Degree Project

Using LSTM Neural Networks To Predict Daily Stock Returns

Author: Jon Cavallie Mester
Supervisor: Daniel Toll
Semester: VT2021
Subject: Computer Science

Abstract

Long short-term memory (LSTM) neural networks have been proven to be effective for time series prediction, even in some instances where the data is non-stationary. This lead us to examine their predictive ability of stock market returns, as the development of stock prices and returns tend to be a non-stationary time series. We used daily stock trading data to let an LSTM train models at predicting daily returns for 60 stocks from the OMX30 and Nasdaq-100 indices. Subsequently, we measured their accuracy, precision, and recall. The mean accuracy was 49.75 percent, meaning that the observed accuracy was close to the accuracy one would observe by randomly selecting a prediction for each day and lower than the accuracy achieved by blindly predicting all days to be positive. Finally, we concluded that further improvements need to be made for models trained by LSTMs to have any notable predictive ability in the area of stock returns.

Keywords: LSTM, RNN, stock prediction, deep learning

Contents

1 Introduction

This is a 7.5 HEC B-level thesis in Computer science investigating the accuracy of Long short-term memory networks, an architecture for deep neural networks that allow them to retain a long and short-term memory, for predicting whether the future daily return of Swedish stocks in the OMX30 index and American stocks in the Nasdaq-100 index, will be positive or negative.

1.1 Background

1.1.1 Stock Trend Prediction

Stock shares are fractions of ownership in a corporation, that typically grant the owner a right to a part of the company's earnings, proceeds from liquidation events, and voting rights. These shares can be bought and sold privately or on stock exchanges. With time, the price of a stock moves up or down. Investors and traders generally buy stocks they believe will move upward in price and sell stocks they believe will move downward. The return of a stock can be expressed as the relative change of its price over time, and the daily return is the relative change over one day.

Investors have attempted to predict the trend of stocks to varying success for a long time. One could take a fundamental approach and look at the company, its leadership, and the market demand to conclude whether the business will grow and have it reflected in its stock valuation. There is technical analysis, which is when traders look at a price chart to find patterns that in the past have been associated with the positive trends of stocks. Another way is to take a quantitative approach, focusing on the historical movement of a stock's price and trading volume to predict its future using statistical methods such as linear regression or machine learning. However, the use of statistical methods has proven to be challenging, as the change of stock price is both non-linear and non-stationary, and traditional models require the time series to be stationary to accurately make any forecasts [1].

Due to the challenges of predicting the behavior of stock prices, some academics have advocated for the random walk hypothesis, which states that stock prices move randomly and thus cannot be predicted at all [2]. Others stand by the efficient-market hypothesis [3], that the price of a stock reflects all information that is known about it at the time, implicating the impossibility of beating the market through any means. There is no consensus of the validity of these theories at the time of writing this.

1.1.3 Machine and deep learning

As previously mentioned, some academics and financial professionals have attempted to predict the future development of stock prices using quantitative methods. One of the most novel methods is machine learning. In short, machine learning can be described as the study of algorithms that improve at a task solely through experience or utilizing data and do not need to be hardcoded by humans.

Deep learning is a subset of machine learning where methods based on deep artificial neural networks are used. Deep learning shows great promise, and the universal approximation theorem states that a deep neural network can approximate any continuous function with an acceptable

accuracy [4]. Deep learning methods can be used to build a model that recognizes whether an image portrays a cat or a dog, converts speech to text, or even translates it to another language. More importantly to this thesis, they can be used to predict the next N samples in a time series.

1.1.2 Deep Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational frameworks that draw inspiration from the animal brain. They are layered networks of nodes, where each node is called a neuron. Deep Neural Networks (DNNs) are neural networks with more than two layers.

1.1.3 Recurrent Neural Networks and LSTMs

Recurrent Neural Networks (RNNs) are a type of DNNs that make use of a hidden state, which acts as an internal memory and makes RNNs well-suited for time series prediction. However, the vanilla versions of RNNs tend to, in practice, have a short memory, while it in some instances would be beneficial for them to learn longer-term patterns in a series. For this reason, there is a certain architecture of RNNs called Long short-term memory (LSTM) which through a mechanism called gates enable the network to remember things in both the short and long term. This property is what makes LSTMs perform surprisingly well across many different domains where the data is sequential [5].

Zhu [6] notes that RNNs are very suitable for stock predictions, as they effectively process time-series data and considers its context during the training, and Ma [7] compared the predictive ability for stocks of the commonly used ARIMA model, vanilla ANNs and LSTMs, with results that indicated LSTMs performs the best of them all.

1.2 Related work

There have been quite a few studies examining the application of RNNs in stock prediction. Many have focused on predicting the real price of a stock, which is quite a hard task, and taken more creative approaches such as performing sentiment analysis on news articles to predict price behavior, whereas others have focused on predicting returns, something more in line with our work.

Zhu proposed a prediction method for stock prices using Recurrent Neural Networks [6]. The experiment made use of the Tensorflow machine learning library to build and train a two-layer RNN on Apple (AAPL) stock trading data and demonstrated that it could predict the stock price in the short term with high accuracy, albeit as the time steps increased, the mean average error of the predictions increased. However, they only did so with the Apple stock posing the question of its external validity.

Nelson et al. [8] built binary classification models using LSTM RNNs that, instead of predicting the future price of a stock, predicted whether a stock would be higher or lower than the current price 15 minutes in the future. These were trained on the trading data of stocks on the Brazilian stock exchange, along with a set of technical indicators derived from the trading data. On the five stocks for which the authors published the results, the lowest and highest accuracy of the models was 53.0 and 55.9 percent respectively. However, this study did not look into predicting the daily returns of a stock.

Althelaya et al. [9] evaluated several deep learning architectures in the context of using them to predict the price of the Standard & Poor 500 Index at the end of a day. They benchmarked models based on shallow neural networks, LSTMs, and other forms of RNNs such as the Bidirectional LSTM (BLSTM) and Stacked LSTM (SLSTM). BLSTM differs from normal LSTMs in that it at any point in time during the training preserves information from both the past and the future, allowing it to better understand the context. Stacked LSTMs usually allow for models that capture more complex patterns in a given time series. Their results indicated that LSTMs outperform shallow neural networks in long and short-term prediction, SLSTMs outperform them both, and BLSTMs perform the best out of all selected models.

There are also more creative approaches that have been taken with using LSTMs to predict stock price behavior. Verma et al. [10] performed Indian stock market predictions by feeding news articles and internet content to an LSTM model.

1.3 Knowledge contribution

Researchers have attempted to predict stock markets for decades, and their inherent complexity and chaoticness make it perhaps one of the most difficult tasks computer scientists and deep learning researchers will face.

The knowledge contribution of our thesis is to examine the predictive ability of models trained by LSTMs when it comes to stock returns, and whether they are any better than randomly predicting whether the return will be positive or negative, or than simply stating that all days will be negative, or positive, for that matter. While a similar study has been done in the past by Nelson et al. [8] we intend to train our models at predicting the returns on a longer timeframe and test their accuracy. Moreover, we intend to measure the models' accuracy on 60 stocks traded in both the United States (in the Nasdaq-100 index) and Sweden (in the OMX30 index), in the hopes of building a greater empirical foundation.

1.4 Limitations

This study will only measure the accuracy of standard LSTM models for predicting the daily returns of OMX30 and Nasdaq-100 component stocks, meaning that we will not examine their effectiveness at predicting the price on a longer or shorter time frequency. Moreover, the models to be measured will only be trained on daily stock trading data, meaning that the datasets' intervals will not be more or less frequent, and it will not contain additional data such as news articles or online forum posts. We will not evaluate more advanced variants of LSTMs such as SLSTMs and BLSTMs, to limit the scope of this thesis.

1.5 Target group

The target group is researchers in computer science and deep learning.

2. Theoretical background

To give a better understanding of the methods used in this thesis, the following chapter will elaborate on the theoretical aspects of some key concepts.

2.1 Deep neural networks

Since we will make use of an LSTM neural network to train models at predicting stock returns, this section will give a fundamental explanation of how the LSTM's superclass, deep neural networks, work. Previously, deep neural networks were mentioned, how they are layered networks where each node is called a neuron, and that they have more than two layers of neurons. Let us begin with what the neuron does.

A neuron is a function that takes one or more inputs and produces an output that then may be used as input to another neuron, or observed. For each input, x_k, the neuron multiplies it with a *weight*, w_k. This weight represents the strength of the connection between the neuron in question and the neuron that emitted the input. After having multiplied all inputs with the weights, they get summed up as such [11]:

$$Input\text{: } \vec{x} = (x_1, x_2, \ldots, x_n)$$
$$Weights\text{: } \vec{w} = (w_1, w_2, \ldots, w_n)$$
$$Sum\text{: } s = \Sigma\,(x_1 w_1, x_2 w_2, \ldots, x_n w_n)$$

A neuron also holds a *bias*, a constant value it adds to the sum of the weighted inputs. After having weighted the inputs and added a bias to their sum, the result z can then be input to an activation function that defines the final output of the neuron, y:

$$Input\text{: } \vec{x} = (x_1, x_2, \ldots, x_n)$$
$$Weights\text{: } \vec{w} = (w_1, w_2, \ldots, w_n)$$
$$Sum\text{: } s = \Sigma\,(x_1 w_1, x_2 w_2, \ldots, x_n w_n)$$
$$Bias\text{: } B$$

$$Activation\ function\text{: } \phi$$
$$\hat{y} = \phi(s + B)$$

The activation function used varies, albeit they typically are nonlinear. Some examples are sigmoid functions, which most often return values between 0 and 1, and the hyperbolic tangent (tanh) function, which outputs values between -1 and 1.

As mentioned, DNNs have more than two layers of nodes. The first is the input layer, which consists of neurons that take the initial input to the DNN. The last is the output layer, which produces the final output of the DNN that is to be observed. All other layers intermediate between the input and output layers are called hidden layers, and it is where most of the computation is done. The input layer receives the initial data to the neural network, passes the

computed results to a hidden layer which either passes their output to another hidden layer or the output layer, which then computes and emits the output data of the network.

2.1.1 Loss

During the training of our models, we will make use of a concept called *loss*, also called error. The loss is a score that tells us how good our model is at performing its task [11]. The goal is to minimize the loss in a model, as a lower loss means that the model is better at the task. Several loss functions can be used, and which one to choose largely depends on the task one wants to use the neural network for, but more specifically for this thesis, the loss function widely regarded as most appropriate for binary classification tasks is the binary cross entropy[1].

2.1.2 Optimization and training

Optimizers are a class of algorithms that change different attributes of a neural network, such as the weights and biases in neurons in an attempt to minimize the loss score, something we will use as well during the training of our models [11]. The process of optimizing the neuron weights and biases is often called training.

A way to train a neural network is through backpropagation and gradient descent, by calculating the partial derivatives of the loss function with respect to each parameter and iteratively taking small steps in their opposite direction by updating the network parameters such as the weights and biases [12].

2.2 Recurrent Neural Networks

We went over the fundamentals behind how DNNs work in 2.1. Although an LSTM is technically an instance of an DNN, it is more specifically something called a Recurrent Neural Network (RNN) and for this reason, it has some differing qualities that should, in theory, improve the predictive ability of our models.

The main difference between RNNs and their superclass, DNNs, is that RNNs are well-suited for sequential data and are temporal in nature, due to the fact that they hold a hidden state, effectively a memory, and the output of an RNN is not only a function of its input, but its hidden state [5]. This memory allows the neural network to be used repeatedly over time and remember the past. For this reason, RNNs are more like a directed list of “snapshots" of a neural network over time, and each of these “snapshots” can be referred to as an RNN cell.

Assume we have a sequence of observed time series, $(\vec{o_1}, \vec{o_2}, \ldots, \vec{o_n})$. To predict $\vec{o}_{i+1}$, with an RNN cell we assume it is a function of $\vec{o}_i$ and the previous RNN cell’s hidden state $\vec{h}_i$:

$$\vec{o}_{i+1}, \vec{h}_{i+1} = RNN\left(\vec{o}_i, \vec{h}_i\right)$$

[1] See recommendations in the most common machine learning libraries, for instance: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

Here the RNN function is assumed to calculate an updated hidden state and use it to predict the new output and then return both. The hidden state is usually updated through a simple hyperbolic tangent function.

$$\vec{h}_{i+1} = \tanh\left(\vec{W}_o\vec{o}_i + \vec{W}_h\vec{h}_i + B\right)$$
$$\vec{o}_{i+1} = \sigma\left(\vec{W}_{yh}\vec{h}_i + B_y\right)$$

With time, however, due to the vanishing gradient problem, caused by an algorithm used during the training, the RNNs recollection of the earlier steps vanes in favor of those more recent. While RNNs do have a memory, the standard variants of them have a short one.

2.3 LSTMs

Now we have arrived at the final destination of the theoretical background, the explanation of the very thing we will make use of in our project. As mentioned in 2.3, there is a problem of the vanishing gradient when it comes to RNNs. There are multiple alternate architectures of RNNs that aim to mitigate it. The oldest and most prominent is the Long short-term memory (LSTM) architecture. The architecture of an LSTM unit consists of four components: A cell, a forget gate, an input gate, and an output gate. Using these components, LSTM networks can learn long-term dependencies [5], which is very interesting for applications such as stock prediction.

The LSTM cell state is an addition to the hidden state of RNNs, and it has the same dimensionality as the hidden state vector. The hidden state of an LSTM can be thought of as its short-term memory, and the cell state as the long-term memory. The cell state can get passed on with its information largely unchanged, however, the LSTM does have the capability to change it using gates. Gates are the mechanisms that regulate what information in the cell state should be forgotten or updated, and are essentially what differs an LSTM from standard RNNs. Whereas a standard RNN would update its hidden state for new cells using a simple hyperbolic tangent function, the process is a bit involved with LSTMs. In a slightly reductionist explanation, the LSTM uses gates to calculate what information to forget and update in the LSTM cell state, and then uses the new cell state information to output an updated hidden state.

In an LSTM, the first gate is the forget gate. It is a neuron layer with a sigmoid activation function that takes in the previous hidden state and the input to the LSTM, and outputs a vector of the same dimensionality as the cell state vector, consisting of numbers between 0 and 1.

$Output\ vector\ from\ forget\ gate$: $\vec{f}_t$
$Previous\ hidden\ state$: $\vec{h}_{t-1}$
$Current\ input$: $\vec{x}_t$
$Weight\ vector\ for\ forget\ gate$: $\vec{W}_f$
$Bias\ for\ forget\ gate$: $\vec{B}_f$
$$\vec{f}_t = \sigma\left(\vec{W}_f \cdot \left[\vec{h_{t-1}}, \vec{x}_t\right] + \vec{B}_f\right)$$

After calculating the forget gate vector, the input gate comes into question. The input gate decides what new information to update a cell state with. It consists of two neuron layers; a sigmoid layer and a tanh layer. The sigmoid layer decides *what* information will be updated, and the tanh layer creates a temporary cell state vector with candidates that then can be used to update the actual cell state.

Input gate vector:$\vec{i}_t$
Candidate cell state values:$\tilde{C}_t$

The input gate vector and the candidate cell state is calculated as such:

$$\vec{i}_t = \sigma\left(\vec{W}_i \cdot \left[\vec{h_{t-1}}, \vec{x}_t\right] + \vec{B}_i\right)$$

$$\tilde{C}_t = \tanh\left(\vec{W}_C \cdot \left[\vec{h_{t-1}}, \vec{x}_t\right] + \vec{B_C}\right)$$

Now that we have the forget gate output vector, the input gate output vector, and the candidate state, it is time to update the old cell state $\vec{C}_{t-1}$. First, we multiply $\vec{C}_{t-1}$ with $\vec{f}_t$. That means that if a number in the forget gate output vector is 0, the respective number in the old cell state becomes 0 and is forgotten, and if the number is 1, the cell state's number is left completely intact. Then, we multiply the input gate output vector and the candidate cell state vector and add it to the previous product.
The new cell state, $\vec{C}_t$:

$$\vec{C}_t = \vec{f}_t * \vec{C}_{t-1} + \vec{i}_t * \tilde{C}_t$$

Finally, there is the output gate through which we get the new hidden state h_t. The output from this state is a filtered version of the new cell state. Like the input gate, it consists of a sigmoid and a tanh layer. The sigmoid layer takes the previous hidden state and the input to decide which parts of the cell state will remain in the output. Then the cell state gets put through the tanh layer. Next, we multiply the product of these layers to get the final output.

$$\vec{o_t} = \sigma\left(\vec{W}_o\left[\vec{h_{t-1}}, \vec{x}_t\right] + \vec{B}_o\right)$$

$$\vec{h_t} = \vec{o_t} * \tanh\left(\vec{C_t}\right)$$

3 Method

The method of this study was as such: We developed an LSTM neural network and trained it at classifying whether the daily return of stock would be positive or negative the following day, using the open-source deep learning library Pytorch[2] and historical trading data sourced from Yahoo Finance[3]. For each selected stock in the OMX30 and Nasdaq-100, we trained a model at predicting the next day's return. As the model is performing binary classification, a future positive or negative return, we then measured the accuracy, precision, and recall of the predictions to assess the quality of the models the LSTM neural network produced.

We decided to measure a model doing binary classification, as opposed to having the model predict the real price of a stock. The reason for performing binary prediction is two-fold. Firstly, if one were to assume a stock trader one day would use the model when deciding to buy a stock, they would first and foremost be concerned with what direction it is predicted to move and not the exact price. Secondly, it greatly lowers the solution space as there would only be two possible solutions, instead of having the model predict a real number.

To gain more insight into the dataset and better understanding of the predictive ability of the models, we also measured the distribution of the two classes in the dataset.

3.1 Collection and preprocessing of dataset

We collected up to 10 years of historical trading data for all 30 stocks currently in the OMX30, and 30 of the largest components of the Nasdaq-100, from Yahoo Finance. The data for each stock contains:

- The lowest and highest price of each week (low and high), usually tracked by stock traders to infer a sort of volatility of the stock.
- The last recorded price each week (close), which was used to calculate the weekly returns over time.
- The total number of shares that were traded during the week (volume). In finance, there are a number of different theories on the relationship between volume and the future price of a stock, however, Karpoff found there to be a positive relationship between volume and the magnitude of change of a stock's price [13]. The reason it was included was because of the possibility of the LSTM finding some sort of relationship between the volume and future returns.

To collect this data, we made use of the Python package Yfinance[4], which acts as a wrapper to Yahoo's Finance API and returns the data in a Pandas dataframe, which looks quite a lot like a regular table as can be seen in figure 3.1:

---

[2] Pytorch is a reliable deep learning library used by institutions such as Salesforce and Stanford University. Read more about it: https://pytorch.org

[3] Yahoo Finance is a website offering news, quotes and historical data on stocks. See https://finance.yahoo.com

[4] See https://github.com/ranaroussi/yfinance

Date	Open	High	Low	Close	Adj Close	Volume
2015-01-02	44.574001	44.650002	42.652000	43.862000	43.862000	23822000
2015-01-05	42.910000	43.299999	41.431999	42.018002	42.018002	26842500
2015-01-06	42.012001	42.840000	40.841999	42.256001	42.256001	31309500
2015-01-07	42.669998	42.956001	41.956001	42.189999	42.189999	14842000
2015-01-08	42.562000	42.759998	42.001999	42.124001	42.124001	17212500

Figure 3.1 - The formatting of the data collected from Yahoo Finance

To get the relative return for each day, we subtracted the previous close from the current close, and divided it by the current previous close:

$$r_t = \frac{c_t - c_{t-1}}{c_{t-1}}$$

Where r_t denotes the return for day t, and c denotes the closing price for day t.

Then we independently standardized each feature of the dataset by subtracting the mean of the feature dataset from each value and scaling it to unit variance, leading the feature's dataset to have a mean value of 0 and standard deviation of 1. The standardized value, z, of sample x, where u and s is the mean and standard deviation of the feature's samples respectively: $z = (x - u) / s$

We split each stock's data set into two parts; a training set, used during training of the LSTM model, and a test set used for testing it later on. The training set covered 70 percent of the original data set, and the test set 30 percent.

3.2 Definition of classes

A class can be described as a category of observations that share some common characteristics. We decided to use two classes, the positive return class and the negative return class, defined as such:

$$y = \begin{cases} 1 & if\ r_t \geq 0 \\ 0 & if\ r_t < 0 \end{cases}$$

Where r_t is the current day's return.

3.3 Identifying distribution of classes

For each stock, we also measured the ratio of the positive return class:

$$Positive\ class\ ratio = \frac{PR}{PR + NR}$$

Where PR is the number of occurrences of the positive return class (1), and NR is the number of occurrences of negative return class (0). Due to the fact there are only two classes, the ratio of negative returns can be inferred from it:

$$Negative\ class\ ratio\ =\ 1\ -\ Positive\ class\ ratio$$

3.4 Training of models

To develop and train the LSTM network, we made use of Python 3.7.10 and Google Colab, a web application that allows us to write Python Notebooks in the browser. The hope was that performing the study in Google Colab will allow others to reproduce our results in the web application as well, while abstracting away the need to set up the same hardware, system, or Python configuration locally. Moreover, the open-source deep-learning library Pytorch was used, as it provides a Python API facilitating the creation and training of neural networks and providing users with implementations of loss functions and optimization algorithms.

The network we developed had 5 inputs, each one respectively being a vector of the 30 most recent days' open, close, low, high, and volume. The network also had 2 hidden layers, a hidden state with 50 features, and one output since the model's task is to perform binary classification; positive or negative return, which is a single scalar value between zero and one. During the training, the learning rate was set to 1e-2. This configuration was the result of weeks of tinkering and attempting to achieve an optimal accuracy. We then proceeded to instantiate a loss function called BCEWithLogitsLoss from the Pytorch library, which consists of a sigmoid layer and a Binary Cross Entropy function[5]. Moreover, we used an implementation of the Adam optimization algorithm[6] during the training of the model. The network was then trained using full gradient descent for 300 epochs on the stock's training dataset.

3.5 Assessing predictions with accuracy, precision, and recall

After having trained a model, it was then tested on the stock's test dataset. During the test, we evaluated the model's binary classification quality by measuring its accuracy, the fraction of all correct predictions:

Accuracy = (TP + TN) / (FP + FN + TP + TN)
Where TP is true positives, TN is true negatives, FP is false positives and FN is false negatives.

Additionally, we also measured the precision and recall. The reason for this is that measuring accuracy alone in imbalanced datasets may be misleading [14], and before performing this study we had no prior knowledge of the distribution of classes in the dataset. Consider a stock where 90% of the weeks have a positive return and only 10% have a negative return. A classifier that

[5] Binary Cross Entropy is the de facto standard loss function for binary classification tasks. See https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html

[6] The Adam optimization algorithm is a quite novel method which is computationally efficient and appropriate for non-stationary problems, which dealing with stock prices and returns normally is. See the original paper: https://arxiv.org/abs/1412.6980

simply classifies all weeks as a positive return would have 90% accuracy even though the model will not generalize well to other stocks, or be of very much use at all. Precision, on the other hand, is the proportion of true positives to all positives, and recall is the proportion of true positives to the sum of true positives and false negatives. More formally, they are defined as:

Precision =TP / (TP+FP)
Recall = TP / (TP+FN)

These metrics can give more insight into how well a classification model performs.

3.6 Reliability and Validity

3.6.1 Validity

We measure the accuracy of our binary classification model because it quickly gives an overview of the quality of the model, and the fraction of correct predictions out of all predictions. The accuracy may be useful to assess whether the model is better than random classification. However, it may not tell the full story. That is why we measure precision and recall as well. The precision tells us the probability of the model making a correct positive classification. In other words, if the model is naively classifying everything as a positive even though that may not always be the case, this metric shows us that and allows us to assess whether it actually provides any useful predictions. Likewise, the recall tells us the sensitivity of the model; how probable it is to classify a positive as, in fact, a positive. Together these metrics give us a view of how good the LSTM models are at predicting a positive or negative return of stocks in the OMX30, the probability that it predicts a positive return and is correct, and the probability that an actual positive return was predicted to be positive.

Moreover, we also measure the ratio of the positive class in the dataset, which helps give a more intuitive understanding of the accuracy metric. For example, if a model has a 99 percent accuracy it may appear to be good at first glance. But if 99 percent of the samples are of the positive class, and the model blindly classifies all samples as positive, it is not a very good model.

3.6.2 Reliability

To create and train our LSTM models we made use of the software library Pytorch. They do not guarantee reproducible results across different releases or platforms. However, we have attempted to minimize the sources of nondeterministic behavior by manually seeding our random number generator with the number one [15].

3.6.3 Power

To assess whether the prediction ability of LSTMs is generalizable to stocks as a whole, as opposed to only working on certain cherry-picked stocks such as Apple or a handful of Brazilian stocks, we measured the LSTM’s accuracy, precision and recall on a high number of stocks, more precisely 60 stocks traded on both the Swedish and American stock exchange with up to 10 years of historical data, in order to build a large empirical foundation.

3.6.3 Survivorship bias

Due to the fact that we measured the accuracy of LSTM models on stocks in the OMX30 and Nasdaq-100, our work is susceptible to survivorship bias, as we did not measure their accuracy on stocks that may have been part of the OMX30 in the past, but went bankrupt or performed bad enough to lose their position in the index.

3.6.4 Little data

As the intention of this study was to study the predictive ability of stocks currently in the OMX30 and Nasdaq-100, some of the stocks the models were trained on had very little historical data. The majority of stocks in the index have been publicly traded for 10 years or more, however, some were listed more recently and thus only had a few years of historical trading data, which may negatively or positively affect the models' predictive performance.

3.7 Ethical Considerations

As we do not in any way collect personal data we do not need to adopt measures to ensure privacy and integrity. We do not assess that our method puts someone at risk of harm, as we only collect data that is already publicly available. The data we have collected is publicly available for download on the internet and no consent has been required.

However, there is the possibility that some individuals or organizations may make incorrect or flawed conclusions from this work, apply those conclusions to their financial activities and in turn lose money. It needs to be stated that none of this work should be taken as financial advice.

4 Results

In table 4.1 the individual accuracy, precision, recall, ratio of positive returns against all positive and negative returns, and length of training data for the models for each stock in the OMX30 is presented.

Stock ticker	Accuracy	Precision	Recall	Ratio of positive returns	Length
ABB	0.494	0.505	0.476	0.511	2761
ALFA.ST	0.49	0.501	0.503	0.511	2761
ALIV-SDB.ST	0.502	0.496	0.505	0.494	2761
ASSA-B.ST	0.481	0.516	0.071	0.521	2761
ATCO-A.ST	0.483	0.595	0.155	0.544	2761
AZN.ST	0.494	0.496	0.44	0.503	2761
BOL.ST	0.494	0.496	0.44	0.503	2761
ELUX-B.ST	0.473	0.48	0.457	0.508	2761
ERIC-B.ST	0.501	0.496	0.515	0.495	2761
ESSITY-B.ST	0.491	0.495	0.393	0.506	889
EVO.ST	0.454	0.529	0.036	0.548	1452
GETI-B.ST	0.515	0.518	0.532	0.504	2761
HEXA-B.ST	0.48	0.493	0.078	0.519	2761
HM-B.ST	0.515	0.514	0.861	0.51	2761
INVE-B.ST	0.523	0.541	0.572	0.522	2761
KINV-B.ST	0.514	0.554	0.416	0.529	2761
NDA-SE.ST	0.508	0.509	0.536	0.501	2761
SAND.ST	0.506	0.554	0.113	0.505	2761
SCA-B.ST	0.484	0.523	0.268	0.529	2761
SEB-A.ST	0.455	0.427	0.358	0.486	2761
SECU-B.ST	0.494	0.493	0.383	0.501	2761
SHB-A.ST	0.512	0.485	0.567	0.471	2761
SKA-B.ST	0.532	0.539	0.578	0.511	2761
SKF-B.ST	0.484	0.477	0.423	0.496	2761
SWED-A.ST	0.511	0.508	0.636	0.499	2761

SWMA.ST	0.498	0.5	0.081	0.502	2761
TEL2-B.ST	0.524	0.519	0.533	0.495	2761
TELIA.ST	0.511	0.512	0.558	0.503	2761
VOLV-B.ST	0.52	0.525	0.447	0.502	2761

Table 4.1 - Results for stocks in the OMX30

The mean metrics across all stocks in the OMX30 are displayed in table 4.2.

Mean accuracy	0.498
Standard deviation, accuracy	0.019
Mean precision	0.510
Standard deviation, precision	0.029
Mean recall	0.407
Standard deviation, recall	0.192
Mean ratio of positive returns	0.508
Standard deviation, the ratio of positive returns	0.016

Table 4.2 - Statistics of the results for stocks in the OMX30

In table 4.3 the results for stocks in the Nasdaq-100 are shown.

Stock ticker	**Accuracy**	**Precision**	**Recall**	**Ratio of positive returns**	**Length**
AAPL	0.471	0.522	0.34	0.544	2768
ADBE	0.557	0.564	0.93	0.562	2768
AMAT	0.499	0.526	0.606	0.532	2768
AMD	0.502	0.539	0.328	0.523	2768
AMGN	0.512	0.535	0.383	0.514	2768
AMZN	0.46	0.514	0.368	0.551	2768
AVGO	0.502	0.53	0.617	0.535	2768
BKNG	0.535	0.548	0.814	0.542	2768

CHTR	0.552	0.563	0.616	0.52	2767
CMCSA	0.493	0.511	0.497	0.518	2767
COST	0.496	0.543	0.612	0.558	2768
CSCO	0.466	0.525	0.153	0.542	2768
FB	0.506	0.539	0.433	0.527	2769
GOOG	0.548	0.554	0.839	0.541	2768
GOOGL	0.529	0.54	0.792	0.534	2768
INTC	0.502	0.511	0.892	0.517	2768
INTU	0.474	0.55	0.224	0.549	2768
ISRG	0.428	0.5	0.002	0.572	2768
LRCX	0.507	0.537	0.56	0.534	2768
MDLZ	0.474	0.472	0.254	0.511	2768
MSFT	0.456	0.588	0.138	0.568	2768
MU	0.495	0.515	0.67	0.525	2768
NFLX	0.49	0.548	0.087	0.517	2768
NVDA	0.471	0.569	0.119	0.544	2768
PEP	0.502	0.53	0.642	0.538	2768
PYPL	0.488	0.54	0.538	0.556	1384
QCOM	0.517	0.54	0.53	0.524	27618
SBUX	0.498	0.564	0.276	0.535	2768
TSLA	0.52	0.537	0.531	0.518	2646

Table 4.3 - Results for stocks in the Nasdaq-100

The mean metrics across all stocks in the Nasdaq-100 are displayed in table 4.4.

Mean accuracy	0.497
Standard deviation, accuracy	0.028
Mean precision	0.538
Standard deviation, precision	0.025
Mean recall	0.465
Standard deviation, recall	0.249
Mean ratio of positive returns	0.537

Standard deviation, the ratio of positive returns	0.016

Table 4.4 - Statistics of the results for stocks in the Nasdaq-100

5 Analysis

5.1 Distribution of dataset

As can be seen in table 4.2, the distribution of positive and negative returns across all stocks in the OMX30 was quite symmetrical. On average, 50.8 percent of the days had positive returns, and 49.2 percent of the days had negative returns. The dispersion was quite low. The ratio of positive returns merely had a standard deviation of 1.6 percentages, and the highest and lowest ratio for a stock was 54.8 and 47.1 percent respectively. The ratio of positive returns in the Nasdaq-100, displayed in table 4.4, was slightly higher, at 53.7 percent with the same standard deviation as for the OMX30.

The symmetry of the distribution of the classes means two things: A model blindly classifying 100 percent of the days as one class or the other would be correct about 50 percent of the time. Additionally, random classification would also be correct about 50 percent of the time. Because of this, one could conclude that our models only have a good predictive ability if they are correct more than 50 percent of the time, as they otherwise would be no better than very trivial classification methods.

5.2 Accuracy

Table 4.2 shows that the mean accuracy of the models for stocks in the OMX30 was 49.8 percent. This is quite unimpressive when keeping in mind that, as mentioned in 6.1, the distribution of the two classes is nearly 50 percent and classifications by random chance would achieve similar accuracy. The highest observed accuracy in table 5.1 was with the model trained on the Skanska AB stock (SKA-B.ST), which was 53.2 percent accurate. The lowest observed accuracy was with the model trained on the Evolution AB stock (EVO.ST), which was 45.4 percent accurate. Note, however, that the Evolution AB stock got listed on the OMX30 later than the majority of the stocks, and had about 47 percent less training data than the majority. The importance of that may be negligible, on the other hand, as another stock, Essity AB (ESSITY-B) had 68 percent less training data and the model still had an accuracy of 49.1 percent.

In table 4.4 it is shown that the mean accuracy for stocks in the Nasdaq-100 was 49.7 percent, one promille lower than the mean accuracy for those in the OMX30. Moreover, the standard deviation of the accuracy was higher, at 2.8 percent. The highest observed accuracy in table 5.3

was for the Adobe Inc. stock (ADBE), at 55.7 percent, and the lowest was for Microsoft Corporation (MSFT) at a 45.6 percent accuracy.

5.2 Recall

The mean recall for models trained on stocks in the OMX30 shown in table 4.2 was 40.7 percent, meaning that for days that turned out to have a positive return, the models failed to predict them as positive 59.3 percent of the time. However, the standard deviation of the recall was 19.2 percentages, and the highest and lowest observed recall was 86.1 and 3.6 percent, respectively. For the stocks in Nasdaq-100, as is shown in table 4.4, the recall was slightly higher at 46.5 percent. Overall, observed recalls are quite low and indicate there are many false negatives, however, the significance of that depends on for what purposes one would use the models.

5.3 Precision

The mean precision for stocks in the OMX30, shown in table 4.2, was 51 percent, meaning that 51 percent of predictions that the next day would have a positive return were correct and 49 percent were incorrect. The standard deviation was lower than with the recall, however, at only 2.9 percentages. The highest and lowest observed precision was, respectively, 59.5 and 47.7 percent. As seen in table 4.4, the mean precision for Nasdaq-100 stocks was 53.8 percent and only a 2.8 percentage standard deviation. As is the case with recall, the significance of this metric largely depends on for what purposes the model is intended to be used.

7 Discussion

7.1 Accuracy

We found that the predictive ability of the trained models barely outperforms that of random prediction or blindly predicting one single class all the time. The mean accuracy for stocks in the OMX30 was 49.8 percent and for stocks in the Nasdaq-100 49.7 percent. This is slightly less accurate than the results presented by Nelson et al. [8], who also trained a binary classification model using LSTM RNNs. The highest and lowest accuracy presented there was 53.0 and 55.9 percent respectively. Note, however, that their models predicted the returns 15 minutes in the future, as opposed to the following day. It may be the case that binary prediction on a shorter time frame is more prone to be accurate, something which is supported by the findings of Zhu [6]. Their models were also trained on a wide array of technical indicators derived from the trading data, which could possibly improve the models. On the other hand, Nelson et al. only published the results for five stocks, whereas we trained and tested our models on 60 stocks both traded in the United States and Sweden, perhaps meaning their published results are more prone to selection bias and less generalizable.

7.2 Generalizability

Firstly, there is the question of whether our results are generalizable to stocks not in the OMX30 or Nasdaq-100 indices. The OMX30 only includes the 30 most traded stocks on the Nasdaq Stockholm stock exchange, and the Nasdaq-100 only includes 100 of the most traded stocks at the American Nasdaq exchange. If these stocks are traded more heavily than other stocks, it could mean their prices behave differently from less traded stocks on the exchanges.

Due to the fact that we trained and tested our models on stocks from two different exchanges, one American and one Swedish, with very similar results, we argue that the presented results may be somewhat generalizable to highly traded stocks across North America and Europe.

8 Conclusions and Future Work

The purpose of this thesis was to examine the ability of models trained by an LSTM neural network to predict the returns of stocks in the OMX30 and Nasdaq-100 indices. We trained models for 60 stocks in total and examined their accuracy, precision and recall. The accuracy of the models did not differ much from that one would expect from randomly predicting a class, or predicting *all* days to be of one single class. Thus, we conclude that the predictive ability of the models trained by the LSTM network is poor and that further improvement is needed to achieve better accuracy.

Because the returns of a stock tend to be more stationary than the stock prices themselves, it should in theory be easier for a neural network to learn how to predict returns than prices. Moreover, the task of binary classification has a much smaller solution space than if a model were to predict the real price of a stock. For these reasons, we argue that performing binary prediction on stock returns should be a much easier task than predicting real prices, and since our models show bad predictive ability in the first task, it is questionable whether the second task would fare any better. We suggest that further improvements in predicting returns is necessary before moving on to harder tasks such as predicting prices, which may serve as guidance to other researchers in the area.

As for future work, one needs to be aware of the dire reality that most people who attempt to trade stocks fail. In 1999, the National American Securities Administrators Association reported that 70 percent of professional traders lost money, and more recently, it has been shown that the median 12-month return for users of the social trading site eToro was -36.3 percent. When dealing with machine learning tasks, one is usually considered successful if a model manages to perform nearly as well as a human, but what if most humans are terrible at predicting future returns? It may be just so that a worthwhile machine learning model not only needs to measure up to the performance of the average human, but to those exceptional few that are successful for nonrandom reasons. To create exceptionally performing models, why not get inspired by the highly skilled human traders? After all, neural networks were conceived through the mimicry of the animal brain, so perhaps there are developments that can be made by imitating and automating the process performed by the elite in trading. For instance, many profitable traders make use of technical analysis, a method where one is looking for specific patterns in a stock's chart that are associated with moves of high magnitude in either direction. This is quite different, as it transforms the problem from at all times attempting to predict the next move of the market, to attempting to see patterns signifying an upcoming move of great magnitude. Could a deep learning model be trained to recognize patterns preceding high magnitude moves, and could it be accurate enough to be applicable in real life?

References

1. J. Cao, Z. Li and J. Li, "Financial time series forecasting model based on CEEMDAN and LSTM", *Physica A: Statistical Mechanics and its Applications*, vol. 519, pp. 127-139, 2019. Available: https://www.sciencedirect.com/science/article/pii/S0378437118314985. [Accessed 3 June 2021].
2. E. Fama, "Random Walks in Stock Market Prices", *Financial Analysts Journal*, vol. 51, no. 1, pp. 75-80, 1995. Available: 10.2469/faj.v51.n1.1861 [Accessed 3 June 2021].
3. B. Malkiel, "The Efficient Market Hypothesis and Its Critics", *Journal of Economic Perspectives*, vol. 17, no. 1, pp. 59-82, 2003. Available: 10.1257/089533003321164958 [Accessed 3 June 2021].
4. K. Hornik, "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, vol. 4, no. 2, pp. 251-257, 1991. Available: 10.1016/0893-6080(91)90009-t [Accessed 3 June 2021].
5. A. Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network", *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020. Available: 10.1016/j.physd.2019.132306 [Accessed 3 June 2021].
6. Y. Zhu, "Stock price prediction using the RNN model", *Journal of Physics: Conference Series*, vol. 1650, p. 032103, 2020. Available: 10.1088/1742-6596/1650/3/032103 [Accessed 3 June 2021].
7. Q. Ma, "Comparison of ARIMA, ANN and LSTM for Stock Price Prediction", *E3S Web of Conferences*, vol. 218, p. 01026, 2020. Available: 10.1051/e3sconf/202021801026 [Accessed 3 June 2021].
8. D. Nelson, A. Pereira and R. de Oliveira, "Stock market's price movement prediction with LSTM neural networks", *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017. Available: 10.1109/ijcnn.2017.7966019 [Accessed 3 June 2021].
9. K. Althelaya, E. El-Alfy and S. Mohammed, "Evaluation of bidirectional LSTM for short-and long-term stock market prediction", *2018 9th International Conference on Information and Communication Systems (ICICS)*, 2018. Available: 10.1109/iacs.2018.8355458 [Accessed 3 June 2021].
10. I. Verma, L. Dey and H. Meisheri, "Detecting, quantifying and accessing impact of news events on Indian stock indices", *Proceedings of the International Conference on Web Intelligence*, 2017. Available: 10.1145/3106426.3106482 [Accessed 3 June 2021].
11. S. Amidi and A. Amidi, "CS 229 - Deep Learning Cheatsheet", *Stanford.edu*, 2021. [Online]. Available: https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning. [Accessed: 03- Jun- 2021].
12. "CS231n Convolutional Neural Networks for Visual Recognition", *Cs231n.github.io*, 2021. [Online]. Available: https://cs231n.github.io/optimization-1/. [Accessed: 03- Jun- 2021].

13. J. Karpoff, "The Relation Between Price Changes and Trading Volume: A Survey", *The Journal of Financial and Quantitative Analysis*, vol. 22, no. 1, p. 109, 1987. Available: 10.2307/2330874 [Accessed 3 June 2021].
14. "Classification: Accuracy | Machine Learning Crash Course", *Google Developers*, 2021. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy. [Accessed: 03- Jun- 2021].
15. "Reproducibility — PyTorch 1.8.1 documentation", *Pytorch.org*, 2021. [Online]. Available: https://pytorch.org/docs/stable/notes/randomness.html. [Accessed: 03- Jun- 2021].

A Appendix 1

A1 Jupyter Notebook with code

To see the code used for creating, training and testing the LSTM neural network and its models, see this link (https://osf.io/jweph/). It can be imported into Google Colab and run on dedicated hardware for free.