Vysoká škola ekonomická v Praze
FAKULTA FINANCÍ A ÚČETNICTVÍ
KATEDRA BANKOVNICTVÍ A POJIŠŤOVNICTVÍ

Diplomová práce

PRAHA 2018 **ING. JURAJ ČURPEK**

Vysoká škola ekonomická v Praze
FAKULTA FINANCÍ A ÚČETNICTVÍ
KATEDRA BANKOVNICTVÍ A POJIŠŤOVNICTVÍ
Studijní obor: Finanční inženýrství

Time Series Forecasts using the Neural Networks

Autor diplomové práce: Ing. Juraj Čurpek
Vedoucí práce: Ing. Milan Fičura
Rok obhajoby: 2018

Poděkování

Rád bych poděkoval vedoucímu mé diplomové práce za odborné připomínky a věcné rady při zpracování této práce.

Prohlášení

Prohlašuji, že jsem svoji diplomovou práci na téma Time Series Forecasts using the Neural Networks vypracoval samostatně a veškerou použitou literaturu a další prameny jsem řádně označil a uvedl v přiloženem seznamu literatury.

V Praze dne 4.1.2018 .....

Ing. Juraj Čurpek

Abstrakt

V této diplomové práci se věnujeme analýze schopnosti takzvaných Long Short-Term Memory neuronovych síti předpovědet náhodně vybrané finanční časové řady dostupné na webu Yahoo!Finance. Nejprve jsme určili pět nejvhodnějších modelů LSTM, které byly použity pro předpovědi 150 finančních časových řad z různých odvětví. Poté jsme z nich vypočítali statistické ukazatele týkající se predikovatelnosti časových řad - Hurstov koeficient, metrická entropie a největší Lyapunov exponent. Nakonec pomoci jednoduché regresní analýzy a korelačních koeficientů se snažíme určit vztah mezi kvalitou předpovědi reprezentovanou průměrným RMSE a statistikami vztahující se k predikovatelnosti časových řad s následným porovnáním s teoretickými předpoklady.

Abstract

This diploma thesis is devoted to the analysis of the Long Short-Term Memory neural networks performance in prediction of randomly selected financial time series from Yahoo!Finance. Firstly, we determined five most precise LSTM models used for predictions of 150 financial time series of stocks from various industries. Then we calculated the statistical measures related to the time series predictability - the Hurst Coefficient, Metric Entropy and the largest Lyapunov Exponent. By estimation of the simple regression lines and the correlation coefficients we try to determine possible relationship between the quality of prediction represented by the average RMSE and each of the statistics related to the time series predictability with the consequent comparison with the theoretical assumptions.

Contents

Introduction

The idea of forecasting the future events is merely as old as humankind existence. In case of financial time series of stock prices the things are quite difficult - they are non-linear, non-stationary and because their values change with high frequency, they exhibit a high volatility and irregularity with consequent negligible effect of seasonal changes, such that their log returns are used for statistical analysis. Besides using a wide-spread Box-Jenkins Methodology to a time series modelling, where a time series is considered a realization of the underlying stochastic process, recently also the artificial neural networks have been applied to model and to forecast their values which appear to be more appropriate for non-stationary time series forecasting. Furthermore, the time series predictability depends on the presence of memory in form of trends, how fast the information in time series decays and on the level of its chaotic behavior. These factors can be quantified by statistical measures which originate from the Chaos Theory.

In summary, the goal of this diploma thesis is to examine the forecasting performance of the Long Short-Time Memory neural networks applied on 150 randomly selected time series of stock prices log returns from Yahoo!Finance using an average RMSE, and its comparison to the measures linked to the predictability of the time series, namely Hurst coefficient, Metric entropy and the largest Lyapunov exponent, to find any statistical relationship between them. To secure the comparability we used five most precise neural network models chosen on small sample of various stocks. Furthermore, the whole process is automatized and programmed in Python programming language with the script provided in the Appendix of this thesis and containing the basic functions.

The diploma thesis consists of five chapters. We briefly describe time series most important characteristics and measures related to their predictability in chapter one. Then in the second chapter the brief introduction to the artificial neural networks is provided. The third chapter provides an overview of the methodology we used, with an emphasis given also to the technical details. The fourth chapter is about presenting the results in form of tables, heatmaps, graphs with additional comments. Finally, we comprise our results with some explanations in Conclusion part and give the reader some implications for the further study.

Chapter 1

General Introduction to the Time Series

This chapter is devoted to the fundamentals of the univariate time series, which mathematical apparatus and terminology used applies also to the financial time series. In addition, the second part gives some insight into various measurements related to the time series predictability.

1.1 Time Series Basic Characteristics

Many processes in the world generate measurable data such a temperature, unemployment rate, stock prices or number of sunspots, which can be recorded at equally spaced time intervals. Then this sequence of measurements of some quantifiable variable made at regular time intervals is called the univariate time series.[1] According to the time interval length we distinguish between two groups of time series: [1]

1. **Short-term time series** - observations are made in less than one-year time intervals, e.g. quarterly, weekly, daily time series

2. **Long-term time series** - observations are made in time intervals with length of one year or more, e.g. yearly time series

In economics, a term time series means the sequence of some specific thematically and spatially defined economic variable or index that is chronologically ordered from the past to present. A common notation of the time series is $y_t, t = 1,..,T$ or $Y = \{y_t, 1,..,T\}$. Note that we do not have to necessary deal with the time series of a measurable variable but also with any index or calculated economic variable based on actual observed economic variables. Another distinction of time series is based on the nature of tracked variable:

[1] In this thesis we do not deal with the multivariable time series, i.e. time series of more than one variable, that are a vector of m univariate time series $Y_t = (y_{1t}, t_{2t}, .., y_{mt})^T$.[2]

1. **Interval time series** - or flow time series, the values depend on the time interval length, e.g. GDP per year

2. **Instantaneous time series** - or stock time series, represent the values monitored in certain moments, e.g. closing stock price

3. **Derived time series** - time series of some characteristic that is derived or calculated from the observed values, e.g. daily average stock price

Furthermore, we can divide the time series into stationary and non-stationary. This division is very important because most methods for modelling and describing the time series assume that the working series, i.e. time series which we apply the chosen method on, is (approximately) stationary. The reason is that the stationary time series are easy to model and predict because their properties remain constant in time. The weakly stationary (or covariance-stationary) time series must satisfy three requirements: [3]

1. $E(y_t) = \mu, \forall t \in T$ - constant mean

2. $Var(y_t) = \sigma^2 < \infty, \forall t \in T$ - constant variance

3. $Cov(y_t, y_{t-k}) = \gamma_k, \forall t, k \in T$ - autocorrelation depends only on the interval k between two observations, not on time itself

The stationary processes have the mean reverting property which means that the time series fluctuate around a long-term mean value, and in case of an infinite time series the mean value would be crossed infinite times.

However, the real financial time series are far from being stationary processes. Fortunately, there are several ways or techniques of transforming the non-stationary time series into stationary ones. Then after applying the statistical forecasting methods on the stationarized time series, we can also obtain the models of the original time series by reversing the mathematical transformation. [4], [5]

- **Differencing the time series** - new differenced time series is a set of changes between two consecutive values: $y_t^{'} = y_t - y_{t-1}$ and has $T-1$ observations. Appropriate for the time series with a stochastic trend (variance is not constant) which are also called a difference-stationary.

- **De-trending the time series** - used for the time series with an evident deterministic trend (mean is not constant) where we firstly remove a trend via regression fit, and then model the residuals. Such time series are called a trend-stationary.

- **Logaritmization** - used for the time series which values increase exponentially (variance is non-constant). By taking the logarithm of the values the variance might be stabilized.

There are several ways of deciding whether time series is stationary or not:

- **Graphical analysis** - time series with evident trend or some irregular patterns are non-stationary
- **Plot of the autocorrelation function** - for non-stationary time series the ACF declines slowly and some values cross threshold. Used together with the partial autocorrelation function (PACF). [2]
- **Unit-root tests** - test the presence of the unit-root in the time series (I(1) process), i.e. the stochastic trend. The common one is the Augumented Dickey-Fuller test with following hypotheses: [7]

  $H_0 : y_t$ is I(1), i.e. difference stationary

  $H_1 : y_t$ is I(0), i.e. stationary or trend stationary

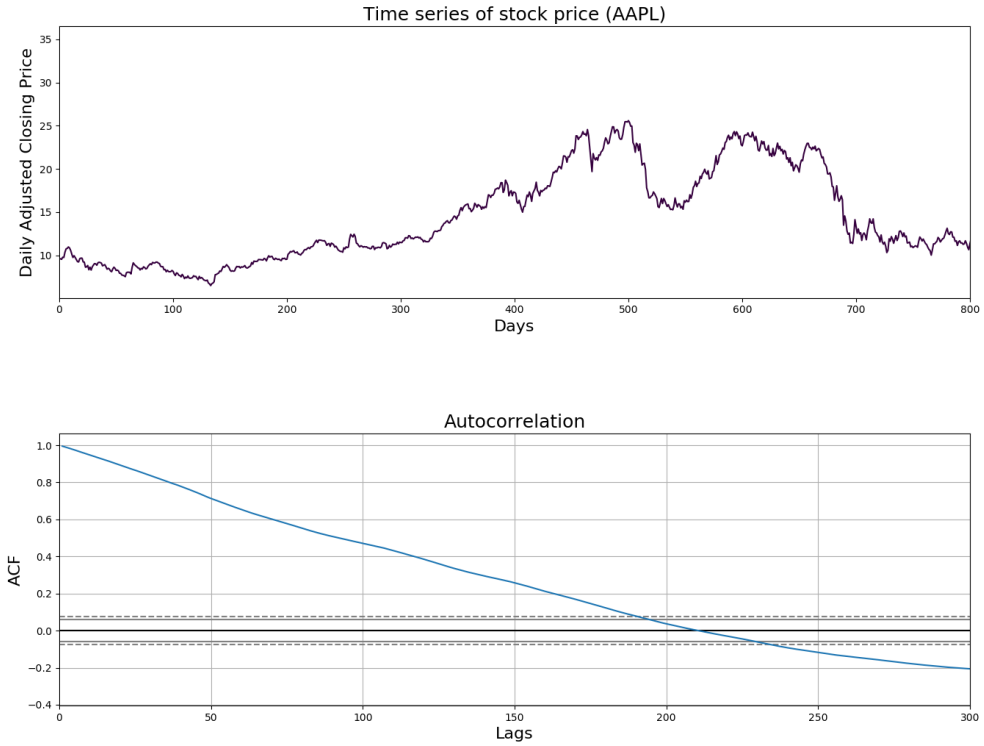

Figure 1.1: Collage of the non-stationary time series plot and corresponding ACF plot. (Author's own work)

[2] PACF indicates only a direct correlation between the values Y_t, Y_{t-k} conditional on the values between t and $t-k$. [6]

The goal of the time series analysis is to know the generating process through mathematical or statistical models. The process itself can be either linear or non-linear. Many real-life processes are actually non-linear, even though many time series models widely used are only linear. The linear process is defined as a stochastic process [3] $(y_t, t \in T)$ that is for every $t \in T$: [8]

$$y_t = \sum_{j=0}^{\infty} a_j \varepsilon_{t-j}, \tag{1.1}$$

where a_0=1 and ε_t is for every $t \in T$ independent and identically distributed with $E(\varepsilon_t)$=0, $E(\varepsilon_t^2) < \infty$ and $\sum_{j=0}^{\infty} \|a_j\| < \infty$. Then the non-linear process is any stochastic process that is not linear, hence it displays the behaviour such that time-changing variance, asymmetric cycles, higher-moment structures etc. that cannot be modeled by any linear processes.

1.2 Predictability of Time Series

Prediction of time series basically means to make a forecast about its future values based on the historical ones. The real-world financial time series are assumed to be nonlinear which means that their behaviour cannot be expressed as the linear combination of any linear process and even a small change in parameters can cause a huge change in the behaviour. There are two approaches how to describe this phenomena, either they are stochastic or deterministic with presence of the chaos.

If the financial time series are assumed to be generated by the stochastic (random) process, their randomness has some properties like the probability distribution etc. and, as already mentioned, can be either stationary - if their properties do not change with time, or non-stationary - if the properties of the stochastic process change in time. Another approach assumes that time series can be deterministic with presence of chaos, and as a result, the time series might appear to be random even though they are governed by some nonlinear deterministic equation. Note that chaos is defined as irregular longterm behaviour that depends on the initial conditions. Then they can be possibly predictable in the short term but in the long term they appear to be stochastic. [10]

1.2.1 Hurst Coefficient

Now we take a look at the so-called Hurst coefficient (or exponent) which is used to measure the predictability of the time series. Originally, it was developed by the British hydrologist Harold E. Hurst to model the water level of Nile river for appropriate building of the Asuan

[3] Stochastic process is the temporally ordered series of the random variables $\{y(s,t), s \in S, t \in T\}$, where S is the sample space and T is index series usually assumed to be a series of integers, i.e. $T = \{0, \pm 1, \pm 2, \dots\}$. Simply said, it is impossible to predict the future values with 100% probability. [9]

water reservoir. Several years later, the mathematician Benoit Mandelbrot continued with the Hurst's work and introduced the term Generalized Hurst Exponent as a measure of long-term memory of a time series, i.e. how much are trends persistent over time.

Furthermore, he characterized two effects that describe the long-term memory of the time series. The first one is called the Joseph effect occurs when the movements in the time series are actually part of a long-term trend. The second one is called the Noah effect and means the tendency of time series to change in unexpected way. [11]

According to a value of the Hurst exponent there are three types of the time series:

1. $H < 0.5$ - **Anti-persistent time series** - mean-reverting time series: because the values have tendency to return back to long-term mean, a decrease will be most probably followed by the increase and vice versa

2. H = 0.5 - **Random walk time series** - the observations are not correlated, so the probability that the future value will increase is the same as it will go down

3. $H > 0.5$ - **Persistent time series** - in the short term the values have tendency to continue in the same behaviour, i.e. if the values increase, then the future values will increase as well in the short term and vice versa

The Hurst coefficient is formally defined as follows:[12]

$$E\left[\frac{R(n)}{S(n)}\right] = Cn^H, n \to \infty, \tag{1.2}$$

where the left hand side means the expected value of the rescaled range after dividing the whole time series of length N into n small subseries of length i, i.e. $X = \{x_i, 1, .., n\}$, with $S(n)$ being the set of standard deviations of n subsets of size i, C is arbitrary constant and $R(n)$ means the range - the set of differences between the maximum and minimum for n subsets of size i:

$$R(n) = max(x_i, i = 1, 2, ..., n) - min(x_i, i = 1, 2, ..., n) \tag{1.3}$$

In reality, we deal with the finite number of subsets n, hence, at first, we calculate the expected value for the whole time series which is the average value of all sub-results of all small parts given by initial dividing, therefore:

$$E\left[\frac{R(i)}{S(i)}\right] = Ci^H, \tag{1.4}$$

$$log\left(E\left[\frac{R(i)}{S(i)}\right]\right) = log(C) + Hlog(i). \tag{1.5}$$

From equation (1.5) is evident that the Hurst coefficient H can be estimated as the slope of the least squares fit if we have several values of i. This approach is called the Rescaled Range Analysis (R/S analysis) since it includes computing of the rescaled range. Note that this approach can be applied to the stationary time series only and , unfortunately, does not show very good computational convergence.

Another frequent estimation method is called the Detrended Fluctuation Analysis (DFA) which unlike the R/S analysis can deal with the non-stationarity. This method was originally created to study the long-range power-law correlations in DNA sequences. Now it is seen useful for application to any nonstationary time series. The calculation algorithm can be briefly described as follows - assume we have time series with N observations. At first we integrate it and then divide it into equaly-sized non-overlaping subperiod of length n. In every subsample we find a local trend by the least squares method, usually a polynomial fit of some order to properly fit the data. After that the integrated time series is detrended by subtracting the local trend in each subperiod, so that we can calculate the root- mean-square fluctuation of this integrated and detrended time series also called the scaling function $F(n)$:

$$F(n) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y(k) - y_n(k))^2}. \tag{1.6}$$

Because this calculation is repeated over all time scales that are represented by the box size n, we find the relationship between the $F(n)$ and this box size n. Note that typically the relationship is increasing and takes the following form:

$$F(n) = Cn^{H_F}, \tag{1.7}$$

where H_F is in this case the Hurst scaling exponent. By plotting it on a log-log plot one may spot the presence of a so-called power law scaling [4]. Hence the fluctuations can be characterized by the slope of the straight line between $logF(n)$ and $log(n)$ which gives the value of the Hurst scaling exponent. [13]

Because DFA analysis can be applied both on the non-stationary and stationary time series, the Hurst scaling exponent can have greater value than one and relates to the classical Hurst exponent as follows: [15]

$$H = \begin{cases} H_F & \text{if time series is stationary} \\ H_F - 1 & \text{if time series is non-stationary} \end{cases}$$

[4] The power law represents a type of regularities found in many economics phenomena, and has the form of $Y = kX^{\alpha}$, where X,Y are the observed variables, k is some constant and α is the power law exponent. [14]

Hence it behaves the same way as the classical Hurst exponent for stationary time series, but not for the non-stationary cases, so scaling Hurst exponent brings another classification of time series: [16]

1. $H_F \approx 1$ - **1/f noise or pink noise** - exhibits the self-similarity and its power is inversely proportional to the density
2. $H_F > 1$ - **non-stationary random walk**
3. $H_F \approx 1.5$ - **Brown noise** - generated by cummulation of the increments of the stationary white noise (it is an integral of the white noise)

1.2.2 Lyapunov Exponent

Non-linear time series dynamics can be characterized by its chaotic behaviour. The chaos is typically characterized by the fact that tiny variations in the initial conditions can cause dramatic variations in the state of system after some time. Hence if we have two neighbouring system's states those initial conditions differ in a tiny size, e.g. due to small disturbations, after some time they diverge exponentially. To quantify the level of divergence of two neighbouring states in chaotic dynamic system, i.e. how much chaotic the time series is, we use the Lyapunov exponent (or the Lyapunov characteristic exponent). In general, the Lyapunov exponent of a dynamical system characterizes the rate of separation of the initially infinitesimally close points that create the so-called orbits (trajectories in phase space) due to some system of equations. Note that we can use n Lyapunov exponents for general n-dimensional system, but in this thesis we consider, for obvious reasons, one-dimensional system - time series, thus we deal with a single Lyapunov exponent. Let us assume two orbits in phase space with initial (at time zero) separation δZ_0. Their divergence can be quantified as: [17]

$$\delta Z(t) \approx e^{\lambda t} |\delta Z_0|, \tag{1.8}$$

where λ is the Lyapunov exponent. To asses the predictability of dynamical system we define so-called Maximal Lyapunov exponent considering $t \to \infty$:

$$\lambda = \lim_{\delta Z_0 \to 0} \frac{1}{t} \ln \frac{|\delta Z(t)|}{|\delta Z_0|}. \tag{1.9}$$

Then the Maximal Lyapunov exponent can be used to asses the stability of the system that is related to its predictability, that is defined as its inverse value. According to the sign of the Lyapunov exponent there are three possibilities. [18], [19], [20]

1. $\lambda > 0$ - **Chaotic system** - the more positive λ is, the shorter period can be predicted, i.e.the prediction power diminishes more rapidly

2. $\lambda = 0$ - **Marginally stable orbit** - some kind of steady state orbit

3. $\lambda < 0$ - **Stable periodic orbit** - the more negative λ, the greater the stability

1.2.3 Metric Entropy

In general, the financial time series can be non-stationary either due to the high level of variability or the due to presence of many irregularities in time series or both. The former case (variation) does not usually cause the unpredictability unlike the latter one - the presence of the irregularities has a huge impact of predictability. To measure the impact of irregularities we use the Metric entropy (Kolmogorov - Sinai entropy) which originates from the information theory.[5] Hence the Metric entropy ca be used to measure the rate of lost of predictability because it is proportional to the loss of information. In other words, information is the removal of uncertainty, and thereby loss of information leads to the higher values of entropy. In our case, when time series is a stream of numbers given by past observations, the Metric entropy provides the answer about how far into future we can make a prediction with given information. Theoretical definition is beyond the scope of this thesis but for illustrative purposes we show some basic steps of its derivation

The Metric entropy is derived from the entropy of partition $Q = \{Q_1, ..., Q_k)$ given by dividing the observation into k pair-wise disjoint pieces: [22]

$$H(Q) = -\sum_{m=1}^{k} \mu(Q_m) \ln \mu(Q_m). \tag{1.10}$$

The Metric entropy is defined as the supremum of the rate of change of entropy if we do the finer partitioning at each iteration or time step over all choices of the initial partition Q_0 indicated as intersection $\bigvee_{n=0}^{N} T^{-n}Q$.

$$K(T,Q) = \lim_{N\to\infty} \frac{}{N} H(\bigvee_{n=0}^{N} T^{-n}Q), \tag{1.11}$$

$$K(T) = \sup_{Q} H(T,Q). \tag{1.12}$$

For continuous system, the Metric entropy has units of inverse time, and for discrete systems has units of inverse iteration. It is always non-negative and the bigger values mean the less accurate prediction is possible. Generally, according to the value of the Metric entropy we differentiate between three types of system: [10]

[5] Note that term the entropy has actually its origin in thermodynamics where it quantifies the level of disorder and uncertainty of the dynamical system.

1. $K = \infty$ - **Random system** - system cannot be predicted at all, even one-step prediction is not possible

2. $0 < K < \infty$ - **Partially predictable but chaotic system** - the lower values, the more predictable system

3. $K = 0$ - **Periodic/non-chaotic system** - all the information can be predicted

However, this theoretical definition is related to the infinite data series with other characteristics like inifinitely accurate precision and resolution etc. For practical purposes, where time series data are finite, with specific sampling rate and limited resolution, the so-called Approximate Entropy Method is used to calculate the metric entropy without using big amount of data or alternatively the Sample Entropy Method. [21]

The Approximate entropy is, simply said, the value of the negative natural logarithm of the conditional probability that a short subsample of data with some specific pattern is similarly repeated during the time series. Hence it characterizes the amount of regularities or irregularities of fluctuations in time series which is related to the predictability or unpredictability of time series, i.e. small Approximate entropy means that time series contains a lot of similar patterns and vice versa. In addition, the Approximate entropy can measure Gaussianity of distribution in a way that it has a theoretical maximum for random Gaussian process, so the possible deviations can indicate the lack of Gaussianity.

On the other hand, the Sample entropy is the negative natural logarithm of the empirical conditional probability that two sequences which are similar for so-called embedded dimension points are similar at the following point expect for the self-matches. The interpretation is the same as for the Approximate entropy.

However, the Approximate entropy method has its disadvantages like higher sensitivity to bias in short time series, and the computationally slower algorithm. Fortunately, the Sample entropy can solve these issues because it was found out that the Sample entropy can preserve relative consistency more frequently than the Approximate entropy, and it is more robust. On the other hand, the Approximate entropy is more similar to the definition of the Shannon's definition of information entropy.

Although the Sample entropy can be used on the non-stationary data we must pay attention to the calculation, especially when spikes are present because they can increase the variance and the bias the results. Note that the Sample and Approximate entropy are very sensitive to the presence of spikes. This problem can be addressed by removing these spikes from the time series by some sensible algorithm, bu in this thesis we do not have to deal with this issue. [23]

Chapter 2

The Artificial Neural Networks

2.1 Theoretical View of the Artificial Neural Networks

In present, we are used to the powerful capabilities of the computers which software is nothing but the long sequence of well chosen and structured commands called the algorithm. The traditional algorithmic approach is powerful in solving mathematical problems or tasks that can be easily algorithmized. However, some tasks, viewed as simple from the perspective of the human being, like facial recognition or language processing are very difficult to programme, i.e. the code for these task would be extremely complex. Fortunately, we can try to mimic the processes how the human mind works, and thereby create the artificial neural networks (ANN) to deal with these problems. Although it is not certainly proven that the ANN works the same way as a human brain does, but it certainly draws some inspiration from it, like the fact that brain can learn from the experience.

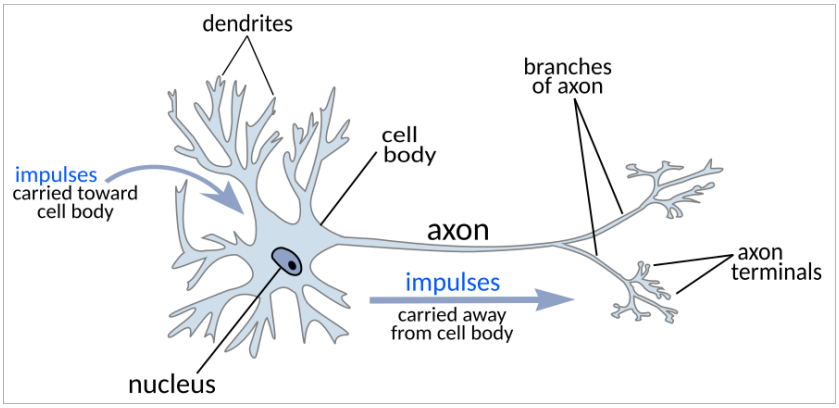

Figure 2.1: A schematic image of human neuron. (Credit wpclipart.com webpage)

The basic element of the human brain is the neuron (depicted on the figure 2.1) consisting of a nucleus, a large number of dendrites (up to 200 000 but 10 000 typically) that collect the input signals and one axon that sends the output to the another neurons via several boutons.

The information is carried via electrochemical pathways and if the collected inputs (electrical signals from dendrites) are greater than some so-called firing threshold, the neuron sends the electrical impulse to the other neurons, i.e fires the impulse. These interconnected neurons create the neural network resided in our brains which consists of the neurons and their connection - synapses and are responsible for the power of the human brain. The estimated number of the neurons in the human brain is about 100 billion and number of synapses is about 100 trillion.

There are three concepts that can describe the behaviour of the human brain. First one is the concept of the connection strengths between the neurons which are responsible for the storing the information. New experience or repetitive stimuli lead to change of the connection strengths, thus some synapses are reinforced, some are created and some disappear. The second one is the fact that the neuron is either excitatory, which increases the firing rate, or inhibitory which decreases the firing rate. The level of excitation or inhibition is positively related to the connection strength. The third one is the concept of so-called transfer function which determines how the firing rate depends on the input, e.g. neuron can take a sum of its input or something completely different. [24]

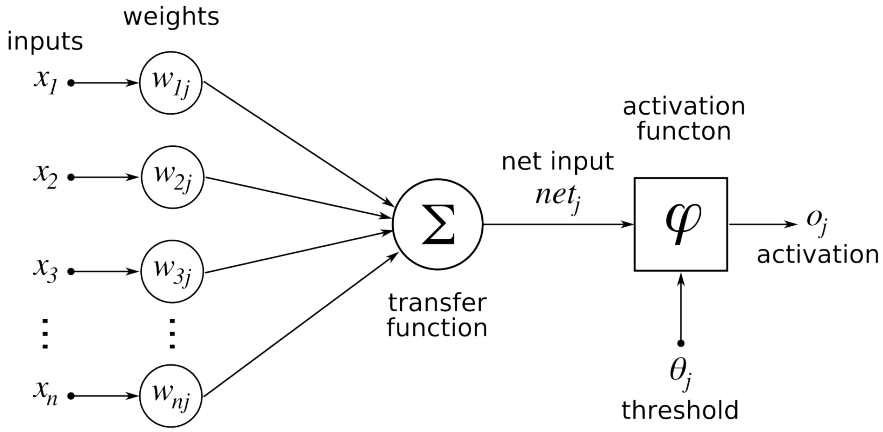

Figure 2.2: A model of the artificial neuron. (Credit innoarchitech.com webpage)

The artificial neural networks try to capture only the basic characteristics of the biological neural networks because the real neurons are in fact more complicated, and like in the case of the biological neural networks the power of the ANN comes from the massive parallel information processing after the ANN was trained for the specific problem. The general model of the artificial neuron is depicted on figure 2.2. Each neuron has several inputs which are weighted individually and then are, in the simplest case, just summed together and passed into the activation function which converts the net input into output. There are many activation functions, depending on the neural network type, for example step function that creates output of 1 if the net input is higher than a specific threshold and 0 otherwise,

or normalized sigmoid activation function that is bounded to all values between 0 and 1 for all real numbers etc.[25]

Weights of connections between neurons are stored in a matrix w, where w_{ij} denotes weight of the connection from neuron i to neuron j. Every neuron/unit j has a potential p_j which is calculated as weighted sum of all of its N inputs neurons stored in the vector x and the overall bias (also known as threshold unit) is usually represented as an extra input unit whose activation always equals one. The output is then: [28]

$$output = f(p_j) = f(w * x + b), \tag{2.1}$$

where f is some activation function. The most often used activation function is so-called sigmoid (logistic) function depicted on the figure 2.3, which can be defined as:

$$\sigma(z) = \frac{1}{1 + exp^{-z}}, \tag{2.2}$$

so if we use the sigmoid function, the output is in our case:

$$output = \frac{1}{1 + exp^{-w*x-b}}. \tag{2.3}$$

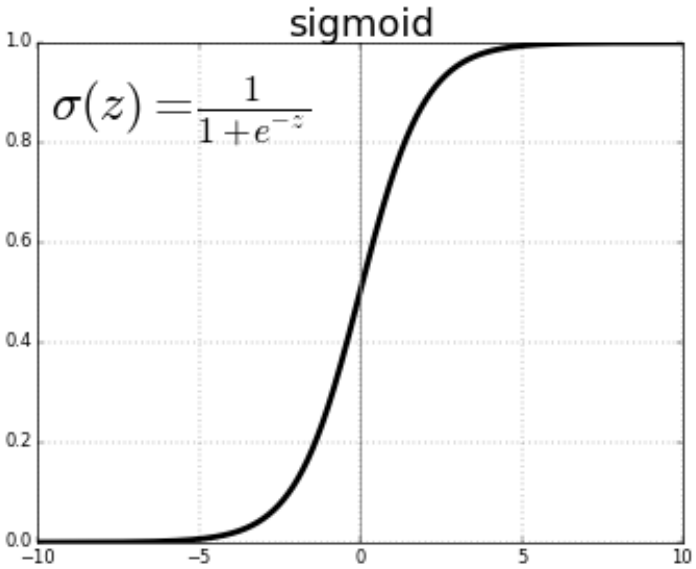

Figure 2.3: A sigmoid activation function. (Credit ml4a.github.io webpage)

The interconnected artificial neurons create the artificial neural networks. Although there are many different architectures of the ANN, the most of them has the basic structure depicted on the figure 2.4. The ANN consists of set of input nodes which take the inputs, one or more

layers of the hidden nodes and the output nodes which provide us with the output. Note that there can be any number of nodes per layer (depending on the input data and given problem) and the issue of choosing the proper number of nodes and layers is crucial in optimizing the neural network. The information from the input nodes can flow in one way or both ways depending on the architecture, and is represented as activation values in each node meaning that the greater activation means higher value. Then each neuron passes the activation value on the next node according to the connection strengths and transfer function. [26]

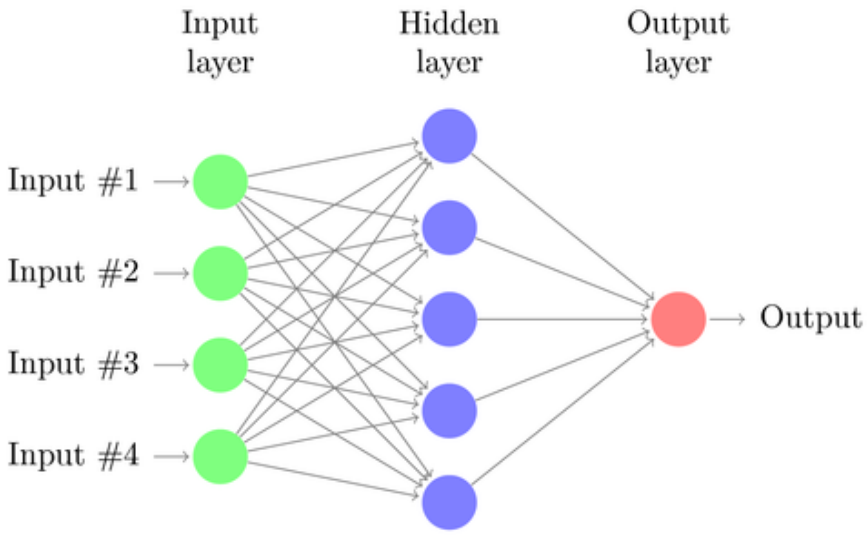

Figure 2.4: A general model of the artificial neural network. (Credit texample.net webpage)

2.2 Types of the Artificial Neural Networks

In the relevant literature, one may find a huge number of various neural network types, but the two most common types are: [33]

- **Feed-forward neural network** - The simplest type of the ANN consisting of the perceptrons in several layers where the information flows in one direction only - from the input node throughout several hidden nodes (if any) to the output nodes. There are no cycles or loops present in this type of the ANN. In case of the single layer feed-forward neural networks, only linear functions can be approximated, but on the other hand, the linearity causes the learning algorithm to converge to the optimal solution.

  In reality, the multi-layer feed-forward neural networks are usually used. They consist of the input nodes, output nodes and at least one hidden layer. The hidden layer receives inputs form the layer in the previous level, i.e. the first hidden layer is fed by

the information from the input layer, the second hidden layer receives the inputs from the first hidden layer etc., and send the outputs to the layer in the next level. Note that there are no connections between the nodes within one layer. The schematic image of such multi-layer feedforward neural network is depicted on the figure below.

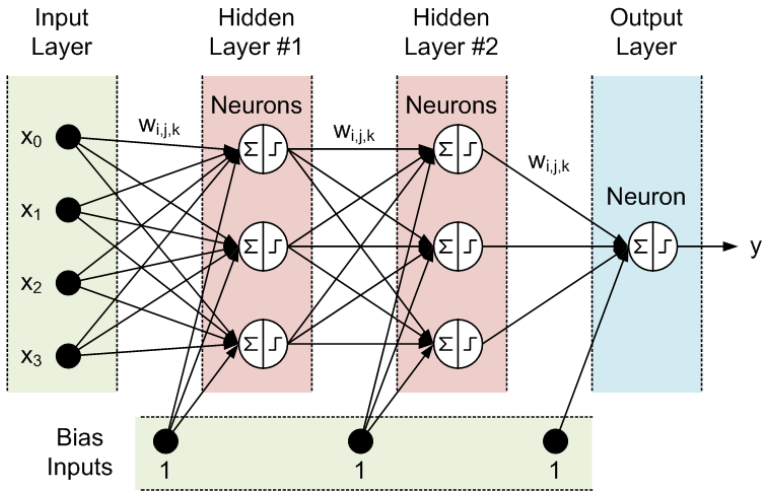

Figure 2.5: The feed-forward neural network with two hidden layers. (Credit mql5.com webpage)

- **Recurrent neural network** - As opposed to the feed-forward neural networks, in the recurrent neural networks a bi-directional flow of data in a form of directed cycles is possible, e.g. the hidden unit can be connected with itself over a weighted connection or the with the input units or output units can be connected to the hidden units etc. Hence, it can be viewed as a sequence of the multiple copies of the same network as depicted on the figure 2.6. This fact allows the recurrent neural network to exhibit a dynamic temporal behaviour via its internal memory due to the recurrent connections, and thereby it is capable of learning the relationship between the sequence of the inputs. Thus the recurrent neural networks are appropriate for the time series forecasts.

  The most well known recurrent neural networks types are, for instance the Jordan networks (the activation values from the output units in the output layer are fed back to the input layer to the extra set of the input state units), the Elman networks (the activation values from the hidden units in the hidden layer are fed back to the input layer to the extra set of the input context units), and even theoretical concepts like the Hopfield network (or auto-associator network which consists of the set of mutually interconnected neurons that update their activation values independently of each other and in asynchronous manner, so all units/neurons are both input and output neurons). For the purpose of this thesis we use the special type of the recurrent neural networks

called the Long Short-term Memory network (LSTM), which is able to preserve the error term backpropagated through layers and time, and it is described in the section 2.4.

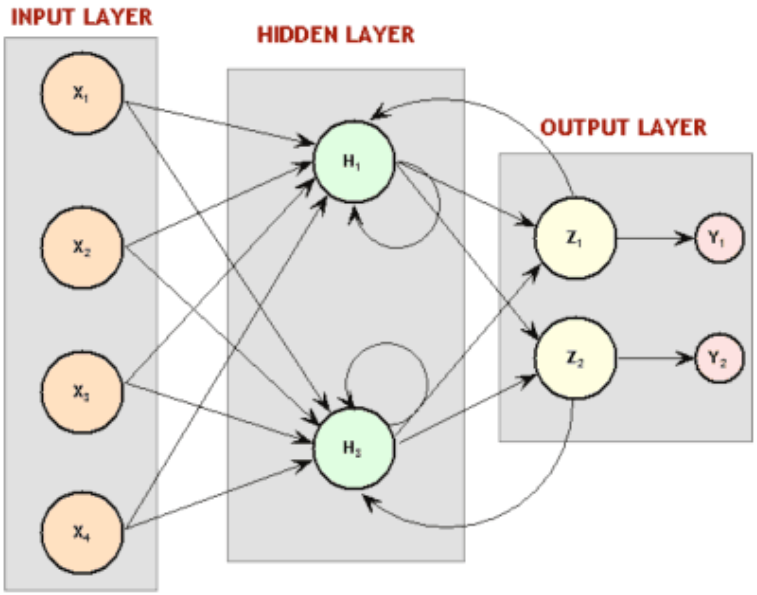

Figure 2.6: The recurrent neural network. (Credit mattmoocar.me webpage)

Apart from that, there are four more types of the ANNs. [29]

- **Radial basis function neural network** - this kind of the ANN gained its name from the type of an activation function used - the radial basis function[1], typically Gaussian, which is nonlinear. The output is then the linear combination of the radial basis activation function of the inputs.
- **Kohonen self organizing neural network** - due to its self organizing mechanism it applies a different type of learning from the inputs that is so-called competitive learning approach (in contrast to the error minimization approach) without a supervision. They are used for visualization of high-dimensional data in lower dimensional spaces, mostly two, or for describing the hidden structures in some unlabeled data. The inputs are represented in such manner that a topological relationship is maintained within the network, hence they are sometimes named as Kohonen maps.
- **Modular neural network** - it is a complex of several independent neural networks, which receive independent inputs and perform the sub-tasks which contribute to the

[1]The radial basis function $f(x)$ is a function that depends on the distance from a certain point of origin c, i.e. $f(x) = f(\|x-c\|)$ and is radially symmetric about that origin. The norm is usually the Euclidean distance, even though the other types of norm can be used. [34]

task of the whole neural network. They are moderated by an intermediary which adopts the received inputs from each of these individual ANNs, performs their processing and eventually generates the final output for the modular neural network.

- **Physical neural network** - as opposed to the other types of the ANNs that use software only, this type is based on the hardware capabilities to create a physical structure imitating the neuron. Specifically, the neural synapse is created by the electrically adjustable resistance material. Although the neurons and their synapses are of physical origin, the whole network is emulated by the software.

2.3 Learning of the Neural Networks

As we have already mentioned, the ANN posses the ability to learn from experience. Although this learning process was inspired by the biological nervous systems, one has to point out that unlike the humans, the ANN has no perception about what it is learning, it just learns how to solve given problem based on the learning. Thus the process of learning can be defined as an improvement of performance at a given task with experience. The ANN learns by adjusting (strengthening or weakening) the connection strengths - their weights between the neurons which mimics the altering of the synapses in the human brain. The goal of the learning is to find a set of weight matrices that should map any input to a correct output based on the training sample. [27]

There are several different types of algorithms used for the ANN training, three most important ones are:

- **Supervised Learning** - during the training we provide the ANN with the set of inputs and their desired target outputs. Then we compare the actual output with target output and calculate the error as a difference between them. The corrections of the weights are then made in such manner that we want to minimize this error term.

- **Unsupervised Learning** - we provide the ANN only with the set of the inputs and let the ANN to find some meaningful pattern in the input data without any additional feedback (we do not have any target output). It is widely used in data mining and recommendation algorithms to predict the preferences.

- **Reinforcement Learning** - it is somehow similar to the supervised learning that we have some feedback, but instead of the set of target output there is a reward based on the ANN performance. The goal is to maximize the reward the ANN receives through trial and error. It resembles the learning patterns found in nature, e.g. animal must take a certain actions in order to receive food etc.

The learning method used mostly for supervised ANN training is called the backpropagation included in the gradient descent algorithm for minimization of the error term in the so-called cost (or loss) function of two variables - weights and biases: [28]

$$C(w,b) \equiv \frac{1}{2n}\sum_{x} \|y(x)-a\|^2, \tag{2.4}$$

where $\|y(x)-a\|$ is the difference between desired output and actual output and n is the total number of training inputs. The goal is to minimize this cost function, i.e. to find such set of weights and biases such that $C(w,b) \approx 0$. The gradient descent algorithm is based on the fact that $C(w,b)$ decreases fastest if the set (w,b) moves in the direction of the negative gradient of $C(w,b)$.

Now let us the simplify our problem and assume that we have cost function for some general $w_{l,k}$ and b_k. Then the change of the cost function is:

$$\Delta C \approx \frac{\partial C}{\partial w_{l,k}}\Delta w_{l,k} + \frac{\partial C}{\partial b_k}\Delta b_k, \tag{2.5}$$

$$\Delta C \approx \nabla C * \Delta(w,b), \tag{2.6}$$

where the gradient vector of the cost function is:

$$\nabla C \equiv \left(\frac{\partial C}{\partial w_{l,k}}, \frac{\partial C}{\partial b_k}\right)^T. \tag{2.7}$$

The goal of backpropagation is to calculate the partial derivatives of the cost function $C(w,b)$ present in the gradient equation with respect to any weight w or bias b. It gained its name from the fact that the error terms are computed backwards from the output layer back to the input one. Then these error terms are used to calculate the gradient vectors and to adjust the weights in order to minimize the cost function.

In case of the recurrent neural networks the method of backpropagation is slightly altered due to the nature of the RNN to the so-called BackPropagation Through Time algorithm. The recurrent neural network can be viewed as an unfolded sequence of layers in feed-forward neural network. Hence we calculate the gradients of the multi-layer feed-forward network with a fixed number of layers such that the activations in each layer represent the activations of the recurrent neural network at time step t so that the activation for time $t = 0$ is at the highest layer. The activations themselves are calculated using the activations from the layer one step below, i.e. at time $t = 1$, and in synchronous way. [30]

While training the RNNs one may run into two problems - exploding and vanishing gradients. Recall that our learning algorithm is based on decreasing the error-term but if we cannot adjust the weights properly such that the gradient diminishes, the neural network

would have difficulties to be properly trained. It is actually very easy for the RNNs that the gradient suddenly explodes - rapidly increases, or vanishes - rapidly diminishes because the layers in the unfolded RNN are related through a multiplication, so the overall gradient is a sum of the products of partial derivatives assigned to each time step. [31]

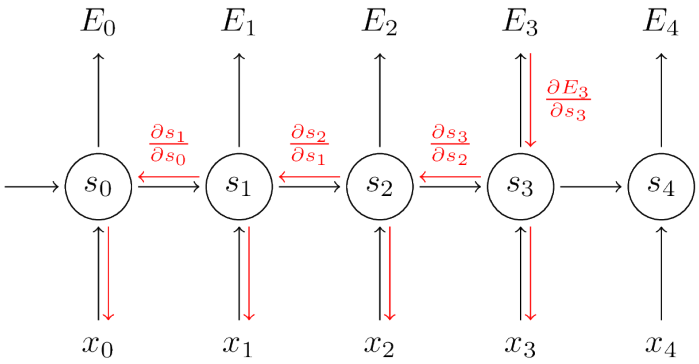

Figure 2.7: The RNN as a unfolded copies of the same neural network. (Credit wildml.com webpage)

To get some mathematical insight into the problem, assume we have already summed up the gradient at each time step in RNN depicted on the figure 2.7 above:

$$\frac{\partial C}{\partial w} = \sum_t \frac{\partial C_t}{\partial w}, \tag{2.8}$$

and each temporal contribution is a sum of the product of partial derivatives obtained through chain rule of derivation because each output y_t depends on hidden state at time t which is calculated as some function of previous hidden state and current input:

$$\frac{\partial C_t}{\partial w} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial y_t} \frac{\partial y_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial w}. \tag{2.9}$$

We just added up every temporal contribution to the overall gradient because the weight w is used in partial derivative for each time step up to the output y_t. After some algebra presented in (Razvan Pascanu et. al 2012 [32]) and simplifying we obtain the following condition:

$$\| \frac{\partial s_t}{\partial s_k} \| \leq \eta^{t-k}, \tag{2.10}$$

thus when $t-k$ is large the gradient diminishes exponentialy to 0 for $\eta < 1$, and for $\eta > 1$ the gradient rises exponentially to infinity.

2.4 Using the Neural Networks for Time Series Prediction

In general, a real time series prediction is difficult because of a presence of temporal dependence between the observations. The financial time series are mostly forecasted by applying the Box-Jenkins method in form of linear ARMA models (or their modifications) because of their simplicity and the fact that they might be effective on the most of the problems. However, they have a few limitations: [35]

- **Focus on the linear relationships** - many real-world time series might be non-linear and chaotic
- **Assume a fixed temporal dependence** - the relationship between observations might vary with time
- **Applied to stationary time series** - we often find a real-world time series non-stationary, so they must be stationarized before the ARMA model is to be used
- **Used with complete data only** - time series may often suffer from missing data

To tackle these limitations the artificial neural networks are used due to their ability to approximate any mapping function from inputs to outputs. They have several advantages regarding the time series predictions:[36]

- **Can handle the non-linearity** - the ANNs can approximate any non-linear function
- **Make no stationarity assumptions** - the ANNs can be applied directly on the non-stationary time series
- **Require minimal data preprocessing** - using the traditional approach we must identify trends, seasonal and cyclical patterns found in the stationary time series
- **Can cope with missing data** - the ANNs can learn the patterns even from incomplete dataset

There are many ANNs models that can be used for predictions of the time series, e.g. feed-forward neural networks, recurrent neural networks, radial basis functions neural networks etc. Although the feed-forward neural networks are the mostly used type of the ANNs, their mapping function is static, so the temporal dependence is ought to be specified while designing the appropriate model. On the other hand, the RNNs are able to learn not only the mapping function from inputs to outputs, but also the mapping function for the inputs over time to the outputs, i.e. they can learn the temporal dependence from the given data. For

that reason in this thesis we focus on the special case of the recurrent neural network model called Long Short-Term Memory network.

Long Short-Term Memory networks - introduced in 1997 by Hochreiter and Schmidhuber as the special type of the RNN that should reduce the vanishing gradient problem. They are able to learn both short-term and long-term dependencies in the data. Similarly to the basic RNN architecture, the LSTM networks contain the sequence of self-repeating modules due to the loops, but these modules or memory blocks are quite different in this case. The memory block consists of four basic parts - a gated cell that contains the information about the temporal state of the network outside of the RNN flow, an input gate that decides what information should be stored in the cell state, an output gate that decides what should be an output, and a forget gate that decides what information should be forgotten (disappear).

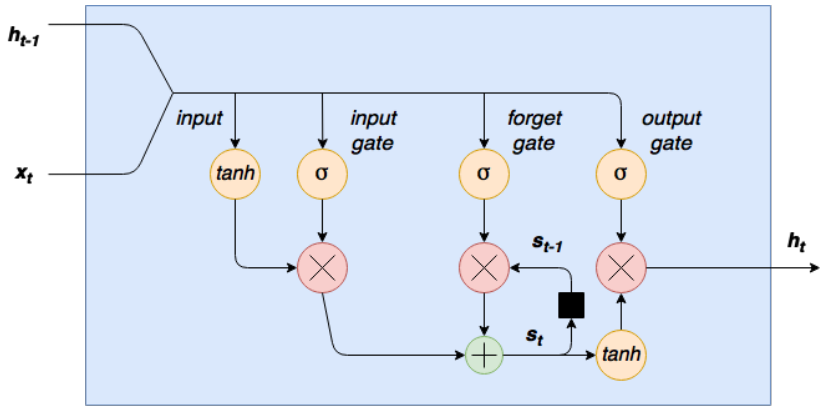

Figure 2.8: A diagram of LSTM cell. (Credit adventuresinmachinelearning.com webpage)

The behaviour of these three multiplicative gates is similar to the neural network nodes in sense that they serve to control the information flow to the memory gated cell using the weights that are subject to the RNN learning. Each gate consist of a simple neural sigmoid layer, so each of them outputs a value between 0 and 1. In case of updating the cell state to the new value, we combine the value from the input sigmoid layer, which defines what to update, without the information defined in the forget gate, with a *tanh* layer that produces the vector of candidate cell values. The final output is generated similarly, i.e. at first the output sigmoidal layer defines what to output (what parts of the cell state) and then it is multiplied by the cell state transformed by *tanh* layer. [37]

Chapter 3

Methodology

This chapter is devoted to the methods of obtaining data, their manipulation and their final processing, i.e. it describes the process of how we obtained the results from time series data.

3.1 Process Description

To achieve the goals of this thesis we had to calculate the statistics describing the predictability, and apply the neural networks on 150 time series of stock prices. For obvious reasons, it was rational to automatize this process using some programming. We chose to programme the whole process in Python language (Anaconda distribution of Python version 3.6.1, Anaconda version 4.4.0), using mostly the data science libraries Pandas, Numpy, Nolds, Keras and TensorFlow. The algorithm consists of several steps:

1. Download the time series of stock prices from Yahoo!Finance server using Python package pandas-datareader included in library Pandas, and convert them into daily log returns.

2. Divide the sample into training and testing sub-sample to train, and then generate and asses the chosen LSTM performance by RMSE measure.

3. Calculate the statistics concerning the predictability of time series using the Nolds library: Hurst exponent (R/S and DFA method), Lyapunov exponent, Sample entropy.

4. Save the results of both the LSTM results and predictability statistics calculations to an Excel file.

For better illustration of the process for a single time series, a flowchart is depicted on the next page.

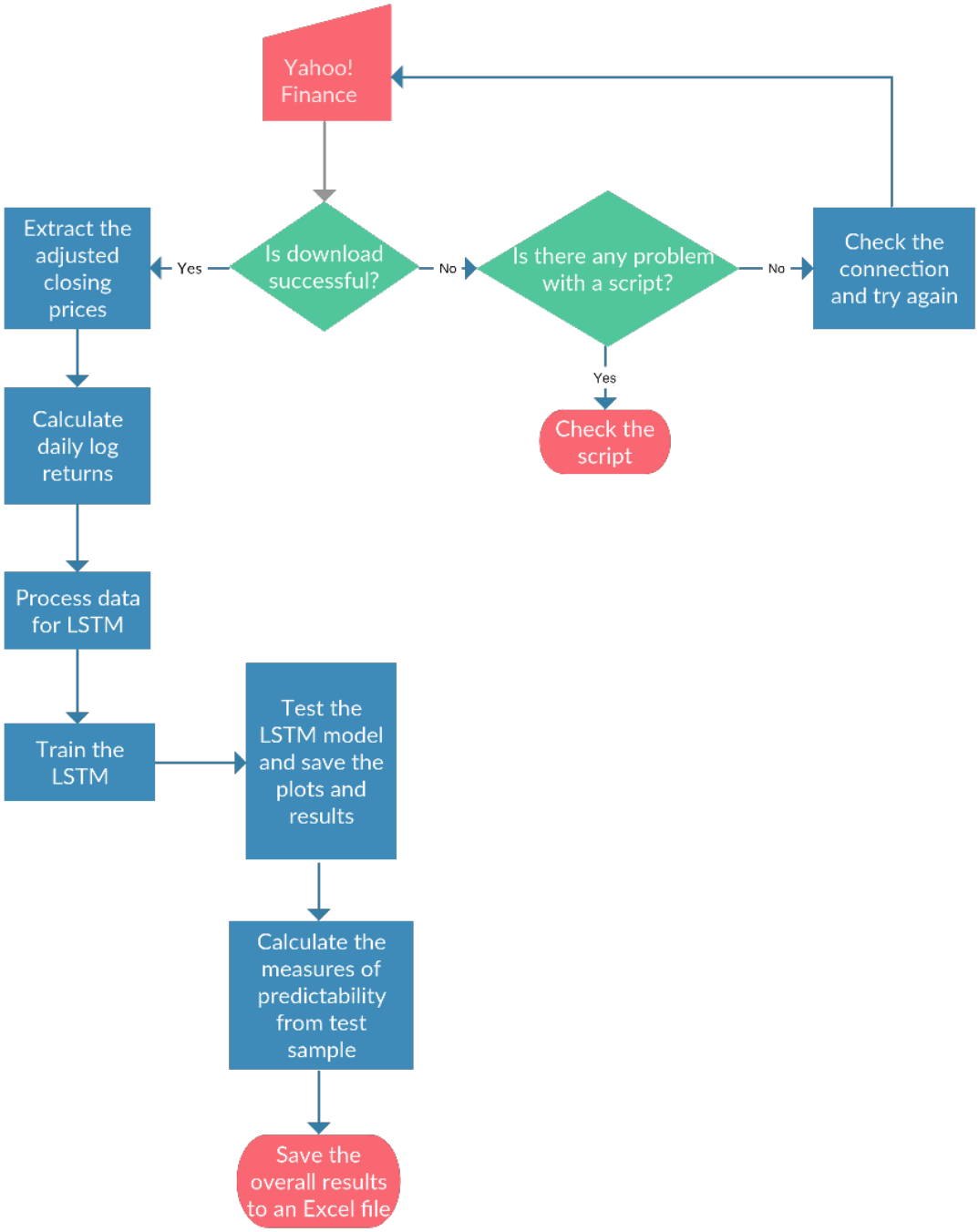

Figure 3.1: The process flowchart for generating the results from a single time series. (Author's own work.)

3.2 Data Sources

The only source of the financial data is the Yahoo!Finance website accessed via Python package pandas-datareader that is able to download the data specified by the ticker symbol. For the more information visit the pandas-datareader documentation webpage.

Because we used the Python script to automatize the whole process, we could easily download a huge number of various time series from different industries to generate the ANN and to calculate the statistics related to the predictability. The only thing we hae to do was to write the list of the tickers into script and then the algorithm did the rest of the job. However, pandas-datareader needs a stable internet connection, so sometimes it might occur that it cannot reach the Yahoo!Finance server and the whole script just stops, and then one needs to know what was the last ticker successfully handled, so he or she can continue with the next one. To alleviate this possible inconvenience, the script prints out the name of the successfully downloaded ticker, and furthermore, we suggest not to use too many tickers at once for easier orientation.

Data acquired from the Yahoo!Finance have a form of time series organized in a table with six columns (pandas DataFrame), namely opening daily price, the highest price, the lowest price, closing price, the adjusted closing price and the volume as depicted below. It does not count an indexing column Date as an actual column. Then we just had to extract the daily adjusted closing price from the fifth column and transform it to their log returns which is our working time series.

Date	Open	High	Low	Close	Adj Close	Volume
2006-01-03	35.099998	35.400002	34.799999	35.369999	23.282021	33221200
2006-01-04	35.349998	35.400002	35.099998	35.320000	23.249096	24017100
2006-01-05	35.270000	35.349998	35.099998	35.230000	23.189857	18857700
2006-01-06	35.380001	35.529999	35.189999	35.470001	23.347839	22084000
2006-01-09	35.400002	35.430000	35.240002	35.380001	23.288599	20716000
2006-01-10	35.270000	35.310001	35.049999	35.189999	23.163538	21041600
2006-01-11	35.139999	35.549999	35.099998	35.430000	23.321508	22394400
2006-01-12	35.450001	35.459999	34.939999	35.000000	23.038471	22159000
2006-01-13	34.900002	35.250000	34.799999	35.099998	23.104286	17216600
2006-01-17	35.630001	35.630001	34.820000	34.939999	22.998968	16068800
2006-01-18	34.900002	34.910000	34.610001	34.820000	22.919977	20154300
2006-01-19	34.900002	34.900002	34.500000	34.680000	22.827833	29312200
2006-01-20	34.299999	34.500000	33.220001	33.369999	21.965532	88077400
2006-01-23	33.529999	33.669998	33.259998	33.290001	21.912874	38813000

Figure 3.2: Data obtained from Yahoo!Finance. (Author's own work.)

The fact that we did not used the prices but their log returns has some theoretic reasons. Firstly, an one-period log return is defined as: [38]

$$r_t = log(P_t) - log(P_{t-1}) = log\frac{P_t}{P_t} = log(1+R_t), \tag{3.1}$$

where we consider a logarithm with a natural base and R_t is the gross one-period return. The log returns of the financial stocks are said to follow some well-known properties which make their statistical analysis much easier: [39]

- Stationarity - the asset prices are often non-stationary, but their log returns fluctuate around some constant level, i.e. their sample mean is close to zero
- Asymmetric distribution - the log returns distribution is symmetric- negatively skewed with heavy tails
- Volatility clustering - the large changes of prices both positive and negative occur in clusters
- Aggregational Gaussianity - the log returns are time additive, i.e. a return over n days in an aggregation of n one-period returns, and due to Central Limit Theorem for a large time horizons they converge to the normal distribution
- Long-range dependence - the serial autocorrelations are tiny, but on the other hand the absolute values or squares of returns for a large number of lags indicate some significant autocorrelation

3.3 Artificial Neural Networks Generation and Tuning

In this section we describe how the artificial neural networks we used were constructed. Since in this thesis we deal with a numerous time series, we want to obtain a comparable data through use of several artificial neural network that are, obviously, trained on each of time series. Even a common sense would tell us that for different time series one should find a different most appropriate LSTM, i.e. the LSTM with different architecture. For instance, the LSTM with four LSTM memory blocks could perform better at predicting the time series of one stock than another one that could be better predicted by the LSTM with just two LSTM memory blocks. Since our goal is to have comparable and standardized results, we seek for five most precise LSTM networks that could be effectively applied on all our data, and then their RMSE can be averaged .

To design five the most precise LSTMs we chose eight representative stocks from different industries (chosen by rule of thumb) and then trained the ANN with different parameters, namely number of LSTM memory blocks and the number of learning epochs. As a results we got the heatmaps for each stock of how the RMSE changes for LSTM memory blocks ranging from two to ten, and epochs ranging from three to fifteen. The interval was chosen by rule of thumb respecting the fact that our final LSTM should not be a computationally intensive. Then we picked five best LSTM model according to the smallest average RMSE.

The programming of our artificial neural network that is actually a so-called Long Short-Term Memory network type was inspired by the Machine Learning Mastery website. For generating a LSTM network we used a Python deep learning library Keras which extensive

documentation is available online. The idea is that the prediction of the time series is treated as a regression problem, thus we trained the neural network to predict the successive value from the information obtained from the recent values.

At first, we extracted our data from downloaded pandas DataFrame into NumPy array with integer values and converted them into floating values and then we continued with the preprocessing the data. It is a common practice that the input values should be normalized because real data are mostly distant from each other. And because the common activation functions such as sigmoid, hyperbolic tangent and gaussian produce result that ranges between $[0,1]$ or $[-1,1]$, it should be important to normalize the values to that range. The common normalization approach is so-called Min-Max Normalization according to the equation below. [40] For doing that we just used the preprocessing class from the scikit-learn library called MinMaxScaler. Furthermore, we divided our dataset into training and testing sample.

$$y = \frac{x - min(x)}{max(x) - min(x)}. \tag{3.2}$$

To grant the reproductibility of our results we set he specific seed numbers value in the script, so it always starts generating the pseudo-random numbers from the same seed. To define a regression problem we used a function called *create_dataset* which (obviously) creates a new dataset with the columns $X1 - Xn$ according to the window size and output Y that are gradually lagged by one period while taking two arguments as inputs: *dataset* which are the downloaded data and the *look_back* parameter which describes the number of lags (previous time steps) to use an inputs for forecasting the next time period. Then the lagged recent inputs $X1 - Xn$ were used to predict the next period output value of Y. However, we did not treat the past observations as the separate inputs, but rather as time steps of one input feature. In addition, while using the Keras for LSTM generating, the input data should be in a specific form of $[samples, timesteps, features]$. This can be done be using a function $numpy.reshape()$. In our case we reshaped the dataset in a way that the feature dimension was 1 and the columns were the time steps dimension.

The LSTM network was then designed using these parameters: number of LSTM memory units/blocks in a hidden layer and the number of epochs. We did not change the default setting of one input in visible layer and one output. Also we used a sigmoid activation function for the LSTM blocks and linear activation function for output layer. Note that we designed the neural networks with only a single hidden LSTM layer.

After fitting a model, we estimated its performance by using a RMSE measure, both for the train and test sample. Finally, we made a forecast both the train and test sample and plotted them to get some visual overview of the model. The original data are in blue, the forecast for the training sample is in green, and for the testing sample is in red.

To find the optimal modesls of LSTM we chose eight representative stocks from different industries and train the LSTM network with different configuration to see which one provides us with the lowest average RMSE. The RMSE is simply defined as the square root of the mean square error where error terms or residuals are simply differences between the actually observed values and the values predicted by a model:[41]

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_{obs,i} - y_{model,i})^2}{n}}. \tag{3.3}$$

Sometimes we need to calculate the average RMSE from several samples. To do so, the average RMSE for the whole population P consisting of k sub-populations n_j is calculated a little bit differently than classical (unweighted) average.

$$\overline{RMSE} = \sqrt{\frac{\sum_{i=1}^{P}(Y_{obs,i} - Y_{model,i})^2}{P}} = \sqrt{\frac{\sum_{j=1}^{k}\sum_{i=1}^{n_j}(y_{obs,i,j} - y_{model,i,j})^2}{\sum_{j=1}^{k} n_j}}, \tag{3.4}$$

and using the equation 3.2 we get:

$$\overline{RMSE} = \sqrt{\frac{\sum_{j=1}^{k} RMSE_j^2 n_j}{\sum_{j=1}^{k} n_j}}. \tag{3.5}$$

3.4 Calculation of the Predictability Statistics

The calculations were made using a Python library (based on NumPy) called Nolds which is capable of calculating the non-linear measures for dynamical systems based on one-dimensional time series. It is not usually implemented in Pandas library of Anaconda distribution, so it has to be downloaded and installed from the python package index, specifically Nolds package is available on this website where is also link to its documentation with further description.

3.4.1 Hurst Coefficient Estimation

Recall that for Hurst exponent (or coefficient) estimation we can use two approaches described in Section 1.2.1 that are rescaled range (R/S) and the detrended fluctuation analysis (DFA). These two approaches are also implemented in Nolds package with commands $nolds.hurst\_rs(data)$ and $nolds.dfa(data, order = 2, fit = 'RANSAC')$. Note that these commands can use more optional parameters, but for our work we need only a single parameter $data$ and DFA methods uses the polynomial order of two for detrending procedure. Furthermore, we used the RANSAC [1] fitting algorithm that seemed to give us more precise

[1] Stands for RANdom SAmple Consensus which is an iterative method that is robust in estimating parameters of a mathematical model from a set of data containing outliers.[43]

results but it must have been rerun several times because sometimes it could not cope with calculation of fit, so we used its average values from several runs.

The algorithm $nolds.hurst\_rs(data)$ returns a classical Hurst coefficient/exponent in range between 0 and 1. However, time series used must be stationary which was attained by using the log returns of stock prices.

On the other hand, the algorithm $nolds.dfa(data)$ can be applied also on the non-stationary time series, hence it is more versatile. The output value can be compared to the value of the Hurst coefficient gained by the former algorithm simply as H=DFA, if the process is stationary and H=DFA-1 for non-stationary process. The values should be similar but hardly identical due to different algorithm.

3.4.2 Lyapunov Exponent Estimation

The largest Lyapunov exponent was estimated by the algorithm of Rosenstein et al.[42] using the command $nolds.lyap\_r(data, emb\_dim = 14)$. Note that Nolds package provides also another way of calculating the Lyapunov exponents - $nolds.lyap\_e(data)$ which estimates the Lyapunov spectrum from which we can easily extract the largest Lyapunov exponent, because in case of the univariate time series there is only a single Lyapunov exponent. However, it is sensitive to proper choice of the parameters, so it we rather stuck to the first algorithm. After short parameters tuning we used the value of 14 for embedded dimension where it seemed to give us the consistent values - the smaller or larger values of embedded dimension for the same time series sometimes led to the both negative and positive values of the largest Lyapunov exponent, because time series in test sample showed very low mean frequency which caused some calculation difficulties.

3.4.3 Sample Entropy Estimation

In our thesis we estimated the Metric entropy throughout calculating the Sample entropy using a command $nolds.sampen(data, tolerance = 0.02)$. Although Sample entropy is not the same as the Approximate entropy, it is much less computationally intensive than latter type of the entropy. Furthermore, as described in Section 1.2.3, the Sample entropy is more consistent, so it does not require an extensive parameters tuning. The parameter for embedded dimension is defaulted to value of two and although the parameter describing tolerance is usually set to be some multiple of the standard deviation of time series - mostly $0,2 * standard\ deviation$, for Sample Entropy it may be suggested to leave it as a constant value. After some empirical tests we used a tolerance parameter with a constant value of 0,02 that gave is the consistent values.[44]

3.5 Linear Regression and Correlation

To find any relationship between the prediction quality and the predictability we used two common approaches - a simple linear regression between the average RMSE and a single independent variable (in our case namely Hurst Coefficient estimated by R/S and DFA analysis, Sample Entropy and the largest Lyapunov Exponent) and the correlation analysis.

Simple Linear Regression - as already suggested, we assume a relationship between single dependent endogenous variable (response) and single independent exogenous variable (predictor), so that the changes in dependent variable are caused by the changes in the explanatory variable and there is some linear relationship between these variables described by a theoretical regression function: [45]

$$y = \alpha + \beta x + \varepsilon, \tag{3.6}$$

where y is dependent variable and x is independent variable or predictor, parameter α is called the y-intercept, β is the slope of the equation and parameter ε is called random error term or residual that accounts for the changes in y that cannot be explained by the linear relationship with x and must satisfy four conditions:

1. E(ε)=0 - the average value of the residual is zero
2. var(ε)=const. - the variance of the residuals is constant (homoskedasticity)
3. cov($\varepsilon_i, \varepsilon_j$)=0 for all $i \neq j$ - the residual values are independent of each other
4. the residual values are normally distributed for any given value of x

The residuals are estimated from the linear regression model equation using the actual observations as a difference between the actual observed values y_i and estimated values $\hat{y}_i$, so for each pair of observations we get the i^{th} residual as follows:

$$\varepsilon_i = y_i - \hat{y}_i = y_i - (a + b * x). \tag{3.7}$$

The assessment of the model is done, firstly, by checking how proposed model fits the data (R-squared, the overall F-test, t-test and the Root Mean Square Error), and secondly, by validation if the residuals satisfy the conditions mentioned above.

Considering the nature of our data, the coefficient of determination R^2 (or R-squared) describing a portion of the total variation in the dependent variable that is explained by the variation in the independent variable, was of smaller importance here, but we rather took a look at the significance of the of regression coefficients described by t-test. The t-test (hypothesis test for the regression slope) is used to check if there is a statistically significant

linear relationship between an independent and the dependent variable,so the regression line slope coefficient b significantly differs from zero. Note that it is based on using Student's t-distribution. Thus the null and alternative hypotheses are as follows:

$H_0 : b = 0$

$H_1 : b \neq 0$

We used the p-value approach, so in statistical software SPSS we computed the p-value (cumulative probability that the theoretical t-score following the Student's t-test is more extreme than calculated value of t-score) and compare it with the chosen significance level. When p-value is $<$ than chosen significance level, then we reject the H_0, so that the slope of the regression line significantly differs from zero.

Then we have to examine if the residuals satisfy four well-known coditions:

1. E(ε)=0 - examined by the scatter plots of the residuals versus the predicted values, where the values of the residuals - should be randomly scattered around zero

2. var(ε)=const. - examined using the scatter plots of the residuals versus the predicted values variables - there should be no pattern.

3. cov($\varepsilon_i, \varepsilon_j$)=0 for all $i \neq j$ - examined using the scatter plots of the residuals versus the predicted values variables. No autocorrelation should be indicated by the absence of any pattern. Alternatively we can use the Durbin-Watson test [2].

4. normal distribution - examined using the histogram and normal probability plot.

Correlation Analysis - used to describe a linear relationship between two variables in a form of correlation coefficients that quantify its strength and direction. They range within an interval of [-1,1], so according to their value we distinguish between three basic cases:

1. $r(x,y) > 0$ - positive correlation - if $r(x,y) = +1$ indicates perfect positive correlation

2. $r(x,y) = 0$ - no correlation

3. $r(x,y) < 0$ - negative correlation - if $r(x,y) = -1$ indicates perfect negative correlation

Please note that the correlation coefficient describes only a linear relationship between two variables, i.e. the possible non-linear relationships are not captured, and the values closer to zero mean weaker linear relationship only.

[2] Let's have the residuals ε_i sorted in time order, then

$$DW = \frac{\sum_{i=1}^{n-1}(\varepsilon_i - \varepsilon_{i+1})^2}{\sum_{i=1}^{n}\varepsilon_i^2}.$$

DW < 2 means positive serial correlation, DW > 2 means negative serial correlation. However, it tests only the serial correlation. If the residuals are independent, the Durbin-Watson test value is around value of 2.[46].

In this thesis we used a Pearson correlation coefficient between two random variables. Formally, the Pearson product-moment correlation coefficient of two random variables is calculated as follows: [47]

$$r(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}. \tag{3.8}$$

Furthermore, because we performed a simple linear regression, so there were only a single regression coefficient, the Pearson correlation coefficient can be also calculated as a square root of the R-squared with the sign being identical to the sign of a slope of the line.

Chapter 4

The Results

In this chapter the overall results of this thesis are presented in the form of heatmaps, plots, tables and short discussion about them. At first, there are results of determining the most precise models of the LSTM which are applied in following part on a huge number (150) of various time series. Then there are the results conerning any possible relation between the LSTM performance characterized by the average RMSE calculated from the best five models and each of the predictability statistics characterized by the Hurst coefficient - via Rescaled Range analysis, DFA exponent, and then the largest Lyapunov exponent and Sample entropy.

In the whole thesis we used data from Yahoo! Finance - daily adjusting closing price of the stock in USD converted into their log returns with 2893 observations (since 1.1.2006 to 30.6.2017). Since more data means more information the LSTM can train on, we wanted to include pre-crisis and post-crisis period and also use a relatively recent data in our sample. However, more data also means more computing time, so we did not include too distant periods and it took approximately between 30 to 45 seconds for LSTM to train and then generate the prediction from a single dataset. The downloaded sample was divided into training sample consisting of 2314 observations, and test sample consisting of the rest 579 observations which is in the agreement with standardly used division of the original sample, such that testing sample contains approx. 20% of the observations.

4.1 The Best LSTM Models Selection

As previously mentioned, we tried to select five most precise LSTM models that could relatively accurate and effectively forecast all the stocks we used in the next part, so then we could compare their forecasting ability while being applied on a different time series with different degree of predictability. Basically, we can influence LSTM network forecasting ability either via changing its architecture, i.e. selecting the proper number of memory

blocks, or via training process, i.e. modulating the number of training epochs.

We can set the desired number of memory blocks in single layer by using the variable *numBlocks* in the script. The more memory blocks does not always mean the better forecasting capability, so we tried to run the script with several memory blocks and then chose the model (ideally) with the lowest RMSE. Adding the memory blocks has a small influence on computing time, so it is not a problem to have a LSTM with a large number of them.

On the other hands, adding the number of training epochs leads to a significant increase in computing time, so we do not prefer a very long lasting training. However, in general, the more training epochs we have, the smaller RMSE gets, although after some number of epochs the over-training can occur. Thus we must set some reasonable threshold RMSE that takes into account the fact that after some training the increments of refining the model become smaller and smaller, so its is not preferable to continue with further training. Furthermore, we must also consider the number of blocks in our selection process, that can cause the RMSE to be a different for even the same number of training periods.

For our selection process we chose eight stocks from different industries that should be representative to some extent, and thereby five selected LSTM models would be able to handle with the various stocks. Representative in our case means the one that could characterize the respecting industry like being typical for that sector, having a huge trading volumes etc. Those stocks were Citigroup Inc. (ticker ”C”), Wells Fargo & Company (ticker ”WFC”) for the financial sector, Pfizer Inc. (ticker ”PFE”) for the healthcare sector, Comcast Corporation (ticker ”CMCSA”) for the services sector, PG&E Corporation (ticker ”PCG”) for the sector of utilities, General Electric Company (ticker ‘”GE”) for the sector of industrial goods, Apple Inc. (ticker ”AAPL”) for the sector of consumer goods and the Micron Technology, Inc. (ticker ”MU”) for the technology industry.

The range of memory blocks of the LSTM was chosen empirically from the previous experience from two to ten, because too large number of the memory blocks has the problem that the network can be too much complex and cannot be properly used for our type of the problem.

Similarly, the range of the training epochs was chosen empirically from three to fifteen to capture the relation how increasing LSTM learning leads to the smaller marginal improvement. On the other hand, for some time series the small number of learning epochs is not enough to generate a sufficiently accurate model.

As a result we generated eight heatmaps, each for the respective stock, containing the value of RMSE, number of memory blocks and number of training epochs. To better visualize the low or high values we used the heatmaps where the specific colour from a given colour scheme is assigned to a given value of RMSE.

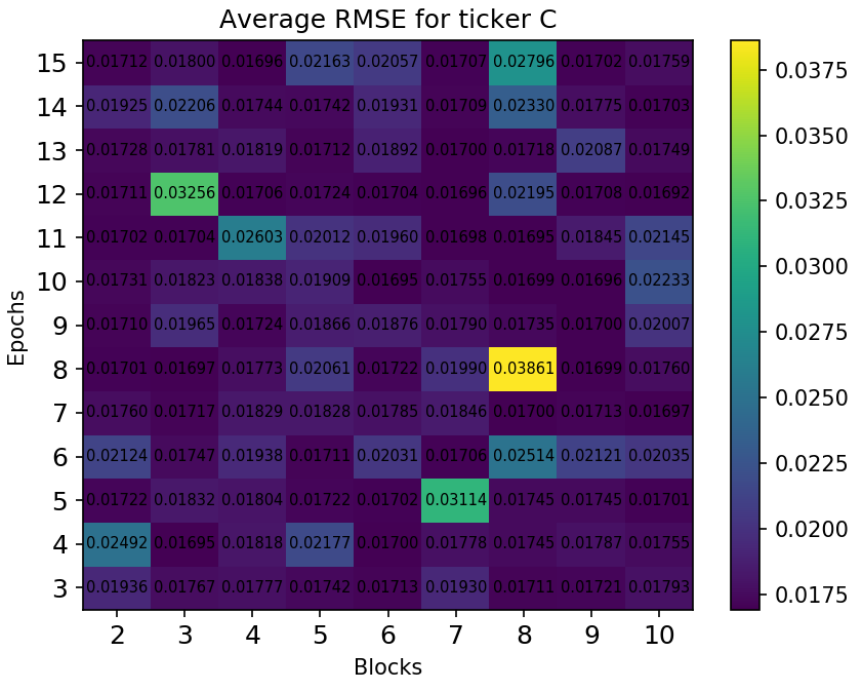

Figure 4.1: Heatmap of the RMSE for ticker C.

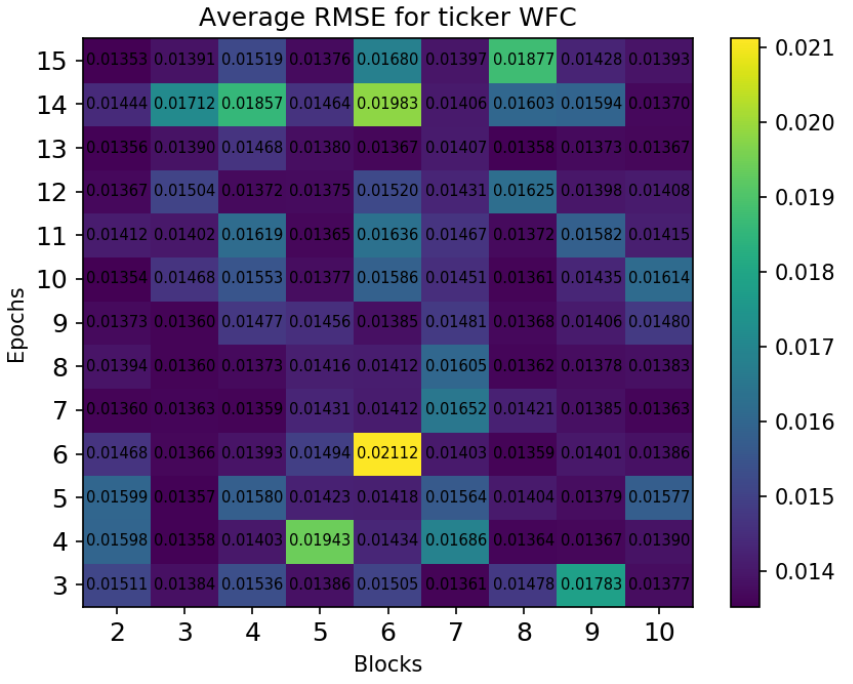

Figure 4.2: Heatmap of the RMSE for ticker WFC.

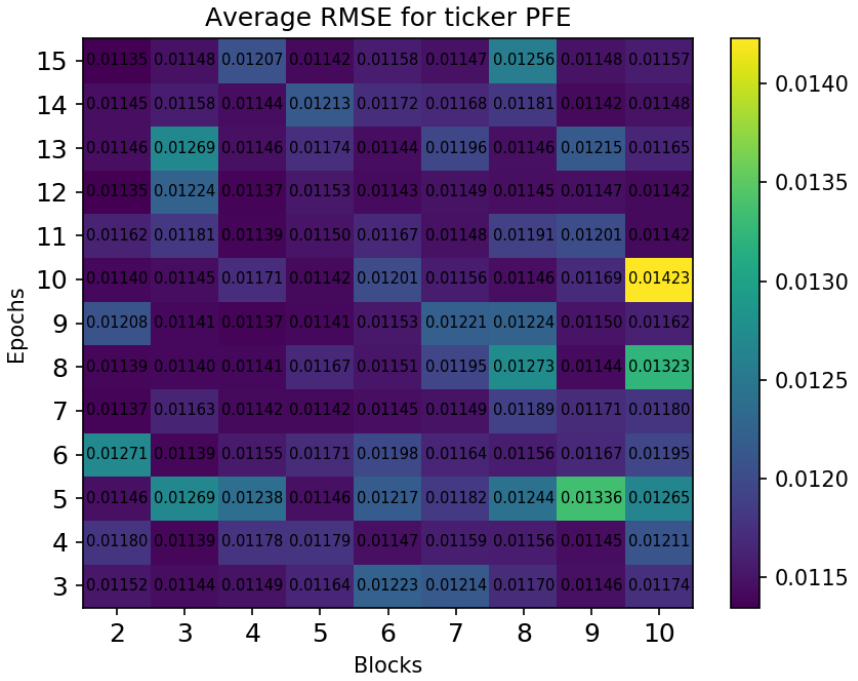

Figure 4.3: Heatmap of the RMSE for ticker PFE.

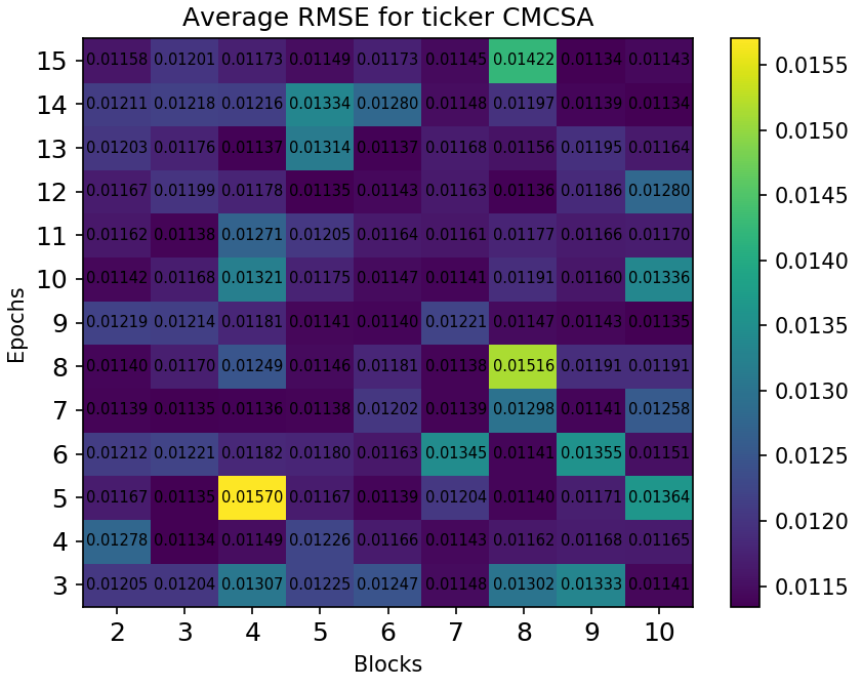

Figure 4.4: Heatmap of the RMSE for ticker CMCSA.

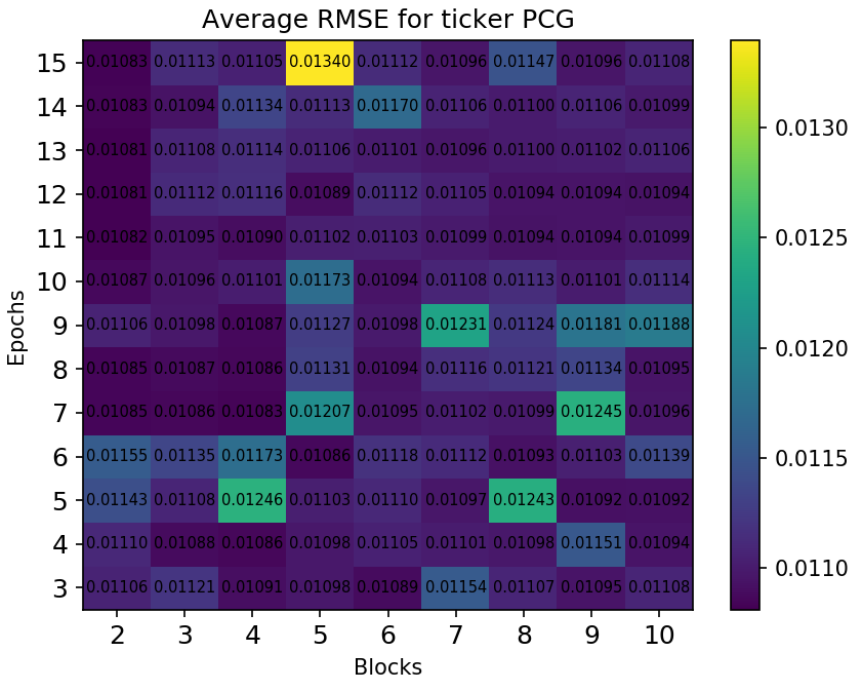

Figure 4.5: Heatmap of the RMSE for ticker PCG.

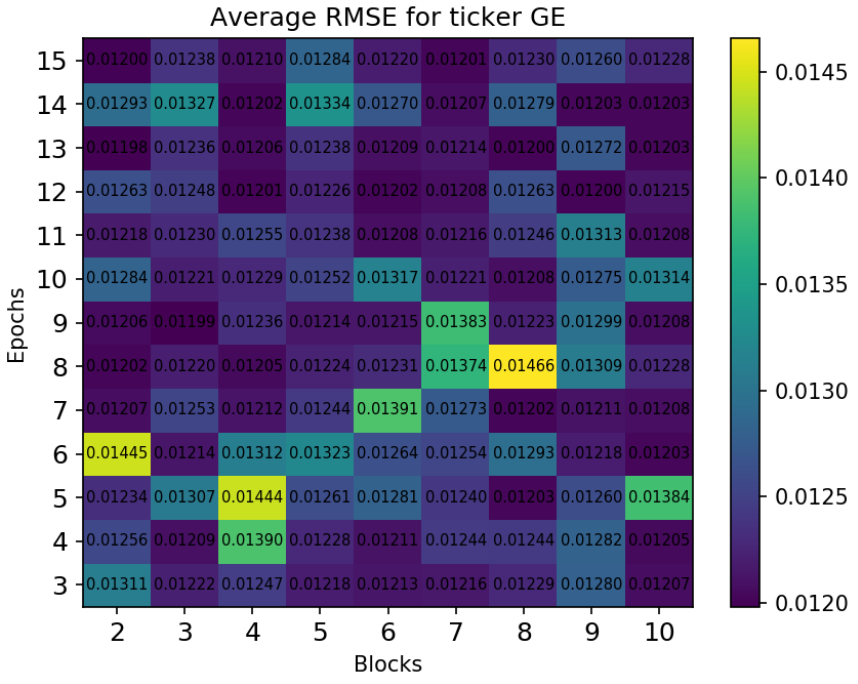

Figure 4.6: Heatmap of the RMSE for ticker GE.

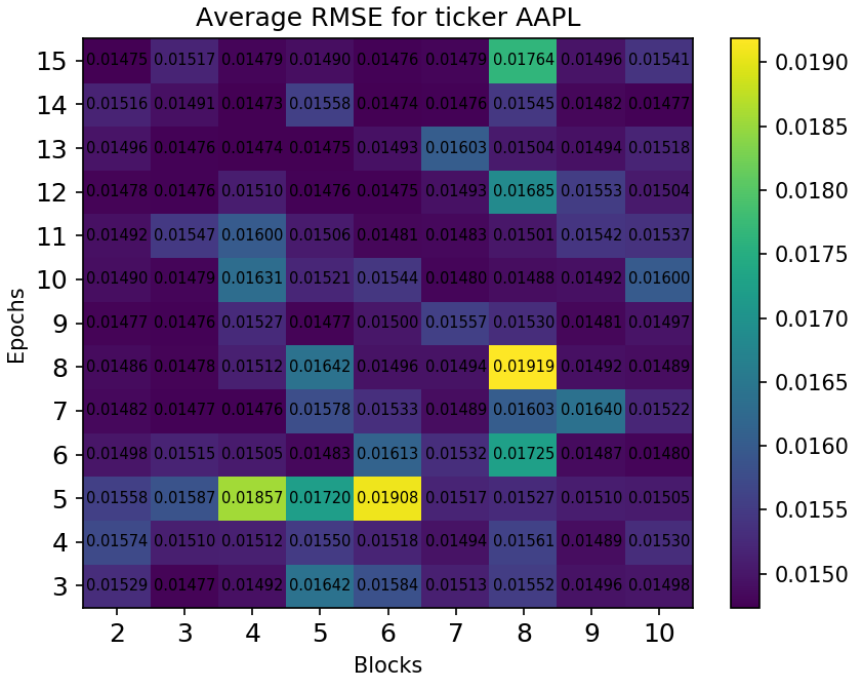

Figure 4.7: Heatmap of the RMSE for ticker AAPL.

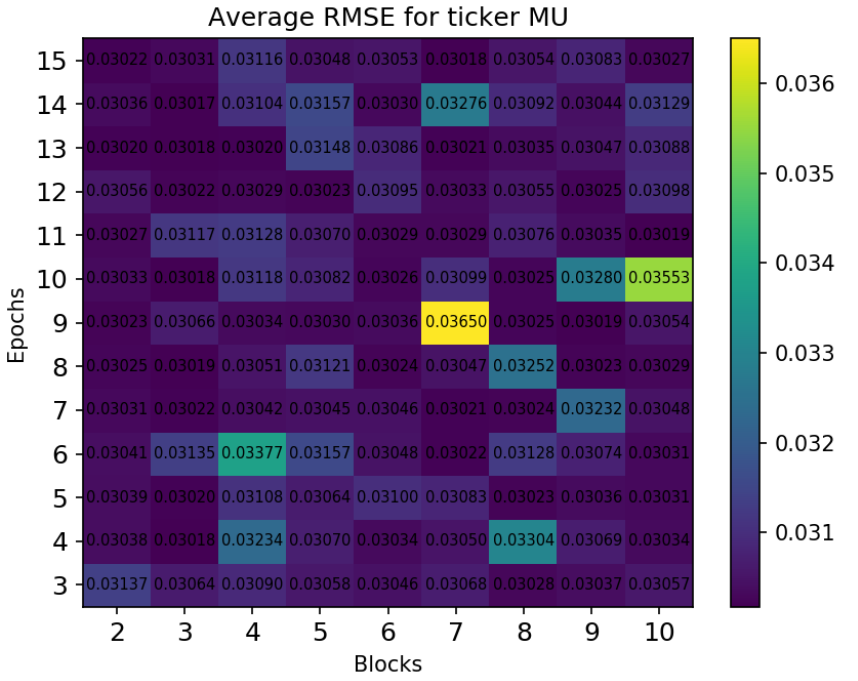

Figure 4.8: Heatmap of the RMSE for ticker MU.

Although a single heatmap gives us a quite sufficient view of the RMSE range, and thereby it is easy to identify which LSTM configuration has the lowest RMSE, the overall searching for the optimal architecture is more complex. We needed to choose the models that are able to predict the whole range of time series of log returns as precise as possible. As anticipated, for different stocks, the lowest RMSE is attainable by different LSTM network configuration as seen on table below.

Table 4.1: Stocks with their minimal RMSE.

Company	**Ticker**	**The lowest RMSE**	**Blocks**	**Epochs**
Citigroup Inc.	C	0,01696	7	12
Wells Fargo & Company	WFC	0,01353	2	15
Pfizer Inc.	PFE	0,01135	2	15
Comcast Corporation	CMCSA	0,01134	9	15
PG&E Corporation	PCG	0,01081	2	12
General Electric Company	GE	0,01198	2	13
Apple Inc.	AAPL	0,01474	4	14
Micron Technology, Inc.	MU	0,03017	3	14

Finally, we tried to determine the most precisel configurations of our LSTM network by calculating the average RMSE of the whole population consisting of all eight stocks. Its calculation was described in the Methodology chapter, and the results are below in table 4.2. The best five models configurations were the ones with the minimal average RMSE which are summarized in table below based on the overall heatmap on the next page.

Table 4.2: LSTM configurations with minimal RMSE.

Order	**Average RMSE**	**Blocks**	**Epochs**
1.	0,01632	2	15
2.	0,01633	3	4
3.	0,01634	3	8
4.	0,01636	2	8
5.	0.01637	7	15

Then we made 150 predictions of log returns of various stocks using these five LSTM networks configuration and subsequently compared their average RMSE with the predictability statistics.

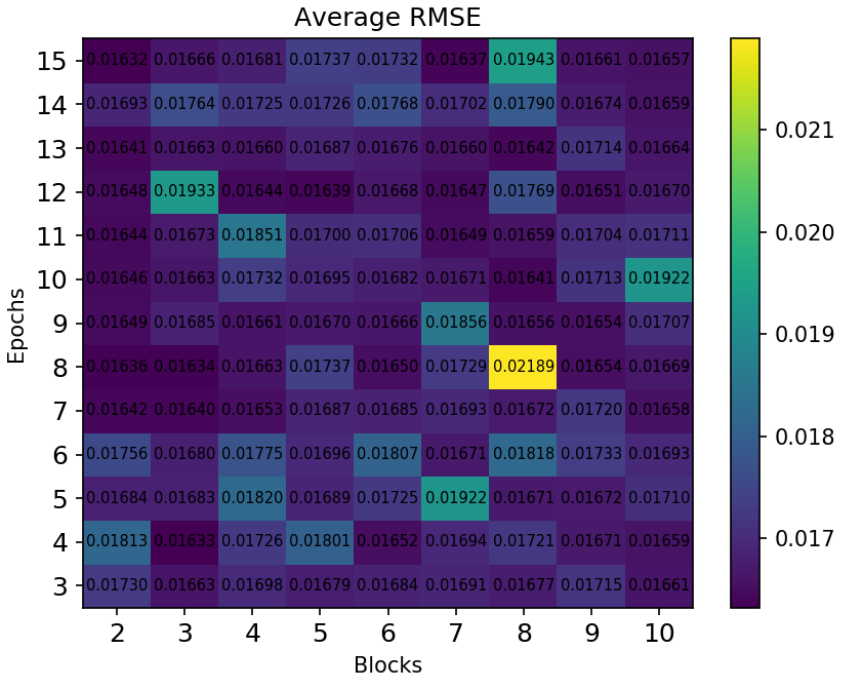

Figure 4.9: Heatmap of the average RMSE for all tickers.

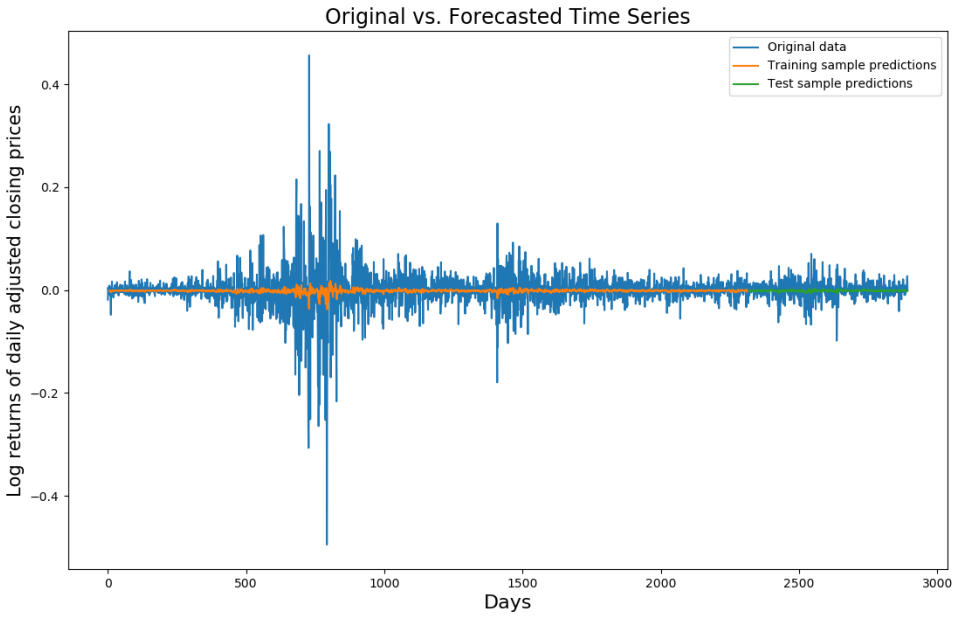

Figure 4.10: Forecast made by the most precise LSTM network for ticker C.

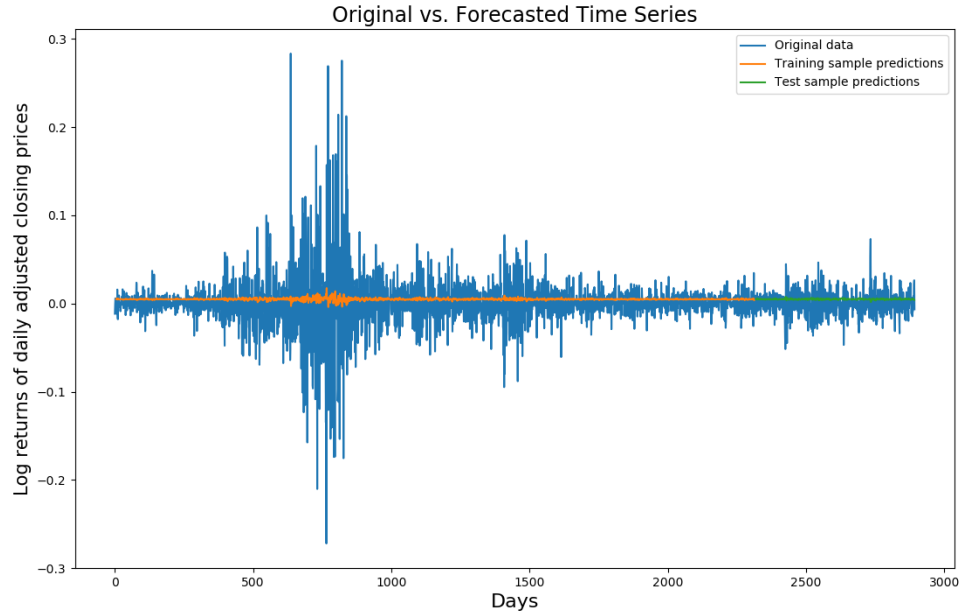

Figure 4.11: Forecast made by the most precise LSTM network for ticker WFC.

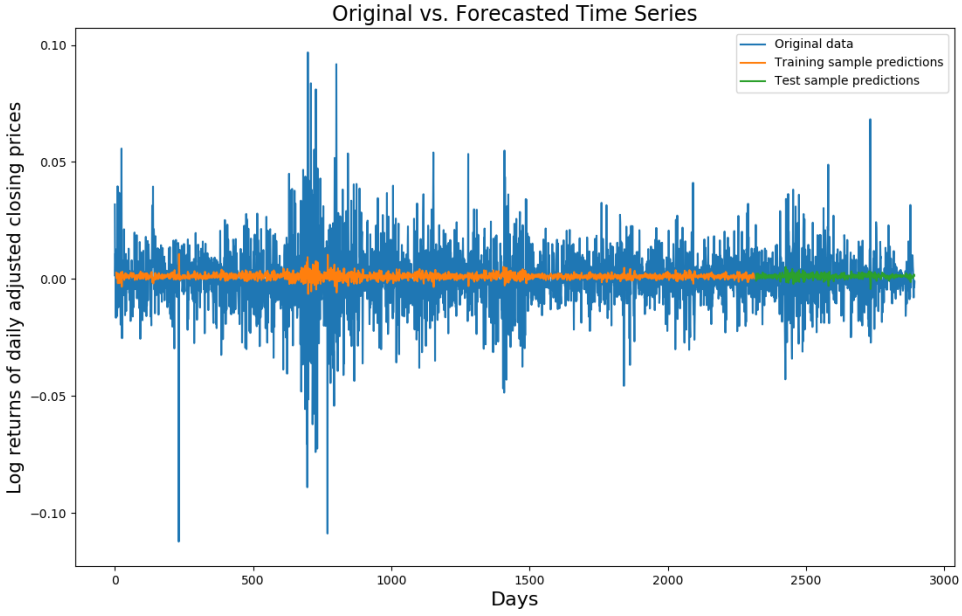

Figure 4.12: Forecast made by the most precise LSTM network for ticker PFE.

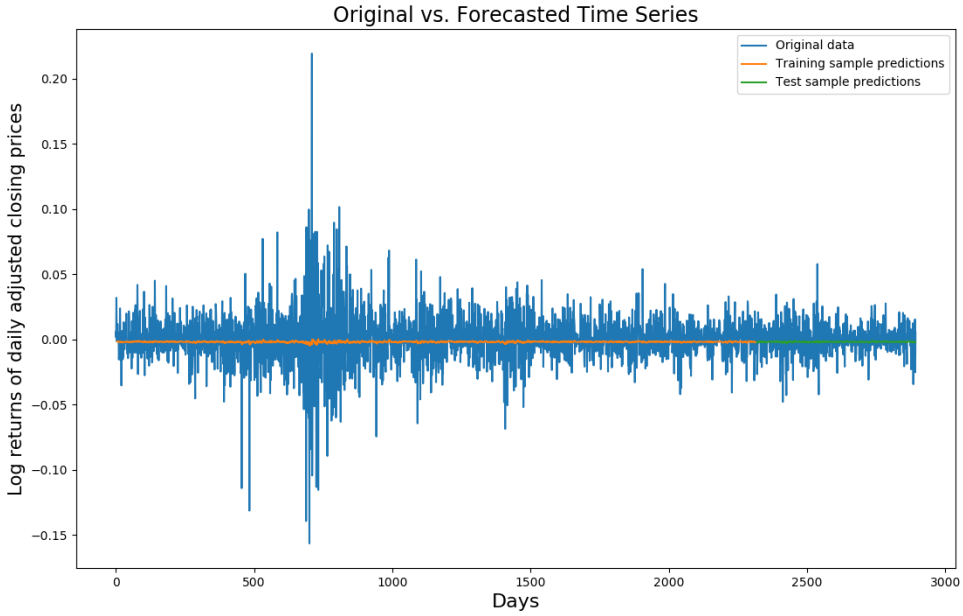

Figure 4.13: Forecast made by the most precise LSTM network for ticker CMCSA.

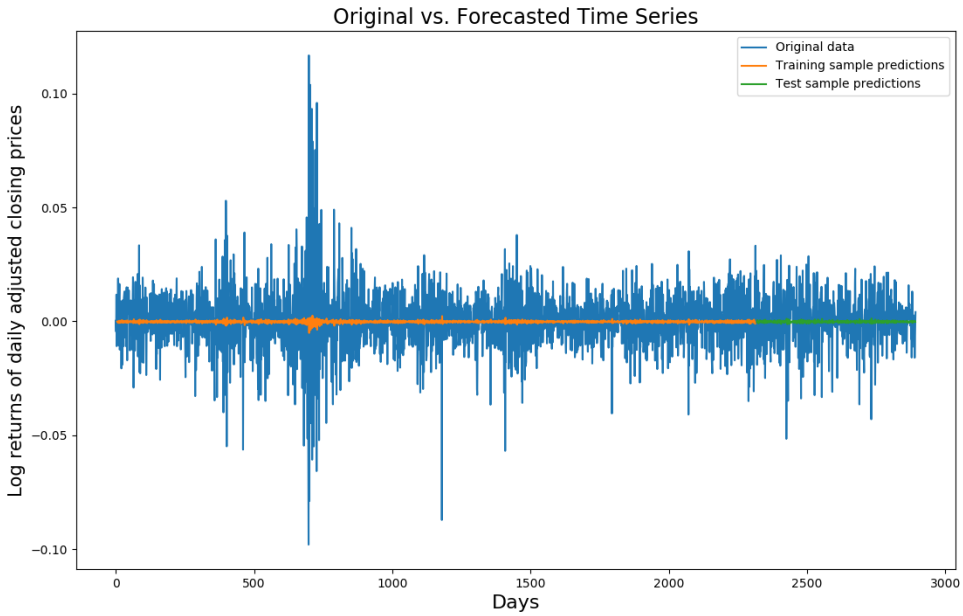

Figure 4.14: Forecast made by the most precise LSTM network for ticker PCG.

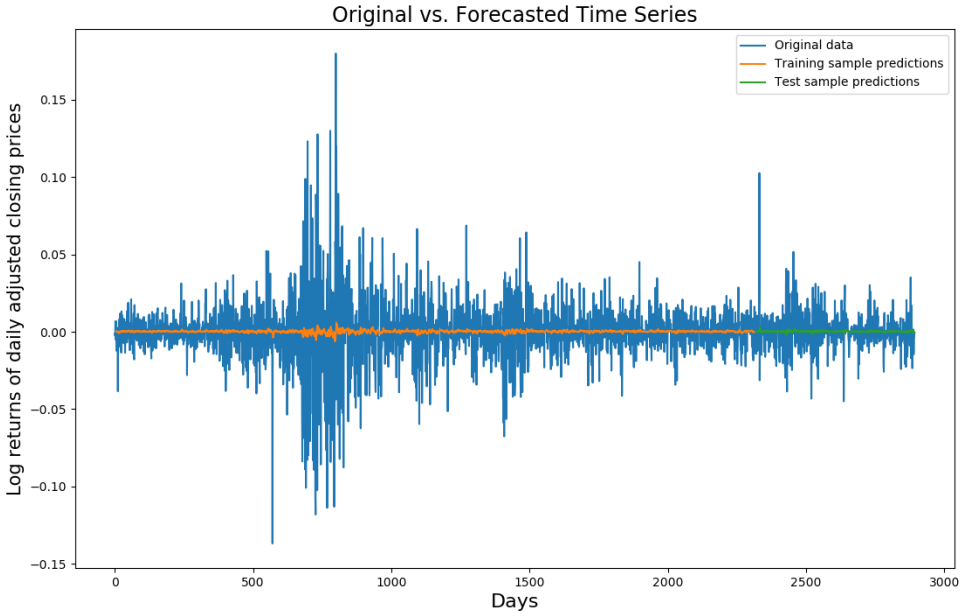

Figure 4.15: Forecast made by the most precise LSTM network for ticker GE.

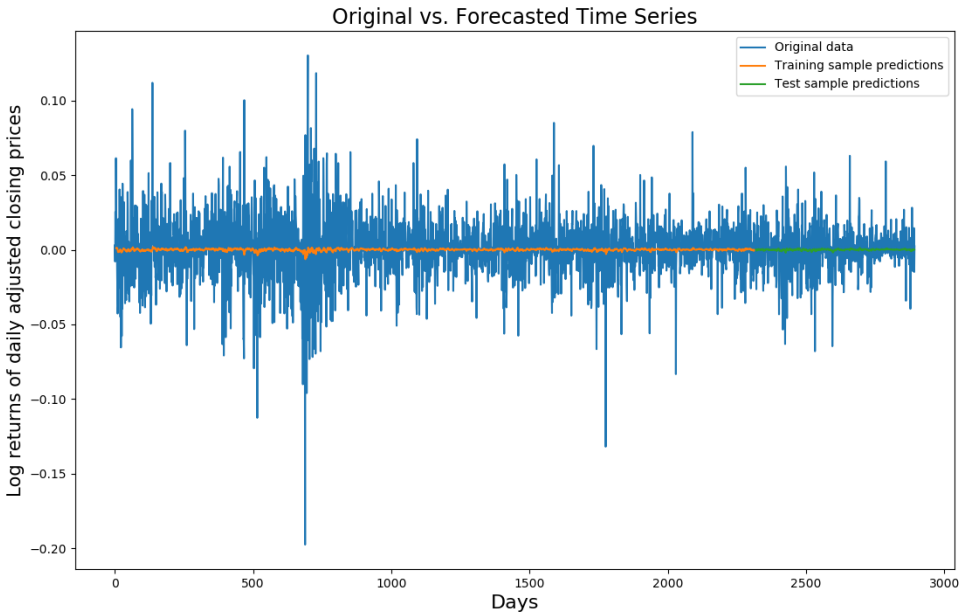

Figure 4.16: Forecast made by the most precise LSTM network for ticker AAPL.

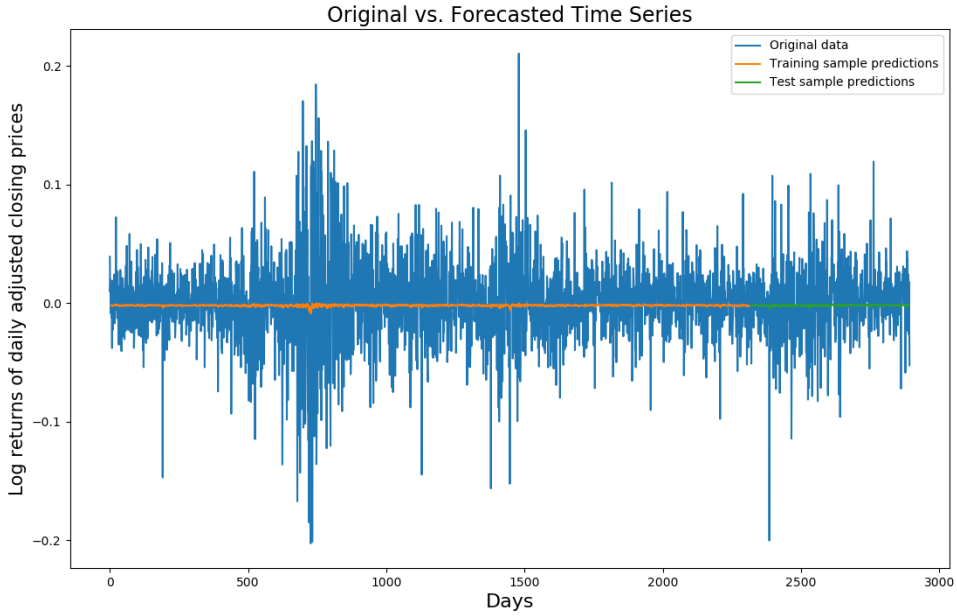

Figure 4.17: Forecast made by the most precise LSTM network for ticker MU.

4.2 Final Results and LSTM Networks Forecasts

In this section we provide the complete results for 150 tickers including the predictability statistics and the quality of fit attained by the five best LSTM models represented by the average RSME to achieve a comparability. Please note that the three statistics concerning the predictability - Hurst coefficient (calculated by R/S Analysis, DFA coefficient), Sample entropy and the largest Lyapunov exponent are calculated for test sample because the quality of prediction was calculated, obviously, on the test sample. These 150 tickers were selected randomly in a such way that they should have represent the all possible types of stocks in a given industry, i.e. stocks with high and low market capitalization, with high and low prices etc. Furttjermore, we also present the plots with forecast made by the most precise LSTM network for three stocks with the lowest average RMSE and for two stocks with the largest average RMSE (the third largest average RMSE was calculated for ticker MU which plot is presented in previous section). A table with results is presented on the next few pages.

Table 4.3: The complete results.

Ticker	Hurst (R/S)	Hurst (DFA)	Sample Entropy	Lyapunov Exp.	Average RMSE
KO	0,433362	0,443187	0,074153	0,031612	0,008476
CCA	0,423997	0,317632	0,086004	0,030915	0,009260
SO	0,471019	0,464852	0,118179	0,030975	0,009566
AFG	0,434640	0,464067	0,113748	0,032585	0,009864
MCD	0,462324	0,479140	0,103243	0,036631	0,010179
VZ	0,558248	0,665616	0,137737	0,033464	0,010216
MO	0,501747	0,449411	0,136761	0,030915	0,010302
XEL	0,492087	0,467948	0,156358	0,033335	0,010397
SR	0,500937	0,654029	0,161954	0,029832	0,010399
K	0,450584	0,435835	0,148735	0,032530	0,010542
AEP	0,491104	0,465843	0,169735	0,034324	0,010723
ED	0,494267	0,527142	0,162984	0,030797	0,010734
NEE	0,449724	0,462785	0,159092	0,031353	0,010830
AON	0,392178	0,370315	0,135128	0,032708	0,010957
PCG	0,446947	0,514053	0,175842	0,030665	0,010981
UTX	0,528158	0,612052	0,139535	0,026877	0,011034
PPL	0,478234	0,496162	0,159355	0,031518	0,011058
DUK	0,537218	0,618877	0,156475	0,026697	0,011064
CMS	0,461502	0,544036	0,159934	0,032479	0,011074
WEC	0,463311	0,596130	0,175385	0,031514	0,011205
AFL	0,469069	0,443591	0,127926	0,032793	0,011289
GSK	0,417554	0,517509	0,188517	0,031609	0,011347
PFE	0,500051	0,511348	0,148566	0,030666	0,011450
MORN	0,540198	0,553426	0,159901	0,032818	0,011481
CMCSA	0,505444	0,589872	0,182327	0,030852	0,011494
DIS	0,499716	0,447105	0,140663	0,030039	0,011657
LNT	0,450112	0,630596	0,179517	0,028785	0,011720
NHC	0,519410	0,582883	0,204302	0,032874	0,011853
PEG	0,468216	0,453717	0,228280	0,030977	0,012122
WMT	0,446933	0,442128	0,159926	0,032946	0,012176
GE	0,493760	0,449361	0,159842	0,030898	0,012335
NI	0,473970	0,454635	0,246026	0,023250	0,012503
UTL	0,468826	0,442331	0,221447	0,027045	0,012706

PSA	0,477902	0,617624	0,221808	0,033827	0,012725
USB	0,457614	0,446673	0,206774	0,032304	0,013008
MRK	0,403960	0,456853	0,195013	0,030636	0,013012
ORCL	0,426034	0,440102	0,186657	0,033886	0,013065
MGEE	0,457597	0,628709	0,277713	0,031147	0,013148
SIRI	0,491985	0,569696	0,227789	0,033922	0,013168
BNS	0,520160	0,535347	0,225270	0,032383	0,013172
SBUX	0,463202	0,564864	0,210700	0,031522	0,013195
ICE	0,503530	0,479323	0,209547	0,040716	0,013502
IPG	0,441656	0,515585	0,253477	0,034483	0,013598
BA	0,471835	0,565301	0,234115	0,030118	0,013748
ABT	0,527650	0,465620	0,224195	0,035416	0,013792
CSCO	0,552415	0,562640	0,179407	0,033106	0,013848
WFC	0,497858	0,531927	0,264038	0,034048	0,013889
HRL	0,498282	0,447156	0,238772	0,034658	0,013916
BBT	0,471701	0,484956	0,263371	0,026059	0,013926
AVX	0,385966	0,558174	0,307718	0,027937	0,013926
PNC	0,438588	0,450357	0,263088	0,033621	0,013973
IR	0,486695	0,694757	0,246655	0,034685	0,014014
EMR	0,451647	0,642847	0,306361	0,031170	0,014090
AIG	0,465313	0,538013	0,185069	0,030479	0,014095
EXC	0,512962	0,651267	0,284983	0,033243	0,014125
ESE	0,494858	0,537665	0,252730	0,032445	0,014139
NKE	0,513825	0,490032	0,255100	0,032227	0,014206
WINA	0,514655	0,415615	0,291698	0,031753	0,014320
ETN	0,447863	0,501975	0,282423	0,027163	0,014446
TWX	0,524523	0,590530	0,174194	0,023300	0,014479
UNP	0,487072	0,564409	0,312720	0,028896	0,014574
PLD	0,493924	0,600394	0,229968	0,033917	0,014590
ADBE	0,417397	0,560037	0,231366	0,035069	0,014651
SNY	0,487415	0,490842	0,324959	0,034036	0,014744
GOOG	0,486607	0,537771	0,236183	0,036634	0,014862
MDLZ	0,473441	0,602011	0,272888	0,038151	0,014911
INTU	0,499180	0,570137	0,204318	0,040369	0,014954
JCI	0,482266	0,561938	0,274919	0,035130	0,015006

NPK	0,487825	0,437116	0,366101	0,031334	0,015031
HSBC	0,503361	0,600279	0,277141	0,032095	0,015076
AAPL	0,526475	0,476089	0,270422	0,034661	0,015078
FOXA	0,479244	0,508016	0,276100	0,033920	0,015232
AOS	0,532658	0,548325	0,279465	0,028470	0,015268
GGP	0,499185	0,551008	0,330570	0,035128	0,015291
ANTM	0,492488	0,671903	0,326342	0,031802	0,015303
CBS	0,494131	0,560235	0,324938	0,028685	0,015310
BSX	0,398664	0,392395	0,273822	0,030918	0,015339
XLNX	0,530187	0,484864	0,271616	0,031409	0,015353
BK	0,502064	0,475034	0,249207	0,029427	0,015359
HIG	0,428160	0,494276	0,219022	0,032321	0,015388
RELL	0,431947	0,485968	0,286429	0,032947	0,015832
EML	0,471130	0,325828	0,322948	0,027396	0,015898
CWT	0,507431	0,441634	0,339508	0,032435	0,016131
AET	0,500364	0,467471	0,319937	0,029868	0,016190
GS	0,509846	0,503546	0,326315	0,027404	0,016281
PBH	0,501974	0,517238	0,314394	0,033746	0,016289
COLB	0,471764	0,563211	0,391887	0,033886	0,016445
F	0,485647	0,611914	0,290268	0,030343	0,016559
USLM	0,464353	0,369749	0,305762	0,033034	0,016826
MGIC	0,489277	0,551613	0,368022	0,037121	0,016932
CRWS	0,444272	0,361965	0,290990	0,036465	0,016964
C	0,500498	0,527844	0,343508	0,027571	0,017200
NWL	0,428407	0,481449	0,267096	0,038037	0,017211
LEN	0,540863	0,454047	0,405416	0,028167	0,017217
RHT	0,501975	0,612892	0,296823	0,031227	0,017359
DDR	0,486234	0,547826	0,293681	0,030800	0,017389
KAI	0,481499	0,446780	0,379844	0,035898	0,017478
PHM	0,522078	0,502782	0,407447	0,030903	0,017543
CSGP	0,493272	0,525712	0,325839	0,029376	0,017565
SHG	0,505253	0,449269	0,470099	0,029770	0,017570
EBAY	0,459149	0,534346	0,313000	0,028885	0,017643
NVO	0,496537	0,570761	0,351189	0,040574	0,017690
KEY	0,459500	0,470576	0,405266	0,027032	0,017702

BH	0,519310	0,494661	0,433178	0,034500	0,017708
ATNI	0,509559	0,478370	0,378196	0,033437	0,017847
KR	0,504622	0,598622	0,316003	0,034495	0,017928
JNPR	0,471046	0,509273	0,305893	0,028379	0,018049
ARTNA	0,552311	0,524170	0,407925	0,035130	0,018058
TV	0,413000	0,552981	0,434919	0,036107	0,018086
KB	0,488792	0,594369	0,490299	0,032379	0,018453
CTL	0,478818	0,564454	0,343039	0,036390	0,018557
MS	0,502492	0,478037	0,394251	0,027930	0,018711
BAC	0,521108	0,604444	0,406645	0,028188	0,018727
YORW	0,456823	0,526960	0,467503	0,030341	0,018893
DISCA	0,463078	0,523638	0,464229	0,036107	0,019030
QCOM	0,469125	0,516821	0,299765	0,030195	0,019076
GENC	0,450354	0,528242	0,440521	0,029770	0,019260
AMSWA	0,473550	0,406770	0,419651	0,038489	0,019363
AGN	0,538112	0,536278	0,363474	0,031034	0,019816
MTU	0,465508	0,588587	0,462148	0,034500	0,019884
SNE	0,479773	0,456999	0,439646	0,030341	0,019926
CNO	0,485588	0,544452	0,447650	0,034997	0,019939
WWW	0,494770	0,531928	0,510716	0,028780	0,019973
WF	0,427593	0,450599	0,529767	0,030718	0,020212
SVA	0,509320	0,707567	0,239734	0,038439	0,020443
DJCO	0,437016	0,421086	0,495992	0,032965	0,020488
CTS	0,458111	0,517807	0,374783	0,031712	0,020535
JHX	0,469123	0,538633	0,493520	0,032610	0,020576
PLCE	0,494525	0,571650	0,507571	0,035404	0,020713
MTSC	0,493140	0,573123	0,380173	0,035057	0,020858
RF	0,506736	0,454774	0,462132	0,032379	0,021017
AJRD	0,499635	0,593897	0,483295	0,034997	0,021522
KBH	0,499056	0,498459	0,543712	0,035065	0,021534
TS	0,478073	0,494825	0,596556	0,027878	0,021690
JBLU	0,491262	0,490979	0,597891	0,033503	0,021817
IBN	0,457453	0,554871	0,562046	0,032234	0,021829
HNP	0,424122	0,428320	0,593109	0,026436	0,022244
ERJ	0,468034	0,517185	0,602687	0,039733	0,022245

GILT	0,494564	0,505115	0,525823	0,030855	0,022580
SAN	0,491838	0,558506	0,582347	0,035347	0,023161
GSH	0,510465	0,545509	0,516780	0,033700	0,023504
MYL	0,479425	0,516011	0,582450	0,030735	0,024966
BCS	0,452041	0,563272	0,541090	0,030844	0,025403
TI	0,534818	0,477052	0,645743	0,031971	0,025445
ATRO	0,512339	0,547543	0,627950	0,035011	0,027473
DB	0,525748	0,579358	0,714194	0,034432	0,027617
ACY	0,469204	0,484010	0,648541	0,037256	0,029403
MU	0,472787	0,528408	0,796248	0,028796	0,030317
CBI	0,513612	0,521579	0,678456	0,035664	0,030738
NYNY	0,510600	0,536193	0,822532	0,036508	0,035085

As seen on table with complete results, the lowest average RMSE of test sample was achieved for prediction of stock with ticker KO - The Coca-Cola Company - 0,00847601. On the contrary, the largest RMSE, and thereby the worst forecast was made for stock with ticker NYNY - Empire Resorts, Inc. - 0,0350849936. On the next pages there are forecast plots for the best and worst predictions.

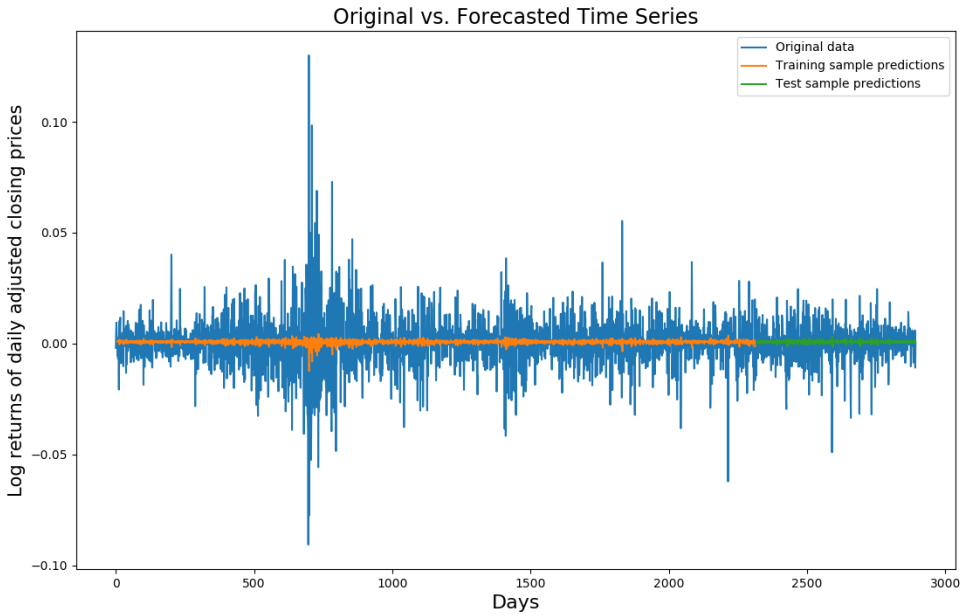

Figure 4.18: Stock with the most accurate forecast (KO).

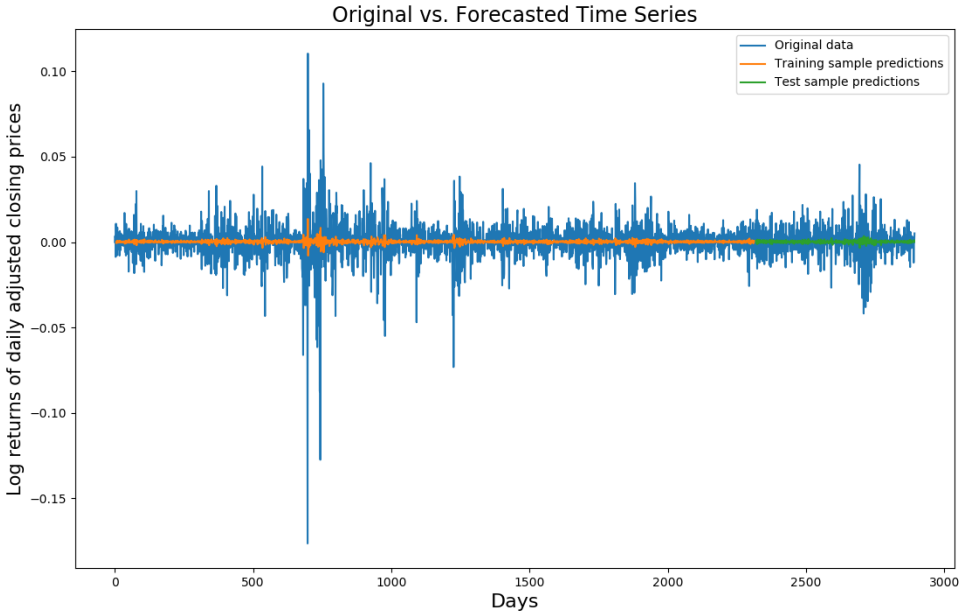

Figure 4.19: Stock with the second most accurate forecast (CCA).

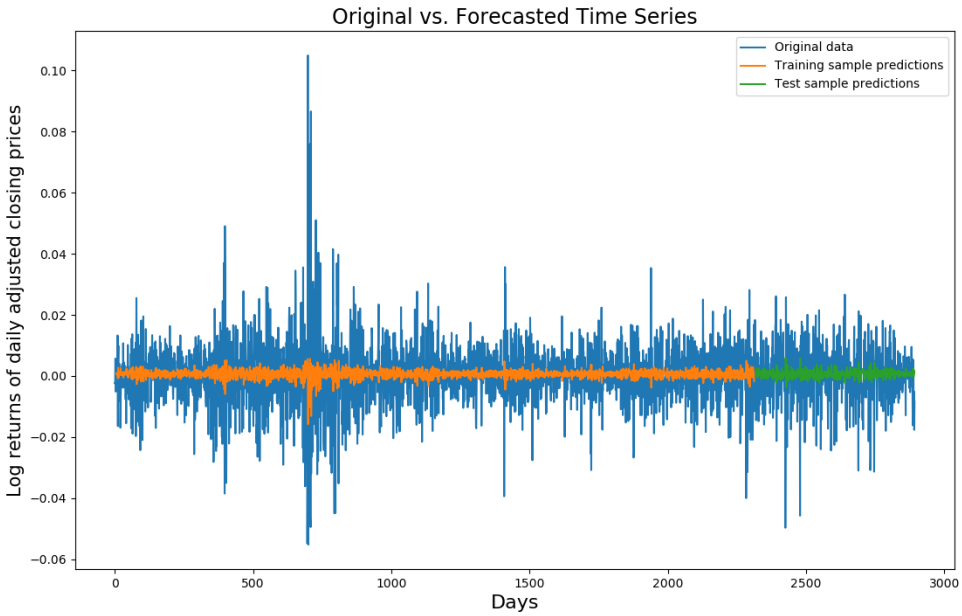

Figure 4.20: Stock with the third most accurate forecast (SO).

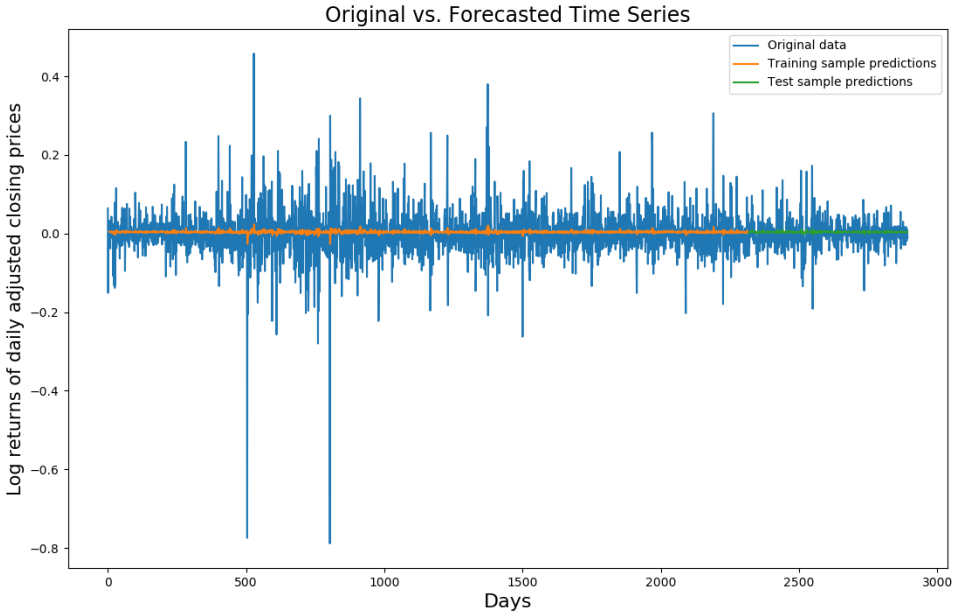

Figure 4.21: Stock with the least accurate forecast (NYNY).

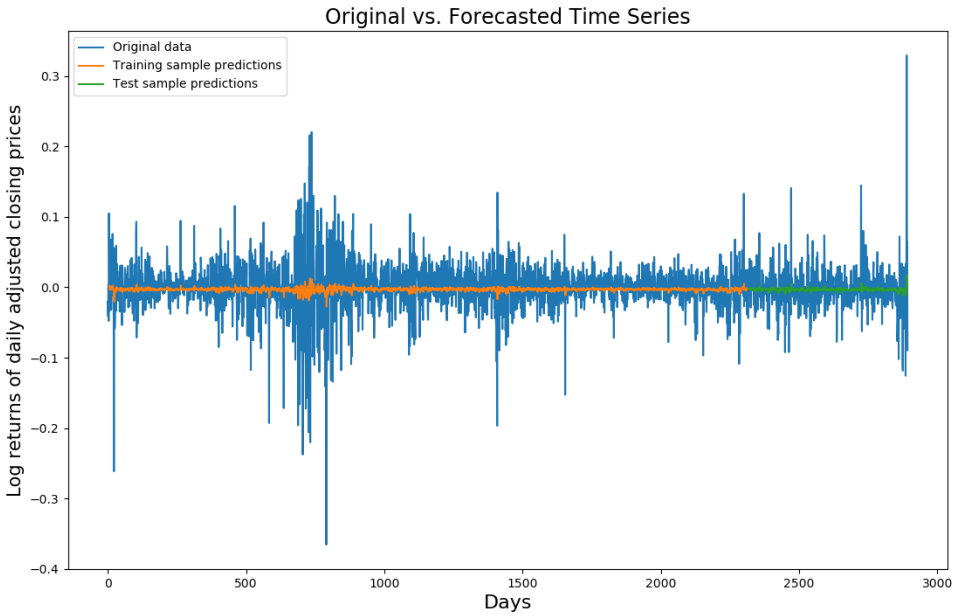

Figure 4.22: Stock with the second least accurate forecast (CBI).

4.3 Quality of Prediction vs. Predictability Statistics

Finally, we tried to determine a possible relationship, if any, between the calculated average RMSE and Hurst coefficient (estimated by R/S Analysis and DFA), Sample entropy and the largest Lyapunov exponent. At first, we generated four simple plots with error bars below to visually inspect for any relationships.

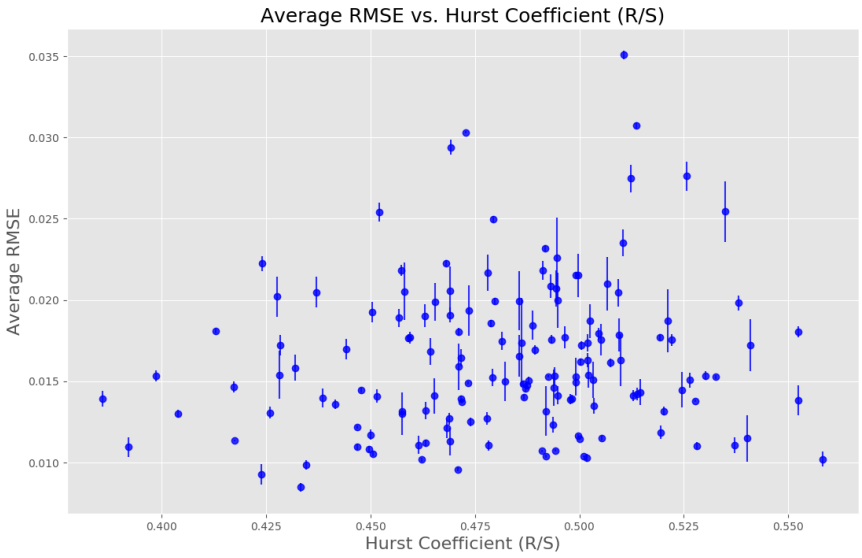

Figure 4.23: Average RMSE vs. Hurst coefficient estimated using R/S Analysis.

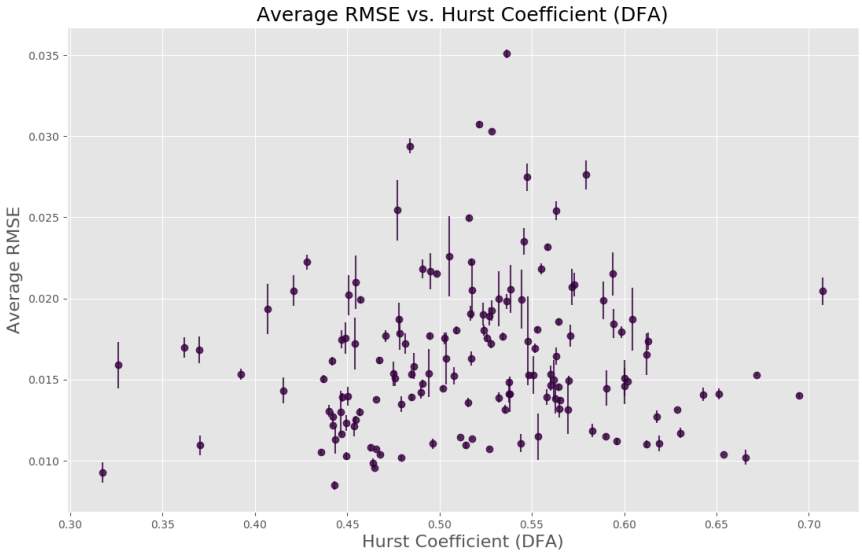

Figure 4.24: Average RMSE vs. Hurst coefficient estimated using DFA.

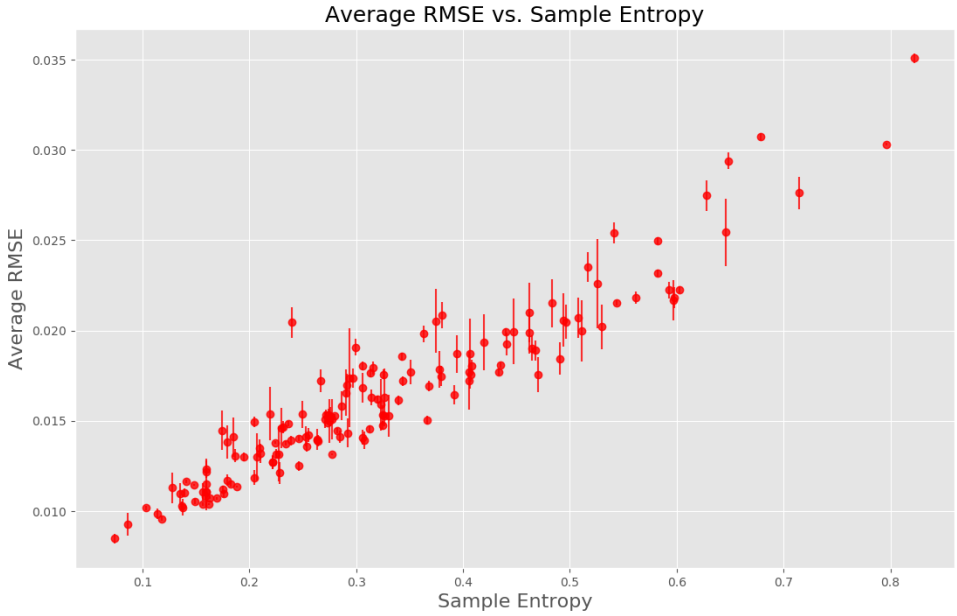

Figure 4.25: Average RMSE vs. Sample Entropy.

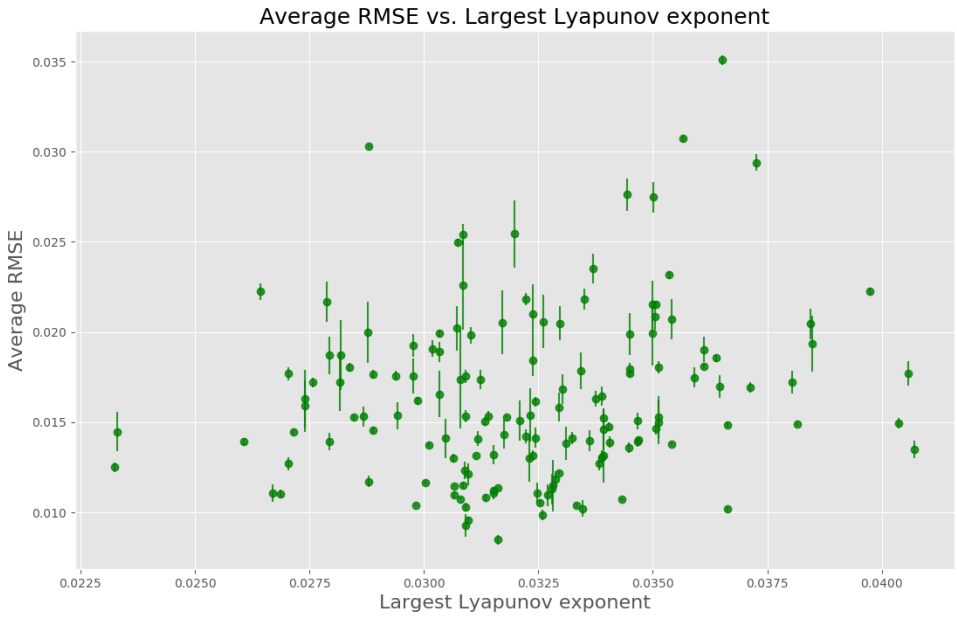

Figure 4.26: Average RMSE vs. Lyapunov Exponent.

The graphs 4.23 - 4.26 clearly indicate that there might be some linear relationship between the average RMSE and Sample Entropy, but the relationship between the rest of the measures of predictability is not that detectable. To asses the relationship between the average RMSE and the Hurst Coefficient we tried to estimate some simple regression curve on a distance of Hurst coefficient from value of 0,5 (completely random time series). In general, to estimate any relationship between forecasting precision and the predictability statistics we performed in SPSS a simple regression analysis to find some regression models - linear, quadratic and exponential - that can describe how level of the average RMSE is associated to the value of aforementioned measures of predictability with the results presented on next few pages. Note that in reality the forecasting precision depends not only on a single measure of predictability we used, because they deal with the (slightly) different aspect of what can influence the prediction quality, but we wanted to study the relationship between the average RMSE and a single factor concerning the forecastability. Thus these regression models were developed for some rough explanatory purposes and not for any precise forecasts.

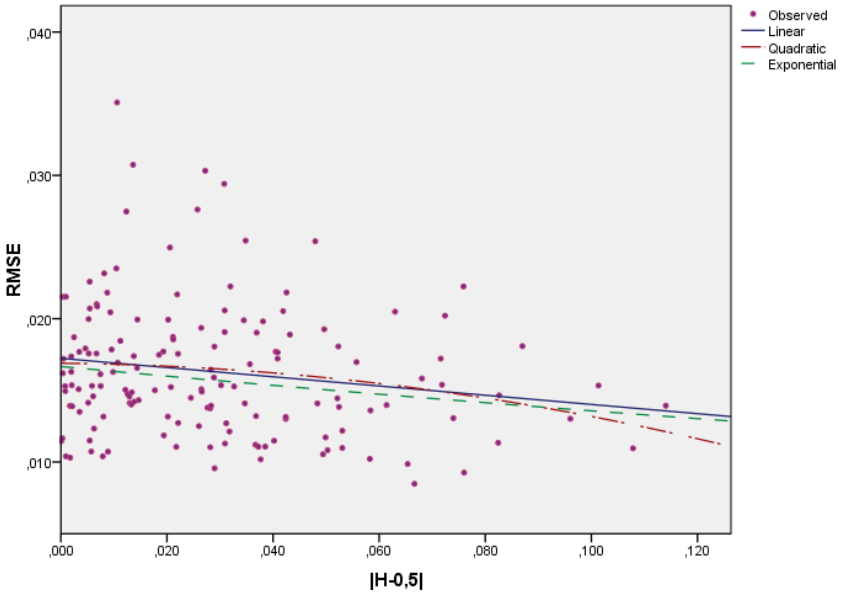

Figure 4.27: Fitted plot of RMSE vs. distance of Hurst coefficient (R/S) from 0,5.

Model Summary and Parameter Estimates

RMSE

Equation	Model Summary					Parameter Estimates		
	R Square	F	df1	df2	Sig.	Constant	b1	b2
Linear	,029	4,460	1	148	,036	,017	-,032	
Quadratic	,033	2,478	2	147	,087	,017	-,003	-,339
Exponential	,035	5,426	1	148	,021	,017	-2,052	

Figure 4.28: Fitting results - linear, quadratic and exponential trend.

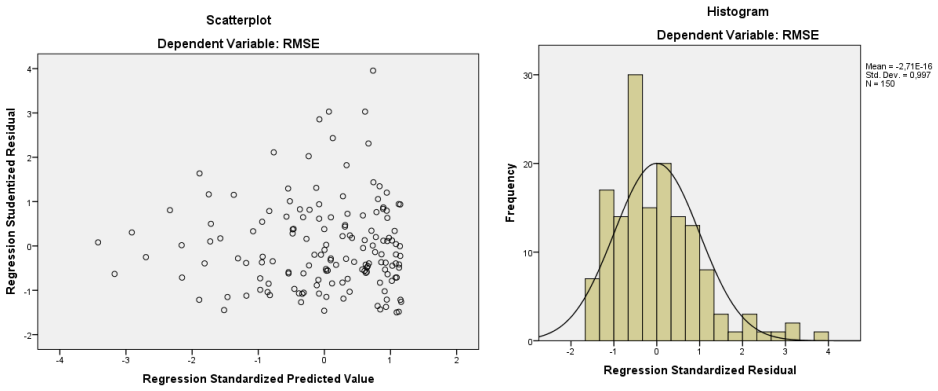

Figure 4.29: Residuals diagnostics for exponential trend.

Although it is evident that the values of RMSE tend to be higher around value of 0,5 for Hurst coefficient, they are more dispersed when time series is random. Thus it is a little bit difficult to find some simple regression model because the LSTM can achieve a relatively low RMSE even when the Hurst coefficient is close to 0,5. To get some approximate results we tried a linear, quadratic and exponential fitting line presented in table above. Because all of them show a decreasing tendency and their coefficients are statistically significant, we can state that the more distant values of Hurst coefficient are, the smaller average RMSE our LSTM models tended to achieve. The largest R-squared was achieved by exponential trend which p-value was 0,0016 for *constant* and 0,0354 for $b1$. The residuals analysis showed that the assumptions about the residuals were clearly not satisfied.

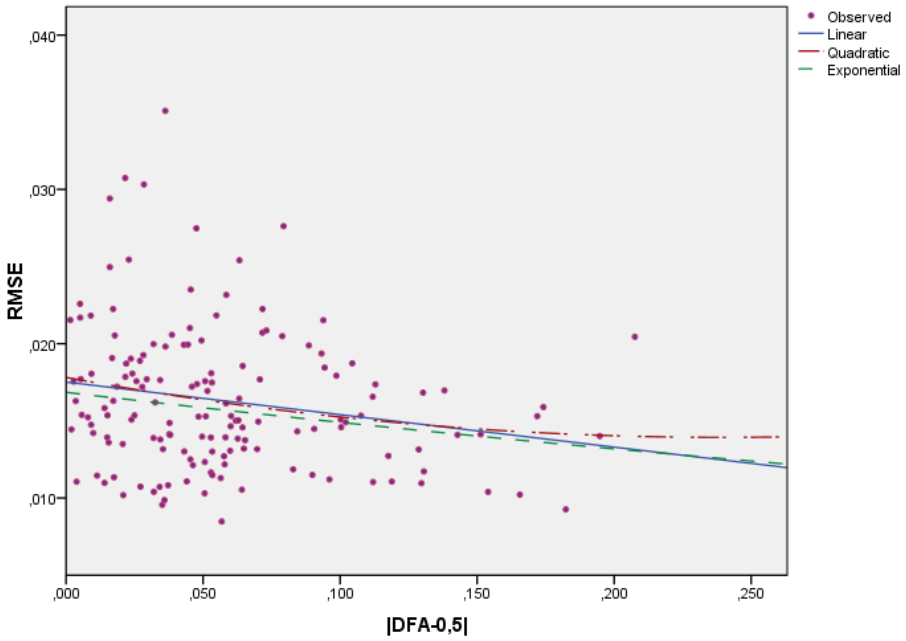

Figure 4.30: Fitted plot of RMSE vs. distance of Hurst coefficient (DFA) from 0,5.

Model Summary and Parameter Estimates

RMSE

Equation	Model Summary					Parameter Estimates		
	R Square	F	df1	df2	Sig.	Constant	b1	b2
Linear	,038	5,871	1	148	,017	,018	-,021	
Quadratic	,039	3,019	2	147	,052	,018	-,032	,068
Exponential	,038	5,911	1	148	,016	,017	-1,227	

Figure 4.31: Fitting results - linear, quadratic and exponential trend.

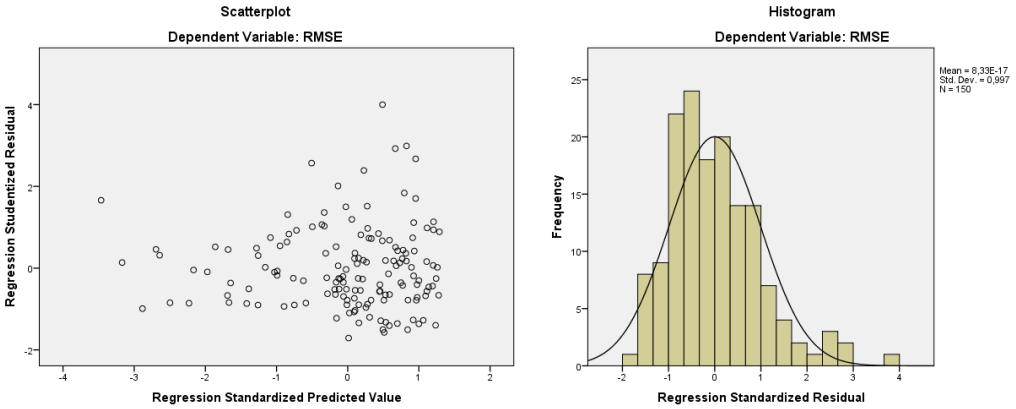

Figure 4.32: Residuals diagnostics for linear trend.

Likewise in the previous case, there is a larger dispersion of RMSE values around DFA with values close to 0,5. Again we tried a linear, quadratic and exponential fitting line presented in table above, and all of them show a decreasing tendency, so we can state that the more distant values of Hurst coefficient are, the smaller average RMSE our LSTM models tended to achieve. The most appropriate model was one with the linear trend which parameters had p-value very close to zero for *constant* and 0,0166 for $b1$, but the assumptions about the residuals were not satisfied in this case, but for some rough model with significant parameters, it is satisfactory even though all R-squared values are low.

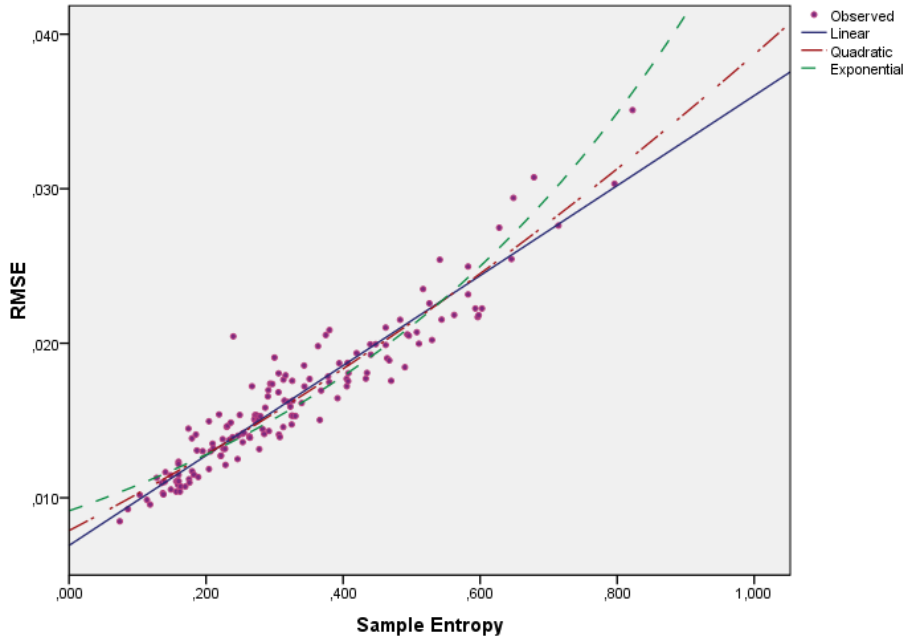

Figure 4.33: Fitted plot of RMSE vs. Sample Entropy.

Model Summary and Parameter Estimates

RMSE

Equation	Model Summary					Parameter Estimates		
	R Square	F	df1	df2	Sig.	Constant	b1	b2
Linear	,900	1332,619	1	148	,000	,007	,029	
Quadratic	,902	679,766	2	147	,000	,008	,023	,008
Exponential	,883	1115,720	1	148	,000	,009	1,673	

Figure 4.34: Fitting results - linear, quadratic and exponential trend.

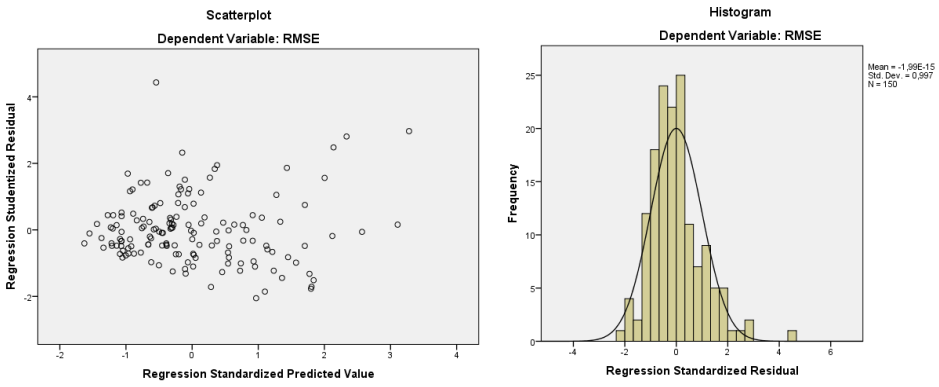

Figure 4.35: Residuals diagnostics for linear trend.

It is clear from the graph that the average RMSE increases with the Sample Entropy. All three fitting lines - a linear, quadratic and exponential one were calculated with the statistically significant F-tests. The best model was the one containing the quadratic trend line with high R-squared of 0,92, but its parameter $b2$ was insignificant with value of 0,0597, so we continued with an analysis of the model with linear fit which coefficient were significant (their p-values were very close to zero). The residuals analysis showed that DW statistics was around 1,94, their mean value fluctuated around zero, but still looked a little bit heteroskedastic, and the histogram indicates that the residuals were not normally distributed. Thus as expected the Sample Entropy alone cannot be the only explanatory variable, but as opposed to the previous results, it gave us the statistically significant result with high R-squared, so that the level of average RMSE conditioned on the Sample Entropy increases if the values of Sample Entropy rise.

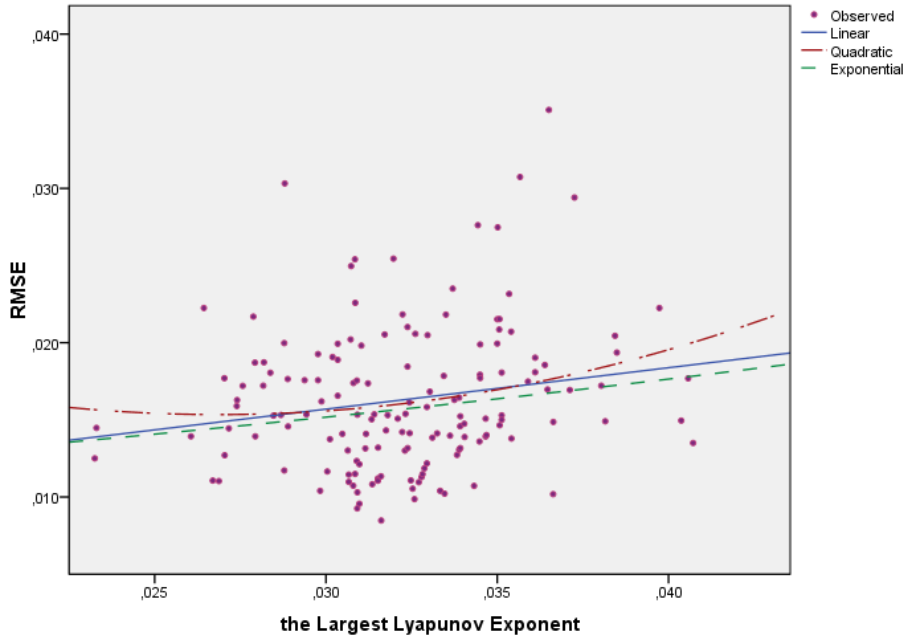

Figure 4.36: .

Model Summary and Parameter Estimates

RMSE

Equation	Model Summary					Parameter Estimates		
	R Square	F	df1	df2	Sig.	Constant	b1	b2
Linear	,034	5,205	1	148	,024	,008	,269	
Quadratic	,041	3,107	2	147	,048	,033	-1,309	24,364
Exponential	,032	4,856	1	148	,029	,010	15,068	

Figure 4.37: Fitting results - linear, quadratic and exponential trend.

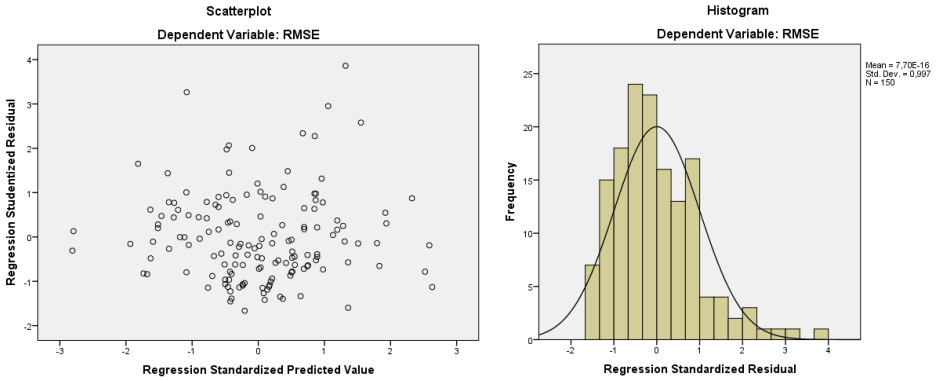

Figure 4.38: Residuals diagnostics for linear trend.

By looking at the figure 4.36 it is noticeable that there is no clear linear relationship visible. Again we tried a linear, quadratic and exponential fitting line presented in table above. Because again all of them show an increasing tendency, we can state that the larger the largest Lyapunov exponent is, the larger the average RMSE our LSTM models tends to be, i.e. the time series become less predictable. Although the highest R-squared was achieved for the quadratic trend which F-test was significant at the significance level of 0,05, its parameters were not - p-value for *constant* was 0,1967, 0,4065 for $b1$ and 0,3157 for $b2$. Therefore we used the model with linear fit for subsequent residuals analysis which parameter were statistically significant with p-value 0,0465 for *constant* and 0,0235 for $b1$. It showed that the assumptions were not satisfied as seen on the plots of residuals.

Below there is a correlation table describing the intensity of the linear relationship between normalized RMSE and each of the predictability statistics using the Pearson correlation coefficient estimated by SPSS. Because we are just interested in how the levels of the average RMSE moves depending on each predictability statistics, this approach is much more simpler than finding the regression line.

Correlations

		Sample Entropy	Lyapunov Exponent	|H-0,5|	|DFA-0,5|	RMSE
Sample Entropy	Pearson Correlation	1	,120	-,158	-,221**	,949**
	Sig. (2-tailed)		,143	,053	,006	,000
	N	150	150	150	150	150
Lyapunov Exponent	Pearson Correlation	,120	1	-,104	,032	,184*
	Sig. (2-tailed)	,143		,206	,697	,024
	N	150	150	150	150	150
|H-0,5|	Pearson Correlation	-,158	-,104	1	,061	-,171*
	Sig. (2-tailed)	,053	,206		,461	,036
	N	150	150	150	150	150
|DFA-0,5|	Pearson Correlation	-,221**	,032	,061	1	-,195*
	Sig. (2-tailed)	,006	,697	,461		,017
	N	150	150	150	150	150
RMSE	Pearson Correlation	,949**	,184*	-,171*	-,195*	1
	Sig. (2-tailed)	,000	,024	,036	,017	
	N	150	150	150	150	150

**. Correlation is significant at the 0.01 level (2-tailed).

*. Correlation is significant at the 0.05 level (2-tailed).

Figure 4.39: Pearson correlation coefficient.

The correlation table reveals that there are significant correlation coefficients between the predictability statistics and the RMSE. The correlation coefficient between the Sample entropy and RMSE is very large - 0,949 which means nearly perfect correlation. Thus the presence of irregularities has a huge impact on how the LSTM can forecast the future values. The correlation coefficient between RMSE and the largest Lyapunov coefficient implies the small but significant positive correlation, so the chaotic behaviour tends to increase the RMSE. The correlation coefficients between transformed Hurst coefficient and RMSE both for estimation using R/S analysis and DFA method signifies the small but significant negative correlation, which means that the closer to the value of 0,5 the Hurst coefficient gets, the worse prediction the LSTM makes.

Furthermore, recall from the section 3.5 that the Pearson correlation coefficients can be estimated from the R-squared obtained by linear fit in simple linear regression. Thus using the R-squared we get the correlation coefficient between the Sample entropy and RMSE as -0,9486, the correlation coefficient between RMSE and the largest Lyapunov coefficient as +0,1844, the correlation coefficients between transformed Hurst coefficient (R/S) and RMSE as -0,1703 and the correlation coefficients between transformed Hurst coefficient (DFA) and RMSE as -0,1949 which is in agreement with the values presented in table on the previous page.

Chapter 5

Conclusion

One of the most prominent feature of the financial time series is their non-stationarity and non-linearity. These two qualities make the stock prices time series modeling and prediction difficult, so their log returns are used for the statistical analysis instead. Fortunately, the recent advances in the field of machine learning lead to the development of the specific type of the artificial neural networks called the Long Short-Term Memory network belonging to the Recurrent Neural Network class which is capable to capture various temporal relationships bounded to the real-world time series, and thereby can be used for their forecasting.

Apart from that, time series predictability is related to the time series characteristics describing their long-term memory presence, information decay, presence of irregularities or chaotic behavior. These measures called namely the Hurst coefficient, Metric entropy and Lyapunov exponent can be estimated to some accuracy using the modern computational methods, and thereby represent some guidance for classifying time series according to their forecasting difficulty.

This diploma thesis was devoted to the both matters discussed above - time series predictions using the LSTM networks and assessing their predictability using aforesaid statistical measures. Then we tried to determine a possible relationship between the quality of predictions and these measures of predictability. Apart from using the log returns, to get the comparable results, at first, we selected five LSTM models (with a single hidden LSTM layer) with certain values of parameters which lead to the lowest average RMSE from forecasting eight time series of stocks from 7 different industries. Then we made a forecasts of 150 randomly selected log returns of financial stocks available on Yahoo!Finance to obtain the RMSE of a test sample as a measure of prediction quality. The measures of predictability were calculated on the same test sample, and afterwards, we tried to asses any relationship between them using the plots, simple linear regression and the correlation coefficients.

LSTM networks predictions - In our thesis we used the time series of log returns of daily adjusting closing price of the stocks in USD with 2894 observations which correspond to the period from 1.1.2006 to 30.6.2017, that was split into training and test sample on which the forecast by LSTM networks was made. Because we desired to get the comparable results, we identified five LSTM networks that could have the best forecasting abilities. Thus we used 8 time series from 7 different industries to find five optimal models applicable to a wide variety of stocks by changing its two parameters - number of memory blocks and number of training epochs. Note that we used 2 stocks from financial industry as we realized that there

is a much greater availability of suitable time series than in the other industries.

The main part was about making predictions of 150 randomly selected stocks having available data for our chosen time periods, using five LSTM networks. Although they were selected randomly, we had to omit few of them because of the data incompleteness, shortness etc. Furthermore, we tried to preserve the ratio of numbers of stocks between different industries given by the previous tuning part but we did not strictly adhere to it because of the different level of suitable stocks found among the industries. The lowest average RMSE attained was 0,008476 for ticker KO. On the other hand, the largest average RMSE obtained was 0,035085 for ticker NYNY which is a quite large compared to the mean value of average RMSE of all 150 stocks which was 0,016303.

Quality of forecasts vs. predictability - To asses the predictability of selected 150 time series we calculated three statistics on test sample - Hurst coefficient using a Rescaled Range Analysis and Detrended Fluctuation Analysis, Metric entropy estimated by calculating a Sample entropy and the largest Lyapunov exponent to asses a long-term memory, stability and chaotic behaviour of the given time series. Then we tried to find out if there is any statistical relationship between the prediction quality (average RMSE) and each of these measures of predictability, using a simple linear regression to find some simple and approximate model, and the correlation coefficients.

The estimation of the Hurst Coefficient using two methods showed us that the R/S analysis gave us in average lower values than those estimated by DFA method, which were also spread in a wider range. Specifically, the R/S Analysis resulted in the Hurst coefficient ranging from 0,385966 to 0,558248, and approximately half of the time series were estimated as being persistent and a half of them were anti-persistent. Furthermore, the Hurst coefficient was estimated also as DFA statistics. For DFA measures using RANSAC fitting method the values ranged from 0,317632 to 0,707567, and slightly less than half of time series being anti-persistent and the rest of values being either close to random or being persistent. To find some relation to the average RMSE, we transformed their values into the distance from random time series - 0,5 as |H-0,5| and |DFA-0,5| measures. In both cases the simple linear regression indicated a decreasing behaviour of RMSE when the Hurst coefficient is more distant from the value of 0,5 which means that time series with trends were predicted better than time series with random behaviour. The same result was also provided by the Pearson correlation coefficient with the values of -0,171 and -0,195 respectively, which indicate the weak but statistically significant negative correlation.

The Sample entropy ranged from 0,074153 to 0,822532 meaning that our time series contained mostly smaller irregularities or chaotic behavior. The simple linear regression resulted in the models with high R-squared indicating increasing tendency of average RMSE when the Sample entropy rises which is evident from the corresponding plots. Furthermore, as expected, the Pearson correlation coefficient with the value of 0,949 indicated the strong positive correlation. Therefore we found a strong evidence that the LSTM network generates less precise forecast of the time series which contains more irregular patterns or exhibit the chaotic behaviour.

The largest Lyapunov exponent ranged only from 0,023250 to 0,040716 indicating a weakly chaotic and nearly stable system. The corresponding plot showed some rough increasing tendency of the average RMSE when the largest Lyapunov exponent increases but we would suggest to use a larger sample to obtain larger range of the larger Lyapunov ex-

ponent values. Note that the largest Lyapunov exponent does not indicate only chaos but also the presence of noise, so the wider range would refine a possible relationship. The simple linear regression models indicated increasing average RMSE while the Lyapunov exponent increases. Furthermore, the Pearson correlation coefficient with the value of 0,184 showed the weak but statistically significant positive correlation. Therefore, higher values of the largest Lyapunov exponent indicating the chaos tend to be associated with the worse predictions.

In summary, we succeeded in generating of five LSTM networks that were able to relatively accurately forecast 150 time series of log returns of randomly selected stocks. To get the consistent results we used five the most appropriate LSTM models selected using a small sample of time series before we started with forecasting above mentioned 150 time series of which we calculated their average value of RMSE for each stock.

The calculated measures of predictability showed us that our randomly selected time series differed in long-term memory and the irregular patterns, but were relatively similar in terms of the chaotic behaviour and stability. In general, our results are in agreement with the theory so that the random time series were tend to be predicted less accurately that those with Hurst coefficient value more distant from 0,5. More complex time series or with severe irregularities quantified by the Sample entropy were predicted with smaller precision and those with a presence of chaos defined through the largest Lyapunov exponent resulted in the higher RMSE of prediction as expected. Note that our results point out to the LSTM networks ability to learn and accurately forecast time series. If our LSTM networks were designed incorrectly, we would end up with the spurious results of average RMSE with no relation to the time series predictability. Hence we can state state that irregular patterns are strongly related to the accuracy of how the appropriately designed LSTM networks in our thesis are able to make the forecasts. Long-term memory and chaotic behaviour of time series have weaker but still significant influence on our LSTM networks forecasting capabilities.

We think that further study comprising a larger time series sample even from less efficient markets would provide the data with a much broader range of aforesaid measures of predictability, and thereby refine their relation to the quality of forecast, especially in case of the largest Lyapunov exponent. Apart from that, the forecasting abilities depend also on the information from the training set, so that the less accurate predictions might have been partially caused by the fact that LSTM networks did not learn some patterns. Therefore, we would also suggest a further study that takes into account the measures of predictability estimated from the training data as well, and also using the LSTM networks with more than one hidden layer (also called stacked LSTM networks).

List of References

[1] Arlt, J., Arltová, M., Rubliková, E.: Analýza ekonomických časových řad s příklady. Skripta VŠE Praha, 148 str., 2002. ISBN 80-245-0307-7. [cit. 2017-08-19].

[2] Zivot, Eric. Multivariate Time Series. Class notes [online]. [cit. 2017-08-26]. Available from: https://faculty.washington.edu/ezivot/econ584/notes/multivariatetimeseries.pdf.

[3] Stigler, Matthieu.Stationarity [online]. [cit. 2017-08-27]. Available from: http://matthieustigler.github.io/Lectures/Lect1Station.pdf.

[4] NIST/SEMATECH e-Handbook of Statistical Methods.Stationarity [online]. [cit. 2017-08-27]. Available from: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm.

[5] Nau, Robert. Stationarity and differencing [online]. [cit. 2017-08-27]. Available from: https://people.duke.edu/~rnau/411diff.htm.

[6] Romer, Megam. Partial Autocorrelation Function (PACF). The Pennsylvania State University [online]. [cit. 2017-08-27]. Available from: https://onlinecourses.science.psu.edu/stat510/node/62.

[7] Zivot, Eric. Unit Root Tests [online]. [cit. 2017-08-27]. Available from: https://faculty.washington.edu/ezivot/econ584/notes/unitroot.pdf.

[8] Kunst, Robert. Nonlinear time series [online]. [cit. 2017-08-27]. Available from: http://homepage.univie.ac.at/robert.kunst/nonlin1.pdf.

[9] Adhikari, Ratnadip et al. An Introductory Study on Time Series Modeling and Forecasting [online]. [cit. 2017-09-23]. Available from: http://www.realtechsupport.org/UB/SR/time/Agrawal_TimeSeriesAnalysis.pdf.

[10] SEVIL, Hakki Erhan. ON THE PREDICTABILITY OF TIME SERIES BY METRIC ENTROPY [online]. [cit. 2017-07-19]. Available from: http://openaccess.iyte.edu.tr/bitstream/handle/11147/3194/T000543.pdf?sequence=1&isAllowed=y.

[11] Mansukhani, Subir, The Hurst Exponent: Predictability of Time Series [online]. [cit. 2017-06-15]. Available from: http://analytics-magazine.org/the-hurst-exponent-predictability-of-time-series/.

[12] Racinei, Roman, Estimating the Hurst Exponent [online]. [cit. 2017-06-15]. Available from: http://mosaic.mpi-cbg.de/docs/Racine2011.pdf.

[13] PhysioNet webpage, Detrended Fluctuation Analysis (DFA) [online]. [cit. 2017-10-09]. Available from: https://physionet.org/physiotools/dfa/.

[14] Power laws [online]. [cit. 2017-10-09]. Available from: http://pages.stern.nyu.edu/~xgabaix/papers/powerLaws.pdf/.

[15] Hardstone R, Poil S-S, Schiavone G, et al. Detrended Fluctuation Analysis: A Scale-Free View on Neuronal Oscillations. Frontiers in Physiology. 2012;3:450. doi:10.3389/fphys.2012.00450. [online]. [cit. 2017-10-09]. Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3510427/.

[16] Mert, Ahmet et al. DETRENDED FLUCTUATION ANALYSIS FOR EMPIRICAL MODE DECOMPOSITION BASED DENOISING [online]. [cit. 2017-10-09]. Available from: http://pages.stern.nyu.edu/~xgabaix/papers/powerLaws.pdf/.

[17] Kriz, R. Chaos in GDP. Working paper at Acta Polytechnica Vol. 51 No. 5/2011 [online]. [cit. 2017-07-17]. Available on https://ojs.cvut.cz/ojs/index.php/ap/article/viewFile/1442/1274

[18] Marwan, Norbert. Lyapunov Exponents [online]. [cit. 2017-07-17]. Available on http://www.agnld.uni-potsdam.de/~marwan/matlab-tutorials/html/lyapunov.html

[19] Wolf, Alan et al. Determining Lyapunov exponents from a time series. University of Texas at Austin [online]. [cit. 2017-07-17]. Available on http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.152.3162&rep=rep1&type=pdf

[20] Elert, Glenn. Measuring Chaos. The Chaos Hypertextbook. [online]. [cit. 2017-07-17]. Available on https://hypertextbook.com/chaos/lyapunov-1/

[21] Benedetto, Francesco et al., MAXIMUM ENTROPY ESTIMATOR FOR THE PREDICTABILITY OF ENERGY COMMODITY MARKET TIME SERIES [online]. Working papers. ISSN 2279-6916 [cit. 2017-07-19]. Available from: http://dipeco.uniroma3.it/db/docs/wp%20192.pdf.

[22] Yakov G. Sinai.THE ENTROPY OF A DYNAMICAL SYSTEM. [online]. [cit. 2017-07-19]. Available from: http://www.abelprize.no/c61094/binfil/download.php?tid=61211.

[23] Richman, Joshua S. et al. Sample Entropy chapter. Methods in Enzymology. Elsevier, 2004 [online]. [cit. 2017-10-19]. Available from: http://www.sciencedirect.com/science/article/pii/S0076687904840114.

[24] Reignold, Eyal. Artificial Neural Networks Technology. [online]. [cit. 2017-07-31]. Available from: http://www.psych.utoronto.ca/users/reingold/courses/ai/cache/neural2.html.

[25] Jacobson, Lee. Introduction to Artificial Neural Networks. [online]. [cit. 2017-07-31]. Available from: http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7.

[26] Reignold, Eyal. Artificial Neural Networks. [online]. [cit. 2017-08-01]. Available from: http://www.psych.utoronto.ca/users/reingold/courses/ai/Welcome.html.

[27] Jacobson, Lee. Introduction to Artificial Neural Networks Part 2 - Learning. [online]. [cit. 2017-08-01]. Available from: http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-2-learning/8.

[28] Nielsen, Micheal A. Neural Networks and Deep Learning/ Using neural nets to recognize handwritten digits. [online]. [cit. 2017-08-02]. Available from: http://neuralnetworksanddeeplearning.com/chap1.html.

[29] Allerin. Six types of neural networks. [online]. [cit. 2017-08-02]. Available from: https://www.allerin.com/blog/six-types-of-neural-networks.

[30] Mache, Niels. Backpropagation Through Time (BPTT). [online]. [cit. 2017-08-21]. Available from: http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node163.html.

[31] DEEPLEARNING4J. A Beginner's Guide to Recurrent Networks and LSTMs. [online]. [cit. 2017-08-21]. Available from: https://deeplearning4j.org/lstm.html#vanishing.

[32] Pascanu, Razvan; Mikolov, Tomas; Bengio, Yoshua. On the difficulty of training Recurrent Neural Networks. [online]. [cit. 2017-08-21]. Available from: https://arxiv.org/pdf/1211.5063.pdf.

[33] Kroese, Ben, Smagt, Patric van der. An introduction to Neural Networks. The University ofAmsterdam, 1996. [online]. [cit. 2017-08-14]. Available from: https://www.infor.uva.es/~teodoro/neuro-intro.pdf.

[34] Neto, Joao. Radial Basis Functions. [online]. [cit. 2017-08-14]. Available from: http://www.di.fc.ul.pt/~jpn/r/rbf/rbf.html.

[35] Brownlee, Jason. The Promise of Recurrent Neural Networks for Time Series Forecasting. [online]. [cit. 2017-08-11]. Available from: http://machinelearningmastery.com/promise-recurrent-neural-networks-time-series-forecasting/.

[36] Mahmoud Abou-Nasr. Time Series forecasting with Recurrent Neural Networks NN3 Competition. [online]. [cit. 2017-08-11]. Available from: http://www.neural-forecasting-competition.com/downloads/NN3/methods/24-NN3_Mahmoud_Abunasr_competition_complete.pdf.

[37] Olah, Christopher. Understanding LSTM Networks. [online]. [cit. 2017-08-21]. Available from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[38] Fan, Jianqing. The Elements of Financial Econometrics. Asset Returns chapter. [online]. [cit. 2017-11-30]. Available from: http://orfe.princeton.edu/~jqfan/fan/FinEcon/chap1.pdf/.

[39] Fryzlewicz, Piotr. Modeling and forecasting financial log-returns as locally stationary wavelet procesess. [online]. [cit. 2017-11-30]. Available from: http://stats.lse.ac.uk/fryzlewicz/fints/fts4.pdf/.

[40] How to Normalize and Standardize Time Series Data in Python. Machine Learning Mastery. [online]. [cit. 2017-11-04]. Available on http://machinelearningmastery.com/normalize-standardize-time-series-data-python/.

[41] CIRPwiki webpage. Statistics. [online]. [cit. 2017-11-05]. Available from: http://cirpwiki.info/wiki/Statistics/.

[42] MIRWAIS. Largest Lyapunov Exponent with Rosenstein's Algorithm.[online].[cit. 2017-11-05]. Available on http://www.mathworks.com/matlabcentral/fileexchange/38424-largest-lyapunov-exponent-with-rosenstein-s-algorithm?requestedDomain=www.mathworks.com

[43] The Point Cloud Library webpage. How to use Random Sample Consensus model. [online]. [cit 2017-11-05]. Available on http://pointclouds.org/documentation/tutorials/random_sample_consensus.php

[44] Yentes, Jennifer M. et all. Effect of parameter selection on entropy calculation for long walking trials.Gait & Posture , Volume 60 , 128 - 134l. [online]. [cit 2017-11-05]. Available on https://www.sciencedirect.com/science/article/pii/S0966636217310226

[45] Simon Laura and Young, Derek. Lesson 1: Simple Linear Regression. The Pennsylvania State University. [online]. [cit 2017-11-05]. Available on https://onlinecourses.science.psu.edu/stat501/node/251

[46] Statistics How To webpage. Durbin Watson Test & Test Statistic. [online]. [cit 2017-11-05]. Available on http://www.statisticshowto.com/durbin-watson-test-coefficient/

[47] Simon Laura and Young, Derek. 1.6 - (Pearson) Correlation Coefficient.[online]. [cit 2017-11-05]. Available on https://onlinecourses.science.psu.edu/stat501/node/256

Appendix

```
#A script contains four basic functions:
#Downloads data: yahoo_download(tickers)
#LSTM network generation and prediction of time series: LSTM_gener(time_series, numBlocks, numEpochs)
#Measures of predictability of time series calculation: predict_stat(time_series, sample=0)
#Saves the all results as structured Excel table: save_res(tickers, PREDICTABILITY, RMSE)


#downloading the data from Yahoo!Finance using a given list of tickers, e.g. ['GE','AAPL']
def yahoo_download(tickers):
        all_stock=list()
        import pandas
        from pandas_datareader import data
        import matplotlib.pyplot as plt
        import numpy
        start_date = '2006 01 01'
        end_date = '2017 06 30'
        for i in tickers:
                panel_data=data.get_data_yahoo(i,  start_date, end_date)
                print(i, 'downloaded successfully')
                stock=panel_data[panel_data.columns[4:5]] #adjusted closing price
                stock=numpy.log(stock).diff()#log returns
                stock=stock.dropna() #droping NaN values
                all_stock.append(stock)
        return all_stock


#generates a LSTM, makes a prediction, plots the graph and returns the RMSE and NRMSE
def LSTM_gener(time_series, numBlocks, numEpochs):
        RMSE=list()
        import math
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import LSTM
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.metrics import mean_squared_error
        import matplotlib.pyplot as plt
        import pandas
        import numpy

        # fix random seed for reproducibility
        from numpy.random import seed
        seed(121)
        from tensorflow import set_random_seed
        set_random_seed(122)

        # convert an array of values into a dataset matrix
        def create_dataset(dataset, look_back=1):
                dataX, dataY = [], []
                for i in range(len(dataset) look_back 1):
                        a = dataset[i:(i+look_back), 0]
                        dataX.append(a)
                        dataY.append(dataset[i + look_back, 0])
                return numpy.array(dataX), numpy.array(dataY)
```

```
# load the dataset
for i in range(len(time_series)):
        stock=time_series[i]
        print('LSTM initiated for', tickers[i])
        ts = stock.values
        aver=numpy.mean(ts)

        dataset=stock.values
        dataset = dataset.astype('float64')

        # normalize the dataset
        scaler = MinMaxScaler(feature_range=(0, 1))
        dataset = scaler.fit_transform(dataset)

        # split into train and test sets
        train_size = int(len(dataset) * 0.80)
        test_size = len(dataset)   train_size
        train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
        print(len(train), len(test))

        # reshape into X=t and Y=t+1
        look_back = 5
        trainX, trainY = create_dataset(train, look_back)
        testX, testY = create_dataset(test, look_back)

        # reshape input to be [samples, time steps, features]
        trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
        testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))

        # create and fit the LSTM network
        model = Sequential()
        model.add(LSTM(numBlocks, input_shape=(look_back, 1)))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        model.fit(trainX, trainY, epochs=numEpochs, batch_size=1, verbose=2)

        # make predictions
        trainPredict = model.predict(trainX)
        testPredict = model.predict(testX)

        # invert predictions
        trainPredict = scaler.inverse_transform(trainPredict)
        trainY = scaler.inverse_transform([trainY])
        testPredict = scaler.inverse_transform(testPredict)
        testY = scaler.inverse_transform([testY])

        # calculate root mean squared error
        trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
        print('Train Score: \%.4f RMSE' \% (trainScore))
        testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
        print('Test Score:\%.4f RMSE' \% (testScore))
        rmse=[testScore]
        RMSE.append(rmse)

        # shift train predictions for plotting
        trainPredictPlot = numpy.empty_like(dataset)
        trainPredictPlot[:, :] = numpy.nan
        trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

        # shift test predictions for plotting
        testPredictPlot = numpy.empty_like(dataset)
        testPredictPlot[:, :] = numpy.nan
        testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset) 1, :] = testPredict

        # plot baseline and predictions
        plt.figure(figsize=(13,8))
        plt.title('Original vs. Forecasted Time Series', size=18)
        original=plt.plot(scaler.inverse_transform(dataset), label='Original data')
        trainSample=plt.plot(trainPredictPlot,label='Predictions of the training sample')
```

```
                testSample=plt.plot(testPredictPlot, label='Predictions of the test sample')
                plt.legend(['Original data','Training sample predictions','Test sample predictions'])
                plt.ylabel('Log returns of daily adjusted closing prices', size=15)
                plt.xlabel('Days', size=16)
                plt.savefig('Folder/'+tickers[i], dpi=100)
                plt.close()

        return RMSE


#calculates the measures of predictability, and returns a list of them
#uses a nolds package
#argument sample refers to the sample of time series used for calculation
# 0 whole sample (default), 1 training sample, 2 test sample
def predict_stat(time_series,sample=0):
        PREDICTABILITY=list()
        import pandas
        import numpy
        import nolds

        for i in range(len(time_series)):
                ts=time_series[i].values
                ts=numpy.reshape(ts,(len(ts)))
                if sample>0:
                        if sample==1:
                                ts=ts[:int(0.8*len(ts))] #training set
                        else:
                                ts=ts[int(0.8*len(ts)):] #test set

                hurstexp = nolds.hurst_rs(ts, fit='poly')
                dfaexp = nolds.dfa(ts,order=2, fit_trend='RANSAC')
                entropy=nolds.sampen(ts, tolerance=0.02)
                lyapexp=nolds.lyap_r(ts, emb_dim=14)

                predictability=[hurstexp]+[dfaexp]+[entropy]+[lyapexp]
                PREDICTABILITY.append(predictability)

        return PREDICTABILITY


#saves the results into the structured Excel file and outputs the results
#every row corresponds to a single ticket with all calculated statistics
def save_res(tickers,PREDICTABILITY,RMSE):
        results=list()
        from pandas import DataFrame
        for i in range(len(tickers)):
                all=[tickers[i]]+PREDICTABILITY[i]+RMSE[i]
                results.append(all)
        rs=DataFrame(results,columns=['Ticker','Hurst','DFA','Entropy','Lyapunov','RMSE'])
        rs.to_excel('results.xlsx', sheet_name='results', index=False)
        return rs
```