AKADEMIA GÓRNICZO – HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ INŻYNIERII MECHANICZNEJ I ROBOTYKI

PRACA DYPLOMOWA
inżynierska

Uczenie głębokie w zadaniu uniwersalnego przewidywania ruchów cenowych akcji

Deep learning model for general stock price movement prediction

Autor: **Jakub Paweł Mosiński**
Kierunek studiów: Automatyka i Robotyka
Opiekun pracy: **dr inż. Ziemowit Dworakowski**

……..
podpis

Kraków, rok 2020

Contents

Chapter 1

Introduction

1.1 Stock Exchange

Contemporary, almost every country has a stock market. The first stock exchange formed in 1801, the London Stock Exchange (LSE). Unfortunately, companies were not allowed to issue shares until 1825. However, the New York Stock Exchange (NYSE) established in 1817 could trade shares form the very beginning. The NYSE dominated the global economy for more than a century until its first challenger in 1971, NASDAQ. It has been organized differently from a traditional stock exchange. NASDAQ does not have a physical location. Instead, it is held solely on a computer network where all trades are performed electronically. Through the years, competition between NASDAQ and the NYSE forced both markets to expand and innovate. In 2007, the NYSE merged with Euronext to create the first transatlantic stock exchange in the world - NYSE Euronext.

Nowadays, nearly every country has a stock market. In the advanced world, major stock markets rose in the 19th and 20th centuries, shortly after the LSE and NYSE. Each day, trillions of dollars are exchanged globally. Stock markets are the powerhouse of the capitalist system. Therefore, the understanding of their importance, as well as the ability to manage their potential, is imperative to prosperity.

1.2 Trading vs Investing

There are two types of stock market players, the traders and the investors. The former claim that the short-term price behaviour can be predicted using technical indicators, price or candle formations (see figures 1.1, 1.2). Many hedge funds worldwide are known to incorporate technical analysis to their decision-making process. Some studies claim that technical analysis does work. Nonetheless, there must be high volatility in the price to benefit from that approach [35].

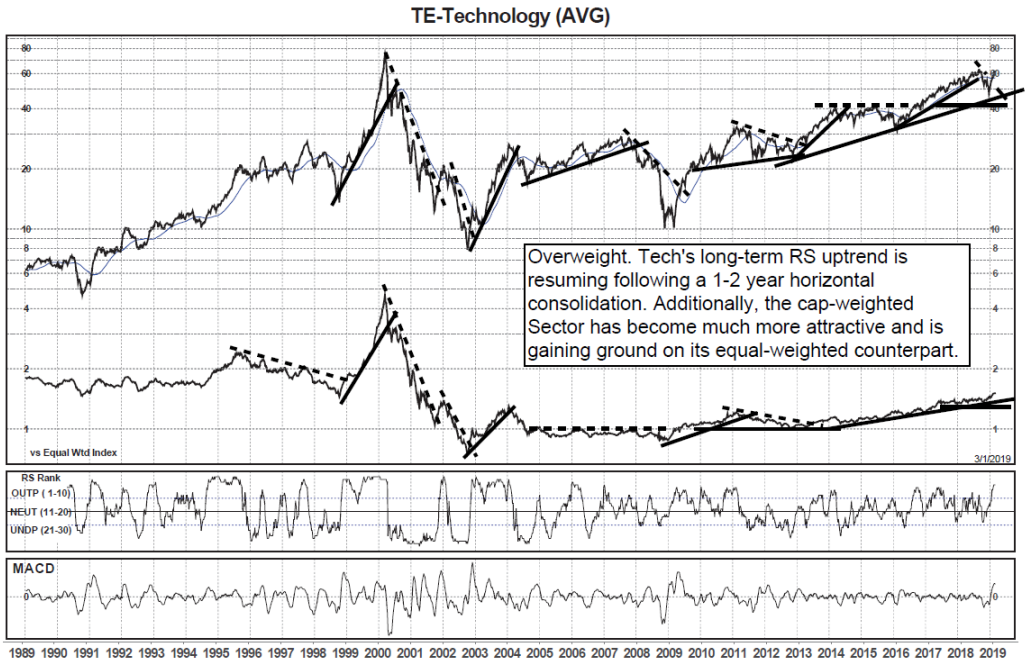

Figure 1.1: Example of technical analysis

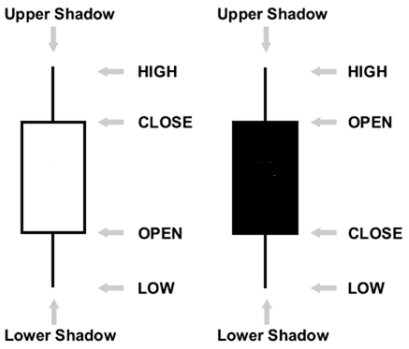

Figure 1.2: Candle stick

On the other hand, there are fundamentalists, who say that the short-term price behaviour is a random walk. Therefore, the only way to benefit from the stock market is to analyze companies based on earnings, revenue, the potential of their products in the long run. Hoffmann and Shefrin in their paper [15] stated that individual investors incorporating technical analysis methodologies perform worse than the fundamentalists. It might be because the approach is complicated. Therefore, not suitable for everyone or the technical analysis is just not working.

Hopefully, there are available methods to resolve this argument. Artificial neural networks (ANN) are known as universal function approximators [16]. It means that they can model complex formulas or signals such as stock price behaviour. The only issue is

that deep neural networks require substantial amounts of data and computational resources to learn the desired function. Fortunately, we live in the data age with the availability of superior computing power. What is more, many investment funds and banks are known to use deep learning techniques to predict future prices. Nevertheless, there is little publicly available research regarding that topic. The studies available often focus on a specific company. Thus, do not prove whether there are general rules for stock trading based on technical analysis. ANN can arguably be used to predict if traiding is indeed a valid option.

1.3 Aim and Scope

The purpose of the diploma thesis is to examine whether the future stock price movement can be predicted based on previous price history and volume employing universal rules. Most of the related studies focus on modelling the specific stock company or index. They do not answer the stated query. For that reason, this dissertation proposes a deep learning model for general stock price movement prediction.

The study includes:

- literature review of previous and contemporary methods of financial data modelling,
- implementation of deep recurrent neural network with Long Short-Term Memory cells,
- architecture design and hyperparameter tuning,
- data preprocessing and feature engineering,
- influence of history horizon as well as prediction horizon on accuracy,
- potential of price movement prediction based on technical data

1.4 Structure of The Thesis

The following content of the thesis is structured as follows: Chapter 2 - Related Work, sums up previously undertaken research in the topic of stock market price forecasting and use of technical analysis for trading purposes. It provides an insight into the argument between technical and fundamental investors. Later on, reveals the use of statistical methods, machine and deep learning for modelling price dynamics. Chapter 3 - Theoretical Background, provides the necessary theoretical background to conceive the proposed algorithms. It begins with the introduction to deep learning and reveals the underlying mechanisms of artificial neural networks. Moreover, the chapter discusses the possible

problems and solutions connected to developing deep learning systems. Chapter 4 - Implementation, addresses used technologies and goes through the architecture design process. Lastly, Chapter 5 - Last Thoughts, summarizes the study outcomes, gives further work propositions and draws to the discussion.

Chapter 2

Related Work

2.1 Price Predictability

Some claim that the markets perform a random walk [8] and the prices can not be predicted. It is impossible to profit from the stock markets with repetitive performance. Burton Malkiel, a Professor of Economics at Princeton University, argued in his book [26] that the stock price movement is as random as flipping a coin. However, there are professors and investors, who do not agree with the above hypothesis. Professors Andrew W. Lo and Archie Craig MacKinlay performed several experiments to prove the existence of price trends in stock markets with success [23]. What is more, Fama stated in his work in 1970 that stock prices are informationally efficient [25]. It means that the stock market prices can be predicted based on trading history. The information regarding a particular stock such as profitability, public image or investors attitude is reflected in the price. Therefore, if the information obtained from stock prices is pre-processed efficiently and appropriate algorithms are applied, the trend may be predicted. Furthermore, Tshilidzi Marwala inferred that the more artificial intelligence-based transactions, the more efficient the markets become [27]. The reasoning stems from the fact that computers confine emotional and unreasonable behaviour of the human trader. Thus, making the stock price more predictable.

2.2 Statistical and Machine Learning Models

Statistical models such as ARMA [20], ARCH [38], GARCH [22] were universally employed in financial time series modelling. However, machine learning methodologies provide a superior capability to discover relationships between features and labels. Furthermore, they determine model parameters by fitting training samples. These technics can not only be used for trend prediction but also to take appropriate trading decisions.

Support Vector Machine (SVM) [9, 24, 31], Naive Bayesian (NB) [9, 31], Artificial Neural Network (ANN) [10, 14, 31], Random Forest (RF) [9, 24, 31] are commonly applied algorithms. They do not recognize sequential patterns. Therefore, technical analysis methodologies (see table 2.1) are used for feature extraction as it allows to recognize important characteristics of price dynamics (see figure 2.1).

Name of indicator	**Formula**
SMA: Simple n-day Moving Average	$\frac{C_t + C_{t-1} + \ldots + C_{t-(n-1)}}{n}$
EMA: Weighted n-day Moving Average	$\frac{nC_t + (n-1)C_{t-1} + \ldots + C_{t-(n-1)}}{n + (n-1) + \ldots + 1}$
MOM: Momentum	$C_t - C_{t-(n-1)}$
STCK: Stochastic K%	$\frac{C_t - LL_{t-(n-1)}}{HH_{t-(n-1)} - LL_{t-(n-1)}} * 100$
STCD: Stochastic D%	$\frac{\sum_{i=0}^{n-1} K_{t-i}}{10}\%$
RSI: Relative Strength Index	$100 - \frac{100}{1 + (\sum_{i=0}^{n-1} UP_{t-1}/n)/(\sum_{i=0}^{n-1} DW_{t-1}/n)}$
MACD: Moving Average Convergence Divergence	$MACD(n)_{t-1} + \frac{2}{n+1} * (DIFF_t - MACD(n)_{t-1})$
LW: Larry William's R%	$\frac{H_n - C_t}{H_n - L_n} * 100$
A/D: Accumulation/Distribiution Oscilator	$\frac{H_t - C_{t-1}}{H_t - L_t}$
CCI: Commodity Channel Index	$\frac{M_t - SM_t}{0.015 D_t}$

Table 2.1: Popular technical indicators [18]

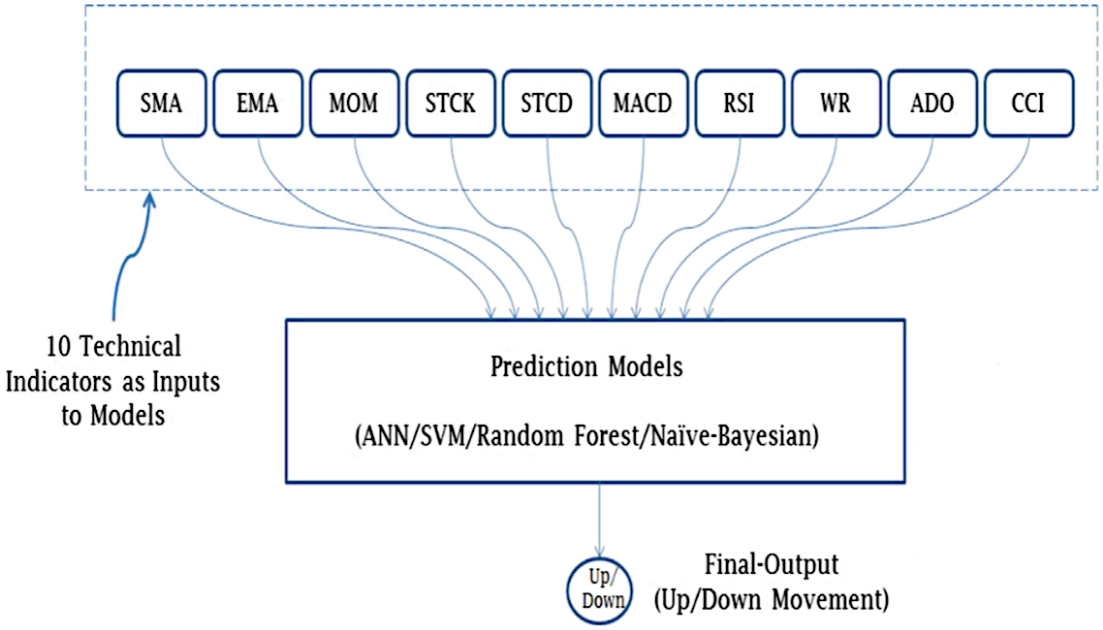

Figure 2.1: Technical indicators as features [9]

Further studies on feature selection provided by Rajashree and Pradipta Dash [9] foud that binary trend signal is beneficial for use with machine learning algorithms (see figure 2.2). They used technical analysis strategies based on technical indicators to determine whether the price would rise or fall.

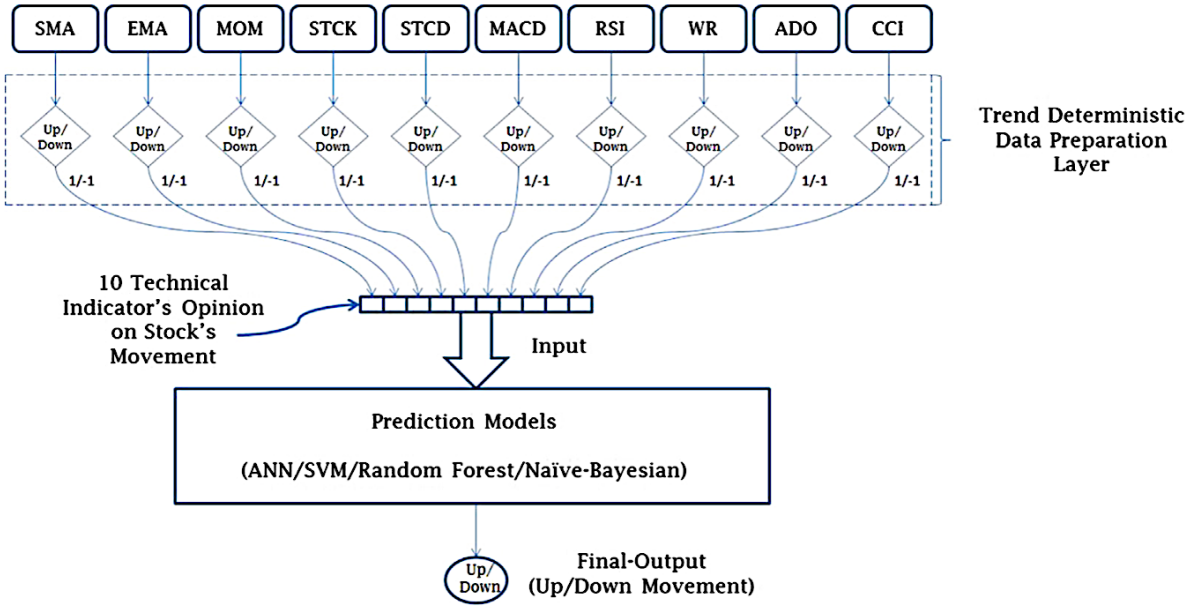

Figure 2.2: Trend signals from technical indicators as features [9]

Rajashree's and Pradipta's proposed method achieved accuracy of 90% in movement prediction for next day closing price. Nonetheless, they evaluated their algorithm only on

two stock indices which is not enough to draw a general conclusion. What is more, the use of the same index for training and validation may lead to false results. It is because training and validation examples might be similar to a great extent. Furthermore, the data did not have a uniform distribution of price rises and falls. Taking everything into consideration the result should be taken with a pinch of salt.

Another paper was published proposing a two-stage approach for stock price forecasting based on technical analysis [31]. That time, the authors used Support Vector Regression (SVR) to predict future values of technical indicators which were passed as features to the stock price prediction algorithm (see figure 2.3).

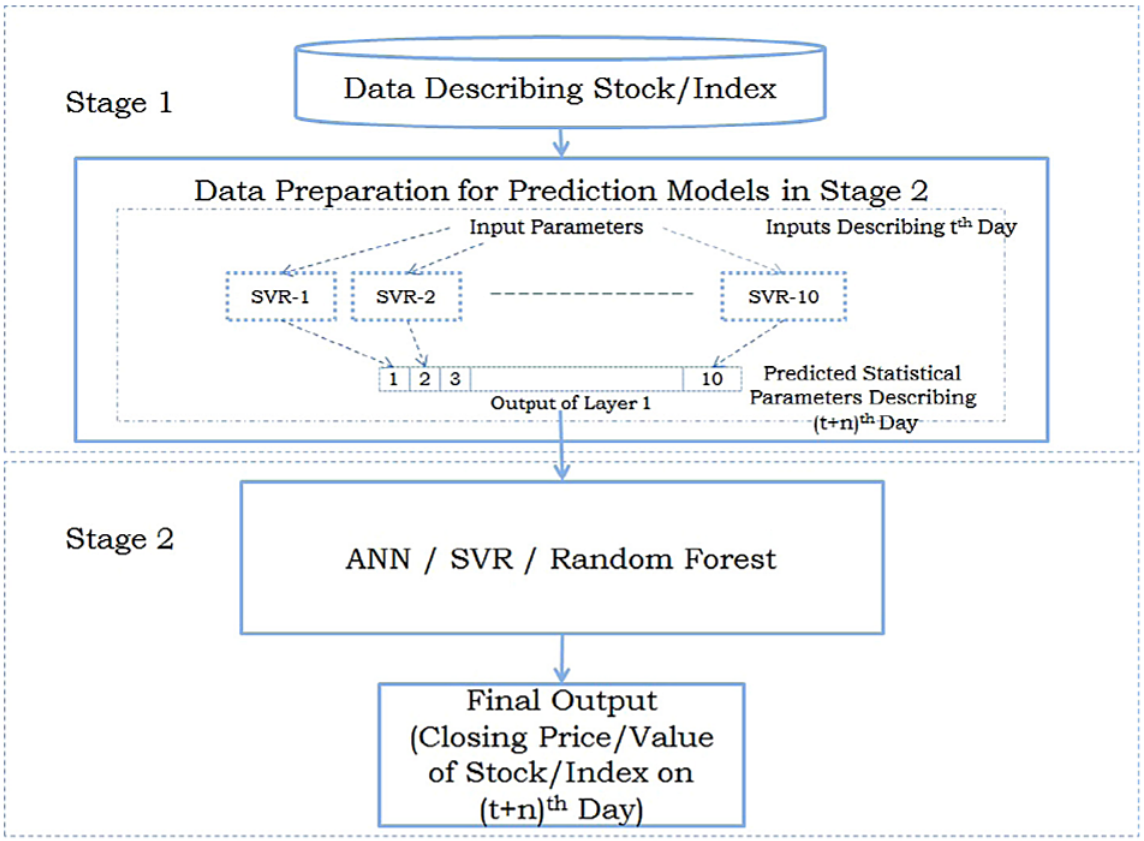

Figure 2.3: Two stage price prediction algorithm [31]

Their method proved to be beneficial for more than 4th-day price prediction. Yet again, the algorithm was trained and evaluated only on two indices which raises the previously mentioned concerns.

2.3 Deep Learning Models

With the development of computational power and data availability, deep learning techniques dominated the financial data modelling segment. Based on deep learning achieve-

ments in the field of speech recognition [13] and generally time series, these models may provide ground-breaking performance in stock price movement prediction as well as price forecasting. They incorporate architectures based on Recurrent Neural Networks (RNN) [7] for the ability to recognize time dependencies. Nonetheless, RNN faced the problem of the vanishing gradient. That is, they lose information with the length of a sequence. To resolve the issue Long Short-Term Memory (LSTM) cells were introduced. They include memory and gated units to retain previous values and prioritize learning from more current data in a sequence. LSTM based architechtures were shown to be more effective than conventional RNNs [3]. For that reason, LSTM networks are often used for stock price movement prediction [1, 2, 6, 10, 11, 28, 33]. Furthermore, LSTMs do not require hand-crafted features, such as technical indicators. If enough data is given, suitable data-dependent patterns are automatically detected [21]. Nonetheless, calculating the relative change of the price history data resulted in faster learning process [11]. It allows the neural network to be more sensitive to the change in price. That kind of normalization enables the algorithm to fit the wider range of input values. Furthermore, studies have shown that denoising the data with the exponentially weighted moving average is beneficial for further predictions into the future [28].

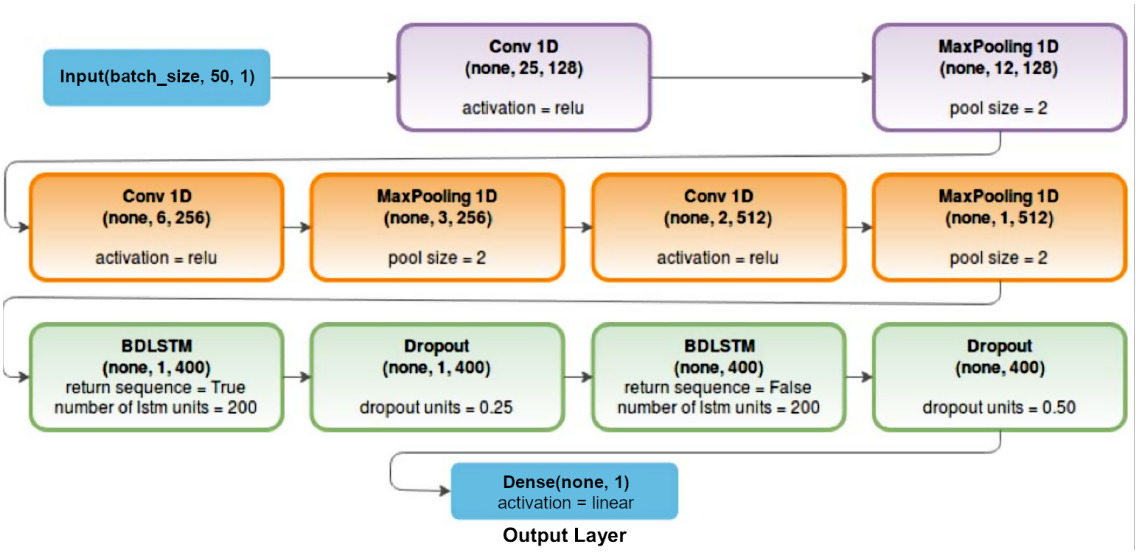

Figure 2.4: Hybrid model [11]

There were also experiments carried out with convolutional neural networks (CNN) [2, 12, 14, 24, 33]. However, the most important factor is data preparation. Therefore, technical indicators [12] or wavelets trasformations [19] are applied. There were also proposed hybrid models with one dimensional CNN and bidirectional LSTM [11] where convolutional layers applied filters to obtain relevant features. Then the bi-LSTM recognized time dependencies (see figure 2.4). Further studies showed that multi-pipeline

architecture resulted in the trend prediction accuracy increase by 9% compared to the single-pipeline model.

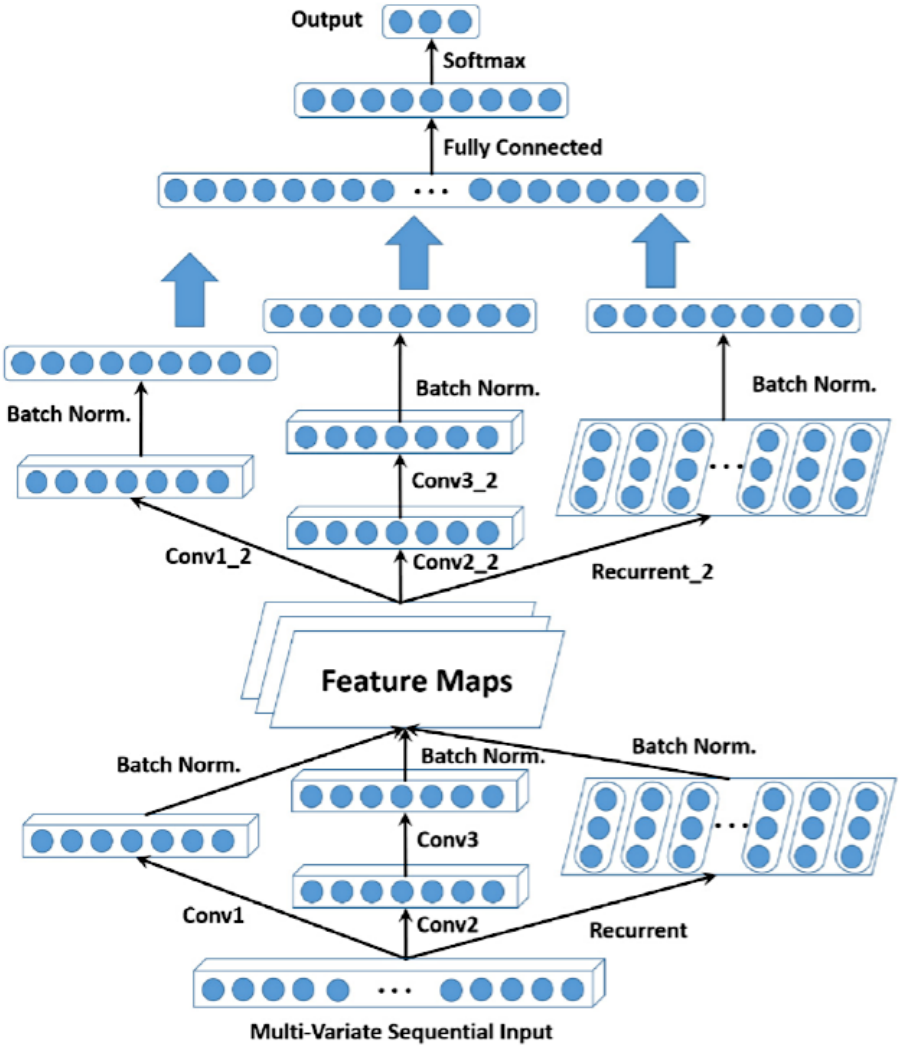

Figure 2.5: Muti-filters neural network [24]

Different study took the idea of deep feature extraction even further. The researchers used deep learning model to create feature maps that were later fed into another multi-pipeline network (see figure 2.6). The proposed solution achieved 7% better accuracy than standard recurrent neural networks [24].

The presented results ranged from 55.6% [28] to 72% [12] in price movement prediction. However, it is tough to compare the studies as the outcome depends on the training time, amount of data and validation dataset. Furthermore, most of the studies focused only on several stocks or indexes. For these reasons, it is difficult to draw general conclusions.

Chapter 3

Theoretical Background

3.1 What is Deep Learning?

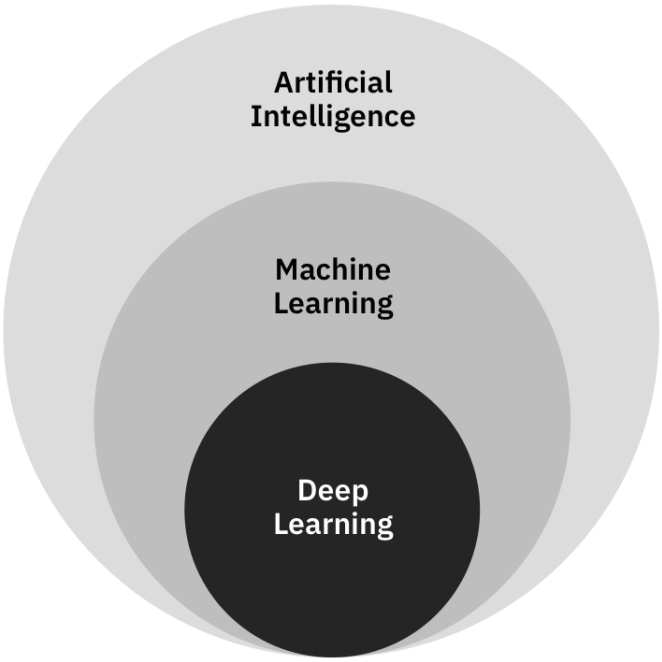

Figure 3.1: The relationship between artificial intelligence, machine learning, and deep learning

Artificial intelligence is the science of creating machines that take actions on their own. Machine learning uses algorithms that learn the solution by finding patterns in data without being explicitly programmed. Finally, Deep learning employs multilayer artificial neural

networks inspired by the human brain (see figure 3.1, 3.2).

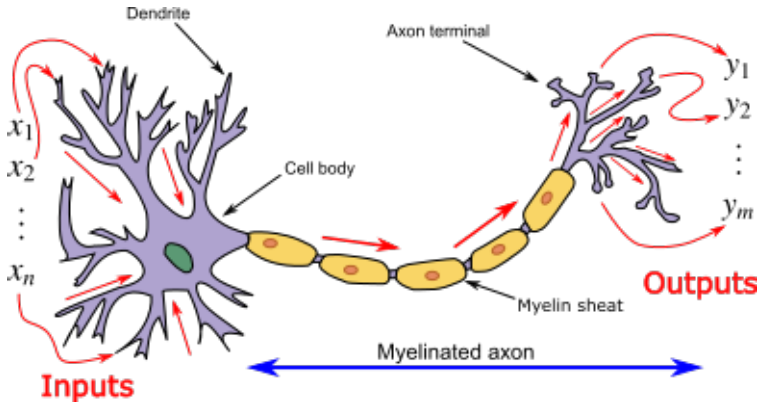

Figure 3.2: Neuron [37]

There is evidence that the architecture of the brain is homogeneous among different regions and uses only one "learning algorithm". There were experiments on ferrets, where scientists cut the connection between the ears and the auditory cortex and rewired the optical nerve to it [36]. With time, the part of the brain responsible for hearing learned to interpret signals from visual receptors and ferrets could see again. This principle is called "neuroplasticity".

First multilayer artificial neural networks were introduced in the 1960s. They are the building blocks of modern deep learning. The word deep corresponds to the numerous layers in the network, wherewith deeper layers the algorithm extracts higher-level features. The feed-forward neural network can be addressed as deep when it has more that one hidden layer. These architectures are known to learn better features than shallower ones and can benefit from raw data [21, 29]. Nevertheless, deep learning techniques require a substantial amount of data as well as computing power to learn. For that reason, there were very few applications for deep learning until the 21st century. Nowadays, deep learning is one of the fastest developing fields of computer science. Its applications range from financial forecasting [11], medicine [34], speech recognition [13] to self-driving cars [5].

3.2 Artificial Neural Network

This chapter is based on a book "Neural Networks and Deep Learning" by Michael Nielsen [29].

3.2.1 Perceptron

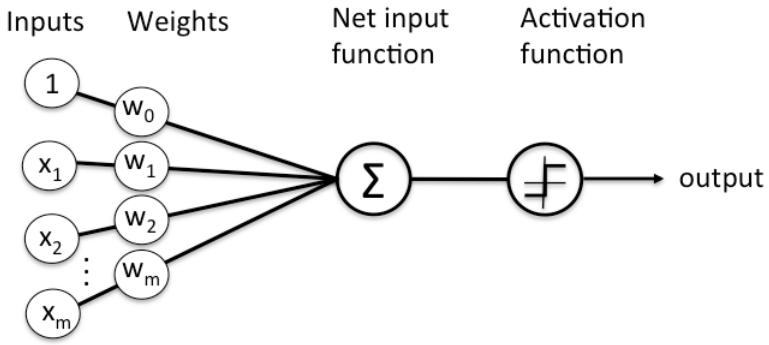

Figure 3.3: Single perceptron

The most elementary unit of an artificial neural network is an artificial neuron or in other words, a perceptron (see figure 3.3). It takes inputs and multiplies them by corresponding weights. Then adds bias and applies an activation function (see figure 3.4).

$$output = activation(\sum(weights * inputs) + bias) \tag{3.1}$$

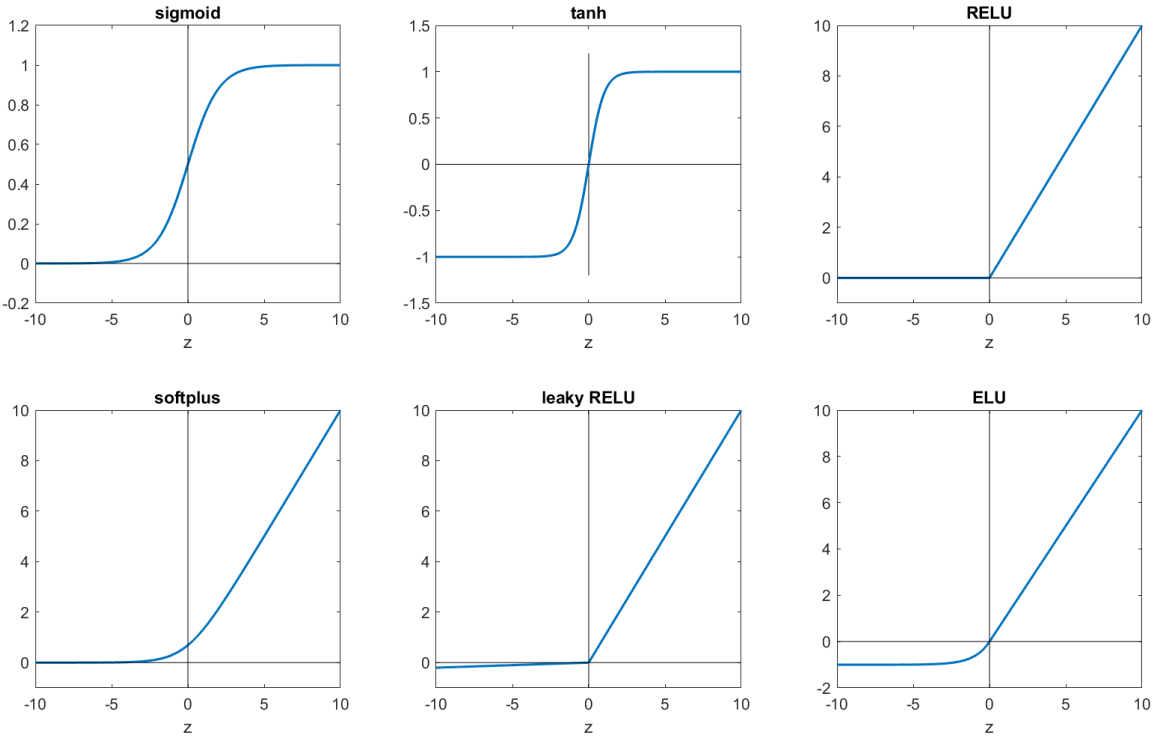

Figure 3.4: Types of activation functions

Types of activation functions:

$$\textbf{sigmoid: } y = \frac{1}{1-e^{-z}} \tag{3.2}$$

$$\textbf{tanh: } y = \tanh(z) \tag{3.3}$$

$$\textbf{RELU: } y = \max(0,z) \tag{3.4}$$

$$\textbf{softplus: } y = \log(1+e^{z}) \tag{3.5}$$

$$\textbf{leaky RELU: } y = \begin{cases} z, & z > 0 \\ 0.01z & z \leq 0 \end{cases} \tag{3.6}$$

$$\textbf{ELU: } y = \begin{cases} z, & z > 0 \\ \alpha(e^{z}-1), & z \leq 0 \end{cases} \tag{3.7}$$

3.2.2 Deep Neural Network

A deep neural network consists of many perceptrons arranged in layers (see figure 3.5). This architecture mimics the functionality of the human brain. At a fundamental level, neurons are computational units that take electrical signals called "spikes" as inputs. Then they are channelled to outputs (axons), see figure 3.2.

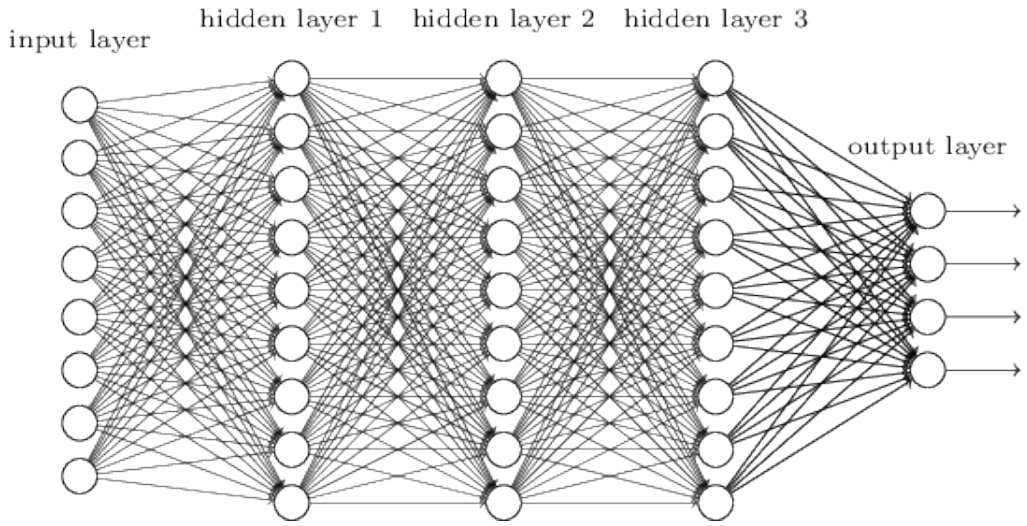

Figure 3.5: Multilayer perceptron (deep neural network)

The feed-forward algorithm for one hidden layer artificial neural network with one

output $h_\omega(x)$ - a hypothesis function.

$$\begin{aligned}
a_1^{(2)} &= g(\omega_{10}^{(1)}x_0 + \omega_{11}^{(1)}x_1 + \omega_{12}^{(1)}x_2 + \ldots + \omega_{1m}^{(1)}x_m) \\
a_2^{(2)} &= g(\omega_{20}^{(1)}x_0 + \omega_{21}^{(1)}x_1 + \omega_{22}^{(1)}x_2 + \ldots + \omega_{2m}^{(1)}x_m) \\
&\vdots \\
a_n^{(2)} &= g(\omega_{30}^{(1)}x_0 + \omega_{31}^{(1)}x_1 + \omega_{32}^{(1)}x_2 + \ldots + \omega_{1m}^{(3)}x_m)
\end{aligned} \tag{3.8}$$

$$h_\omega(x) = a_1^{(3)} = g(\omega_{10}^{(2)}a_0^{(2)} + \omega_{11}^{(2)}a_1^{(2)} + \omega_{12}^{(2)}a_2^{(2)} + \ldots + \omega_{1n}^{(2)}a_n^{(2)}) \tag{3.9}$$

where

$a_i^{(j)}$ – "activation" of unit i in layer j, $a_0^{(j)} = 1$ (bias)

$g()$ – activation function

x_k – input k, $x_0 = 1$(bias)

$\omega^{(j)}$ – matrix of weights controlling function mapping from layer j to layer $j+1$

In order to create a deep neural network, the above algorithm should be performed for each neuron in a multilayer architecture. Below is a simplistic representation of a function mapping.

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ \cdots \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ a_2^{(3)} \\ \cdots \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} h_\omega(x)_1 \\ h_\omega(x)_2 \\ h_\omega(x)_3 \\ \cdots \end{bmatrix} \tag{3.10}$$

3.3 Learning

People are punished for bad behaviour and showed propper practice to correct their actions. Neural networks utilize a similar approach which is called supervised learning. A cost function is a metric of how poorly the network performs. The backpropagation algorithm reasons from the network's actions to enhance its performance. It fine-tunes the weighs by minimizing the cost function.

3.3.1 Cost Function

Cost function is evaluated based on how good did the network performed its task. To do that, it requires the predicted output and the corresponding true values. Cathegorical cross-entropy loss function is used for classification (see figure 3.6). It represents the

average logarithmic error. The simplified version for binary classification is represented below.

$$J(\omega) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\omega(x^{(i)}), y^{(i)}) \tag{3.11}$$

$$\text{Cost}(h_\omega(x), y) = -\log(h_\omega(x)) \qquad \text{if y = 1} \tag{3.12}$$

$$\text{Cost}(h_\omega(x), y) = -\log(1 - h_\omega(x)) \qquad \text{if y = 0} \tag{3.13}$$

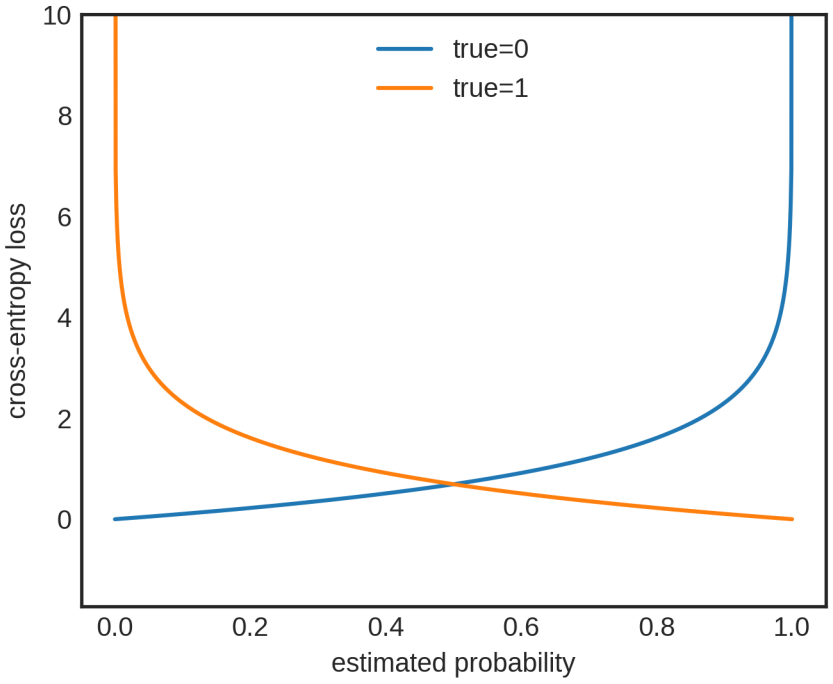

Figure 3.6: Categorical cross-entropy loss function plot

From the previously mentioned equations and figure 3.6 arrive obvious conclusions.

$$\text{Cost}(h_\omega(x), y) = 0 \text{ if } h_\omega(x) = y \tag{3.14}$$

$$\text{Cost}(h_\omega(x), y) \to \infty \text{ if } y = 0 \text{ and } h_\omega(x) \to 1 \tag{3.15}$$

$$\text{Cost}(h_\omega(x), y) \to \infty \text{ if } y = 1 \text{ and } h_\omega(x) \to 0 \tag{3.16}$$

Categorical cross-entropy function can also be written in one line.

$$J(\omega) = -\frac{1}{m}\sum_{i=1}^{m} [y^{(i)} \log(h_\omega(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\omega(x^{(i)})] \tag{3.17}$$

It is worth to notice that when y is equal to 0, then the first term $h_\omega(x^{(i)}))$ is zero and has no effect on the result. Similarly, when y is equal to 1, the second term $(1 - y^{(i)}) log(1 - h_\omega(x^{(i)})$ is zero and has no effect on the result.

For deep neural networks, the propper cross-entropy function is a bit more complicated.

$$J(\omega) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\omega(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\omega(x^{(i)}))_k)\right] + \\ + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\omega_{j,i}^{(l)})^2 \tag{3.18}$$

where

L – total number of layers in the network

s_l – number of units (not counting bias unit) in layer l

K – number of output units/classes

λ – regularization parameter

The λ parameter minimizes the values of weights. It is important not to overfit the data, which will be discussed later.

3.3.2 Backpropagation Algorithm

The subsection is based on Sebastian Rudner's paper "An overview of gradient descent optimization algorithms" [32].

The goal of backpropagation is to minimize the cost function in the parameter space.

$$\min_{\omega} J(\omega) \tag{3.19}$$

Gradient Descent

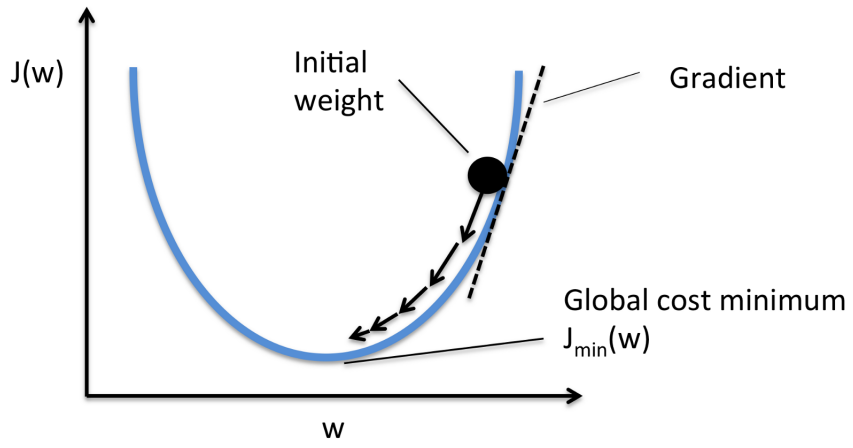

Figure 3.7: Gradient descent

The most popular algorithm for backpropagation is gradient descent. The algorithm calculates partial derivatives of a cost function for each weight. It "shows" in which direction to move in the parameter space. Then it subtracts the derivatives from the corresponding weights and takes the step (see figure 3.7). Furthermore, the learning rate α determines how "big" these steps are. The pseudo-code of gradient descent is shown below.

$$\begin{aligned} &Repeat\ \{ \\ &\omega = \omega - \alpha \frac{\partial}{\partial \omega} J(\omega) \\ &\} \end{aligned} \tag{3.20}$$

To properly calculate the partial derivatives, the weights should be updated simultaneously. There is no point in showing the derived formula because it differs based on activation and loss functions. Nevertheless, it is worth to mention that the derivatives are calculated using the chain rule. That way, it is possible to determine how sensitive is the output for each weight.

Learning Rate

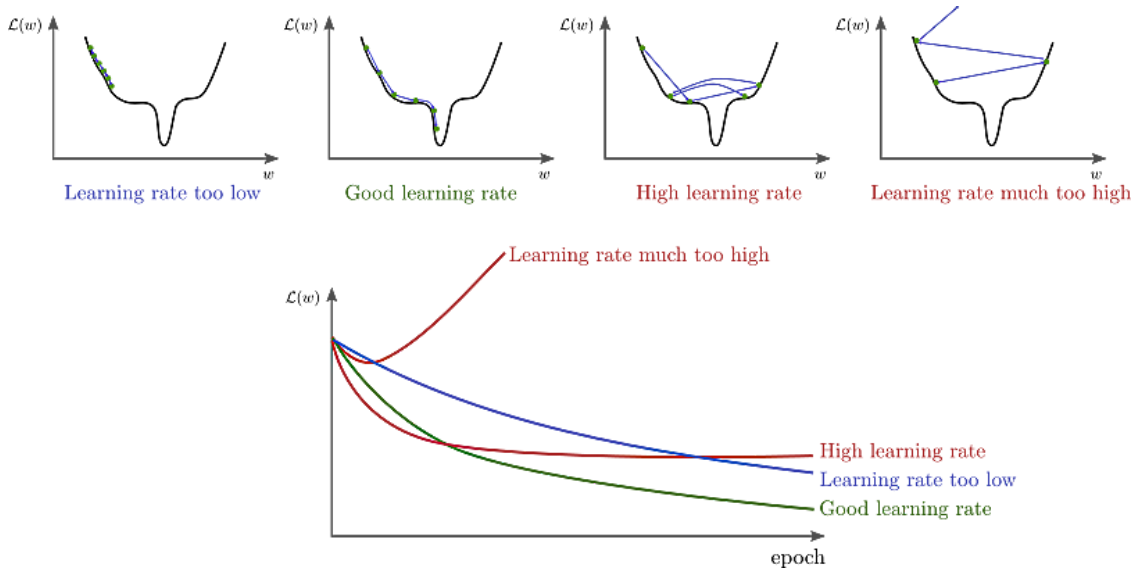

Figure 3.8: Learning rate choice

The most important parameter to set for gradient descent algorithm is the learning rate. The proper setup of α is essential for the network's good performance. If the learning rate is too big, it would be impossible to find the minimum and even cause the algorithm to diverge. On the other hand, too small learning rate slows down the process of learning and may find the local instead of the global minimum (see figure 3.8).

Stochastic and Batch Gradient Descent

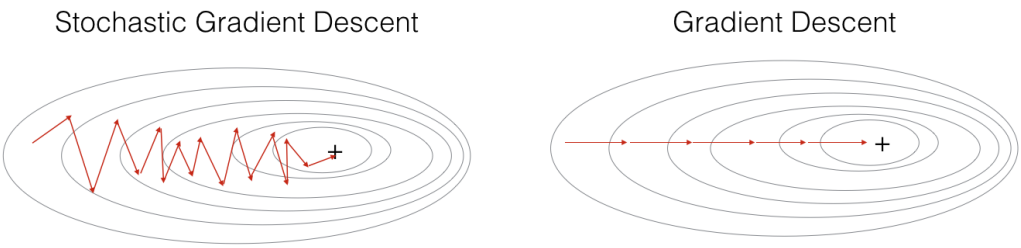

Figure 3.9: Comparison of gradient descent variations

The gradient descent algorithm requires extensive computing power to consider the whole data space in each iteration. However, the stochastic gradient descent takes a randomly chosen data point to calculate the appropriate step. It may minimize a cost function in more iterations. Nevertheless, each step requires less time. To further improve the algorithm, batch gradient descent was introduced. Instead of considering only one data point at each step, it accounts for several data points grouped in batches. That approach is superior for larger data sets with more complicated parameter spaces (see figure 3.9).

Adaptive Moment Estimation

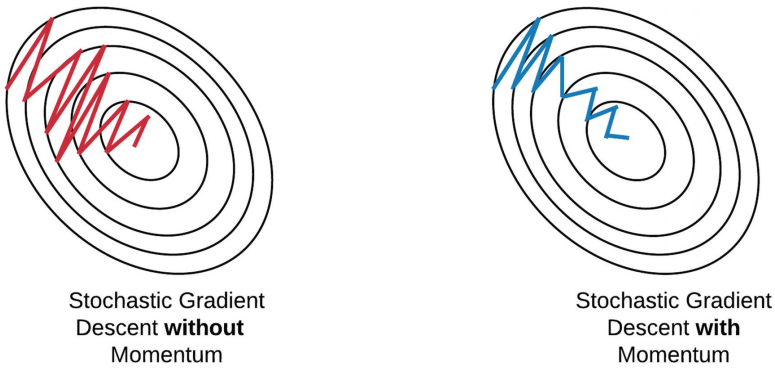

Figure 3.10: Gradient descent with and without momentum

To speed up the learning process the gradient descent with momentum was proposed. It incorporates the first moment of a gradient, which is considered a random variable. It dampens the oscillations of the stochastic gradient descent (see figure 3.10). For that reason, a higher learning rate can be used to speed up the learning process. The exponentially weighted moving average m_t is used to estimate the expected value of the gradient g_t.

$$m_t = \beta m_{t-1} + (1-\beta) g_t \tag{3.21}$$

$$\hat{m}_t = E[g_t] = \frac{m_t}{1-\beta} \tag{3.22}$$

where

β – new hyper-parameter

Stochastic gradient descent with momentum weight update.

$$\omega_t = \omega_{t-1} - \alpha \hat{m}_{t-1} \tag{3.23}$$

Stochastic gradient descent with momentum has one downside. It is similar mathematically to a ball rolling down a hill. It gains inertia and is hard to stop. Therefore, it might overshoot a minimum. Another algorithm was proposed - RMSprop. It relies on the second moment v of the gradient g. It can scale the learning rate to decrease the step as the slope flattens.

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \tag{3.24}$$

$$\hat{v}_t = E[g_t^2] = \frac{v_t}{1-\beta} \tag{3.25}$$

RMSprop weight update.

$$\omega_t = \omega_{t-1} - \alpha \frac{g_t}{\sqrt{\hat{v}_{t-1} + \varepsilon}} \tag{3.26}$$

where

ε – value to prevent division by 0

Nowadays, the most popular optimization algorithm in deep learning is Adam - adaptive moment estimation. It combines both the stochastic gradient descent with momentum and RMSprop. Adam has the benefits of inertia and scaling the step to prevent overshooting.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{3.27}$$

$$\hat{m}_t = E[g_t] = \frac{m_t}{1-\beta_1^t} \tag{3.28}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{3.29}$$

$$\hat{v}_t = E[g_t^2] = \frac{v_t}{1-\beta_2^t} \tag{3.30}$$

Adam algorithm weight update.

$$\omega_t = \omega_{t-1} - \alpha \frac{\hat{m}_{t-1}}{\sqrt{\hat{v}_{t-1} + \varepsilon}} \tag{3.31}$$

3.4 Recurrent Neural Networks

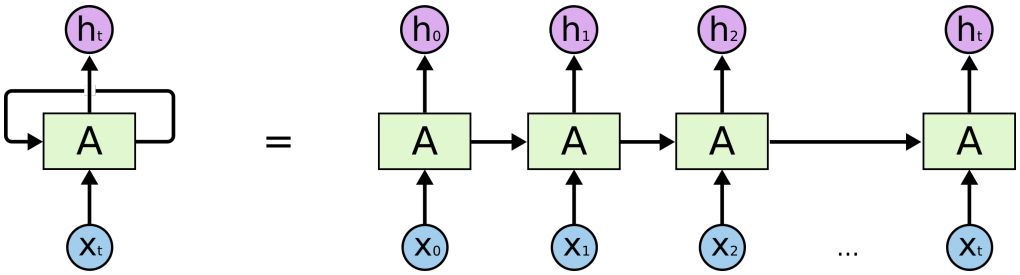

Figure 3.11: Unfolded RNN

In a feed-forward neural network, all inputs and outputs are assumed to be independent of each other. It does not account for time-dependent data. For that reason, RNNs are to make use of sequential information. Recurrent neural networks have a feedback loop so that the output depends on the previous and current input. If unfolded like in figure 3.11 they can be regarded as deep feed-forward neural networks with the same weights throughout all layers [21].

$$h_t = \tanh(\omega_h h_{t-1} + \omega_x x_t) \tag{3.32}$$

$$y_t = \omega_y h_t \tag{3.33}$$

where

x_t – input state

y_t – output state

h_t – current timestamp

h_{t-1} – previous timestamp

ω_h – weight assiosiated with the previous timestamp

ω_x – weight assiosiated with the current input

ω_y – weight assiosiated with the output

Backpropagation Through Time

The learning process differs for recurrent neural networks. The algorithm is called back-propagation through time (BPTT), where the RNN is unfolded and treated as a multi-layer feed-forward neural network [30]. The weights are the same for each timestamp in the sequence. The algorithm has to go through each element in the sequence to update the weights. For that reason, the total error is the sum of the errors at each time step. Below is the overview of the BPTT.

1. Get the error
2. Unroll the network
3. Compute the gradient for each time step
4. Combine the gradients
5. Update the weights

The biggest downfall of recurrent neural networks is that they are computationally demanding. Repetitive weight update for every timestamp is a slow process. What is more, while backpropagating through time, there might occur two problems.

- Exploding gradient - the algorithm assigns inapropriatly high importance to some weights. Using optimization techniques with adaptive learning rate such as RMSprop or Adam can counter that effect.
- Vanishing gradient - contribution from the earlier steps becomes insignificant in the gradient descent step. It can be prevented using the LSTM cell.

3.4.1 Long Short-Term Memory

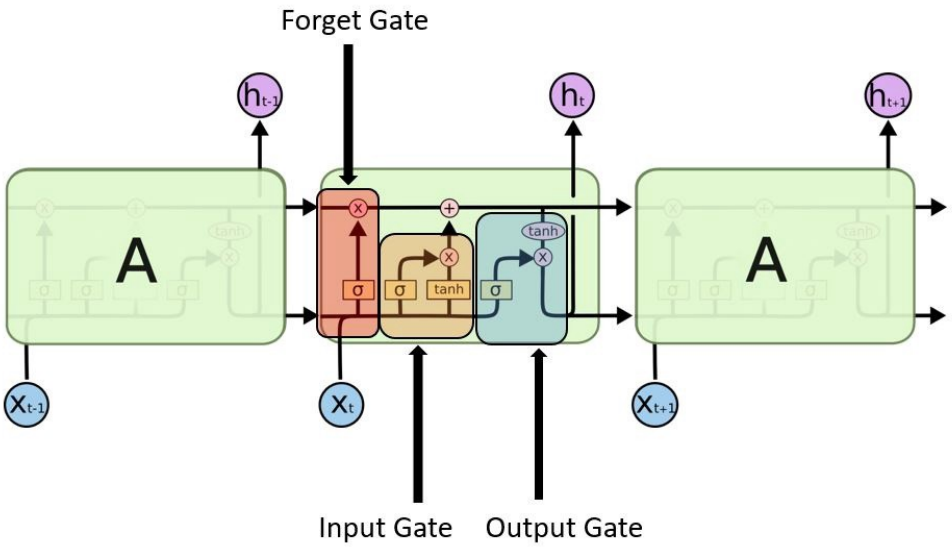

Figure 3.12: LSTM gates

Long short-term memory or LSTM in short, as its name suggests, accounts for long time dependencies. Every LSTM cell consists of three gates, forget gate, input gate and output

gate (see figure 3.12). That architecture creates a memory able to store broad information without decay.

Forget Gate

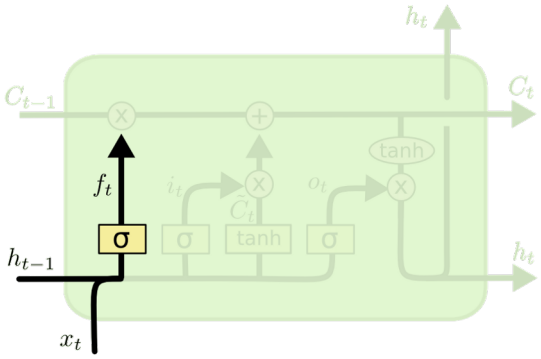

Figure 3.13: LSTM forget gate

$$f_t = \sigma(\omega_f[h_{t-1}, x_t] + b_f) \tag{3.34}$$

The forget gate (see figure 3.13) decides which information should be kept or omitted. It takes the previous hidden state h_{t-1} and current input x_t. Then by applying sigmoid function σ returns output f_t between 0 (forget) and 1 (keep). The output from the forget gate is multiplied with previous cell state C_{t-1} to drop irrelevant information.

Input Gate

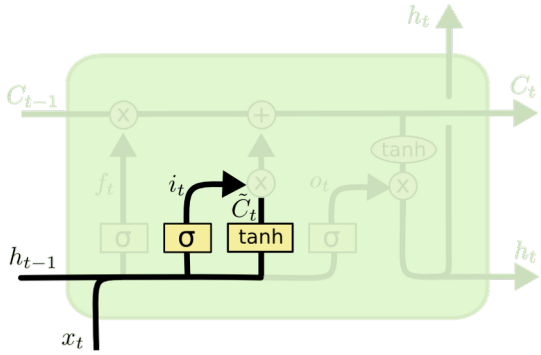

Figure 3.14: LSTM input gate

$$i_t = \sigma(\omega_i[h_{t-1}, x_t] + b_i) \tag{3.35}$$

$$\tilde{C}_t = \tanh(\omega_c[h_{t-1}, x_t] + b_C) \tag{3.36}$$

The input gate (see figure 3.14) updates the cell state C_t. The sigmoid function σ determines which values of previous hidden state h_{t-1} and current imput x_t will be updated by returning

the value i_t between 0 (do not update) and 1 (update). Then the *tanh* gives weightage to the passed values and returns the level of importance $\tilde{C}_t$ ranging from -1 to 1, which also helps to regulate the network. Then $\tilde{C}_t$ is multiplied by i_t which determines which outputs form *tanh* should be used to update the cell state C_t.

Output Gate

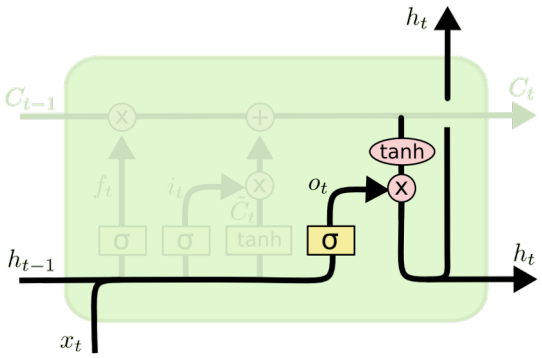

Figure 3.15: LSTM output gate

$$o_t = \sigma(\omega_o[h_{t-1}, x_t] + b_o) \tag{3.37}$$

$$h_t = o_t * \tanh(C_t) \tag{3.38}$$

Finally, the output gate (see figure 3.15) decides what the next hidden state should be. It applies sigmoid function σ to resolve which states to let through o_t and *tanh* regularizes the current cell state C_t. Multiplication of both outputs determines which information the hidden state should carry.

3.5 Hyperparameter Tuning

Hyperparameters are the paramaters that are set prior to the learning process. They are not derived via training.

3.5.1 Fitting The Data

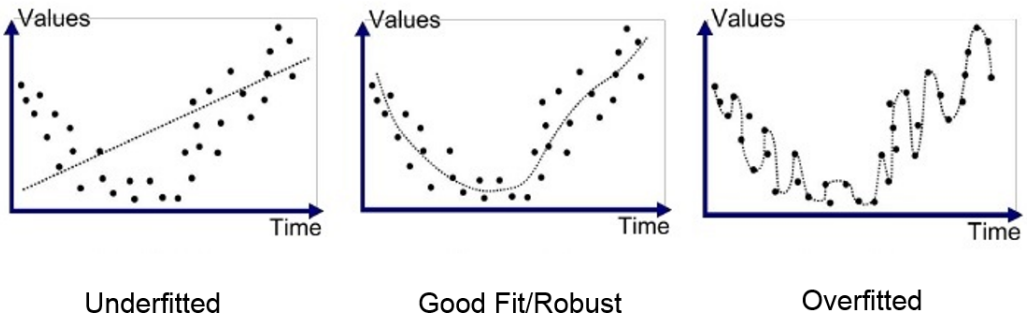

Figure 3.16: Problems regarding fitting the data

Hyperparameter optimization is an important task for a data scientist. It allows for enhancing the algorithm's performance. One of the main characteristics to consider is the network's complexity. If the model is over-complex, it might be prone to overfitting. On the other hand, if it is too simplistic, the network might not fit the data correctly, which results in underfitting (see figure 3.16, 3.17).

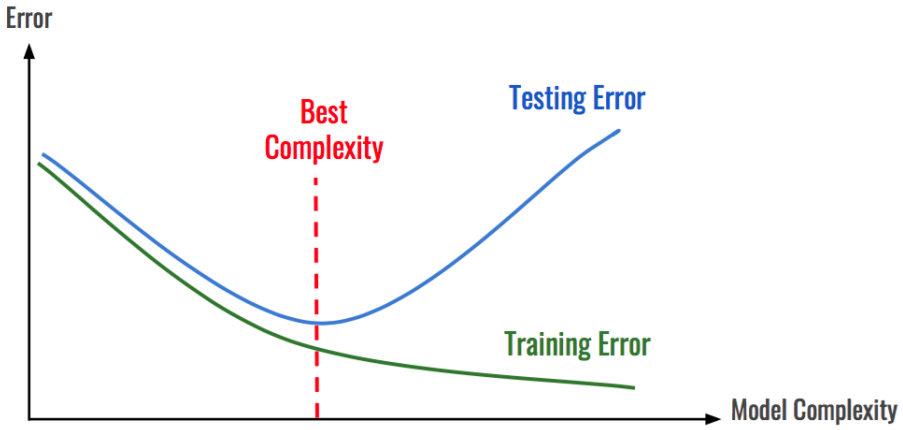

Figure 3.17: Model complexity

Nevertheless, there are regularization methods to prevent overfitting in complex architectures. One approach is to introduce dropout layers to the network. They randomly select neurons that are ignored during training iteration. For that reason, the network becomes less sensitive to the specific weights. That, in turn, results in a better generalization.

Another commonly added layer is batch normalization. The layer activations are often the inputs of the following layer. For that reason, the input distribution changes with each learning step. That forces intermediate layers to continuously adjust to the changing inputs. The Batch normalization layer normalizes the activations of each layer and limits covariate

shift [17]. That enables learning on a more stable distribution of inputs. Thus, accelerating the training of the network.

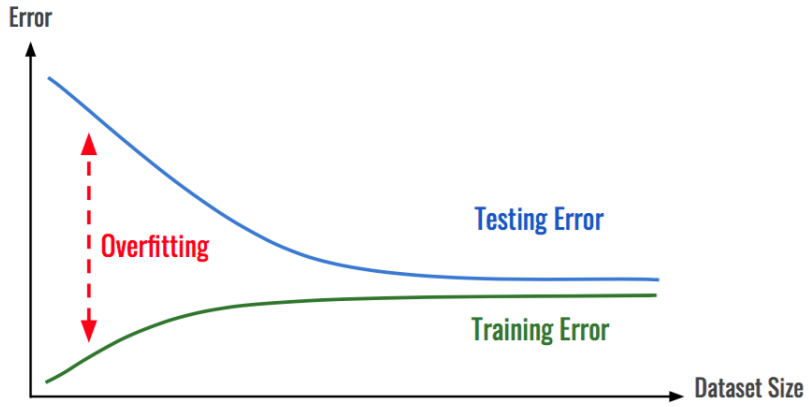

Figure 3.18: Dataset size

The available amount of data determines the level of complexity of the model (see figure 3.18). Too few sample points may not allow to properly tune the numerous algorithm parameters. Therefore, to better approximate the objective function, the relevant amount of data is required.

There are several approaches to find the model hyperparameters that yield the best score on the validation set metric.

- Manual
- Grid search
- Random search
- Sequential model-based optimization (SMBO)

Manual search for optimal hyperparameters can be time-consuming and intractable in complex search spaces. Grid search might benefit from an automated approach. Nevertheless, the search space is often high-dimensional, and it is not possible to try every parameter combination in a reasonable time. The random search may acquire good results in low dimensional search spaces. However, these methods are relatively inefficient because they do not choose the next hyperparameters to evaluate based on previous results. Grid and random search are uninformed by past evaluations, and as a result, often spend a significant amount of time evaluating not promising hyperparameters. Therefore, to find an optimal solution for complex models, SMBO is the usual choice.

3.5.2 Sequential Model-Based Optimization

Sequential model-based optimization can be summarised in one sentence. Create a probability model ("surrogate") of the objective function to find the most promising hyperparameters to test on the true objective function. It is less computationally consuming to optimize the surrogate function than running the neural network. What is more, these methods take into account the previous outcomes, so they perform an informed search. Thus, the increased performance.

$$P(score|hyperparametrs) \tag{3.39}$$

The algorithm outline:

1. Create a surrogate probability model of the true objective function.
2. Find the best performing hyperparameters on the surrogate.
3. Test the true objective function with the found hyperparameters.
4. Update the surrogate model with the new results.
5. Repeat steps 2-4 until optimization constraints are reached.

Expected Improvement

The SMBO algorithms try to maximize expected improvement EI_{y^*} with respect to the given set of proposed hyperparameters x [4].

$$EI_{y*}(x) := \int_{-\infty}^{\infty} \max(y^* - y, 0) p(y|x) dy \tag{3.40}$$

where

$y -$ score

$y^* -$ threshold score

$x -$ set of hyperparameters

$p(y|x) -$ surrogate - probability of y given x

Tree-structured Parzen Estimator

The tree-structured Parzen estimator (TPE) surrogate function does not model $p(y|x)$ directly. Instead, it models $p(x|y)$ as well as $p(y)$ and applies Bayes rule.

$$p(y|x) = \frac{p(x|y) * p(y)}{p(x)} \tag{3.41}$$

TPE algorithm [4]:

1. Create search space - define distributions describing hyperparameters.

2. Perform a random search to initialize the algorithm.

3. Split the observations into two groups based on y^* threshold: good ($y < y^*$) and bad ($y \geq y^*$).

4. Model the $p(x|y)$ as two density functions $l(x)$ and $g(x)$ based on belonging to a good or bad group:

$$p(x|y) = \begin{cases} l(x), & y < y^* \\ g(x), & y \geq y^* \end{cases} \tag{3.42}$$

   The density functions are modelled by Parzen estimators also known as kernel density estimators.

5. Model $p(y)$ so that:

$$p(y < y*) = \gamma \tag{3.43}$$

   where

   $\gamma -$ defines percentile split into two groups

6. Taking equations 3.41, 3.42, 3.43 and substituting them to equation 3.40 it can be shown that:

$$EI_{y*} \propto \frac{l(x)}{g(x)} \tag{3.44}$$

   Which means, to maximize the expected improvement the algorithm must maximize the ratio $\frac{l(x)}{g(x)}$.

7. Iterate in the loop updating the surrogate function, and optimizing expected value to find the best hyperparameters x^*.

Chapter 4

Implementation

4.1 Technology

The algorithm is implemented in Python programming language using a high-level neural network API - Keras on a TensorFlow backend. It allows for fast prototyping and research. Furthermore, the network is trained using cloud computing on Kaggle servers with the Nvidia Tesla P100 graphical processing unit (GPU).

4.2 Dataset

The data used is a Huge Stock Dataset uploaded by Boris Marjanovic with a CC0: Public Domain licence. It provides the full historical daily price and volume data for all US-based stocks and ETFs trading on the NYSE, NASDAQ, and NYSE MKT. The dataset is comprised of 7195 stocks history to the year 2017. The data is presented in a CSV format as follows: Date, Open, High, Low, Close, Volume, OpenInt (see figure 4.1). Additionally, the prices have been adjusted for dividends and splits.

	Date	Open	High	Low	Close	Volume	OpenInt
0	2004-08-19	50.000	52.03	47.980	50.170	44703800	0
1	2004-08-20	50.505	54.54	50.250	54.155	22857200	0
2	2004-08-23	55.375	56.74	54.525	54.700	18274400	0
3	2004-08-24	55.620	55.80	51.785	52.435	15262600	0
4	2004-08-25	52.480	54.00	51.940	53.000	9197800	0

Figure 4.1: Example of Google stock data

4.3 Network Architecture

The network is based on a sequential model. It has three CuDNNLSTM layers optimized for GPU computations, each followed by Dropout and Batch Normalization. Next is a Dense layer with a rectified linear unit (RELU) activation function followed by Dropout. Finally, there is a Dense layer with 2 neurons activated by Softmax function for the classification task (see figure 4.2).

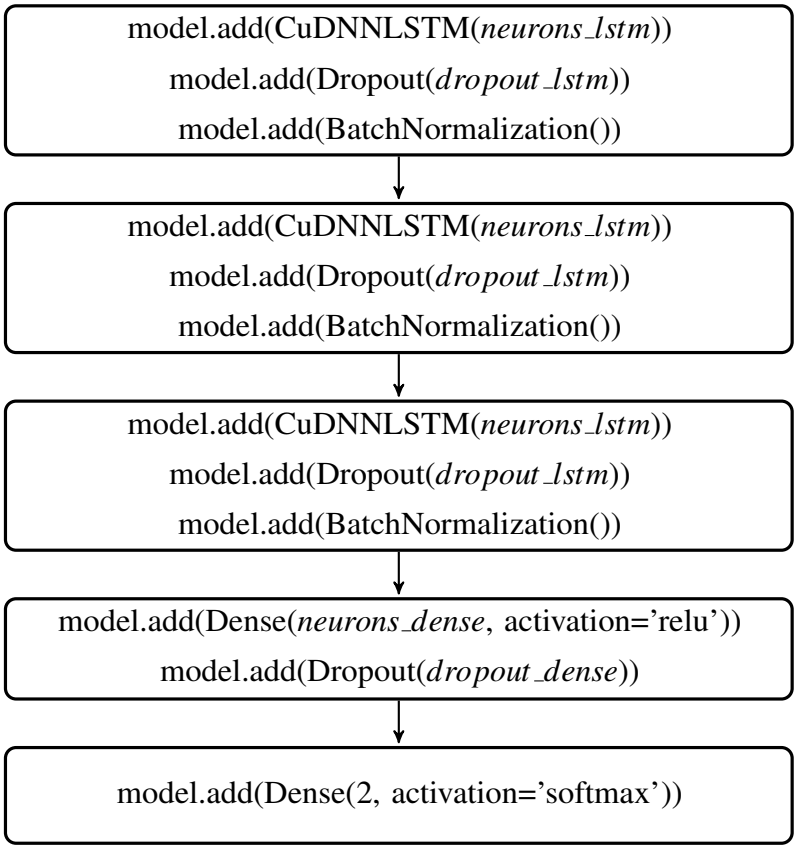

Figure 4.2: Model architecture

As the loss function, the sparse categorical cross-entropy was chosen. It is the alternative to the categorical cross-entropy loss function that allows passing labels as a one-dimensional category class to reduce memory usage. Moreover, Adam backpropagation algorithm was adopted for its leading performance.

The sequences of 60 days price and volume data were obtained using a sliding window approach. The data was scaled and normalized so that train and test sets have a uniform distribution of rises and falls of the third-day closing price. Furthermore, the percentage changes of the Open, High, Low, Close and Volume, were used as features. It is worth to mention that the train and test sets hold different stocks so that the validation process is honest.

Hyperparameter Optimization

The network was optimized using Hyperopt library. It supports sequential model-based optimization techniques. The algorithm used was the Tree-structured Parzen Estimator (TPE) surrogate function. The hyperparameters optimized were:

- $batch\_size$ - batch size
- $learning\_rate$ - learning rate
- $neurons\_lstm$ - number of neurons in LSTM layers
- $neurons\_dense$ - number of neurons in a Dense layer
- $dropout\_lstm$ - percentage of dropout units in LSTM layers
- $dropout\_dense$ - percentage of dropout units in a Dense layer

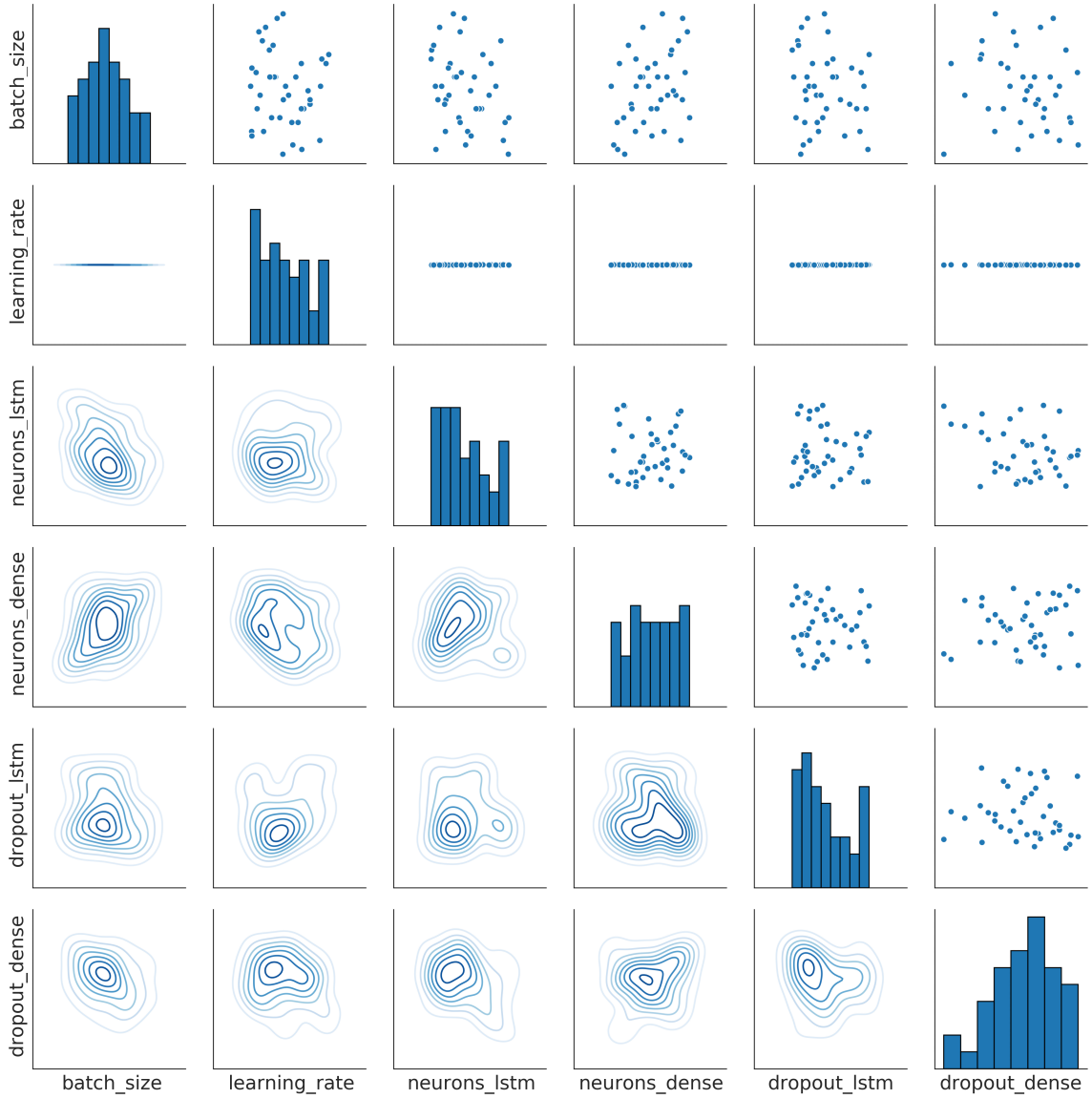

Figure 4.3: Optimization process (axis values have been omitted for better readability)

The optimization algorithm run in pursuit of a vector of hyperparameters values corresponding to the lowest validation loss. Figure 4.3 is a 2D representation of high-dimensional parameter space. It depicts the histogram of a single parameter sampling on the diagonal and relationships between parameters elsewhere. The upper part of the figure represents examined values for optimization, and the lower section describes a sapling density of hyperparameter space. The distributions are not uniform proving the informed character of the algorithm. The SMBO predicts where the possible optimal hyperparameters may remain in the parameter space. The algorithm did just 40 iterations that profited with high-grade results. Because of the high computational demands, the objective function was evaluated on one epoch on 500 training and 300 validation companies.

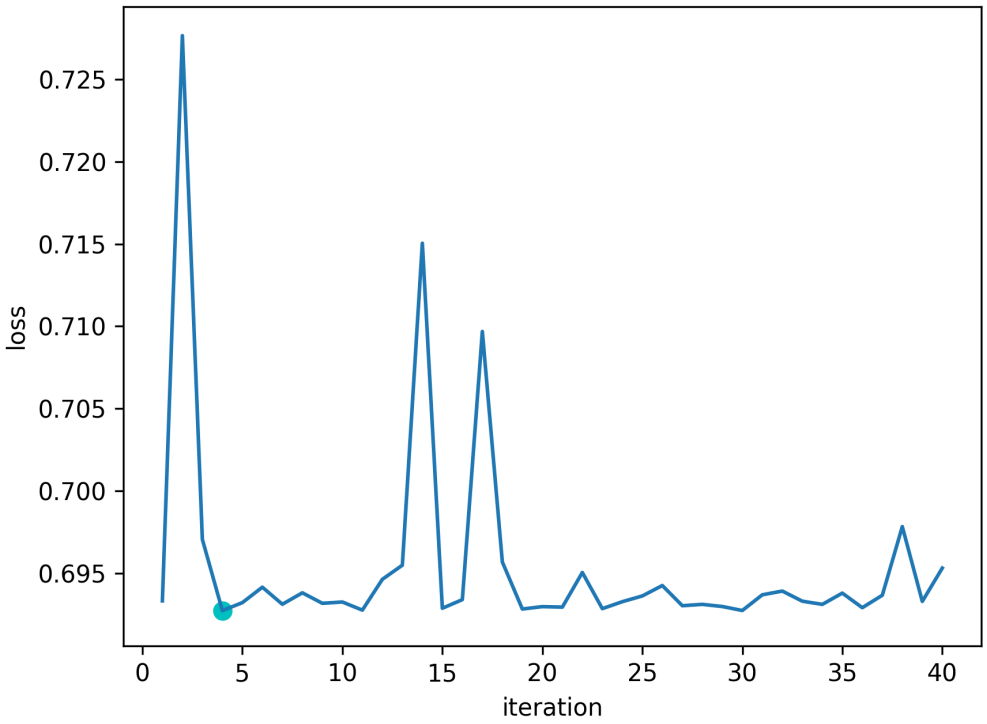

Figure 4.4: Convergence plot

In figure 4.4 the cyan dot corresponds to the lowest validation loss. The optimization algorithm found the best solution in the 4th iteration and further searched the parameter space. The best performing hyperparameters on the true objective function are presented in table 4.1.

Hyperparameter	**Value**
batch_size	45
learning_rate	4.4e-5
neurons_lstm	152
neurons_dense	231
dropout_lstm	0.17
dropout_dense	0.39

Table 4.1: Best hyperparameters found by SMBO

The obtained hyperparameters were implemented to establish the conclusive neural network architecture (see figure 4.5).

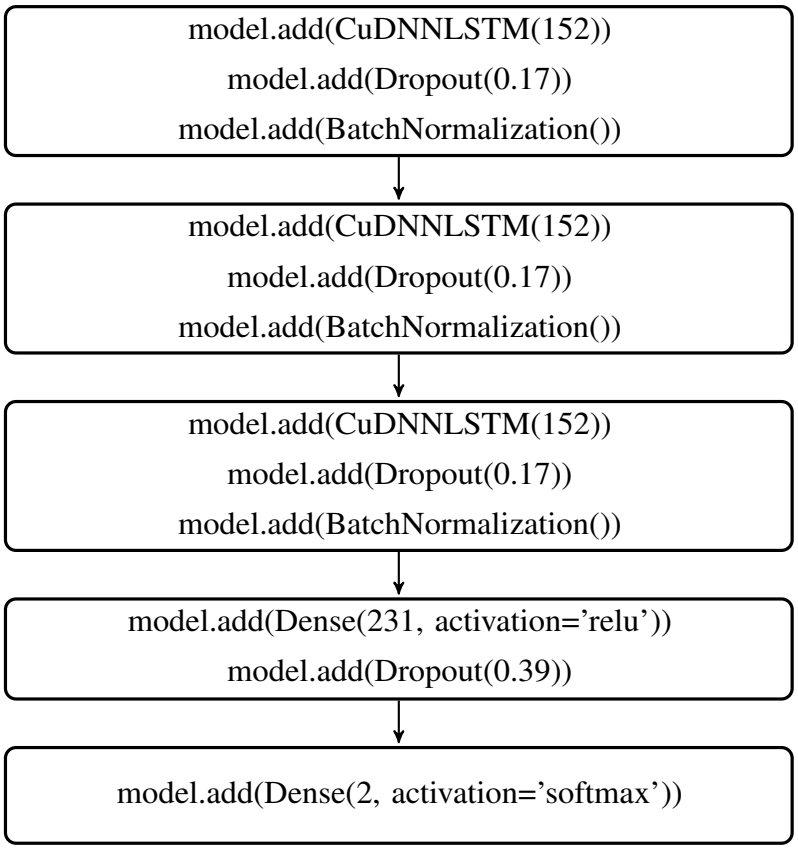

Figure 4.5: Final model architecture

Repeatability Test

To check, if the optimization process was indeed valid, the repeatability test was performed. The input and validation data, as well as the hyperparameters, were unchanged during the process. The purpose of that test was to establish whether the randomly initiated network weights, aka different models would provide repetitive performance. It would state that the optimization process found a local minimum. Furthermore, only 20 models were trained because of the computational complexity of the algorithm.

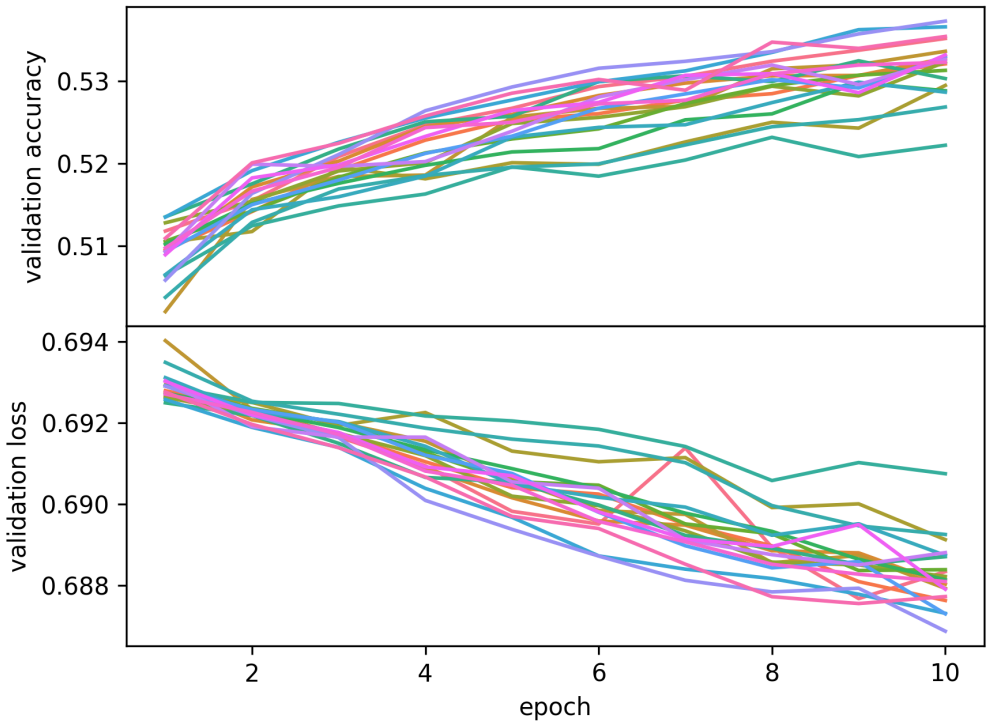

Figure 4.6: Repeatability test results

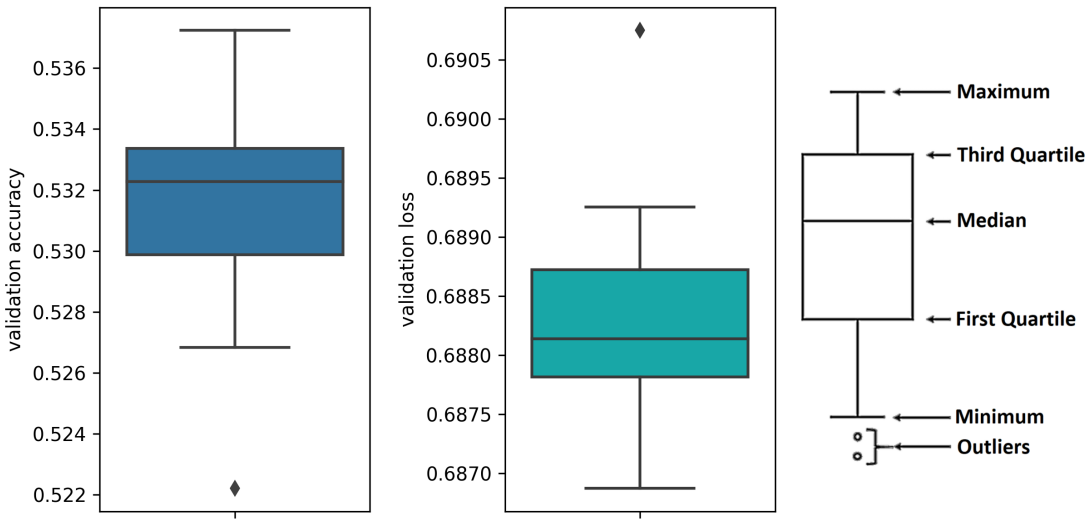

Figure 4.7: Repeatability test evaluation

In figures 4.6 and 4.7, it can be observed that most of the models had the validation accuracy between 53% and 53.35%, which allows saying that the optimization process was valid. However, to determine the truely optimal hyperparameters, the SMBO should be performed on final training parameters for more iterations, and the hyperparameters

should be thoroughly checked with repeatability tests. Unfortunately, it is not possible in this case.

4.4 Feature Engineering

Feature Engineering has a considerable impact on the performance of the machine learning algorithm. In the case of deep learning, the network should be able to extract relevant features on its own. However, moderate data preparation might speed up the learning process. For that reason, several different feature sets were considered.

1. Raw Price History and Volume - scaled and normalized Low, Open, High, Close and Volume.

2. Pct. Change of Full Price History and Volume - calculated percentage change to emphasize the importance of change.

3. Pct. Change of Closing Price and Volume - the Principle Component Analysis (PCA) showed that the Closing price holds 99% of variance. Thus, it is worth to check if dropping Low, Open, High has a meaningful impact on accuracy.

4. Pct. Change of Denoized Full Price History and Volume - denoized with 10 periods exponentially weighted moving average (EWMA).

5. Technical Indicators - often proposed method for stock market prediction with machine learning models. Obtained 10 popular indicators, see figure 2.1.

Feature Set	**Accuracy**	**Std**
Raw Price History and Volume	51.37%	0.12%
Pct. Change of Full Price History and Volume	**53.44%**	0.11%
Pct. Change of Closing Price and Volume	52.74%	0.13%
Pct. Change of Denoized Full Price History and Volume	52.73%	0.14%
Technical Indicators	52.36%	0.13%

Table 4.2: Accuracy of trend prediction based on feature choice

For each feature set, a model was trained on 500 stocks for 10 epochs. Unfortunately, the algorithm is time-consuming (almost 2 hours to train one model) and it was not possible to evaluate several networks for each feature set. However, to obtain more relevant results, each model was validated on 10 sets comprised of different 300 companies. Table 4.2 represents the mean accuracy of validation sets and corresponding standard deviations.

Calculating the percentage change of the price history and volume significantly increased model accuracy from 51.37% to 53.44%, which confirms that the model is change sensitive. Dropping Open, High, and Low price history decreased performance even though they contain little information. From that fact, we can reason that the price movement during each day holds relevant information making candlestick analysis a promising approach for short-term price forecasting. What is more, denoising the price data did not provide better performance for a 3-day trend prediction. The information loss associated with applying EWMA lessened prediction accuracy. Moreover, the results are comparable within standard deviation to using only Close and Volume features. It might stem from the fact that denoising resulted in variance loss between Open, High, Close, Low. Lastly, obtaining technical indicators may be advantageous for shallow machine learning models. Unfortunately, connected information loss decreased deep architecture's performance.

From the conducted tests, we can observe that any information loss has a relevant impact on model performance. It concludes that future and previous prices are weakly correlated. Therefore, every piece of information has great importance to the prediction accuracy. Moreover, calculating the percentage change of full price history and volume provided the best results.

4.5 Influence of Prediction and History Horizon

To test the influence of prediction and history horizon, the training data was acquired using moving windows of widths 20, 30 and 60 days. Furthermore, for each of the sequence lengths, the model was trained for different prediction targets: 1, 3, 6 and 10 days ahead. The network was trained on 500 stocks for 10 epochs and evaluated on another 3000 companies divided into 10 sets.

History Horizon	**Future Horizon**							
	1 day		**3 days**		**6 days**		**10 days**	
	Accuracy	**Std**	**Accuracy**	**Std**	**Accuracy**	**Std**	**Accuracy**	**Std**
20 days	52.92%	0.14%	**53.23%**	0.21%	52.84%	0.10%	52.55%	0.19%
30 days	52.94%	0.15%	53.32%	0.15%	**53.43%**	0.19%	52.98%	0.22%
60 days	53.05%	0.20%	53.40%	0.26%	**53.86%**	0.16%	53.48%	0.23%

Table 4.3: Prediction accuracy based on sequence length and n-day prediction into the future

From table 4.3 stems that the longer the history horizon, the higher the prediction accuracy. Furthermore, reasoning from the data there exists the optimal into the future prediction horizon of approximately 10-20% of the history horizon. That drives to the conclusion, the shorter the prediction horizon, the more unpredictable the price behaviour.

What is more, a decrease in validation accuracy for longer than optimal future horizon can be the result of insufficient information stored in the too narrow history window. Taking into account standard deviations, the distributions slightly overlap. However, if we assume the normal distributions of the validation accuracies, the hypothesis is unlikely to be false. Nonetheless, further tests should be performed to account for the performance variability between models (see figure 4.7), and to narrow down the optimal horizon length.

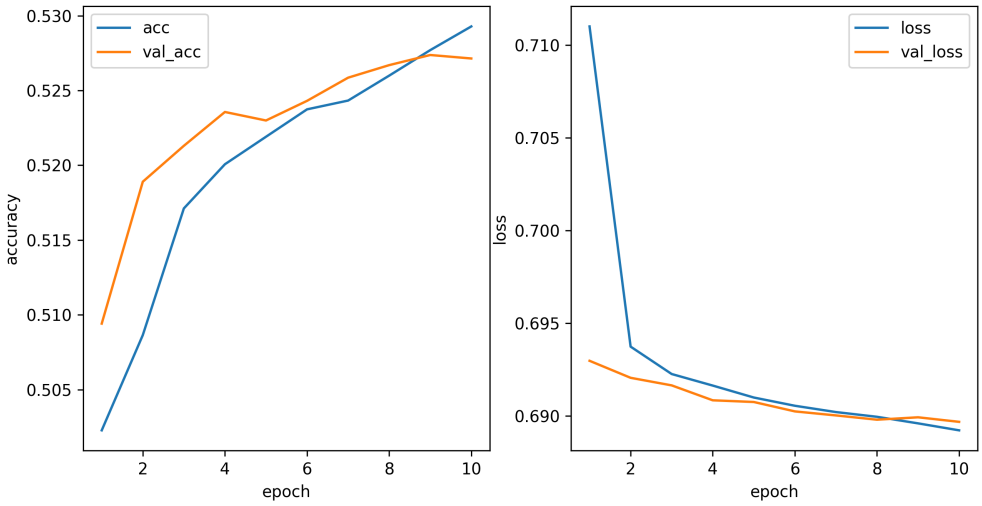

Figure 4.8: Training procedure for 30 day sequence length and 1 day prediction horizon

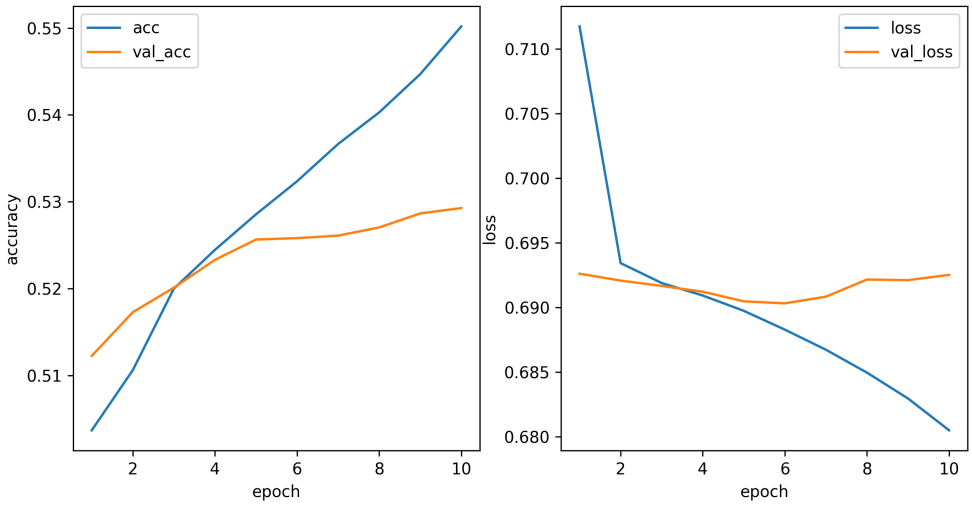

Figure 4.9: Training procedure for 30 day sequence length and 6 days prediction horizon

What is worth to mention, is that the longer the prediction horizon, the more prone to overfitting the network. Figures 4.8, 4.9 depict the training procedure for the 30-day

history horizon. The only difference between the two learning processes are the labels. It is visible that for the 6 days into the future occurred a high-variance problem (overfitting) opposed to 1-day prediction horizon.

4.6 Final Performance

To check how accurately we can predict a short time price movement, the proposed model was trained on 90% of the entire dataset (approx. 6.4k stocks) and tested on the remaining companies (700). However, the training process required a different approach than before. The data did not fit in RAM. There are several popular approaches to that problem:

- Data generator - it is used to prepare a single batch for the network training. However, getting and preprocessing the data may often take longer than the one iteration of the algorithm. For that reason, not properly parallelized and optimized data generators can result in substantially increased training time.
- Batches read from binary files - there exist file formats such as TFRecord that allow for dataset storage as a sequence of binary records. It allows the algorithm to get a single batch from the file without loading the whole dataset to the RAM. Nonetheless, Kaggle does not allow storage of files greater than 4.9GB. Therefore, the preprocessed data does not fit in the plausible size.

To deal with the problem the following training algorithm was proposed:

Algorithm 1 Network training procedure

```
start_days = range(0, step)
for epoch in range(0, num_epochs) do
    shuffle(start_days)
    for start_day in start_days do
        inputs, labels = preprocess(stocks, start_day, step)
        model_fit_once(inputs, labels)
    end for
end for
```

The main thought behind the above algorithm is to efficiently use the dataset for training. For that reason, the features from each stock are calculated using a moving window approach with a *step* different from one. That allows to create the *inputs* and *labels* that fit in the RAM whilst maintaining diversity in the training set. However, to obtain every possible sequence, the inner *for* loop was designed. The *start_day* corresponds to the initial window index to obtain sequences from a stock. If we collect

the sequences for $step = 2$ and $start\_day = 1$ we would have the sequence starting days such as $1, 3, 5, \ldots$. If we now take $start\_day = 2$, we would obtain sequence starting days such as $2, 4, 6, \ldots$. Taking into account both sets, we acquire every possible sequence which corresponds to 2 iterations of the inner loop. Furthermore, the list $start\_days$ is shuffled to maintain diversity between each iteration through training sets. It prevents the subsequent training set to contain the same sequences shifted just by one day.

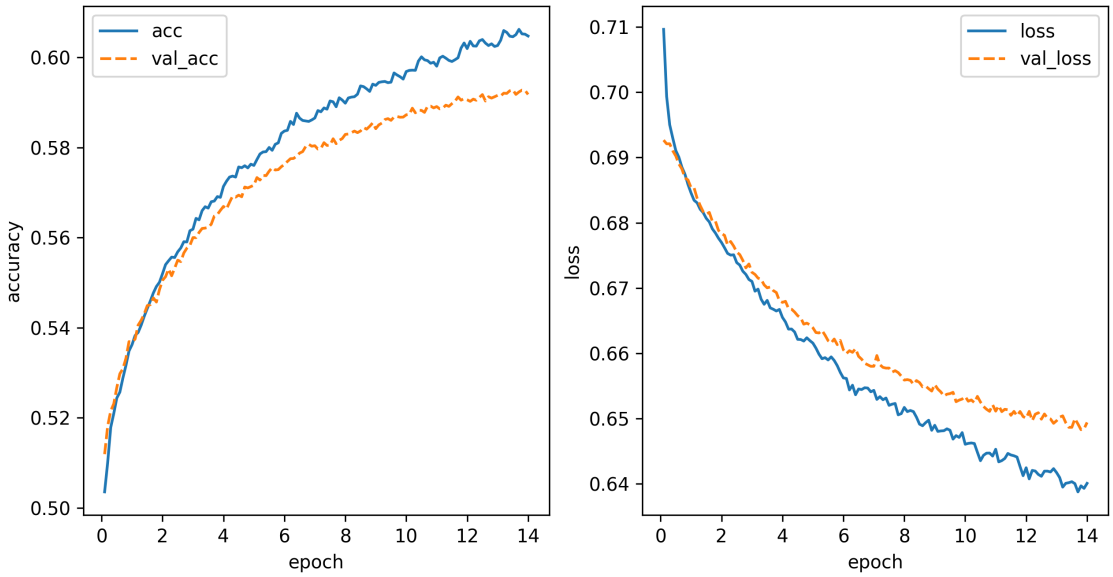

Figure 4.10: Final training

In figure 4.10, we can observe that the increase in validation accuracy begins to settle down. Therefore, further training is discontinued. The model achieved 59.3% accuracy in general stock price movement prediction. The results prove that price history carries information reflecting future behaviour. What is more, there exist common price dynamics. Thus, the same rules can be applied to predict the prices of various stock companies.

Chapter 5

Last Thoughts

5.1 Discussion

The proposed method achieved 59.3% in general stock price movement prediction for 3-days into the future window based on 60 days of historical technical data. Comparing the results to the related work utilizing LSTM networks, the algorithm exceeded 55.6% accuracy [28]. It might stem from the fact that the final training benefited from a large dataset or thorough hyperparameter tuning. However, some studies achieved the accuracy of 72% in price movement prediction [12] employing different architectures. For that reason, trying various models incorporating convolutional layers, hybrid or multi-pipeline architectures could benefit from better performance. What is worth to keep in mind is that the achieved results are strongly dependent on the dataset and training time. Moreover, higher accuracies might be the result of the dishonest validation process and improper data normalization. For example, if we train the network on a bearish trend where 70% of the time the price raises, then the model would achieve 70% accuracy by always predicting an increase in price value. For that reason, obtaining a uniform distribution of price rises and falls is crucial. Another aspect that could decrease the performance of the model is the changing behaviour of the stock market through the years. The dataset consists of the full history of stock companies to the year 2017. The investors might have made buy, sell or hold decisions differently. Over the years, stock markets became more dynamic. Increase in internet speed allowed for high-speed transactions, tools for technical analysis became more available, and computer algorithms could exchange stocks in milliseconds. Thus, price dynamics in the 2010s could be very different than in the 1990s.

The undertaken research proved that there exists a correlation between future price behaviour and its history. However promising the achieved 59.3% accuracy is, the model could not be used alone for stock trading. To develop a working strategy, the knowledge of price movement is not enough. It is necessary to be able to manage the risk of a

transaction. Explaining, we would have to know what is the probable profit and loss with the corresponding success probability. For that reason, the model can be seen as an auxiliary tool rather than a trading system. The solution might be a multi-class classification problem of predicting levels of potential movement, maybe as a Fibonacci Sequence such as $0-1, 1-2, 2-3, 3-5, 5-8, \ldots$ % raise or fall. Another approach would be solving a regression problem - forecasting a price value. However, these problems would be much harder than just price movement prediction. Furthermore, the algorithm's performance should be tested on different trends, price variabilities to know the strengths and weaknesses of the method. What is more, a portfolio selection algorithm would be required to pick the most promising stocks to trade.

As mentioned before, the model's performance could be enhanced by trying different layers and architectures. Another way may be training the network on different time frames (intra-day, intra-hour). It would not only provide more training samples but also allow to check whether the same rules apply for different time frames as states the technical analysis. Furthermore, retraining the model before using it on a particular stock might also enhance its performance. The pre-trained model reflecting the general price dynamics could allow fitting to the particular stock market company for better predictions, which would be otherwise impossible because of the training data shortage. Additionally, the model should be re-trained once a year to adjust to the changing behaviour of the stock exchange. Besides, training the model on NYSE, Nasdaq and NYSE MKT markets does not guarantee that it would perform similarly on other stock exchanges.

5.2 Summary and Conclusion

During this study, deep LSTM neural model was developed to check whether there exist general stock price dynamics. Firstly, a sequential model-based optimization technique with tree-structured Parzen estimator performed hyperparameter tuning. Later, the repeatability test proved that the optimization process was viable and gave insight into the algorithm performance variability. Following, feature engineering allowed to choose the most promising data preprocessing operations to enhance the learning of the artificial neural network. Further research included the study of history and future horizon length influence on price movement forcasting. Lastly, the final model was trained and tested on the whole dataset incorporating data from over 7k stock companies.

The findings are listed below:

- proposed method achieved 59.3% accuracy for general 3-day price movement prediction,

- there exist general price dynamics,
- future prices are correlated with historical technical data,
- LSTM networks provide satisfactory results for stock market forecasting,
- percentage change of price history as features yielded the best results for LSTM network,
- any information loss during data preprocessing has a negative influence on the algorithm performance,
- technical indicators as well as candlesitck formations can yield positive results,
- there exists an optimal future horizon length which is approximately 10-20% of history horizon,

5.3 Further Study Opportunities

Further work could extend this study beyond technical analysis. It would be advisable to include fundamental features as well as news reports as inputs. What is more, trying different models such as GRU, Attention networks, CNNs, hybrid and multi-pipeline approaches could lead to better results. Moreover, opposed to binary classification such as UP or Down, it might be worth to introduce levels of potential movement - strong/weak up, strong/weak down. Another angle of interest is a regression problem - price value prediction.

Bibliography

[1] Yujin Baek and Ha Young Kim. "ModAugNet: A new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module". In: *Expert Systems with Applications* 113 (2018), pp. 457–480.

[2] A Jayanth Balaji, DS Harish Ram, and Binoy B Nair. "Applicability of deep learning models for stock price forecasting an empirical study on Bankex data". In: *Procedia computer science* 143 (2018), pp. 947–953.

[3] Wei Bao, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory". In: *PloS one* 12.7 (2017), e0180944.

[4] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*. 2011, pp. 2546–2554.

[5] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[6] Kai Chen, Yi Zhou, and Fangyan Dai. "A LSTM-based method for stock returns prediction: A case study of China stock market". In: *2015 IEEE International Conference on Big Data (Big Data)*. IEEE. 2015, pp. 2823–2824.

[7] Jerome T Connor, R Douglas Martin, and Les E Atlas. "Recurrent neural networks and robust time series prediction". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 240–254.

[8] Paul H Cootner. *The random character of stock market prices*. MIT press, 1967.

[9] Rajashree Dash and Pradipta Kishore Dash. "A hybrid stock trading framework integrating technical analysis with machine learning techniques". In: *The Journal of Finance and Data Science* 2.1 (2016), pp. 42–57.

[10] Luca Di Persio and Oleksandr Honchar. "Artificial neural networks architectures for stock price prediction: Comparisons and applications". In: *International journal of circuits, systems and signal processing* 10 (2016), pp. 403–413.

[11] Jithin Eapen, Doina Bein, and Abhishek Verma. “Novel Deep Learning Model with CNN and Bi-Directional LSTM for Improved Stock Market Index Prediction”. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2019, pp. 0264–0270.

[12] M Ugur Gudelek, S Arda Boluk, and A Murat Ozbayoglu. “A deep learning based stock trading model with 2-D CNN trend detection”. In: *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2017, pp. 1–8.

[13] Geoffrey Hinton et al. “Deep neural networks for acoustic modeling in speech recognition”. In: *IEEE Signal processing magazine* 29 (2012).

[14] M Hiransha et al. “NSE stock market prediction using deep-learning models”. In: *Procedia computer science* 132 (2018), pp. 1351–1362.

[15] Arvid OI Hoffmann and Hersh Shefrin. “Technical analysis and individual investors”. In: *Journal of Economic Behavior & Organization* 107 (2014), pp. 487–511.

[16] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[17] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[18] Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. “Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange”. In: *Expert systems with Applications* 38.5 (2011), pp. 5311–5319.

[19] Yoshinori Kishikawa and Shozo Tokinaga. “Prediction of stock trends by using the wavelet transform and the multi-stage fuzzy inference system optimized by the GA”. In: *IEICE transactions on fundamentals of electronics, communications and computer sciences* 83.2 (2000), pp. 357–366.

[20] Cem Kocak. “ARMA (p, q) type high order fuzzy time series forecast method based on fuzzy logic relations”. In: *Applied Soft Computing* 58 (2017), pp. 92–103.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[22] Zhe Lin. “Modelling and forecasting the stock market volatility of SSE Composite Index using GARCH models”. In: *Future Generation Computer Systems* 79 (2018), pp. 960–972.

[23] Andrew W Lo and A Craig MacKinlay. *A non-random walk down Wall Street*. Princeton University Press, 2002.

[24] Wen Long, Zhichen Lu, and Lingxiao Cui. "Deep learning-based feature engineering for stock price movement prediction". In: *Knowledge-Based Systems* 164 (2019), pp. 163–173.

[25] Burton G Malkiel and Eugene F Fama. "Efficient capital markets: A review of theory and empirical work". In: *The journal of Finance* 25.2 (1970), pp. 383–417.

[26] Burton Gordon Malkiel. *A random walk down Wall Street: including a life-cycle guide to personal investing*. WW Norton & Company, 1999.

[27] Tshilidzi Marwala and Evan Hurwitz. *Artificial intelligence and economic theory: skynet in the market*. Springer, 2017.

[28] David MQ Nelson, Adriano CM Pereira, and Renato A de Oliveira. "Stock market's price movement prediction with LSTM neural networks". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 1419–1426.

[29] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA: 2015.

[30] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. 2013, pp. 1310–1318.

[31] Jigar Patel et al. "Predicting stock market index using fusion of machine learning techniques". In: *Expert Systems with Applications* 42.4 (2015), pp. 2162–2172.

[32] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[33] Sreelekshmy Selvin et al. "Stock price prediction using LSTM, RNN and CNN-sliding window model". In: *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE. 2017, pp. 1643–1647.

[34] Dinggang Shen, Guorong Wu, and Heung-Il Suk. "Deep learning in medical image analysis". In: *Annual review of biomedical engineering* 19 (2017), pp. 221–248.

[35] David M Smith et al. "Sentiment and the effectiveness of technical analysis: Evidence from the hedge fund industry". In: *Journal of Financial and Quantitative Analysis* 51.6 (2016), pp. 1991–2013.

[36] Laurie Von Melchner, Sarah L Pallas, and Mriganka Sur. "Visual behaviour mediated by retinal projections directed to the auditory pathway". In: *Nature* 404.6780 (2000), p. 871.

[37] Prof. Loc Vu-Quoc. *Neuron*. 2018. URL: https://commons.wikimedia.org/wiki/File:Neuron3.png.

[38] Gilles Zumbach and Luis Fernández. "Option pricing with realistic ARCH processes". In: *Quantitative Finance* 14.1 (2014), pp. 143–170.