Lastly, section 1.4 outlines the general structure of the thesis.

1.1 Motivation

Predicting the development of financial instruments like stocks carries obvious economical benefits, and there are countless strategies that attempt to achieve this. Lately, an effort has been put into using machine learning techniques to model stock prices. The models have shown mixed success, and it can be difficult to compare the techniques when they are applied for different stocks and evaluated using different metrics. Considering the recent renaissance for artificial neural networks, a comparative evaluation of these models is timely.

1.1.1 Stock Markets

Stock markets are hard to predict. Driven by supply and demand, macroeconomic changes such as inflation or political instability can affect whole markets, while local events like company financial announcements or product releases impact individual stocks. Traders also look for signals in the price development that may indicate future prices. Because some trading is based on these indicators, price movement by itself can also trigger trading activity, causing further price adjustments. Investors and traders should ideally consider all these factors when evaluating a stock, however market participants are often partitioned into two groups: Traders that make decisions based on news, facts and numbers are known as fundamental analysts, while those who look for signals in price history are using technical analysis.

Both camps of traders are exploring how to utilize computers to efficiently find trading opportunities. Formulating the problem for a computer using the fundamental approach is challenging, because of difficult input like news written in natural language. The approach taken in [45] uses natural language models together with a technique called machine learning to infer how a given news story will affect a related stock in the immediate future.

Technical analysis on the other hand, uses only the trading history of a given stock, otherwise known as a time series. In statistics, time series prediction is a well-known problem, accompanied by numerous mathematical models. As a result of an increased interest surrounding machine learning however, other kinds of models have also been applied in the context of technical analysis. Early attempts include [52], and more recently [27]. From a research point of view, predicting stock prices in this manner is still a relatively new concept, and recent advances in machine learning leave much to be explored.

The methodology followed in this thesis resembles technical analysis, in the sense that predictions are based solely on past trade information. Technical analysis traditionally relies on analyzing trades using a range of mathematical functions and visual inspection of the data. In this study however, such analysis is implicitly handled by a computer using machine learning.

1.1.2 Machine Learning

The field of artificial intelligence has set the ambitious goal of making machines either seemingly or genuinely intelligent. As stated in [43], the field covers subjects like natural language processing, reasoning, learning, visual perception and physical movement, to name a few. The sub-field of artificial intelligence known as machine learning attempts to make computers learn from observations. While statistical time series models are specialized for their task, machine learning algorithms are general tools that can be fitted to a vast number of problems, including stock price prediction. This study is centered on a family of models referred to as Artificial Neural Networks (ANNs).

History

ANNs are not a new concept, and can be traced back to 1957, in [41]. The initial versions were problematic, and [35] showed that simple networks failed to model basic functions like the Exclusive OR (XOR) operation. If the ANNs were constructed with several layers working together in a deeper network, non-linear problems like the XOR operation could be modeled. [42] proposed using the backpropagation algorithm to train ANNs, which was also applicable for deep networks. It was not without faults however, and deeper models learned less efficiently.

In 1992 another machine learning model called the Support Vector Machine (SVM) was extended to allow solving non-linear problems, [2]. The SVM gained significant attention, while the ANN struggled to compete.

Today, the issue of training deep ANNs has largely been eliminated due to breakthroughs in learning procedures like [17]. The progress has sparked new interest in these algorithms, and the family of ANN models keeps expanding, making it increasingly challenging to pick which one to apply.

Model Variety

Machine learning models are often differentiated by whether it learns with supervision or completely by itself. Supervised learning operates by supplying a model with input, along with what they are expected to lead to, referred to as ideal output. The learning algorithm then adjusts the model to reproduce the ideal values on its own. Unsupervised models on the other hand learn from the input data alone, and are expected to output the same kind of values for input that is similar. Supervised models include SVMs, Feed Forward Neural Networks (FFNNs) and Hidden Markov Models (HMMs). Unsupervised algorithms are found in k-means clustering, Restricted Boltzmann Machines (RBMs) and auto-encoders.

For time series prediction the presence or absence of supervision is usually not a deciding factor, as both kinds may be used. A more suitable way of discriminating between models in that context is by considering how models connect to the data: Temporally or spatially. In a time series prediction problem, the changing of a variable through time is considered a temporal development. If multiple variables are modeled, any connections between them within the same time step are spatial. It is possible to regard a segment of the time series as a single, spatial input, meaning spatial models can also fit temporal data. However, no explicit understanding of temporal relationships is present in these models.

A lot of effort has been put into learning spatial models, and most of the previously mentioned models are spatially inclined by default. HMMs capture temporal dependencies, but they work with discrete states and have a limited representational ability, as pointed out in [49]. Certain variations of the ANN allow for modeling temporal relationships. Such models include Recurrent Neural Networks (RNNs), Conditional Restricted Boltzmann Machines (CRBMs) and Time-Delay Neural Networks (TDNNs).

When working with ANNs, finding an optimal architecture is part of the challenge. Several recent temporal ANN models could provide a good starting point. Most networks may additionally be configured as deep networks, presenting even more architectural freedom. Experimentation should provide clues as to how the modern neural network performs for non-linear time series prediction, more specifically stock price forecasting.

1.2 Problem Formulation

This study compares a selection of different neural networks, when applied for short-term, daily stock price prediction. Each model is evaluated in the context of following a financial strategy that trades every prediction of the model.

Five neural networks are compared to each other, a baseline represented by an SVM and the natural development of each stock. The SVM was chosen due to its history of outperforming ANNs. It also trains deterministically, meaning there is no uncertainty surrounding its results. The neural networks are selected such that they either were created explicitly to model time series, or have shown good results for other applications.

Models are trained independently across four stocks and two indices to make the results statistically relevant. Indices are aggregations of multiple stocks, and make a better representation of the market as a whole. They are commonly used for benchmarking financial models, and are therefore also included in this study.

1.3 Related Work

In the context of predicting time series using neural networks, there is much research available. [51] used a TDNN to recognize phonemes in speech. In [24], Echo State Networks (ESNs) were introduced and evaluated on several time series. [49, 50] leveraged CRBMs to model human motion capture data.

When it comes to modeling financial markets in particular, some research revolves around evaluating a single model for prediction. [27] presents a hybrid model using an Autoregressive Integrated Moving Average (ARIMA) and an ANN. The model was evaluated using several data sets, including one for exchange rates. Lower error rates were achieved for the hybrid solution, compared to either model alone. [3] finds that forecasting exchange rates using Deep Belief Networks (DBNs) outperforms regular FFNNs.

Rather than researching one model, comparative studies are devoted to evaluating multiple types of models. [44] considers TDNNs, RNNs and probabilistic neural networks, and finds that they all deliver reasonable performance when predicting stock trends. [29] regards a range of single, independent studies in order to theoretically evaluate ANNs in general as a tool for financial forecasting.

1.4 Outline

Following this introduction, the rest of the thesis is structured into five chapters: Background, methodology, experiments, conclusion and future work.

1.4.1 Background

The background chapter covers relevant techniques and concepts that the experiment revolves around. Topics include introductions to stock markets, machine learning and neural networks in general. Specific neural network architectures are also discussed, with each model being listed under either the supervised or unsupervised section.

1.4.2 Methodology

Practical notes regarding what was done, in addition to how and why, are detailed in the methodology chapter. This includes a description of the project implementation and its functions, what libraries are used and how results are collected. Data selection is also discussed, along with how data are preprocessed. Finally, the candidate models are presented along with their respective hyper-parameters.

1.4.3 Experiments

Obtained results are presented and discussed in chapter 4. An explanation of how to read the results is given in section 4.1. Section 4.2 considers each model separately, stating their performance as well as comparing them to a baseline model. Lastly, section 4.3 analyzes the results and compares the models to each other.

1.4.4 Conclusion

Chapter 5 concludes the thesis, emphasizing notable findings in light of the introductory problem statement.

1.4.5 Future Work

Chapter 6 highlights relevant topics that could be considered for further research.

Chapter 2

Background

This chapter describes models and research relevant to the study in this thesis. Sections 2.1 and 2.2 provide introductory material to stock markets and machine learning in general, while section 2.3 describes what neural networks are and how they work. Sections 2.4 and 2.5 detail specific models and techniques for supervised and unsupervised learning, respectively.

2.1 Stock Markets

[8] defines a market as an area or arena in which commercial dealings are conducted. *Financial markets* can be described as aggregations of buyers and sellers involved in trading a financial instrument like stocks, bonds, commodities or currencies. A *stock market* is a financial market where company shares are traded. While the stock market is an abstract term, the actual trades may be executed over the counter, at a stock exchange, an electronic communication network or similar. Trades that are executed through stock exchanges are easily tracked, because of the centralized and transparent nature of the exchange. This makes stock markets more approachable for studying, unlike for instance foreign exchange markets where data has to be aggregated from multiple decentralized sources.

2.1.1 Stock Exchanges

A *stock exchange* is a common hub for buyers and sellers to find each other and fulfill trades. [9] defines it as a market where securities are bought and sold, although note that the word *market* in this setting refers to a physical place, unlike stock markets. Activity at the exchange is visible to other participants, which in turn will drive the price. It is the principle of supply and demand, put into action.

A stock is regarded as *liquid*, if it sees significant amounts of trading activity. The more activity, the easier it is to find a buyer when someone tries to sell, and vice versa. For less liquid stocks some participants might struggle to complete trades. If the participant compromises on a less favorable price, chances of fulfilling the trade increase. When settling at another price point than intended, the price difference is known as *slippage*.

By only considering liquid stocks, trades are usually filled almost instantly, provided normal market conditions.

Because of the centralized, transparent nature of stock markets, gathering historical data from them is both easily and widely done. Some traders and analysts use these data in order to model future price movements. This is known as *technical analysis*, because it relies exclusively on the price development.[1]

Numerous stock exchanges exist today, each of them listing a different selection of stocks. Two well-known American exchanges are the New York Stock Exchange (NYSE) and the National Association of Securities Dealers Automated Quotations (NASDAQ). Except for both being based in New York, there are various differences related to the kind of companies they list, and how they execute trades. The NASDAQ is known to list companies related to technology, and runs algorithms to automate the trading process. The NYSE on the other hand, runs a more traditional auction-based trading. The differences between the NASDAQ and the NYSE are discussed further in [22].

2.1.2 Market Positions

From the moment someone buys shares in a company, they have *entered* the market and are exposed to price changes. If the stock value goes up, then they have earned a *profit* equal the total price difference. Profits are realized when the trader decides to sell the shares back at the higher price. In this case, we say the trader entered the market in a *long* position, because the trader bought shares expecting the price to rise.

It is also possible to realize profits in declining markets, although the procedure is slightly more convoluted: A trader expecting a stock to fall, can borrow shares from a third party, like a stock broker. The trader would now immediately sell these shares on the market. If the price falls at a later point, the trader can then buy back the same amount of shares, at a lower price, and repay them to the third party. Again, the difference in price at the time of loaning and returning the shares makes the profit. Traders that borrow shares expecting them to fall in value are said to enter the market in a *short* position.

2.1.3 Indices

An *index* is a collection of stocks, which value is based on the underlying share prices. Exactly how much a given stock affects the index price varies, but the calculation usually involves the number of shares and their price.

Since stock indices represent the aggregation of multiple stocks, they are often consulted to sample the stock market as a whole. For example, consider a publicly listed company and an index from the same market. If

[1]The counterpart to technical analysis is called fundamental analysis, and uses fundamental facts and numbers regarding the business to make a decision.

both the index and stock fall in value, the decline is likely caused indirectly by something outside of the company. Similarly, the index would react mildly if only one of its stocks fell in value. In other words, peculiarities local to individual stocks are rendered insignificant in the index, and the general market value is retained.

Being a list of stocks, indices technically can not be invested in directly. The simple solution is to manually buy shares in the underlying companies, based on the weighting scheme of the index. This is expensive, time consuming and requires manual adjustments according to changes in the index. Indices can still be bought indirectly, through index funds and exchange-traded funds.

2.1.4 Stock Market Data

Stock market data are available in various forms, ranging from fine-grained information concerning each trade, to one *data point* every month. Exactly what the data points contain may vary, but a widely used composition consists of six variables: Time, open, high, low, close and volume.

Time is a reference to when the data point is from. *Open* and *close* is the share price at the beginning and end of the period, while *high* and *low* refers to the highest and lowest point the value reached throughout the period. Lastly, *volume* is the number of shares that were traded in total.

Deciding the time span of each data point plays an important role; long periods prevent making short-term forecasts, while short periods are noisier due to their detailed nature.

For clarity, this report will sometimes refer to data points as stock prices, even though more information is actually contained in each point. Furthermore, although it is possible to plot all the variables except volume in what is known as a candlestick chart, I will plot only closing prices, using line charts. This is for the sake of brevity and clarity, but be aware that there are more variables in the actual data.

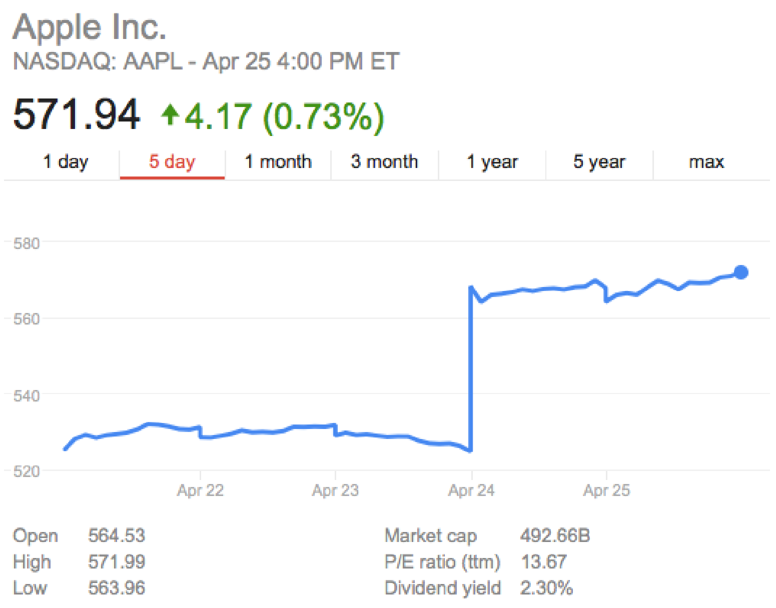
Continuity

Representing share price by a real value with a fixed number of decimal places is common, even though the exact representation varies between markets. By this definition we are no longer dealing with continuous, real values, but rather discrete, countable values.

For practical purposes however, we may regard stocks as continuous. This is possible because we can still compare a real number to a share price; the real number 1/3 is between discrete values 0.33 and 0.34.

Market Gaps

Some stock exchanges allow certain participants to trade after closing hours. *After-hours* trades also impact the share value, but the price will not necessarily be reflected until the next day for the average trader. This delay can result in a discontinuous jump in the price charts (figure 2.1),


Apple Inc.
NASDAQ: AAPL - Apr 25 4:00 PM ET
571.94 ↑4.17 (0.73%)

Open	564.53	Market cap	492.66B
High	571.99	P/E ratio (ttm)	13.67
Low	563.96	Dividend yield	2.30%

Figure 2.1: After-hours trades cause a jump discontinuity. Screenshot taken from [14].

which often is hard to predict. Example: If a company publicly announces its earnings after the exchange has closed, the market is likely to *gap* the next workday.

Large jump discontinuities are also observed when a company decides to *split* its shares; the value of each share is halved and the number of shares is doubled. Stock splitting is not a frequent phenomenon, and stock data are easily adjusted to remove the gap.

Changing Trends

Looking into the history of a stock may provide clues to the development of future price points, but there certainly are occasions where this is not enough. Ranging from macroeconomic changes and natural disasters to management illness and product releases, countless factors can contribute to explaining price changes.

How such events impact a stock varies in terms of duration and magnitude. Investors often refer to markets that are expected to, or currently are rising as *bull* markets. *Bear* markets on the other hand, are declining markets. A model that has been fitted solely to bull market data, will probably struggle under bear market conditions.

Bull and bear are broad terms, describing an increasing or decreasing pattern, respectively. In addition to the few patterns we have given names, it is not unthinkable there are many and more subtle patterns hidden in the data. Due to the influence of external factors and market conditions, these patterns are also likely to change through time.

Noise

Depending on the strategies and goals of the investor, trading shares in a way that breaks with the current trend might seem rational: Some strategies are intended to oppose trends at the right moment, in order to enable trading at optimal prices. Investors may also buy shares at a seemingly bad time, with expectations of long-term profits.

Regardless of how profitable a trade turns out to be, it will affect the share price. This means that even under bull conditions, there are times when the value goes down, and vice versa. When looking at market data in retrospect, we see this as *noise*.

2.2 Machine Learning

Programming computers traditionally involves writing code specific to the domain of the problem in question. Machine learning takes an alternative approach, by not being tailored for any particular set of problems: Rather than hard-coding rules based on our knowledge, we let the computer learn a model that fits *samples* of the data.

2.2.1 Sample Data

A prerequisite to performing machine learning is to have available data to learn from. This differs from an approach where a domain expert is hired to hand-craft a system based on prior experience. Because learned models are derived from data, it is important that the data are suitable.

There should be a correlation between the data, or a transformation of it, and the outcomes we wish to learn. Many machine learning techniques rely on transforming the data in order to separate samples more easily. The kernel trick used in Support Vector Machines (SVMs) and features learned in Artificial Neural Networks (ANNs) are examples of this. If the data consist purely of noise, any learned rule is unlikely to hold in the future. Identical input vectors that lead to different output also suffer by the problem of inconsistency, and are difficult to learn from.

Most of the time models are trained on *samples* of the data, either because of scarce availability or scaling issues. In order to train a model that generalizes well enough to correctly handle unseen data, the data samples must make a decent representation of what could come. For example, it would be hard for a model to classify a penguin as a bird, if all the birds in the training data were seagulls; the birds used for learning all fly, while the flightless penguin swims.

Once samples of the data are gathered, it is common practice to partition them into three sets, namely for training, validation and testing. The *training set* contains samples to which a *learning algorithm* fits the model. If the model is fitted very precisely, it can provoke a phenomenon known as *overfitting*. Overfitted models tend to be inaccurate in use, an issue further discussed in section 2.2.8. To prevent overfitting, performance can be tested against a *validation set* at regular intervals; once training stops improving validation performance, it is a sign the model might be overfitting. This is known as *cross-validation*, and is commonly practiced. To evaluate the final model, a separate *test set* is used. All vectors in the test set are *out-of-sample*, or *unseen*, meaning the model has not learned from these samples. Test set performance therefore indicates real-world performance. The size ratio between different sets varies, but the training set usually consists of at least 50% of the data.

The format of the data might be restricted by the learning algorithm. Two common types of learning, supervised and unsupervised, make different demands as to how each data sample is presented during training.

2.2.2 Supervised Learning

Supervised learning requires each element in the training set to contain *input vectors* or *samples* in addition to the expected *ideal output* they lead to. The machine learning algorithm must then adjust the model so that it fits the data.

Some supervised machine learning algorithms do not need any training at all; the k-Nearest Neighbors (kNN) algorithm for instance is a spatial model that keeps the training data in memory and infers from them directly, meaning no explicit learning takes place. Algorithms like the ANN on the other hand are slow learners, requiring running for many iterations over the training set before settling on its parameters.

2.2.3 Unsupervised Learning

In some cases it is not clear what we wish the input vectors to lead to, but we would still like a certain grouping of the data. Consider for instance a collection of music tracks; if we wanted to partition them into groups or clusters of similar musical style, we could do so without having to explicitly tell the algorithm what the styles are. *Unsupervised learning* takes unlabeled input vectors, and outputs the same kind of response for similar samples.

Restricted Boltzmann Machines (RBMs) are unsupervised variations of the ANN, and identify *features* that are descriptive of different input. These features are typically represented by binary values, implying whether the feature is present in the input vector. The semantic meaning behind each feature is not given. Unsupervised learning and models such as the RBM are responsible for breakthroughs in deep learning, where it can be used in conjunction with supervised training. This is discussed further in sections 2.3.4 and 2.5.3.

2.2.4 Connectivity

Some machine learning models are designed to better model certain types of data. In modeling time series for instance, Recurrent Neural Networks (RNNs) and Conditional Restricted Boltzmann Machines (CRBMs) explicitly capture *temporal* relations between variables at different time steps.

Such models will be referred to as temporally connected models. Models that consider each input vector as an isolated case without temporal connections are described as having *spatial* connectivity. Fully connected Feed Forward Neural Networks (FFNNs) and SVMs are examples of spatial models.

It should be noted that most temporal models also connect spatially to other variables within the same time step, and are therefore correctly referred to as *spatio-temporal*. All temporally connected models investigated in this thesis are also spatio-temporal models, but will be referred to as temporally connected for simplicity. Similarly, spatial models may also be used to model time series, if values from several consecutive time steps are contained in each input vector.

2.2.5 Output Semantics and Problem Space

Exactly what the output of a machine learning algorithm represents, varies between models. The kNN algorithm produces a natural number, while ANNs typically output one or more real values depending on the activation function, see section 2.3. The problem at hand largely dictates which kind of model to apply.

Classification attempts to assign the input data to a certain class, like the image of a car to a class associated with vehicles. A classifier like kNN is made for these scenarios, and will return a discrete value associated with the class it finds more likely.

In contrast to classification, *regression* outputs continuous values. For a system that predicts rainfall activity, we would likely wish for greater accuracy than what is practically achievable through discrete values. Furthermore, there is no theoretical upper limit to the amount of rain. Since the number of classes limits the output range of classifiers, we need to use a regression model. ANNs are capable of solving regression problems by outputting floating-point numbers.

2.2.6 Hyper-parameters

Also known as meta-parameters, *hyper-parameters* compose a set of variables that define how the machine learning algorithm operates. They differ from regular *parameters*, which are the variables the learning algorithm aims to optimize.[2] Because of the variety between algorithms, machine learning techniques have different hyper-parameters. Usually, a mechanism to control overfitting is offered: The hyper-parameter C in SVMs, defines how much misclassification of training samples penalizes the solution, see section 2.4.5. ANNs can similarly control overfitting by tuning the *learning rate* and number of *epochs*, which define how fast the network will adjust its parameters and for how long, respectively. Learning rate, epochs and other general ANN hyper-parameters are discussed in section 2.3.3.

[2]Machine learning algorithms that do not use parameters also exist. Examples of *non-parametric* models include the kNN algorithm.

2.2.7 Separability

As discussed in section 2.2.1, a machine learning model attempts to separate dissimilar input vectors. Shown in figure 2.2 are examples of two two-dimensional data sets each containing two classes represented by pluses and circles.

Linear Separability

In figure 2.2a the two classes are clearly contained in two separate colonies. By drawing a straight line we separate the classes. Once new points are presented, we simply see on which side of the line or *decision boundary* they fall, and assign them to the corresponding class. Examples of such binary, linear classifiers include basic SVMs and single-layered ANNs.

Non-linear Separability

Figure 2.2b presents a data set that is inseparable by any linear decision boundary. In order to correctly classify all the points, the decision boundary must be non-linear. The kNN algorithm, SVMs using the kernel trick and multi-layered ANNs are capable of handling non-linearity.

Even though a problem is non-linearly separable, it does not mean that any non-linear classifier will do. Algorithms like unsupervised k-means clustering will struggle in landscapes where one class engulfs another; since both classes have similar means, a new point belonging to the inner class might be assigned to the outer. ANNs are more moldable in this regard, and are restricted by the number of neurons and layers.

Separability in Regression

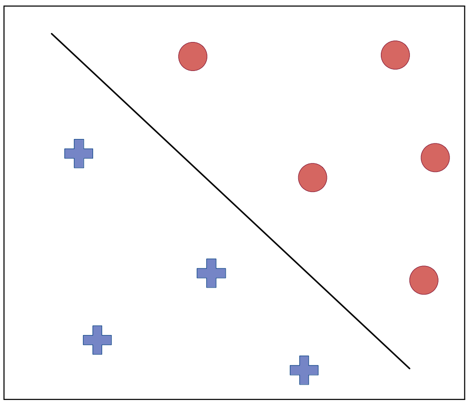
Although we are not directly interested in separability for regression problems, linearity still matters. A linear model has to make its prediction based on linear relationships between input and weights, while non-linear models do not have this restriction.

2.2.8 Data Fitting

Figure 2.3 illustrates decision boundaries for two different non-linear classifiers on the same data set. Is it possible to tell which one is better?

Underfitting

Continuing the bird analogy, we assume circles represent birds and pluses fish. The classifier in figure 2.3a has clearly not adapted to the data very closely, as two birds have been classified as fish, and vice versa. While it might seem hopeless at first, this case of underfitting is not necessarily all bad; in some cases it can help smooth out noise coming from outliers, making a more robust model. Assuming the misclassified birds are penguins, their inability to fly and swimming habits certainly


(a) A linear decision boundary.

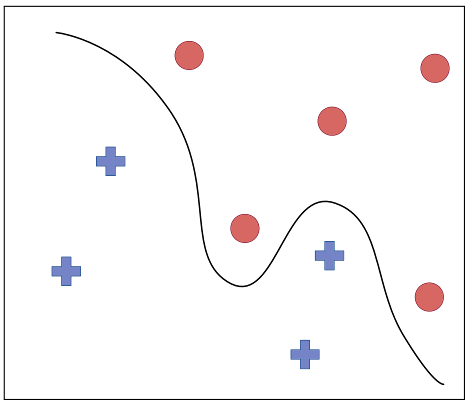

(b) Non-linear classifiers can learn very precise decision boundaries.

Figure 2.2: Decision boundaries for two-dimensional data sets. Shape of point denotes class.

makes them outliers among other birds. Classifying other samples with similar properties as birds is likely to turn out wrong. Allowing a few misclassifications can therefore be an acceptable trade-off.

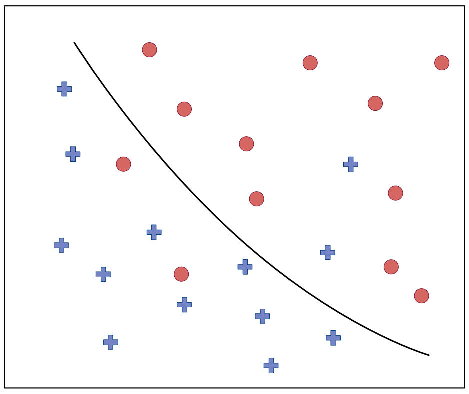
Overfitting

Figure 2.3b shows how an overfitted model responds to the same data set. All points are positioned correctly with respect to the decision boundary, however by looking at it visually it seems slightly unlikely and potentially undesirable; by fitting the data this closely we also adapt to any noise that is present. Using the bird analogy, penguins might arguably be regarded as noise, at least if it is impossible to separate a penguin from certain fish, based on the data alone. Out-of-sample performance in these situations is typically worse, because any correlations found in the noise is by definition unlikely to repeat outside the training samples.

If the training set is large enough and provides a good representation of the underlying source without introducing noise, the problem of overfitting is reduced.

2.3 Neural Networks

Biologist Emerson M. Pugh once said, “If the human brain were so simple that we could understand it, we would be so simple that we couldn’t” [39]. Exactly how the brain works remains unknown, but neuroscientists continuously attempt understanding more of it. As explained in [43, p. 738], the basic premise is that mental activity consists primarily of electrochemical activity in networks of brain cells called neurons. Computer scientists have adapted this understanding of the brain, in order to create the Artificial Neural Network (ANN).


(a) This classifier has not molded its decision boundary very close to the data, which may result in underfitting.

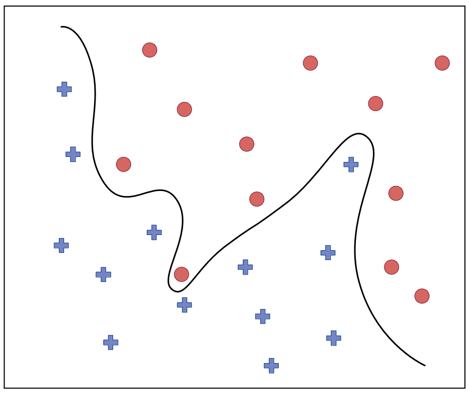

(b) The decision boundary separates every training sample correctly, typically resulting in overfitting.

Figure 2.3: Two classifiers trained on the same data set.

Model	Learning	Connectivity
Feed Forward Neural Network	Supervised	Spatial
Recurrent Neural Network	Supervised	Temporal
Echo State Network	Supervised	Temporal
Time-Delay Neural Network	Supervised	Temporal
Convolutional Neural Network	Supervised	Temporal
Support Vector Machine	Supervised	Spatial
Restricted Boltzmann Machine	Unsupervised	Spatial
Deep Belief Network	Unsupervised	Spatial
Conditional Restricted Boltzmann Machine	Unsupervised	Temporal

Table 2.1: Characteristics of several relevant ANNs. The SVM is also included for comparison.

Multiple types of ANNs are discussed in the background chapter, and a general overview of them is listed in table 2.1. As discussed in section 2.2.4, the temporal models are strictly speaking spatio-temporal.

2.3.1 Biological Inspiration

Figure 2.4 illustrates how a *biological neuron* fits together. In [30] our current understanding of the neuron is explained. A biological neuron receives input signals through its dendrites. These signals are processed in the soma, which in turn adjusts the output frequency accordingly. A neuron always outputs electric spikes through the axon, and the neuron activity is measured by the frequency of these signals.

In a biological brain, these neurons connect to each other to form a network; the axon has a number of terminals that form synapses with with dendrites of other neurons. These synapses allow for propagating signals through a network neurons.

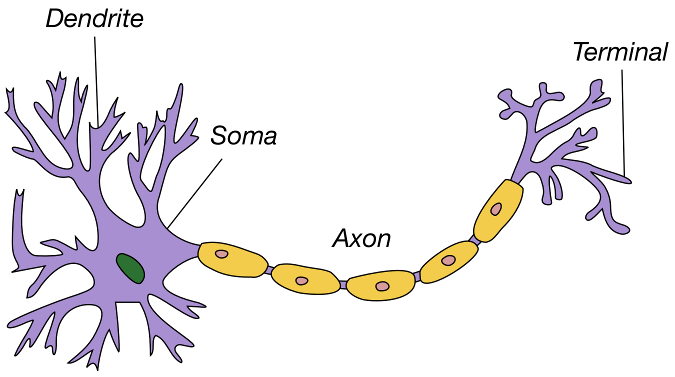


Figure 2.4: Anatomy of a biological neuron. The illustration is a subtle adaption of [25].

2.3.2 Artificial Adaption

Similar to biological neural networks, the ANN consists of connected artificial *neurons*, also known as *units* or *nodes*. Each node has a number of input channels and output channels. A node's sum of all input is called its *pre-activation*. An output or *activation* is produced by the *activation function*, using the pre-activation as its only argument. Figure 2.5a conceptually illustrates an artificial neuron.

Activation Function

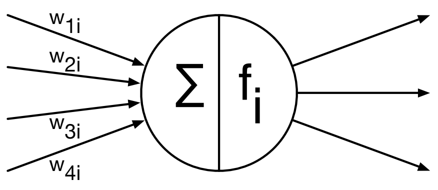
There are multiple activation functions to choose from. Threshold functions that return zero or one based on whether the input is less or greater than a certain limit, were used as activation functions early on. Also known as *perceptrons*, these networks were first introduced in [41].

Among the most popular activation functions, we find the *sigmoid functions*. Known for their characteristic s-shape, sigmoid functions are continuous and compress their output within a range. They are monotonically increasing, and can in some cases be described as a smooth variation of the threshold function. The *logistic function* is a widely used sigmoid function, which outputs within the range $(0, 1)$. The *black lines* in figure 2.6 plot the function.

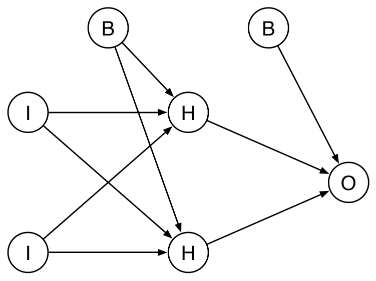
If the network models a classification problem, the use of the more computationally expensive softmax function for the output layer may be considered. This function ensures that all output sum to one, producing values that resemble probabilities.

Network Dynamics

As depicted in figure 2.5b, a basic artificial neural network is a weighted directed graph. Nodes represent neurons, and are arranged in *layers*. The network contains an input layer and an output layer, but in between there can be any number of *hidden layers*. Units within hidden layers are called



(a) An artificial neuron. $w_{1i}...w_{4i}$ are weights from node 1...4 to i and f_i is the neuron's activation function.



(b) A simple ANN architecture. I, H, B and O represents input, hidden, bias and output nodes, respectively. Weights not shown.

Figure 2.5: Artificial adaption of the neural network.

hidden units, because they represent features that are hidden in the data. By contrast, input and output units represent actual data points or classes. *Bias nodes* output a constant value, thereby determining the default state of connected neurons. This particular architecture is known as an Feed Forward Neural Network (FFNN), because all connections are directed towards the output layer.

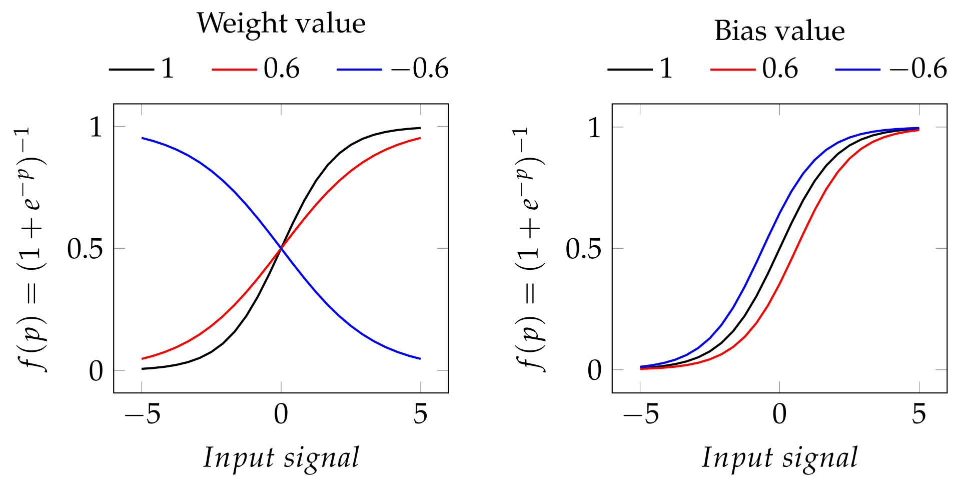
Note that ANNs are often conceptualized as a vertical *stack* of layers. Figure 2.5b and certain other figures in this thesis illustrate networks with their layers distributed left to right, in order to improve space utilization. References to the *top* layer therefore might translate to the *rightmost* layer in a figure, and equivalently the bottom layer may refer to the leftmost layer a figure. Additionally, references to the next or previous layer correspond to the layer above or below, respectively.

While a node will distribute the result of the activation function on all output channels, the receiving neurons will read the signal differently; each connection has an associated weight by which the signals are multiplied. These weights allow a learning algorithm to determine how different input units will influence its outcome. Specifically, pre-activations are calculated like so:

$$p_j = \begin{cases} x_j, & \text{input neurons} \\ \sum_i p_i w_{ij} + b_j, & \text{otherwise} \end{cases} \tag{2.1}$$

where w_{ij} is the weighted connection from neurons i to j and b_j is the bias. Note that the input layer is a special case that simply uses the input vector as pre-activations, denoted by x_j.

Weights Exactly how weights influence the outcome depends on the activation function. As seen in equation (2.1), weights work as coefficients regulating the size of the pre-activation. There are different weights between different pairs of neurons, but for the purpose of illustration we only consider a single neuron with one input channel and bias. To



(a) Weights change the steepness and direction of the activation function.

(b) Bias shifts the activation function, favoring certain regions of the output.

Figure 2.6: Influencing the logistic activation function by altering weights and bias.

get a better understanding, the commonly used logistic function may be regarded as an example.

Assuming a constant bias of zero, figure 2.6a illustrates that different weight values affect the behavior of the activation function. The black curve represents the standard logistic function, which appears when the weight is 1. In other words, the input is left unchanged. Setting the weight to 0.6 means every input is reduced. As shown by the red curve, the steepness of the activation function is thus decreased, and more extreme input are needed to produce *saturated* activations. An activation is regarded as saturated, when it is close to the output bounds of the function. Lastly, the blue curve is an inverted version of the red curve, due to a weight of -0.6.

Bias Figure 2.6b shows the effects of different bias values, using a fixed input weight at one. The black curve uses a bias of zero. Setting the bias to 0.6, pre-activations are shifted by 0.6, as seen in equation (2.1). This also shifts the activation, as shown by the red curve. Similarly, the blue curve uses a bias of -0.6 and is shifted in the other direction. Intuitively, the bias determines where the default activation lies, as an input of zero will give different activations depending on whether the black, red or blue curve is followed. Weights, together with bias values, define the *parameters* of an ANN.

Function Approximation

ANNs are often described as function approximators, following the results of [5, 19] which show that ANNs can approximate any continuous function arbitrarily well, given enough hidden units.

2.3.3 Backpropagation

Although ANNs are technically capable of modeling a wide range of problems, learning a sensible set of weights still represents an active field of research. Most supervised ANNs today learn by a technique known as backpropagation, or some variation of it.

Backpropagation is a form of *stochastic gradient descent*. The gradient is stochastic, in the sense that the parameters or weights are initialized randomly. Descending the gradient refers to reducing the error of each node. What this intuitively means is that we have a population of random weights, which we gradually tune towards being less erroneous. Backpropagation runs in *iterations*, traditionally one for every training sample. Once the entire training set is traversed, the algorithm has trained for one *epoch*. The parameters are adjusted until some stopping condition is met, which usually is based on the global error of the network or the number of epochs.

Algorithm

Being presented with a sample from the training set, backpropagation first runs it through the network in order to get an actual output. The algorithm is supervised, meaning we already know the ideal output. By comparing actual and ideal output, we get the output error: $e_i = \hat{y}_i - y_i$, where i is the output neuron index. As illustrated in figure 2.7a, this forward pass will necessarily also calculate pre-activations and activations throughout the network, denoted p_i and o_i respectively.

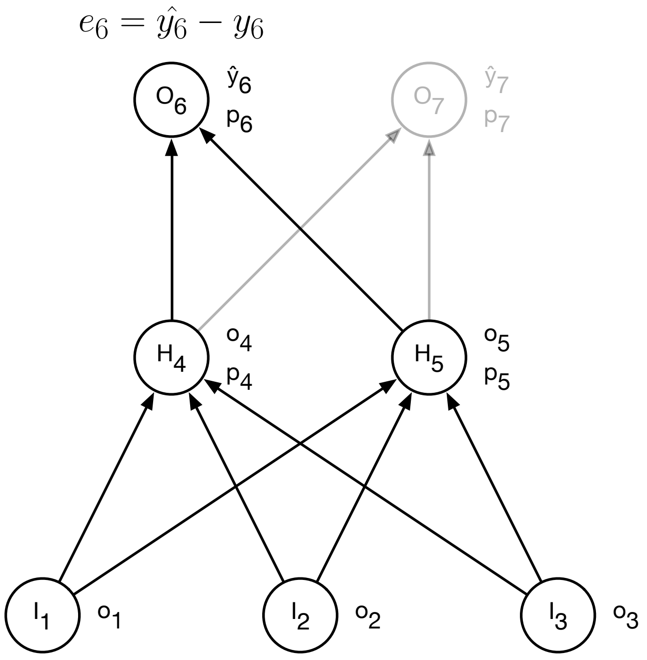
Once the output error is obtained, we can start propagating the error through the network in reverse, hence the name backpropagation. Intuitively, we calculate how much each node contributed to the error in the layer above it, and adjust the associated weights accordingly. Contribution towards error is measured by the steepness of the error gradient. It is calculated by finding the partial derivative of the error function, with respect to a weight w_{ij}:

$$\frac{\partial E}{\partial w_{ij}} = o_i \delta_j \tag{2.2}$$

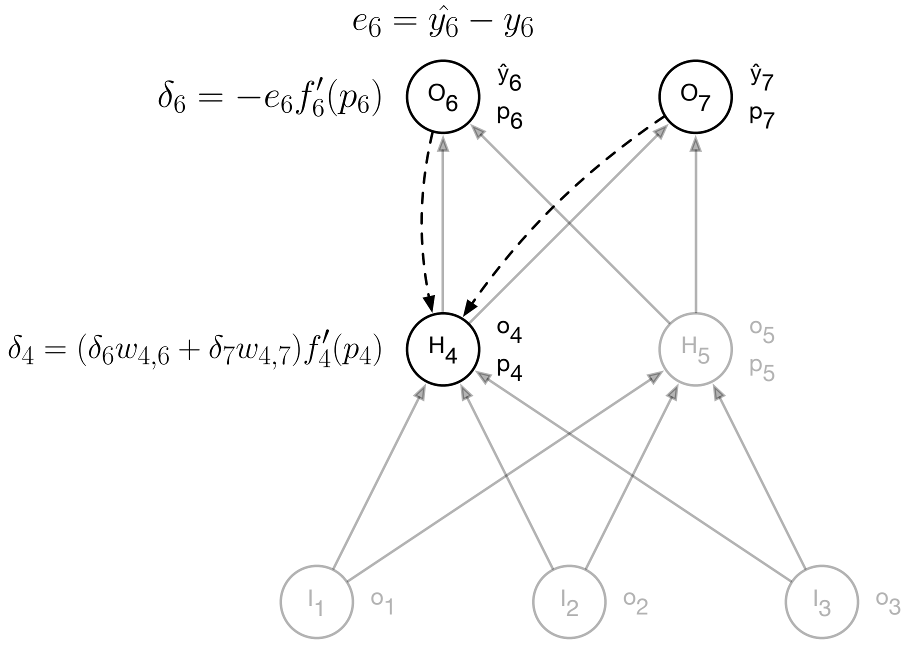
where δ_j represents the delta of neuron j in a given layer, and o_i is the activation of neuron i in the layer below.

With all the activations already calculated during the forward pass, what remain to solve equation (2.2) are the node deltas δ. As specified in equation (2.3), node deltas are calculated differently depending on the neuron type:

$$\delta_i = \begin{cases} -e_i f_i'(p_i), & \text{output neurons} \\ f_i'(p_i) \sum_j w_{ij}\delta_j, & \text{otherwise} \end{cases} \tag{2.3}$$



(a) A forward pass is done to calculate output error.



(b) Node deltas are propagated through the network in reverse.

Figure 2.7: The backpropagation algorithm.

where f_i is the neuron's activation function and p_i is its pre-activation as defined by equation (2.1). An illustration is given in figure 2.7b.

The node deltas can now be used to calculate the error gradient for each neuron, following equation (2.2). All that remains is then to update the weights, based on the error gradients:

$$\Delta w_{ij} = \eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w'_{ij} \tag{2.4}$$

where η is the *learning rate*, α the *momentum* and $\Delta w'_{ij}$ represents the weight delta from the previous iteration (set to zero for the first iteration). The learning rate hyper-parameter controls how much we let each training example influence the weight change. Momentum allows weights to change faster, given the change proceeds in the same direction through iterations.

Parallelization

As described here, backpropagation runs one iteration for every training sample, resulting in just as many weight updates. Learning can be greatly sped up by parallelizing, for example using a Graphics Processing Unit (GPU). In order to run iterations of backpropagation in parallel, weight updates naturally can not be performed after each training sample; that would require a sequential learning process. Instead, weight changes are accumulated and applied after every parallel iteration, each of which covers multiple training samples.

Limitations

For neural networks with multiple hidden layers, the algorithm introduces a problem called *vanishing gradients*. Backpropagation uses the derivative of the activation function. Exemplifying with the logistic function, the gradient of extreme input come very close to zero. During the backpropagation, some weights will see small updates, due to vanishing gradients. As extreme activations are encountered further down the network, the effect is amplified to the point where weights barely change at all. The result is that top layers optimize normally, while lower layers require significantly more training.

Underfitting is not the only problem with deep architectures, as they are also prone to overfit. Multiple layers create expressive models where successive layers reason over increasingly higher-level, abstract features. Although this is a powerful property, it also means that these models have the complexity to fit fine details in the samples. As discussed in section 2.2.8, this will often lead to overfitting.

Dropout

By making a small change in the backpropagation algorithm, overfitting in deep architectures can be reduced. The technique is known as *dropout*, and was introduced in [18]. During training, each hidden node is assigned a probability of being disabled, making it produce an activation value of zero. The decision whether to drop a node is made independently each time a node calculates its output, and the effect is necessarily only active until next activation.

Deactivating, or dropping neurons will hurt the performance of the network, since successive nodes can no longer rely on the preceding layer being complete. Furthermore, it is not predictable which nodes will be disabled. This means the hidden units have to be more general in their activations, potentially having to account for several disabled nodes. Experiments in [18] show a higher error rate with dropout for training set classification, but lower error rates for out-of-sample data. Note that dropout is strictly used for training.

Dropout adds one additional hyper-parameter, the probability of disabling a node. Possibly the optimal value of this parameter depends on the problem and data, but [18] points to 0.5 as a safe reference point.

Applications

Backpropagation is often exemplified using FFNNs, where each neuron in a layer is connected to all in the next. This is usually for pedagogical reasons, however the algorithm itself does not specifically require a fully connected architecture, as long as the network is directed and acyclic. Combined with great parallelizability for fast training, this makes backpropagation an effective and versatile learning algorithm for supervised learning.

2.3.4 Deep Learning

As discussed in section 2.3.3, training deep networks using backpropagation presents some challenges. *Deep learning* is a popular term that refers to techniques that enables training deep architectures. Because deep networks are prone to both underfitting and overfitting, deep learning algorithms address one or both of these issues. Unsupervised pre-training is discussed in section 2.5.3, and can be regarded as one of the breakthroughs in training deep models. The weight sharing found in Convolutional Neural Networks (CNNs) reduce overfitting by forcing groups of neurons to use identical weights, see section 2.4.4. Dropout has already been mentioned in section 2.3.3, and is another measure for reducing overfitting.

2.4 Supervised Models

The following section introduces models that use supervised learning to adjust their weights, as explained in section 2.2.2. These models require every training sample to be accompanied by an ideal output.

2.4.1 Recurrent Neural Network

While FFNNs only allow forwards connections, a Recurrent Neural Network (RNN) may contain *feedback signals*. This implies that RNNs keep updating an internal state known as *context* or *memory*, which in turn is considered when computing future states. Theoretically the temporal dependency may reach arbitrarily far back in time, although current learning algorithms and computational power enforce practical restrictions.

Introduced in [11], the Elman network is an early example of recurrent models. As depicted in figure 2.8, the model includes dedicated *context units*, marked C. Input, hidden, bias and output neurons are also shown. Each hidden unit outputs to its own context unit upon activation, making the context layer act as an internal memory. The context feeds delayed values back into the hidden layer, enabling the network to detect features based on current input and past states. Note that activations going into the context are weighted by a fixed value of one, essentially copying them over. Similarly, context neurons use the identity activation function, which simply outputs the input: $f(p) = p$.

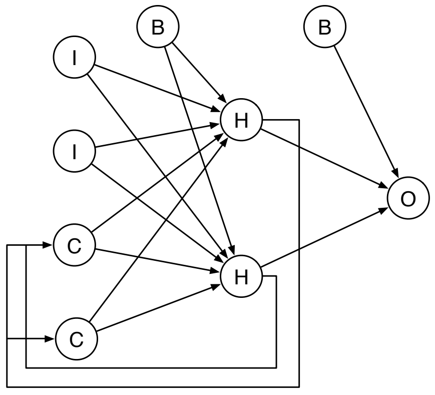


Figure 2.8: The recurrent architecture known as an Elman network. C are context units.

Backpropagation Through Time

Elman networks, and RNNs in general, may learn by using a slightly modified version of backpropagation known as Backpropagation Through Time (BPTT). BPTT *unfolds* the network with respect to time, creating a deep, layered architecture (illustrated in figure 2.9). Because the feedbacks create a circular pattern, we cannot fully unfold the network. A hyper-parameter limits how far back the network should be traversed. Figure 2.9 unfolds two time steps.

The unfolded model can learn through normal backpropagation, with a couple of exceptions. Firstly, the input sequence is distributed over multiple layers, rather than the standard procedure of using the first layer

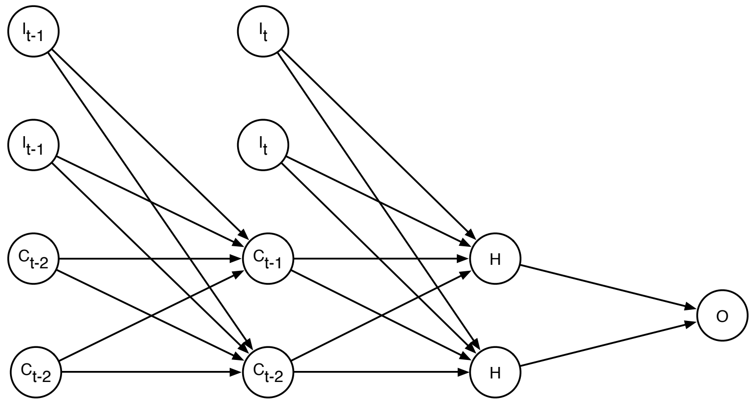


Figure 2.9: BPTT unfolds an RNN two steps through time, making it finite. Bias neurons are not shown.

as the only input layer. Secondly, the backwards pass will result in different weight adjustments for the same node at different unfolded layers. This can be dealt with by weight averaging. Note that the backpropagation issue of vanishing gradients mentioned in section 2.3.3 is especially relevant for BPTT, due to the circular structure of RNNs.

2.4.2 Echo State Network

Many attempts have been made to exploit recurrent architectures. A family of RNN models relies on using a fixed hidden *pool* or *reservoir* of neurons in order to simplify the learning procedure. Models like Echo State Networks (ESNs) [24] and Liquid State Machines (LSMs) [32] both fall under this category known as *reservoir computing*. As these two approaches are similar to each other, I choose to focus on the ESN due to its good results on the chaotic Mackey-Glass time series in the original paper [24]. LSM neurons are more biologically accurate, however this in itself is not a goal for this particular setting.

Using the same mathematical notation as in [24], figure 2.10 depicts an example ESN architecture: The input layer $\boldsymbol{u}$, a sparsely (and typically randomly) connected reservoir $\boldsymbol{x}$ and an output layer $\boldsymbol{y}$. There are at most four sets of weights: $\boldsymbol{W}^{in}$ for input to reservoir, $\boldsymbol{W}$ for internal connections and $\boldsymbol{W}^{out}$ for incoming output connections. Lastly the network may optionally incorporate output feedback connections, with corresponding weights $\boldsymbol{W}^{back}$.

The next output layer state of an ESN is calculated as follows

$$\boldsymbol{y}(n+1) = \boldsymbol{f}^{out}(\boldsymbol{W}^{out}(\boldsymbol{u}(n+1), \boldsymbol{x}(n+1), \boldsymbol{y}(n))) \tag{2.5}$$

where $\boldsymbol{f}^{out} = (f_1^{out}, ..., f_i^{out}, ..., f_L^{out})$, f_i^{out} is the activation function for output activation y_i and $(\boldsymbol{u}(n+1), \boldsymbol{x}(n+1), \boldsymbol{y}(n))$ is the concatenation between the respective activations. Note that the network can be configured not to

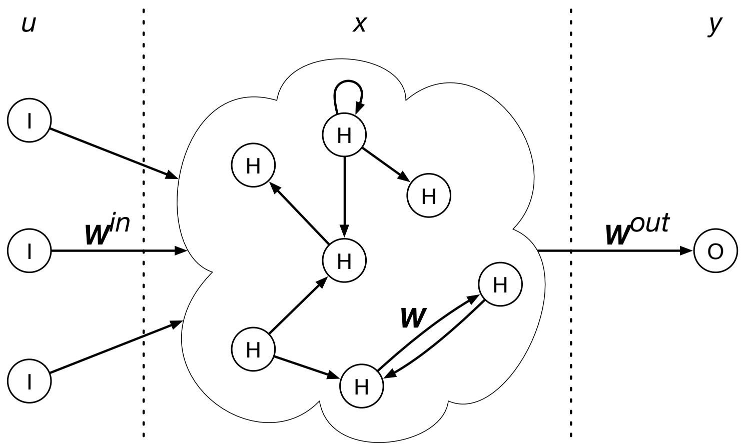


Figure 2.10: An ESN architecture. Illustration inspired by figure 1 in [24].

include connections from input to output and output to output. In doing so, equation (2.5) is simplified by eliminating the dependency on $\boldsymbol{u}(n+1)$ and $\boldsymbol{y}(n)$.

Equation (2.5) depends on an updated internal state:

$$\boldsymbol{x}(n+1) = \boldsymbol{f}(\boldsymbol{W}^{in}\boldsymbol{u}(n+1) + \boldsymbol{W}\boldsymbol{x}(n) + \boldsymbol{W}^{back}\boldsymbol{y}(n)) \tag{2.6}$$

where $\boldsymbol{f} = (f_1, ..., f_i, ..., f_N)$, f_i is the activation function for internal activation x_i. As mentioned the output-to-reservoir connections are optional, and architectures like figure 2.10 that do not use them may eliminate the expression $\boldsymbol{W}^{back}\boldsymbol{y}(n)$ from equation (2.6).

By fixing all but the output weights the learning problem is reduced to a linear regression problem. This is the key feature that makes ESNs efficient to train, but it also requires a reasonably weighted reservoir. There are several hyper-parameters that impact how the weights are initialized. As the name suggests, *input scaling* is an interval defining the upper and lower limit between which input weights are randomly sampled. *Spectral radius* of the reservoir can be adjusted to scale its weights. [31] points out that tasks with longer temporal dependencies usually require larger spectral radii. On the other hand, a large spectral radius might violate the *echo state property*, expressing that internal states should be uniquely defined by the input and its fading history, see [24, 31]. In other words, we want the input to echo around the network for a certain number of steps, but not indefinitely. Specifically, a spectral radius below one is likely to maintain the echo state property. The *reservoir capacity* controls the number of internal neurons.

2.4.3 Time-Delay Neural Network

[51] considers the task of recognizing *phonemes*, the atomic sound units that make up speech. Extracting them from audio input is challenging for fully connected FFNNs, mainly because spoken sentences contain pauses

of variable length between words. If the model is presented with an unseen sample where the location and length of pauses are dissimilar to training samples, it is likely to produce ambiguous activations that are *washed out* by pause noise. Traditionally, phoneme recognition therefore required heavy pre-processing of the audio input, in order to identify where individual phonemes begin and end.

Figure 2.11 shows a variation of the network architecture introduced in [51], intended for phoneme recognition. Time-Delay Neural Networks (TDNNs) use special neurons that have separate weights for different *delays* of an input. How many time steps to delay, is domain-specific. In the original paper [51], each neuron in the first hidden layer connected to three different versions of the input variables through time. Figure 2.11 denotes the time steps $t...t-5$ for variables $v0...v3$. Every row in the hidden layer is associated with a contiguous segment of time steps. Each neuron within the same row is connected to the input neurons representing said time steps.

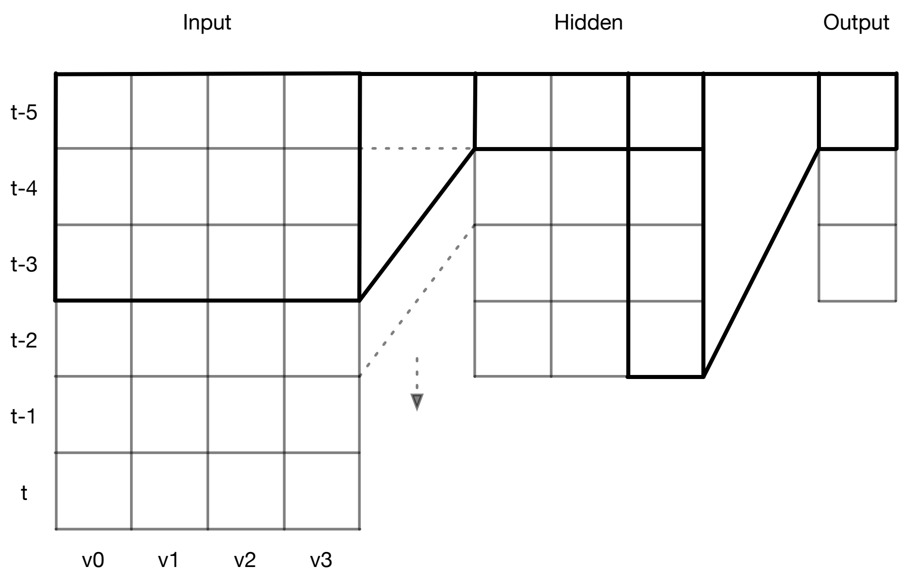


Figure 2.11: A fully unfolded TDNN architecture. The illustration is inspired by figure 2 in [51].

Because each TDNN hidden unit covers full spatial states within its own time window, [51] suggests units within the same row share their weights. This ensures features are recognizable at all time window positions of the input. Ultimately this means that even if two recognizable features are separated by noise, some of the hidden neurons are likely to find them without washing them out with noise. Sharing weights also reduces the number of parameters considerably.

As illustrated by figure 2.11, TDNNs are easily viewed as a partially connected FFNN. Training a TDNN can then be done using backpropagation, or other FFNN learning algorithms.

2.4.4 Convolutional Neural Network

Although an FFNN trained with backpropagation theoretically might handle complex problems given enough neurons, it often requires significant amounts of time and resources. This is especially true when modeling high-dimensional data such as the pixels in images; not only can images be large in terms of resolution, but color images additionally consist of separate values for each color channel.

The first Convolutional Neural Network (CNN) appeared in [13]. It was developed with image recognition in mind, inspired by how biological brains process visual input by dividing the signals into smaller areas called *receptive fields* [20]. Several flavors of the artificial CNN exists, many of which are based on [28].

In addition to fully connected feed forward layers, CNNs introduce convolutional and sub-sampling layers. *Convolutional layers* represent the artificial counterpart to biological receptive fields, while *sub-sampling layers* offer a simple, dynamic way of reducing the data dimensionality.

As shown in figure 2.12, CNN layers may be separated into several *feature maps*: The first convolutional layer (marked *C*) consists of four feature maps, while the second has two feature maps. As the name suggests, each feature map represents the presence or absence of a particular feature in different parts, or receptive fields, of the input image. The input layer may also consist of several maps, as in the case for color images.

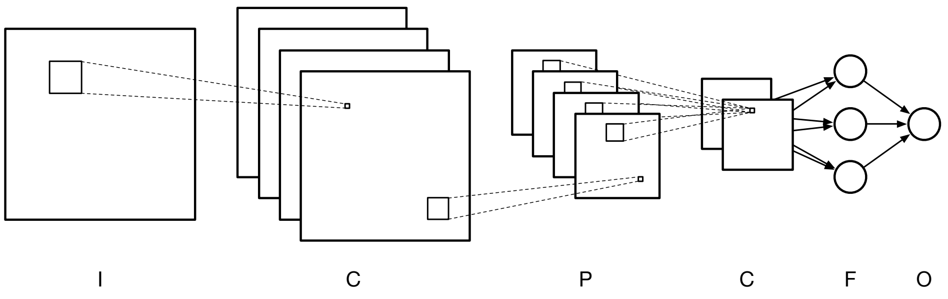


Figure 2.12: A CNN architecture, inspired by figure 2 in [28]. *C*, *P* and *F* are convolutional, pooling and fully connected layers, respectively.

Convolution

Convolutional layers exploit the spatial relationship between groups of pixels. For each neuron in a convolutional layer feature map, its activation represents the presence of a feature in the associated receptive field. Note that all neurons in a feature map respond to the same feature, but at different locations in the previous layer. This enables them to share the same set of weights, reducing the parameter count and allowing for efficient learning.

Feature maps are constructed so that neighboring receptive fields are represented by neighboring feature neurons. This ensures that the spatial relationship between local pixels is preserved in the feature map. Because of this property we are able to stack convolutional layers on top of each other, creating deep networks.

Receptive fields may overlap, making it possible to isolate a feature from its environment. Imagine trying to classify a small tennis ball in a large picture; because we allow overlapping receptive fields, the chances of a neuron having its receptive field filled with the tennis ball increases, rather than four neurons containing 25% of a tennis ball.

Another property of convolutional neurons is that their receptive fields actually span across all feature maps within their input layer. This ensures that multiple features are considered when finding new ones. Such behavior resembles fully connected FFNNs.

Sub-sampling

Convolutional layers do not necessarily produce feature maps of significantly smaller size than the input maps. Furthermore, the number of feature maps in a convolutional layer may be arbitrarily large, depending on how many features we wish to extract. To help deal with high-dimensional data and vast amounts of features, CNNs use sub-sampling or *pooling* in order to reduce dimensionality.

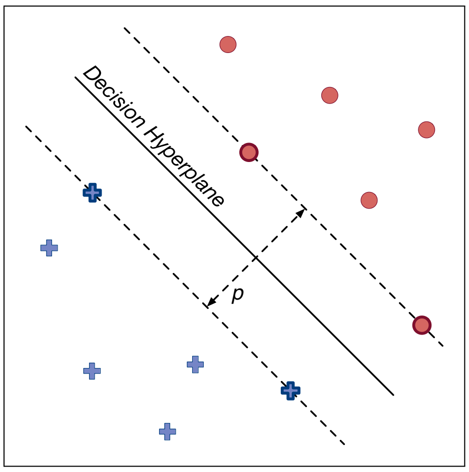
There are multiple ways of sub-sampling feature maps, and two common techniques are min-pooling and max-pooling. Similar to the receptive fields of convolutional neurons, a pooling neuron is associated with a spatial *region* within the previous layer. Assuming max-pooling, pooling neurons activate with the same value as the largest activation within their respective regions. For min-pooling, the smallest activation is assigned. No activation function, bias or weights are used. To ensure sufficient scaling, pooling regions often do not overlap. The size of the regions further determines the extent of the scaling effect.

Sub-sampling layers retain the same number of feature maps as in the previous layer, and each neuron only samples from a single map. The effect is a dimensionality reduction where the most prominent feature activations are retained.

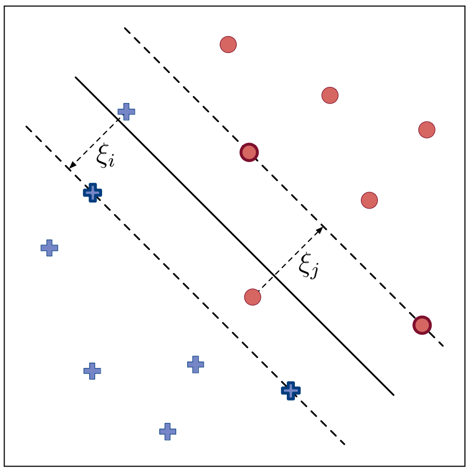
2.4.5 Support Vector Machine

Support Vector Machines (SVMs) represent a well-known, widely applied mathematical method for learning from data samples or *vectors*. The simplest SVM is a linear classifier. Main aspects relevant to the thesis are presented here, however consulting other sources like [33, 43] is recommended for a thorough understanding of SVMs.

Intuitively speaking, an SVM attempts to find the *decision hyperplane* that maximizes distance to the data points. By maximizing this margin we make an educated guess at the optimal solution, rather than accepting



(a) Four support vectors define the decision hyperplane.



(b) Slack variables allow some training vectors to be misclassified for better generalization.

Figure 2.13: SVM learning. Illustrations inspired by figures 15.3 and 15.5 in [33].

any valid solution. The decision hyperplane is defined by a weight vector $\boldsymbol{w}$ and an intercept term b, which also represent the variables we wish to optimize. *Support vectors* are the data points located closest to the decision hyperplane. Furthermore, support vectors are also aligned along one out of two hyperplanes depending on their class, shown in figure 2.13a. The distance between these two hyperplanes is known as the margin, defined as $p = 2/||\boldsymbol{w}||$ where $||\boldsymbol{w}||$ denotes the norm of $\boldsymbol{w}$.

Figure 2.13a shows the margin p, along with three hyperplanes: Support vectors from each class are aligned along a respective hyperplane, with the decision hyperplane centered between them. The SVM will attempt to maximize the margin p, in order to separate the classes as cleanly as possible. SVMs reformulate this as a minimization problem: Maximizing $2/||\boldsymbol{w}||$ is the same as minimizing $\frac{1}{2}||\boldsymbol{w}||$. Given that $||\boldsymbol{w}|| = \boldsymbol{w}^\top \boldsymbol{w}$, the final optimization problem is defined:

$$arg\ min_{w,b} \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} \tag{2.7}$$

subject to:

$$y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 \tag{2.8}$$

where $\boldsymbol{x}_i$ is a training vector and y_i is its target class valued as either negative or positive one. The term $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)$ defines what is known as the *functional margin*, and may be thought of as a measure to whether a vector is correctly separated by the decision hyperplane or not.

To help deal with noise and outliers, a slack variable ξ_i for each training vector is introduced. This allows some vectors to violate the constraint in equation (2.8), if their slack variables are sufficiently large. Two misclassifications allowed due to slack variables are illustrated in figure 2.13b. To prevent large slack values, and by extension an underfitted decision hyperplane, the updated minimization problem is:

$$arg\,min_{w,b}\,\frac{1}{2}\boldsymbol{w}^\top\boldsymbol{w} + C\sum_i \xi_i \tag{2.9}$$

subject to:

$$y_i(\boldsymbol{w}^\top\boldsymbol{x}_i + b) \geq 1 - \xi_i \tag{2.10}$$

This makes C a hyper-parameter of the SVM, which controls the penalty of misclassified vectors.

Kernel functions

Equations (2.9) and (2.10) express the SVM *primal form*. It is more intuitive to understand, and therefore suitable for a basic introduction. There is however, an optimized formulation of SVMs known as the *dual form*. Exploring the dual form is beyond the scope of this thesis, but there is one crucial detail that makes it worth mentioning: At its core, the dual form relies on calculating dot products between pairs of data vectors. This can be exploited through *kernel functions*.

[33] describes kernel functions as functions that correspond to a dot product in some expanded feature space. Given that the dual form SVM relies on such dot products, we can replace them with kernel functions. In fact, because kernel functions operate in a higher dimensional space than the original vector, data vectors become more easily separable. In other words, linearly separating vectors in a higher dimension actually corresponds to non-linear separation in the lower dimension. Translating vectors to a higher dimension is not feasible however, and that is where the dual form is forgiving; it does not require mapping vectors to a higher dimension, only calculating the dot product in the higher dimension. Kernel functions do this so efficiently, that the procedure has become known as the *kernel trick*. A large kernel output between an input vector and support vector, contributes towards classifying the input vector similarly to the support vector.

There are numerous kernels to choose from, one of which is the Gaussian kernel:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}) \tag{2.11}$$

Gaussian kernels produce values close to one for similar vectors, but drops rapidly towards zero as their distance increases. It does not matter in what direction the vectors are located with respect to each other, as only their proximity to each other is regarded. Intuitively this forms a Gaussian curve in a hyper-dimensional space with its width implied by the hyper-parameter σ. Intuitively, a large σ smooth out data at the expense of less complexity.

Support Vector Regression

SVMs can be modified to model regression problems. This was first introduced in [10], and is known as Support Vector Regression (SVR). Training targets now consist of real values rather than class indicators, and the optimization constraints in equation (2.8) are updated to account for this change:

$$y_i - \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - b \geq \epsilon \tag{2.12a}$$

$$\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b - y_i \geq \epsilon \tag{2.12b}$$

where $\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$ is the dot product between $\boldsymbol{w}$ and $\boldsymbol{x}_i$, and together with b represents the model's prediction of $\boldsymbol{x}_i$. Equation (2.12) ensures the optimizing continues until all prediction errors fall within a certain threshold, defined by the hyper-parameter ϵ. In other words, ϵ defines how accurately the model is fitted. Slack variables are omitted in equation (2.12) for clarity, but they are typically applied for better generalization.

2.5 Unsupervised Models

The neural networks discussed to this point, are all trained using supervised learning algorithms. As discussed in section 2.2.3, some machine learning models learn from input data alone, without ideal outputs for the model to train towards. In this section several kinds of unsupervised neural networks will be discussed.

2.5.1 Restricted Boltzmann Machine

Similar to what the FFNN represents within supervised networks, the Restricted Boltzmann Machine (RBM) is a widely applied, basic unsupervised neural network. In addition to bias nodes, RBMs consist of two main components: A *visible* and a *hidden* layer. Unlike FFNNs, visible and hidden units are connected by *undirected* connections. This implies the model can infer the hidden states given a visible layer, but also the other way around. Intuitively, we can think of the visible layer as the input layer, and hidden layer as the output layer, although it will become apparent that this is a slightly inaccurate analogy. Figure 2.14 illustrates the structure of an RBM.

RBMs are *stochastic*: Where FFNN neurons deterministically generate activations, the RBM creates *probabilities*. Every time we want to read the state of a neuron, it has to be sampled based on its probability. Sampling is done by picking a random number between zero and one, and then comparing it to the probability. If the random number is lower than the probability, then the unit activates with the value one, or zero otherwise. In other words, the RBM is a stochastic *binary* model with hidden states.

Let us assume the visible layer is given, and we want to infer the hidden or *latent* features. Calculating the probability of hidden unit h_j given the visible layer $\boldsymbol{v}$, is done by summing the weighted input signals

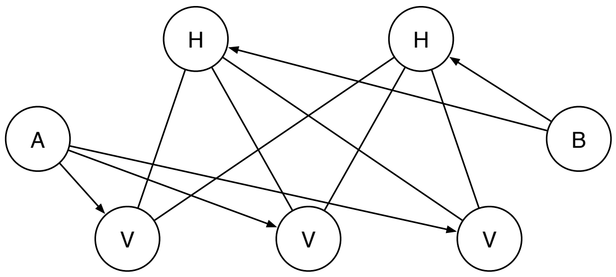


Figure 2.14: Structure of a restricted Boltzmann machine. Visible (V) to hidden (H) connections are undirected.

of h_j and bias, before passing the result through the logistic function f. Equation (2.13) describes this precisely:

$$p(h_j|\boldsymbol{v}) = f(b_j + \sum_i v_i w_{ij}) \tag{2.13}$$

where the logistics function is defined as $f(x) = 1/1 + e^{-x}$, b_j is bias for latent unit j and w_{ij} is the weight between visible unit i and hidden unit j.

As was earlier mentioned, visible to hidden connections are undirected. By first sampling the hidden state, we can compute visible probabilities following equation (2.14):

$$p(v_i|\boldsymbol{h}) = f(a_i + \sum_j h_j w_{ij}) \tag{2.14}$$

where a_i represents bias for visible unit i.

Note how these newly calculated visible probabilities may be sampled, producing a new visible state. Starting with a visible state, sampling a hidden state before *reconstructing* another visible state is known as one iteration of alternating *Gibbs sampling*. For a trained RBM, the reconstruction following many iterations of Gibbs sampling is expected to come as close to the original input vector as is possible. Because the hidden layer normally is smaller than the input layer, this can only be done if the latent features are descriptive of the data.

There are several use cases for RBMs. The learned latent features can be applied for feature extraction, perhaps for use in conjunction with other machine learning models. Because hidden features describe the data more efficiently, they can also be used for compression. Another scenario is filling out incomplete input data, by inferring reconstructions. The continuation of a time series can also be predicted by reconstructing future time steps. Note that some applications use the hidden state as output, while other rely on the visible layer. Also, the visible layer is not a pure input layer, because it sometimes holds reconstructions rather than the original input.

Contrastive Divergence

The Contrastive Divergence (CD) algorithm introduced in [16], is a learning algorithm for Product of Experts (PoE). PoEs work by normalizing the product of several expert models. By considering each hidden unit in an RBM as an expert, RBMs can be expressed as a special case of PoE. CD training therefore is interesting also in context of RBMs. For more information regarding PoE models and their relationship to RBMs, the reader is referred to [16].

Being an unsupervised model, the RBM can not calculate and propagate error similar to FFNNs. However, as already discussed we can perform alternate Gibbs sampling to create reconstructions of the original input sample. CD exploits this property, and calculates error based on differences between original and reconstructed visible states. Weights are updated following equation (2.15):

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}) \tag{2.15}$$

where ϵ is the learning rate and angled brackets denote expectation under a specified distribution, namely data or reconstruction. In other words, $\langle v_i h_j \rangle_{data}$ is the product between original visible unit j and hidden unit j sampled from the input vector. $\langle v_i h_j \rangle_{recon}$ is computed equivalently, except using the reconstruction.

$\mathrm{CD_k}$ is an extension to CD, where alternating Gibbs sampling is performed k times. Although it increases the computational burden, [15] states that $\mathrm{CD_k}$ usually results in better models.

Continuous Input

Due to the use of binary units, the traditional RBM only accepts data on binary form. Sometimes referred to as Gaussian-Bernoulli RBMs, these slightly modified RBMs take real-valued numbers as input, but still keep a binary representation of the latent features.

Removing the binary restriction on the input is essential for many applications, but not without consequence. During training, there no longer are any bounds for the reconstruction; unlike binary values, real numbers are unbounded. Ultimately, some of the reconstructed values deviate greatly from the actual input. Given equation (2.15), this may lead to large weight changes, resulting in an unstable training process. For this reason it is recommended to consider reduced learning rates, see [15].

Although it is possible to use a Gaussian-Gaussian RBM, [15] points to why sticking with binary latent variables is desirable. Firstly, we are asking the network to represent continuous input as binary features. Although some finer details of the data will be lost, a decent representation could be possible if the most prominent features are extracted. This is commonly known as a form of *regularization*, and helps create models that do not overfit. Secondly, using Gaussian hidden units will further amplify the above-mentioned issue of training instability, resulting in too slow or unpredictable learning.

2.5.2 Conditional Restricted Boltzmann Machine

Originally used with time series of motion capture data, the Conditional Restricted Boltzmann Machine (CRBM) as described in [50] offers a way of modeling temporal relationships between multiple variables. Motion capture data track multiple variables, for example representing the angles of different joints in the body.

CRBMs are extensions to RBMs, where multiple visible layers represent the visible state at different time steps. Figure 2.15 shows an example where three time steps are considered: The current state, and the two steps preceding it. We say the model has an *order* of two, because it requires at least two time steps to be sampled.

The additional visible layers that hold states for previous time steps use directed connections, thereby acting more like a *dynamical bias* rather than regular visible layers. In other words, the current visible layer and latent state are both inferred *conditional* on previous visible states.

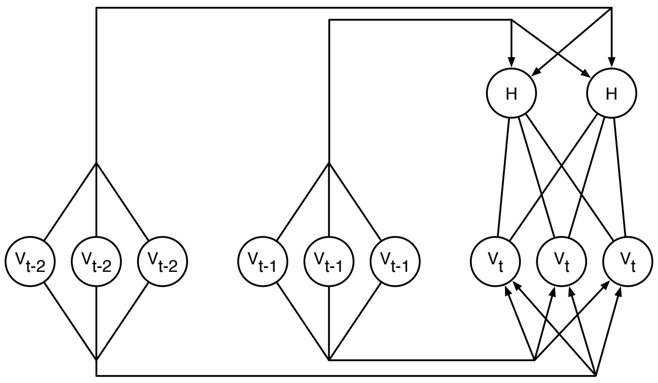


Figure 2.15: The structure of a CRBM. Layers are fully connected, even though the figure aggregates some connections for brevity. Bias not shown.

Similarly to RBMs, CRBMs can predict the continuation of a time series by inferring future time steps. It is achieved by only providing input to the directed visible layers, and initializing the undirected visible layer to a neutral state. After several iterations of alternating Gibbs sampling, the last reconstruction in the undirected visible layer now contains a prediction of the next time step.

An ability to capture multiple styles of motion within the same network became apparent in [50], and further explored with the factored CRBM introduced in [49]. Understanding multiple styles is beneficial for most applications, and the model also proved to be able to transition smoothly between them in a generative setting. In these experiments however, the CRBM was usually stacked in a deep configuration, as explained in section 2.5.3.

2.5.3 Deep Belief Network

Deep Belief Networks (DBNs) consist of several RBMs stacked on top of each other, where all but the topmost layer uses directed connections. A simple DBN architecture is depicted in figure 2.16. Note that the network is rotated in order to save space. The visible layer is located to the left in the figure, but will be referred to as the bottom layer, and vice versa for the rightmost hidden layer. The DBN can model complex problems due to its deep nature. Directed connections enable fast inference, while still retaining RBM properties in the top layer.

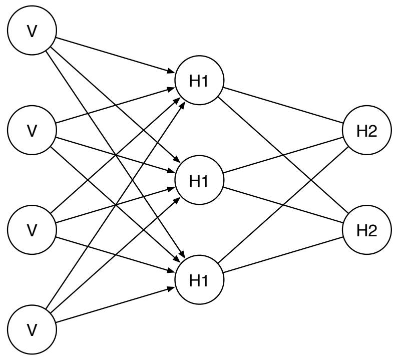


Figure 2.16: A minimal deep belief network. Bias nodes are not shown.

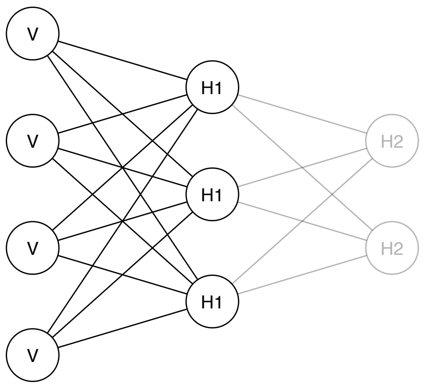
Just like the RBM, DBNs are *generative* models that can probabilistically generate input samples, as well as infer hidden features. Unlike other deep models that suffer from the shortcomings of backpropagation (section 2.3.3), DBNs are unsupervised and require other means of learning. In [17], a *greedy layer-wise* training algorithm was introduced specifically for DBNs. This learning algorithm has later been applied to supervised deep networks as well, and is commonly referred to as *unsupervised pre-training*.

Unsupervised Pre-training

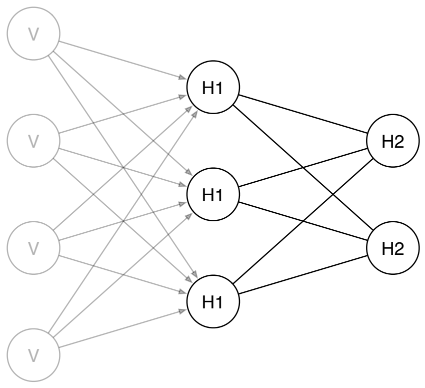
Unsupervised pre-training only considers two layers at a time, completely adjusting the weights between them before advancing further. It begins at the bottom, with the input layer and the first hidden layer. The algorithm now treats the two layers as the visible and hidden layer of an RBM, and adjusts the weights according to CD training as explained in section 2.5.1. Figure 2.17a visualizes the process at this stage.

Once the weights have converged they are locked in place, changing the connections back to being directed. Now the procedure advances one layer up, considering the first and second hidden layer as an RBM. This is shown in figure 2.17b. Note how the training vectors at this stage are actually the latent states of the previously trained RBM. This process is repeated until all the weights have been optimized. The last set of connections is then kept undirected, to complete the DBN.

When using unsupervised pre-training for supervised networks, learn-



(a) Training the first set of weights in the DBN.



(b) Building upon the already trained lower layers, the two top hidden layers are finally adjusted.

Figure 2.17: Performing unsupervised pre-training on a simple DBN.

ing must stop at the final hidden layer; the last set of weights are attached to the output layer, which has to be trained using ideal values and supervised methods. Typically, backpropagation is applied following the pre-training to adjust the final layer and *fine-tune* the pre-trained weights. A decreased learning rate is often used to promote finer changes in the initialized weights.

The results of [17] show that training a DBN with three hidden layers using unsupervised pre-training, produced superior predictions compared to other discriminative algorithms. They were measured by number of misclassifications of handwritten digits, using Mixed National Institute of Standards and Technology database (MNIST).

Chapter 3

Methodology

This chapter describes the procedures used to obtain the results. Section 3.1 states the machine learning problem, and theoretically justifies why it is a challenge. Section 3.2 details the experimental procedures, including implementation notes, model optimization and evaluation metrics. Section 3.3 describes the selection of stocks, as well as how the stocks are pre-processed. Finally, section 3.4 states the models to be tested, and their hyper-parameters.

3.1 Challenges

Artificial Neural Networks (ANNs) can be thought of as function approximators, see section 2.3.2. We may view a stock as a function of time: $f(t) = v$, where t is the time and v the share value. Remembering that v is actually a discrete value (section 2.1.4), this is strictly speaking a discontinuous function. Furthermore, modeling a stock as a function of time is unlikely to yield good results; one would have to find a consistent connection between time and price alone.

Equation (3.1) describes a more realistic problem formulation:

$$f(v_{t-1}, v_{t-2}, ..., v_{t-n}) = v_t \tag{3.1}$$

where v_t is the share value at time t. By contrast to the previous function, we here use the n most recent share values as input rather than the time. This allows us to search for temporal patterns that lead to the same conclusion. Note that equation (3.1) is simplified, and does not take into account all the variables included in the experiments. As described in section 3.3, multiple variables are included in the input data. Spatial patterns between them will therefore also be modeled.

Equation (3.1) introduces another challenge, however: There is nothing preventing a stock from having the same pattern lead to different outcomes. For example, a stock that develops with the sequence down, down and up, may very well continue down, down and down the next month. In this simplified scenario, the pattern of two negative price changes have led to both positive and negative change at different occasions. By extension, equation (3.1) is expected to output dissimilar values for certain equal, or similar input. Because functions are defined as always producing the same output for equal input, this implies that equation (3.1) is not a function. We can no longer rely blindly on the ANN's ability to approximate functions as discussed in section 2.3.2.

3.2 Test Scenario

The goal of the experiments in this work is to train neural networks for daily stock data prediction and compare their performance. Six types of models are evaluated, consisting of five neural networks and one Support Vector Regression (SVR) as a baseline. Reasons for including each of them as well as choice of hyper-parameters will be discussed in section 3.4. The models to be tested are as follows:

- Support Vector Regression
- Feed Forward Neural Network
- Echo State Network
- Conditional Restricted Boltzmann Machine
- Time-Delay Neural Network
- Convolutional Neural Network

Choice of data sets is discussed in section 3.3. Two indices and three stocks constitute the data selection:

- S&P 500
- NASDAQ Composite
- Bank of America Corporation
- Microsoft Corporation
- Petróleo Brasileiro S.A. - Petrobras
- Exxon Mobile Corporation

3.2.1 Implementation Notes

A substantial part of applying machine learning to a problem is related to data processing, requiring working efficiently with vectors and matrices. In order to streamline the implementation process, all code was written in MATLAB®, using libraries for model implementations. MATLAB® was chosen for its scripting features, built-in matrix operators and active community, although it should be mentioned that Python also has much traction within the neural network community.

Project Functionality

To benchmark the different models seamlessly, a common test environment was implemented. The main functions of the code are described in the following paragraphs.

Data Processing Data should not be partitioned directly into training samples, but rather *preprocessed* first in order to optimize learning. Preprocessing should be reversible in such a way that the predictions can be converted back to stock prices. Several approaches were tried, including simply normalizing the variables within a range, or calculating the percentage each variable changes between time steps. The final approach considers change in values with respect to time or other variables. Information regarding the day of week is encoded into each input vector. Lastly, a common normalization scheme is also applied, where the data are converted to having zero mean, and a standard deviation of one. Considerable time was invested into finding a reasonable data processing scheme, and the exact approach is detailed in section 3.3.3.

Library Interfacing Libraries are relied on as much as possible to reduce development time, refer to table 3.1 for an overview. In order to use the libraries in the project, an adapter for each library is written to streamline model configuration, training and testing.

One of the largest challenges encountered during this study, was finding promising neural networks that were already implemented in a reasonable language. While researching the models, it turned out that MATLAB® had the largest number of models available through libraries, compared to other languages. Most of the libraries are open source and under constant development. This poses a risk, considering there might be bugs in the implementations. A significant part of the project consisted of trying and discarding different implementations. Some were discarded due to bugs, missing features or simply incompatibility issues related to hardware or drivers.

An unsupervised version of the Convolutional Neural Network (CNN) known as the Convolutional Restricted Boltzmann Machine (convRBM) was initially included in this study. This model had few implementations available, and like many other convolutional libraries only supported square receptive fields, as opposed to any rectangular shape. As is discussed in section 3.4.6, rectangular receptive fields are preferred due to the input variables being scaled differently and having dissimilar meaning. Although this issue was easily fixed in the library, other bugs were encountered, which ultimately led to the exclusion of convRBMs in this study.

Result Extraction Once models are configured, trained and tested, different measures of performance are generated from the predictions. Measures used are detailed in section 3.2.4. The results, consisting of predictions and performance measures, are then exported to different files. Note that the exported files are not necessarily human readable, but are intended for seamless inclusion in this thesis.

Caching Every time a new performance measure was added as part of the development process, models had to be recreated and retrained in order to export the new values. Some of the models require significant resources to train, which quickly rendered this naïve approach unfeasible. A caching mechanism was implemented, to automatically store and retrieve models and predictions based on the same data set. This way, predictions are simply loaded from disk, and updated performance measures are efficiently calculated and exported. Caching also prevents slight deviations between exports normally caused by the stochastic learning of neural networks. The data sets themselves are also cached using a similar mechanism.

Spatial Inputs

Several of the models in this study expect spatial data, as expressed in table 2.1. For these models, data can not be presented as a time series, but have to be converted to a spatial format. The procedure slides an imaginary window across the time series, and copies the values within each window position into a new training or test sample.

CRBM Implementation

The sample code from [50] was used as the Conditional Restricted Boltzmann Machine (CRBM) implementation. Because it is a stand-alone implementation intended for modeling human motion capture data, some modifications were made to accommodate both stock data and the rest of the project.

As mentioned in section 2.5.1, Restricted Boltzmann Machines (RBMs) in general can generate the continuation of a time series by inferring visible states. This would normally imply that the length of the visible layer limits the number of steps we can predict. The code in [50] works this way, because only one frame of motion was generated at a time in that study. As explained further in section 3.2.4, models are evaluated for both single- and double-step prediction in this thesis. To prevent complications in the code, the problem was worked around by injecting correct target prices in the training vectors. This effectively means that double-step training samples are identical to single-step vectors, except for the last time step of the price we are predicting.

Libraries

Table 3.1 lists the libraries used in the experiment. Note that the *purpose* column only indicates what I used the library for, not what the library has to offer.

Library	Purpose	URL
Support Vector Regression	SVR	[4]
Deep Neural Network	FFNN	[47]
Neural Network Toolbox	TDNN	[37]
ESN Toolbox	ESN	[23]
Sample code, [50]	CRBM	[48]
ConvNet	CNN	[7]
Datafeed Toolbox	Data	[6]

Table 3.1: Libraries leveraged for experiments.

Cross-validation (section 2.2.1) is not supported by all the libraries used. Not validating models on a separate data set is an obvious handicap and makes balancing underfitting and overfitting a challenge. However, in order to not give advantage to any model, cross-validation is discarded altogether. The number of epochs to train was determined using the same approach as with the rest of the hyper-parameters, explained in section 3.2.2.

3.2.2 Tuning Hyper-parameters

Each kind of model in this experiment requires individual hyper-parameters to be set (section 2.2.6). Although these values sometimes can be set directly by reasoning, verifying and tuning through empirical testing is often required. Carefully adjusting hyper-parameters to each stock is undesirable, as we wish to avoid overfitting them to particular stocks. Instead, one of the data sets is selected to optimize hyper-parameters. Models are trained and evaluated on this time series, using the approach explained in sections 3.2.3 and 3.2.4.

Because each data set is unique, the one used for tuning hyper-parameters should not be arbitrarily chosen. Indices have fewer peculiarities as they are composed of multiple stocks and closely represent the market as a whole, as discussed in section 2.1.3. Hyper-parameters tuned to an index are therefore likely to suit other stocks as well. S&P 500 (GSPC) represents a wide selection of US companies from multiple industries, and will therefore be used for tuning purposes.

GSPC displays a bullish development during the test set, as is explored in section 3.3.1. Some hyper-parameter configurations let models predict bull price moves exclusively. Although this leads to good performance under bull market conditions, these models are potentially worthless once the market changes. Furthermore, the GSPC test set is actually almost balanced between the number of upward and downward movements; it is the size of these movements that makes the stock rise. Specifically, 56.0% of all movements are positive. Models are therefore tuned to be reasonably balanced between bull and bear predictions.

3.2.3 Training

Each time series contains ten years of daily stock or index data, starting from 1-1-2005. The first 70% percent of each series is used for training, leaving the last 30% for testing. Models are trained and tested separately on each time series, which makes 36 models total (six models for every time series).

Neural networks are initialized from a random state, making learning non-deterministic. CRBMs additionally rely on sampling random variables, while the Echo State Network (ESN) does not even optimize its random internal parameters. To help smooth out the randomness, models are trained identically ten times, each time with different initial states. Results are then *averaged* and presented along with *standard deviation*. SVR learning is deterministic, and is therefore exempted from this rule. Taking the additional neural networks into consideration, the total number of trained models in this study is 306.

Neural networks learn by being exposed to one training sample at a time, see section 2.3.3. In order to prevent the network to be biased towards certain samples that are clustered towards the end of the training set, training samples are always randomly shuffled. Again, this is only relevant for neural networks, as the SVR is invariant to sample order.

A common practice when evaluating machine learning models is to create the test set from random samples of the full data set, leaving the rest for training. Imagine for instance, a data set where the first 70% is bull, while the last 30% is bear. By randomly sampling the test set, we ensure that the model is tested and trained on both bull and bear samples. Although this will likely lead to better results, the technique conflicts with how stock market prediction works. Specifically, it does not make sense to train on data more recent than the samples we test on; it is the future prices we want to predict, not the past. To keep the experiments realistic, test data are defined as the most recent 30% of a given data set.

The models are trained using *ten* time steps of data to make predictions. This number was found following some preliminary tests, which showed that greater values tended to overfit models. To give the models equal testing conditions, they all use ten time steps to make predictions. ESNs do not explicitly limit the number of steps to model, and rather relies on the echo state property to wash out features (section 2.4.2). These models are therefore the only models not necessarily limited to ten time steps of input.

3.2.4 Evaluating

In evaluating a model, multiple aspects are considered. Equation (3.1) describes a *single-step* prediction problem: Given recent development, predict the next immediate value. Considering the noisy nature of stock data however, the next value might give less coherent patterns than the one following it. Models are therefore also evaluated for *double-step* prediction, where time step $t+1$ is predicted given steps $t-1$ through $t-n$. Keeping within the scope of this thesis with short-term predictions, only single-step and double-step predictions are considered. Short-term predictions also expose a larger potential for profit, simply because every movement can be traded closely. This is exemplified in table 4.1, where the ideal profit values are observed as higher for single-step prediction, compared to double-step.

Indicators used to evaluate models are listed below. They are calculated after the predictions have been post-processed back to actual price points.

Mean Squared Error

Mean Squared Error (MSE) is a widely used indicator for measuring error within machine learning, and expresses the average distance between predicted and ideal values. When accurate predictions are important, MSE provides an indication. It is calculated by averaging the squared prediction errors, meaning a low MSE translates to predictions being close to ideal values. Equation (3.2) defines MSE precisely:

$$MSE = \frac{1}{n} \sum (\hat{y}_t - y_t)^2 \tag{3.2}$$

where n is the number of predictions and $\hat{y}_t$ and y_t are predicted and ideal values, respectively.

Although the accuracy of prediction is perhaps not the most significant factor in the context of this thesis, MSE is a widely used performance indicator in other research. Including it requires little effort, and allows the reader to compare results to other studies.

Directional Accuracy

The direction that the price develops is relevant to traders, as the direction directly affects whether to take a long or a short position in the market. A prediction for time step t is regarded as having correct direction if the price change since $t-1$ has the same sign as the ideal change. *Directional accuracy* is measured by observing how many out of all predictions have correct direction. Specifically it is calculated by dividing the number of correctly predicted directions divided by the total number of predictions. High values indicate an ability to predict the direction of price movements. Directional accuracy DA is formulated mathematically in equation (3.3):

$$DA = \frac{1}{n-1} \sum pos((y_t - y_{t-1})(\hat{y}_t - y_{t-1})) \tag{3.3}$$

where pos is an unary operator defined as:

$$pos(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Bull Ratio

Looking at historical stock index development, the market may be described as being generally bullish in recent years. See figures 3.1 and 3.2

for development of indices GSPC and NASDAQ Composite (IXIC). A model that always predicts positive price development would perform well under bull market conditions, possibly producing a good directional accuracy. Such a model could not be considered reliable however, as it will underperform once the market turns. In order to find balanced models that predict adequately in both directions, the ratio between the number of upward and downward predictions is included. It is calculated by dividing the number of upward predictions by the total. A *bull ratio* of 0.6 therefore means the model predicted 60% of the test samples as bullish.

Although bull stocks tend to contain a larger number of bull than bear price moves, they are still reasonably balanced. This can be seen in table 4.1, where bull ratios for all time series are observed relatively close to 50%. This further advocates using bull ratio as a performance measure, given that extreme bull ratios are not present in real data.

Profit

There are many ways prediction models can be incorporated into a financial system, and the indicators mentioned above are meant to give a diverse view of general performance. The final metric of evaluation however, is very specific: Assuming a daily trading strategy that strictly follows the predictions of a model, what *profits* would it generate? This puts the model in a binary setting, where bull predictions result in long market positions, and bear predictions result in short market positions. Profits are purely theoretical, and no effort has been put into simulating market slippage, broker commissions or other fees. A profit of 0.1 translates to 10% income during the whole test period, which spans approximately three years.

For double-step prediction, profits are calculated by alternating trades between two separate accounts. Since trades in that case are active for two time steps, a trade is guaranteed to exit the market before a new trade initiates within the same account. Furthermore, this strategy ensures that all predictions will be traded. The final profits are calculated with respect to the sum of initial and final capital across accounts.

3.3 Data

Data selection and preprocessing lay the foundation for the experiments.

3.3.1 Data Selection

There is a plethora of stocks to choose from, all with different attributes. Table 3.2 lists the stocks and indices, along with their ticker symbols, that will be used for learning and testing the models. The chosen instruments are well-known, highly liquid stocks and indices. The stocks are listed at the New York Stock Exchange (NYSE) or the National Association of Securities Dealers Automated Quotations (NASDAQ), and together span across multiple industries.

Symbol	Name	Type	Exchange
GSPC	S&P 500	Index	N/A
IXIC	NASDAQ Composite	Index	N/A
BAC	Bank of America Corporation	Stock	NYSE
MSFT	Microsoft Corporation	Stock	NASDAQ
PBR	Petróleo Brasileiro S.A. - Petrobras	Stock	NYSE
XOM	Exxon Mobile Corporation	Stock	NYSE

Table 3.2: The stock selection.

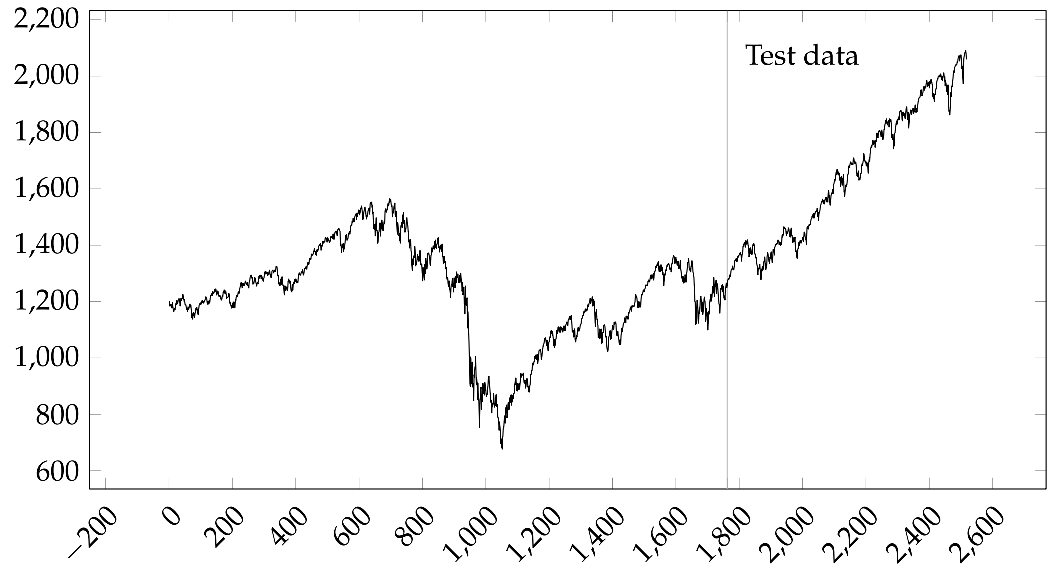


Figure 3.1: GSPC daily closing price development between 1-1-2005 and 1-1-2015.

Indices

In addition to single companies, two indices each consisting of multiple stocks are also included in the data selection.

S&P 500 Five hundred US leading companies constitute S&P 500 (GSPC), providing a diversified selection of stocks. [21] claims the index covers approximately 80% of available market capitalization, making it a relevant topic for research. GSPC price development is shown in figure 3.1. Bull conditions are observed throughout the test data, while the training set has an approximately neutral total development.

NASDAQ Composite Containing over 3000 instruments listed on the NASDAQ, NASDAQ Composite (IXIC) consists mainly of stocks within the technology sector. Although having a strong representation within one sector makes the index less diverse, the sheer amount of companies represented differentiates this index from other indices, rendering it interesting

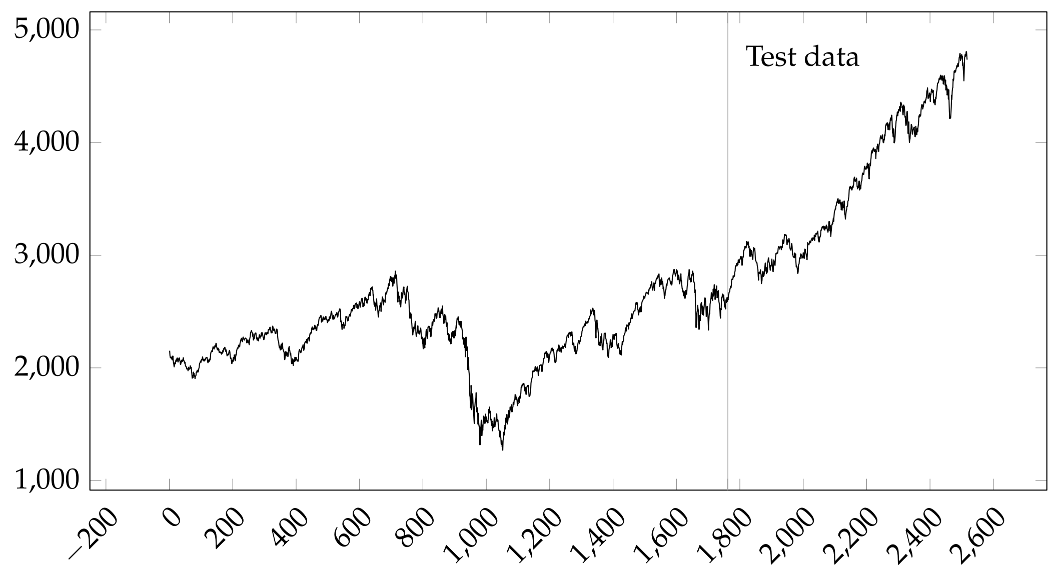


Figure 3.2: IXIC daily closing price development between 1-1-2005 and 1-1-2015.

to experiment with. Furthermore, [36] states that only companies listed exclusively on the NASDAQ are eligible for inclusion in IXIC. This ensures diversity for the experiment. Visually IXIC carries a strong resemblance to GSPC, as seen in figure 3.2.

Stocks

Three liquid companies are included in the experiments, representing diverse combinations across sectors, stock exchanges and market conditions.

Bank of America Corporation Listed on the NYSE, Bank of America Corporation (BAC) is currently serving 49 million consumers and small businesses, according to [1]. The company was founded in 1874, and represents a liquid stock within the financial sector. Judging by the price plot in figure 3.3, the share has suffered a massive drop during the past ten years. Looking at the intersection between training and test samples however, we observe that it changes to less volatile bull conditions. Different market conditions between training and test data makes this a particularly interesting stock.

Microsoft Corporation Founded in 1975, Microsoft Corporation (MSFT) quickly grew to become a major name within the technology sector. The company is listed on the NASDAQ, and employs over 120 thousand people worldwide, according to its website [34]. In figure 3.4 we can observe how the training samples contain both bull and bear periods, ultimately resulting in a neutral trend. Test data on the other hand, is decidedly bullish.

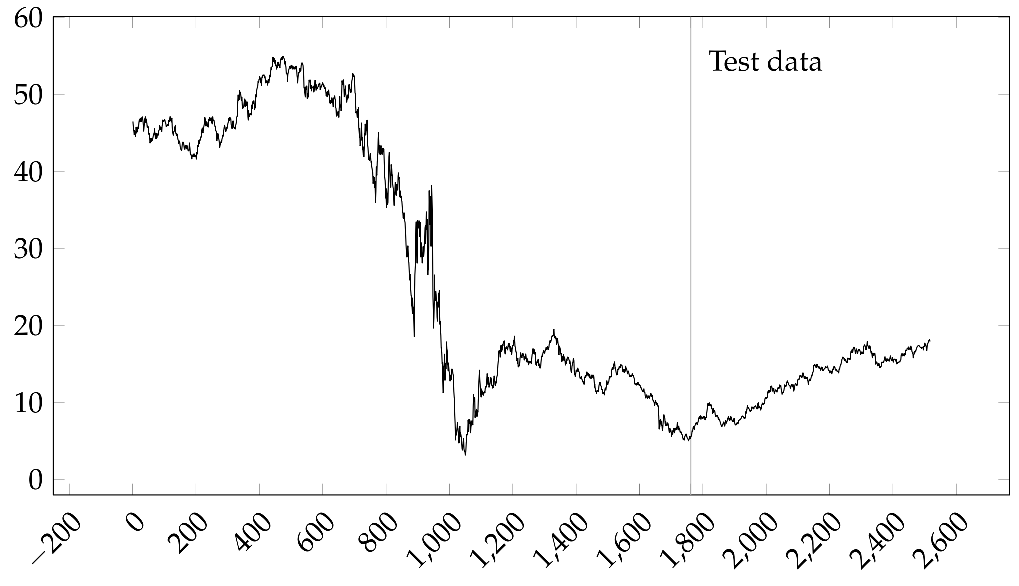


Figure 3.3: BAC daily closing price development between 1-1-2005 and 1-1-2015.

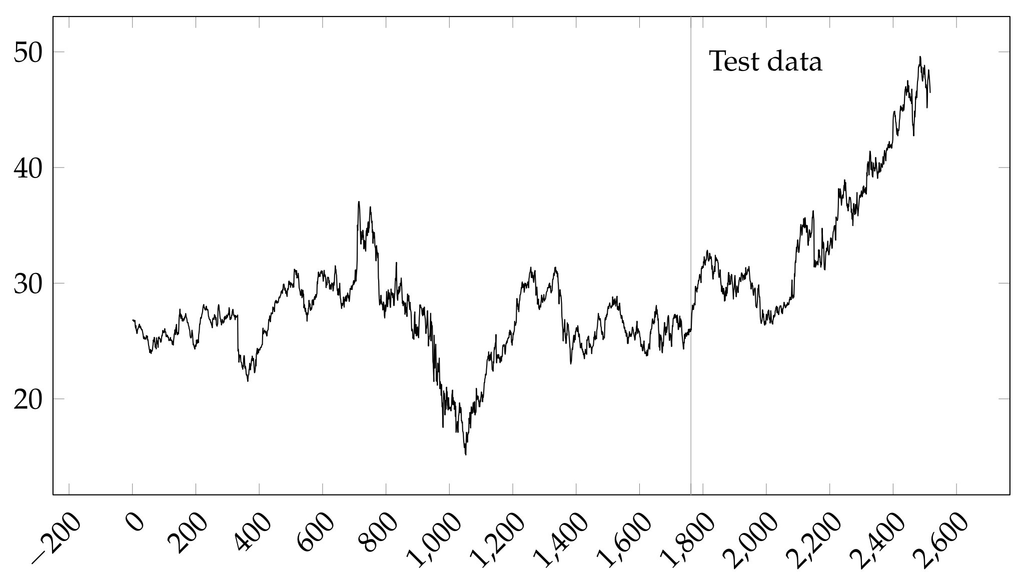


Figure 3.4: MSFT daily closing price development between 1-1-2005 and 1-1-2015.

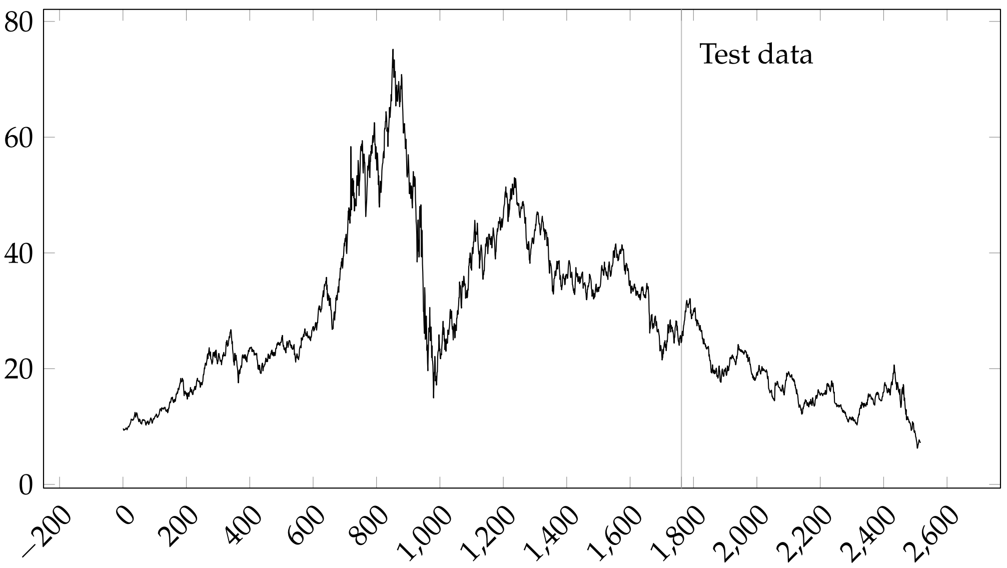


Figure 3.5: PBR daily closing price development between 1-1-2005 and 1-1-2015.

Petróleo Brasileiro S.A. - Petrobras With upward of 135 production platforms and 31000 kilometers of pipeline [38], Petróleo Brasileiro S.A. - Petrobras (PBR) is a significant actor in the oil industry. As seen in figure 3.5, the training set is largely volatile, containing both bull and bear segments. Test samples follow a general downward movement. The data has been adjusted for two stock splits in the training samples, both of which doubled the number of shares. The adjustment procedure is detailed in section 3.3.4.

Exxon Mobile Corporation According to its website [12], Exxon Mobile Corporation (XOM) is the largest publicly traded international oil and gas company in the world. Listed on the NYSE and categorized in the basic minerals sector, the company has grown about twice its original value since 2005, as seen in figure 3.6. On the other hand, the development has been volatile, including a solid period of bearish conditions halfway through the training set. Test samples trend in favor of the company, although with some volatility.

3.3.2 Granularity

As mentioned in section 2.1.4, stock market data are available at different resolutions. A large resolution provides more details, at the expense of being noisier. A courser granularity like weekly data points would spare the model from the noise found in intra-day data. On the other hand, it will greatly limit the number of training samples, since each time step is a whole week. A compromise is to consider daily data points, where we are still not exposed to intra-day noise, and also able to produce a reasonable

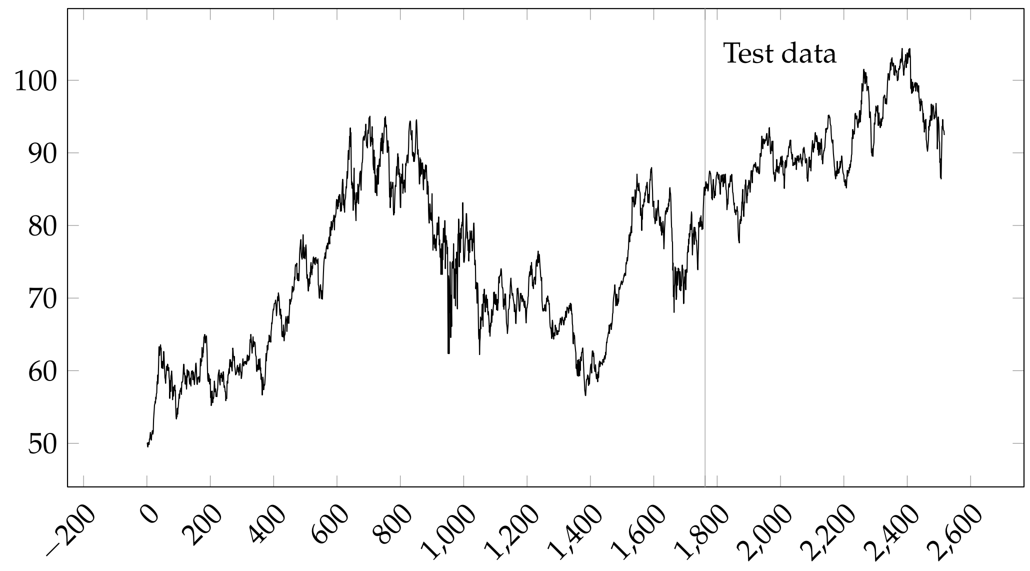


Figure 3.6: XOM daily closing price development between 1-1-2005 and 1-1-2015.

amount of training cases.

Daily data does have its drawbacks. Firstly, every day is not the same. Market participants that follow strategies based on weekly data for example, are likely to enter the market in a weekly, cyclical pattern. If this is true, then accounting for the day of the week might impact a model's chances of a successful prediction. Secondly, psychological factors also play a role in people's decisions; perhaps Mondays turn out to be generally more bearish than Fridays, due to the proximity to weekends. Thirdly, daily data are affected by the gaps created in the market when the exchanges are closed, as described in section 2.1.4. This impacts every data point, but perhaps to a larger extent for Mondays, where the market has been closed the whole weekend. Other potentially predictable factors that are not exclusive to daily data, include larger cyclical patterns, seasonality and company events.

3.3.3 Preprocessing

In an attempt to alleviate some of the issues associated with a daily data resolution, each training sample should include information about the *date*. Because most of the problems are associated with the day of the week, encoding date information as classes corresponding to each day of the week is a reasonable approach. Alternatively, we might encode the day as a single floating number, where a value close to zero or one translates to a day close to Monday or Sunday, respectively. The former solution is applied in this work, following an intuition that such an encoding is easily processed by the models, explicitly emphasizing the exact day of the week.

The share value of a stock changes over time, and even the largest companies started off small. By training a model on raw data directly, the model may not be able to yield reliable predictions later on when the general prices are at another level. In an effort to make new data recognizable, data are considered as a *percentage of change*, rather than actual price points.

Looking at Change

Figure 3.7 illustrates how the data are initially prepared. O, H, L, C and V stands for open, high, low, close and volume respectively, while $D1 ... D5$ are the binary variables representing each working day. The value an arrow points to can be calculated by measuring its change in percent with respect to the value where the arrow begins. Assuming we have two values a and b, and an arrow connecting them from a to b, we can calculate the change p from a to b as follows: $p = \frac{b-a}{a}$.

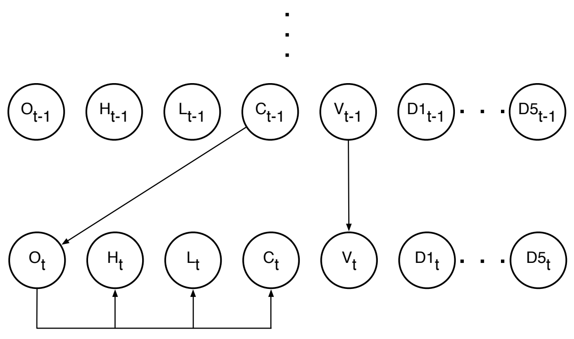


Figure 3.7: Some values are seen as a percentage of change with respect to another variable. The figure shows calculation of a supervised input vector.

The data could have been preprocessed in a straight forward temporal manner, where a variable at time t is calculated with respect to itself at time $t-1$. However, as the opening price is a continuation from the closing price the day before, the temporal connection from C_{t-1} to O_t in figure 3.7 reflects that relationship. Similarly, high, low and close values are local within a day, and are therefore captured by spatial connections from O_t. Volume uses another metric than the prices do, and should be calculated with respect to itself in the previous time step. Note that $D1 ... D5$ are not touched at all, leaving them in their raw binary states.

Closing prices make up ideal targets for the supervised models, and need translation similar to the input vectors. Since the most recent information available during prediction is the closing price, target closing prices should be seen in relation to this value. In other words, changes in target values are calculated in a straight forward temporal manner.

RBMs do not make use of target values directly, but rather have them contained within each input vector, see section 2.5.1. For these models, closing prices in input samples are calculated with respect to themselves in earlier time steps, just like the target values for supervised models. Calculations for the other variables remain unchanged.

Normalization

The second and final step of preprocessing is ensuring that the data has a mean of zero and a standard deviation of one. This is especially important for ANN models, to prevent large input values from slowing down the learning process; large values imply large weights, which require a long time to fit. Alternatively the learning rate could be increased, but in our case we have a mixture of large and small values (volume and price). An increase in learning rate would make learning from the price variables unreliable. Unlike the first step of preprocessing discussed in section 3.3.3, this is a well-known preprocessing technique for neural networks. Assuming a matrix p that consists of multiple variables i through time j, the normalized value x_{ij} is calculated from p_{ij} as follows:

$$x_{ij} = \frac{p_{ij} - \overline{p_i}}{\sigma_{p_i}} \tag{3.5}$$

where $\overline{p_i}$ and σ_{p_i} is the mean and standard deviation of variable i through time, denoted p_i. Input vectors and target values are normalized independently.

3.3.4 Adjusting for Splits

Companies sometimes decide to split their shares, as mentioned in section 2.1.4. When a split occurs, the number of shares is multiplied, while the price per share is divided accordingly. Unless the data are *adjusted*, we are left with large gaps in the data samples. For the stock selection used in this study, splits occur only in PBR.

As mentioned in section 3.3.3, our data samples contain four variables for price, one for volume and five for day encoding. Both price variables and volume are affected by stock splits. Assuming a two-for-one split as observed in PBR, every price value from the beginning to the day of the split have to be halved. Similarly, the volume has to be doubled to reflect the larger amount of shares. Adjustments are done prior to any preprocessing, to prevent factoring in the transformation of the data.

3.4 Models

The models and their hyper-parameters are presented in this section. The intuition behind the hyper-parameters is discussed throughout chapter 2, usually under the section associated with the model in question.

3.4.1 Support Vector Regression

Support Vector Regression (SVR) is included in the experiments as baseline for several reasons. It is a well-known deterministic model that makes an educated guess at the optimal solution, as opposed to accepting any valid solution. Furthermore it has few hyper-parameters, meaning optimizing it is less cumbersome.

C	ϵ	Kernel	σ
1000	$3e^{-5}$	Gaussian	2.5

Table 3.3: SVR hyper-parameters.

Architecture	Learning rate	Epochs	Dropout	Initialization
$100, 20, 1$	0.001	1000	0.5	RBM pre-training

Table 3.4: FFNN hyper-parameters.

Using a Gaussian kernel, three hyper-parameters are available. A combination of trial and error and grid search on GSPC (section 3.2.2) was applied to find reasonable values. The values are listed in table 3.3. Even though the intuition behind the hyper-parameters is discussed in section 2.4.5, comments regarding the values in table 3.3 will not be made here; a value might appear high or low, but its effect is defined in conjunction with the data. Analyzing the effect of SVR parameters on stock data is beyond the scope of this thesis.

3.4.2 Feed Forward Neural Network

A modern Feed Forward Neural Network (FFNN) trained with dropout and initialized with unsupervised pre-training can allow for deeper and more complex architectures, see sections 2.3.3 and 2.5.3. After attempting several approaches. However, preliminary tests indicated that simpler architectures seem to produce better and more consistent results.

Table 3.4 lists the hyper-parameters. Each number under *architecture* refers to the number of neurons in a given layer, starting at the input layer. With each data point consisting of ten values and the model requiring ten lagged inputs (section 3.2.3), the input layer must contain 100 neurons. One hidden layer of 20 neurons is used, and finally the single output neuron. Deeper architectures were tested, but the added complexity only led to worse results on the test set.

The learning rate was set low at 0.001 to allow stable training through 1000 epochs. Furthermore, the model was initialized by unsupervised pre-training (section 2.5.3), as well as applying a dropout rate of 0.5 during fine-tuning with backpropagation (section 2.3.3). These are all measures to help prevent overfitting the data, with the number of epochs and learning rate being especially delicate. Learning rate is decreased by a factor of ten when fine-tuning the model, to prevent deviating too far from the initialized state.

3.4.3 Echo State Network

Recurrent Neural Networks (RNNs) were created specifically to model temporal data, but conventional training with Backpropagation Through Time (BPTT) introduces several problems, as mentioned in section 2.4.1. Echo State Network (ESN) is an RNN that reduces learning to a linear

Reservoir capacity	Input scaling	Spectral radius
500	$[-1, 1]$	0.8

Table 3.5: ESN hyper-parameters.

optimization problem by leaving internal weights in their initial, random state. Being a recurrent network with efficient learning makes the ESN a relevant research candidate for stock prediction.

Given the efficient learning of ESNs, adjusting hyper-parameters by trial and error is more feasible. [31] recommends initially adjusting all parameters but the reservoir capacity, which should be fixed at a relatively small size to make the trials more computationally tractable. As long as the reservoir is at least larger than the number of values the network should remember, the other hyper-parameter values should effortlessly scale to larger reservoir sizes later on. Following these advices, the initial reservoir capacity is fixed at 100. This corresponds to ten time steps of ten variables. As seen in table 3.5 the reservoir is finally adjusted to 500 neurons.

In order to inhibit activations in the network due to far-reaching temporal connections, the spectral radius is set to 0.8. Input scaling defines the range from which the model samples its input weights, and because the input is normalized around zero with a standard deviation of one (section 3.3.3), the weights will also be sampled within the interval $[-1, 1]$.

The model contains no output feedback, as it is supposed to react to the input values directly, rather than generate signals based on its previous predictions.

3.4.4 Conditional Restricted Boltzmann Machine

Due to their success for modeling human motion, the Conditional Restricted Boltzmann Machine (CRBM) is selected for evaluation. The stochastic nature of CRBMs combined with noisy stock data makes learning a delicate process. Limiting learning rate to 0.0001 allowed for 200 epochs of training, as seen in table 3.6. Achieving more epochs by using slow learning was aimed for, following the intuition that RBMs are stochastic and might need additional iterations to converge.

Two stacked CRBMs comprise the final model, together providing one visible and two hidden layers. The lower CRBM uses six data points to generate a hidden state of five neurons. In other words, it takes 60 values and compresses them to five binary values. Six such binary states are needed for the second CRBM to infer its hidden state of five binary neurons.

As detailed in section 2.5.2, predictions are made by locking the past five data points in place while letting the sixth be generated from some initial state. In the experiments this initial state was zero. Because of the pre-processing we know that zero represents the average of each variable, and thus is a reasonable starting point.

Even though the network requires only five data points in the visible layer to infer a hidden state, five such states are also needed at the second

Architecture	Learning rate	Epochs
$6 \cdot 10, 6 \cdot 5, 1 \cdot 5$	0.0001	200

Table 3.6: CRBM hyper-parameters.

Architecture	Epochs	Learning rate
$3 \cdot 10, 4 \cdot 5, 5 \cdot 3, 1 \cdot 1$	100	0.001

Table 3.7: TDNN hyper-parameters.

layer. In order to generate them the model therefore needs a total of ten data points.

3.4.5 Time-Delay Neural Network

Following the intuition that time series data can be decomposed into smaller components, the input layer of Time-Delay Neural Networks (TDNNs) are delayed three steps. [51] also used three input delays for modeling speech, arguing that three steps would be enough to capture the typical phoneme. Even though phonemes are exclusive to speech, dividing the data into smaller chunks might still be beneficial. Intuitively these components might represent features like slight price increases, or volatile, neutral movement approaching the weekend, however the exact features learned remain hidden.

Five neurons make up a time step in the first hidden layer, in order to start narrowing the network. In other words, $3 \cdot 10 = 30$ input values translate into five features. The second hidden layer requires four steps of the first hidden layer and extracts three features. That makes each of the second hidden layer neurons connected to $4 \cdot 5 = 20$ nodes. The output node takes five steps from the second hidden layer, totaling $5 \cdot 3 = 15$ input units. In order to produce enough lagged features within the network, the actual number of neurons within each layer is as follows: 100, 40, 15, and 1. Note that 100 input neurons amounts to ten time steps of data.

Large deviations were observed while optimizing hyper-parameters for the TDNN. Learning rate was lowered to 0.001 in an attempt to stabilize the learning process.

3.4.6 Convolutional Neural Network

Section 2.4.4 discussed Convolutional Neural Networks (CNNs) in the context of image recognition. These models can also be applied for time series prediction however, assuming each data vector is correctly formatted; by considering the number of time steps as the width of an image, and the number of variables as the image height, the CNN can learn from time series data.

Layer configuration of the CNN is shown in table 3.8. Using a history of ten time steps, the input layer is a ten-by-ten matrix. The first convolutional layer uses a receptive field spanning two steps of input, in

Architecture	Momentum	Dropout	Epochs	Learning rate
I, C, P, C, F, O	0.9	0.5	500	0.01

Table 3.8: CNN hyper-parameters.

other words a ten-by-two input window. This ensures that entire spatial states are contained within each receptive field, and is necessary because the variables have different meaning; we can not assume that the same features used to describe price changes are meaningful also for volume or the day encoding. Convolutional stride is one, and 16 feature maps are produced.

Since the problem is restricted to ten time steps, only one pooling layer is applied. Max-pooling is applied in non-overlapping regions of size two-by-one, thereby halving the dimensionality. Note that the first convolutional layer reduces spatial states to one feature every time step, and explains why only temporal dimensionality is reduced.

A second convolutional layer follows the pooling layer, using a two-by-one kernel and outputting 16 feature maps. 16 fully connected neurons follow, before a single output node concludes the model. The number of epochs was set relatively high at 500, relying on weight sharing in convolutional layers and dropout of 0.5 in fully connected layers to prevent overfitting. Momentum was set to 0.9, as it seemingly improved performance.

3.5 Limitations

There are several factors that affect how this study is carried out. Because these limitations directly affect the performance of the models, they should be kept in mind when digesting the results.

3.5.1 Limited Depth

An effort is put into including multiple models for a broader comparison. This leaves fewer resources to optimize each model individually, creating a trade-off between a broad versus deep scope. Furthermore, the learning efficiency with respect to computational needs vary greatly between libraries and learning algorithms: ESNs are very fast due to their simplified training problem, discussed in section 2.4.2. The CNN on the other hand could not be tuned as much, due to considerable training times. This is one of the reasons why only one time series is used to optimize hyper-parameters, as described in section 3.2.2.

Despite differing training times and other dissimilarities between models, care is taken to optimize model hyper-parameters as fairly as possible. Furthermore, favoring breadth over depth will likely not display the full potential of each model. For these reasons, models should be judged by their relative performance to each other, rather than their absolute performance.

3.5.2 Simplified Profit Estimation

When trading strategies are employed, every trade is associated with several fees: The market might demand a less favorable price than what you ask for, a third party like a broker typically require commission fees and there might be tax fees as well. These aspects are not accounted for, and the profits reported in this study should not be understood as anything else than a general indication of model performance. It should be noted that the stocks in this study represent major companies that are liquid in the market. As discussed in section 2.1.1, liquidity reduces chances of significant slippage.

Chapter 4

Experiments

Results gathered from the experiments are introduced and analyzed in this chapter. Section 4.1 describes how the results should be read, before they are presented in section 4.2. Section 4.3 further reflects upon the results in a broader setting, and compares models to each other.

4.1 How to Read the Results

The results for each model consist of three parts: Performance plots, a table of aggregated results and a textual discussion.

4.1.1 Performance Plots

Three plots are included for each kind of model: One for the postprocessed, actual predicted prices, another for predicted changes in price and a final plot for how the profits develop through the test period. Because of the large quantity of results gathered, plots only cover single-step predictions for MSFT, and are largely intended to leave a visual impression of the models.

Postprocessed Predictions

By postprocessing output of a model, we are left with the actual price values the model has predicted. Plotting every prediction for a test set renders the plots almost unreadable, due to the large number of values. In order to still get an impression of what the results look like, a segment of the last 100 predictions are plotted.

Regarding figure 4.1a as an example, two lines can be seen. The black line represents ideal prices at a given time, while the colored line is the value predicted by the model. Although the two lines might seem similar, remember that the model is only predicting one day at a time. Each prediction in the plot should therefore be seen in context with the previous ideal value, which is when the prediction was made.

As detailed in section 3.2.3, non-deterministic models are trained and tested ten times, to extract average results and standard deviation. To keep the plots clean, a representative model is chosen at random within each group, plotting its predictions.

Because all performance measures are derived from postprocessed predictions, every performance aspect can be read from this plot. The measures discussed in section 3.2.4 will convey most of it more effectively however, and are discussed further in section 4.1.2. Although the postprocessed prediction plots are not intended for extended analysis, irregularities and noisy behavior are effortlessly identified by visually inspecting these plots.

Price Changes

The second figure plots every test set prediction made by the model, after a slight transformation: Each value represents the predicted percentage of change since the previous day, shown in figure 4.1b. This plot is also sampled from one model in each group, for non-deterministic models.

By inspecting price change plots we may observe how large the predicted price changes are. Aggressively volatile predictions are likely to result in low MSE, if the stock usually moves more conservatively. How balanced the predictions are in terms of bull and bear development is also observed in the plot, where many predictions above zero corresponds to a high amount of bull predictions. Lastly, price change plots may reveal sudden changes of behavior during the test set. If these are observed, it might indicate a problem with the model's ability to generalize unseen samples.

Profit Development

Generated profit is an important factor in this study. However, unreliable models might end up with acceptable profits, even though it lost almost all assets halfway through the test set. Figure 4.1c plots equity development for the SVR on MSFT. The model always has an initial balance of 100 units, and an increase to 167.9 corresponds to 67.9% profit.

These plots give an impression of how reliably profits are generated, and significant drops in equity are considered dangerous. For neural networks, all models within each group is plotted, see figure 4.2c for an example. A large deviation between models is an indication of unreliable profit generation.

4.1.2 Aggregated Results

To deal with the volume of data gathered from the experiments, results are aggregated and summarized in tables like table 4.2. For non-deterministic models the reported values are averages between the respective groups of models. To indicate reliability of the results, the standard deviations are also included in parentheses as seen in table 4.3. For information regarding each indicator see section 3.2.4.

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.560	1.000	0.000	62.646
IXIC	0.560	1.000	0.000	142.600
BAC	0.546	1.000	0.000	23120.567
MSFT	0.510	1.000	0.000	1539.572
PBR	0.481	1.000	0.000	3345231.574
XOM	0.497	1.000	0.000	203.512

Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.600	1.000	0.000	18.312
IXIC	0.580	1.000	0.000	33.781
BAC	0.559	1.000	0.000	1208.856
MSFT	0.539	1.000	0.000	215.211
PBR	0.444	1.000	0.000	72657.980
XOM	0.515	1.000	0.000	41.708

Table 4.1: Summarized ideal results.

4.1.3 Textual Description

A characterization of the results is included to highlight aspects of interest with each model. Performance is analyzed and compared to the baseline model in the study, namely the SVR algorithm.

4.2 Results

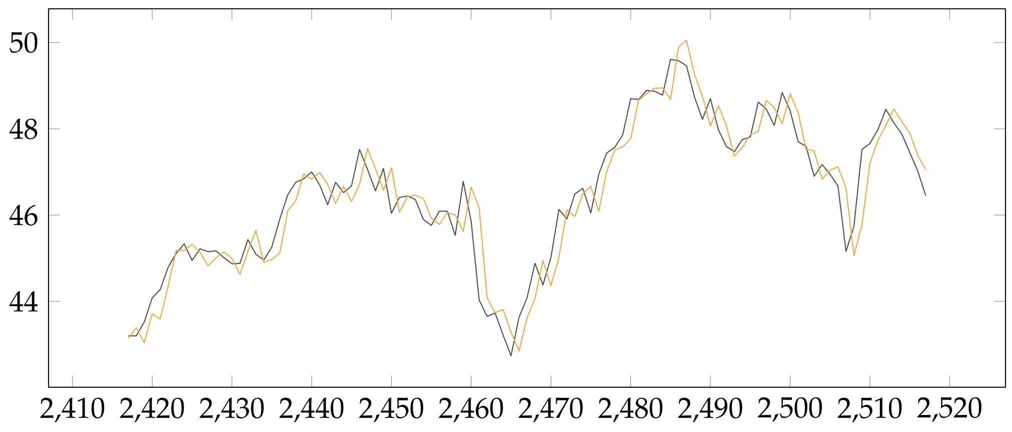
For convenience the ideal results are listed in table 4.1 for each time series. Explanations of the different table columns are given in section 3.2.4.

4.2.1 Support Vector Regression

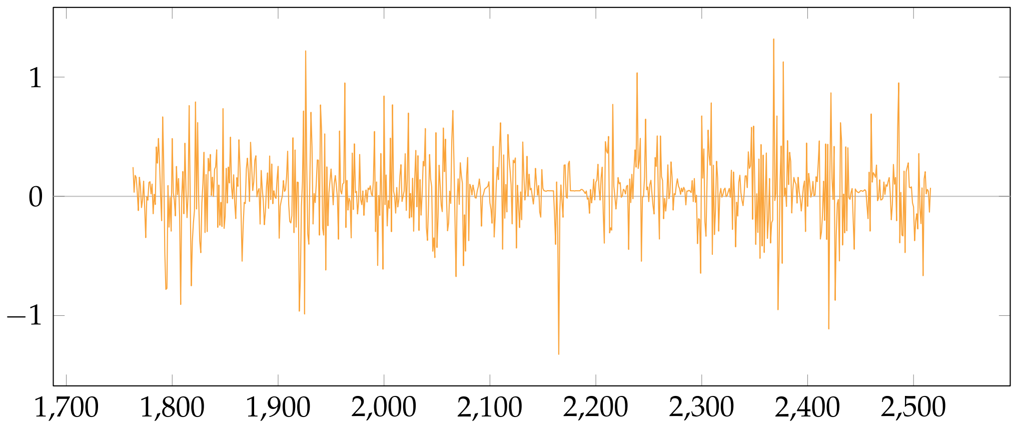
As mentioned in section 3.4.1, the Support Vector Regression (SVR) model represents the baseline of the experiment. With few exceptions, a generally positive performance for both directional accuracy and profit sets the bar for other models.

Single-step prediction

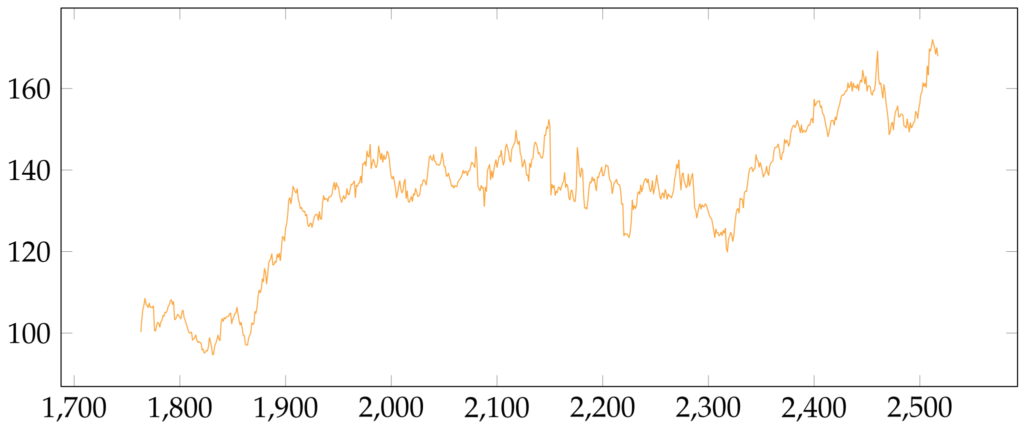
Single-step results for the SVR on MSFT are shown in figure 4.1. Looking at figure 4.1a, the colored prediction line follows the black ideal line closely, never predicting extreme movement in either direction. Predictions expressed as percentage of change are plotted in figure 4.1b. The model seems balanced between bull and bear predictions, as observed by the almost symmetric shape around zero percent change. Development of equity balance is plotted in figure 4.1c, and can be described as increasing in steps, never losing much at a time.



(a) A section of SVR postprocessed predictions.



(b) SVR predictions, expressed as percentage of change.



(c) SVR equity development.

Figure 4.1: SVR prediction of MSFT

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.563	0.521	167.965	0.198
IXIC	0.595	0.510	1083.186	0.201
BAC	0.499	0.517	0.049	0.011
MSFT	0.607	0.535	0.236	0.679
PBR	0.563	0.491	0.221	-0.215
XOM	0.640	0.523	0.832	0.163

Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.593	0.530	330.162	0.157
IXIC	0.616	0.530	2197.454	0.565
BAC	0.507	0.485	0.099	-0.205
MSFT	0.653	0.523	0.481	0.273
PBR	0.589	0.491	0.470	-0.312
XOM	0.702	0.527	1.588	0.200

Table 4.2: Summarized results for SVR.

Table 4.2 displays summarized results for each time series. GSPC is included, although its function as a tuning set should be kept in mind. The model is reasonably balanced between upward and downward predictions, favoring positive change for every series except BAC. Looking back at the plot of BAC in figure 3.3, it is clear that this share has endured bear conditions during the period used for training. This explains why the model is careful to predict positive change, and with bull development in the test data the result is a weak single-step profit for BAC. Negative profits are observed for PBR predictions, losing 21.5% during the three year test period. The high bull ratio of 56.3% poorly reflects the bear market conditions observed in the PBR test set; ideal bull ratio is 48.1%, as seen in table 4.1. Directional accuracy falls to 49.1%.

Double-step prediction

Performance is slightly decreased for double-step prediction, given how profits are improved only for IXIC and XOM, and reduced for the rest. BAC performance is degraded significantly: The SVR model is unable to correctly predict the price direction more than 48.5% of the time, and profit falls to -20.5%. Profits are significantly reduced for PBR as well, further reducing them to -31.2%.

4.2.2 Feed Forward Neural Network

Unlike most models in the experiment, the Feed Forward Neural Network (FFNN) does not model temporal connections explicitly. Interestingly, this generic, fully connected architecture outperforms baseline on several data sets: Improved profits and mostly improved directional accuracy

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.737 (0.038)	0.537 (0.006)	923.361 (34.614)	0.389 (0.092)
IXIC	0.709 (0.036)	0.532 (0.010)	4927.108 (137.016)	0.160 (0.121)
BAC	0.890 (0.039)	0.536 (0.006)	0.562 (0.028)	0.537 (0.234)
MSFT	0.932 (0.013)	0.520 (0.005)	0.524 (0.025)	1.196 (0.277)
PBR	0.844 (0.011)	0.466 (0.003)	1.478 (0.060)	-0.892 (0.023)
XOM	0.733 (0.035)	0.520 (0.008)	8.291 (0.404)	-0.002 (0.090)

Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.663 (0.035)	0.531 (0.006)	2502.478 (72.492)	0.312 (0.060)
IXIC	0.638 (0.040)	0.533 (0.007)	13505.858 (417.278)	0.134 (0.080)
BAC	0.944 (0.020)	0.550 (0.005)	0.727 (0.060)	1.208 (0.265)
MSFT	0.853 (0.018)	0.542 (0.006)	1.159 (0.057)	1.005 (0.130)
PBR	0.829 (0.009)	0.434 (0.003)	3.604 (0.091)	-0.859 (0.014)
XOM	0.623 (0.023)	0.533 (0.007)	24.947 (0.771)	0.026 (0.068)

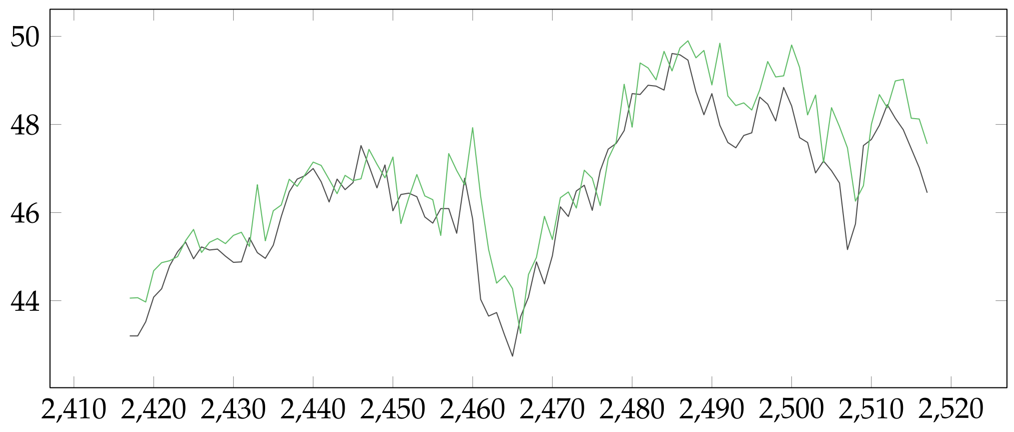
Table 4.3: Summarized results for FFNN.

is observed on GSPC, BAC and MSFT. Significant underperformance is seen for PBR however, with a profit of -89.2% and being directionally inaccurate.

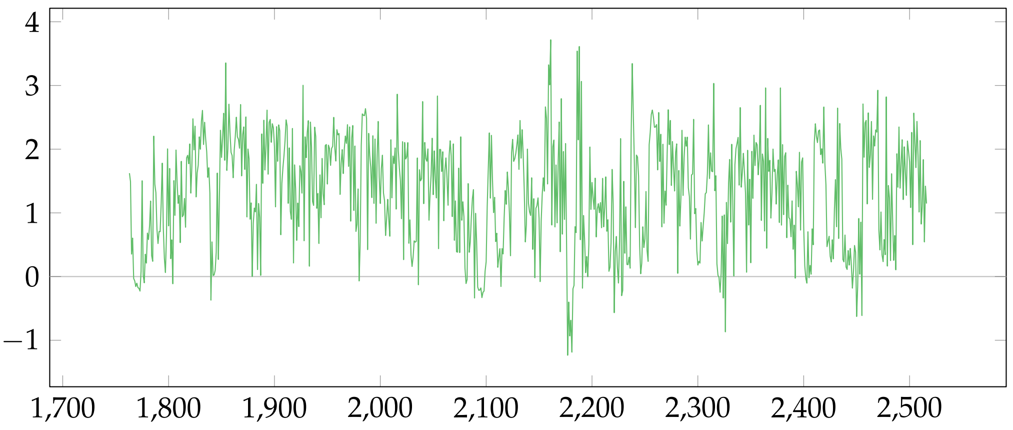
Single-step prediction

As seen in figure 4.2a, the FFNN seems to produce more volatile predictions compared to baseline. As expressed in figure 4.2b the model is biased towards upward movements, producing few predictions corresponding to negative price change. Because MSFT is bullish within the predicted period this does not reduce its performance in terms of profit, although compared to the ideal results for MSFT in table 4.1 it is clearly too bullish. Figure 4.2c pictures a steadily increasing equity balance, with few deviations between models.

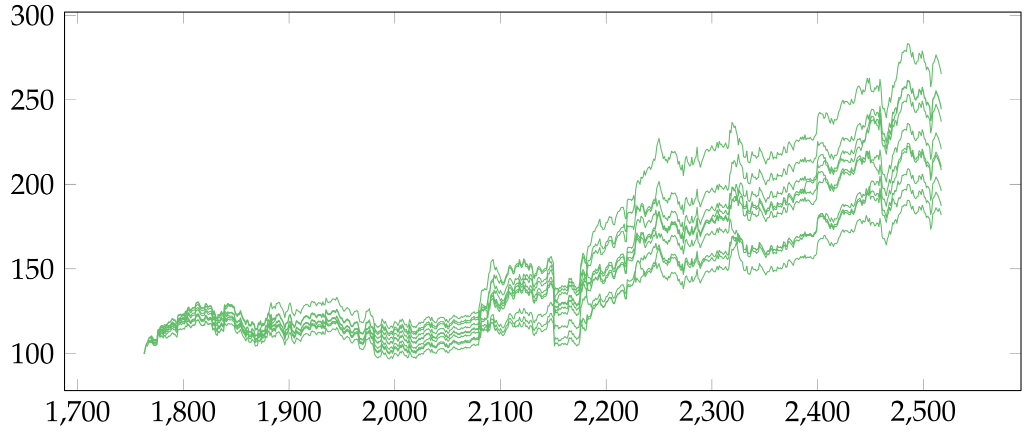
The imbalanced predictions observed for MSFT are improved for other time series, as seen by the bull ratios in table 4.3. Compared to the SVR baseline, it is still more bullish. Directional accuracies are worse than baseline for MSFT and PBR, while improved for the two indices and BAC. For PBR, a high bull ratio leads to poor directional accuracy and negative profits. MSE is significantly greater than baseline, due to large variations between outputs seen in figure 4.2b; where SVR outputs are centered on zero with some occasional spikes, the FFNN predictions deviate more strongly from the current price. Frequently predicting large price movements makes low MSE improbable, considering stocks rarely make such extreme developments for longer periods.



(a) A section of FFNN postprocessed predictions.



(b) FFNN predictions, expressed as percentage of change.



(c) FFNN equity development.

Figure 4.2: FFNN prediction of MSFT

Double-step prediction

Predicting two days ahead seems to enhance directional accuracy for FFNNs; certain averages are increased, while standard deviations for the most part are slightly lowered. PBR directional accuracy decreases, although profits are slightly improved. BAC sees a significant boost in profits, rendering it the most profitable of the time series. Interestingly, the SVR baseline reacts negatively to predicting double-step on BAC.

The tendency to predict bull price changes should be noted. Although it almost guarantees positive results for bull stocks like BAC and MSFT, this behavior will not resonate well with bear stocks. This was discussed in section 3.2.4, and is here exemplified by PBR. Even though the model is able to predict bull conditions for BAC, the performance could be questioned considering its inability to predict bear conditions for PBR.

4.2.3 Echo State Network

Being a recurrent network, the Echo State Network (ESN) provides a good architectural basis for learning stock data. The results gathered in this experiment however, are noisy with significant deviations in profits.

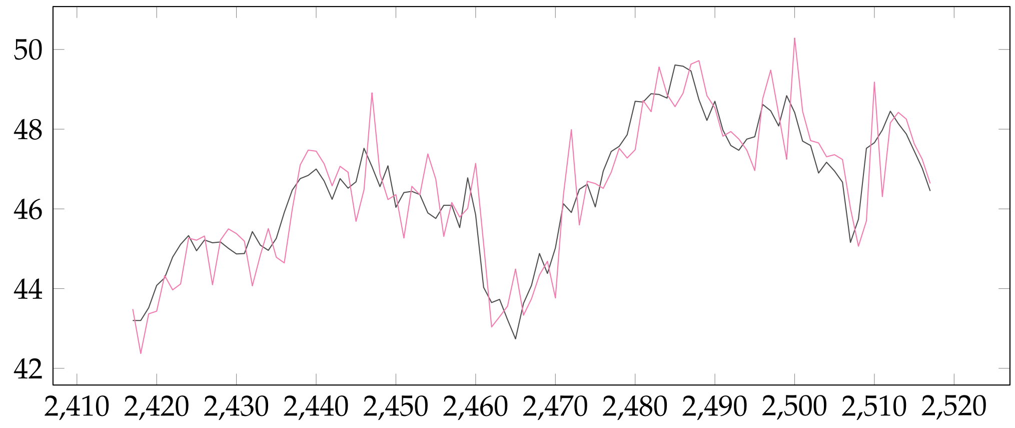
Single-step prediction

ESN predictions follow the ideal values in a noisier way than the SVR, as illustrated in the sample plot figure 4.3a. Figure 4.3b shows a balanced model in terms of bullish and bearish predictions. Although MSFT is bullish during the test period, only 51% of the movements during that period were bullish, see table 4.1. A balanced bull ratio is therefore not disadvantageous by itself. Equity development shows large deviations, as seen in figure 4.3c. Most models end up around or below the initial balance, with only one exception.

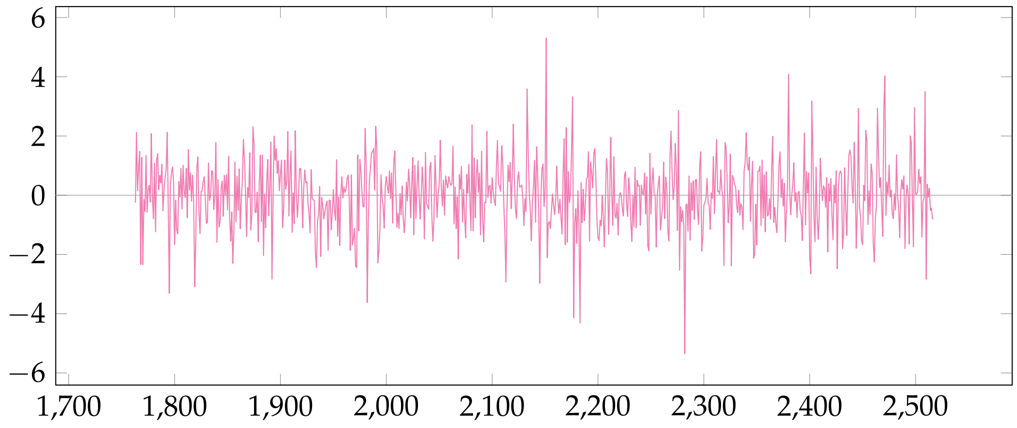
Looking at table 4.4, the ESN models have little variation in their bullishness, the majority averaging close to 50% bull predictions with low standard deviations. Average directional accuracies are also centered on 50%, with little deviation. This clearly underperforms to baseline, which only produced single-step directional accuracy below 50% for PBR. The approximated profits are mostly negative, with disproportionally large standard deviations. MSE values are approximately doubled compared to baseline.

Double-step prediction

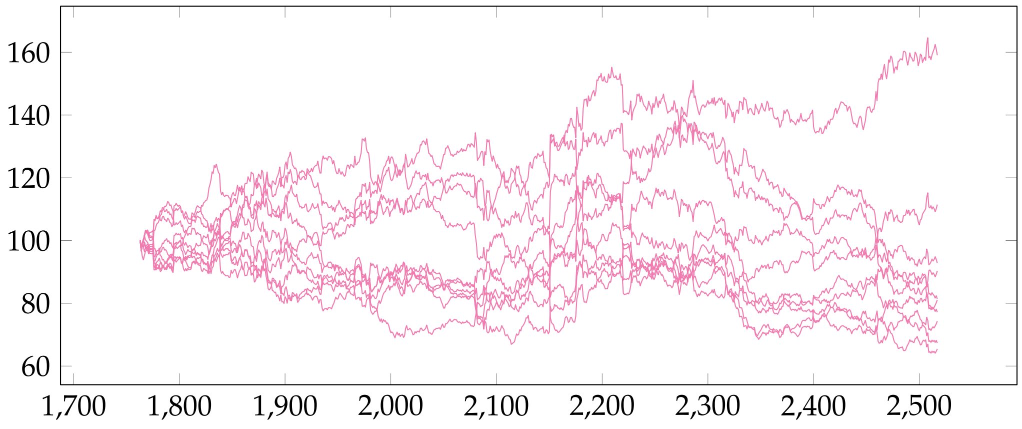
The seemingly noisy behavior is present also for double-step prediction, with higher MSE. Profits for PBR went from -37.7% to 12.9% in double-step. The standard deviation is almost four times larger than the average, rendering the positive profit meaningless.



(a) A section of ESN postprocessed predictions.



(b) ESN predictions, expressed as percentage of change.



(c) ESN equity development.

Figure 4.3: ESN prediction of MSFT

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.525 (0.024)	0.492 (0.015)	346.672 (20.545)	-0.042 (0.236)
IXIC	0.505 (0.024)	0.499 (0.024)	2238.943 (157.331)	0.048 (0.244)
BAC	0.482 (0.020)	0.500 (0.020)	0.161 (0.011)	-0.019 (0.357)
MSFT	0.514 (0.023)	0.504 (0.014)	0.411 (0.029)	-0.099 (0.277)
PBR	0.508 (0.026)	0.498 (0.011)	0.386 (0.026)	-0.377 (0.250)
XOM	0.509 (0.025)	0.503 (0.014)	1.644 (0.078)	-0.025 (0.209)
Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.521 (0.016)	0.500 (0.013)	667.317 (37.043)	-0.012 (0.123)
IXIC	0.508 (0.025)	0.505 (0.019)	4210.319 (155.073)	-0.017 (0.205)
BAC	0.485 (0.017)	0.507 (0.023)	0.293 (0.035)	-0.006 (0.277)
MSFT	0.508 (0.026)	0.486 (0.017)	0.836 (0.052)	-0.160 (0.182)
PBR	0.532 (0.018)	0.502 (0.013)	0.772 (0.037)	0.129 (0.507)
XOM	0.506 (0.035)	0.514 (0.014)	2.817 (0.173)	0.070 (0.172)

Table 4.4: Summarized results for ESN.

4.2.4 Conditional Restricted Boltzmann Machine

The Conditional Restricted Boltzmann Machine (CRBM) employs temporal connections and stochastic sampling to make its predictions. Although it produces positive profits on most time series, profits rarely outdo the SVR baseline; considering standard deviations for profits are larger than the average values for all but GSPC, the CRBM has difficulties competing with the SVR.

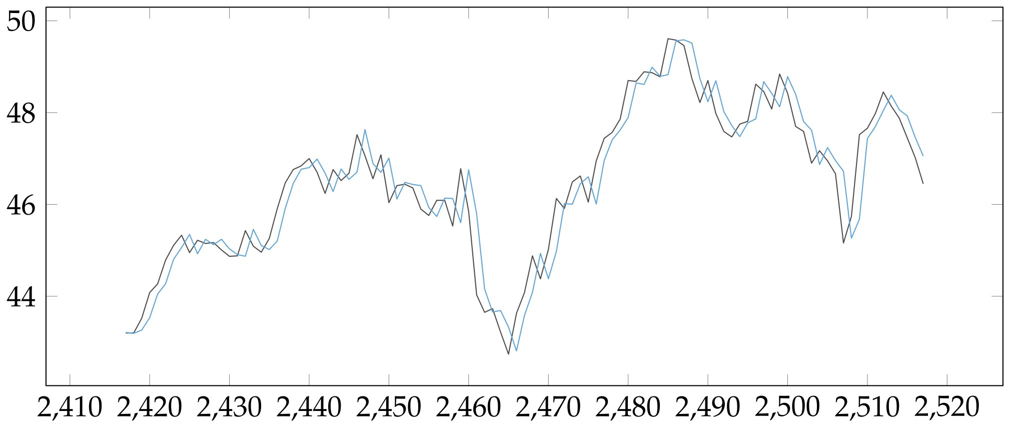
Single-step prediction

Figure 4.4 indicates that CRBMs are conservative in their predictions, mostly keeping within the interval $[-0.2, 0.2]$ as seen in figure 4.4b. This results in low MSE scores that consistently outperform baseline. The MSFT equity development seen in figure 4.4c is reasonable, with most models being able to retain some of the initial positive development.

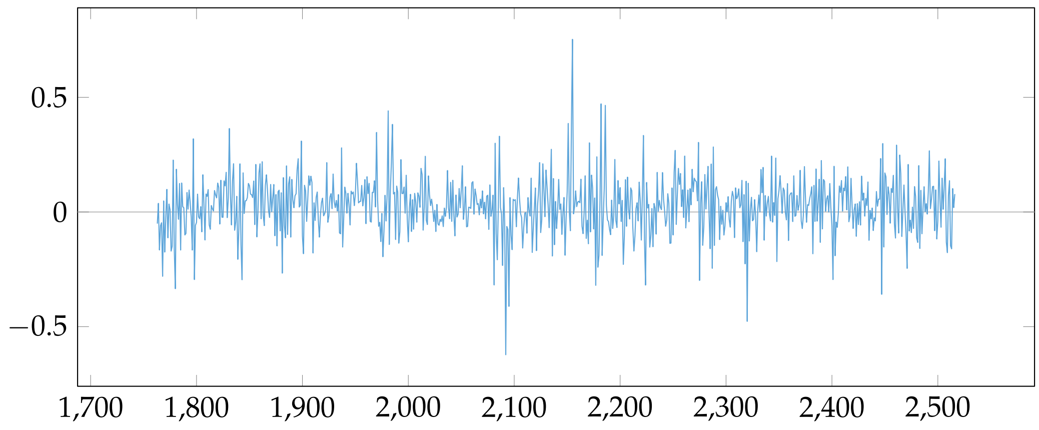
Some minor deviations are found in the bull ratios as seen in table 4.5, and directional accuracy is poorer than with the SVR. The CRBM struggles with BAC, PBR and XOM: Low bull ratio and directional accuracy, combined with large standard deviation in profits for BAC, while PBR and XOM profits are unable to develop positively. The large amount of bear predictions for BAC is understandable, considering its differences between training and test sets.

Double-step prediction

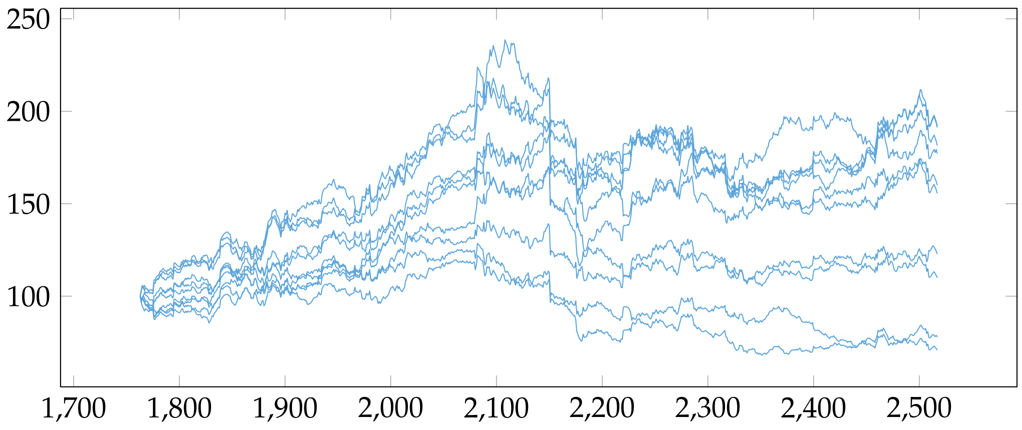
The model has a mixed response to double-step prediction. Bull ratio for BAC is significantly improved, and direction is predicted correctly more frequently. MSE scores are doubled however, and not all profits benefit



(a) A section of CRBM postprocessed predictions.



(b) CRBM predictions, expressed as percentage of change.



(c) CRBM equity development.

Figure 4.4: CRBM prediction of MSFT

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.557 (0.087)	0.513 (0.012)	148.190 (0.699)	0.201 (0.150)
IXIC	0.665 (0.080)	0.492 (0.016)	1007.079 (3.167)	0.030 (0.205)
BAC	0.274 (0.044)	0.486 (0.016)	0.043 (0.000)	0.212 (0.514)
MSFT	0.637 (0.089)	0.510 (0.013)	0.228 (0.001)	0.439 (0.457)
PBR	0.684 (0.060)	0.490 (0.009)	0.221 (0.001)	-0.722 (0.139)
XOM	0.547 (0.051)	0.505 (0.009)	0.755 (0.003)	-0.131 (0.113)

Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.639 (0.028)	0.530 (0.009)	296.489 (1.322)	0.025 (0.052)
IXIC	0.664 (0.045)	0.526 (0.008)	2066.262 (6.690)	0.175 (0.086)
BAC	0.424 (0.047)	0.513 (0.013)	0.089 (0.000)	-0.092 (0.163)
MSFT	0.614 (0.034)	0.539 (0.011)	0.472 (0.002)	0.611 (0.195)
PBR	0.598 (0.044)	0.484 (0.009)	0.472 (0.001)	-0.001 (0.178)
XOM	0.629 (0.048)	0.511 (0.009)	1.446 (0.005)	0.097 (0.095)

Table 4.5: Summarized results for CRBM.

either. MSFT stands out with 53.9% directional accuracy and 61.1% profit on average. Losses on PBR are essentially eliminated on average, although some deviation is still present.

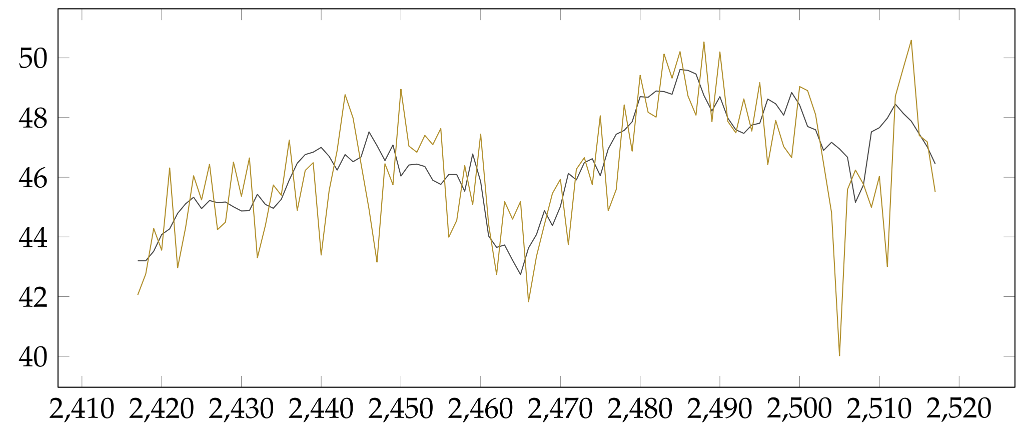
4.2.5 Time-Delay Neural Network

The Time-Delay Neural Network (TDNN) architecture was constructed to better model time series data, through temporal connections and shared weights. Profits are mostly positive, but considering the large deviations and on visually inspecting figure 4.5c, predictions appear unreliable.

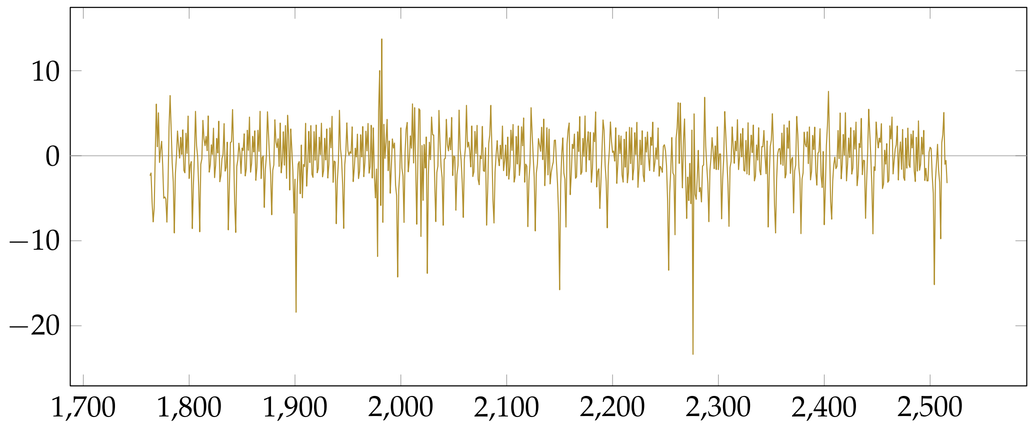
Single-step prediction

Figure 4.5a reveals how the TDNN makes volatile predictions that largely surpass ideal price development. In figure 4.5b, occasional volatile spikes are observed in the predictions. Predictions appear to be centered on zero. As seen in figure 4.5c, equity development seems inconsistent between models, with ultimately about half gaining and half losing equity.

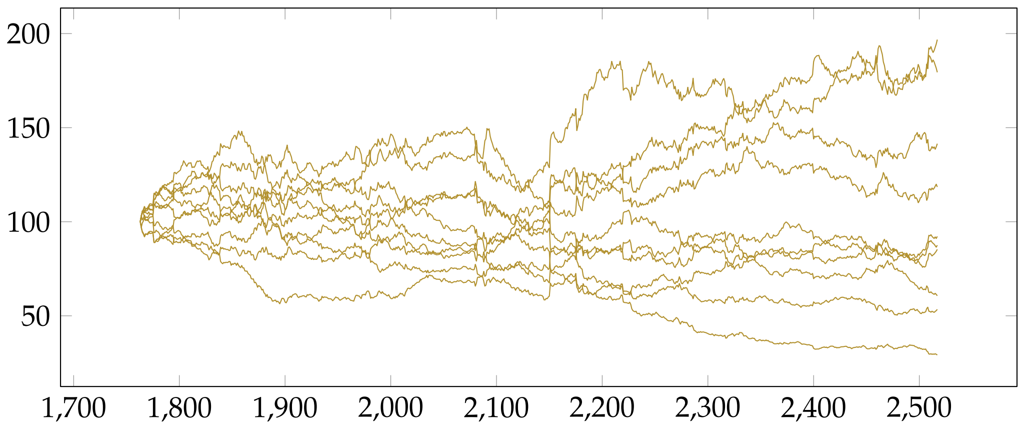
Table 4.6 confirms the balanced bull ratio, which is similar across all test sets. Directional accuracy is generally neutral, with all of them being outweighed by their standard deviations. MSE scores are large due to the volatile predictions. Although there is a majority of positive profits on average, inconsistent results between models render the profits questionable. PBR is the most extreme example, where the models generate -2.2% income on average with 76.7% standard deviation.



(a) A section of TDNN postprocessed predictions.



(b) TDNN predictions, expressed as percentage of change.



(c) TDNN equity development.

Figure 4.5: TDNN prediction of MSFT

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.525 (0.046)	0.501 (0.018)	1191.931 (555.872)	0.025 (0.222)
IXIC	0.526 (0.069)	0.498 (0.020)	6238.692 (3596.731)	0.028 (0.353)
BAC	0.523 (0.057)	0.498 (0.027)	0.556 (0.341)	-0.027 (0.489)
MSFT	0.506 (0.064)	0.500 (0.025)	1.385 (0.735)	0.044 (0.545)
PBR	0.507 (0.056)	0.503 (0.014)	1.304 (0.474)	-0.022 (0.767)
XOM	0.495 (0.074)	0.501 (0.016)	6.412 (3.492)	0.019 (0.240)

Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.495 (0.047)	0.492 (0.020)	2248.418 (1019.988)	-0.021 (0.144)
IXIC	0.527 (0.057)	0.505 (0.015)	12053.568 (4325.732)	0.018 (0.128)
BAC	0.477 (0.072)	0.509 (0.016)	1.044 (0.689)	0.089 (0.470)
MSFT	0.521 (0.051)	0.512 (0.020)	1.940 (1.098)	0.117 (0.244)
PBR	0.510 (0.036)	0.487 (0.014)	2.020 (0.522)	-0.238 (0.518)
XOM	0.495 (0.079)	0.499 (0.014)	11.609 (6.964)	0.001 (0.168)

Table 4.6: Summarized results for TDNN.

Double-step prediction

The most significant changes from single-step prediction include a profit increase to 11.7% on MSFT, while PBR profits are reduced to -23.8%. Profit deviations are otherwise slightly reduced, but still overshadow the average values. An increase in MSE is also observed.

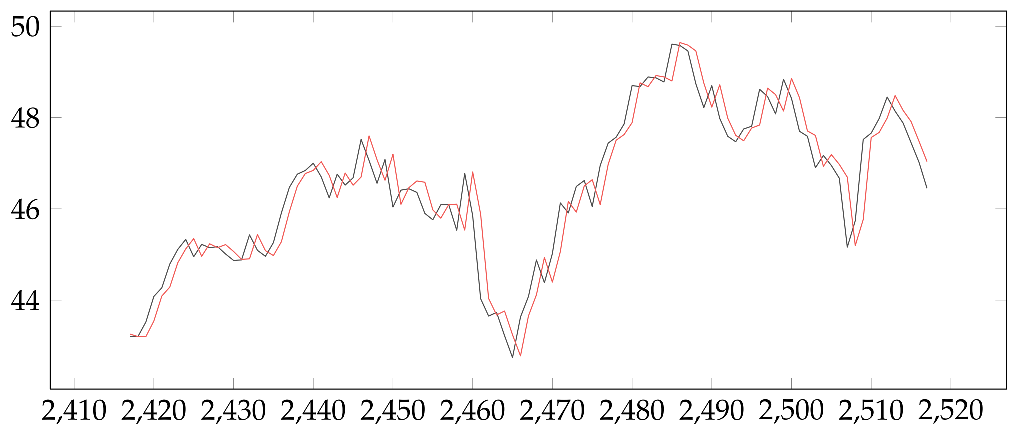
4.2.6 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are widely applied for image classification problems, but are also capable of time series modeling (section 3.4.6). Predictions were generally found to result in better directional accuracy and profits over baseline, while also keeping low MSE scores. Significant profit deviations and extreme bull ratios are observed, and raise some concern.

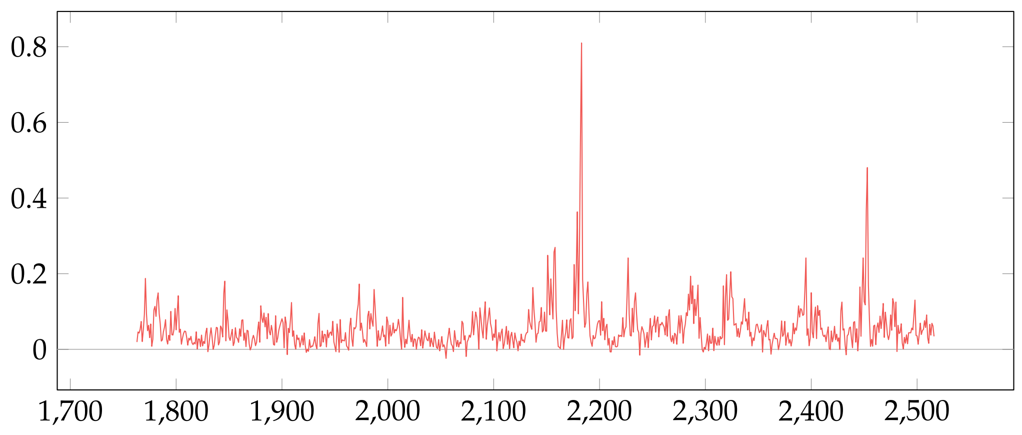
Single-step prediction

Figure 4.6a demonstrates how little the CNN deviates from the previous price in its predictions. The plot in figure 4.6b further illustrates how miniscule changes are predicted, with the majority ranging between zero to 0.1% change. Predictions for MSFT are heavily biased towards positive change. Equity development follows a relatively steady upward trend, as seen in figure 4.6c. Little deviation is observed between models.

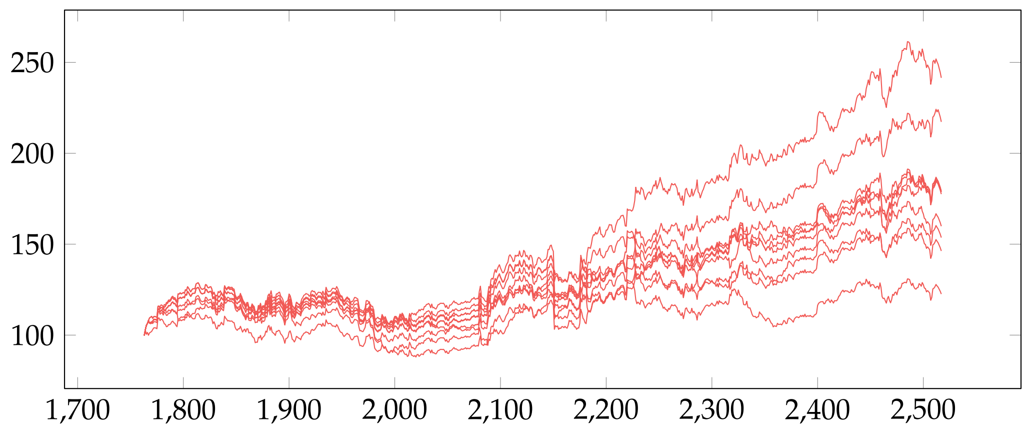
Looking at the single-step bull ratios in table 4.7, it is clear that the CNN favors bullish predictions for many of the time series, although being somewhat conservative for BAC. The most extreme bull ratio is observed for PBR, which incidentally is the only time series that is in decline during the test period. Directional accuracy is better for indices



(a) A section of CNN postprocessed predictions.



(b) CNN predictions, expressed as percentage of change.



(c) CNN equity development.

Figure 4.6: CNN prediction of MSFT

Single-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.902 (0.141)	0.543 (0.018)	146.285 (0.398)	0.408 (0.186)
IXIC	0.939 (0.048)	0.552 (0.009)	1000.697 (4.811)	0.571 (0.241)
BAC	0.644 (0.171)	0.511 (0.020)	0.043 (0.000)	0.325 (0.640)
MSFT	0.921 (0.055)	0.510 (0.011)	0.227 (0.001)	0.755 (0.342)
PBR	0.964 (0.046)	0.480 (0.003)	0.220 (0.001)	-0.666 (0.130)
XOM	0.938 (0.052)	0.499 (0.007)	0.746 (0.006)	0.017 (0.136)

Double-step prediction				
Time series	Bull ratio	Dir. accuracy	MSE	Profit
GSPC	0.820 (0.214)	0.555 (0.036)	290.305 (0.844)	0.305 (0.145)
IXIC	0.789 (0.166)	0.543 (0.030)	2024.303 (8.651)	0.441 (0.302)
BAC	0.746 (0.247)	0.527 (0.026)	0.085 (0.000)	0.895 (0.769)
MSFT	0.872 (0.112)	0.531 (0.011)	0.463 (0.002)	0.556 (0.320)
PBR	0.969 (0.034)	0.446 (0.006)	0.464 (0.002)	-0.615 (0.127)
XOM	0.863 (0.182)	0.509 (0.015)	1.429 (0.005)	0.033 (0.119)

Table 4.7: Summarized results for CNN.

than stocks, with PBR and XOM falling below 50% on average. The profit for PBR underperforms to SVR: -66.6% against -21.5 baseline. XOM profits are close to neutral, and overshadowed by large deviations. Profits for other time series range between 32.5% and 75.5%. Profits for BAC seem unreliable however, with a standard deviation nearing twice the average. PBR and XOM are the only time series that the CNN delivers lower profits compared to baseline. MSE scores are slightly lower.

Double-step prediction

The CNNs produced less extreme bull ratios for double-step prediction, except for PBR and BAC. Keeping in mind how BAC develops during the test set, this has a generally good influence on performance for that particular stock. Improved directional accuracy is seen on most of the data sets, although deviations also went up, especially for the indices. Despite significant profit improvement on BAC, the other time series stayed roughly the same, or decreased. MSE values doubled from single-step prediction, and are now typically above the SVR.

4.3 Analysis

In this section the results presented in section 4.2 are discussed and compared across models. As stated in section 1.2, models are analyzed in the context of using them directly as a trading strategy. Section 3.2.4 explains how profits reported in the results indicate trading performance. For this reason, discussions in this section are centered around profit.

Figure 4.7 visualizes average single-step profits for each model, in addition to the natural development of each test set. The natural test set

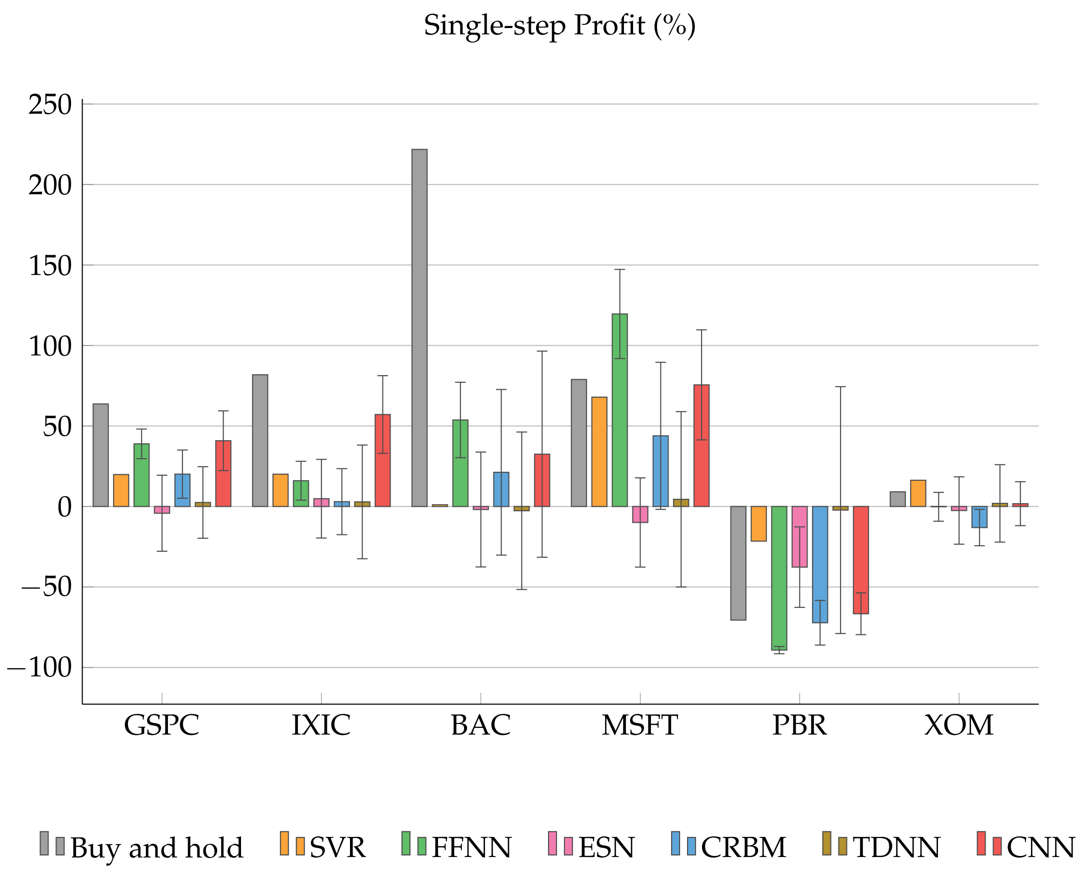


Figure 4.7: Profits for single-step strategies and buy and hold.

development is also known as the *buy and hold* strategy, since it represents the profits generated by buying shares at the onset of the test period, and selling at the end. Non-deterministic models have their *standard deviations* marked by thin lines. Unless specified otherwise, discussions refer to the single-step profits seen in figure 4.7, but the double-step equivalent is shown in figure 4.8.

4.3.1 Index Performance

This study includes two indices. How the models performed on them is discussed in the following.

S&P 500

As detailed in section 3.2.2, the GSPC index is used for tuning the hyper-parameters. Looking at figure 4.7 it does not appear to carry any significant advantage, with all models generating a profit below the buy and hold.

FFNN and CNN stand out with a profit comparable to, but lower than GSPC itself. CNNs show some deviations. The SVR baseline underperforms in comparison, approximately returning a third of buy and hold. Looking back at the results in section 4.2, we see that the SVR

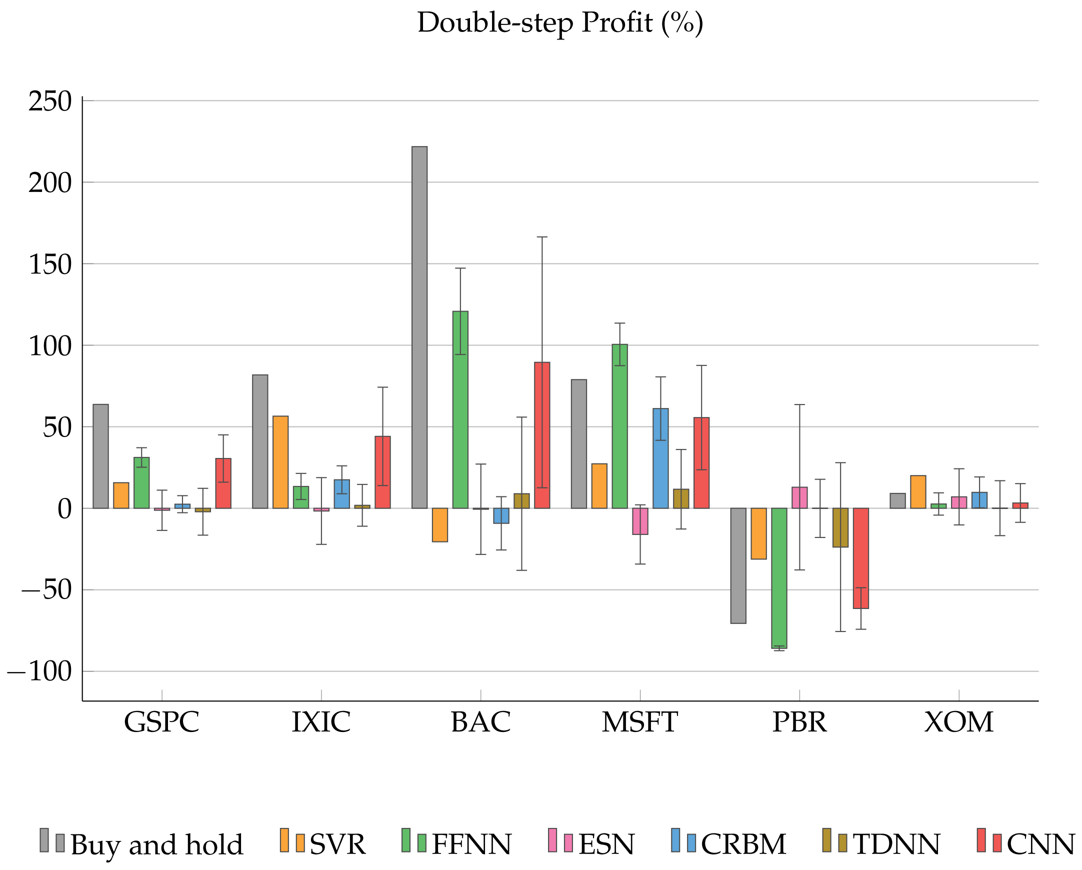


Figure 4.8: Profits for double-step strategies and buy and hold.

has a bull ratio of 56.3%, while the FFNN and CNN have 73.7% and 90.2% respectively. 90.2% seems dangerously high, although seemingly unproblematic for GSPC, given its bull market conditions. Ideal bull ratio is 56%, which also is the ratio of the SVR. The CRBM has an average profit similar to the SVR, although is considered worse because of the significant deviations. ESN and TDNN models both meander around zero profit, with relatively large deviations.

Double-step prediction had a negative impact on most models, with the ESN and TDNN staying about the same. CRBM dropped significantly, barely staying positive.

NASDAQ Composite

IXIC is visually similar to GSPC as seen in figure 3.2, but differences between the two stocks are observed in the profits. The buy and hold strategy outperforms the average of any model, with only the CNN getting close. Its good performance is explained once again by the unlikely high bull ratio, and it is questionable whether the model has actually found any patterns beyond the general bull trend. SVR and FFNN make slight profits, although with significant deviations for the FFNN. Bull ratios and directional accuracies are comparable to GSPC for the two models. The ESN, CRBM and TDNN generate unreliable profits, staying barely positive on average with high standard deviations.

Looking at figure 4.8, double-step prediction profit with SVR is approaching the buy and hold strategy. There is no obvious explanation to the improvement, beyond what is stated in section 3.2.4. The CRBM also sees an improved profit, with significantly reduced standard deviation. On the other hand CNN profit falls slightly, making it underperform to the SVR.

4.3.2 Stock Performance

Performance with respect to the four selected stocks is here compared and discussed.

Bank of America Corporation

Dissimilar market conditions during training and testing make profits for BAC especially interesting (section 3.3.1). 221.7% does the stock move by itself, further than any of the other test sets.

The change in market conditions seems to confuse most of the models, as seen by their large standard deviations. SVR, ESN and TDNN profits are around zero on average. CRBM and CNN generate some profits, but are outweighed by large deviations. The FFNN is the only model that delivers significant profits with reasonable deviations for BAC. Although it has a high bull ratio of 89.0%, it is impressive that it manages to see the BAC test set as bullish given the bear training samples.

Predicting two days ahead further improves profits for the FFNN, which achieves over half of buy and hold. The CNN benefits similarly, increasing profits about as much as the FFNN. Deviations stay largely the same for both models. The SVR and CRBM respond negatively to double-step prediction, both declining into negative profits.

Microsoft Corporation

With a bull test set development, buy and hold on MSFT is at the same level as IXIC. Unlike the index however, most models generate significant profits on MSFT. The SVR and CNN profit similarly to buy and hold, and the FFNN surpasses it on average with significant margin. Once again, the ESN and TDNN fall behind.

Performance on MSFT can perhaps say something about the ability to find usable features in the data: Looking at the price development in figure 3.4, the training set contains both bull and bear segments. Because these segments neutralize each other, the total movement is insignificantly small. Combined with shuffled training samples (section 3.2.3), the training set should be minimally biased towards any direction. If this is true it gives more credibility to the results, even at high bull ratios. With that being said, the FFNN and CNN have notably high bull ratios at 93.2% and 92.1%, respectively. Although the SVR delivers less profit, it does so with 60.7% bull ratio.

Moving on to double-step prediction, the SVR loses a significant amount of profit. The FFNN and CNN are also slightly adjusted down, although the former is still above buy and hold even when standard deviation is subtracted. The CRBM significantly reduces its deviations, while increasing average profit.

Petróleo Brasileiro S.A. - Petrobras

Because of the bear development in PBR test samples, buy and hold for this stock is negative. As discussed in section 2.1.2 however, positive profit may still be achieved through short market positions.

None of the models generate positive profit for single-step prediction on PBR, however some of them achieve a reduced loss compared to buy and hold. Because of large deviations in the TDNN, its miniscule average loss is not considered. The SVR on the other hand, significantly reduces its losses from PBR, outperforming any of the neural networks. ESN performance approaches baseline, although with some deviations. Notably, the FFNN, CRBM and CNN all perform about as badly as buy and hold, failing to reduce the loss.

It is apparent that PBR in the period 2005-2015 is a difficult time series to model, and there are a number of reasons why this may be. As discussed in section 3.3.1 the training data are volatile, displaying large and aggressive movements. Similar movements are not observed in the test data. Furthermore, PBR also split shares twice during the training period. Although this has been adjusted for numerically, any potential patterns in the data are likely to change: Due to the lower price per share, more traders will be able to participate in the trading, thereby shaping the price differently. Perhaps performance could be improved by training the models on a reduced test set solely consisting of samples succeeding the most recent split.

Results are similar for double-step prediction, although the ESN manages to reach slight positive profit on average. Due to large deviations however, this accomplishment is not read much into.

Exxon Mobile Corporation

Price development for XOM is volatile with a slight positive change in the test set, as seen in figure 3.6. Referring to figure 4.7, neither models nor the buy and hold strategy generate significant profits. The SVR outperforms the neural networks, also surpassing the stock itself.

Double-step does not make much difference, except a slight change for the CRBM, which now approximately matches the buy and hold strategy.

4.3.3 CNN Bull Ratio

Looking at the bull ratios in table 4.7, it is clear that the CNN tends to bias strongly towards upward price changes. A similar behavior is observed for the FFNN in table 4.3, however it is more restrained in comparison.

Two core differences between FFNNs and CNNs are the convolutional and pooling layers. It is not unlikely that the weight sharing regularizes the model too strongly, making it learn slightly underfitted weights. Another possibility is that sub-sampling removes too many details from the features, making it difficult to respond to fine patterns. Due to the broad scope of this thesis, the focus has been to exploit the characteristics of each model rather than figuring out the optimal configuration. Given the recent success of CNNs for other applications, the one-sided results extracted here are perhaps unexpected.

4.3.4 Mean Squared Error Performance

As mentioned in section 3.2.4, a measure of MSE is included mainly because of its function as a standard performance measure in the machine learning community. MSE provides a measure of how accurate predictions are with respect to ideal values, and pointing out these scores may help the reader relate this study to other research. Single-step MSE performance relative to baseline will be described in the following.

The TDNN produces the worst MSE scores in this study. Errors approximately six times larger than baseline were consistently observed for each data set. The FFNN and ESN follow up with twice as large MSE scores than the SVR. Both the CRBM and CNN slightly outperformed baseline, with the CNN being marginally better of the two.

4.3.5 Invariance to Temporal Architectures

Based on the results in this study, it does not seem like temporal architectures improve prediction quality compared to a pure spatial connectivity. It should be noted however, that temporal models usually contain fewer parameters due to being partially connected. Fewer parameters may theoretically lead to more efficient learning, due to the limited scale of the training sets and input samples it does not make a significant difference.

Although the FFNN is a fully connected network, it uses deep learning techniques such as unsupervised pre-training, fine-tuning with backpropagation and dropout. These measures help regularize the network, and likely prevent it from overfitting to noise.

4.3.6 Large Deviations

Arguably the biggest disadvantage ANNs have to the SVR in noisy environments is that their learning is stochastic. While the baseline produces one result and sticks to it, the neural networks see significant deviations between individual models. In some cases the standard deviation in profit is greater than the average, like the CRBM or CNN on BAC. Although some deviation is to be expected, observing variations this large raises the question whether these models actually see anything useful in the data beyond the noise.

4.3.7 Supervised or Unsupervised

As mentioned in section 1.3, [3] found that Deep Belief Networks (DBNs) outperform FFNNs for exchange rate prediction. Although the DBN included in this study is based on another RBM, namely the CRBM, this study opposes those results. There is no indication in [3] that the FFNN was trained using unsupervised pre-training or dropout however, and this is likely to explain some of the differences.

4.3.8 Value as a Trading Strategy

When considering using a model directly as a trading strategy, consistent and positive profits are valued. Referring to the single-step profit comparison in figure 4.7, the buy and hold strategy has a great performance on most data sets, but fails in bear markets like PBR. Incidentally, all models also failed at generating positive profit on PBR, which leads to the conclusion that long-term investments are more profitable than trading every prediction a given model makes. This study has not attempted to find the most profitable trading strategy, but rather uses a simple strategy to evaluate the models. By looking at the results, models can be selected for incorporation in more sophisticated strategies. Based on this study, the SVR, FFNN and CNN stand out with mostly positive results, and would have been viable to trade after given the observed profits. As discussed in section 3.5 however, further optimization of hyper-parameters will likely

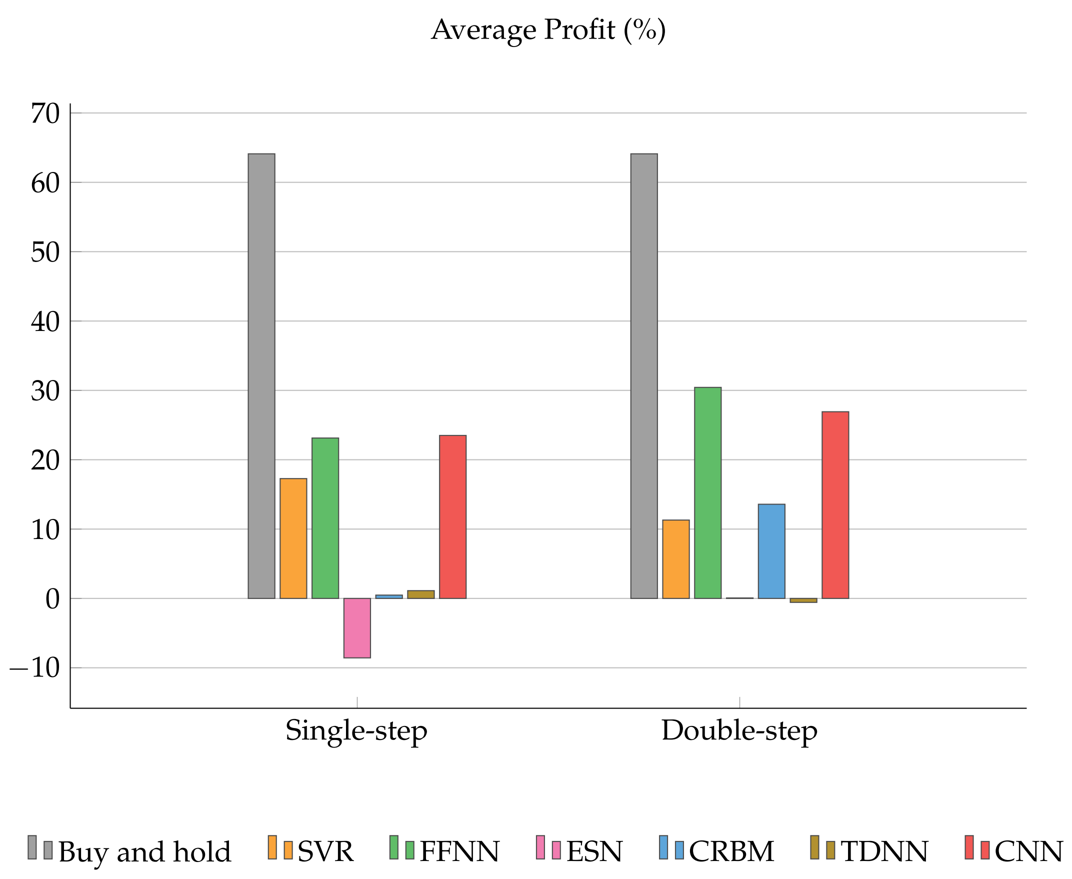


Figure 4.9: Average profits for single-step and double-step strategies and buy and hold.

improve the results, as well as using a more elaborate trading strategy. On the other hand, the simplified profit estimation used here is likely too optimistic, as there are fees associated with executing a trade (section 3.5).

4.3.9 Average Profits

In order to make a clear distinction between results, the profits for each model can be averaged across all time series. Reducing performance down to a single number does remove a lot of information, but since profit is the main performance measure, it is also very telling.

Figure 4.9 plots the average profits. Despite their significant losses on PBR, the FFNN and CNN deliver strong profits on the other time series which average greater than the SVR. Predicting double-step improves average profits for most of the neural networks, creating a greater gap between them and the SVR, which additionally generates less profit compared to single-step prediction. As expected the buy and hold strategy strongly outperforms the models in both cases.

Based on figure 4.9 alone, the FFNN and CNN perform similarly for single-step, while the FFNN outperforms all the models in double-step prediction. FFNNs deliver best average profits overall, with the CNN following closely.

4.3.10 Single-step vs. Double-step

Double-step prediction was included in an attempt to reduce noise in the data, see section 3.2.4. By comparing results however, models seem to gain from it about as many times as they degrade by it. Having said that, figure 4.9 shows improvement on average for the neural networks, when applied for double-step prediction. Given how the SVR performs worse for predicting two days ahead, drawing an absolute conclusion regarding the effect of double-step prediction is problematic. Double-step prediction was intended to smooth out some of the noise, thereby increasing prediction quality and profitability. The neural networks seemingly adhere to this intuition, while the SVR performs better for single-step prediction.

It can be argued that the double-step strategy described in section 3.2.4 is too simple, and could for instance be improved by closing open positions that go against the most recent prediction. As mentioned in section 3.2.4 however, there are many ways machine learning models can be incorporated in a trading strategy. Since the goal of the study is to compare models, rather than trading strategies, it makes sense to use a strategy that fully follows every prediction the model makes.

Chapter 5

Conclusion

This study aims to compare modern Artificial Neural Networks (ANNs), when applied for short-term stock price prediction using daily data. Models are evaluated in the context of following a financial strategy that trades every prediction a given model makes. The goal is to discover the relative performance between models, and see if any of them are viable to trade according to.

The results varied, between both time series and models. The Support Vector Regression (SVR), Feed Forward Neural Network (FFNN) and Convolutional Neural Network (CNN) showed comparable results, mostly producing positive profits. Under closer inspection however, the CNN had an unreasonably strong bias towards upward movement: For every time series except Bank of America Corporation (BAC), over 90% of the predictions were bull. The Conditional Restricted Boltzmann Machine (CRBM) also produced more positive than negative profits on average, but they are not significant enough to justify the observed large deviations. Lastly, the Echo State Network (ESN) and Time-Delay Neural Network (TDNN) displayed noisy behavior, generating profits averaging close to zero with significant deviations.

Given the quantity of models evaluated, less time is dedicated to hyper-parameter optimization of individual models. This is a limitation of the thesis, and implies that the reported performance might not accurately represent the full potential of each model. Furthermore, profit calculations do not account for real-world fees, such as market slippage or broker commissions. The reported profits are still indicative of performance, but should mainly be regarded with respect to other models in this study.

A universal observation for all models on the declining stock Petróleo Brasileiro S.A. - Petrobras (PBR), is a negative average profit. PBR is the only time series with a negative development in the test set, and its training set is dominated by aggressively volatile patterns. Furthermore, the company split its shares two-for-one twice during the training set. Although this was adjusted for numerically, stock splits lower the price, essentially rendering the shares accessible to more traders. In turn this may potentially obscure any established patterns. None of the models made a positive profit for PBR, although the SVR reduced its losses significantly compared to the natural development of the stock. Perhaps the models would benefit from a selectively sampled training set, where only samples succeeding the date of the most recent split are included.

Three models stand out with better performance: The SVR, FFNN and CNN. Due to large deviations and high bull ratios however, the CNN is regarded as potentially unreliable despite good profits. Both the SVR and FFNN show potential, and to some extent actually complement each other: The SVR is able to generate a profit on Exxon Mobile Corporation (XOM) while also significantly reducing losses on PBR. For BAC and Microsoft Corporation (MSFT) on the other hand, the FFNN sees significant profits that outperform the other models. This encourages further research, for instance on using models in an optimized trading strategy, or together in a hybrid solution. See chapter 6 for more details.

When comparing profits averaged across all time series, the FFNN outperforms the other models: 23.13% and 30.43% for single-step and double-step prediction, respectively. The CNN follows closely behind, with 23.50% single-step and 26.92% double-step profits. Average SVR profits were observed at 17.28% for single-step and 11.30% for double-step, thereby opposing the expectation that predicting two days ahead results in greater profits. Based on the average profits, the three abovementioned models all seem viable to trade according to. Given the extreme bull ratio of the CNN and the slightly lower profits for SVR, the FFNN is deemed most viable for trading.

Chapter 6

Future Work

Predicting financial markets with machine learning is a comprehensive problem to research. Examples of related research topics are listed in this chapter.

6.1 Ensemble Modeling

No single model was observed to perform adequately on every time series. As discussed in section 4.3.8 however, there is usually at least one model that shows reasonable performance. Perhaps it is possible to combine several models in an ensemble configuration, and get better predictions.

There is existing research on this topic like [27, 54], both of which introduce novel ensemble models for time series prediction. Combining models can be achieved in multiple ways however, and comparing different ensemble configurations for financial prediction could prove beneficial. A thorough overview of different hybrid configurations and their terminology is given in [46].

6.2 Enhance Training Samples

This study considers each stock and index as isolated cases. While this makes it easier to reason over the results, it also reduces the variety of samples that the model is exposed to during training. For models learning to predict a stock like Bank of America Corporation (BAC) for instance, the bear training set makes it challenging to predict bull test set development. Using a more diverse training set composed from multiple stocks could possibly benefit the predictions.

As discussed in section 2.1.3, some price movements are local to single stocks, while others affect whole markets. If input vectors are preprocessed to filter out price changes not local to the stock, any patterns found are guaranteed to come from the stock itself. A simpler approach to preprocessing is to simply include index prices in the input vectors. However, this has the side effect of making the modeling problem more complex.

6.3 Analyze Feature Detectors

Although the dynamics of neural networks are simple to understand, knowing exactly what features the networks learn is another story. For image classification, convolutional networks are known to extract object edges as low-level features, see for example [53]. We can confirm them as reasonable, because they activate similarly to biological neurons found in the visual cortex of cats [26] and macaque monkeys [40]. [53] goes further, and considers how different layers contribute to the total network performance. Leading a similar study in the context of financial prediction could help to understand why models perform differently.

Bibliography

[1] Bank of America. *Bank of America's History, Heritage & Timeline*. URL: http://about.bankofamerica.com/en-us/our-story/our-history-and-heritage.html#today (visited on 04/04/2015).

[2] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers." In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

[3] Jing Chao, Furao Shen, and Jinxi Zhao. "Forecasting exchange rate with deep belief networks." In: *Neural Networks (IJCNN), The 2011 International Joint Conference on*. July 2011, pp. 1259–1266. DOI: 10.1109/IJCNN.2011.6033368.

[4] Ronnie Clark. *Support Vector Regression*. URL: http://www.mathworks.com/matlabcentral/fileexchange/43429-support-vector-regression (visited on 04/22/2015).

[5] G. Cybenko. "Approximation by superpositions of a sigmoidal function." English. In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274.

[6] MATLAB®. *Datafeed Toolbox*. URL: http://mathworks.com/products/datafeed/ (visited on 04/22/2015).

[7] Sergey Demyanov. *ConvNet*. URL: http://github.com/sdemyanov/ConvNet (visited on 04/22/2015).

[8] Oxford Dictionaries. *Definition of Market*. URL: http://www.oxforddictionaries.com/definition/american_english/market (visited on 04/21/2015).

[9] Oxford Dictionaries. *Definition of Stock Exchange*. URL: http://www.oxforddictionaries.com/definition/american_english/stock-exchange (visited on 04/21/2015).

[10] Harris Drucker et al. "Support vector regression machines." In: *Advances in neural information processing systems* 9 (1997), pp. 155–161.

[11] Jeffrey L. Elman. "Finding Structure in Time." In: *Cognitive Science* 14.2 (1990), pp. 179–211. ISSN: 1551-6709. DOI: 10 . 1207 / s15516709cog1402 _ 1. URL: http : / / dx . doi . org / 10 . 1207 / s15516709cog1402_1.

[12] ExxonMobil. *About us*. URL: http://corporate.exxonmobil.com/en/company/about-us (visited on 04/04/2015).

[13] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." English. In: *Biological Cybernetics* 36.4 (1980), pp. 193–202. ISSN: 0340-1200. DOI: 10.1007/BF00344251. URL: http://dx.doi.org/10.1007/BF00344251.

[14] Google. *Apple Inc.* URL: http://www.google.com/finance?q=NASDAQ:AAPL (visited on 04/25/2014).

[15] Geoffrey Hinton. *A Practical Guide to Training Restricted Boltzmann Machines*. Tech. rep. 2010. URL: http://www.cs.toronto.edu/%5C~%7B%7Dhinton/absps/guideTR.pdf.

[16] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence." In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[17] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. "A Fast Learning Algorithm for Deep Belief Nets." In: *Neural Computation* 18.7 (July 2006), pp. 1527–1554. ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. URL: http://dx.doi.org/10.1162/neco.2006.18.7.1527.

[18] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors." In: *CoRR* abs/1207.0580 (2012).

[19] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks." In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: http://dx.doi.org/10.1016/0893-6080(91)90009-T. URL: http://www.sciencedirect.com/science/article/pii/089360809190009T.

[20] D. H. Hubel and T. N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex." In: *The Journal of Physiology* 195.1 (1968), pp. 215–243. eprint: http://jp.physoc.org/content/195/1/215.full.pdf+html. URL: http://jp.physoc.org/content/195/1/215.abstract.

[21] S&P Dow Jones Indices. *S&P 500*. URL: http://us.spindices.com/indices/equity/sp-500 (visited on 03/03/2015).

[22] Investopedia. *Electronic Trading: The Nasdaq Vs. The NYSE*. URL: http://www.investopedia.com/university/electronictrading/trading1.asp (visited on 04/30/2015).

[23] Herbert Jaeger. *ESN Toolbox*. URL: http://organic.elis.ugent.be/sites/organic.elis.ugent.be/modules/pubdlcnt/pubdlcnt.php?file=http://minds.jacobs-university.de/sites/default/files/uploads/SW/ESNToolbox.zip&nid=129 (visited on 04/22/2015).

[24] Herbert Jaeger. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note." In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148 (2001), p. 34.

[25] Quasar Jarosz. *Neuron Hand-tuned*. URL: http://upload.wikimedia.org/wikipedia/commons/b/bc/Neuron_Hand-tuned.svg (visited on 04/29/2015).

[26] J. P. Jones and L. A. Palmer. "The two-dimensional spatial structure of simple receptive fields in cat striate cortex." In: *Journal of Neurophysiology* 58.6 (1987), pp. 1187–1211. ISSN: 0022-3077.

[27] Mehdi Khashei and Mehdi Bijari. "An Artificial Neural Network (P,D,Q) Model for Timeseries Forecasting." In: *Expert Syst. Appl.* 37.1 (Jan. 2010), pp. 479–489. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2009.05.044. URL: http://dx.doi.org/10.1016/j.eswa.2009.05.044.

[28] Y. Lecun et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[29] Yuhong Li and Weihua Ma. "Applications of Artificial Neural Networks in Financial Economics: A Survey." In: *Computational Intelligence and Design (ISCID), 2010 International Symposium on*. Vol. 1. Oct. 2010, pp. 211–214. DOI: 10.1109/ISCID.2010.70.

[30] R. Llinas. "Neuron." In: *Scholarpedia* 3.8 (2008), p. 1490.

[31] Mantas Lukoševičius. "A Practical Guide to Applying Echo State Networks." English. In: *Neural Networks: Tricks of the Trade*. Ed. by Grégoire Montavon, GenevièveB. Orr, and Klaus-Robert Müller. Vol. 7700. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 659–686. ISBN: 978-3-642-35288-1. DOI: 10.1007/978-3-642-35289-8_36. URL: http://dx.doi.org/10.1007/978-3-642-35289-8_36.

[32] W Maass, T Natschläger, and H Markram. "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations." In: *Neural Computation* 14.11 (Nov. 2002), pp. 2531–2560. ISSN: 0899-7667. DOI: 10.1162/089976602760407955.

[33] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008. ISBN: 0521865719, 9780521865715.

[34] Microsoft. *Facts About Microsoft*. URL: http://news.microsoft.com/facts-about-microsoft/#About (visited on 04/04/2015).

[35] Marvin Minsky and Papert Seymour. "Perceptrons." In: (1969).

[36] NASDAQ. *NASDAQ Composite Index*. URL: http://www.nasdaq.com/markets/composite-eligibility-criteria.aspx (visited on 04/04/2015).

[37] MATLAB®. *Neural Network Toolbox*. URL: http://mathworks.com/products/neural-network/ (visited on 04/22/2015).

[38] Petrobras. *About us - Petrobras*. URL: http://www.petrobras.com/en/about-us/ (visited on 04/04/2015).

[39] George Edgin Pugh. In: *The biological origin of human values*. London: Routledge & Kegan Paul, 1978, pp. 1–1000. ISBN: 0-7100-8925-2.

[40] Dario L. Ringach. "Spatial Structure and Symmetry of Simple-Cell Receptive Fields in Macaque Primary Visual Cortex." In: *Journal of Neurophysiology* 88.1 (2002), pp. 455–463. ISSN: 0022-3077.

[41] Frank Rosenblatt. *The perceptron–a perceiving and recognizing automaton*. Tech. rep. 85-460-1. 00344. Buffalo, NY: Cornell Aeronautical Laboratory, 1957.

[42] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (Oct. 9, 1986), pp. 533–536. URL: http://dx.doi.org/10.1038/323533a0.

[43] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Edinburgh Gate, Harlow, Essex CM20 2JE, England: Pearson Education Limited, 2014. ISBN: 1292024208, 9781292024202.

[44] E.W. Saad, D.V. Prokhorov, and D.C. Wunsch. "Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks." In: *Neural Networks, IEEE Transactions on* 9.6 (Nov. 1998), pp. 1456–1470. ISSN: 1045-9227. DOI: 10.1109/72.728395.

[45] Robert P. Schumaker and Hsinchun Chen. "Textual Analysis of Stock Market Prediction Using Breaking Financial News: The AZFin Text System." In: *ACM Trans. Inf. Syst.* 27.2 (Mar. 2009), 12:1–12:19. ISSN: 1046-8188. DOI: 10.1145/1462198.1462204. URL: http://doi.acm.org/10.1145/1462198.1462204.

[46] AmandaJ.C. Sharkey. "Types of Multinet System." English. In: *Multiple Classifier Systems*. Ed. by Fabio Roli and Josef Kittler. Vol. 2364. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 108–117. ISBN: 978-3-540-43818-2. DOI: 10.1007/3-540-45428-4_11. URL: http://dx.doi.org/10.1007/3-540-45428-4_11.

[47] Masayuki Tanaka. *Deep Neural Network*. URL: http://www.mathworks.com/matlabcentral/fileexchange/42853-deep-neural-network (visited on 04/22/2015).

[48] Graham Taylor. *Modeling Human Motion Using Binary Latent Variables*. URL: http://www.uoguelph.ca/~gwtaylor/publications/nips2006mhmublv/code.html (visited on 04/22/2015).

[49] Graham W. Taylor and Geoffrey E. Hinton. "Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style." In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: ACM, 2009, pp. 1025–1032. ISBN: 978-1-60558-516-1. DOI: 10.1145/1553374.1553505. URL: http://doi.acm.org/10.1145/1553374.1553505.

[50] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. "Modeling human motion using binary latent variables." In: *Advances in neural information processing systems* 19 (2007), p. 1345.

[51] A. Waibel et al. "Phoneme recognition using time-delay neural networks." In: *Acoustics, Speech and Signal Processing, IEEE Transactions on* 37.3 (Mar. 1989), pp. 328–339. ISSN: 0096-3518. DOI: 10.1109/29.21701.

[52] H. White. "Economic prediction using neural networks: the case of IBM daily stock returns." In: *Neural Networks, 1988., IEEE International Conference on*. July 1988, 451–458 vol.2. DOI: 10.1109/ICNN.1988.23959.

[53] MatthewD. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks." English. In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Vol. 8689. Lecture Notes in Computer Science. Springer International Publishing, 2014, pp. 818–833. ISBN: 978-3-319-10589-5. DOI: 10.1007/978-3-319-10590-1_53. URL: http://dx.doi.org/10.1007/978-3-319-10590-1_53.

[54] G Peter Zhang. "A neural network ensemble method with jittered training data for time series forecasting." In: *Information Sciences* 177.23 (2007), pp. 5329–5346.