SEQUENCE-TO-SEQUENCE LEARNING OF FINANCIAL TIME SERIES IN ALGORITHMIC TRADING

Examensarbete Systemarkitekturutbildningen

Philip Arvidsson
Tobias Ånhed

VT 2017:KSAIXX

HÖGSKOLAN I BORÅS

HÖGSKOLAN
I BORÅS

BESÖKSADRESS: JÄRNVÄGSGATAN 5 · POSTADRESS: ALLÉGATAN 1, 501 90 BORÅS
TFN: 033-435 40 00 · E-POST: INST.HIT@HB.SE · WEBB: WWW.HB.SE/HIT

Svensk titel: Sekvens-till-sekvens-inlärning av finansiella tidsserier inom algoritmisk handel

Engelsk titel: Sequence-to-Sequence Learning of Financial Time Series in Algorithmic Trading

Utgivningsår: 2017

Författare: Philip Arvidsson, Tobias Ånhed

Handledare: Patrick Gabrielsson

Abstract

Predicting the behavior of financial markets is largely an unsolved problem. The problem has been approached with many different methods ranging from binary logic, statistical calculations and genetic algorithms. In this thesis, the problem is approached with a machine learning method, namely the Long Short-Term Memory (LSTM) variant of Recurrent Neural Networks (RNNs). Recurrent neural networks are artificial neural networks (ANNs)—a machine learning algorithm mimicking the neural processing of the mammalian nervous system—specifically designed for time series sequences. The thesis investigates the capability of the LSTM in modeling financial market behavior as well as compare it to the traditional RNN, evaluating their performances using various measures.

Keywords: deep learning, machine learning, quantitative finance, algorithmic trading, black-box trading, lstm, rnn, time series forecasting, prediction, tensorflow, keras, forex, neural network, econometrics

Sammanfattning

Prediktion av den finansiella marknadens beteende är i stort ett olöst problem. Problemet har tagits an på flera sätt med olika metoder så som binär logik, statistiska uträkningar och genetiska algoritmer. I den här uppsatsen kommer problemet undersökas med maskininlärning, mer specifikt Long Short-Term Memory (LSTM), en variant av rekurrenta neurala nätverk (RNN). Rekurrenta neurala nätverk är en typ av artificiellt neuralt nätverk (ANN), en maskininlärningsalgoritm som ska efterlikna de neurala processerna hos däggdjurs nervsystem, specifikt utformat för tidsserier. I uppsatsen undersöks kapaciteten hos ett LSTM att modellera finansmarknadens beteenden och jämförs den mot ett traditionellt RNN, mer specifikt mäts deras effektivitet på olika vis.

Nyckelord: finans, algoritmisk handel, tidsserier, prediktion, maskininlärning, forex, neurala nätverk, tensorflow, keras, kvantitativ finans, lstm, rnn, ekonometri

This page intentionally left blank.

Sequence-to-Sequence Learning of Financial Time Series in Algorithmic Trading

UNIVERSITY
OF BORÅS

Philip Arvidsson
philip@philiparvidsson.com

Tobias Ånhed
anhedtobias@gmail.com

March 21, 2017

"Research is to see what everybody else has seen, and to think what nobody else has thought."

ALBERT SZENT-GYORGYI

ACKNOWLEDGEMENTS

We'd like to express our gratitude to our supervisor, Patrick Gabrielsson, without whom this thesis would not have been written. He has guided us throughout our research and shared with us his vast knowledge and experience in the fields of quantitative finance and machine learning.

ABSTRACT

Predicting the behavior of financial markets is largely an unsolved problem. The problem has been approached with many different methods ranging from binary logic, statistical calculations and genetic algorithms. In this thesis, the problem is approached with a machine learning method, namely the Long Short-Term Memory (LSTM) variant of Recurrent Neural Networks (RNNS). Recurrent neural networks are artificial neural networks (ANNS)—a machine learning algorithm mimicking the neural processing of the mammalian nervous system—specifically designed for time series sequences. The thesis investigates the capability of the LSTM in modeling financial market behavior as well as compare it to the traditional RNN, evaluating their performances using various measures.

CONTENTS

1 Introduction

Time series forecasting—predicting future values of variables within temporal datasets—is largely an unsolved problem in complex or chaotic domains such as weather (e.g. humidity, temperature or wind speed) and economics (e.g. currency exchange rates or stock prices). Examining the problem with the latest models in the field of machine learning—Long Short-Term Memory (LSTM) based Recurrent Neural Networks (RNNs)—gives rise to new hope of being able to predict time series in these domains.

1.1 Background and Motivation

Algorithmic trading combines the fields of finance and algorithmics/machine learning. To understand the problem, a high level of knowledge is desirable in both of these—and surrounding—domains. Below, financial markets and artificial neural networks as well as their connection to algorithmic trading is briefly summarized. It is further explained in the Theory chapter.

1.1.1 *Financial Markets*

The price movements on financial markets are difficult (even impossible, according to the efficient-market hypothesis) to predict (Fama, 1995). If one were able to predict the movements of, for example, stock prices or currency exchange rates, that information could be used to "beat the market" (essentially, buying before a rise and selling before a drop) to continuously increase the value of an investment portfolio, yielding a positive net return.

Predicting a financial market, whether the method used for prediction consists of a fundamental- or technical approach, rests on devising a certain successful strategy—looking at and considering a variety of factors and drawing a conclusion as to how the market should be operated on (for example, deciding at what point in time buying a certain financial asset is appropriate).

Furthermore, if the prediction process could be automated through the application of an algorithm that decides when a financial asset should be bought or sold with some level of accuracy (with respect to its predictions of price movements)—in practice, letting the algorithm perform trading operations autonomously on the market with real money—the algorithm would turn out to be profitable, generating continuous positive net returns over time.

Many attempts have been made with some success. During the last decade, there has been a rise in algorithmic trading from roughly 20 percent of the volume on US equity markets in 2005 up to over 50 percent in 2010 (Kaya, 2016), and although the rise seems to have reached its limit during recent years, the sheer volume of high-frequency trading implies some level of credibility for the approach.

1.1.2 *Artificial Neural Networks*

Artificial neural networks (ANNs) have been applied for several decades to solve multinomial logistic regression (determining what *class* a data point belongs to) and linear regression (determining the *value* of a dependent variable as a function of a data point's independent variables) problems. The early versions of ANNs (e.g. the single layer perceptron) could only solve *linearly separable* problems (Rosenblatt, 1958). An ANN, expressed in simple terms, is a mathematical model of a biological brain, consisting of neurons, synapses and dendrites, modeled by the ANN through cells with scalars, activation functions and weights.

The earlier models were also unable to predict time series (i.e. determining what class a data point belongs to with respect to its movement over time, or determining the value of a dependent variable as a function of the independent variables' movement over time), something that was later solved through the introduction of recurrent neural networks (Rumelhart et al., 1986), which can handle datasets with a temporal component dictating their intradependence and distribution within the dataset, where each point in the sequence is correlated to previous (with respect to time) points through several known and unknown factors.

RNNs have historically had problems with vanishing (or exploding) gradients—meaning they break down during training and become unable to model the problem well (Pascanu et al., 2012)—but this problem was solved through the introduction of the LSTM-based RNNs (Hochreiter and Schmidhuber, 1997).

Regardless of whether the problem is approached as a matter of classification or regression, the temporal aspect renders algorithms and models not specifically designed for the task impotent since one single input dataset could belong to several different classes depending on where, in time or order, it appears in the data sequence. The problem should therefore be examined with algorithms designed specifically for time series, such as LSTM-based RNNs.

Furthermore, *sequence-to-sequence* learning is a method for training the network on output sequences rather than providing it with a single target value following the input sequence; instead of giving the model an input sequence and the next, single value following the sequence, the model is given the *sequence* following the input sequence. This is also the method used to model the problem in this thesis and is explained in detail in the theory chapter.

1.2 Current Research

Currently, in the field of machine learning, hidden Markov models (HMMs), dynamic Bayesian networks (DBNs), tapped-delay neural networks (TDNNs) and RNNs are commonly used to handle sequential datasets, and have been applied with some degree of success to financial markets (Saad et al., 1998; Kita et al., 2012; Zhang, 2004).

Although much research is currently being done into the application of LSTM-based RNNs in financial markets, the research is still in its infancy, and although the LSTM-based RNN has been around for twenty years, it is only recently that progress is being made to a significant degree in practice. The volume of data created in the past couple of years—about 90 percent of all data ever created by humans (Devakunchari, 2014)—together with relatively cheap but very fast graphics processing units (GPUs), have opened up the field of machine learning to more cost efficient research.

More recently, deep LSTM-based RNNs have been applied with some level of success in predicting future time series from historical sequential datasets; such as predicting the weather for the next twenty-four hours (Zaytar and Amrani, 2016), or predicting the next sequence of words in natural language (Sutskever et al., 2014).

While natural language has obvious structure (i.e. *grammar*), weather, instead, is seemlingy chaotic, controlled by many unknown factors. Despite this; considering the success in weather prediction, LSTM-based RNNs may have the ability to take into account these factors, especially given an appropriate set of *features*—input data processed to make certain aspects more prominent, thus enabling the algorithm to model the problem more easily.

1.3 Problem Statement

Financial time series are used ubiquitously in algorithmic trading. In algorithmic trading, it is imperative that accurate predictions are made about numerous variables (such as volatility and returns) in order to time market entry and exit.

Since deep LSTM-based RNNs (specifically using sequence-to-sequence learning) have not been applied within the algorithmic trading domain before, and since they have shown success in solving similar problems in other domains, it raises the question whether the technique can be used to predict a future sequence of financial variables that can be used to time both entry and exit positions within a certain time horizon. The uniqueness of this approach is the prediction of *sequences* at a time, rather than predicting a single data point on each iteration.

Assuming that correlations exist along the temporal dimension of the dataset, the problem is reduced to a matter of finding an appropriate set of features enhancing the correlations, on which to train the LSTM-based RNN. Expressed in a more concise manner, we attempt to answer the question:

— Can price movements on financial markets be modeled through the application of LSTM-based RNNs, specifically using *sequence-to-sequence* learning, and if so, how does the LSTM-based RNN compare to the traditional RNN.

2 Theory

A deeper understanding of all relevant aspects of the thesis problem is laid out below. Some references are made without being explained further; these are to be considered as lying outside the delimitations of the thesis.

2.1 Financial Markets

A financial market is a trading market in which people trade financial securities, commodities and other assets. A market participant places a bid or ask for a certain security; a binding commitment to buy or sell that security. All current asks and bids are kept track of through the use of the *order book*. When a matching bid or ask is placed for the same security, a *transaction* occurs—the asset changes ownership to the purchasing party (having placed a bid for the security) and money is transferred to the selling party (having placed an ask for the security).

The number of asks and bids in the market at any point in time for a given price level comprises the market *depth*, and the total number of *transactions* (i.e. matching pairs of asks and bids) comprise the traded *volume* in the market (as opposed to the number of contracts in the market; the *market volume*).

Both bids and asks can be placed as *market orders*, meaning that they immediately trade at the current best market price (not counting slippage costs). The alternative is to specify a certain price—the limit price—guaranteeing that the order will trade at a price no worse than the limit price, hence sacrificing immediacy for certainty in regard to the execution price (although the order risks not trading at all if the market never reaches the limit price). In fact, it is these, so called, limit orders that show up in the limit order book.

Transactions also carry transaction costs, including a *brokerage fee*—a fee charged by the broker to conduct transactions between buyers and sellers, as well as an *exchange fee*.

2.1.1 *The Foreign Exchange Market*

The Foreign Exchange Market (forex) is a global decentralized financial market where currency pairs are traded. Although it is a financial market like many others, it has a certain set of attributes distinguishing it from others: Its traded volume is the largest of any financial market, it is a global, decentralized market, operated by a broker-dealer, as opposed to the more common centralized exchanges (except for e.g. forex derivatives traded on the cme), it is the most liquid asset of all asset classes, it is always open (except weekends), price movements are generally very small and broker-dealers offer highly leveraged margin accounts.

Market movements are announced as price interest points—*pips* (on other financial markets, known as *ticks*). A pip designates a minimal change in price. For example, a price movement from 1.2345 to 1.2346 represents a single pip, i.e. a tick step of 0.0001. A pip is always a change in the last decimal digit of the price (the currency exchange rate).

2.1.2 *Technical Analysis*

Technical analysis of financial assets is a method for understanding the asset value (and its movements) only by looking at previous market data (i.e. volume, ask price, bid price etc.). This can be looked at in contrast to *fundamental* analysis; estimating the intrinsic value of an asset by looking at various aspects and properties of the underlying company, market sector and macroeconomic factors; for example, looking at company financials or social trends.

In regard to the efficient-market hypothesis—the hypothesis that the market is efficient to such a degree that the market price immediately incorporates and reflects all relevant information (Fama, 1995)—the effectiveness of technical analysis is disputed: The quote “I realized that technical analysis didn’t work when I turned the chart upside down and didn’t get a different answer” has been attributed to Warren Buffett, who prefers a fundamental analysis approach to investment. However, the overwhelming empirical evidence of market inefficiencies suggests that the efficient market hypothesis can be refuted, hence providing a solid foundation for predicting future market behavior.

2.1.3 *Japanese Candlesticks*

Often a vital part in charting analysis, the Japanese candlestick provides a simple method of aggregating and plotting market behavior (Nison, 2001). Instead of plotting each individual pip (resulting in a very flat and *very* wide chart), pips are aggregated into *candlesticks*. This method can be used with any time frame (dictating how many pips end up in a single candlestick).

The candlestick is made up of three parts: The real body, the upper shadow and the lower shadow. The real body is bounded by open and close prices for all pips in the time frame aggregated into the candlestick. For example, if the time frame is one minute, assuming that the last pip has a higher value than the first pip, the lower edge of the real body denotes the first pip encountered that minute (the *open price*), while the upper edge of the real body denotes the *last* pip during that minute (the *close price*). Otherwise, if the last pip has a lower value than the first pip, the lower edge denotes the last pip, and the upper edge denotes the first pip. Usually, candlesticks with a close price higher than open price are represented by a white (sometimes green) body while candlesticks with a close price lower than an open price are represented by a black (sometimes red) body.

The top of the upper shadow, positioned on top of the real body, represents the *high price* during that minute; the highest value encountered in any pip that occurred during the time interval that the candlestick represents. Likewise, the bottom of the lower shadow represents the *low price*; the lowest value found in any pip during the time frame.

Together, these provide the open, high, low and close prices during the candlestick's time frame (often referred to as OHLC data.)

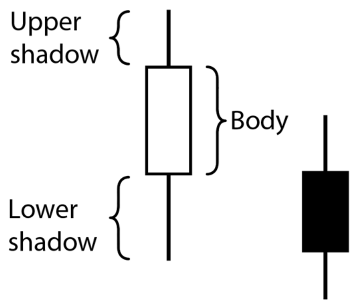


Figure 2.1: The Japanese candlestick and its parts.

Chart patterns (see below) often rely on Japanese candlestick aggregation to provide meaningful indications regarding market behavior. Moreover, the time frame represented by the candlestick can be set to any amount of time, allowing the trader to perform the charting analysis using various resolutions.

2.1.4 *Chart Patterns*

Giving more credibility to the idea of technical analysis, there exist certain *chart patterns* that can be observed prior to specific market expectations and behavior (Kirkpatrick and Dahlquist, 2010). The patterns can bee seen in the price movements of a financial security (sometimes combined with looking at the market volume or any other aspect of market behavior). As a concrete example of a pattern, there is the *bull flag*, represented by a strong upwards movement (in the case of the bull flag, called the flagpole, usually combined with a surge in market volume), after which a trend channel (representing the flag) is charted. When the price breaks through the upper boundary of the trend channel, the bull flag pattern is completed and a continuation of the rise is expected to follow.

There is also the *double top* chart pattern (illustrated below), which is a bearish reversal pattern; the pattern can be identified as two distinct price tops reaching roughly the same level. If the price of the financial security drops below a certain lower boundary of a trend channel charted around the pattern, it is expected that the price will drop further afterwards.

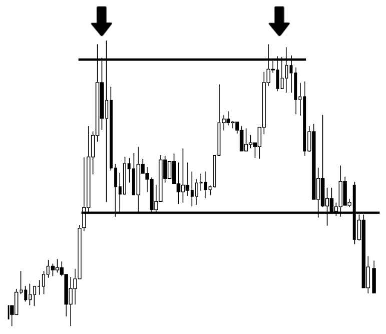


Figure 2.2: The bearish trend-reversal double top chart pattern.

There are a several different chart patterns indicating and predicting certain market behavior. They raise an interesting and relevant point, which will be touched upon again when the intricacies of neural network models are discussed below: If there are certain patterns recurring before certain market behavior, hypothetically, a properly configured neural network model should be able to learn to identify them under the proper circumstances and thus show potential in predicting market behavior.

These chart patterns are of great importance to the thesis problems, as—which will be detailed in the machine learning section—recurring chart patterns is a vital component for machine learning algorithms to model problems well.

No matter whether the trader is a human or a machine, chart patterns are a vital part of a technical analysis approach to trading strategies. For the sake of the thesis problem, we ascribe a high level of credibility to these chart patterns—it is not part of this thesis to investigate their validity in technical analysis trading.

2.1.5 *Technical Indicators*

While the occurrence of chart patterns are used as indicators for trading in technical analysis, there are also other technical indicators in the form of certain measures and computations of market properties.

As part of the trading strategy, like chart patterns, these technical *indicators* are used to aid in trading decisions. All of the technical indicators are based on already-available market information embedded in e.g. prices, volume etc. The technical indicators are computed on this information, giving evaluation measures that can be used to make certain aspects of the market behavior more prominent.

SIMPLE MOVING AVERAGE (SMA)

The Simple Moving Average is one of the most basic indicators used for trading. It is calculated by simply summing a certain number of price points (in sequence) together and dividing it by the number of price points sampled. For example, to calculate the SMA of a price at time t: Sample N points before t, add the prices together and divide the resulting value by N:

$$SMA^N(t) = \frac{1}{N}\sum_{i=0}^{N-1} Price(t-i)$$

Equation 2.1: Simple moving average formula.

RELATIVE STRENGTH INDEX (RSI)

The Relative Strength Index is a technical indicator used to assess the strength and weakness of a financial asset based on the closing prices over a certain period of time. The RSI measures the velocity and magnitude of price movements, computing the momentum—the rate of average gain versus average loss in prices over a certain period.

The period used to calculate the RSI is normally set to fourteen days, but the period can also be arbitrarily chosen.

The RSI gives a value on a scale from 0 to 100, with low and high values set at 30 and 70, respectively. It is reasoned that an RSI indicator showing a value lower than 30 indicates that a financial asset is oversold, where as a value higher than 70 indicates that it is overbought (Wilder, 1978).

Although the RSI is slightly more complex to calculate than the SMA, a variation of the RSI—called Cutler's RSI—provides a simpler way of calculating it. Cutler reasoned that the original RSI, introduced by Wilder (1978), using a *smoothed moving average*, provided very different results depending on what data point (time) in the sequence at which computation started. Because of this, Cutler's RSI is instead based on the SMA:

$$\begin{aligned}
D^N(t) &= \frac{1}{N}\sum_{i=0}^{N-1} max(Price(t-i-1) - Price(t-i), 0)\\
U^N(t) &= \frac{1}{N}\sum_{i=0}^{N-1} max(Price(t-i) - Price(t-i-1), 0)\\
RS^N(t) &= \frac{U^N(t)}{D^N(t)}\\
RSI^N(t) &= 100 - \frac{100}{1+RS^N(t)}
\end{aligned}$$

Equation 2.2: Cutler's relative strength index formula.

There are many more technical trading indicators. All of them claim to measure various aspects of the data, and some of them work together to provide a useful indication.

2.1.6 *Financial Measurements*

BID-ASK SPREAD

Although not strictly a trading indicator, the spread (difference between the best ask and best bid prices) is used as a model feature in our setup. It is a measurement of the transaction cost associated with market orders and an indicator for the liquidity of the market. The formula for the spread is trivial:

$$Spread(t) = Ask_t - Bid_t$$

Equation 2.3: Bid-ask spread formula.

SHARPE RATIO

A more sophisticated measurement is the Sharpe Ratio (Sharpe, 1966), which provides a way to analyze the risk-adjusted performance of financial assets, portfolios and trading strategies, taking into account the *risk* factor. A larger Shape ratio indicates better performance, while a lower Sharpe Ratio indicates worse performance. Moreover, comparing the Sharpe Ratio of two or more financial financial assets, portfolios or trading strategies, a greater Sharpe Ratio is characteristic of better performance with respect to risk[1]. The Sharpe Ratio is calculated as follows:

$$\begin{aligned}
S_a &= \frac{E[R_a - R_b]}{\sigma_a}\\
&= \frac{E[R_a - R_b]}{\sqrt{Var[R_a - R_b]}}
\end{aligned}$$

Equation 2.4: The Sharpe Ratio formula.

In the formula above R_a is the asset return, R_b is the return on a benchmark asset (the risk free rate of return on an index), $E[R_a - R_b]$ is the expected value of the excess of the asset return over the benchmark return and σ_a is the standard deviation of the asset excess return.

2.1.7 *Non-stationary Market Behavior*

A *stationary* process is a stochastic process whose joint probability distribution does not change when shifted in time. Consequently, measurements such as mean and standard deviation remain constant.

Contrary to the above, non-stationarity implies that the stochastic process is heteroskedastic with respect to time, meaning that the expected variance changes (Tunnicliffe-Wilson, 1989). In turn, this means that measurements performed on a market might work well for

[1] A greater Sharpe Ratio implies higher return for the same risk or the same return for a lower risk

technical analysis at one point in time, but might also stop working when the market meta-behavior—the hypothetical set of hyper-parameters dictating the market behavior—changes.

A common technique used to transform a non-stationary process into a (weakly) stationary process, is to remove the trend from the process (for processes with a deterministic trend) or to use differencing (for processes with one or more unit roots). Therefore, prices in financial time series are often differenced to create difference-stationary processes. In fact, the normalized, first difference of financial time series prices is commonly used. This is also known as the return, and is calculated according to Equation 2.5:

$$Return(t) = \frac{Price(t) - Price(t-1)}{Price(t-1)}$$

To further stabilize the variance of the time series, log returns can also be used, i.e. Log(Return).

Equation 2.5: Return formula.

2.1.8 *Trading Strategies*

A trading strategy—a set of rules or meta-rules for deciding when market entry or exit decisions are appropriate to take—builds on top of the knowledge presented in this chapter. Multiple decisions need to be made when constructing a trading strategy: A choice must be made in regard to what market or security to trade, which technical indicators to use, how to read the indicators in a way relevant to the market or security and changes in market regimes.

A trading strategy can be based on a single technical indicator (Connors and Alvarez, 2008) or a multitude of them. The technical indicators used in successful trading strategies are generally *trade secrets* (Chan, 2013); finding the best set of technical indicators to use in a trading strategy is thus one of the main challenges when developing one.

2.2 Machine Learning

Thinking about problem solving in general, it becomes obvious that different problems present varying levels of difficulty. Some problems are trivial while others are infinitely complex, with most lying somewhere inbetween. Although not always true (and this touches upon the $P = NP$ problem[2]), generally, the more trivial a problem and its solution, the more trivial its algorithm. For example, the problem of finding prime numbers can be solved by applying The Sieve of Eratosthenes (Horsley, 1772). Although computationally expensive (depending on how large prime numbers are being tested), the algorithm itself consists of only a handful of very concise rules, repeated over and over, that, when applied, definitely leads to a solution.

Other problems, however, are not as well understood or well-defined. As an example, determining whether a stove is hot is not trivial; it needs to be taken into account for what reason we are measuring it (determining whether it is acceptable to place a piece of steel on the stove implies a different definition of the word *hot* compared to determining whether it is a good idea to place your hand on it), whether the mean temperature of the stove is to be measured, at what point in time the temperature needs to be known, and so on. This creates a complex *context* for the problem—a vast amount of factors working together, determining how the problem should be understood—which is required to produce a trustworthy solution.

Enter *machine learning*: By providing a vast amount of measurements of all factors deemed relevant, a machine learning algorithm can be trained to infer the relevant context from the data and model the problem appropriately. This model can then be used by an intelligent agent to make autonomous decisions. As such, a machine learning algorithm can "reason" about the input data in the sense that it can create models of *concept delimitation*—for example, it can determine where to draw the line between *hot* and *not hot* by looking at the input data in various ways and making its decision by looking at common factors in various groups of data points.

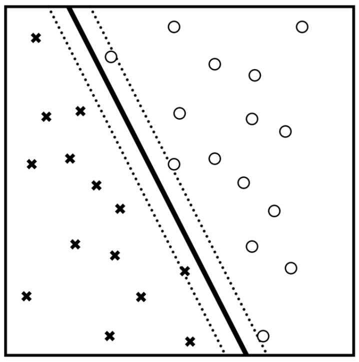


Figure 2.3: An illustration of a machine learning algorithm learning to create a decision boundary between two "concepts."

All on its own, an algorithm (such as *decision trees*) can emulate the creation of these internal concepts without any kind of target data—i.e. without being told what to do with the input data. Again, this works because the algorithm can perform certain measurements on the data, grouping or splitting it into segments that fit certain criteria. This is called unsupervised learning. More relevant to this thesis, however, is *supervised* learning which is laid out in more detail below.

[2]Informally, whether a problem whose solution can be quickly *verified* can also be quickly *computed.*

2.2.1 *Generalization*

Machine learning works by *generalizing* problems, given specific input data. That is, generalization means that a machine learning algorithm, through training, creates a model of the problem and learns to produce solutions for it. If the model produces satisfactory solutions on previously unseen inputs, the algorithm has created a generalized model of the problem. Goodfellow et al. (2016, p. 110) correctly points generalization out as the main area of concern in the field of machine learning:

> *The central challenge in machine learning is that we must perform well on new, previously unseen inputs—not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called generalization.*

2.2.2 *Regression*

Regression, as opposed to classification (determining what discrete class a data point belongs to), is used to estimate real-numbered variables from relationships between dependent and independent variables. That is, when a problem has a solution that is not discrete (i.e. binary; yes/no, or trinary; left, forward, right and so on) but rather, is real-numbered (e.g. temperature or length), regression is used. An ANN can be configured to handle both problems of regression and classification by selecting the appropriate activation function for the neural network's final layer.

2.2.3 *Supervised Learning*

Supervised learning means that a machine learning algorithm is trained by providing it with *target* data. Explained more colloquially, it can be understood as showing a toddler different objects and telling her what the different objects are, over and over again until the toddler can identify the different objects on her own. Algorithmically, we are now talking about *artificial neural networks*, which, as the name implies, bears certain similarities to the human brain. Machine learning algorithms trained with supervised learning can learn to identify a number of factors that are not easily defined by logic systems. Going back to the example with the stove presented earlier; by presenting the ANN with a temperature (and other features relevant to identifying the correct answer) and telling it whether that temperature is hot or not (as indicated by, for example, a human supervisor making an informed decision), the ANN eventually learns to model the problem by modifying its internal state.

With enough training, the ANN can then be presented with data and asked whether the data represents a hot or a cold stove, making decisions very close to the ones made by the supervisor during the learning phase. Not only that, but being presented with *unseen* data, the ANN will show an ability to make a sane decision in regard to the data representing a hot stove or a cold stove—the ANN has learned to make decisions on its own, on previously unseen data.

2.2.4 *Artificial Neural Networks*

Firstly, an artificial neural network has a hierarchical structure and uses a learning algorithm for modeling a problem, a concept based on the biological brain. The algorithm is a simplistic abstraction of neural processing in the mammalian central nervous sytem; a neuron is stimulated (a weight sum of input values are fed into a neuron), the neuron is activated (the weighted sum is passed through the activation function) and its output (activation) is relayed to neighboring neurons (fed to neurons in the next layer). Although artificial neural networks can be configured in many different ways, all setups normally consist of an input layer (a number of input nodes or neurons), one or more hidden layers, and an output layer.

Training an ANN consists of two major phases; the feed-forward phase and the backpropagation phase. During the feed-forward phase, weighted sums of the inputs are forward-propagated through the network and passed through each neuron's activation function. At the end of the feed-forward phase, the output error between the actual output and the true output is calculated, after which the backpropagation phase begins. During the backpropagation phase, the error is backward-propagated through the network. Lastly, the *weights* (see illustration below) are adjusted to reduce the error.

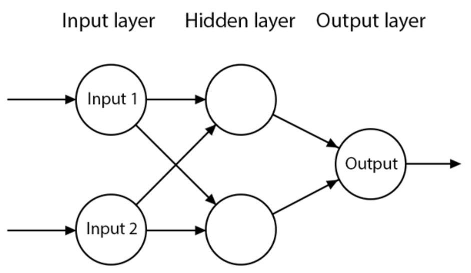


Figure 2.4: A small ANN with arrows between nodes representing the weights.

An input value is fed to each input node and the resulting values are multiplied by their respective weights—connection strengths to neurons in the next layer (illustrated by the lines between cells in fig. 2.4 above)—added together and then passed on as input values to nodes in the next layer in the network. For supervised learning—training the network by providing it with target data—the output node values are then compared to the target data, an error is calculated and the weights are adjusted accordingly. On the next iteration, the resulting output node values will be a little bit closer to the target data. The process is repeated until the network has reached a certain level of proficiency in modeling the problem and providing accurate output values. The end result is a set of weights that gives the algorithm a way of modeling the problem, thus producing *correct* output for a given set of inputs.

THE FEED-FORWARD PHASE

First, input is fed to the *input layer cells*. For the input layer, these means that their final *output values* are set to the data being fed to the rest of network.

After that, the *output* is calculated for each node or cell in consecutive layers according to Equation 2.6 during the *feed-forward phase*:

$$\begin{aligned} \sigma(x) &= \frac{1}{1+e^{-x}} \\ out_c &= \sigma\left(\sum_p out_p \cdot weight_{p,c} + \beta\right) \end{aligned}$$

where p is a cell in the previous layer and

β is the bias value from the previous layer.

Equation 2.6: ANN cell output formula using the logistic function as the activation function.

Or, in pseudocode:

```
**function** GET_OUTPUT($c$)
    $net \leftarrow 0$
    **for** $p$ in PREVIOUS_LAYER ($c$) **do**
        $net \leftarrow net$ + GET_OUTPUT($p$) $\times$ WEIGHT($p$, $c$)
    **end for**
    $out \leftarrow$ ACTIVATION($net$ + BIAS(PREVIOUS_LAYER($c$))
    **return** $out$
**end function**
```

Algorithm 2.1: Pseudocode for calculating cell output in an ANN.

After this, the *error* needs to be calculated. This is done by calculating the sum of squared errors[3] of the output neurons. The goal is to minimize it (during the backpropagation phase), but first, it needs to be calculated, which is done using Equation 2.7:

$$E_{total} = \frac{1}{2}\sum_o (target_o - out_o)^2$$

where o is a cell in the output layer

Equation 2.7: Squared error calculation of an ANN output.

At this point, the feed-forward phase is completed and an error (deviation from *target output*) has been calculated. It is time to move on to the backpropagation phase of the ANN learning process.

THE BACKPROPAGATION PHASE

After the error has been calculated, it is *propagated* back throughout the ANN, starting at the output nodes. For each weight in each layer, the *gradient*—the partial derivative of the error function with respect to each weight—is computed. Using the value of the gradient, the *weight* is adjusted for each node. The magnitude of the change to the network's internal state for each learning iteration is dictated by a *learning rate*—a multiplicative variable dictating the rate of change in the internal state.

For each node, the delta (δ) value is computed for the output layer neurons:

$$\begin{aligned} \delta_o &= \frac{\partial E_{total}}{\partial out_o} \times \frac{\partial out_o}{\partial net_o} \\ &= \frac{\partial E_{total}}{\partial out_o} \times \sigma'(net_o) \end{aligned}$$

where net_o is the weighted sum of all inputs to a node (logits).

Equation 2.8: Computing the δ value for an output neuron.

Then, iterating through the layers backwards, all delta values are computed:

$$\delta_c = \left(\sum_h \delta_h \times weight_{c,h}\right) \times \sigma'(net_c)$$

where h is a cell in the *next* layer in the ANN and c is the cell that the delta value is being computed for.

Equation 2.9: Computing the δ value for a hidden layer neuron.

After this, the computed delta values are used to adjust the weights in the ANN and the next activation of the model with the same input data will produce a lower error value.

2.2.5 *Gradients*

As shown above, gradients—partial derivatives—are calculated during the backpropagation phase to adjust the internal state into one that more aptly can reproduce the supervised solution to the problem—the target data. The adjustment done to the weights is proportional to the gradient; a gradient of larger magnitude means a larger adjustment made to the associated weight in the ANNs composition.

The *chain rule* is applied to compute the gradient for each node throughout the ANN (starting at an output node or neuron):

$$(f \circ g)' = (f' \circ g) \cdot g' = f'(g(x)) \cdot g'(x)$$

Equation 2.10: The chain rule, used for computing gradients in the ANN.

[3]The sum of squared errors is one method of calculating the total error, used for the sake of communicating the theory of neural network internals. It is not the only method viable for the purpose of calculating the total error value..

That is; how much a weight in the ANN needs to be adjusted is dependent on its gradient with respect to each node going back to the output node whose error is being minimized—the chain rule applied in sequence to each neuron along the way. For deeper ANNS, this is not problem-free, something that is discussed in the vanishing gradients section below.

2.2.6 *Optimization*

The gradients described above can be optimized in different ways. Not only are there several different error functions (also called *objective functions* or *loss functions*), but there are also several different methods for optimizing weights. The weight optimization function is called the gradient descent optimization method. This optimization method—not to be confused with the error or objective function—is the method used for minimizing the *output* of the error function through an iterative process (training).

STOCHASTIC GRADIENT DESCENT (SGD)

Perhaps the most basic of optimization methods, the Stochastic Gradient Descent is an incremental gradient descent, finding a local minima or maximum through an iterative process, where weights are adjusted little by little, repeatedly.

ROOT-MEAN-SQUARE PROPAGATION (RMSPROP)

The Root-Mean-Square Propagation is an improvement over the Stochastic Gradient Descent. Compared to the SGD, it introduces the concept of momentum, maintaining a more direct route towards a local minima (Tieleman and Hinton, 2012).

ADAPTIVE MOMENT ESTIMATION (ADAM)

Adaptive moment estimation—generally called Adam—is, in practice, a patched version of the root-mean-square propagation method. They are identical except for the fact that Adam uses running averages of both the gradients as well as their magnitudes (Kingma and Ba, 2014).

2.2.7 *Overfitting*

If the ANN is configured incorrectly in regard to the problem to be solved, *overfitting* can occur. It can be intuitively understood through visual representation (see below), but can also be described in the following way: Overfitting occurs when the ANN is too complex (i.e. consists of too many neurons, layers or a combination thereof) in regard to the training data volume and/or problem complexity: The multi-dimensional polynomial represented by the ANNS internal state intersects every data point in the training data—*but*—inbetween the data points, it deviates from the expected or reasonable positioning. This renders the ANN able to solve any problem presented in the training data perfectly, but *unable* to solve any problem that it has not already encountered in the training data. Overfitting can be solved by removing neurons from the network (i.e. simplifying the ANN configuration) or providing a greater volume of training data, which, in turn, fills up the spaces inbetween the previous training set data points, preventing polynomial deviation.

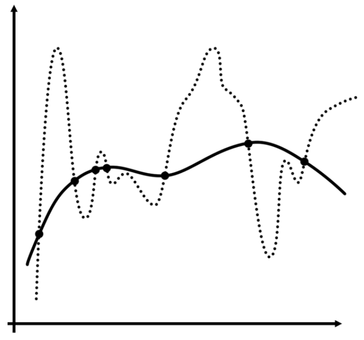


Figure 2.5: Overfitting—the solid curve is a more sane representation of the data points, compared to the dotted curve, which has been *overfitted.*

Another way of countering this problem is a method called *dropout* (Hinton et al., 2014). In effect, it does exactly what is prescribed as a solution above; reducing the complexity of the ANN. However, it does this not by reconfiguring the network permanently, but by *randomly disabling nodes during training*. That is, at the beginning of a training iteration, a certain portion (called the *dropout probability*) of the neurons is disabled, and the iteration is performed as if the disabled neurons were not part of the model. After the iteration has completed fully, the neurons are re-enabled and another set of neurons are selected randomly again to be disabled on the next training iteration. Dropout is only applied during training.

2.2.8 *Feature Extraction*

The extraction and selection of *features* can be very important depending on the problem. Rather than feeding a neural network model raw input data—data that has not been touched—which might work well depending on the problem being solved, certain computations are performed on the input data to create features. Feature extraction involves performing computations on the input data to create new features, making certain aspects of the input data more prominent to the neural network model as well as reducing redundancy. This, in turn, reduces training time, allowing the ANN to model the problem faster as well as reducing overfitting (Bermingham et al., 2015).

On top of feature extraction, feature *selection* is performed: Certain features are kept while others are discarded from the input data. For example, when features are highly correlated with each other, only one of the features might be kept, reducing the dimensionality, which also helps reduce training time and model overfitting.

Another important aspect regarding features needs to be brought up; feature *scaling* (also called *data normalization*). Feature scaling is a method used to standardize the range (i.e. $feature_{min}$ to $feature_{max}$) of features. For example, it is unwanted for one feature to have a very large range, and another, independent feature variable, to have a very small range. The larger a range is compared to other feature variable ranges, the more it will dictate the value of the error function, causing the error function to "focus" more on the large range features than on others, mistakingly ascribing the large range feature a greater importance in the optimization process, which leads to a much slower convergence of neural network weights.

All in all, feature extraction, selection and scaling play an important role in optimizing neural network training for specific problem solving tasks.

2.2.9 *Recurrent Neural Networks*

A *recurrent* neural network (RNN) is an artificial network that is "rolled out" in the sense that connections between neurons form a *cycle*. Essentially, what was understood as the *depth* in the ANN is more akin to the *width*—the length of the input sequence—of the RNN. While RNNS can also be deep, the architecture of (deep) RNNS could be viewed as multiple ANNS stacked with lateral connections between layers.

The RNN, rather than merely outputting values forward through the layers in the network (as does the ANN), output values are also fed back as inputs to previous layers, hence creating a feedback loop. This creates an internal state in the neural network—a short-term memory allowing information to persist or linger—which takes into account previous input values when processing newer ones. In practice, this means that a single, specific value can carry different meanings depending on previous inputs to the RNN, or expressed in another way; that specific value can mean something different depending on where, in the sequence of values, it is encountered. As a trivial example; in natural language, the same word carries different meanings depending on where, in a sentence, it occurs.

Perhaps self-explanatory, this is why RNNS work well for time series; each input affects the next iteration (essentially creating a *memory*), and so the order of the inputs becomes important, meaning that the RNN takes entire sequences into account when modeling a problem rather than single inputs. It is not necessary for the data points in the sequence to be evenly spaced in time, although that is the most common scenario depending on the problem being modeled.

There are several different RNN architectures, some more known and used than others. Due to the nature of RNNS and the application of the chain rule for calculating gradients, they have all been plagued by the vanishing gradients problem.

2.2.10 *Vanishing Gradients*

It turns out that the gradient in multi-layer neural networks is unstable. Not only that; the stability decreases exponentially with depth (or, in the case of RNNS, also sequence length).

As the chain rule is applied throughout the neural network—explained in the description of the backpropagation phase earlier—the delta values comprise a series of multiplications. Continuing the use of the logistic function from earlier in this example, the problem of vanishing gradients is easy to understand.

As the maximum derivative of the logistic function is $\frac{1}{4}$, a long series of multiplications throughout the nodes in the neural network converge to zero.

$$\sigma'(x) = \frac{e^x}{(1+e^x)^2}$$
$$\sigma'(0) = \frac{1}{4}$$

Equation 2.11: The derivative of the logistic function and its maximum value at $x = 0$.

In an ANN, the depth (width or *roll out* in the LSTM) dictates the length of the multiplication series: In the last hidden layer, for example, the chain rule gives the following gradient computation:

$$\begin{aligned}\delta_c &= \frac{\partial E_{total}}{\partial out_c} \times \frac{\partial out_c}{\partial net_c} \\ &= \frac{\partial E_{total}}{\partial out_c} \times \sigma'(net_c) \\ \frac{\partial E_{total}}{\partial weight_c} &= \delta_c \times \frac{\partial net_c}{\partial weight_c}\end{aligned}$$

Equation 2.12: Gradient calculation formula.

As the gradient is computed for cells farther upstream in the neural network, the multiplication becomes longer and longer—more and more delta values are multiplied together, resulting in a convergence ($\frac{1}{4}^n$ where n is the number of cells along the path from the current node to the output node, ignoring other multiplicative factors) towards zero in the limit.

2.2.11 *Long Short-Term Memory*

In 1997, Hochreiter and Schmidhuber introduced the Long Short-Term-Memory (LSTM). Building upon the traditional RNN, they introduced three *gates* that manage the internal cell state: The input, output and forget gates.

During training, these gates are used to decide how to adjust the internal state to adapt it to the information being fed to the LSTM. For example, the first step is to decide how much of the alread-existing internal cell state to throw away. This is managed by the forget gate. Then, the input gate decides which of the internal values to update, after which the internal state is adjusted and adapted to the new information. Lastly, the output gate decides what the LSTM will output. Although this output is based on the internal state, it is not the entire state—rather, how much of the internal state, and what parts of the internal state, are output, is managed by the output gate.

These gates work together to carefully protect the internal state from the vanishing gradient problem—the forget gate can decide to "turn off" adjustments completely for many training iterations if they are not deemed necessary by the LSTM, which in turn reduces the problem and vastly increases the information persistence in the internal state.

It should be noted, however, that the problem is not fundamentally solved, but rather, its effect is reduced to the point of LSTM-based RNNS being *much* more competent at modeling long-term temporal relationships in the input sequences compared to traditional RNNS.

2.2.12 *Sequence-to-Sequence Learning*

Sequence-to-sequence learning implies that the network is being trained to produce *sequences* as output, from input sequences. Rather than producing a single output data point predicted to follow the input sequence, the LSTM produces an output *sequence*, predicted as the sequence to follow the provided input sequence. For example, provided with a natural language sentence of words, the entire following sentence (sequence of words) would be predicted. This approach is also used in machine translation (Sutskever et al., 2014); an input sentence is provided, and an output sentence is predicted in the foreign language. For financial time series, this implies feeding the LSTM with a sequence of data points and training it on the *sequence* of data points following the input sequence. That is, the LSTM is provided with (for example) five minutes of data as the input sequence and the following next five minutes of data as the target data sequence. Sequence-to-sequence learning, thus, involves a many-to-many mapping of the time series data.

2.3 Algorithmic Trading

Algorithmic trading (also known as algo-trading, automated trading and black-box trading) is a way of automating trading by delegating market entry and exit decisions to an algorithm (i.e. letting an algorithm decide when to buy, sell or short a certain stock).

2.3.1 *Black-box Trading*

Although an algorithm is normally thought of as being constructed by human reasoning with steps laid out in sequence, it should now be understood as a machine learning algorithm learning to construct solutions to various problems. The ANN models the problem, emulating a way of reasoning about it. Thus, algorithmic trading ties together the knowledge presented in the theory chapter, building upon it to create a machine learning algorithm attempting to understand aspects of financial market behavior.

The term *black-box trading* hints at this; the intricacies of a trained neural network cannot be understood (or, put in a simpler way, it is not known why the ANN takes certain decisions, and it cannot be known easily by looking at the internal state of the ANN without actually activating it with some input data). In practice, the internal state of the ANN represents an algorithm that produces a conclusion in regard to a presented problem.

Despite this, it is expected that the ANN models market behavior to some extent to provide relevant output data.

Since algorithmic trading algorithms can be fully automated, under the right circumstances (for example, low data transfer latency to the broker and a high level of computational power together with an effective algorithm [in terms of market behavior prediction performance]), returns could potentially be generated arbitrarily quickly, though being limited by the available trading capital and market liquidity. This fact is backed up by real-world algorithmic trading algorithms generating positive net returns, although most of these models are trade secrets.

2.3.2 *Non-stationarity*

Machine learning algorithms, used in a trading scenario as explained above, model the trading problem by relying on recurring patterns in the input data. Due to non-stationarity, two identical financial time series input sequences acquired at two different points in time may have different solutions (i.e. output sequence predictions). Just as a human trader continously needs to adapt to changing market conditions, an automated trading system also needs to continously adapt its behavior.

This problem is a fact of complex time series and cannot be fully overcome, but can be countered to some degree by, for example, using input time series from a fairly short period of time if the neural model is not complex enough: The market behavior today is vastly different compared to before the stock market crash in 2008—just as human traders cannot use the same strategies today as they used back in 2007, a machine learing trading algorithm trained on data from that time period would have developed a strategy that does not work in today's market.

— Lastly, with all knowledge in the theory chapter in mind, the methodology chapter and our particular experiments will become easily understandable, as well as issues encountered during our research.

3 Related Work

As much as we were inspired by related work, we also encountered the problem of breaking new ground during our research. Especially in regard to the application of the LSTM-based RNN on financial markets using sequence-to-sequence learning, finding related work turned out to be somewhat difficult.

3.1 Financial Market Predictions

Enormous amounts of research has been made into the prediction of financial markets, and we have waded through a large volume of it during the thesis work. We found Vahala (2016) particularly helpful in approaching the problem of financial market prediction using the methods in this thesis.

When it comes to feature extraction, we did not have the time to test all features that we felt were applicable to the problem. A very interesting and inspiring piece of work in regard to feature extraction is Chan (2013).

3.2 Sequence-to-Sequence Learning

The bulk volume of research on sequence-to-sequence learning has been done on machine translation. Although algorithmic trading is an entirely different problem, the model configuration is very similar in its setup. The theory behind sequence-to-sequence learning with an LSTM model is covered by Sutskever et al. (2014), wherein the practical application of sequence-to-sequence learning is explained as well.

Chollet (2015)—specifically the Keras documentation—provides a deep insight into the intricacies of sequence-to-sequence learning and its practical applications.

Although modeling the stochastic processes on financial markets is a different (and perhaps more complex) problem compared to modeling long-term weather, Zaytar and Amrani (2016) provides a deeper insight into the theory of the LSTM-based RNN, detailing it further than what was done in this thesis, as well was providing a solid foundation for understanding the use of the LSTM in modeling and predicting stochastic processes.

4 Methodology

Up to this point, the theory behind algorithmic trading has been explained thoroughly. In this chapter, the methods and approaches used to carry out the experiments will be detailed.

4.1 Approach

There are, of course, different ways to approach the thesis problem and answer the research question. The approaches we picked for setting up the experiments is not better or worse than other approaches in any definite or absolute manner. The research question relies heavily on real-numbered evaluation measures, narrowing down the number of feasible approaches to the quantitative kind. The candidate approaches and the rationale behind our final picks are laid out below.

4.1.1 *Candidate Approaches*

We discussed several different approaches during the initial stages of our thesis research. Eventually, we came up with several candidate approaches, some more relevant than others. Many were discarded quickly, while others were discussed thoroughly, with the more relevant ones detailed below.

autoencoder feature extraction

Applying an autoencoder to the raw input data would produce an interesting "compressed" view of the data (bounded by entropy), potentially allowing us to keep only relevant data. This is mentioned in the Future Work chapter as it is still a very interesting approach to the problem. We decided against it due to constraints on time and computational resources.

implementing a trading strategy

The ultimate measure of performance of predictions, building a trading strategy would give definite answers to the question regarding predicted market behavior. Again limited by time, this was decided against.

statistical evaluation measures

There are many statistical evaluation measures to choose from. For the sake of the thesis problem—comparing the lstm against the traditional rnn—we found mape and rmse to be relevant. We decided to implement this approach as it provides real-numbered evaluation measures, giving reproducible and testable results.

visualized graphs

Although not adequate for answering the research question on their own, graphs provide an intuitive sense for the problem and our models' performance. For the skilled technical trader, this intuitive approach can be as useful as real-valued measures. This approach was also implemented.

4.1.2 *Selected Approaches*

There are a couple of reasons for our particular choice of approach in investigating the research question; first off, it is important that real-numbered measures are presented as a way of displaying definite and reproducible evaluation of the models' performance. Secondly, although trading algorithms rely entirely on *data*, visualizing the results can give an intuitive sense of the models' ability to predict market behavior: Although these graphs do not answer any questions definitively, they serve the purpose of reinforcing conclusions drawn from evaluation measures.

4.1.3 *Discarded Approaches*

There are many other methods for evaluating the performance with the most important aspect of the predictions being whether they can be used for successful trading on the Foreign Exchange Market. For example, a buy-and-hold decision based on predictions could be simulated, on which returns could be computed, giving a trading performance estimate of the predictions. Unforunately, there was not enough time to perform such experiments.

4.2 Technology

A plethora of techonology exists to aid in performing our particular thesis experiments and although there are many viable alternatives, a choice had to be made. The rationale behind most choices made in regard to technology was popularity (i.e. avoiding esoteric or unknown alternatives), availability, performance and relevance to the scientific community, thus facilitating reproducibility of our experiments and results to a greater extent.

4.2.1 *Docker*

A lesson learned; the same technology sometimes works different on different computers. Thankfully, Docker enabled us to reproduce and deploy an identical environment on different machines, making collaboration much easier. Take note!

Although not required to perform the experiments, Docker made it so much easier to collaborate using different hardware configurations, allowing everyone involved to reproduce the setup on their machine.

4.2.2 *Python*

We used Python 3 (specifically version 3.5.2+) as the programming language of choice as it is ubiquitously used by the scientific computing community and academics all over the world. Documentation is rich and there is a vast amount of tools and third-party libraries available.

4.2.3 *Keras and TensorFlow*

Keras—a high-level deep learning library (Chollet, 2015)—was used to implemented the machine learning algorithm models. It is ideal as it provides simple setup and execution of machine learning algorithms, making it almost trivial to perform practical research within the field.

A TensorFlow (Abadi et al., 2015) backend was used in conjunction with Keras, enabling us to easily perform computations on the computer's GPU and increasing computational performance many times over.

Much happened during the time of writing the thesis in regard to the development of both Keras and TensorFlow. Despite both libraries providing rich documentation, they both changed to such a degree during the writing of the thesis that we had to rework our source code many times over. During the first weeks of writing the thesis, TensorFlow was not available on PyPI[1] and so we had to compile it manually. As of finishing this thesis, both Keras and TensorFlow can be installed easily via the Python package manager.

4.2.4 *Other Third-party Libraries*

For computational efficiency, pandas and numpy were our go-to solutions, providing native code computational efficiency in Python. Graphs and visualizations were done with matplotlib. Keras uses h5py for saving and loading models in the HDF5 file format.

4.3 Data

Although there are many different financial assets and financial markets with various degrees of liquidity etc, we needed freely available high-frequency data to perform intraday trading experiments. This fact limited our choices of data sources, so many were ruled out early in the process.

4.3.1 *Selection*

The requirement on our part of high-frequency data to carry out the experiments led us to the FOREX market, for which freely available market data can be acquired. We obtained the data from Dukascopy[2] by opening a trading account and downloading bulk, high-frequency data. The currency exchange between the Euro (€) and the U.S. dollar ($) was our choice of data for model training as it constitutes a highly liquid financial market.

Beyond the selection of a data source, we had to limit the amount of data to put through the neural model, being limited by computational resources. We initially decided to build a model for prediction of market behavior by looking at the past three months of financial data (November through January), but later decided on using the two weeks of trading data prior to the trading day to perform predictions on.

The acquired data contained tick data, i.e. entries for each *pip* change, thus providing us with asks and bids, market volume and timestamps. We did not have access to the *market depth*—the number of bids and asks at each price level.

4.3.2 *Preprocessing*

During preprocessing, irrelevant parts are removed, missing values are filled in[3], normalization and regularization is done and so on, essentially compiling the data to a set useful for computing features on and using for model training.

All parameters relevant to training of our network model—considering our technical analysis approach—were already part of the acquired high-frequency data.

The pip data was aggregated into one-minute candlesticks, containing OHLC prices, with missing candlesticks forward-filled from the previously existing candlestick in order to avoid look-ahead bias.

The magnitude of price changes on the FOREX market are small—which poses a problem for the neural network model as explained in the theory chapter—but it could not be normalized over an entire data set as that would violate the rule of not looking into the future, since, in a live trading scenario, the model would only have access to already-seen prices, which would make it impossible to normalize the price with future prices in mind. Instead, this was solved during feature extraction through careful choice of features.

[1] PyPI is the Python Package Index, a repository of software for the Python programming language that can be installed via the `pip` command.

[2] Website: https://www.dukascopy.com/

[3] Depending on the problem being solved, missing values can be ignored, interpolated or forward-filled. In our case, forward-filling was the only viable alternative; using previous values and copying them forward in time to fill the holes in order to avoid look-ahead bias.

4.4 Evaluation

The models were evaluated in several ways. Partly, the models' predictions were visualized to provide an intuitive overview of their performance. To provide a real-numbered estimate of their performance, measurements were computed on their predictions.

4.4.1 *Visualization*

Although merely giving an intuitive sense of the models' performance, visualizing the results still provides an important insight into to the performance of the models. The graphs are interesting to look at in comparison to each other; it is evident that the LSTM model is superior to the traditional RNN, displaying a higher ability in predicting market behavior *(see Appendix A, B, C and D).*

Furthermore, these graphs were used to confirm the integrity of the models—that they did what they were supposed to do.

4.4.2 *Evaluation Measures*

A more important aspect of the model evaluation process is the quantification of the models' performance, using real-valued measurements. Using the mean absolute percentage error (MAPE) and root-mean-square error (RMSE), the models were evaluated and compared against each other. The models were evaluated at several points in time to avoid serendipitous results.

mean absolute percentage error (mape)

The Mean Absolute Percentage Error is a measure of prediction accuracy of a forecasting method. It is used to compute the accuracy of a trend estimation as a percentage.

$$M = \frac{100}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

where A_t is the actual value and F_t is the forecast value

Equation 4.1: The mean absolute percentage error formula.

root-mean-square error (rmse)

The Root-Mean-Square Error is a measure of difference between actual values and values predicted by a model. It represents a standard deviation of the differences between actual predicted values.

$$RMSE(\hat{\theta}) = \sqrt{MSE(\hat{\theta})}$$

where MSE is the mean square error of the predicted values.

Equation 4.2: The root-mean-square error formula.

5 Experiments

The experiments were carried out many times over. Partly because the model was being tweaked to provide better predictions, partly because new features were being tested and partly because discussion and literature studies during the thesis research work pointed us in new directions. Regardless, presented below are our final approaches to the problems and their setups.

5.1 Models

The choice of neural network types were decided from the start—the LSTM-based RNN and the traditional RNN–but the network architecture (layers, cells, dropout probability etc) was tweaked and adjusted many times over. We began our work by reading literature, building upon previous researchers experience, eventually coming up with a model setup that seemed adequate for our kind of approach. Beyond that, we intuitively added and removed layers and cells in each layer, trying to understand the setup better.

5.2 Model Configurations

Our final setup for the LSTM was a model with six layers; an input layer with 2048 cells, a dropout layer with 20% dropout probability, a hidden layer with 1024 cells, another dropout layer with 20% dropout probability, another hidden layer with 512 cells and a final layer with the same dropout probability again, ending with a time-distributed dense output layer with linear activation functions.

For the RNN, the final configuration was again six layers; an input layer with 1024 cells, a dropout layer with 20% dropout probability, a hidden layer with 512 cells, another dropout layer with 20% dropout probability, another hidden layer with 256 cells and a final layer with the same dropout probability again, ending with a time-distributed dense output layer with linear activation functions, thus having the same configuration as the LSTM.

5.3 Sliding Window

The models were setup to take input sequences of ten data points and to predict the ten minutes following the input sequence, thus configured as a many-to-many (*10-to-10*) neural prediction model. We used a sliding window with a width of ten data points—representing ten minutes of data—and moved it forward one data point (one minute) for each training iteration.

5.4 Features

Using high-frequency data, all features used were based on a technical analysis approach, thoroughly discussed in the theory chapter. Although several different feature configurations were attempted, we eventually settled on the ones detailed below. All were normalized by using an online normalization function, Equation 5.1

$$norm(x, t) = \frac{x_t - x_{t-1}}{x_{t-1}}$$

Equation 5.1: The normalization function used.

returns

$$F_1(t) = norm(C_{bid}, t)$$

Equation 5.2: The first extracted feature—close bid returns.

Where C_{bid} is the close bid of an OHLC data point in the sequence at time t.

volume

$$F_2(t) = norm(SMA^5(V_{bid}), t)$$

Equation 5.3: The second extracted feature—a five-point period simple moving average of the market bid volume.

Where V_{bid} is the bid volume.

relative strength index

$$F_3(t) = norm(RSI^{10}(C_{bid}), t)$$

Equation 5.4: The third extracted feature—a ten-point relative strength index computed on the close bid.

spread

$$F_4(t) = norm(Spread, t)$$

Equation 5.5: The fourth extracted feature—the market price spread.

The features were then scaled by hand to provide a standardized range across the independent variables; each feature was scaled independently on a few training datasets to produce values of roughly equal range, as well as to keep values within the range of −1 and 1. This was done on training data to prevent introducing future data into the model, thus, the normalization might not produce values of equal range or within the limits for test data to the same degree. It proved better than not performing this "manual" normalization at all, without which weights would have become skewed in the model.

5.5 Model Training

Each model was trained for a set amount of iterations; 10,000. It should be noted that the LSTM is more computationally intensive and thus required more time to reach the same number of iterations.

Training was done on the fourteen consecutive trading days prior to the prediction day. Predictions were limited to February 9th, 2017 to avoid encountering many different marking regimes during the training data time period, potentially simplifying generalization of market behavior and thereby potentially increasing model performance.

5.6 Hardware

Training was done for many different setups on several different machines during our research. For the sake of completeness, the hardware specification of the machine used to train our final models used in the experiments is presented in Table 5.1 below.

Motherboard	MSI Z97−5
Memory	Hyper-X Savage 16GB
Graphics card	MSI Nvidia Geforce GTX 760 2GB
Processor	Intel i5−4690K
Storage	Samsung 840 EVO 250GB

Table 5.1: The hardware confguration of the system used for model training.

6 Results

Below, our results are presented. See the appendix for graphed visualizations of the predictions. All predictions are made on FOREX market data for the EUR/USD currency pair from February 9th, 2017—the day following the fourteen consecutive trading days of data that the models were trained on.

6.1 Measurements

	06.00–07.00		07.00–08.00	
	LSTM	RNN	LSTM	RNN
MAPE	**51.0738%**	135.0695%	253.9935%	**237.0991%**
RMSE	**0.0261**	0.2096	**0.0455**	0.0981

	08.00–09.00		09.00–10.00	
	LSTM	RNN	LSTM	RNN
MAPE	**147.7115%**	154.5905%	131.6576%	**120.2149%**
RMSE	**0.0592**	0.1735	0.1857	**0.1843**

	10.00–11.00		11.00–12.00	
	LSTM	RNN	LSTM	RNN
MAPE	**109.234 %**	377.7257%	**250.9250%**	348.9639%
RMSE	0.3678	**0.1159**	**0.1083**	0.1265

	12.00–13.00		13.00–14.00	
	LSTM	RNN	LSTM	RNN
MAPE	**152.8492%**	217.5117%	**170.5649%**	225.3889%
RMSE	**0.1166**	0.1689	**0.1001**	0.1468

	14.00–15.00		15.00–16.00	
	LSTM	RNN	LSTM	RNN
MAPE	**159.9678%**	168.4376%	**166.1331%**	255.8209%
RMSE	**0.1472**	0.1836	**0.2515**	0.2559

Table 6.1: Measurement results over a day of trading. The better result is denoted with **bold** numbers.

7 Discussion

Looking at the graphs (see Appendix A, B, C and D), already is it evident that the application of our LSTM-based RNN using sequence-to-sequence learning is vastly superior to the traditional RNN model. This is also confirmed by looking at the evaluation measurements (Table 6.1) performed on the predictions: The LSTM outperforms the RNN on most evaluation measures, proving to be more much more apt at modeling market behavior compared to the traditional RNN.

7.1 Financial Market Prediction

Reaching back to the original research question; using the set of features selected for the experiments setup in this thesis, the LSTM-based RNN was unable to model market behavior to such a degree that a definitive call could be made as to whether the model generalized for the market regime encountered during training or not.

Many attempts have been made at predicting financial market behavior by modeling price returns, with some displaying a certain level of success. In any case, a larger data set could potentially provide more information, allowing the LSTM generalize better for market behavior. That, however, would also require a much larger model (i.e. a deeper model configuration with many more cells in each layer) which, in turn, would place further demands on computational capacity and time, which, in the end, put a limit on our ability to pursue the problem further. The more complex the model, and the larger the dataset, the worse the RNN performs in comparison to the LSTM.

7.2 Visualizations

Although the predictions are visualized *(see Appendix A, B, C and D)* in the form of graphs, there are interesting things to note. Below, we reason about the behavior of the models' ability in predicting market behavior.

The vanishing gradient problem becomes somewhat evident in the graphs (see Appendix A, B, C and D). The RNN is unable to produce much more than a simple recurring pattern.

Appendix A

The predictions produced by the LSTM (solid line) are intuitively accurate and could possibly have been used for trading. The large fall is not predicted by the model, but the predicted price movements still follow the market trend well up to that point.

Looking at the RNN (dashed line) predictions, there is no evident generalization of market behavior.

Appendix B

Again, we see the general market trend predicted to an extent by the LSTM, giving an interesting result.

Looking at the RNN predictions again, we find no evidence of the market behavior being modeled to any useful extent.

Appendix C

Although not giving accurate predictions, the LSTM seems to have modeled the *double top chart pattern*: After two tops and a price fall breaking the bottom of the trend channel; the model predicts a continuation of the price drop, as would likely be expected by a human trader performing technical trading analysis on the chart.

The RNN produces no useful predictions.

Appendix D

Here, we see the LSTM model the general market trend over the prediction period and onwards, but missing an important opportunity for profitable returns; the large price movement during the prediction period. The LSTM did not model the market behavior well enough.

The RNN produces no useful predictions.

8 Conclusion

Looking at the results, it is obvious that the LSTM-based RNN performed much better than the traditional RNN. We conclude that, given the same time series data, on the premises in the experiments, the LSTM model is much more competent at generalizing the time series problems. The RNN is greatly improved upon by the LSTM thanks to its handling of the vanishing gradient problem.

The LSTM displayed some ability in modeling market behavior (see chapter Discussion) while the RNN could not handle even the relatively small amount of input data used in the experiments. For complex problems requiring deep and/or wide neural network architectures, the RNN performs worse compared to the LSTM. However, given a simple enough input problem, the RNN proves to be a viable alternative under certain circumstances given the fact that it places a lower demand on computational resources, potentially modeling the problem faster (measured in real-world time) compared to the LSTM. This, however, only applies to problems of trivial difficulty relative to the complex problems that the LSTM is able to model.

The predictions in the graphs are interesting, and although some promise of modeling market behavior is displayed, it is likely that the model is encountering a pattern similar to one seen during the training data. A larger model with more input data would likely have modeled the behavior of the market to a greater extent, producing even better predictions of price movements.

In retrospect—now that all experiments have been performed and their results compiled—our firm belief is that q-learning is more suited to the type of problem posed in this thesis (Gabrielsson and Johansson, 2015). Although we have neither proved nor disproved the random walk hypothesis (Fama, 1995), the statistical distribution of price market movements are *(a)* too complex to model using the small-scale LSTM models in our setup, *(b)* require large amounts of data and *(c)* consume—given appropriate configuration with respect to layers and number of cells as well as training data volume—enormous computational resources.

9 Future Work

Limited, as we all are, by time, there was no opportunity to test all ideas that we had. It would be interesting to continue working with this, approaching different aspects of the problem in new ways. Below are some ideas that we had during the time of writing this thesis.

9.1 Trading Performance Estimates

It would be interesting to performance more in-depth experiments on the predictions such as building a trading strategy upon the models' predictions and evaluating trading performance.

9.2 Q-learning Approach

Although some interesting results were attained with respect to market behavior predictions, it would be even more interesting to approach the problem with q-learning. Q-learning is not about predicting future values (i.e. returns etc.) but rather about predicting appropriate *behavior*. In regard to the thesis problem, this implies predicting appropriate market *actions* or *operations*; predicting when to buy or sell a financial asset. Although the problem has been approached with q-learning before (Gabrielsson and Johansson, 2015), we were unable to find any publications that have approached it with a deep learning model that solves the vanishing/exploding gradient problem, i.e. the LSTM-based RNN used in this thesis. Combining q-learning with a LSTM-based RNN would be very interesting indeed.

9.3 Exploring Larger Models

Due to time constraints, training was limited to relatively small datasets and models. A larger model, although requiring significantly more training time and computational resources, might be able to provide more accurate predictions than the model presented in the results in this thesis.

9.4 Trying Different Features

The market data contains all relevant data for a strictly technical analysis, but feature extraction and selection makes relevant data aspects more prominent. Ceteris paribus, finding a better set of features could produce more accurate predictions.

Although not necessarily well-suited for regression, it would be interesting to see the results of training a model on the seven features selected by Gabrielsson and Johansson (2015).

9.5 Autoencoder Feature Extraction

Instead of manual feature extraction, backtesting on a large amount of data, one could hypothetically build an autoencoder to compress the data, limiting the autoencoder to fewer and fewer cells and layers until a minimal working configuration has been found, and then use it for feature extraction and using the output as input for the LSTM model. It would be very interesting to see whether the LSTM could model market behavior using the autoencoder output as features.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bermingham, M. L., Pong-Wong, R., Spiliopoulou, A., Hayward, C., Rudan, I., Campbell, H., Wright, A. F., Wilson, J. F., Agakov, F., Navarro, P., et al. (2015). Application of high-dimensional feature selection: evaluation for genomic prediction in man. *Scientific reports*, 5.

Chan, E. (2013). *Algorithmic Trading: Winning Strategies and Their Rationale*. Wiley, 1 edition.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Connors, L. and Alvarez, C. (2008). *Short Term Trading Strategies That work*.

Devakunchari, R. (2014). Analysis on big data over the years. *International Journal of Scientific and Research Publications*, 4(1).

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

Gabrielsson, P. and Johansson, U. (2015). High-frequency equity index futures trading using recurrent reinforcement learning with candlesticks.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

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

Hochreiter, S. and Schmidhuber, J. (1997). Long short–term memory. *Neural Computation*, 9(8):1733–1780.

Horsley, S. (1772). The sieve of eratosthenes. being an account of his method of finding all the prime numbers, by the rev. samuel horsley, f. r. s. *Philosophical Transactions*, 62(1775):327–347.

Kaya, O. (2016). High-frequency trading—reaching the limits.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kirkpatrick, C. D. and Dahlquist, J. R. (2010). *Technical Analysis: The Complete Resource for Financial Market Technicians*. Financial Time Press, 2 edition.

Kita, E., Zuo, Y., Harada, M., and Mizuno, T. (2012). Application of bayesian network to stock price prediction.

Nison, S. (2001). *Japanese Candlestick Charting Techniques*. Prentice Hall Press, 2 edition.

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Saad, E. W., Prokhorov, D. V., and II, D. C. W. (1998). Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks.

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of business*, 39(1):119–138.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2).

Tunnicliffe-Wilson, G. (1989). Non-linear and non-stationary time series analysis. *Journal of Time Series Analysis*, 10(4):385–386.

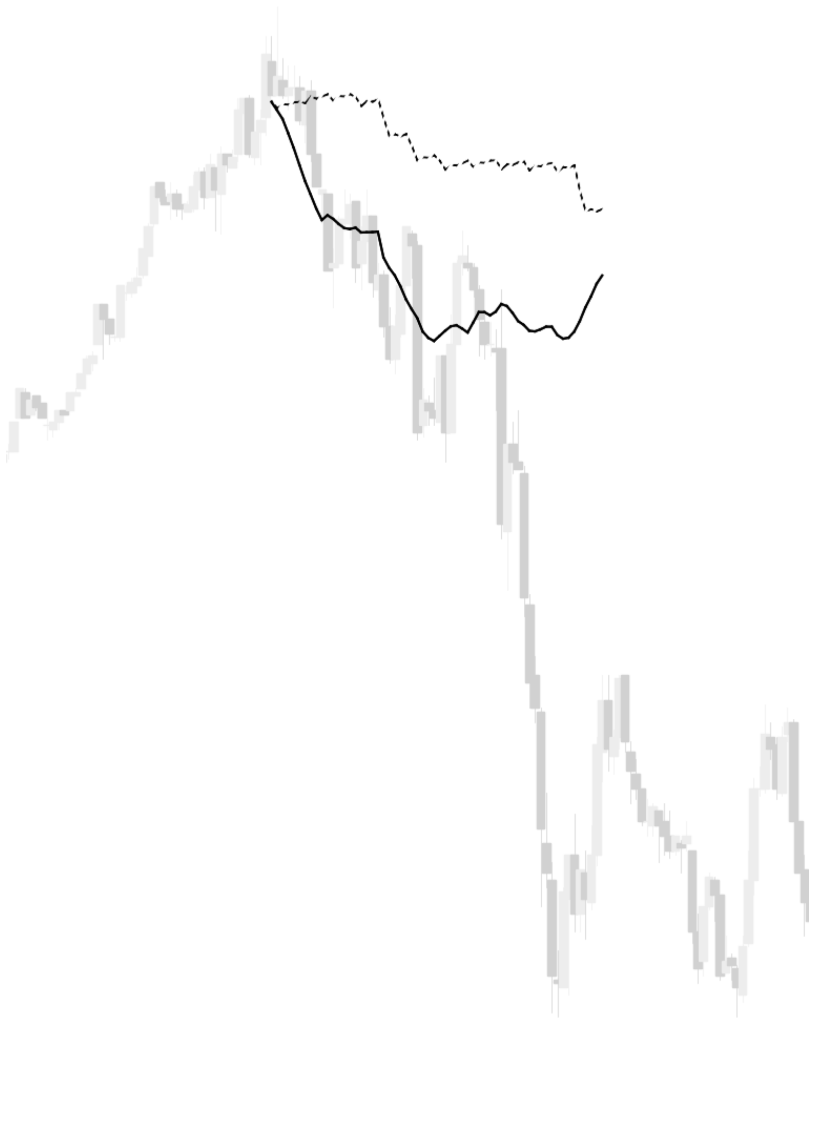
Vahala, J. (2016). Prediction of financial markets using deep learning.

Wilder, J. W. (1978). *New Concepts in Technical Trading Systems*. Trend Research, 1st edition.

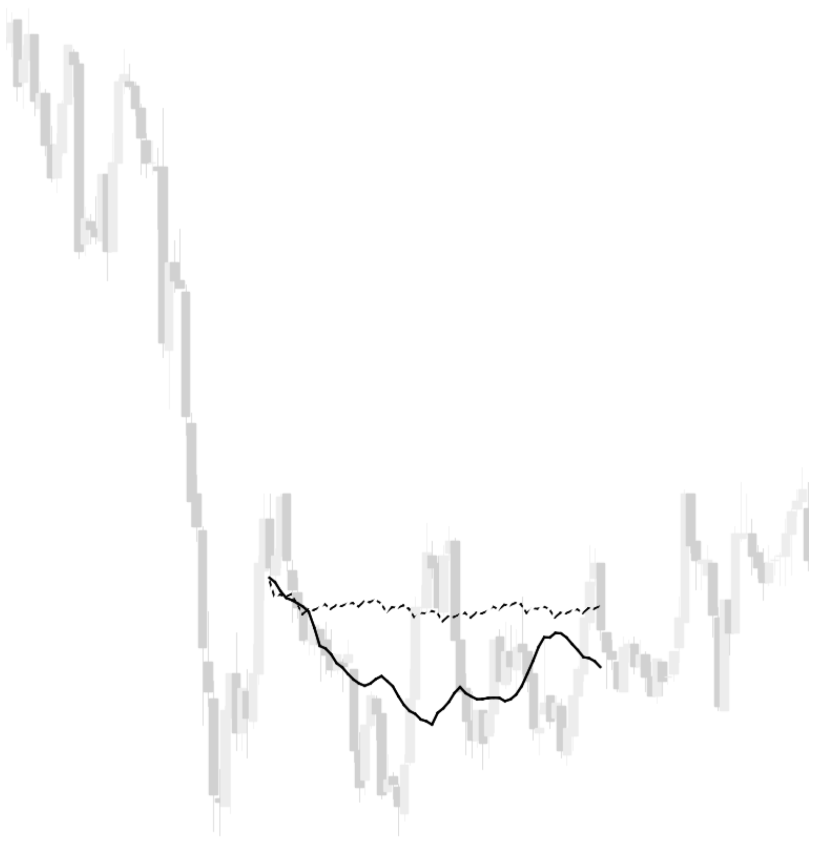
Zaytar, M. A. and Amrani, C. E. (2016). Sequence to sequence weather forecasting with long. *International Journal of Computer Applications*, 143(11).

Zhang, Y. (2004). Prediction of financial time series with hidden markov models.

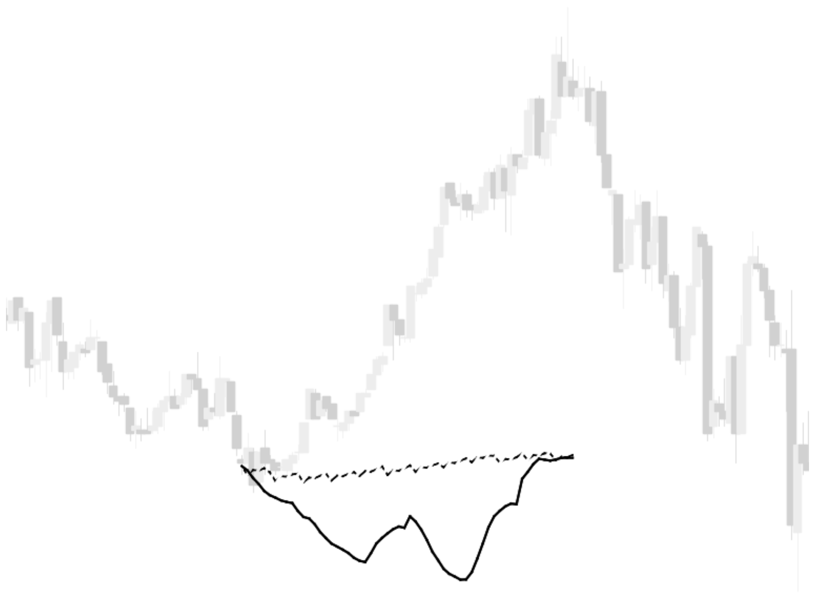
Appendix A



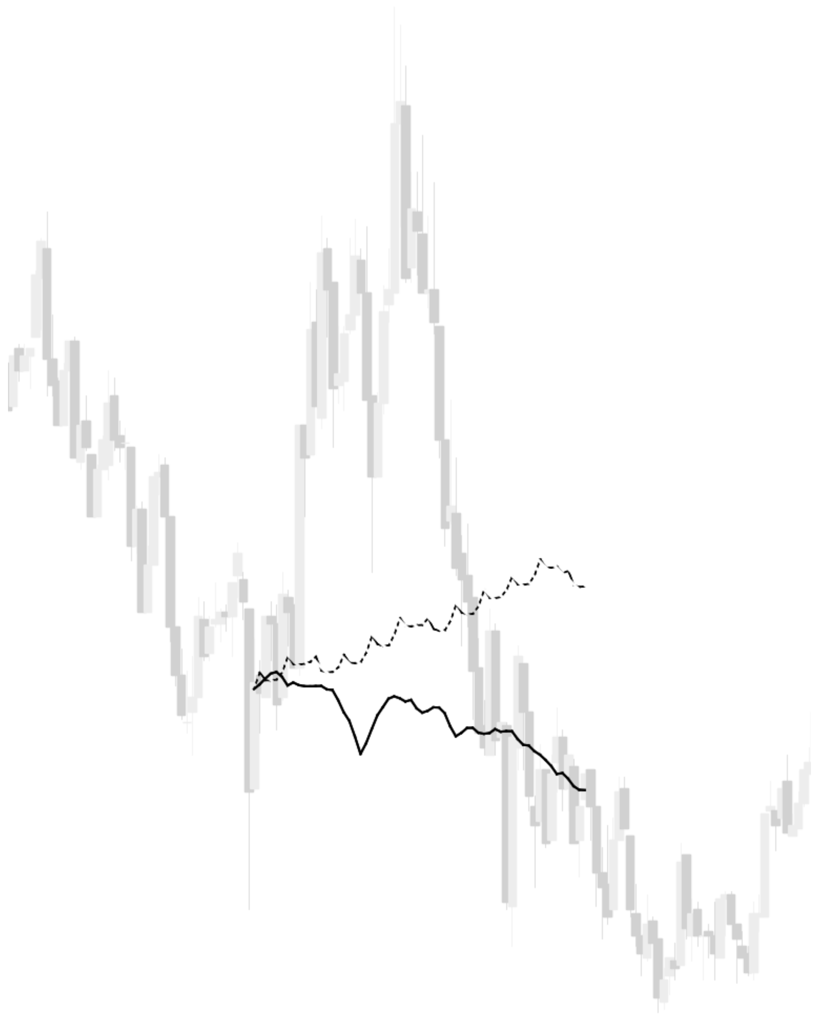
Appendix B



Appendix C



Appendix D



This page intentionally left blank.

Högskolan i Borås är en modern högskola mitt i city. Vi bedriver utbildningar inom ekonomi och informatik, biblioteks- och informationsvetenskap, mode och textil, beteendevetenskap och lärarutbildning, teknik samt vårdvetenskap.

På **institutionen Handels- och IT-högskolan (HIT)** har vi tagit fasta på studenternas framtida behov. Därför har vi skapat utbildningar där anställningsbarhet är ett nyckelord. Ämnesintegration, helhet och sammanhang är andra viktiga begrepp. På institutionen råder en närhet, såväl mellan studenter och lärare som mellan företag och utbildning.

Våra **ekonomiutbildningar** ger studenterna möjlighet att lära sig mer om olika företag och förvaltningar och hur styrning och organisering av dessa verksamheter sker. De får även lära sig om samhällsutveckling och om organisationers anpassning till omvärlden. De får möjlighet att förbättra sin förmåga att analysera, utveckla och styra verksamheter, oavsett om de vill ägna sig åt revision, administration eller marknadsföring. Bland våra **IT-utbildningar** finns alltid något för dem som vill designa framtidens IT-baserade kommunikationslösningar, som vill analysera behov av och krav på organisationers information för att designa deras innehållsstrukturer, bedriva integrerad IT- och affärsutveckling, utveckla sin förmåga att analysera och designa verksamheter eller inrikta sig mot programmering och utveckling för god IT-användning i företag och organisationer.

Forskningsverksamheten vid institutionen är såväl professions- som design- och utvecklingsinriktad. Den övergripande forskningsprofilen för institutionen är handels- och tjänsteutveckling i vilken kunskaper och kompetenser inom såväl informatik som företagsekonomi utgör viktiga grundstenar. Forskningen är välrenommerad och fokuserar på inriktningarna affärsdesign och Co-design. Forskningen är också professionsorienterad, vilket bland annat tar sig uttryck i att forskningen i många fall bedrivs på aktionsforskningsbaserade grunder med företag och offentliga organisationer på lokal, nationell och internationell arena. Forskningens design och professionsinriktning manifesteras också i InnovationLab, som är institutionens och Högskolans enhet för forskningsstödjande systemutveckling.

HÖGSKOLAN
I BORÅS

BESÖKSADRESS: JÄRNVÄGSGATAN 5 · POSTADRESS: ALLÉGATAN 1, 501 90 BORÅS
TFN: 033-435 40 00 · E-POST: INST.HIT@HB.SE · WEBB: WWW.HB.SE/HIT