Tampere University

Lauri Eneh

NEURAL NETWORKS IN HOUSEHOLD INVESTOR BEHAVIOR PREDICTION

Bachelor's Thesis
Faculty of Information Technology and Communication Sciences
Information Technology
May 2020

ABSTRACT

Lauri Eneh: Neural networks in household investor behavior prediction
Bachelor's Thesis
Tampere University
Information Technology
May 2020

Machine learning (ML) has been widely applied to various fields and areas, including the financial market. In this study, the behavior of active Finnish household investors is being predicted using three different neural network types: the multi-layer perceptron (MLP), long short-term memory (LSTM) and convolutional neural network (CNN). The target of the networks is to predict, whether an investor net-buys, net-sells or keeps their position on the next market day. In essence, the problem at hand is a three-class classification problem. The aim of this thesis is to find out which of the aforementioned networks receives the best F1 prediction scores and to get some insight to which inputs are of most importance. For the latter aim, attention mechanism is used. The data at hand, received from EuroClear, contains detailed information on all trades executed in Helsinki stock exchange. Focus is narrowed down to the 100 most active individual household traders of Nokia stock between the years 2006 and 2009.

First, a short review of the functionality of the stock exchange is made, and most central behavioral finance findings presented. The review shows, that investors do not act rationally in the markets. Next, the basic working principles of neural networks is explained, after which the used data and data preparation processes are brought out. From the data, 13 inputs are created, which are used to predict future household investor behavior. Finally, the experiment setting is explained and results presented.

Out of the networks experimented on in this study, the LSTM received the best prediction scores. However, none of the models falls significantly behind. Insight by the attention mechanism suggests, that the most important feature is the previous action of the investors. In the end, suggestions for receiving more satisfying results in future research are discussed. More sophisticated and complex models could give more satisfactory results.

Keywords: neural networks, long short-term memory, behavioral finance, empirical finance

TIIVISTELMÄ

Lauri Eneh: Neuroverkot yksityissijoittajan ostokäyttäytymisen ennustaisessa
Kandidaatin työ
Tampereen yliopisto
Tietotekniikka
Toukokuu 2020

Koneoppimisen tekniikoita on käytetty laajasti monilla eri tieteenaloilla, myös rahoitusdatan analysoinnissa. Tässä työssä aktiivisten yksityissijoittajien käyttäytymistä pyritään ennustamaan ja tulkitsemaan kolmella eri neuroverkkotyypillä: monikerroksisella perseptroniverkolla (engl. multilayer perceptron, MLP), pitkä-lyhytkestomuistilla (engl. long short-term memory) ja konvoluutioneuroverkolla (engl. convolutional neural network, CNN). Verkkojen ulostulona ennustetaan sitä, että myykö, ostaako vai pitääkö sijoittaja osakkeensa seuraavana päivänä. Pohjimmiltaan kyseessä on kolmiluokkainen luokitteluongelma. Työn tavoitteena on selvittää mikä mainituista verkoista saavuttaa parhaan F1 ennustustuloksen, sekä mitkä syötteet vaikuttavat eniten ennustettavuuteen. Syötteiden tärkeyden tutkimisessa käytetään huomiomekanismia (engl. attention mechanism). Käytössä oleva EuroClearin tuottama data sisältää yksityiskohtaista tietoa kaikista Helsingin pörssissä käydyistä kaupoista. Työssä keskitytään tarkastelemaan sataa aktiivisimmin Nokia Oyj:n osakkeella kauppaa käynyttä sijoittajaa vuosien 2006 ja 2009 välillä.

Työssä tehdään aluksi lyhyt katsaus rahoitusmarkkinoiden toimintaan ja esitellään behavioraalisen rahoituksen keskeisimmät löydökset. Katsauksessa havaitaan, että sijoittajat käyttäytyvät markkinoilla usein epärationaalisesti. Seuraavaksi tarkastellaan käytettyjen neuroverkkojen toimintaa ja periaatetta, jonka jälkeen tutkimuksessa käytetty data esitellään. Datasta prosessoidaan verkoille 13 syötettä, joiden avulla tulevaa ostokäyttäytymistä pyritään ennustamaan. Lopuksi selostetaan tutkimusmenetelmät ja esitellään tulokset.

Työssä käytetyistä verkoista LSTM yltää parempiin ennustustuloksiin kuin työssä esitelty vertailumalli. Huomiomekanismin mukaan tärkein käytökseen vaikuttava tekijä on sijoittajien oma kaupankäyntihistoria. Tutkimuksen perusteella ei voida todeta, että mallit eivät kykene ennustamaan sijoittajien käyttäytymistä. Tulokset antavat hyvin suuntaa mahdollisille jatkotutkimuksille. Jatkossa kiitettävien tuloksien saavuttamiseksi voidaan hyödyntää monimutkaisempia ja edistyneempiä malleja.

Avainsanat: neuroverkot, long short-term memory, käyttäytymistaloustiede, empiirinen rahoitus

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

Tampereella

CONTENTS

1 INTRODUCTION

We seem to be amidst a revolution, like one only seen a few times in history. Thousands of years ago, the agricultural revolution changed the way humans live. The Industrial Revolution, a few centuries ago, revolutionised the labor conditions, and just some decades ago, the technological revolution once again dramatically reshaped our everyday life. The most recent revolution is that of Artificial Intelligence (AI) and Machine Learning (ML). (Aoun 2017)

In recent years, many applications have been introduced that programmatically mimicking intellectual human behavior (referred to as AI). These include self-driving cars, disease diagnostics and financial market analysis, just to mention a few. Given that many of these applications make use of ML and have been found extremely utilitarian, it is futile to stress the potential of ML in handling data originating from complex environments. Here, ML refers to systems distinguished by their learning property, which allows the systems to self-adjust their parameters. This experimental study explores the possibility of ML methods, more specifically neural networks, being deployed to analyze complicated human decision making in the highly complex financial markets.

Behavioral finance studies human decision making under risk. What makes investors trade has been analyzed before, but mainly using regression (see Grinblatt and Keloharju 2001, Kaniel et al. 2012). The complicated investor decision-making process might be better analysed with methods capable of handling the non-linearity, such as ML. To my best knowledge, investor behavior has not been analyzed using ML techniques before.

It has been argued that ML leads to black-box methods lacking interpretability. In empirical finance, the objective is rather in understanding associations than in pure behavior prediction. However, recent developments in ML allows for more efficient interpretation of the models' "reasoning" without sacrificing on their accuracy. With attention mechanism, interpretability can be achieved in prediction environments. Interestingly, attention mechanisms first proposed for image recognition (Zhou et al. 2016, Xu et al. 2015) have been successfully applied to interpret financial data (Mäkinen et al. 2019, Tran et al. 2019).

In this study, neural networks are used to predict active household investor buy, hold and sell decisions. The research questions can be formulated as three separate questions:

1. Can future decisions of active household investors be predicted using neural networks based on their past activity and the market state?
2. Which network type produces best out of sample prediction results?

3. Which input variables influence active household investors' decision making the most?

Considering the last question, attention mechanism can be used to gain understanding about the temporal information relevant for predicting and explaining investor trading behavior. All questions are motivated by the fact, that gaining a deeper understanding about the behavior of investors would allow for the creation of robust models, which could, in turn, help us predict, examine and prevent, for example, financial crises in the future.

In essence, the problem at hand is a 3 class classification problem: the investor either net-sells, net-buys or holds their position on the next day. The output data consists of daily (only market days, excluding weekends and midweek holidays) observations about the change of the investors' position. The input data is comprised of both market state data and the activities of other active household investors. Market inputs will remain the same for all investors, but investor specific inputs vary. Various neural networks will be trained, including multi-layer perceptron (MLP), long short-term memory (LSTM) and convolutional neural networks (CNN). The goal is to develop investor agnostic models, which can predict future actions of investors never seen by the model before.

To analyze the behavior of active household investors, an interesting data set will be used, containing information on each transaction made in Helsinki stock exchange from 1996 to 2016. The focus will be on the most liquid stock, that of Nokia Oyj, and on a time period ranging from the beginning of 2006 to the end of 2009.

The rest of this study is structured as follows: Chapter 2 briefly discusses the basics of the stock market and continues to introduce the field of behavioral finance. Chapter 3, defines the concept of neural networks and focused on the different network types used in this study. Also, the function of attention mask is explained. Filtering of the data and the selection criteria for the inputs are introduced in chapter 4. Finally, the results will be presented in chapter 5 followed by the conclusions in chapter 6.

2 FINANCIAL MARKET AND INVESTOR BEHAVIOR

The financial markets allow the trade of assets among various agents. From a financial perspective, agents trade dividend-paying or interest yielding assets, such as stocks and bonds, as well as derivatives. In turn, from an economic perspective, agents trade time, risk and beliefs. Naturally, those agents are heterogeneous and have a different valuation of time, risk and beliefs. In classical economics, those beliefs are combined to form a market equilibrium, represented as the market prices for time, risk and beliefs. (Hens 2010) As stated before, this study will focus solely on stocks.

A stock is a type of equity investment asset representing ownership of a corporation. The owner of a stock is entitled to the earnings of the corporation paid as a dividend. Fundamentally, the value of a stock is determined by the net present value of the expected cash-flow discounted by the yield requirement. Hence, the fundamental price of a stock varies only with changes in the expected cash-flow or the yield requirement. (Knüpfer & Puttonen 2018) However, for example, Shiller (1981) shows that volatility cannot be explained solely by changes in the stock's fundamentals.

In practice, the market value of a stock is the price it is traded at publicly in an exchange and it can fluctuate greatly during trading hours (Knüpfer & Puttonen 2018). Today, many exchanges, including Nasdaq exchanges, function on limit order submissions. Limit orders are submitted to the system including the quantity and price with which the maker is willing to buy (bid) or sell (ask). A trade is executed, once a bid price is greater or equal to the ask price. (Mäkinen et al. 2019) Consequently, new bids and asks move the market price, and the price reflects the collective valuation of individual traders.

2.1 Market efficiency and arbitrage

Market efficiency theory put forward by Fama (1970), defines markets as efficient, meaning that all available information is already incorporated in the prices. The hypothesis is presented in three levels of efficiency (Fama 1970).

In the weak form, the hypothesis states, that the future stock prices can not be predicted using price history, as it already is reflected in the price at the moment. In semi-strong effective markets, no publicly available information can give an upper hand, since it is already evaluated by the market. Finally, in strong markets, even inside information is already priced in by the market. (Fama 1970). In spite of the hypothesis, there is no clear consensus among researchers, on how widely the theory applies.

The markets are also widely considered to be arbitrage-free. An arbitrage opportunity is a trading scenario, where an investor could receive positive returns without requiring any payments or increased risk-bearing. Financial researchers and practitioners agree, that arbitrage strategies ought to be considered non-existent, due to their rarity. Should an arbitrage possibility present itself, it would without delay be utilized driving the prices back to arbitrage freedom. (Hens 2010)

The markets being efficient and arbitrage-free does not imply that there can be no deviance from the "correct" price. They rather imply, that these deviations are random. Moreover, what allows efficient markets are the actors, constantly thriving to maximise expected gains, striping away potential arbitrage. This is the only way available information can be reflected correctly in the market prices. Not all actors in the market need to behave rationally in order for the market to be efficient. The hypotheses simply suggest, that the aggregate markets are rational. (Knüpfer & Puttonen 2018)

2.2 Behavioral finance

Many economic theories and models heavily build on the assumption that investors are rational and try to maximise their expected utility (Hens 2010). According to Shefrin and Statman (1985), it has been well known for a long time, that investors do not act according to Expected Utility Theory. As reported by Thaler (2005), they are motivated by greed and loss aversion, overreaction, overconfidence, heuristics and other biases, and do not act rationally at all times. Behavioral finance studies deviations of this perfectly rational behavior (Hens 2010).

Instead of Expected Utility Theory, a better approach to analysing investor decision making under risk is Prospect Theory, first presented by Kahneman and Tversky in 1979. They suggest, that investors react differently on the expected value of losses than on expected gains. Investors evaluate losses and gains with asymmetric emphasis. (Kahneman & Tversky 1979)

Another important finding in behavioral finance, build upon prospect theory, is the Disposition Effect, formulated by Shefrin and Statman (1985) and further demonstrated by other researchers (Dhar & Zhu 2006). Shefrin and Statman found, that investors tend to hold on to losing investment and to realise gains irrationally (Shefrin & Statman 1985).

Evidence has been presented, that the amplitude of an investors disposition effect is related to their level of sophistication (Dhar & Zhu 2006). Other studies have found, that investors in various categories tend to act differently (see e.g. Grinblatt and Keloharju 2001). However, much research has not been published concerning the degree to in which investing patterns are universal among different categories. On the contrary, in financial literature, rough categorization is often used, implicitly implying, that investors within a category are homogeneous (see e.g. Grinblatt and Keloharju 2001, Foucault et al. 2011). In this study homogeneity among a narrowly defined category, active Finnish household investors is also assumed.

As of today, empirical finance literature relies heavily on linear regression for understanding investor behavior (see Grinblatt and Keloharju 2001, Kaniel et al. 2012) However, decision making of individual investors can be very complex in non-linear environments. As Hsieh (1991) states, the stock market definitely represents a non-linear phenomenon. Data-driven models, which neural networks are, could most likely provide a more general framework for financial time series modelling than their linear parametric predecessor.

3 NEURAL NETWORKS

Neural networks have emerged as a practical supervised learning technology, having been successfully applied in many fields, with a majority of these applications being concerned with pattern recognition. Historically, studies in biological networks have inspired many concepts of neural computing. However, the point of view of statistical pattern recognition offers a more direct route to the concepts. (Bishop 1995)

The core idea is to model the output with a non-linear combination of features derived from the normalized inputs. In neural networks, a large number of individual units, neurons, are connected with different weights to others and by linear combinations process the data. The weights are adjusted in the training phase according to the error of the output, allowing the system to learn. (Hastie et al. 2017)

3.1 Multi-layer perceptron

Maybe the most common type of neural network, the multi-layer perceptron (MLP), is a feed-forward network, which consists of consecutive layers of neurons with no feedback loops between neurons. Starting from the input layer, each neuron gets its inputs from the neurons in the previous layer and outputs an activation for the neurons in the next layer. Finally, the last layer produces the outputs of the network, which are interpreted as probabilities. Layers between the input and the output layer are called hidden layers. (Bishop 1995) The number of neurons in the input layer depends on the form of the input data, whereas the output layer size should match the number of output classes. Choosing the correct number of hidden layers is guided by background knowledge and trials. (Hastie et al. 2017) A simple MLP with a single hidden layer is shown in figure 3.1.

The activation of the ith hidden neuron can be obtained by applying a non-linear activation function to the sum of the weighted inputs it receives from the neurons of the previous layer. Using a non-linear activation function allows for the network to handle non-linear patterns. The linear combination including the bias term w_{i0} is

$$s_i = \sum_{j=1}^{J} w_{ji} x_j + w_{i0}, \tag{3.1}$$

where J denotes the amount of inputs, x_j the jth input, w_{ij} the weight for the input j of neuron i and w_{i0} the bias term for neuron i. By introducing an extra input variable $x_0 = 1$,

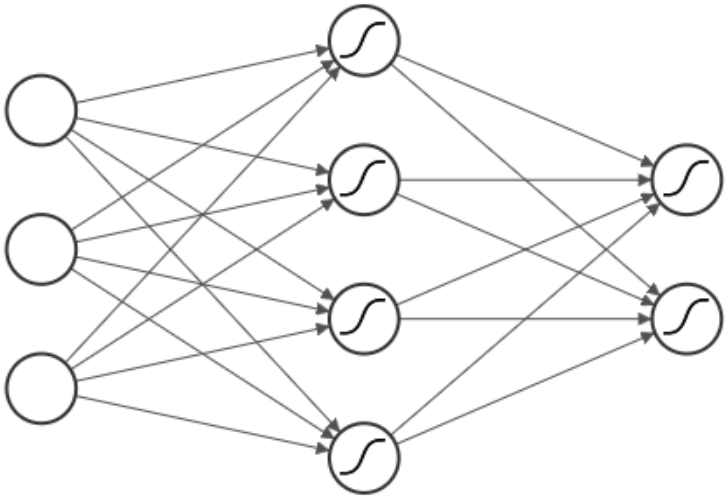

Figure 3.1. *A picture of a simple MLP network structure, with 3 inputs, a hidden layer with 4 neurons and 2 outputs. Neurons with activations are represented by the S-shaped curve. Adapted from Graves 2012.*

equation 3.1 can be rewritten as

$$s_i = \sum_{j=0}^{J} w_{ij} x_j. \tag{3.2}$$

The activation of neuron i can be received with

$$a_i = \sigma_i(s_i), \tag{3.3}$$

where $\sigma_i()$ is the activation function of neuron i. (Bishop 1995) The activation function is often chosen to be the non-linear differentiable *logistic sigmoid* function

$$\sigma(v) = \frac{1}{1 - e^{-v}}, \tag{3.4}$$

which squashes the input value v to a finite range (0,1) (Hastie et al. 2017). The neurons on the output layer work in the exact same way as the hidden neurons, though a different activation may be used (Mäkinen et al. 2019). A neuron is presented in figure 3.2.

The calculation of the activation vector of a layer l when $l > 0$ written in vector form is

$$\boldsymbol{a}^{(l)} = \sigma_l(\boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}), \tag{3.5}$$

where $\boldsymbol{a}$ is a column vector, which consists of activations a_j from the previous layer, $\boldsymbol{W}$ is a matrix consisting of row vectors $\boldsymbol{w}_i$, each holding a weights $w_{i,j}$ for input a_j^{l-1} of neuron i and l indexes the layer. The activation of the first layer ($l = 0$) is the input values of the network.

As mentioned before, the training of the networks consists of adjusting the inner weights

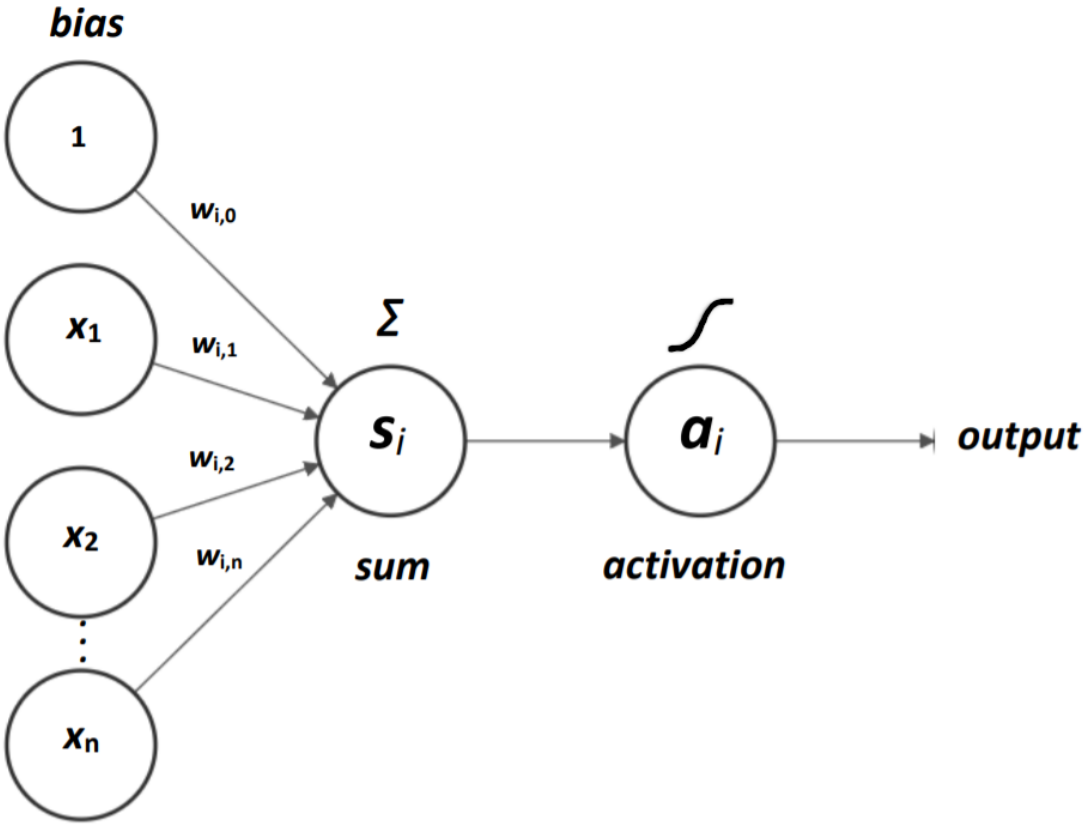

Figure 3.2. *Picture of a neuron with n inputs and a bias. The activation of this particular neuron is outputted and received as input for subsequent neurons. Should the neuron be one of the output layer, the output is interpreted as a probability of the original input belonging to some class. Adapted from Bishop 1995.*

in $\boldsymbol{W}$ according to the output's error. This is done in two phases, *forward pass* and *backward pass* (Bishop 1995).

In the forward pass, weights are stationary, and a prediction is calculated using 3.3. The networks prediction is compared to the known true class and its error is calculated with respect to a loss function. For classification, cross-entropy is often used. In multi-class classification, separate losses for each class of an observation is summed together to receive the loss

$$\mathcal{L}(\boldsymbol{W}) = -\sum_{k=1}^{K} y_{o,k} \log \hat{y}_{o,k}, \tag{3.6}$$

where $\boldsymbol{W}$ refers to weights and biases of the model, K is the number of different classes, $y_{o,k}$ indicates whether observation o belongs to class k out of all possible classes (the ground truth) and $\hat{y}_{o,k}$ is the models predicted probability of observation o belonging to class k. (Hastie et al. 2017)

In backward pass, the goal is to minimize the selected loss function. Algorithms which do this are called *back-propagation algorithms*. A common approach to minimizing the cross entropy is done by *gradient decent*. Due to the structure of the network and the activation

functions being differentiable, the gradient can be easily derived and new wights obtained

$$\boldsymbol{W}' = \boldsymbol{W} - \lambda \frac{\partial \mathcal{L}}{\partial W}, \tag{3.7}$$

where λ denotes the learning rate. (Hastie et al. 2017) Another famous optimization algorithm, which is used in this study, is ADAM (see Kingma and Ba 2014).

3.2 Recurrent neural networks and long short-term memory

Recurrent neural networks (RNNs) are generalizations of MLPs, in which neurons are connected within or between any layers forming internal feedback loops. These internal loops bring about recursive dynamics and allow delayed activation dependencies across neurons inside the network. (Graves 2012) Typically, in RNNs the inputs of a neuron consist of activations from previous layers, as well as a delayed input from the neuron itself. Furthermore, the sum of these weighted inputs is squashed with an activation function producing the activation of the neuron. RNNs, in which activations from the past affect future activations, encompass implicit memory. (Marhon et al. 2013) RNNs, due to their memory properties are appropriate to use for time-series data (Tsantekidis et al. 2017b).

Common learning algorithms used in training of RNNs, back-propagation through time and real-time recurrent learning, suffer from the error signal, namely the gradient, either blowing up or vanishing (Hochreiter & Schmidhuber 1997). The vanishing gradient makes learning distant connections virtually impossible for RNNs (Tsantekidis et al. 2017b). An improved version of RNNs is long short-term memory (LSTM), originally proposed by Hochreiter and Schmidhuber (1997), and further improved and popularized by other researchers (see e.g. Gers et al. 2000, Gers and Schmidhuber 2000). LSTM battle the vanishing or exploding gradient by exploiting both long-term memory, with the weights being slowly adjusted in training, and short-term memory, represented by the recurrent self connections. (Hochreiter & Schmidhuber 1997).

By introducing new units called Constant Error Carousel (CEC) neurons, which have a single fixed recurrent connection (weight 1.0), constant error flow through a unit during training can be achieved. Furthermore, a multiplicative sigmoid layer called "input gate" is introduced to safeguard the information stored in a CEC neuron, the cell state, from irrelevant inputs. Another multiplicative sigmoid layer, the "output gate", is introduced to ensure no irrelevant information in the cell state could affect other neurons receiving their inputs from it. The resulting, a more complicated unit is called a memory cell. Moreover, memory cells being controlled by the same input and output gate form a memory cell block. (Hochreiter & Schmidhuber 1997)

Gers et al. (2000) further introduced an additional multiplicative sigmoid layer, the "forget gate". The forger gate decides when the information within a CEC neuron should be forgotten by replacing the constant weight from the recurrent connection in a CEC neuron (Gers et al. 2000).

The training of an LSTM consists of steps including a forwards pass and a backward pass. In the former, all neurons are updated, while in the latter, error signals for all weights are computed. (Gers et al. 2000)

In the forward pass, in order to decide on what information should be forgotten in the cell state, the activation of the forget gate $\boldsymbol{f}_t$ at time t is calculated as using information from the previous activation $\boldsymbol{h}_{t-1}$, the current input $\boldsymbol{x}_t$ and a bias $\boldsymbol{b}_f$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{xf}\boldsymbol{x}_t + \boldsymbol{W}_{hf}\boldsymbol{h}_{t-1} + \boldsymbol{b}_f). \tag{3.8}$$

Next, the activation of the input gate

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{xi}\boldsymbol{x}_t + \boldsymbol{W}_{hi}\boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \tag{3.9}$$

is calculated using the same information as in the previous step. A *tanh* function creates a squashed vector representation of the candidate values for the hidden state denoted as $\boldsymbol{c'}_t$,

$$\boldsymbol{c'}_t = tanh(\boldsymbol{W}_{xc}\boldsymbol{x}_t + \boldsymbol{W}_{hc}\boldsymbol{h}_{t-1} + \boldsymbol{b}_c), \tag{3.10}$$

and is multiplied by the activation of the input gate to decide what new information should be added to the cell state, while the previous cell state $\boldsymbol{c}_{t-1}$ is multiplied by the forget gate for let some information be forgotten. This creates the protected cell state denoted as $\boldsymbol{c}_t$ as explained, calculated by

$$\boldsymbol{c}_t = \boldsymbol{f}_t\boldsymbol{c}_{t-1} + \boldsymbol{i}_t\boldsymbol{c'}_t. \tag{3.11}$$

To decide what information from the cell state should be outputted from this memory block, the activation of the output layer $\boldsymbol{o}_t$ is calculated by

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{oc}\boldsymbol{c}_t + \boldsymbol{W}_{ho}\boldsymbol{h}_{t-1} + \boldsymbol{b}_o). \tag{3.12}$$

Finally, the outputted activation $\boldsymbol{h}_t$ is received by multiplying the protected state by the output gates activation

$$\boldsymbol{a}_t = \boldsymbol{o}_t\sigma(\boldsymbol{c}_t). \tag{3.13}$$

In equations 3.6 to 3.11, the activation function σ is as presented in equation 3.4. (Gers, Schmidhuber & Cummins 2000; Hochreiter & Schmidhuber 1997; Tsantekidis et al. 2017b)

To further clarify the process, note that the activation of each gate is a vector of values between 0 and 1, indicating how much of the vector it multiplies element wise should be preserved. An LSTM memory block is shown in figure 3.3.

The weights in the LSTM are learned by optimising the cross entropy defined in equation 3.6, with $\boldsymbol{W}$ referring to the LSTMs wights and biases, i.e. $\boldsymbol{W}_{xf}$, $\boldsymbol{W}_{hf}$, $\boldsymbol{W}_{xi}$, $\boldsymbol{W}_{hi}$, $\boldsymbol{W}_{xc}$, $\boldsymbol{W}_{hc}$, $\boldsymbol{W}_{oc}$, $\boldsymbol{W}_{hc}$, $\boldsymbol{b}_f$, $\boldsymbol{b}_i$, $\boldsymbol{b}_c$ and $\boldsymbol{b}_o$. The loss is summed over each training batch and minimized. (Tsantekidis et al. 2017b)

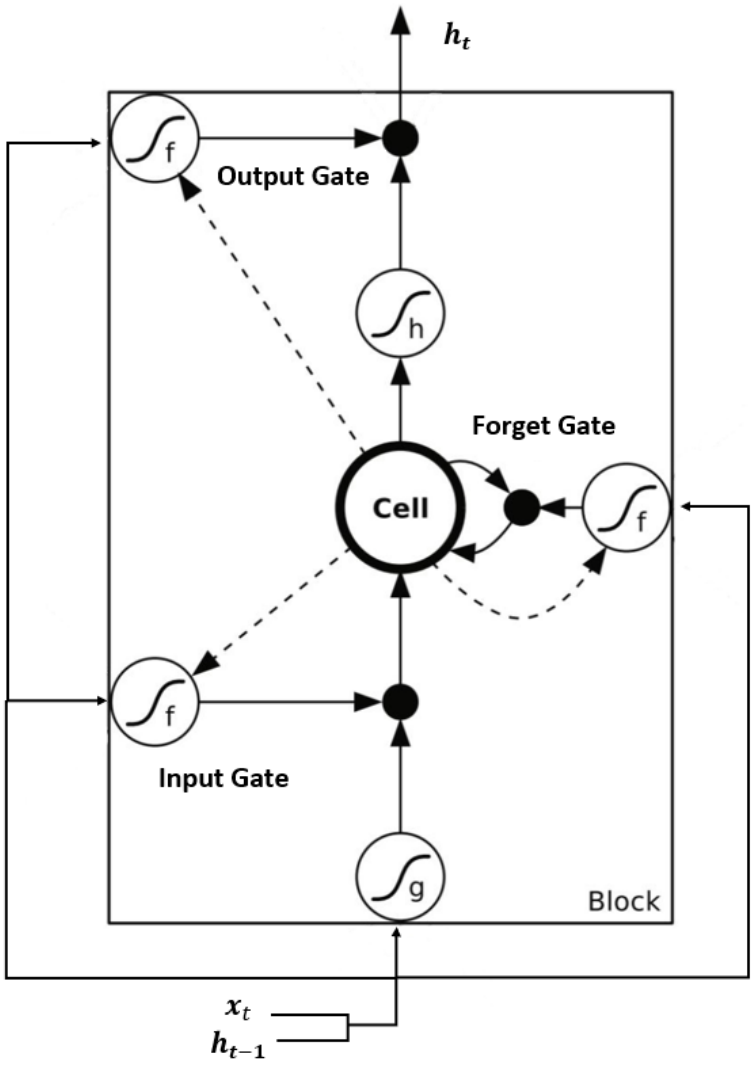

Figure 3.3. *LSTM block with a single cell. The model includes three activation gates with the activation function f. Regular input and output gates have activations g and h. Element-wise multiplications are portrayed as black cells. The connections between the cell and the gates are unweighted for solid lines (fixed weight of 1.0), and weighted for dashed lines. Adapted from Graves 2012.*

With LSTMs, a common optimising method is a variant of back-propagation called back-propagation through time. It is capable of accounting for the information flow of the input time series. (Graves 2012)

3.3 Convolutional neural networks

Convolutional neural networks (CNNs) are a class of neural networks, that mimic the way the mammal brain processes visual data. They are regularised versions of the MLP. Specific neurons can extract specific features from the input data, such as edges and corners, with emphasis on the feature itself being present in relation to other features, not their exact position. Small distortions and nonessential parts or background are left

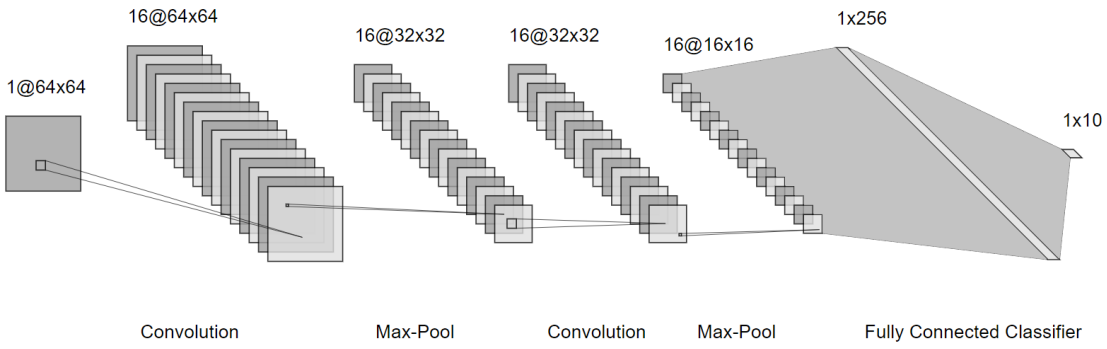

Figure 3.4. *The classification process of a single image of dimension 64×64. First, 16 differently weighted convolution windows of 8×8 neurons are applied, followed by a 2×2 max-pooling. The max pooling takes each the maximum of each 2×2 window, resulting in smaller feature maps. The process is repeated with another convolution-pooling pair. Finally, the last feature map is connected to two fully connected neuron layers, with the last having 10 neurons representing 10 possible classes.*

ignored. Adjacent data points are convolved in order to find certain relations. (LeCun & Bengio 1995)

In image recognition, normalized, resized and centered images are fed to the CNN input layer. Activations in the following hidden layers are calculated by multiplying adjacent activations from previous layers, all with specific weights assigned to them. Convolutional layers are followed by pooling layers to perform sub-sampling or averaging. This allows for the resolution to be lowered, while still keeping information of found features. (LeCun & Bengio 1995) After the last convolutional and pooling layer pair, fully connected layers are often used to combine the extracted features into a classification (Tsantekidis et al. 2017a). This process is described visually in figure 3.4, which shows an example CNN structure of an image recognition task.

Although CNNs were originally proposed for image recognition, in which they excel, they are capable of capturing patterns in time-series, both in feature and time-space. Therefore, they have been found useful in other domains where the relation of observation matters, for example in speech recognition or time-series prediction. (LeCun & Bengio 1995) Time series analysis using CNNs works similarly to with images, although the inputs dimensionalities are different. A common approach is to format the data in an $n \times k$ matrix, where n is the number of time steps and k is the number of features on each time step. The convolution is performed with m convolution filter of size (t, k), spanning one step at a time though the entire feature depth n, capturing the k features over t time steps. The result of such a convolution layer is a $n - t + 1 \times m$ matrix, to which further convolutions or poolings can be applied. Connecting the extracted features to a fully connected layer allows for prediction based on the features extracted. (Kim 2014) Figure 3.5 shows an example of feature-dimensional convolution.

In a similar manner to the MLP and LSTM, CNN parameters are learned by minimizing the loss function, for example the categorical cross entropy loss defined in equation 3.6,

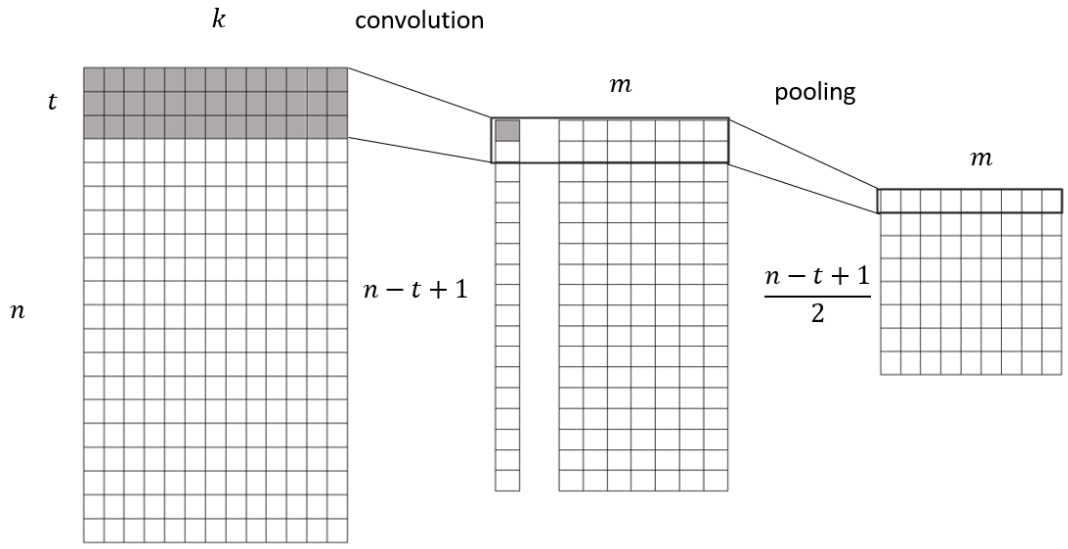

Figure 3.5. *m filters of size (t, k) on an $n \times k$ matrix. The gray area is the first filter of m filters. After the convolution, the dimensionalities shrink. Pooling is a max pooling of 2, taking the maximum value of each row on the columns. The resulting matrix can be further connected. Adapted from Kim 2014.*

where $\boldsymbol{W}$ is used to refer to the weights and biases in the CNN. Also with CNN, the most common minimizing method is gradient decent presented in equation 3.7. Again, ADAM is used in this study (see Kingma and Ba 2014). (Tsantekidis et al. 2017a)

3.4 Regularisation

The loss function defined on the training set allows for the optimised performance only on the training set. The real goal, however, is to optimise performance on previously unseen test data. Over fitting occurs, when the model learns the training data too accurately. The ability of a model to generalise well on unseen data is referred to as generalisation (Graves 2012). Regularisation techniques can be applied to improve generalisation.

A common regularisation technique, dropout, proposed by Hinton et al. (2012), refers to randomly dropping out a proportion of neurons in a neural network, preventing complex co-adaptations on training data. During the forward pass of each training sample, neurons have a predefined probability of its activations being "dropped out" from the information flow. Hinton et al. (2012) proposed the dropout probability to be 0.5, resulting in only half of the neurons being present at each forward pass. Hence, hidden units can not rely on presence of other hidden units, and under the circumstances, the network can not depend on a single passed value, and will be more robust (Mäkinen et al. 2019).

Another popular regularization technique is to penalise the model for large weights. A penalty term depending on the weight matrix $\boldsymbol{W}$ is added to the loss function to me minimised, encouraging the use of smaller coefficient. (Hastie et al. 2017)

Using a separate validation data set to assess whether the model is over fitting, is also a

common approach. Once the performance on the validation data stops improving, while the performance on training data keeps getting better, it can be deduced, that over fitting is occurring. (Graves 2012)

3.5 Attention layer

The method used to analyse the importance of different inputs is attention. Attention has been recently used in generation (Graves 2012), natural language processing (Bahdanau et al. 2016; Zhou et al. 2016) and stock price jump detection (Mäkinen et al. 2019). By giving weights, i.e. focusing on different parts of the input sequence, the attention mechanism generates an output. Hence, it enables observations at specific locations to have a greater impact in determining the output. (Mäkinen et al. 2019)

Here, the attention layer originally proposed for sentence relation classification by Zhou et al. (2016) is utilized in the way Mäkinen et al. (2019) used it on LOB data. In the attention layer, $\boldsymbol{H}$ denotes the layer's input matrix which consists of the recurrent layer's output vectors $\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_L$, where L is the input length?. The output vector $\boldsymbol{h}^*$ is calculated as the weighted sum of several output vectors:

$$\boldsymbol{M} = \tanh(\boldsymbol{H}) \tag{3.14}$$

$$\boldsymbol{\alpha} = \text{softmax}(\boldsymbol{w}^\mathsf{T}\boldsymbol{M}) \tag{3.15}$$

$$\boldsymbol{r} = \boldsymbol{H}\boldsymbol{\alpha}^\mathsf{T}, \tag{3.16}$$

where $\boldsymbol{H} \in \mathbb{R}^{dw \times L}$, where dw denotes the dimension of the observation vectors, w is a parameter vector, which is trained and w^T is its transpose and L is the sequence length. Finally, the output of the attention layer is

$$\boldsymbol{h}^* = \tanh(\boldsymbol{r}). \tag{3.17}$$

(Zhou et al. 2016) The softmax in equation 3.15 is a function that squashes the inputs to a range [1, 0]:

$$\text{softmax}(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}}, \tag{3.18}$$

where i denotes an element and j sums all the elements in the vector. The activation is calculated element-wise. (Bishop 1995)

Attention is a mechanism allowing for the neural network model to emphasise features most relevant for a given sample. Inputs with importance will have a high coefficient in the attention matrix $\boldsymbol{H}$. (Zhou et al. 2016) In this study, an attention layer is used to interpret the importance of different features of inputs. The goal is to get insight of the most relevant inputs which affect the behavior of active household investors.

4 DATA

The data used in this study is provided by EuroClear Finland. The data consists of tables with detailed information on transactions executed in Nasdaq Helsinki stock exchange between years 1996 and 2016. Each transaction, whether done by, for example, a corporation, a public administration organization, a mutual fund or a household, is present with information on the asset, transaction type, price, volume and trading date among others.

In addition to the data holding information on the entities partaking in the transaction, another data set is used, containing information on each trade executed in the Helsinki stock exchange. The instrument in question, as well as the price, volume currency and time, presented to the nearest second, are presented. This allows for the inspection of intra-day changes in the stocks of interest.

4.1 Selected data

To ensure that there are enough data whilst restricting the computational complexity of this study, the focus will be narrowed to the stock most actively traded by household investors in Nasdaq Helsinki between 2006 and 2010, that being Nokia Oyj stock. To further narrow down, only the 100 most active household investors were selected for inspection in this study.

Furthermore, the data available sets some restrictions on the extent of this study. Detailed intra-day stock data is available only from the beginning of the year 2006, whereas detailed investor data ceases being available in November 2009. Hence, the time frame chosen is a period from February 2006 to November 2009. The actual prepared data is 914 trading days, due to various inputs needing to be calculated some days prior to the actual date they are used. Further, the selected data is divided into training and testing data sets.

Training data is fed to the neural networks in the training phase, and by adjusting the weights through the optimization algorithm, the networks is able to learn. After training, the testing set, which until now has been kept separate from the model, is used for assessment of the predictive capabilities of the model developed. (Graves 2012; Hastie et al. 2017)

Altogether 893 days are selected for training. To better resemble the way these models can be utilized, they will be evaluated using walk forward validation. First the models are

trained for over two and a half trading years or 662 days. After this, they are tested with the next 21 days or a month of trading days. Next, the models are fit with these testing data, and the resulting model again evaluated with the next 21 days. This process will be repeated 12 times. In total, there is 252 days or a full trading year of testing data. Table 4.1 shows all the 12 different training and testing periods.

Data set	Training period	Testing period
1.	0 - 661	662 - 682
2.	0 - 682	683 - 703
3.	0 - 703	704 - 724
4.	0 - 724	725 - 745
5.	0 - 745	746 - 766
6.	0 - 766	767 - 787
7.	0 - 787	788 - 808
8.	0 - 808	809 - 829
9.	0 - 829	830 - 850
10.	0 - 850	851 - 871
11.	0 - 871	872 - 892
12.	0 - 892	893 - 913

Table 4.1. *Testing data division in to 12 sets of length 21.*

This validation is similar to to the way such models could be used. Over time, more information on the investors will accumulate, and it can be used to better train the models. This split it chosen to utilize all the available data. Less emphasis is given to the most recent data, in order to the models no to overfit on it.

4.2 Investor activity (output data)

The ground truth vector for the neural network models used in this study for day j is denoted as $y_j \in \mathbb{R}^A$, where A is the amount of active traders being considered. Hence, the ground truth matrix is $\boldsymbol{Y} \in N^{A \times T'}$, where T' is the amount of trading days in the period. Consequently, each element $y_{i,j}$, $1 \leq i \leq A$, and $1 \leq j \leq T'$ holds a value of 1, 0 or -1, indicating, whether the ith investor on day j has net-bought, kept their position or net-sold, respectively. Later in this study, class 0 refers to a sample belonging to the stationary class, class 1 to net-selling and class 2 to net-buying class.

All together in the selected 100 active household investors over 914 days, there are 91400 data points of output data. Of them, 90% represent investors holding on to their position, i.e. class 0, resulting in a high imbalance between the stationary class and the net-buy and net-sell ones, which both account for roughly 5%. Class imbalance and sparsity in classification is a common problem in machine learning (Zou et al. 2016). The problem will be tackled with setting a class weights to the minority classes, giving relatively more

emphasis on samples belonging to them (Chollet et al. 2015).

4.3 Market and investors state (input data)

Various inputs capturing the state of the market and the past activities of the traders are fed to the models as inputs. The inputs are enumerated and explained below. The stock specific, for example the intra-day volatility, is fed to the model as a vector allowing for the model to receive the intra-day volatility for multiple days prior. All inputs used in this study are:

1. Log-return vector
2. Realized intra-day volatility vector within 5-minute intervals
3. Total volume vector
4. Euro-volume of the previous transaction day
5. Time passed since previous transaction day
6. The action of the investor on the previous day
7. The proportion of the 100 investors being active on the previous day

The log-return of the stock on day j calculated over days $j, \ldots, j-k+1$ is

$$\mathsf{R}_{j,k} = \ln \frac{S_j}{S_{j-k}}, \tag{4.1}$$

where S_j is the closing price of the stock on day j. k denotes from how many days the value is calculated.

Grinblatt and Keloharju (2001) suggests that returns over intermediate or long-term horizons have no evidence of affecting the propensity to trade. Thus, log-returns for the underlying stock are calculated over the past 1, 5 and 20 days.

Realized intra-day volatility for day j calculated over days $j, \ldots, j-k+1$ is

$$\mathsf{RV}_{j,k} = \frac{1}{k} \sum_{l=0}^{k-1} \frac{1}{N_{j-l} - 1} \sum_{h=2}^{N_{j-l}} \left(\ln \frac{S_{j-l}(h)}{S_{j-l}(h-1)} \right)^2, \tag{4.2}$$

where $S_j(h)$ is the hth observation on the stock price on day j and N_j is the number of price observations (here sampled every 5 minutes) on day j.

Total volume vector for day j calculated over days $j, \ldots, j-k+1$ is

$$\mathsf{V}_{j,k} = \sum_{l=0}^{k-1} \mathsf{V}_{j-l}, \tag{4.3}$$

where V_j is the overall trading volume of the stock on day j. Again, calculated for $k = 1, 5, 20$.

In addition to the previous inputs concerning the state of the market, inputs specific to investors and their behaviour are also included. Euro-volume for the previous day the investor changed their position is calculated by multiplying the volume with which they bought by the price of the purchase. Then, it is netted for each day. Another input will keep track of the number of trading days this last position changing day occurred. As long as the investor has not changed their position since the beginning of 2006, both the above-mentioned inputs will stay 0 for that particular investor. Finally, inputs representing the action of a particular investor and the 100 investors as a group on the previous day, will be used.

Proper normalization of the inputs of a neural network can highly affect the quality of the predictions, by mitigating the uneven effect of inputs with different size ranges (Hastie et al. 2017). Log-return and intra-day volatility are normalized by nature since they measure relative change. The number of active investors and the time since last trade are normalized by dividing the number of investors in total and the length of the training period, respectfully. Furthermore, the total volume input is normalized using the z-score in order to eliminate irrelevances and noise, possibly originating from different starting volumes:

$$\boldsymbol{x}_{norm.} = \frac{\boldsymbol{x} - \overline{x}}{\sigma_x^2}, \tag{4.4}$$

where $\boldsymbol{x}$ is the vector to be normalized, $\overline{x}$ is the mean and σ_x^2 the variance (Hastie et al. 2017). The mean and variance are calculated from the past 20 days.

Finally, to mitigate the effect of differences in the order of magnitude, euro-volume of an investors last transaction is divided by the maximum euro-volume in the training period of each investor. Due to this, behaviour of investors trading using varying quantities can be compared and analyzed together.

5 RESULTS

The neural networks are implemented using various Python libraries. The main library used, Keras, is an open-source neural network API, allowing for the user-friendly and flexible implementation and testing of neural networks. It, however, only provide the building interface for neural networks, needing to be used on a back-end with the actual algorithm implementations. (Chollet et al. 2015) Consequently, Tensorflow, was used as the back-end. Tensorflow is an open-source platform, with an extensive set of tools allowing for the implementation of state-of-the-art machine learning applications (Abadi et al. 2015).

Keras provides a Model class, allowing to instantiate neural networks by providing just the input and the output layers. Connections between consecutive layers can be specified in a simple manner and all layers required to calculate the activations on the output layer will be included in the model. (Chollet et al. 2015)

In this study, different neural network topologies will be tested to find out which works best for the proposed problem. Various widths, meaning number of neurons per layer, and depths, referring to the number of hidden layers, will be experimented on for each network type (MLP, LSTM, CNN), and the best topology of each will be selected for attention layer examination. All the models will be trained on each of the 100 investors' data on the training periods, resulting in investor agnostic models being capable of predicting the actions of investors previously unseen to the models. The performance will be tested over the 252 day long testing period in 12 sets of length 21. Once the best topology of each network type is found, their attention layer will be qualitatively analyzed.

5.1 Model evaluation metrics

The main metric used is the mean F1 score. The F1 score for class k is the harmonic mean of the precision and recall:

$$\mathrm{F1}_k = \frac{2}{{p_k}^{-1} + r_k^{-1}}, \tag{5.1}$$

where p_k is the precision, the proportion of samples predicted as k actually belonging to k, and r is the recall, the proportion of actual samples belonging to k predicted to belong to k. Precision and recall are calculated by

$$p_k = \frac{tp_k}{tp_k + fp_k} \tag{5.2}$$

and

$$r_k = \frac{tp_k}{tp_k + fn_k}, \tag{5.3}$$

where tp_k is true positives, the number of samples of class k correctly classified as belonging to class k, fp_k is false positives, the number of samples belonging to a different class falsely classified as belonging to class k and fn_k is false negatives, the number of sample belonging to class k falsely classified to belong to another class. (Zou et al. 2016)

In the multi-class problem at hand, the chosen metric will be macro F1, also called the per label F1 or mean F1. It is calculated as the arithmetic mean of each classes individual F1. Mean precision and mean recall are calculated similarly from the per-class scores and used to evaluate the models. Since F1 score is a nonlinear, asymmetric measure, it is commonly used to asses models in cases, where the predicted classes are highly imbalanced. (Zou et al. 2016) A model with a high F1 score is able to correctly classify samples to their respective classes, highlighting the importance of performing well on classes with a small number of samples (Mäkinen et al. 2019).

However, a flaw in the F1 score lies in the fact that it treats the precision and recall as equal (Hand & Christen 2018). Moreover, Hand and Christen (2018) suggests, that precision and recall's relative importance should depend on the problem, for different errors have different effects and implications. However, in the investor behavior problem at hand, there is no clear basis as of how to emphasise different error types.

Another metric used to asses the models is the Cohen's Kappa:

$$\kappa = \frac{p_o - p_c}{1 - p_c}, \tag{5.4}$$

where p_o is the observed agreement ratio and p_c is expected agreement by chance, and both can be calculated from the confusion matrix (Cohen 1960).

A confusion matrix of a three class problem is presented below.

		True class			
		Class 0	Class 1	Class 2	Total
Predicted class	Class 0	a	b	c	$a+b+c$
	Class 1	d	e	f	$d+e+f$
	Class 2	g	h	i	$g+h+i$
	Total	$a+d+g$	$b+e+h$	$c+f+i$	N

From here,

$$p_o = \frac{a+e+i}{N}, \tag{5.5}$$

and

$$p_c = \frac{a+d+g}{N}\frac{a+b+c}{N} + \frac{b+e+h}{N}\frac{d+e+f}{N} + \frac{c+f+i}{N}\frac{g+h+i}{N}. \tag{5.6}$$

Cohen's kappa was originally used to evaluate the agreement between two independent

evaluators, but was later adopted to measure classification performance. As such, it measures the agreement degree between the predicted classes and the true classes. (Pedregosa et al. 2011) It is often seen as a more robust performance measure, since it takes into account the possibility of agreement by chance (Mäkinen et al. 2019). A Cohen's Kappa of 1 indicates, that the level of agreement is not at all affected by chance, whereas a value of 0 suggests the agreement is purely caused by chance. A value less than 0 would indicate a performance worse that randomly guessing.

In addition, to have a reference for capabilities of the implemented networks, their results are compared to two benchmarks, a stationary strategy and a buy-sell prediction strategy. In the former, each prediction is just one of the major class 0, whereas in the latter benchmark, a net-buy day is predicted subsequent to a net-sell day, and vice versa. Here, no position change is predicted for days preceded by a stationary day. This benchmark is later referred to as "buy-sell".

5.2 Network implementation and experiment results

To tackle the problem of class imbalance, class weights of 6.3 and 5.3 for both the minority classes 1 and 2 are used, resulting in the models giving more emphasis for instances of classes 1 and 2, treating an instance of class 1 or 2 as if receiving 6.3 and 5.3 of those samples (Chollet et al. 2015). The optimal weight was found by exploring on weights. The weights are inversely proportional to the number of samples.

Furthermore, dropout layers are added to all the models. The optimal location and dropout rate is explored. For the MLPs, the optimal rate was found to be 0.4, located after each hidden layer. For LSTMs, a dropout rate of 0.25 is introduced after the LSTM layer, and for the CNNs, a dropout of 0.25 after the two consecutive layers is applied.

The attention layer described in chapter 3 is applied straight to the input of all the networks. Experiments were conducted with and without attention layer, but it was noticed that the results did not differ. With only 13 features, the performance betterment received by the attention layers is not expect to be high. With a large number of input features, attention layer can better focus down to particular features and improve the prediction results (Mäkinen et al. 2019).

For the MLP, the input consists of the 13 features introduced in chapter 4 as a vector of length 13, while the target is a one-hot encoded representation of classes 0, 1 and 2. Each MLP model uses a batch size of 16. The first, longer training set is shown for 50 epochs to allow for learning of longer trends. When introducing the new 21 day periods, training is done for only 10 epochs, in order to incorporate the new information but not to overfit on this new data. In both training settings, the learning rate is set to 0.001.

To find the best MLP topology, various networks are evaluated, with 4 different width and depth multipliers, ranging from 1 to 4. The number of neurons on a layer is received by multiplying the base number of neurons by the width multiplier of the network, and the

number of hidden layers is represented by the network depth multiplier. A base number of 16 neurons will be arbitrarily chosen. Consequently, 16 topologies denoted by the width and depth are received. For each of the 16 topologies, the experiment is ran 3 times and the mean of all the used metrics are presented in table 5.1.

Classifier	Mean Precision	Mean Recall	Mean F1	Cohen's κ
1-1 MLP	0.465	0.475	0.458	0.267
2-1 MLP	0.473	0.488	0.466	0.271
3-1 MLP	0.467	0.492	0.464	0.274
4-1 MLP	0.463	0.497	0.463	0.275
1-2 MLP	<u>0.478</u>	0.478	0.462	0.278
2-2 MLP	0.468	0.487	<u>0.467</u>	<u>0.279</u>
3-2 MLP	0.466	0.494	0.464	0.275
4-2 MLP	0.464	<u>0.498</u>	0.464	0.273
1-3 MLP	0.426	0.451	0.430	0.254
2-3 MLP	0.458	0.487	0.461	0.278
3-3 MLP	0.464	0.486	0.462	0.275
4-3 MLP	0.461	0.495	0.463	0.272
1-4 MLP	0.391	0.422	0.401	0.215
2-4 MLP	0.401	0.454	0.411	0.226
3-4 MLP	0.434	0.478	0.441	0.252
4-4 MLP	0.449	0.496	0.451	0.260

Table 5.1. *Metrics for 16 different MLP topologies. In the naming of the classifier, the first number represents the width multiplier and the second the depth multiplier, i.e., the number of hidden layers. The best value for each metric is underlined.*

It can be seen, that the metrics used to evaluate the models do no differ much. Anyhow, the 2-2 MLP performs the best with respect to the F1-score. This best performing topology considering F1 has 2 hidden layers, both with 32 neurons. It also received the best Cohen's κ -score by a slight margin.

Next, different LSTM topologies are experimented on. From initial testing, it was seen, that the performance of LSTM with more than 2 hidden layers quickly deteriorates. Thus, networks depths of only 1 and 2 are presented, with networks widths of 1, 2, 3, 4 and 5. Thus, a total of 10 different topologies are tested. The window tells how many time steps are visible for the network, here chosen to be 20. The inputs consist of sequences of the window length, with all 13 features presented on all time steps, resulting in inputs of size 20×13. Again, the models are trained with a batch size of 16 and over 50 epochs. Testing is done in the same walk forward manner as with the MLP. Mean results of the chosen metrics over 3 training and testing instances are presented in table 5.2.

It seems as if the LSTM networks are more capable of predicting these data with these settings. Various widths and depths receive similar scores, and interestingly, the best

Classifier	Mean Precision	Mean Recall	Mean F1	Cohen's κ
1-1 LSTM	<u>0.468</u>	0.531	<u>0.469</u>	0.277
2-1 LSTM	0.466	0.540	0.466	0.270
3-1 LSTM	0.455	<u>0.542</u>	0.460	0.259
4-1 LSTM	0.462	0.541	0.467	0.273
5-1 LSTM	0.453	<u>0.542</u>	0.460	0.259
1-2 LSTM	0.461	0.509	0.461	0.275
2-2 LSTM	0.456	0.516	0.461	0.273
3-2 LSTM	0.458	0.531	0.462	<u>0.290</u>
4-2 LSTM	0.452	0.531	0.461	0.273
5-2 LSTM	0.456	0.523	0.468	0.288

Table 5.2. *Metrics for 10 different LSTM topologies. In the naming of the classifier, the first number represents the width multiplier and the second the depth multiplier. Again, best value for each metric is underlined.*

scoring topologies with respect to the Cohen's κ and F1-score were completely different topologies, the former having 2 hidden LSTM layers with 48 neurons and the latter 1 layer with 16 neurons. Nonetheless, 1 hidden 16 neuron layer LSTM received the best mean macro F1-score.

Finally, different CNN models are trained, with the input being the same 13 features, now presented over the previous 20 days, resulting in a 20×13 matrix. Again, targets are one-hot encoded representations of the 3 classes. A batch size of 16 is used, and training done for 50 epochs with a learning rate of 0.001. Testing is done in the same way as with the MLP and LSTM.

In the case of CNNs, widths of 1 to 4 and depths 1 and 2 are experimented on. Here, the widths multiplies the base number of filters used in the convolution layers. A base of 16 filters is chosen. Moreover, a base layer is chosen to consist of 2 1D convolutions followed by pooling. Consequently, with depth 2 there are 2 layers of 2 convolutions and a pooling. The convolution windows are chosen to be of length 3. As explained in chapter 3, when using one dimensional convolutions, height of the input reduces due to the 1D convolution windows sliding and max-poolings. Thus, in the case of CNN, only 2 network depths, number of hidden layers, are tested. Altogether, 8 different CNN topologies are trained and evaluated. The results are presented in table 5.3.

Worst of the 3 networks types tested here was the CNN, receiving high macro recall scores but trailing with respect to the precision. Due to this, the F1 score did not rise as high as with the 2 other networks types, neither did the Cohen's κ. Still, the best F1, Cohen's κ and macro Recall was received with a network with 16 filters on each layer and a total of 4 layers, with 2 layers followed by max-pooling of 2, and further followed by 2 layers and a global average-pooling layer.

Classifier	Mean Precision	Mean Recall	Mean F1	Cohen's κ
1-1 CNN	0.437	0.472	0.432	0.231
2-1 CNN	0.414	0.475	0.422	0.215
3-1 CNN	0.426	0.455	0.418	0.208
4-1 CNN	0.430	0.464	0.424	0.203
1-2 CNN	0.440	<u>0.511</u>	<u>0.450</u>	<u>0.252</u>
2-2 CNN	<u>0.441</u>	0.503	0.446	0.228
3-2 CNN	0.429	0.486	0.440	0.228
4-2 CNN	0.419	0.454	0.424	0.200

Table 5.3. *Metrics for 9 different CNN topologies. The first number in the name represents the width multiplier, i.e. the multiplier of the number of filters per convolution layer. The second number is the depth multiplier, the number of time the base layer is present. Again, the best scores are underlined*

Best of each model type rated by F1-score is presented together and compared to the thresholds. The mean precision, mean recall, mean F1 and Cohen's κ are shown in table 5.4

Classifier	Mean Precision	Mean Recall	Mean F1	Cohen's κ
Zeros	0.309	0.333	0.321	0.000
Buy-Sell	<u>0.483</u>	0.487	<u>0.484</u>	<u>0.289</u>
2-2 MLP	0.468	0.487	0.467	0.279
1-1 LSTM	0.468	<u>0.531</u>	0.469	0.277
1-2 CNN	0.440	0.511	0.450	0.252

Table 5.4. *The best performing topology if each model type with respect to the F1 score being compared with the two thresholds. The attention layers is later examined for the MLP, LSTM and CNN presented here.*

All tested networks types scored higher in all 4 evaluation metrics than the zeros benchmark, which constantly predicts stationarity. Compared to the "buy-sell" threshold, LSTM and CNN were able to receive better macro recall, while the MLPs recall is exactly the same. With respect to the other metrics, the models perform worse. However, the differences are minor. It seems like all the models were able to pick up a similar pattern from the used data. It might be, that there exists no other pattern to be discovered, but confidently stating this would require more indepth research. It might just as well be the case, that better prediction scores could be received with different models and settings.

5.3 Input importance

Attention is examined for the best of each of the model type being considered. The attention layer is connected right after the input layer, in the same way done in Mäkinen et

al. (2019). First, a tanh-activation is taken from the input, followed by a dense layers with softmax activation, giving weight to each feature, further flattened to a single dimension. In order to apply the attention to the full time frame, dense layer is repeated for each time steps. Next, the dimensions are permuted to match the original inputs shape. Finally, the original input is multiplied by the attention layers output, applying the weight of a feature to that feature in all the time steps. By examining the weights in the attention layer, insight can be drawn, to which input feature affects the result the most.

Though the attention layer proposed by Zhou et al. (2016) was originally presented to weight different time steps, it also finds use in highlighting different features (Mäkinen et al. 2019). As mentioned before, experiments were conducted without and with at attention layer. When having a large number of inputs, the possibility to focus down on particular input features can have significant performance improvements (Mäkinen et al. 2019). However, with a low number of input features, the performance improvement received by the attention layers is not expect to be significant. The main goal in using it is to gain insight of the relative importance of the inputs.

Turns out, with all 3 model types the highest absolute value was given to the first feature, which is the previous activities of the investor. Furthermore, with all the models, relatively high weight was given to the euro-volume of the last transaction. Interestingly, with the MLP the total trading volume appeared to be important, with the past 5 day trading volume received also a high weight.

The same was found with LSTM and CNN, with relatively high weight given for the trading volume, but for them over 20 previous days. Differing from the other two model types, the LSTM also weighted the log-return on all 3 time frames strongly, while the CNN and MLP gave them the least weight.

According to this tentative and superficial input importance analysis, the previous actions, the magnitude of the last action and the total trading volume of the stock are most important. It should be noted, that the weights given to different features varied with different training instances. However, the relation of the importance was most of the time as presented above. To gain a more exact understanding of the decision making of household investors, more precise studies need to be carried out.

6 CONCLUSIONS

This study aimed to answer, which network type, MLP, LSTM, or CNN can best predict active household investors' decisions: net-buy, stationary or net-sell. Out of the experimented network topologies and experiment settings, the LSTM, with 16 neurons on the single hidden layers produced the best prediction results with respect to F1-score, although the difference in performance was marginal. All the tested models were able to perform similar to the "buy-sell" -benchmark used in this study. The qualitative analysis on the attention layers suggests, that previous action, the magnitude of it and the trading volume of the stock are of most importance.

As mentioned in chapter 2, investors are heterogeneous. Due to this property, an investor agnostic model could find it hard to learn a universal trading pattern for active household investors. However, based on these experiments, one can not conclude, that such investor agnostic models are not capable of predicting future investor behavior, only that the used settings didn't allow for praiseworthy performances. For future research, models more sophisticated can be applied to similar data.

Further studies could focus on the behavior of individual investors. Giving ML models information on the investor and allowing for learning individual trading patterns or strategies, could give more insight to the importance of various features and the predictive abilities of different model types. Analyzing changes in their strategy, relation of active and inactive periods and return on other assets traded could bring more insight. Also, social networks of similarities could further clarify the extent of investor decisions depending on other particular investors. The possibility of nonstationarity of the trading strategies of household investors could also be considered when analyzing using ML methods.

The similarity of the prediction results for all the models could also suggest, that the used data does not have other structure for the models to learn, than the one learned by all of them. To reliably state this, a more thorough study should be conducted. Using different features, and data on different stocks or other assets, could give an exhaustive answer the research questions of this study.

REFERENCES

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*.

Aoun, J. E. (2017). *Robot-Proof: Higher Education in the Age of Artificial Intelligence*. Gildan Media.

Bahdanau, D., Cho, K. and Bengio, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv.org*.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Chollet, F. et al. (2015). *Keras*.

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. eng. *Educational and Psychological Measurement* 20.1, 37–46.

Dhar, R. and Zhu, N. (May 2006). Up close and personal: investor sophistication and the disposition effect. *Management science* 52.5, 726–740.

Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance* 25.2, 383–417.

Foucault, T., Sraer, D. and Thesmar, D. J. (2011). Individual Investors and Volatility. *Journal of Finance* 66.4, 1369–1406.

Gers, F. A. and Schmidhuber, J. (July 2000). Recurrent nets that time and count. *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*. Vol. 3, 189–194 vol.3.

Gers, F. A., Schmidhuber, J. and Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural Computation* 12.10, 2451–2471.

Graves, A. (2012). *Supervised Sequence Labelling with Recurrent Neural Networks*. 1st ed. 2012. Springer Berlin Heidelberg.

Grinblatt, M. and Keloharju, M. (2001). What Makes Investors Trade?: *The Journal of Finance* 56.2, 589–616.

Hand, D. and Christen, P. (2018). A note on using the F-measure for evaluating record linkage algorithms. *Statistics and Computing* 28.3, 539–547.

Hastie, T., Tibshirani, R. and Friedman, J. (2017). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer.

Hens, T. (2010). *Financial Economics A Concise Introduction to Classical and Behavioral Finance*. 1st ed. 2010. Springer Berlin Heidelberg.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation* 9.8, 1735.

Hsieh, D. A. (1991). Chaos and Nonlinear Dynamics: Application to Financial Markets. *Journal of Finance* 46.5, 1839–1877.

Kahneman, D. and Tversky, A. (1979). Prospect Theory: An Analysis of Decision under Risk. *Econometrica* 47.2, 263–291.

Kaniel, R., Liu, S., Saar, G. and Titman, S. (2012). Individual Investor Trading and Return Patterns around Earnings Announcements. *Journal of Finance* 67.2, 639–680.

Kim, Y. (2014). *Convolutional Neural Networks for Sentence Classification*.

Kingma, D. P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*.

Knüpfer, S. and Puttonen, V. (2018). *Moderni rahoitus*. 10., uudistettu painos. Helsinki: Alma Talent.

LeCun, Y. and Bengio, Y. (Nov. 1995). Convolutional Networks for Images, Speech, and Time-Series. *The handbook of brain theory and neural networks*, 255–258.

Mäkinen, Y., Kanniainen, J., Gabbouj, M. and Iosifidis, A. (2019). Forecasting jump arrivals in stock prices: new attention-based network architecture using limit order book data. *Quantitative Finance* 19.12, 2033–2050.

Marhon, S. A., Cameron, C. J. F. and Kremer, S. C. (2013). Handbook on Neural Information Processing. Springer Berlin Heidelberg, 29–65.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 2825–2830.

Shefrin, H. and Statman, M. (1985). The Disposition to Sell Winners Too Early and Ride Losers Too Long: Theory and Evidence. *The Journal of Finance* 40.3, 777–790.

Shiller, R. J. (1981). Do Stock Prices Move Too Much to be Justified by Subsequent Changes in Dividends?: *The American Economic Review* 71.3, 421–436.

Thaler, R. H. (2005). Princeton University Press.

Tran, D. T., Iosifidis, A., Kanniainen, J. and Gabbouj, M. (2019). Temporal Attention-Augmented Bilinear Network for Financial Time-Series Data Analysis. *IEEE Transactions on Neural Networks and Learning Systems* 30.5, 1407–1418.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M. and Iosifidis, A. (July 2017a). Forecasting Stock Prices from the Limit Order Book Using Convolutional Neural Networks. 7–12.

– (Aug. 2017b). Using deep learning to detect price change indications in financial markets. *2017 25th European Signal Processing Conference (EUSIPCO)*, 2511–2515.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R. and Bengio, Y. (2015). *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*.

Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H. and Xu, B. (2016). Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. *Proceedings*

of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Berlin, Germany, 207–212.

Zou, Q., Xie, S., Lin, Z., Wu, M. and Ju, Y. (2016). Finding the Best Classification Threshold in Imbalanced Classification. *Big Data Research* 5, 2–8.