Financial forecasting with Neural Networks and Reservoir Computing

Georgopoulos Spyridon

SID: **3308180004**

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

DECEMBER 2019

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Financial forecasting with Neural Networks and Reservoir Computing

Georgopoulos Spyridon

SID: **3308180004**

Supervisor:	Dr. Stavrinides Stavros
Supervising Committee Members:	Dr. Berberidis Christos,
	Dr. Baltatzis Dimitrios

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Data Science

DECEMBER 2019

THESSALONIKI – GREECE

Abstract

Financial forecasting has emerged as a key field of study in the last decades as budget planning, and decision-making processes play a vital part in establishing and maintaining a healthy and sturdy business. Last years, Machine Learning and Deep Learning's concepts have become prominent in the financial industry due to their ability to handle vast amounts of data and extract time-dependent patterns from non-linear relationships along with the -nowadays- exponentially growing availability of computing power. More and more industries and business operations tend to deploy artificial intelligence technology to automize their processes, minimize their risks and optimize their development in terms of increasing their revenues as a consequence of high quality and robust productivity. The wide range of Deep Learning usage in the finance industry extends from security and fraud detection, underwriting, stock marketing predictions, and chatbot advisory. [1][2][3]

The possible outcomes and variations of Machine Learning applications are fully capable to cover a wide spectrum of needs and ideas and in this work we have developed a theoretical framework to acquire fundamental understanding of the Deep Learning philosophy and concepts, studying a very popular Neural Network type by the name of Recurrent Neural Networks, and another promising one by the name Reservoir Computing and their predictive abilities on time series of data. In addition to this, our work aims to study systematically and evaluate different ways to convert a dataset into a more «friendly» form, to optimize the predicting ability of the models presented. Conversion of data series to stationary sequences proves to be an inevitable process to carry out time-series analysis efficiently and the second chapter of results is delving into this issue. Finally, this work is also enhanced by a short analysis and description of a «shifting» problem we encountered during different evaluation

processes and experiments, with a short explanation and a path to avoid it. (see #Appendix 1)

Acknowledgements

I would like to take this moment to express my gratitude to my supervisor Dr. Stavrinides Stavros for the useful comments, remarks, and engagement through the learning process of this master thesis. The door to Dr. Stavrinide's office was always open whenever I ran into a trouble spot or had a question about my research plan, and I will always treasure my time spent with him, discussing our research plans.

A very special thanks go to Dr. Antoniadis Ioannis for his contributions willingness to explain techniques and share his knowledge in all areas of this project both computational and physical.

I also owe gratitude to my co-worker MSc Tziatzios Panagiotis. He was always willing to help and discuss our common research issues.

Finally, I would also like to acknowledge my MSc associates, Dragogias Ioannis Tsolakis Stathis and Valk Tom. I was privileged to spend time with them both inside and outside of a work mindset.

Georgopoulos Spyridon

02/12/2019

In memory of my mother,

Contents

1. Introduction: Deep Learning and Philosophy

In «Plato's Republic», a plethora of views regarding virtue and justice are undertaken and narrated by Plato, conveying opinions of other famous philosophers and sophists such as Protagoras, Thrasymachus, Gorgias, and others. His famous question to Thrasymachus, on what the concept of justice is, may have given food for thought and debates through the ages, but at the same time, it unveiled another huge subject of great interest; and that is no other than the puzzlement of what a concept is? Such a philosophical question was enough to result in two rival and major currents around the sixteenth century: empiricism and rationalism. Empiricists argued that concepts are based on the "Empiria" – meaning experience in Ancient Greek, and having concepts is closely related to any person's ability to perceptually recognize features on an object based on his thoughts and knowledge gained of it. On the other hand, rationalists believed that having a concept amounts to the ability to rationally draw conclusions that inferentially occur from the concept. Specifically, they claimed that concepts could be described as interconnected nodes in a complex network. Aristotle in Organon introduced the concept of "categories". His central idea was that any sense or notion could be upgraded to a superior one, leading to the ten fundamental attributes depicted in figure 1.1 in Alonso de Proaza's work, "Porphyrian tree", representing Aristotle's categories. This was perhaps the first attempt in history to build an inferential network [4].

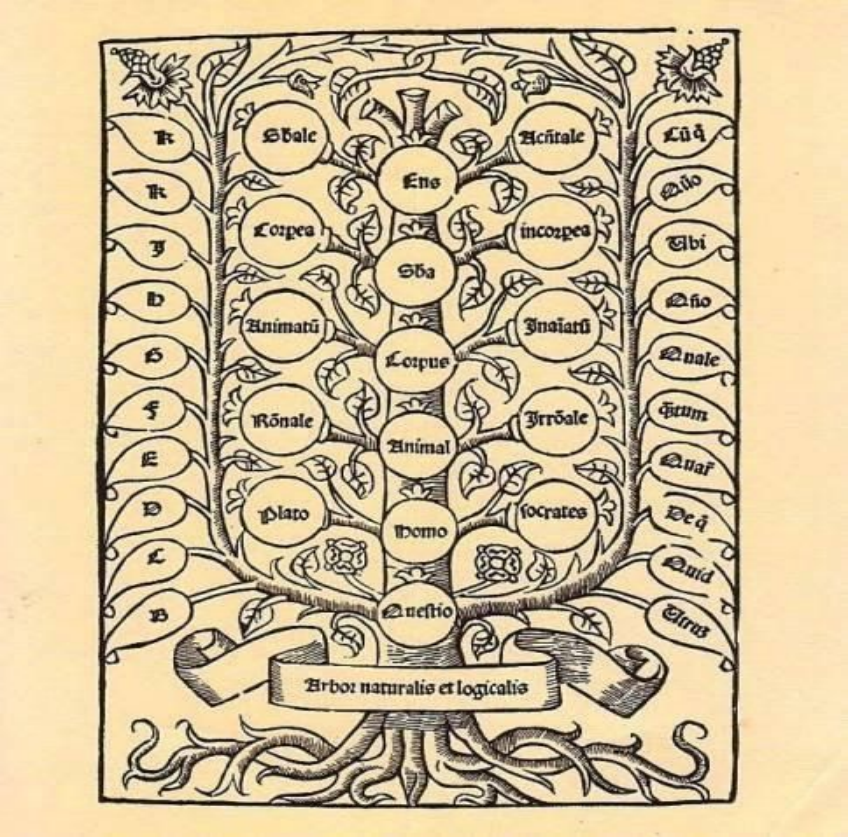


Figure 1.1: Alonso de Proaza's illustration of the Porphyrian tree (sixth-century tree representing Aristotle's categories) in his work "De logica nova" (1512)[5]

Nowadays, the cognitive science approach seems to be the one to approach the finest minds of today's philosophy. According to the aforementioned approach – which, it should be underlined that was derived from both empiricism and rationalism, minds are analogous to computers, and thinking is the equivalent procedure to a great number of complex computations mapping to representational structures in the human mind. It goes without saying that the above description fits the mechanism of Artificial Neural Networks, which are a connection - based systems using vector representations to process information that leads to decision taking processes. In 1781, Immanuel Kant stated at his remarkable philosophical book, "*Kritik der reinen Vernunft"*[6], that *"Concepts without intuitions are empty. Intuitions without concepts are blind"*. In other words, Kant strongly believed that the rationalism and empiricism

had laid the foundations of the cognition theory as conceptual knowledge is a result of both experiences and rules of inference operating together.

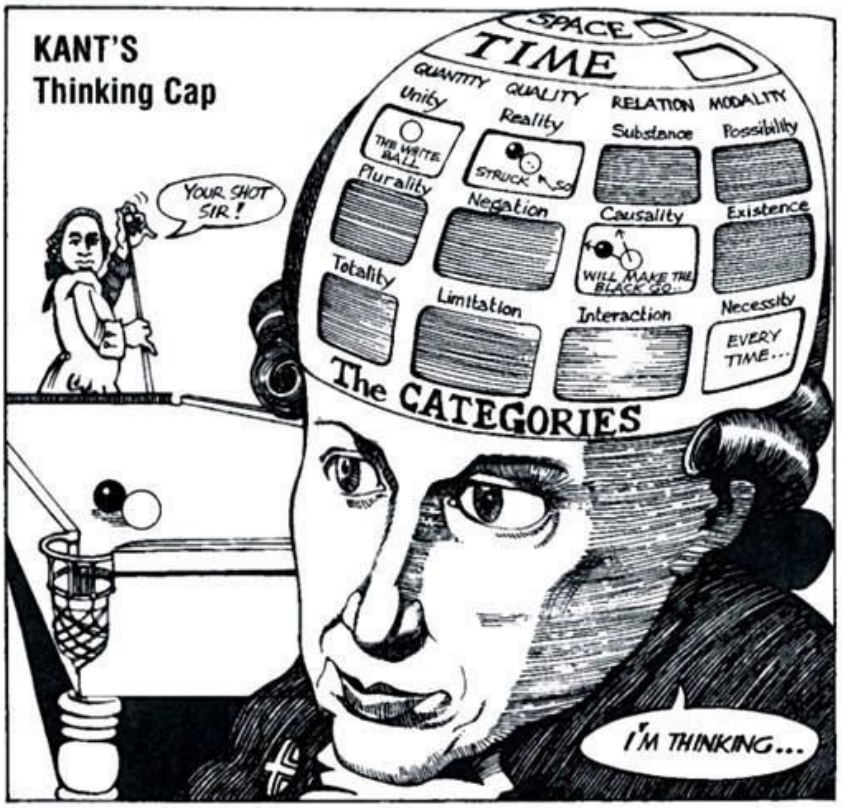


Figure 1.2: Kant: How is a Synthetic A Priori Judgment Possible?

Concluding and adopting Kant's co-working spirit between empiricism and rationalism, I strongly believe that the two concepts presented in this brief introduction should also work together. Philosophy and Machine Learning can walk hand by hand, discovering and unlocking all the secrets of our brain functionalities. Without the human cognition being fully acknowledged, the process of modeling it can never turn to be reality. Dreaming of it is just the start, but the optimal path to it can be narrowed down in these two lines of Stephen Hawking from his speech in Lisbon, in November 2017. "*Perhaps we should all stop for a moment and focus not only on making our AI better and more successful, but also on the benefit of humanity.*"

2. Deep Learning

2.1 Deep Learning and our human brain

Deep Learning's main issue is no other than picking up strategies to mimic the human brain functions and all the other ways evolution has already developed for us. It is an undoubtful fact that we, as humans, don't start thinking in a manner of frames, or from scratch. As the reader goes through these lines, he understands each word based both on his understanding and short memory of the previous ones, as well as his developed training on understanding similar lines and readings. Let's take this chance to delve into the so-called cerebrum part of a human brain and classify the main lobes and their functionalities regarding human senses, linking them to the main methods used by Deep Learning (Artificial Neural Networks, Convolutional Neural Networks, and Recurrent Neural Networks). Artificial Neural Nets are trained to perform tasks by considering examples, without being programmed on specific rules. They aim to address either regression or classification problems and are considered as the main tool in machine learning processes. Secondly comes the Convolutional Neural Networks (CNN), a powerful tool acting within the general frame of the ANN to face image recognition tasks. It is designed in a way to be capable of processing and analyzing pixel data. Finally, we move to Recurrent Neural Networks, which is the leading actor for the current work. RNN is another class of ANN that exhibit temporal dynamic behavior. RNNs can use their internal state (memory) to process sequences of inputs, and this is what makes them special and able to address time series analysis problems.

Figure 2.1a provides an information map of the basic Deep Learning techniques and their contribution to different tasks. On continue, figure 2.1b illustrates a schema of the human brain along with the main lobes, also contributing to different areas of our everyday actions.

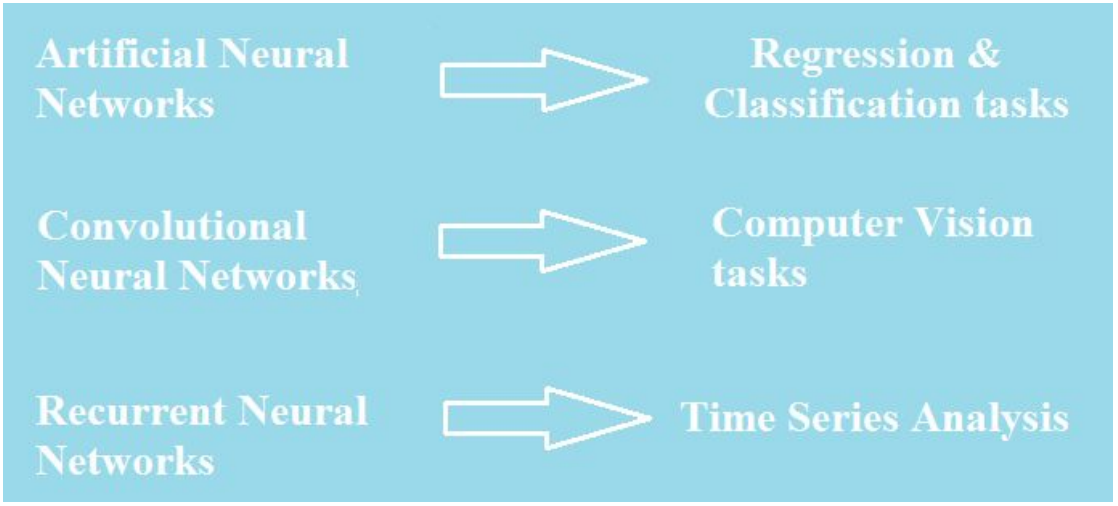


Figure 2.1a: A simple schematic depiction of the main supervised algorithms of Deep Learning networks and their usage.

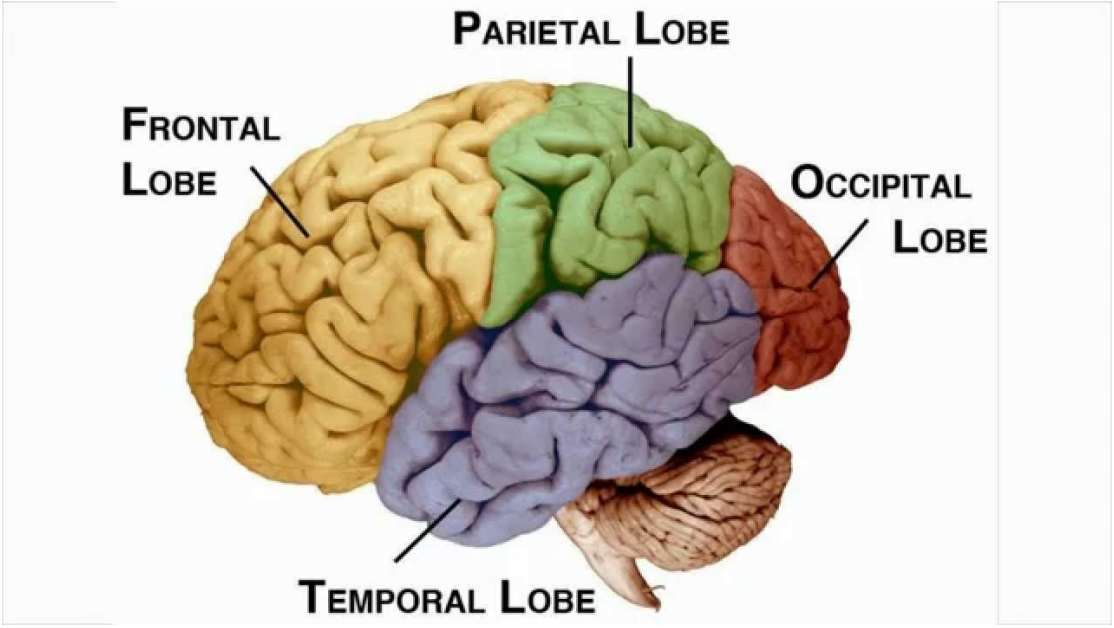


Figure 2.1b: The main lobes of the human brain.

- Occipital lobe: The occipital lobe controls the vision functionalities of the brain through the primary visual cortex and optic radiation. It can, without a doubt, be linked to CNN, as CNN's are responsible for computer vision, image recognition.

- Parietal lobe: This central lobe accounts for perception, sense of the world, arithmetic, spelling, classification of objects, and visuospatial processing. It is yet to be linked to the neural network.

- Temporal lobe: It relates to memory and understanding language. It can be clearly linked to the ANN weights that represent long-term memory. Once the ANN is trained, the network will process inputs the same way as it did the day before and as it will the following ones. This is how the temporal lobe of the human brain also works.

- Frontal lobe: The main contributor for short-term memory, addressing issues such as quick thinking, taking actions, planning, problem solving and behavior control. The RNNs work in a very similar way as they tend to remember cases that happened in previous observations and are capable of applying this gained knowledge in the going forward procedures. [7]

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are machine learning tools based on the biological [8] brain, as the "neural" part of their own name suggests. These brain-inspired systems are intended to replicate the way that humans learn. Their architecture consists of a collection of connected neurons carrying an activation signal and grouped into an input and output layer as well as -in most cases- a number of hidden layers. The input layer is the area to feed the provided data values to the ANN, and the output layer is where the finished computations are placed for us to use. The hidden layers area is the place where the magic happens. The general architecture of an ANN is depicted in figure 2.2. At the input layer, we provide the system with the sample values(signals), and it just passes these values to the next layers. It does not apply any computations on them, and it is not associated with biases or weights. The values are propagated forward through the hidden layers of neurons, and this is the area where several transformations are applied to the input data. All neurons in the hidden layers are connected to each other and are also connected to the neurons of the next layer forming a dynamic complex system. The weight parameter represents the strength of the connection between all these units in terms of greater magnitude of weight from node 1 to node 2, means that neuron 1 has greater influence over neuron 2. The last hidden layer passes on the values to the output layer, where we get the desired number of values in the desired range.

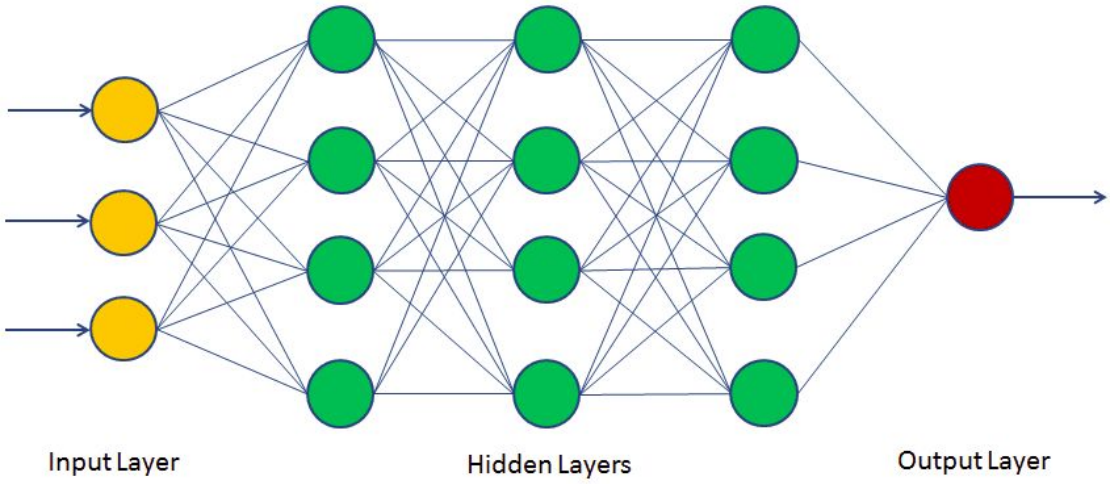


Figure 2.2: A feed forward system basic architecture[9].

2.3 Forward Propagation concept

The above described mechanism can be entitled as the forward propagation procedure. Thus, it refers to the process of feeding the input layer of a neural network with a set of values and get an outcome as a predicted value. This set of initial values is propagated through the hidden layers where multiplication, addition, and activation operations are applied to it, in order to pass it on to the next layer. This procedure is repeated for subsequent layers until it finally reaches the output layer where a final prediction is exported.

2.4 Back Propagation concept

After the forward propagation is completed, it is time to evaluate the result. The neural network backpropagates the information about the error, in reverse, through the network, so that it can alter the parameters properly. In other words, it is the algorithmic way of the system to auto-correct its decisions – outputs by adjusting the network's weights and biases properly. The level of each intervention to the weights is determined by the gradients of the cost function that we will discuss in the next paragraph.

2.5 Cost Function and the Gradient Descent Algorithm

In Machine Learning modeling, chances are that our aim lies in the area of the highest accuracy, or in general, the best scoring of our metric system and thus the model's best performance. The cost/loss function denoted with $J(\theta)$ is the function related to this metric. Particularly, any decrease in $J(\theta)$ increases the model's performance. In other words, the cost function represents a metric of how wrong a model works by estimating the relationship between the target and the predicted value, and helps the learner to correct or change its behavior towards the mistakes made.

Now that the reader has the basic idea of how a model learns we should introduce the Gradient Descent Algorithm to reveal the ways of minimizing the cost function. The Gradient Descents job is to push the model to find the direction or gradient that should move on to minimize the errors $\left|Y_{target} - Y_{predicted}\right|$.

On continue, Gradient Descent is the algorithm that finds the global minimum of the cost function that is going to be optimal for the network. Information travels from the input layers to the output, and then, the calculated error is propagated back to update the node weights leading to more accurate decisions. As depicted in figure (2.3), the model iterates taking fixed-size steps in the error direction and gradually converges to a minimum point where any changes to the parameters will result in zero changes in the cost function.

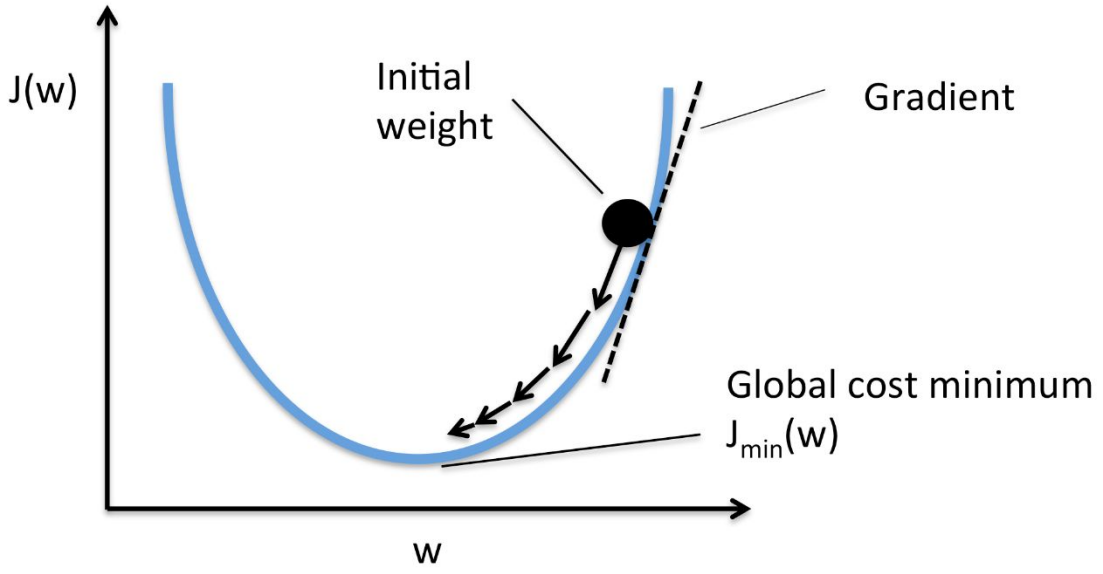


Figure 2.3: Iterations leading to find a minimum point to converge towards the error direction [10].

Delving into the Gradient Descent algorithm types, we end up with the two most popular: the batch gradient descent and the stochastic gradient descent. Both work in a similar way and their differences are related to the way they treat the training data.

1. Stochastic gradient descent uses only one training sample to be fed into the neural system at a time, and thus the parameters of every layer are being updated for every data sample we feed the input layer with. These updates are defined by the following equation

$$\theta_{j+1} = \theta_j - \eta \nabla J(\theta_j; x_i; y_i), \tag{2.1}$$

with η being the learning rate. This method is memory efficient and computationally fast. Moreover, it is characterized by the ability of finding the global minimum as it uses step oscillations to get out of possible local minimums. On the other hand, it loses the advantages of working based on vector operations as it uses one sample at a time.

2. Regarding batch gradient descends the concept remains the same, but the parameter weights are updated only once after every training data has been passed through the network. In mathematical terms, equation (2.1) is iterated over only once, and thus we can write

$$\theta_{j+1} = \theta_j - \eta \nabla J(\theta_j) \tag{2.2}$$

This method uses vectorized operations to process the training inputs, and due to the fact that all computer resources are used to update and pass the inputs within the network, it proves to be more computationally efficient compared to stochastic gradient descent. On the contrary, if the training input is too large, it can exhibit slow convergences and memory issues.

2.6 Recurrent Neural Networks to LSTM

According to what was mentioned above, any RNN systems should exhibit a reasoning use about previous events to inform the later ones to address -in a way- to the flaw of Artificial Neural Networks that cannot make decisions based on already gained knowledge. This reasoning use in terms of temporal dynamic behavior stems from the fact that the connections between nodes of an RNN form a directed graph along a temporal sequence. The ability to take sequential input and produce sequential output by sharing knowledge between nodes is the essence of an RNN system and has resulted in breakthrough achievements and improvements in Natural Language Processing (NLP), image captioning, speech recognition, time series analysis and

many other fiends [1]. Every decision an RNN takes at time t, if strongly affected by the decision it took a moment earlier, t-1, making the system working with two sources of input, corresponding both to the present and to the past and combined respond to new data taking decisions in the same way we do as humans. This feedback loop connecting every neuron to its past is the main reason that RNN differs from every feedforward network, and it can be mathematically defined as follows:

$$h_t = \varphi(Wx_t + Uh_{t-1}) \tag{2.3}$$

Where h_t corresponds to the hidden state at time t and x_t is the input at the same time; x_t is modified by a weight matrix W and then added to the hidden state of the previous time step $t-1$, multiplied by a hidden state matrix U by the name transition matrix. The weight matrices values neglect the importance given by the model to the current and the previous state. Finally, φ is a logistic sigmoid function or a *tanh* function, depending on the nature of the dataset and the task the RNN is being built for. Either way, it needs to be a non-linear and differentiable function. A schematic view of a basic RNN structure is depicted in Fig2.4:

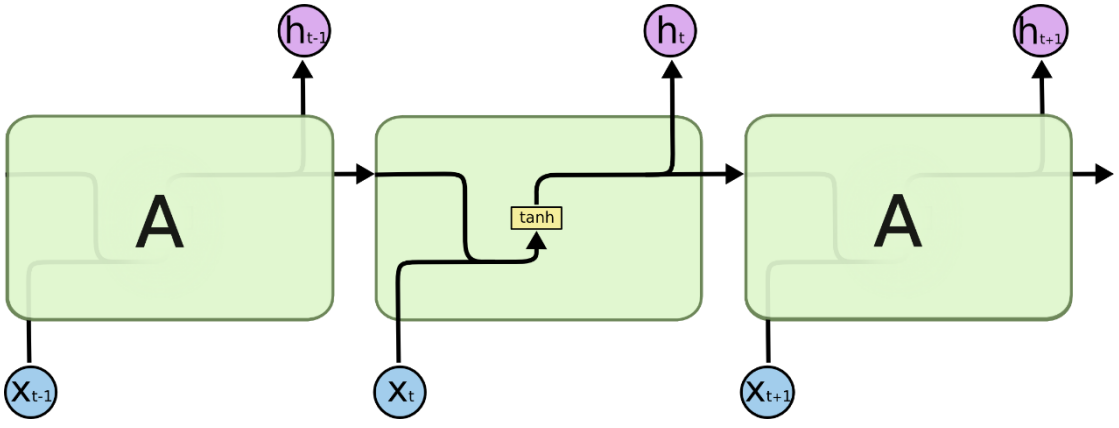


Figure 2.4: The basic RNN architecture, consisting of only one layer, the *tanh* function at time point t [11].

Given that RNN enables the modeling of time-depended data issues such as stock marketing, stock prediction, text generation, and others, we now need to refer to the main enemy; the decay of information through time. This -of major significance- problem comes with the name "Vanishing Gradient Problem" and it mostly occurs when dealing with large time series of data. It was first discovered by Sepp Hochreiter in 1991, a skilled scientist that contributed to the development of Neural Network and Deep Learning as we use it today [12].

As already seen, gradient descent algorithm is responsible to find the global minimum of the cost function and propagate back the calculated error to update the weights of the network in a beneficial way, in terms of accuracy. The case with the RNNs is that not just the neurons before the output layer need to adjust their weights accordingly, but all of the neurons back in time. As explained in [12], [13], when working with long sequences, the derivatives of φ function becomes smaller and smaller. Multiple multiplications with the same small values result in lower and lower gradients. After some more steps, the gradients may vanish completely, which is catastrophic for the network because from that point and then, there is no more training and thus no more learning. So, since the output is based on earlier inputs, the system will end up with training neurons at time t with inputs that are not trained at all.

Similarly, it is also common for gradients having large values to exhibit exploding gradient problems, exhibiting large and larger values leading the system to be unstable. The exploding vanishing problem can be solved by clipping the gradient at a pre-defined threshold. On the other hand, the vanishing problem is way more complex to solve. To address this problem, the long short-term memory (LSTM) block was proposed, once again, by Sepp Hochreiter and his Ph.D. supervisor Jurgen Schmidhuber in this paper in 1997[14]. LSTMs were explicitly designed to address the vanishing gradient problem. Their chain like structure can be seen in figure 2.5a where the four layers of information processing compared to the only one of the basic RNN is obvious.

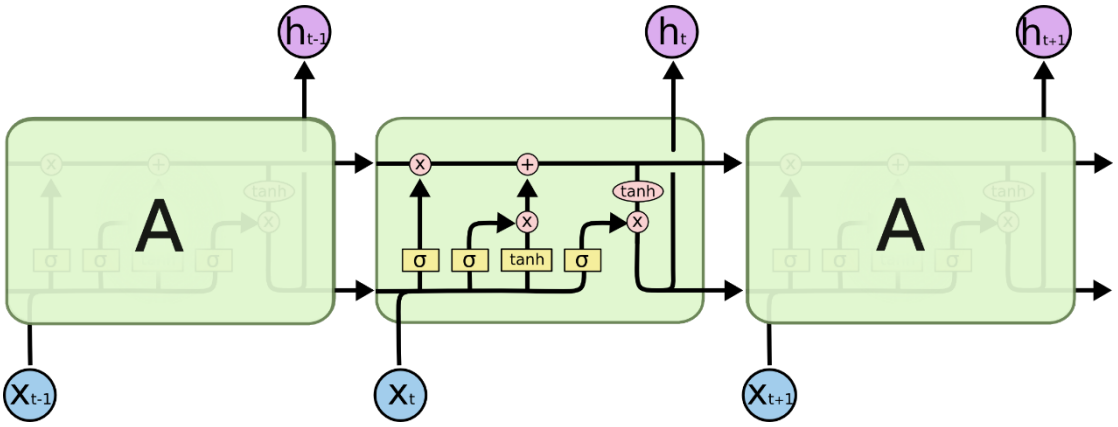


Figure 2.5a: The LSTM basic architecture, depicting all four layers of operations from inputs values to the output ones [11]

All the steps taking place in an LSTM cell are depicted in figure (2.4b). The notation used to represent the physical parameters is the following:

C_{t-1} stands for the input from a memory cell at time t

x_t is the input value at time t

h_t is the output for time t that is forwarded to the output layer and the hidden layer in $t+1$

The initial step of processing the information fed is to modulate the amount of it that the cell state needs to keep or get rid of. In figure (2.5b-1) the first layer is in action where the sigmoid function called "forget gate" and denoted as σ outputs a number in the range of $[0, 1]$ with values near zero being the unwanted information that needs to be forgotten and the values close to one the ones to be kept. The next step combines two procedures. The sigmoid layer called "input gate layer" gets to decide which values the cell needs to update, and the *tanh* layer fulfills this update by creating a vector C_t of new values to be added to the cell state. The multiplication of C_{t-1} *with* f_t will result in a new C_t that forgot everything the cell wanted to get rid of. Afterwards, we add the term $i_t * C_{t-1}$ that represent the new candidate values as visually described in figure (2.5b-3). The 4th and last stage is associated with the output of the cell. As seen in figure (2.5b-4), the first sigmoid filters the parts of the

cell to be pushed forward to the output along with a *tanh* function that scales the information values between $[-1, 1]$ to ensure that the output contains all the parts needed. Thanks to the aforementioned gate-based mechanisms, LSTM has proved to be the cure regarding the vanishing and exploding gradient problems, producing high quality results and be considered well suited to regression tasks related to time series analysis. Now that the reader, hopefully, is more comfortable with RNN and LSTM architecture as an advanced type of deep learning topic, we can move to the next chapter, where we will familiarize ourselves with an extension of deep learning that comes with the name of "Reservoir Computing."

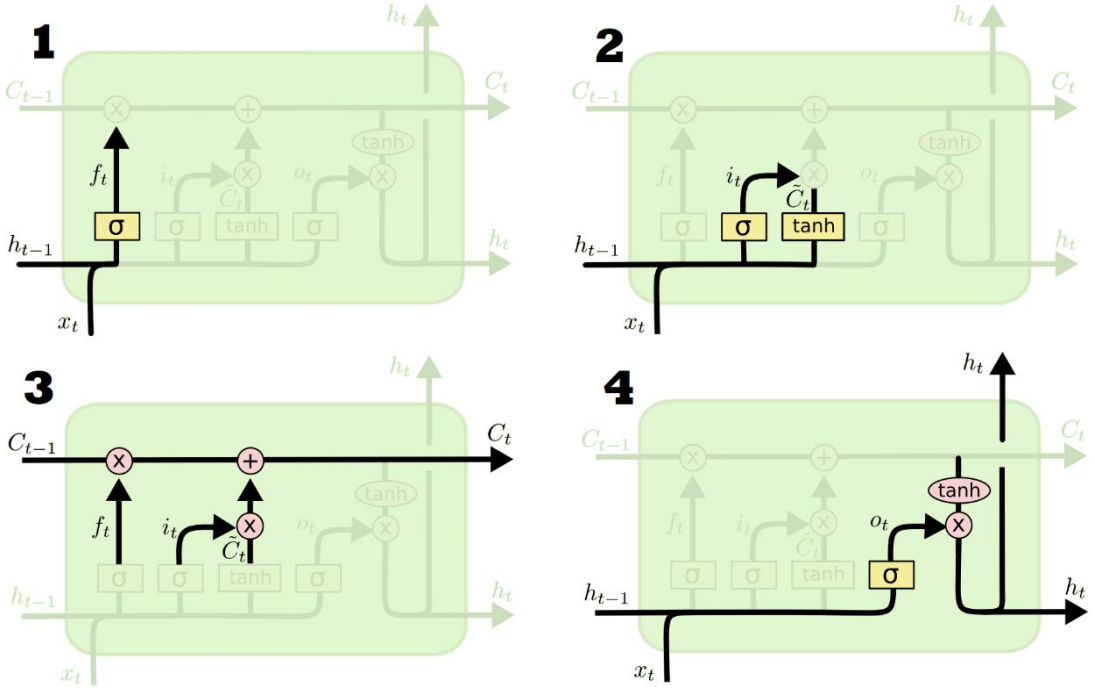


Figure 2.5b: The 4-layer processing parts of an LSTM [11][15]

3. Reservoir Computing

3.1 Introduction to RC

Since the reader has been introduced to the concept of back propagation algorithm, which plays a key role in implementing a neural network, we can take a look at a new concept that can be well analyzed under the title of "an alternative back propagation". It should be underlined that training a recurrent neural network can prove to be a complex task that demotivates many researchers to use them. Some years ago, an alternative way was presented; a way that takes the whole training understanding and development to a new level. The Echo State Network (ESN), along with the Liquid State Machine (LSM), are two machine learning algorithms that were independently developed within the concept of computational predictive models techniques sharing the same basic idea. That is no other than feeding an input signal to a random dynamical system named «reservoir» and map it to a higher dimension, non-linear, space-state representation. Afterwards, a simple readout mechanism is trained to read the reservoir state and map it to the desired output. This results in a massive reduction of the computational time and effort as the training process takes only place in the readout section and the reservoirs connected neurons remain unchanged. The aforementioned training algorithms come under the generic name of "Reservoir Computing", a new and promising framework in the field of predictive models which can be viewed as an extension of Deep Learning. In the present work, we will be using the Echo State Network, which was proposed by Herbert Jaeger [10] [11].

As depicted in figure 3.1, the ESN main components are – the input layer, the reservoir, and an output-readout layer.

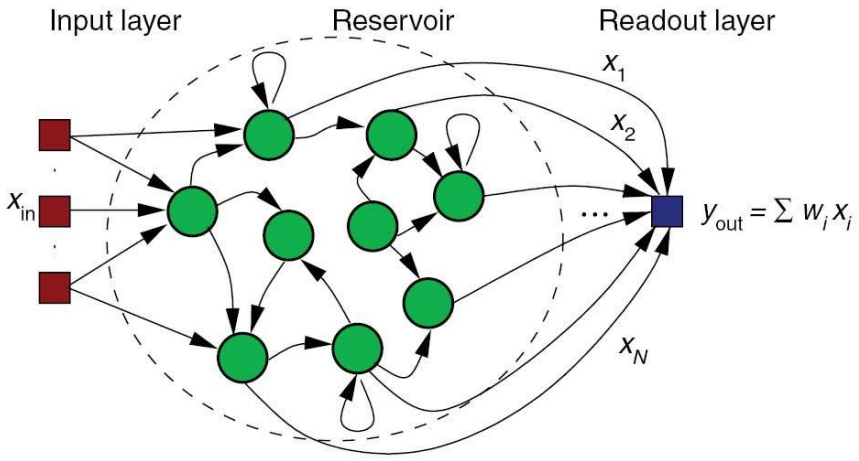


Figure 3.1: A schematic depiction of the three stages of an echo state network system[18].

Input layer: The input layer in an ESN is associated with the task of receiving a signal and distributing it to the reservoir neurons. In this study, the input series u(n) consists of the Nu series of data points time stepped discretized (n, n+1, n+2, ..) that can be either one or multi-dimensional. These series are "pushed" to the reservoir using a randomly generated weight matrix Win. Besides u(n), most reservoirs use a bias input too that contributes to the variability of input dynamics increase.

The Reservoir: The reservoir of an ESN is characterized by a number of sparsely connected neurons (typically 1% connectivity) that transform the input signal to a higher dimension. The connectivity and weights of these neurons are randomly assigned. All neurons are defined by the equation (3.1), which describes the state of each neuron and nodes, and at the same time, reassures that these states depend on the previous states [8][19].

$$x(n+1) = (1-a)x(n) + a\ tanh(Wx(n) + W^{in}u(n+1), \quad (3.1)$$

The state update can be assigned to the *tanh* sigmoid function, which is the most popular regarding reservoir computing studies. Analyzing the above equation, it can be easily understood that each neuron state $x(n+1)$ is derived from its current state $x(n)$, along with input data $W^{in}u(n+1)$ and a non-linear expression describing the state of the other reservoir nodes $Wx(n)$. This is also one of the most critical parameters towards ESN performance. W represents the random weight matrix of neuron connections, which ensures the networks recurrent loops and lead to the essence of this model, the "memory retaining". "Memory retaining" of an ESN is similar to the LSTM memory of previous timesteps and comes with the name of echo state property, which is directly related to the spectral radius value. Spectral radius ρ is a mathematical term denoting the largest maximum eigenvalue of W and needs to be less than 1, for the memory to be retained. Finally, α represents the leaking rate of

the system. In figure 3.2 some reservoir activations are plotted. Notice that the scaling regarding the y axis is [-1,1] as the weights are outputs of the *tanh* function. X axis is the timesteps of the training process; as can be seen that for $t < 40$ the amount of variation for every node is greater than in later times. The initial activations are the result of the initial random states given, and throughout the time, they get less and less variable, as the auto-correct procedures of the reservoir tend to minimize towards as the final activation values. To address this issue, in this work, we will be using an initial length to be used as this $t < 40$ area, to ensure that activations through the training process will come to a stable version more rapidly.

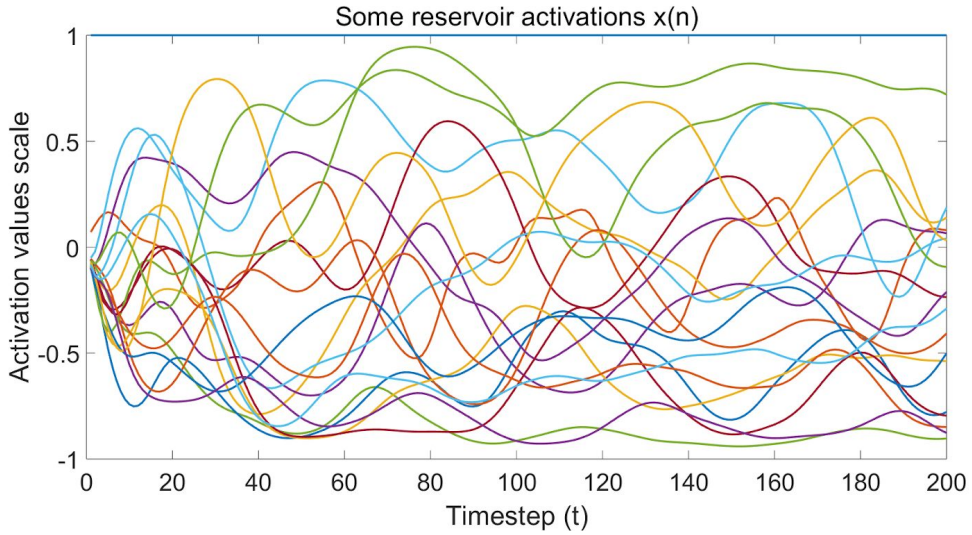


Figure 3.2: The node activity throughout the training process. For $t < 40$, less variability occurs.

Output layer: As soon the reservoir generates the random connections and the random weights associated with them, the output layer takes over. These connections are now fixed and not changeable during the training and testing phases of the algorithm. The output layers task is to take the output of the reservoir to best approximate the target

data $y_{target}(n)$. The novel idea is that only the output layer weights have to be trained by using any linear regression algorithm or a pipeline of them.

After referring to the central idea behind ESN, let's shed some light on the main components and global physical parameters taking action in the whole process of training a network like that, and thus to the strategies we can use towards reservoir optimization. As seen in equation 3.1 the reservoirs dynamical behavior is governed by W, W^{in} *and* α which are related to other physical parameters that affect the model performance.

3.2 The reservoirs size

The reservoir Size N_x denotes the number of neurons existing in the heart of the network. The general feeling is that an increase of a reservoirs capacity results in the lowest errors and thus better performance, as soon as all necessary measures have been taken to avoid overfitting issues. Mantas Lukosevicius states that the reservoir can be too big only when dealing with trivial tasks and there is a lack of available data.[17], [19], [20]

3.3 Spectral Radius

The Spectral Radius ρ is the maximum eigenvalue of W and one of the highest significance physical parameters as it ensures the echo state property and by extension, the recurrent character of the model. For the echo state property to be ensured, ρ needs to be less than 1 according to the majority of published bibliography, but on the other hand, some argue that the property can hold even for larger values even though chaotic behavior may occur leading to failure regarding the reservoir memory[21]. In this work, ρ has been taken as 1[22].

3.4 Input Scaling

Input Scaling x is a parameter that may not be mandatory for the model to work, but just as the Spectral Radius is highly associated with the echo state property, and thus needs to be examined and optimized for efficiency reasons. As its name suggests it scales the columns of W^{in} along with the bias used, and thus it determines the scale of the whole input signal.

3.5 Leaking Rate

The leaking Rate parameter α is the one to define the speed at which the reservoir update dynamics in terms of how quickly the input is being fed (leaked) into the reservoir. It ranges from 0 to 1. Smaller values for leaking rates aim to induce slow dynamics resulting in short term memory extension, and this sensitivity suggests that small changes to this parameter may lead to different results[16]. Most papers related to ESN and chaotic data series propose relatively small values for a, so in the current work, we will be using $a = 3$.

4. Related Work

This chapter is a quick exploration of some results taken from the published bibliography on Reservoir computing, LSTM networks, and time series analysis in general, that aims to familiarize the reader even more with the concepts and make the transition to the results and discussion section more natural and effective.

Starting with Jenny Su's work "Reservoir Computing in Forecasting Financial Markets"[8]; her thesis gives a remarkable analysis of reservoir computing concepts along with results regarding Mackey – Glass systems and stock marketing prices. In her work, she basically used the Mackey Glass equation to study further the role that bias plays as regards the optimization of a reservoir system, a subject not extensively discussed in the bibliography. The final reservoir constructed is used to examine the impact of bias and the output weight matrix to the mean squared error. Figure 4.1 shows the relationship of the mean squared error with the bias scaling constant for some specific reservoir parameters. It is more than obvious that the lowest MSEs occur for biases values greater than 1 and less than 3.

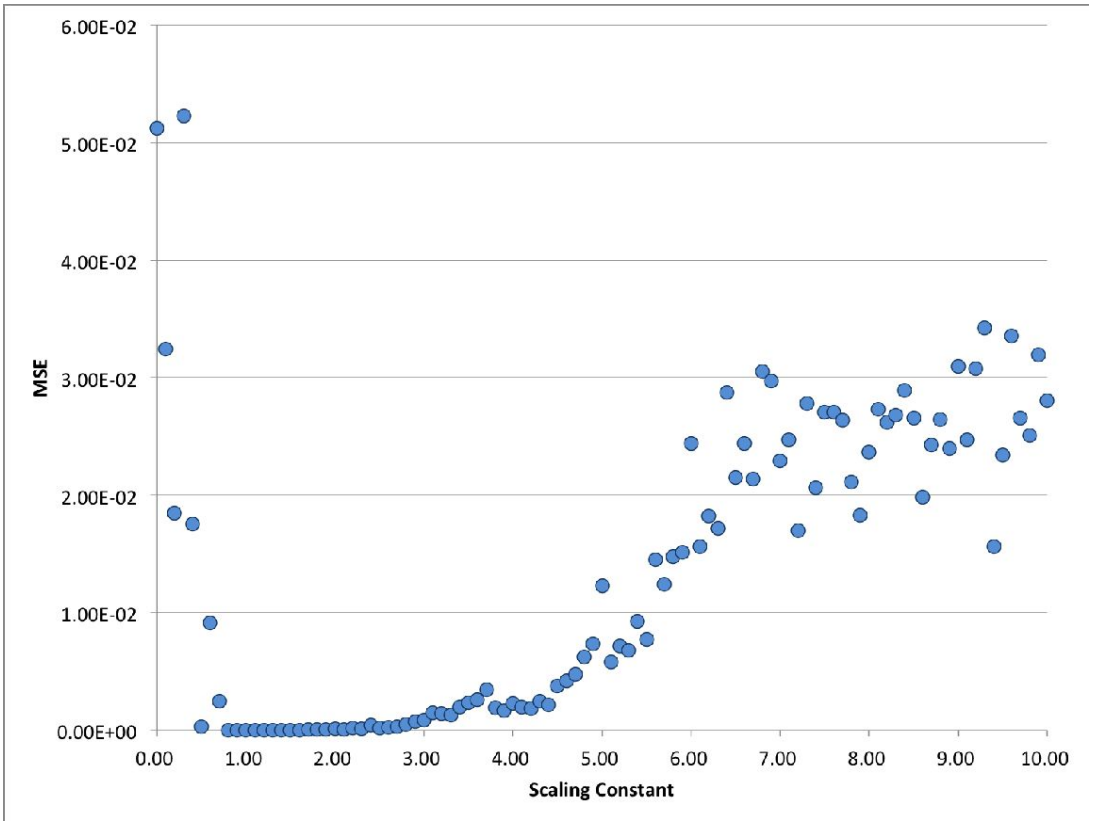


Figure 4.1: An optimal range for the bias value, regarding the minimization of MSE

On continue, as regards the output weight matrix and its impact on the error, Jenny Su underlines that no correlation could be found when plotting the first as a function of the second. The results are depicted in Fig4b

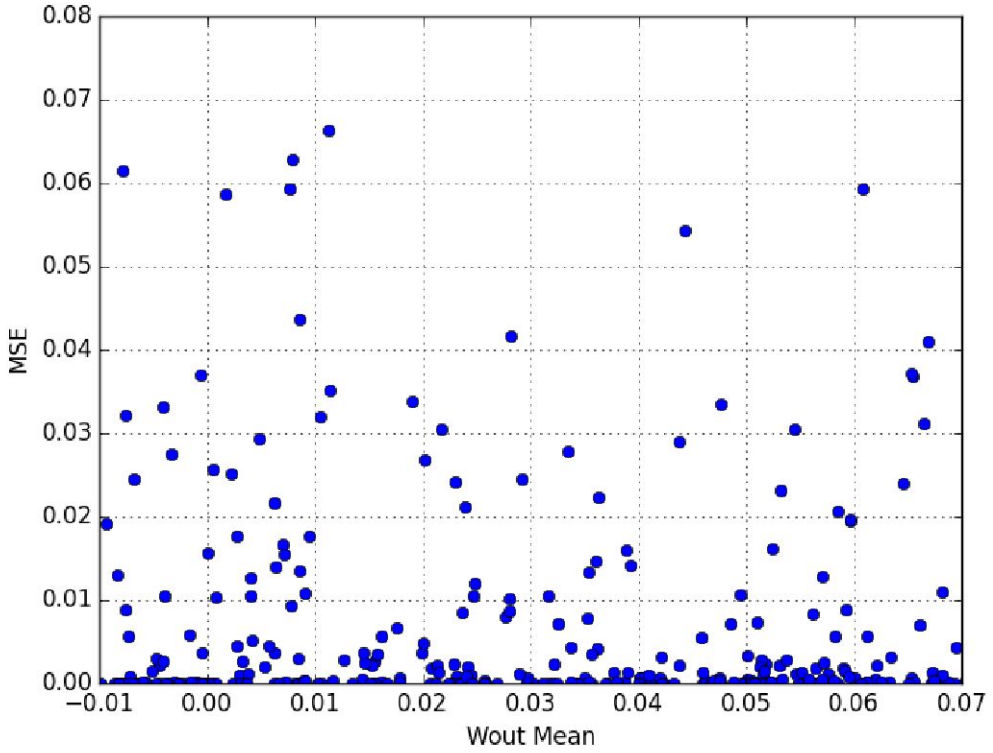


Figure 4.2: MSE as a function of the output weight matrix

After having presented a clear and robust analysis of Reservoir Computing optimization parameters, a section applying this knowledge to the stock index S&P 500 follows.

The case of S&P is also examined in [13] by Edwin Li, in his master thesis, "LSTM Neural Network Models for Market Movement Prediction". In his work, he optimizes to the best possible leven an LSTM network and makes predictions for the S&P stock market using a multivariate regressor and a binary classifier. In his analysis, he found out that the regressor lacks behind in terms of predictive accuracy and this may be caused by a plethora of possible explanations. Among others, the inherently chaotic nature of the dataset is mentioned which is a critical task that the present work wishes to address in the next chapter. Moreover, another reason playing a significant role in his result may be the amount of data used. As mentioned, « Increasing the quantity and the number of features gave better results for both models, so the old machine

learning adage that more data yields better results seems to hold true for this experiment ». Finally, the procedure of feature selection and feature engineering is underlined as a central part of experimenting with stock marketing data sets, although it is not an easy task and may need certain domain knowledge. In the next figures denoted with (4.4 a,b) numbering, I cite the optimization parameters used for both his models, in order to familiarize the reader with the parameter tuning concept, as it covers a great part of the current work too.

Window size	**Learning rate**	**#Hidden layers**
1	0.001	1
Learning rate decay	**Batch size**	**Activation**
0.95	64	ReLU
#LSTM cells	**Training epochs**	**Dropout probability**
32	25	0.2
Optimizer	**Loss function**	**#Features**
Adam	Custom loss	8

Figure 4.4a: Optimization parameters for a regression model

Window size	Learning rate	#Hidden layers	LSTM size
1	0.001	1	32
Training epochs	**Dropout probability**	**Learning rate decay**	**Batch size**
25	0.2	0.95	64
Activation	**Optimizer**	**Loss function**	**#Features**
ReLU	Adam	Cross entropy	8

Figure 4.4b: Optimization parameters for the binary classifier

Konrad Stanek [23], in his work, entitled "Reservoir computing in financial forecasting with committee methods", underlines the importance of converting a financial dataset to stationary before trying to predict or analyze it. In figure 4.5, he depicts a variation of different well-known time series (SP500, DAX, NIKKEI, and EURUSD), comparing their raw form with the stationary one. The method to convert the time series is a relative difference method, mathematically defined as:

$$y_{rel}(t) = \frac{y_n - y_{n-1}}{y_{n-1}} \tag{4.1}$$

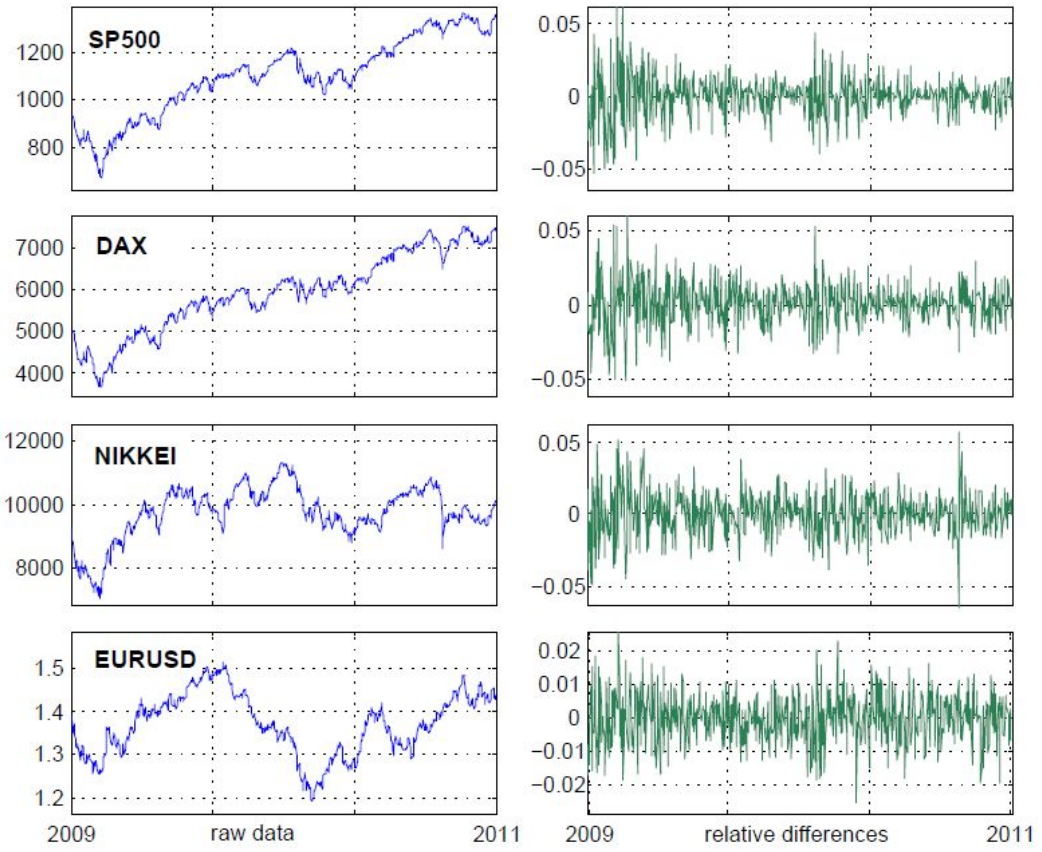


Figure 4.5: Several stock prices evolution within 2 years, raw and stationary

The goal of his work is to compare the capability of different models to predict the SP500 next-day close price. The input used is a complex combination of the above data series. Specifically, the SP500 open, minimum, maximum, close, transaction volume, with NIKKEI and EURUSD close prices. The models used were: naïve, naïve-contrarian, ARIMA, VARIMA and Reservoir Committees of MEAN, EXP, REXP, RMEAN, RIDGE. The results of the experiments showed a superiority of the Reservoir Committee models, overtaking the auto-regressive methods and the naïve method. The results are depicted in figure 4.6 where bars represent average results for every model case, with the 4 left-most bars referring to benchmarking models and the five right-most to the committees.

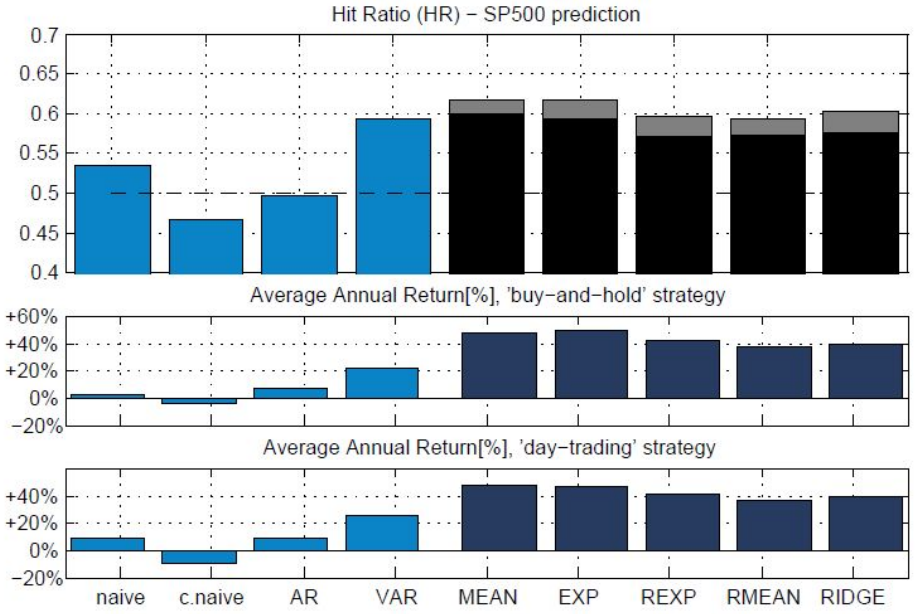


Figure 4.6: SP500 next day prediction – comparative results for different algorithms [23]

Finally, Yufan Wang [24], in his master thesis entitled as "Applications of Recurrent Neural Network on Financial Time Series" comparing a Linear Regression and Echo State Network using Mackey Glass series, marks the "shift" phenomenon that will be analyzed in Appendix#1. Specifically, he states that even if linear regression seems to produce a low error prediction, in reality, it does nothing else than a forward shift of the target values. The prediction values are just a $t+1$ lagged copy of the original ground truth values.

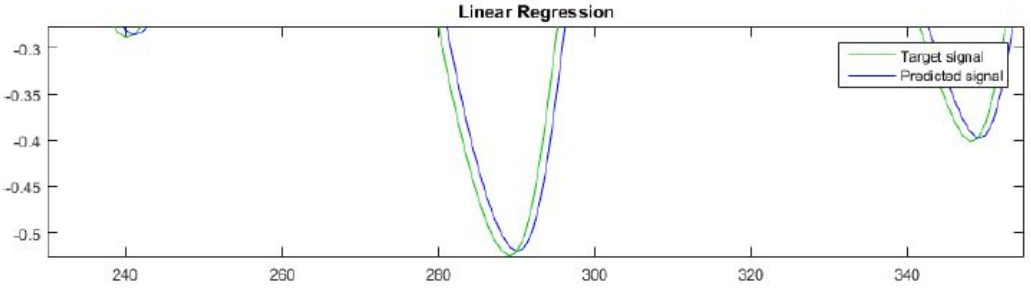


Figure 4.7: Yufan Wang's depiction of Linear Regression shifted prediction [24]

5. Results – Discussion I

In this chapter, we will be using a famous non-linear equation that comes with the name Mackey-Glass. The goal is to establish two different models for series prediction, one using the LSTM network and another using an Echo State Network to make comparisons, extract high quality predictions and expand our understanding of both systems via parameter tuning towards the minimization of the errors.

5.1 Metrics

The results and the comparisons of the next paragraphs will be based on the Mean Squared Error, which is mathematically defined as the following:

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_{rpedicted} - y_{target})^2 \tag{5.1}$$

Equation (5.1) is valid for the Neural Networks models, as described above. On the other hand, and regarding the Echo State Network, we write the above equation on the following formation:

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_{rpedicted}(n) - y_{target}(n))^2 \tag{5.2}$$

As regards the ESN though, the story is different since it works as a matrix-based model. According to the background theory provided in the 3rd chapter, the equation (5.2) should be written as [8], [17]:

$$MSE_{(ESN)} = \left(W^{out}X - Y^{target}\right)^2 \ as, \tag{5.3a}$$

$$Y = W^{out}X \tag{5.3b}$$

Where Y and X are the matrix representations of $y(n)$ and $x(n)$ in respect. By extension, the task of MSE minimization leads to a solution of the following form:

$$W^{out} = Y^{target} X^{-1} \tag{5.4}$$

The last step is to introduce a Tikhonov regularization based on a ridge regression parameter to find a stable solution, and this occurs due to the fact that the system described by the previous equation is an overdetermined one, which can result in unstable solutions. Adding the regularization term equation (5.2) we obtain the following final equation along with its solution:

$$\left(W^{out}X - Y^{target}\right)^2 - \beta \left(W^{out}\right)^2 \tag{5.5a}$$

$$W^{out} = Y^{target} X^T (XX^T + \beta I)^{-1} \tag{5.5b}$$

5.2 The Mackey-Glass equation

The Mackey-Glass equation is a nonlinear time delay differential equation originally derived from Michael Mackey and Leon Glass as a model of chaotic dynamics for medical research purposes at McGill University [23] [24].

$$\frac{dx}{dt} = \beta \frac{x_t}{1+x_t^n} - \gamma x, \quad \gamma, \beta, n > 0 \qquad (5.6a)$$

Where γ, β, n are real numbers which determine the ways that Mackey Glass equation exhibits a range of chaotic and periodic behavior. In addition, x_t denotes the evolution of x at time $(t - \tau)$. It can also be written and examined in a more flexible way as:

$$\frac{dx}{dt} = \lambda - \gamma x, \qquad (5.6b)$$

As obvious in this form, we describe a system in which x values are derived at a rate of λ and decreases at a rate of γx. For some fixed value of $\gamma > 0$ the system will tend to a state equilibrium at λ/γ, when $t \to \infty$. A simple graph (5.1) illustrating the case of Mackey-Glass equation we will be studying in the present section is given below. Taking the chance, the equation is plotted with respect to the training and testing area of the chapter results. The equation was plotted with the following parameters values [8]:

$$\beta = 0.2 \qquad \gamma = 0.1 \qquad \tau = 17$$

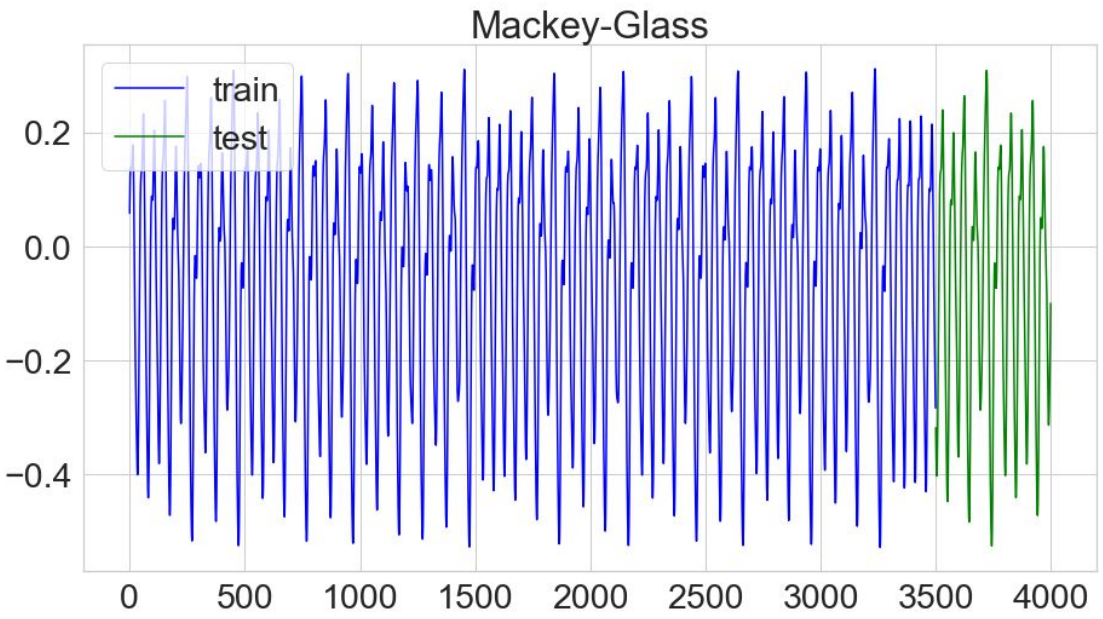


Fig 5.1: Mackey-Glass equation graph for $\beta = 0.2 \mid \gamma = 0.1 \mid \tau = 17$, demonstrating the training and the testing area

5.3 Neural Networks – LSTM

The LSTM model was build using a rolling window technique. The central idea behind rolling techniques is to over-sample your data and create new sequences of observations in a sequence form.

What a rolling window basically means is to conduct regressions over and over again with subsamples of your original full sample. In terms of our task, we feed our model with n number of days and ask a prediction for the n+1 day. Afterward, we refeed the model with [2, n+1] number of days and ask for a prediction of the $(n+2)^{th}$ day.

Table 5.1: Example of a dataset, transformed with a rolling window

$X(n-2)$	$X(n-1)$	$X(n)$	$X(n+1)$
1	2	3	4
2	3	4	5

3	4	5	6

The optimization of an LSTM model can be performed in a plethora of ways to produce an optimal result since the parameters that contribute to the training process are numerous. In this work, we will concentrate on two of them; The units (neurons) of every layer contributing to the data processing, decision propagation and error correcting procedure, and the rolling window depth. For this section and mostly for visualization reasons, we will adopt as metric the Root Mean Squared Error denoted as RMSE. Obviously, the formula to obtain RMSE is the following:

$$RMSE = \sqrt{MSE} \tag{5.7}$$

Figure 5.2 provides information on the evolution of the RMSE of the 500 testing points of Mackey-Glass as the rolling window days fed in the network increased. It is evident from the graph that there is a drop in the RMSE in the range of [4,8] days, which occurs to be the optimal range towards our problem. This range can be explained by the fact that more days may induce noise to the system and prove to be an obstacle rather than help for the hidden layers to minimize the error even more.

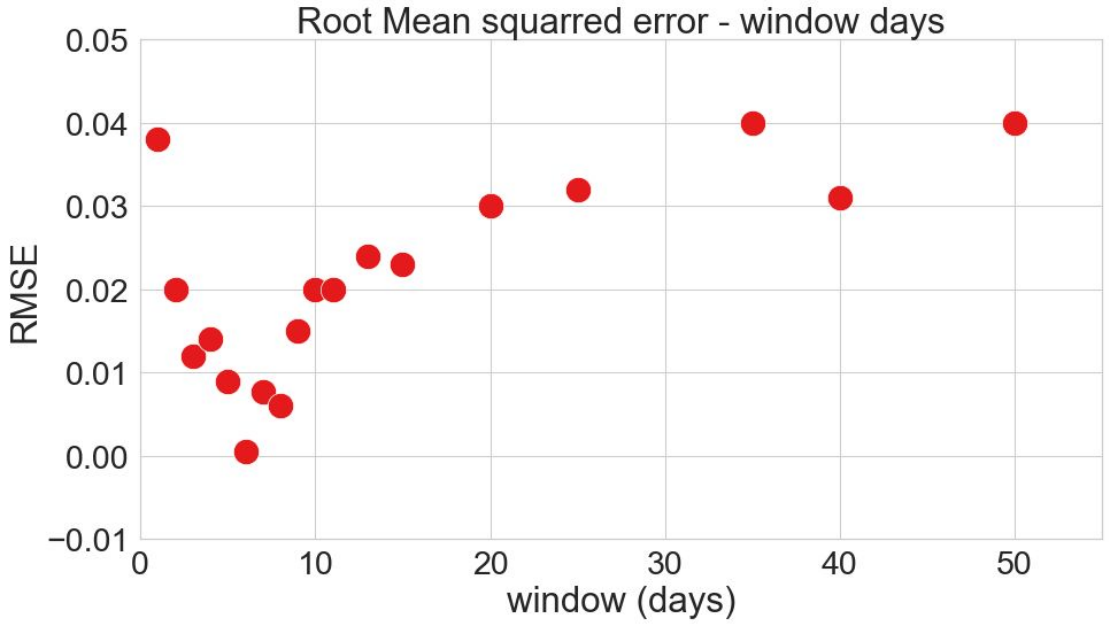


Figure 5.2: RMSE plotted against No of days used in the rolling window technique.

The next step is to start a parameter study based on the units of the hidden layers of our LSTM network. We chose to implement an LSTM with a rolling window of 6 days to be fed in every step of our training process and search the optimal number of units that will minimize the RMSE even more. Figure 5.3 depicts the results of this search. There is a clear trend showing that increasing the number of units results in the RMSE minimization, which could be expected in a way as more units will increase the variability of our dynamical model.

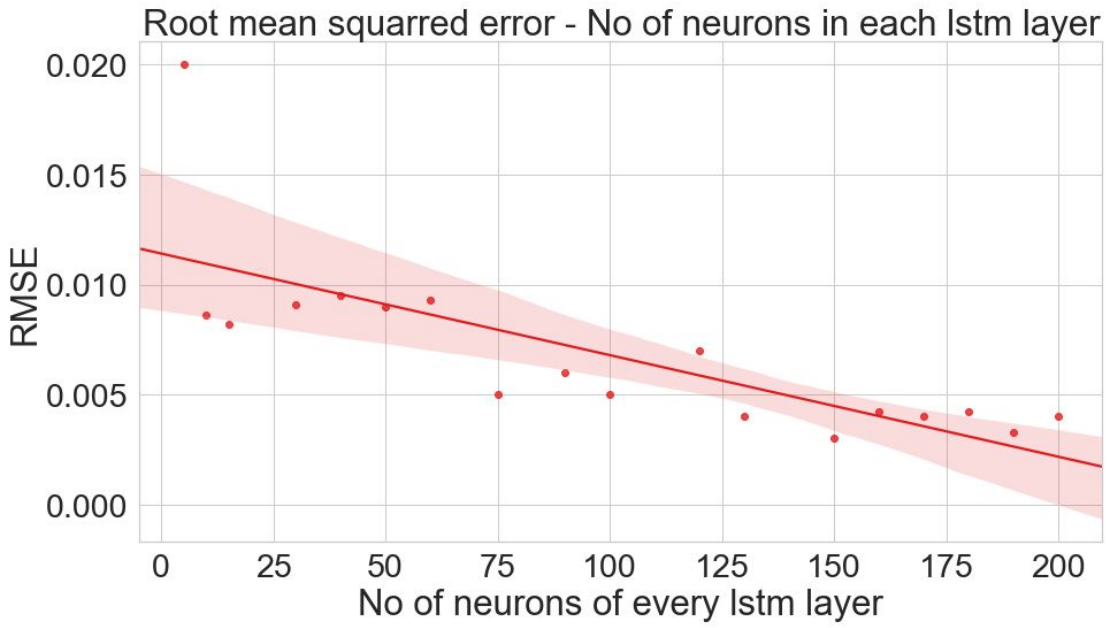


Figure 5.3: RMSE plotted against No of neurons in each of the LSTM layer used in the training process.

On continue, we have now set a basis of optimal parameters, and we can now obtain a clear result based on our gained knowledge to achieve the best possible prediction on the testing part of the Mackey-Glass equation. Firstly, all data fed to the LSTM model should be normalized. Mackey – Glass data points are converted to new values of a certain range without distorting differences in the initial ranges, though. For this task,

we convert the data points to a range of $[0, 1]$. In figure 5.4(a, b) some details related to the LSTM network are shown. Specifically, the overview of the model is depicted along with the train and validation loss evolution during the training activity. The model consists of an input layer, two hidden layers and a final output layer, all equipped with 200 units (neurons) and a 20% dropout. Dropout is a popular tactic that aims to regularize a deep neural network in a way that the overfitting risk is eliminated. Some input units, along with some units located at the hidden layers, are probabilistically excluded from activation and weight updates while the network is training. It is a computationally cheap way to improve model performance and ensure the validity of the results. The activation function used was the rectified linear unit (ReLU function) defined as $y = max(0, x)$ and the optimizing function was set to "Adam" (Adaptive moment estimation), a function that updates network weights and is the most widely used towards time series prediction,. Adam has little memory requirements, and it is appropriate for noisy and non-stationary datasets. Finally, the batch size used to train our model was set to 10. Batch size basically represents the number of sequences that are trained together. Small sized batches ensure the reliability of the fit and the randomness factor is minimized as possible.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 6, 200)            161600
_________________________________________________________________
dropout_1 (Dropout)          (None, 6, 200)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 6, 200)            320800
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 200)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 200)               320800
_________________________________________________________________
dropout_3 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 201
=================================================================
Total params: 803,401
Trainable params: 803,401
Non-trainable params: 0
```

Figure 5.4a: The used LSTM summary (overview)

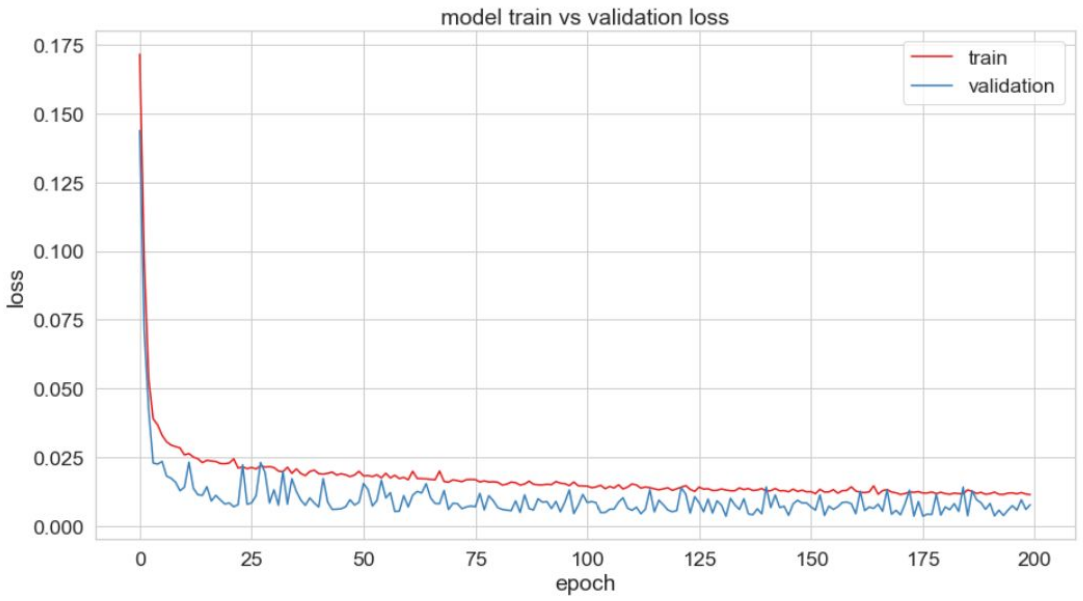


Figure 5.4b: The train and validation loss evolution through the number of epochs used.

Finally, in figure 5.5, the comparison between the original Mackey Glass testing area with the predicted signal of our LSTM network is plotted. There is no doubt that our deep learning model works and predicts high quality results, achieving a root mean squared error of 0.006.

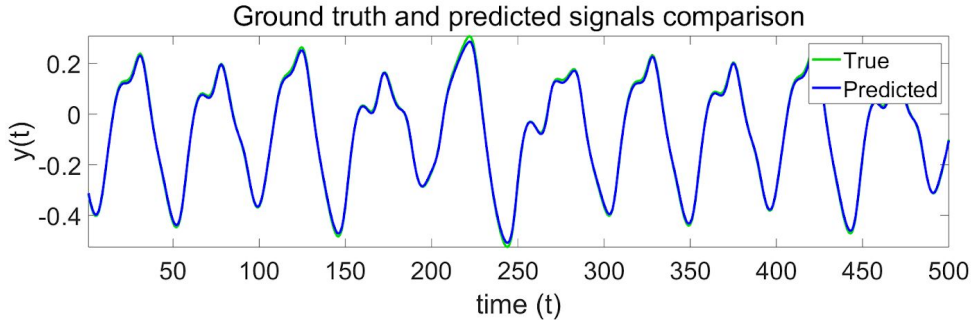


Figure 5.5a: The original Mackey-Glass test data points plotted versus the predicted values of the optimized LSTM model

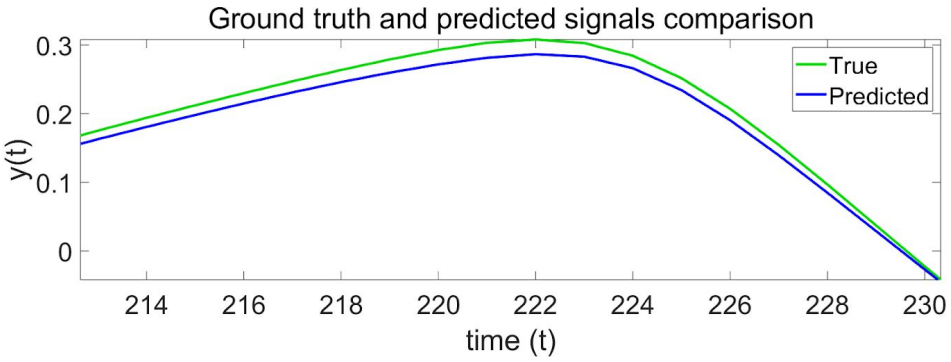


Figure 5.5b: A part of Mackey-Glass test data points zoomed in

5.4 Echo State Network

The next step is no other than go through the same procedure using the echo state network model this time. Based on the gained knowledge from the related work chapter referring to Jenny Su [8] model implementation, we are in a position to optimize an echo state network effortless. We will keep the scaling constant in the range of [1,2], as figure 4.1 suggests. We will also use a value of leaking rate $a = 0.3$ and spectral radius $\rho = 1$. Thus, the only parameter left to be tuned is the reservoir size. In figure 6, the results of several sized reservoirs are listed, along with the mean squared errors corresponding to them. There is a tendency that increasing the reservoir size up to a value around 150 results in RMSE reduction, forming an optimal range of choices between [120,180]. Using a reservoir size of 150 neurons, we tend to get the lowest RMSE regarding our testing set of the Mackey-Glass equation, achieving a value of $RMSE = 0.0015$.

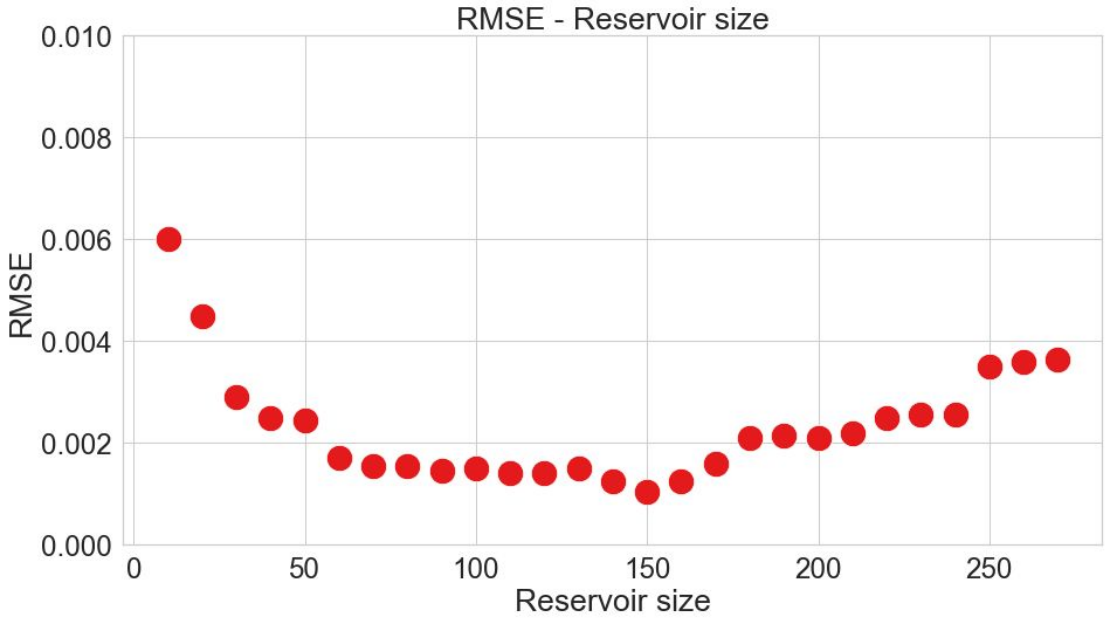


Figure 5.6: RMSE plotted against several reservoirs varying in size to obtain the optimal value

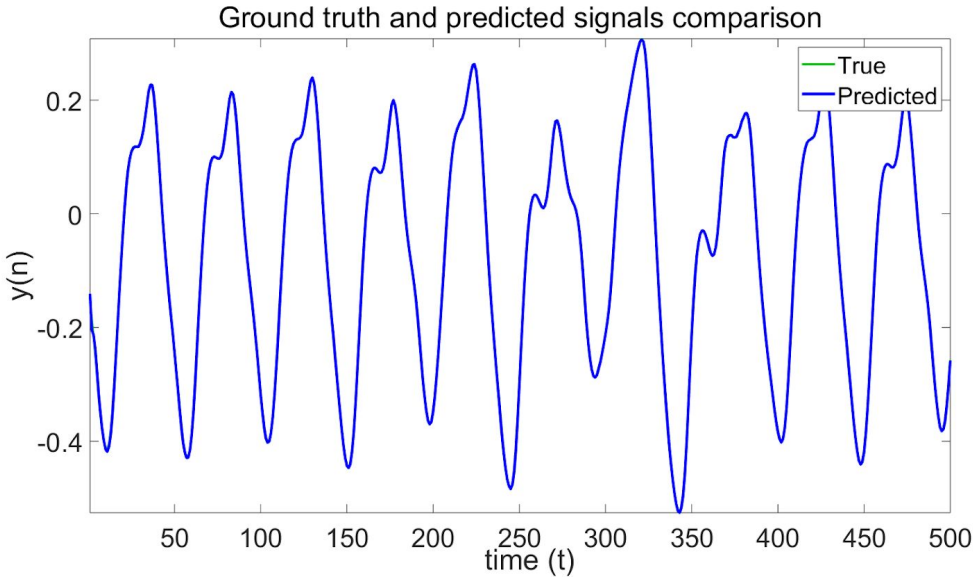


Figure 5.7a: The original Mackey-Glass test data points plotted versus the predicted values of the optimized LSTM model

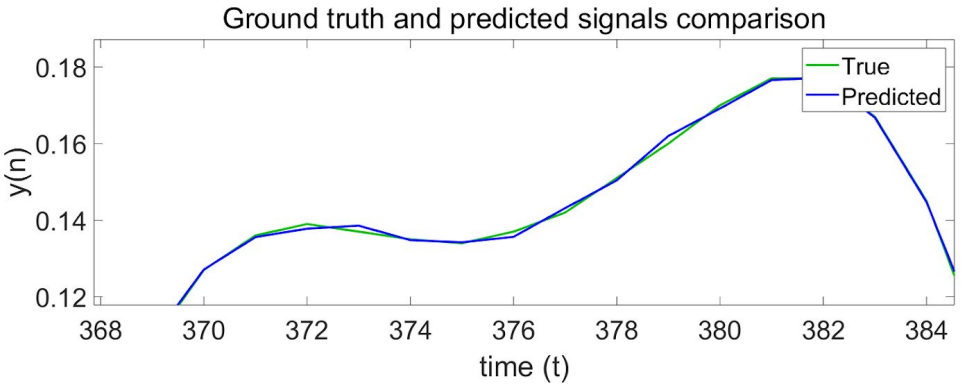


Figure 5.7b: A part of Mackey-Glass test data points zoomed in

Finally, table 5.1 provides information for the two models performances. To summarize, we achieved a high level of accuracy in terms of loss minimization in both of our models with Echo state network having the lead. Furthermore, a big advantage of the ESN compared to the LSTM network is the time needed for the training procedure. This is something expected as the training process of the reservoir computing algorithms takes place just on the output section when the reservoir is fixed. Moreover, there is no need for back propagation.

On the other hand, the training process of a neural network is slow as it requires the small batch size to ensure stable outputs, a large number of epochs to be trained, which is computationally inefficient, especially when dealing with a large amount of data. Finally, both achieve a very high percentage of correct predicted trends. This is a very interesting and key metric point when trying to predict time series, and the next chapter which also relates to non-linear but real and noisy data set will be concentrated on it.

Table 5.1: Final results of the LSTM and ESN comparison.

Model	RMSE	Training time	Trends	Trends percentage
Neural Networks (LSTM)	0.0061	35 min	494/499	98.9%
Echo State Network	0.0015	10 sec	495/499	99.2%

It should be noticed that for validation reasons and to ensure the reliability of the output results, all these tasks were performed not only once, but in 10 folds and all the above numerical values are the average values of the procedures.

6. Results – Discussion II

6.1 Stock data prediction

Neural Networks and similar models are studied extensively as deep investigation and specialization on time series predictive models hold a promise to address real world problems. Such problems are highly related to financial series analysis that govern the future of any business or association. On the other hand, financial and stock marketing time series prediction is a challenging task as it differs from dynamical systems like Mackey-Glass equation we discussed in the last chapter. The financial market corresponds to a non-linear dynamical system whose behavior is governed by noisy data and sensitive fluctuations occurring from factors that either is difficult to predict and take into consideration or totally random and unpredictable events. Stock prices can be affected by political decisions, international level change of policies, migration, physical phenomena and a plethora of other factors, hard to foresee or take into account. However, predicting tasks regarding stock marketing have emerged as a key field of study due to the potential economic profit. In this chapter, a series of techniques to transform the data into a more suitable way as regards the optimization of the output predictions will be presented, along with some experiments to evaluate the techniques using the recurrent neural network presented before [23].

6.2 Data Normalization

Normalization is a process related to data preparation for machine learning tasks of high complexity. The final goal is to convert the raw values of numerical columns to a standard form within a determined scale, without distorting differences in the ranges of the column values. In the previous chapter, the need to normalize data into the

scale of (0,1) was mentioned. Neural Networks demand normalization of data before getting fed to the model. Moreover, normalized data within the same scale make it easier for the gradient descent to converge more rapidly and accurately. The most popular way to normalize data is the one already used before in the previous chapter. That is to convert the data to a certain range $[a, b]$, where the minimum value of the dataset is mapped to α and the largest to b, with respect to other values. A small example is given below:

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{6.1}$$

6.3 Denoise Process

The vast majority of financial time series are noisy in terms of irrelevant information, which leads to random fluctuations that confuse a machine learning model in its effort to extract patterns through the series. By extension, the model's performance may be reduced significantly, crash, produce chaotic results or even present a shift effect (see Appendix#1). There are a plethora of different techniques to address the noise reduction problem, such as wavelet filter, TISEAN or Gaussian filtering. For our scope, the built-in denoising library of Matlab will be used in our experiments that uses an empirical Bayesian method with a Cauchy prior.

6.4 Stationarity of time series

In an intuitive sense, stationarity in terms of time series analysis means that the statistical properties of the series should remain constant over time. Stationarity ensures that properties like the mean, variances, and correlations accurately describe the data at its whole shape. Thus, it should not exhibit any trends upward or downward or seasonal effect, and at the same time, mean, variance or covariance

should be constant with time. On the other hand, a non-stationary time series will make a model describing the data, varying in accuracy at different time points. To check for stationarity, there are two common methods.

- Visual test: Plot the data and check for any trends, seasonality patterns and the evolution of statistical measurements

- Statistical test: The unit root ($a = 1$) indicates that the statistical properties of a time series are a function of time. The most popular statistical test is the ADF (Augmented Dickey Fuller) test, and we will be using it as an example in this paragraph to check the stationarity of a dataset and convert it to stationary.

In figure 6.1 the rolling mean and the rolling standard deviation is plotted against time along with the data points of time series. As obvious, there rolling mean increases over time, following the general upward trend of the series. The rolling standard deviation seems to increase, too, but at a slow rate.

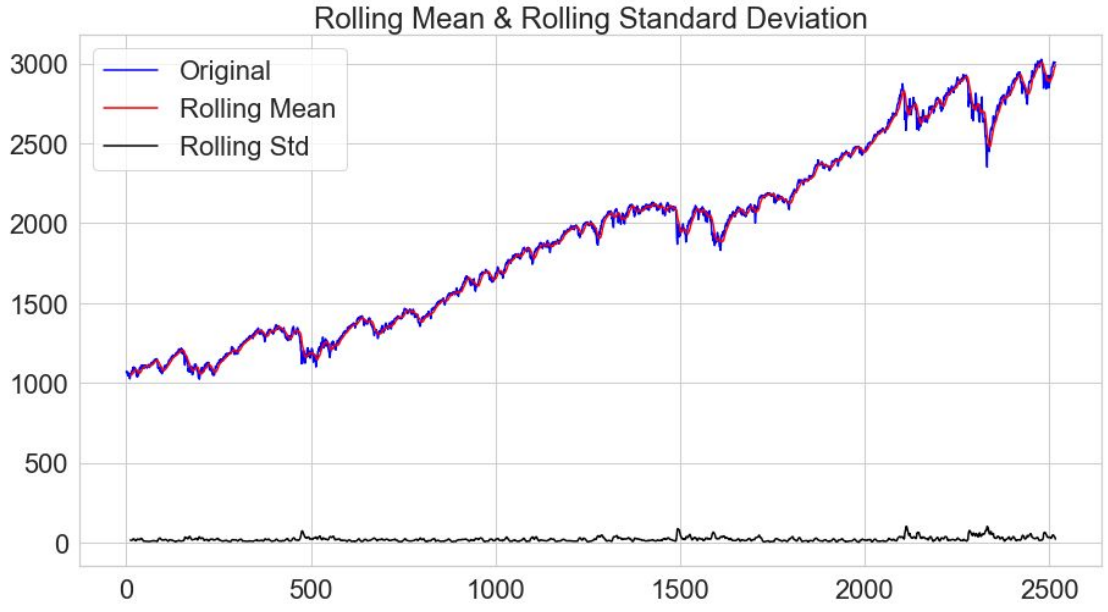


Figure 6.1: S&P500 plotted, along with the standard deviation and the rolling mean.

A time series can be considered as stationary if the p-value is low (null hypothesis), and the critical values at 1%, 5%, 10% confidence intervals are close to the ADF statistics. The null hypothesis suggests that if it holds, the time series has a unit root. Thus, it does not exhibit any time dependency on the structure and that can be checked through the p-value. If the p-value is greated than the threshold $p > 5\,\%$, the null hypothesis cannot be rejected and the time series are not considered to be stationary. In addition, the ADF statistic value is another parameter to be taken into account regarding the process. The more negative is this value, the more likely that the null hypothesis gets rejected.

Following the ADF test, we compute the values depicted in the schemas for three different strategies proposed for stationarity conversion. In fig6.2a, we apply the logarithm function to every point of the series and subtract the rolling mean. In 6.2b we add an exponential decay to the rolling mean before the subtraction, while in 6.2c we apply time shifting, subtracting every point by the one that preceded it and dividing by the point. The mathematic formula is the following:

$$x_{rel}(t) = \frac{x_n - x_{n-1}}{x_{n-1}} \tag{6.1}$$

It should be noticed here that there exist several methods based on data points differences that all aim to make the series stationary. Examples are the simple differences between points, denoted as

$$x_{dif}(t) = x_n - x_{n-1} \tag{6.2}$$

or log differences denoted as:

$$y_{log}(t) = log(\frac{x_n}{x_{n-1}}) \tag{6.3}$$

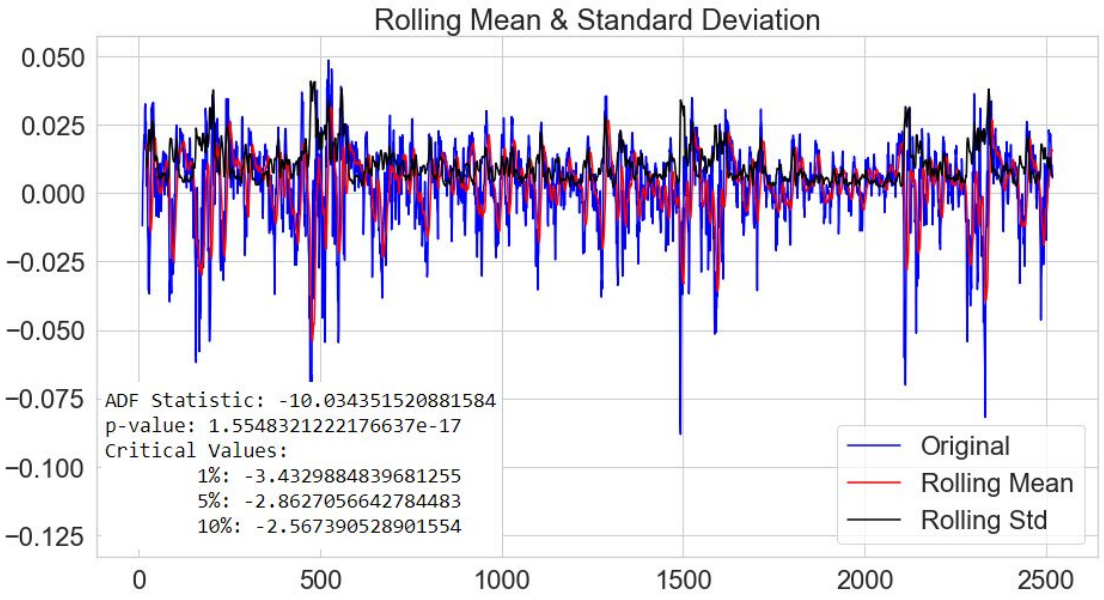


Figure 6.2a: Logarithm differences.

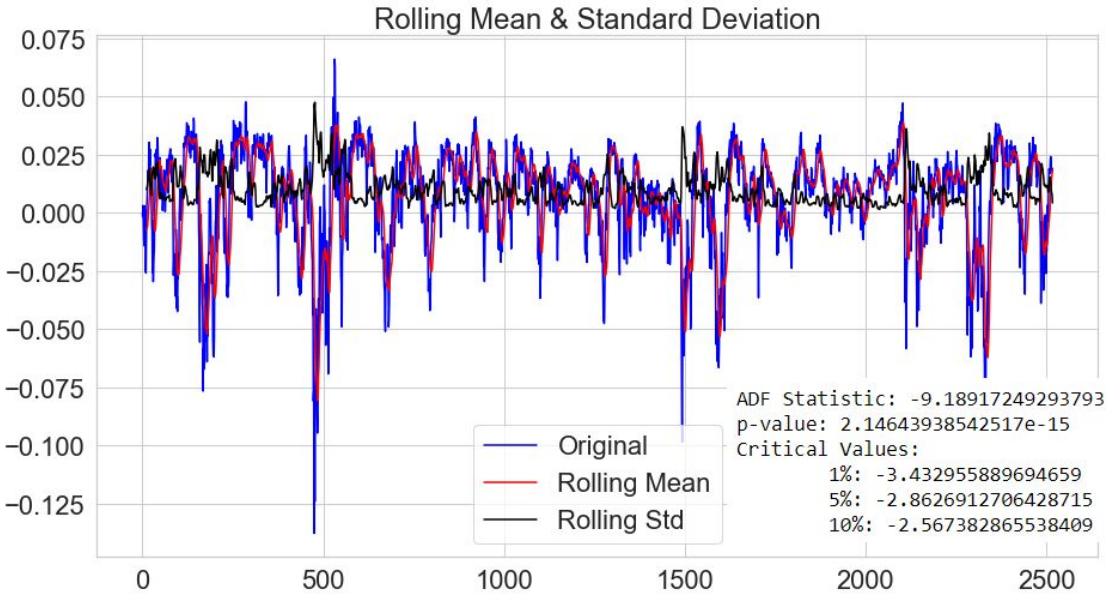


Figure 6.2b: Exponential decay.

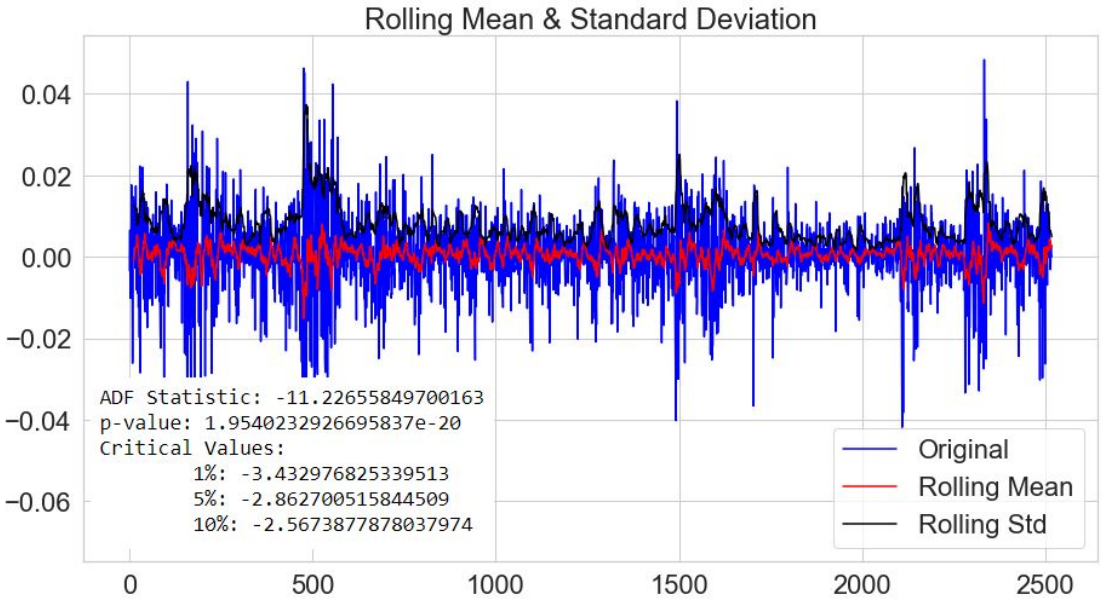


Figure 6.2c: Relative differences.

Taking a quick glance at the ADF critical values and p-values depicted on the inset of every graph, we can easily conclude that all techniques result in series stationarity. ADF statistic value becomes more negative along with p-value which also becomes smaller through our experiment process. All the above implemented techniques are fully capable of turning a time series in a shape that is characterized by time depended on the structure. In this work, we will be using all these techniques along with the denoising matlab tool and the normalization processes, that will derive a final data series form to be fed into our LSTM neural network.

6.5 S&P 500 time series: Trend analysis

In this section, we will be using a train set, derived from the original S&P 500 index. S&P500 or just S&P is a stock market index that measures the stock performance of the 500 largest U.S publicity traded companies. As described in Appendix(I), when trying to predict stock prices or financial data, in general, we encounter the "Shift"

effect. To address this issue, we need to feed our models with more data than just a sequence of series (i.e. a column). More data and columns increase the model's complexity but at the same time, we can build more accurate and reliable models. S&P is found with several columns of data; the "opening" value of the price, the "lowest" value of the day, the "highest" value of the day, and the "closing" value are the ones we will be using in this chapter. Our model will be trained to the values "open", "low", "high", "close" of day t, the "open" value of day $t+1$, and it should predict the closing value of $t+1$. The main reason for this thought is what leads to something productive in terms of using. The values of the previous day, combined with the opening value of the current day, give a potential stock marketing analyst a "window" of several hours to consult his clients on price changes. Our neural network built for the case consists of two hidden layers, and the summary is depicted in figure 6.3. The input sequences are once again mapped to the range $(0, 1)$ by the MinMaxScaler method, and there will be no rolling window technique using since the final goal of the paragraph is to compare some stationarity methods when predicting a real stock marketing price day to day.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 1, 100)            42400
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 100)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 100)            80400
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 100)            0
_________________________________________________________________
lstm_3 (LSTM)                (None, 100)               80400
_________________________________________________________________
dropout_3 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 203,200
Trainable params: 203,200
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.3: Summary of the LSTM model used for the S&P case.

Figure 6.4, 6.5, and 6.6 illustrate all the output plots with respect to every different data preprocessing technique implemented. Every dataset in this figure is normalized, as normalization is the most basic step towards stationarity. Moreover, as described in chapter 5, it is an essential process for the LSTM to operate. Secondly, in figure 6.4, the normalized and denoised S&P is plotted. As obvious, the denoising tool had a strong impact on the model's performance.

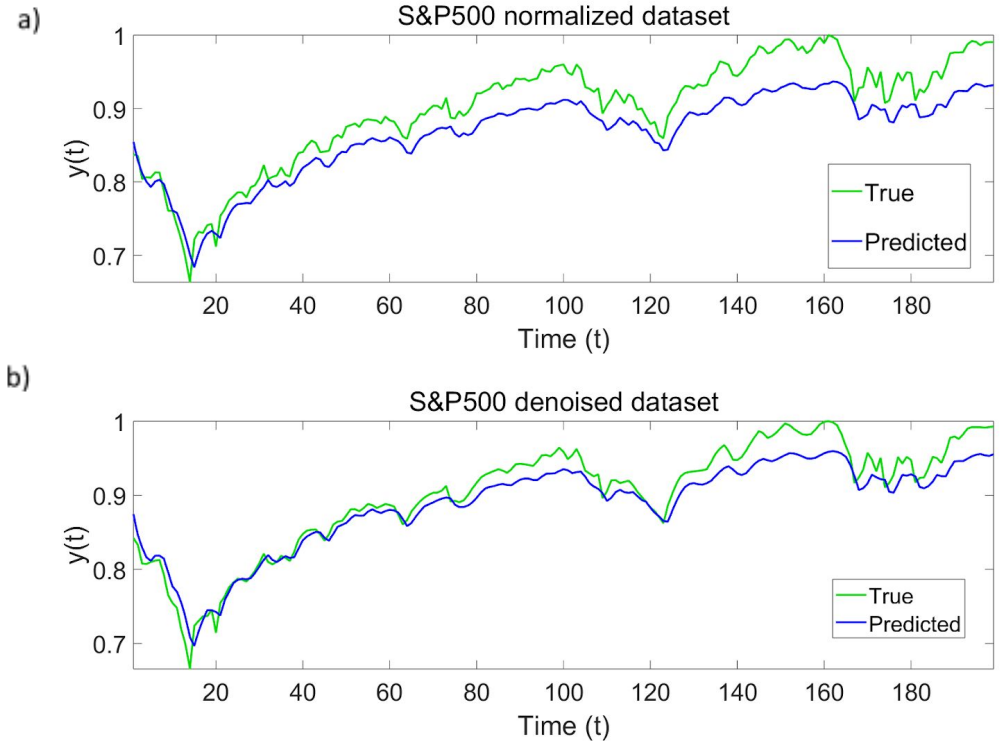


Figure 6.4: S&P series predictions a) Normalized data b) Denoised and normalized data.

Regarding the evaluation process of normalized and denoised techniques, we will use a new metric since MSE or RMSE cannot be a reliable indicator when trying to predict trends. The new metric is a random choice metric. In simple words, we want to know how much better our algorithm performs compared to a random observer. Osbourne [27] was the first to give the theoretical justification for price changes fitting a gaussian distribution using a central limit theorem; he described transactions

as randomly independent and identically distributed. The second step coming from [23], and as Conrad Stanek states, "Market indices rarely change by more than 3% per day, what corresponds to a modest change of input value ~ 0.03". When fitting a gaussian distribution, our $[x-3\sigma, x+3\sigma]$ range, corresponds to 99.73% of our observations. The random choice for the next observation must be bounded, as, given that on day t we get a value of x_0, then on $(t+1)^{th}$ day, we will get:

$$x_1 = x_0 \pm 0.03x_0, \; with\; x \in \mathrm{N} \tag{6.4}$$

and thus, we can write,

$$\bar{x} + 3\sigma = \bar{x} + 0.03\bar{x} \tag{6.5}$$

Solving the above system of two equation, we get a standard deviation of $\sigma = 0.01\bar{x}$, and this is the appropriate formula to use when trying to simulate a random choice observer predicting trends.

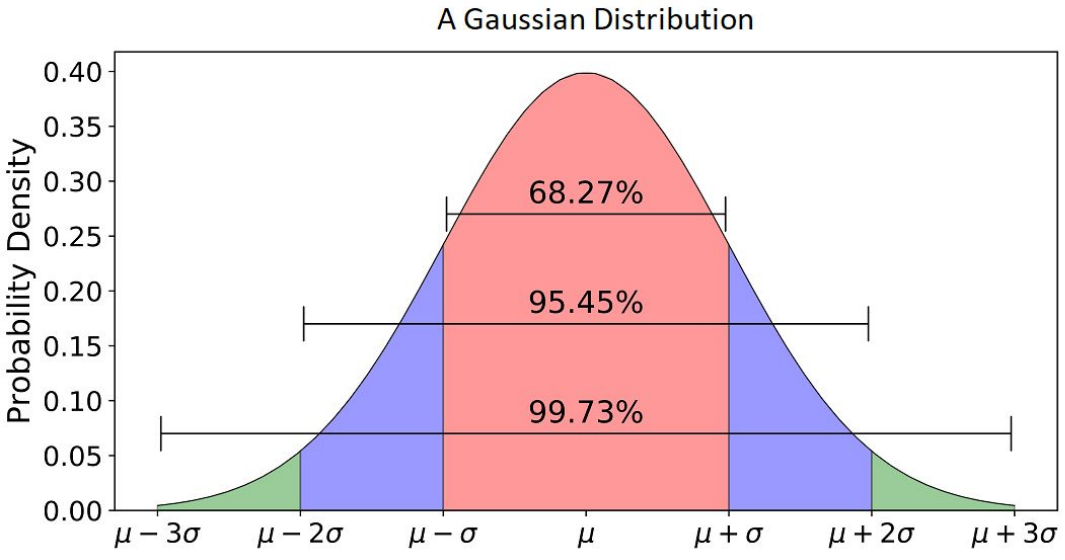


Figure 6.5: A gaussian distribution [28].

The results are depicted in table 6.1. Both normalized and denoised series exhibit a more successful prediction related to the random choice predictions (RCP). The normalized S&P achieved a score of 117/199 trends prediction, which is around 59% success while RCP is around 51% (~102/199 trends). Furthermore, our denoised S&P model seems to improve the current predictions to 142/199 trends which are around 71% while RCP is again way lower and around 53% (~105/199 trends)

Table 6.1: Results on normalized and denoised data against a random choice predictor.

S&P series	Trends predicted	Random choice prediction
Normalized S&P	117/199 ~ (56%)	102/199 ~ (51%)
Denoised S&P	142/199 ~ (71%)	105/199 ~ (53%)

On continue in figure 6.4b, the S&P series are transformed using some already mentioned difference techniques. As already seen, these are powerful methods to address stationarity issues and trends analysis since subtracting x_{t-1} from x_t already gives away the trend. Random choice prediction is of low significance in this case and, thus, we will concentrate on comparing the results with the trends predicted by our models.

The results are depicted in figure 6.6, and figure 6.7, and overall results of all techniques used are shown in table 6.2 where every single technique used is compared using the trend predicting criterion.

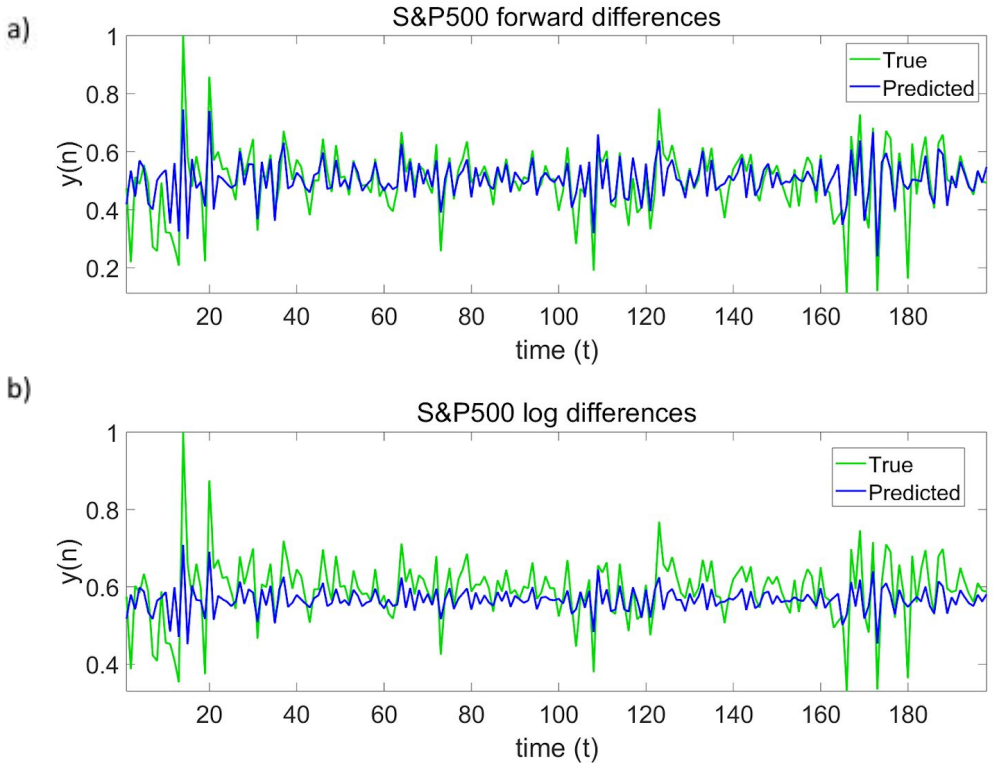


Figure 6.6: S&P series predictions a) Forward differences b) Log differences.

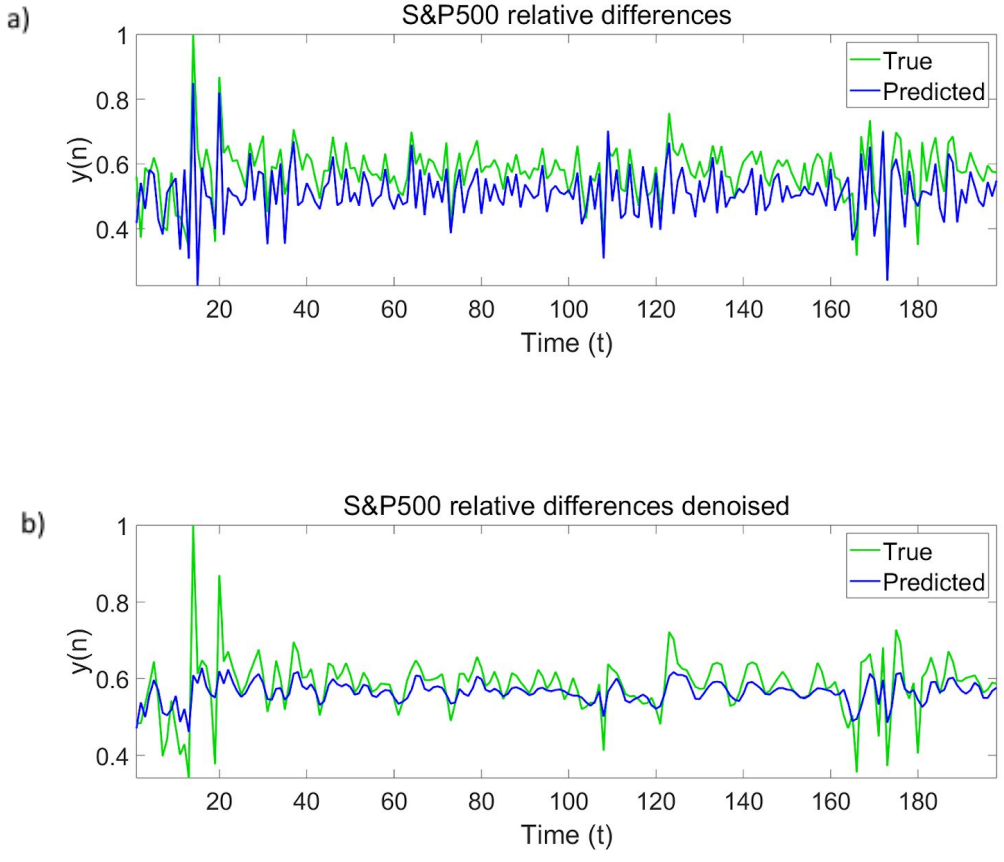


Figure 6.7: S&P series predictions a) Relative differences b) Relative difference on denoised series.

Table 6.2: Overall results on trend predictions

S&P series	Trends	Percentage
Normalized S&P	117/199	58.7%
Denoised S&P	142/199	71.3%
Forward Difference S&P	150/198	75.7%
Log difference S&P	156/198	78.8%
Relative Difference S&P	160/198	80.9%
Denoised Relative Difference S&P	152/198	76.7%

Overall, our model's results indicate that the most successful method is the relative difference of the S&P series, which performs slightly better than the log differences, which is 80.9% and 78.8%, respectively. The forward difference comes third with 150/198 correct predicted trends. Obviously, using differences methods improved the prediction ability of the network significantly.

On continue, since the relative difference method achieved the best score, we were interested in exploring the possibility of two different methods working together. Firstly, the SP500 was denoised by the proper matlab library, and then relative differences were implemented on the denoised SP500. The performance of the model was slightly decreased to 152/198 trends which are around 76.7%, having predicted 8 correct trends less than the relative difference technique implemented alone. The results are depicted in the following table:

Table 6.3: Relative Difference technique and denoised relative differences on SP500.

Relative Difference S&P	160/198	80.9%
Denoised Rel.Diff S&P	152/198	76.7%

The fact that a denoised relative differenced S&P is not better than the normalized relative difference method may be explained by the fact that any different technique when subtracting every next value, is actually denoising the data and applying this technique to an already denoised series may be a redundant action that is not

beneficial for the predicting procedure. On the other hand, as mentioned above, there are plenty of others denoising tools except for the matlab library used, which may be more appropriate and efficient, and that sets a question worth exploring further.

6.6 Where ESN stands?

In the last paragraph, a comparison was carried out among different stationarity techniques to find the one that leads to optimal results and leads to the highest percentage of correct predicted trends for SP500 stock index. In this section, we will be using a code that was written by MSc fellow Panagiotis Tziatzios to compare an Echo State Network with our optimized Neural Network. Panagiotis Tziatzios, in his master thesis entitled «Financial Nonlinear Time-Series Analysis and Prediction with Reservoir Computing», implements and optimizes an ESN using a rolling window technique to compare predictions on financial datasets prices. Firstly, we need to set the parameters of the ESN to meet our needs regarding the comparison process. Since we use a day to day prediction in the present analysis, we need to disable the rolling

window attribute of the ESN, setting "*window_value* = 1". The ESN parameters used are the following:

$$\textit{Reservoir size}: \ 300 \quad | \quad \textit{Spectral radius}: \ 0.7$$
$$\textit{Leaking rate}: \ 0.3 \quad | \quad \textit{Input scaling}: \ [-1, 1]$$

Feeding our optimal dataset (relative differences) to the ESN we get our comparative predictions depicted in table 6.4 and a plot (figure 6.8), visualizing the evolution of true values versus the predicted ones. Finally, the barplot 6.9 illustrates a total overview of this paragraph in terms of successfully predicted trends. As seen, the Neural network model overtakes the ESN in the concept of trend analysis achieving 8 more successful predicted trends. On the other hand, once again, the training time difference between the two models is chaotic. The ESN proves to be capable of producing high quality results in terms of series analysis within seconds, whereas the NN needs a significantly larger amount of time

Table 6.4: Comparison between NN and ESN on trend predictions.

Model	Trends	Percentage	Training Time
Relative Differences Neural Networks	160/198	80.9%	25min
Relative Differences Echo State network	152/198	72%	10sec

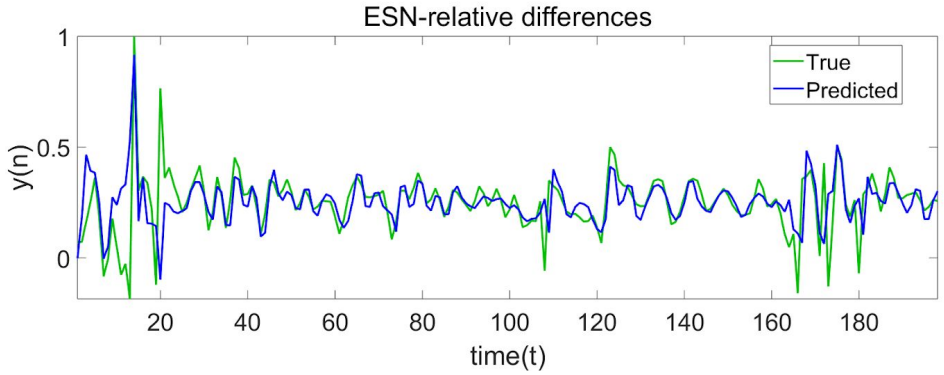


Figure 6.8: True and predicted values of the ESN plot.

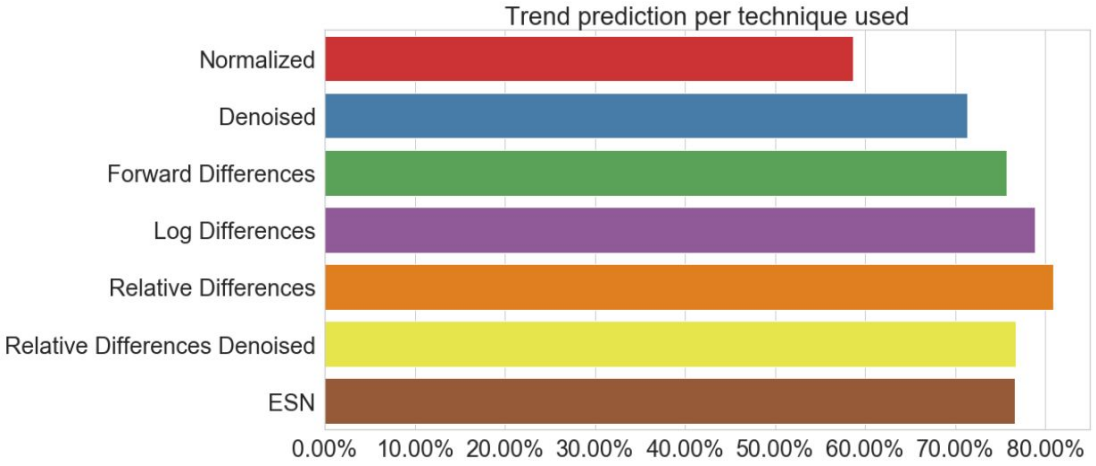


Figure 6.9: Visual comparison bar plots with respect to the technique used.

7. Conclusions

7.1 Discussion

The scope of this thesis was to delve into financial forecasting area and time series analysis developing a theoretical framework of Neural Networks and Reservoir Computing, able to predict day by day prices and trends. The first step to address this issue was to start from Mackey-Glass, the nonlinear time delay differential equation to test our algorithms and optimize their parameters, maximizing the output results to the greatest possible extent.

The Neural Network optimization suggested setting a relatively small number of window days. This can be explained by the fact that feeding a great number of days to a neural network system before predicted a single day might induce noise that rather confuses the model than helps in. Moreover, the general trend regarding the ideal use of a number of neurons of every layer shows a stable plateau in the range of $[150, 200]$, which is not a small value, so a dropout parameter was also necessary. On the Echo State Network's side, we used some parameters already suggested by Jenny Su's master thesis [8], along with a reservoir size derived from our ESN's optimization. Overall, both algorithms show remarkable performance in predicting the values and trends of Mackey-Glass equation system. The root mean squared error for NN was 0.0061, and that of ESN was 0.0015.

Furthermore, the trend prediction for both models was greater than 98%. Concluding, the only great difference between the two algorithms was the training time. The training time is the greatest asset for Reservoir Computing as it takes place only in the output layer, as described in the theoretical part of the thesis. The ESN was able to produce results slightly better than NN only in 10 seconds for every iteration where the NN system was computationally and time demanding to the extent of ~30 minutes for every training process.

On continue, in chapter 6, an analysis of stationarity techniques was implemented and the main methods to achieve it. Financial data such as SP&500 require a great deal of preprocessing actions to be converted into a more suitable form for the algorithm.

After delving into stationarity methods, we proceeded with trend analysis on SP&500 data series using 6 different preprocessing techniques. Specifically, we found that the optimal method was to implement differences techniques rather than denoising and normalizing that lack behind in terms of correct trend predictions. The most successful method was Relative Differences that achieved a score of 152/198 ~ 76.7% on trend prediction. Moreover, a final comparison is made with an ESN model. This time, neural networks seem to be more accurate since the ESN's predicted successful 152/198 trends, 8 less than the NN. The training time as regards ESN, was again incomparably small.

Overall, the present thesis displays two algorithms that seem to be interesting paradigms for understanding the fundamental concepts of time series analysis along with quality predictions that could be applied in stock return applications. Some final points to list are the following:

- "All models are wrong, some are useful" as statistician George Box states. The conditions under any Machine Learning model works should be always well tuned and adjusted to every try, as every prediction is a different problem.

- Carefully convert the data series to a form that can be exploited by the model. This will ensure the reliability of the results, and avoid effects like overfitting, underfitting or "shift" effects.

- Carefully evaluate the results, with the proper metrics. A small RMSE metric maybe not be enough to indicate that any model works as it should

- More features, data and information may be added to the model. In the present work, adding features such us lowest and highest day prices helped the model to extract time dependent patters.

7.2 Future work

Time given, there exist many optimizations and experiments we could carry out and expanding this work or improve some results. Some of them are listed:

- Adding a rolling window to the last models regarding SP&500, both in NN and ESN and make new comparisons of the models.
- Add even more stationarity methods to this work
- Expand the current work to more financial data and stock series

8. References

[1] Usage1, “Heaton, J. B., Polson, N. G., and Witte, J. H. (2017) Deep learning for finance: deep portfolios. Appl. Stochastic Models Bus. Ind., 33: 3– 12. doi: 10.1002/asmb.2209.”

[2] J. B. Heaton, N. G. Polson, and J. H. Witte, “Deep Learning in Finance,” no. February, pp. 1–20, 2016.

[3] R. Culkin and S. R. Das, “Machine Learning in Finance : The Case of Deep Learning for Option Pricing Artificial Intelligence : A Reincarnation,” pp. 1–15, 2017.

[4] A. Greek, “1 . Introduction : Deep Learning and Philosophy.”

[5] T. D. Science, “towards philosophy Deep Learning.”

[6] E. Kant, “http://www.gutenberg.org/ebooks/6343.”

[7] Lobes, “https://www.superdatascience.com/blogs/the-ultimate-guide-to-recurrent-neural-networks-rnn.”

[8] J. Su, “Reservoir Computing in Forecasting Financial Markets,” 2015.

[9] D. co. tutorials- RLanguage, “https://www.datacamp.com/community/tutorials/neural-network-models-r.”

[10] “https://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimiza

tion/."

[11] SChemas, "http://colah.github.io/posts/2015-08-Understanding-LSTMs/."

[12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] E. Li, "LSTM Neural Network Models for Market Movement Prediction," 2018.

[14] O. paper LSTM, "http://www.bioinf.jku.at/publications/older/2604.pdf."

[15] photos taken LSTM, "https://colah.github.io/."

[16] H. Jaeger, "Long Short-Term Memory in Echo State Networks: Details of a Simulation Study," *Report*, no. 27, p. 29+, 2012.

[17] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 127–149, 2009.

[18] "https://nicolabernini.gitbook.io/paperssummary/quantummachinelearning/paper-readthrough-quantum-reservoir-processing-1."

[19] M. Lukoševičius, "A practical guide to applying echo state networks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7700 LECTU, pp. 659–686, 2012.

[20] M. Lukoševicius, "Echo state networks with trained feedbacks," *Networks*, no. 4, 2007.

[21] G. Wainrib and M. N. Galtier, "A local Echo State Property through the largest Lyapunov exponent," *Neural Networks*, vol. 76, pp. 39–45, 2016.

[22] H. Zhang, J. Liang, and Z. Chai, "Stock prediction based on phase space reconstruction and echo state networks," *J. Algorithms Comput. Technol.*, vol. 7, no. 1, pp. 87–100, 2013.

[23] K. Stanek, "Reservoir computing in financial forecasting with committee methods," 2011.

[24] Y. Fang, “Applications of Recurrent Neural Network on Financial Time Series,” pp. 2016–2017, 2017.

[25] mackey original Paper, “Michael C Mackey, Leon Glass, et al. ‘Oscillation and chaos in physiological control systems’. In: Science 197.4300 (1977), pp. 287–289.”

[26] L. Glass and M. Mackey, “Mackey-Glass equation,” *Scholarpedia*, vol. 5, no. 3, p. 6908, 2010.

[27] NormalDistr, “M. F. M. Osborne. ‘Brownian Motion in the Stock Market’. In: Operations Research 7.2 (1959), pp. 145–173. issn: 0030364X.”

[28] NDistr, “https://towardsdatascience.com/understanding-the-68-95-99-7-rule-for-a-normal-distribution-b7b7cbf760c2.”

[29] Adobe, “https://finance.yahoo.com/quote/adbe/financials?ltr=1.”

[30] Brownian, “https://en.wikipedia.org/wiki/Brownian_motion.”

[31] E. Albert, “Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen A. Einstein.”

[32] T. of speculation Bachelier, “https://pdfs.semanticscholar.org/bba7/101ed8278893a5bd205614fce948628af8e3.pdf.”

[33] MarketisRandom, “https://www.worldscientific.com/doi/pdf/10.1142/9789814566926_0002.”

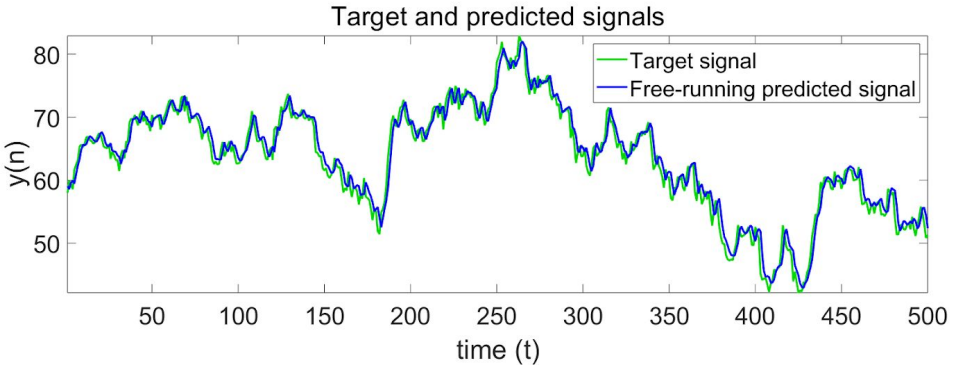
[34] RationalTheoryofprices, “https://www.ingegneria.unisalento.it/c/document_library/get_file?folderId=1344637&name=DLFE-157230.pdf.”

[35] Fema, “en.wikipedia.org/wiki/Efficient-market_hypothesis.”

9. Appendix #1

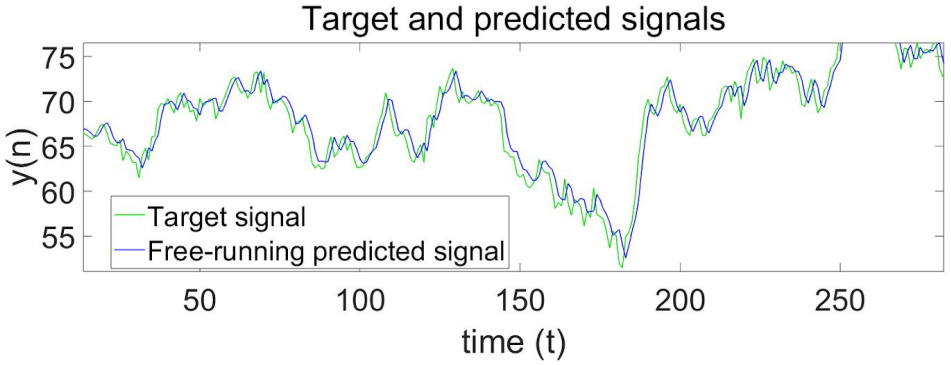
This last short chapter is about a problem we encountered during the evaluation of some predictive models – both in reservoir computing and neural networks, especially when trying to predict the evolution of a stock index or a financial data series. To illustrate the phenomenon figure A#1.1 depicts an adobe stock index series[29], along with a prediction line produced by an LSTM model. The data was split into a training test of 4200 data points and a test set of the last 800 stock values. The model is trained to predict the value at $t+1$ after being trained up to t (day to day prediction). The result seems to be quite impressive and all the usual regression error metrics are also minimal. Specifically:

$$RMSE : 1.71 \quad | \quad MAE : 1.33 \quad | \quad R^2 : 0.96$$



A#1.1: Target and predicted signal at adobes dataset test set

Unfortunately, even though our model seems to give accurate predictions something is fundamentally wrong. In figure A#1.2 a zoomed area of the previous plot is illustrated. As seen, what the model actually does is far from predicting processes. It just uses the data value at time t as the predicted value for time $t+1$. The depiction of the prediction results is merely a new set of values drawn from the real values. At first glance, this is a problem that should be resolved by applying some denoising tools or converting the data series to stationary, but that is not always the solution to this issue. There is not enough bibliography that delves into the "Shift" effect to elaborate or present solutions and work-around ways. In the current work, the problem was undertaken by choosing to train the model on a multivariate scheme (Open, low, high, close values), which may have led to noise reduction and creation of time dependent patterns that the model was able to detect. Other papers with related work overcome this issue again by combining different stock prices or adding seasonality terms to create recognizable time dependent patters.



A#1.2: Target and predicted signals zoomed in. The blue (predicted) data line is shifted by 1 time point ahead.

The "Shift" effect can be explained as a logical reaction by the model that tries to minimize the error through the predicting procedure. Since there is no time pattern in many stock and financial data series, the model behaves like trying to predict a completely random walk. The tremendous and numerous fluctuations of the series are too complex for the model to make patterns out of it, so it maps the t data value to the $t+1$ predicted value.

Some argue that these fluctuations and the series that seem to be a random walk are connected to the Brownian motion (or pedesis)[30], which is a physics term to describe the random motion of particles in a liquid, that occur from their collisions with fast moving fluid molecules. In the context of Brownian motion, a useful theory about predicting natural random phenomena occurred, especially after Albert Einstein's paper "Uber die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen, *Annalen der Physik und Chemie*"[31]. In fact, a similar model was proposed by a young French mathematician named "Louis Bachelier" 5 years before Einstein's publication. Bachelier's model was established in the context of his doctoral dissertation with the name "Theory of speculation"[32] under the supervision of Henri Poincare and was aiming to predict price changes in the stock market. Although it was not so popular by the time, it was later discovered and republished by the American mathematician Leonard Jimmie Savage along with the economist Paul Samuelson who developed Bachelier's ideas even further and published two monumental papers in the history of finance[33][34], which led to the mathematical foundation of the efficient-market hypothesis[35] claiming that price changes will essentially be random in well-informed and competitive markets as the variables related to them exhibit cyclic and serial dependencies, leading to unforecastable changes which should be assumed to behave randomly.

THÉORIE
DE
LA SPÉCULATION,

PAR M. L. BACHELIER.

INTRODUCTION.

Les influences qui déterminent les mouvements de la Bourse sont innombrables, des événements passés, actuels ou même escomptables, ne présentant souvent aucun rapport apparent avec ses variations, se répercutent sur son cours.

A côté des causes en quelque sorte naturelles des variations, interviennent aussi des causes factices : la Bourse agit sur elle-même et le mouvement actuel est fonction, non seulement des mouvements antérieurs, mais aussi de la position de place.

La détermination de ces mouvements se subordonne à un nombre infini de facteurs : il est dès lors impossible d'en espérer la prévision mathématique. Les opinions contradictoires relatives à ces variations se partagent si bien qu'au même instant les acheteurs croient à la hausse et les vendeurs à la baisse.

Le Calcul des probabilités ne pourra sans doute jamais s'appliquer aux mouvements de la cote et la dynamique de la Bourse ne sera jamais une science exacte.

Mais il est possible d'étudier mathématiquement l'état statique du marché à un instant donné, c'est-à-dire d'établir la loi de probabilité des variations de cours qu'admet à cet instant le marché. Si le marché, en effet, ne prévoit pas les mouvements, il les considère comme étant

A#1.3: "*Théorie de la spéculation*" by the French mathematician, Paul Bachelier[32].