Long Short Time Memory in the forecast of financial indices in the Brazilian market (Ibovespa)

Marco Antonio Zavaleta Sanchez

Dissertation presented
to
the Institute of Mathematics and Statistics
of
the University of São Paulo
to
obtain the title
of
Master of Science

Program: Statistics

Advisor: Airlane Pereira Alencar

December - 2022

Long Short Time Memory in the forecast of financial indices in the Brazilian market (Ibovespa)

This is the original version of the dissertation prepared
by candidate Marco Antonio Zavaleta Sanchez,
as submitted to the Judging Commission

Judging Commission:

- Marcelo Lauretto - Escola de Artes, Ciências e Humanidades da Universidade de São Paulo - EACH-USP
- Thelma Sáfadi - Unversidade Federal de Lavras - UFLA

Abstract

The present investigation seeks to evaluate different models of recurrent neural networks such as Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BLSTM) and Gated Recurrent Units (GRU) in comparison with the ARIMA model, whose purpose is to find which of these models is capable of making a better forecast for the closing price of 5 steps forward in the stock index of the Sao Paulo Stock Index (IBOVESPA). The optimization of the parameters makes it possible to reduce the cost function (minimum square error). Based on 8 configurations with more than 720 simulations, we discovered that the ADAMAX optimizer has performed better compared to the NADAM and ADAM optimizers, presenting a lower cost function. In the simulations of the different configurations, the average and the standard deviation of different models have been considered. The GRU model with the ADAMAX optimizer was more efficient in more than 90% of the results obtained. The final configuration was the GRU model with a batch size equal to 5, with 250 epochs, a learning ratio equal to 0.001 and with 30 neurons. This configuration presented a lower mean square error and therefore better forecasts. Therefore, the LSTM and BLSTM models did not present a lower cost function compared to the GRU model. Also, the ARIMA model did not have an optimal result compared to recurrent neural network models..

Keywords. LSTM, Neural Network, ARIMA model, Ibovespa.

Resumo

A presente investigação busca avaliar diferentes modelos de redes neurais recorrentes como Long Short-Term Memory (LSTM), Bidirecional Long Short-Term Memory (BLSTM) e Gated Recurrent Units (GRU) em comparação com o modelo ARIMA, cuja finalidade é determinar qual desses modelos é capaz de fazer uma melhor previsão no preço de fechamento de 5 passos à frente no índice de ações da Bolsa de Valores de São Paulo (IBOVESPA). A otimização dos parâmetros permite reduzir a função custo, por isso, foram estudadas 8 configurações com mais de 720 simulações, descobrindo que o otimizador ADAMAX tem funcionado melhor em relação aos demais otimizadores, apresentando uma função custo menor (erro quadrado médio). Nas simulações das diferentes configurações, foram considerados a média e o desvio padrão nos diferentes modelos. O modelo GRU com o otimizador ADAMAX foi mais eficiente em mais de 90% dos resultados obtidos. A configuração final foi o modelo GRU com tamanho de lote igual a 5, com 250 épocas, taxa de aprendizado igual a 0,001 e com 30 neurônios. Essa configuração apresentou um erro quadrático médio menor e, portanto, melhores previsões. Portanto, os modelos LSTM e BLSTM não apresentaram menor custo função em comparação com o modelo GRU. Além disso, o modelo ARIMA não teve um resultado ótimo em comparação com modelos de redes neurais recorrentes.

Keywords. LSTM, Rede neural, modelo ARIMA, ibovespa.

Acknowledgements

I thank God for allowing me to move forward
to achieve my goals...

I dedicate this work to my dear son
Alexander, for your understanding
and affection during my absence...

Abreviation

ANN	Artificial Neural Networks.
IA	Artificial intelligence.
RNN	Recurrent Neural Networks.
LSTM	Long Short-Term Memory.
ML	Machine Learning.
BLSTM	Bidirectional Long Short-Term Memory..
BPTT	Backpropagation through time.
RTRL	Recurring real time learning.
MLP	Multilayer Perceptron.
MSE	Mean Square Error.
CEC	Constant Error Carousel.
ARIMA	Autoregressive Integrated Moving Average.
GRU	Gated Recurrent Unit.
LR	Learning rate.

List of Symbols

η	Learning rate of the network.
τ	Time unit.
u	Output of a unit.
$\mathrm{Pre}(u)$	Predecessors of u.
$\mathrm{Suc}(u)$	Successors of u.
y_u	Activation of u.
u, v, l, k	Units of the network $\in N$.
$W_{[u,v]}$	The weight that connects the unit v to the unit u.
$X_{[u,v]}$	The input of a unit u coming from a unit v.
z_u	The weighted input of the unit u.
b_u	The bias of the unit u.
s_u	The state of the unit u.
f_u	The squashing function of the unit u.
e_u	The error of the unit u.
ϑ_u	The error signal of the unit u.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Stock Exchange is a market that facilitates the trading of stocks, bonds and securities. The operation of stock markets can reflect positively or negatively on the economy of a country. The official Brazilian stock exchange is IBOVESPA, which is negotiated by B3, which is one of the largest financial market infrastructure companies in the world, whose objective is to provide a safe and efficient environment for people and organizations (Castro et al. 2019).

Trading on the Ibovespa index is done electronically through indexed products such as ETF - Equity, Ibovespa Futures, Ibovespa Mini Futures Option on Ibovespa, etc. In addition, Ibovespa is the largest stock trading center in Latin America and at the end of 2021, the B3 registered an increase of 1.5 million individual investors in the capital market, a growth of 56% compared to the previous year (Alves 2019).

As we mentioned, there are several main alternatives to invest in the Stock Market, one of the purposes being to value the assets of the financial market in order to determine their future price. In other words, knowing or knowing how an asset will behave in the future. For this, it is important to accept the premise

that the markets have trends and it is necessary to foresee this trend under certain models that allow reducing risk.

To reduce the risk it is necessary to do a technical analysis focusing on the study of market movements to determine the direction in which prices will move in the market. Technical analysis plays a very important role in making economic or other forecasts, where it is adaptable and applicable to any operating environment and time frame. In this analysis, it is possible to evaluate the markets in active and inactive periods, considering the absence or presence of trends. This analysis is carried out to have an overview of all markets, since it is important to know the price in the market since it gives valuable indications regarding the future direction of another market (Murphy 2000).

From the appearance of autoregressive models to the appearance of computational models, all of them make increasingly precise estimates to reduce risk in decision making. Autoregressive models have been studied for many years and these have a marked limitation since they are only applied in stationary series, later the Generalized AutoRegressive Conditional Heteroscedasticity models (GARCH) appeared. These models consider recent and historical observations, in the same way this variance varies depending on the observations, that is, its future variance depends on the historical variance with some variants that are capable of predicting volatility in the short and medium term, with less error than the ARIMA models. However, in the early 1970s, recurrent neural networks suitable for processing sequential or time-series data were developed.

LSTM networks are a special type of RRN and were developed by Hochreiter Schmidhuber in 1997. These types of networks are designed to handle long-term

dependency that traditional RNNs do not. This type of network produces quite satisfactory results, however, over the years new models have been appearing with more satisfactory results such as the GRU models among others.

Chapter 2

Artificial neural networks

2.1 Introduction

Artificial neural networks (ANN), or commonly called neural networks, are successfully applied to various fields of human knowledge. Since the publication by McCulloch and Pitts 1943, "A Logical Calculus of Ideas Immanent in Nervous Activity", about the mathematical formalism of neuronal events and the relationships between them, several theoretical studies have been developed about the learning mechanism of biological neurons, which have served as inspiration for the proposal of a computational model through the creation of artificial neurons and artificial neural networks. Fig.2.1

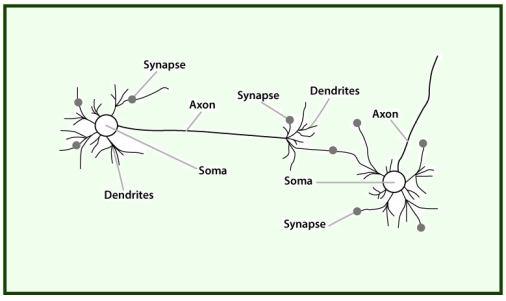

Figure 2.1: Biological neurons. Figure of Daniel S. Yeung and Ng 2009

In 1958 Rosenblatt 1958 developed a hypothetical nervous system called "Perceptron". This system was designed with some fundamental properties of intelligent systems without considering certain special or unknown conditions that occur in some biological organisms.

The most basic type of artificial neuron is called a perceptron. This neural network contains a single input layer and an output node. The perceptron expresses the working of a neural network in mathematical terms. In turn, it follows a learning rule where it initially receives an input vector of real values, whose values are weighted by a multiplier. In a pre-training step, the perceptron learns these weights based on the training data. It then adds all the weighted input values and fires a result value. This output value is always boolean, and it is considered activated if the perceptron output is '1', deactivated if its value is '-1', considering that the threshold value is, in most cases, '0'. Perceptrons consist of several external input links, a threshold, and a single external output link. Likewise, the perceptrons have an internal input b, called bias, which represents the first weight of the neuron allowing the activation threshold of the perceptron to be learned. The perceptron in its most basic form is shown in Figure 2.2 (Staudemeyer and Morris 2019).

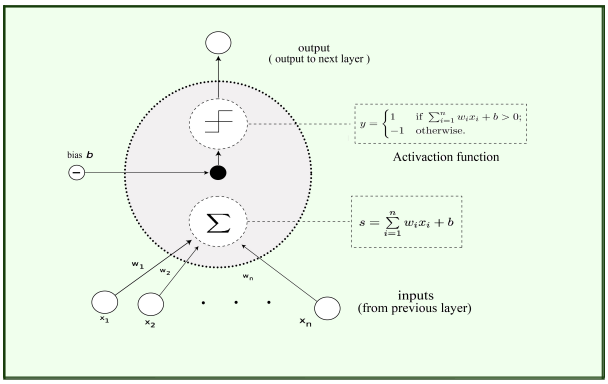

Figure 2.2: Perceptron. Figure of Staudemeyer and Morris 2019.

The analogy between biological neurons and artificial neurons is shown in Table 2.1. Artificial neural networks are fairly simple models to represent brain models at the cellular level, though they help to develop information processing models (Daniel S. Yeung and Ng 2009).

Biological Neurons	Artificial Neurons
Sum	Sum + Activation Function
Dendrite	Input
Axon	Output
Synapse	Weight

Table 2.1: Analogy between biological and artificial neurons.Table extracted from Daniel S. Yeung and Ng 2009.

Artificial Neural Networks, as mentioned initially, will generate a structure similar to biological neural networks. This analogy has the purpose of obtaining a similar functionality. This artificial neural network considers that the nodes are connected by means of synapses (weights). (Haykin 2001) These synaptic connections are directional, therefore information can only travel in one direction. Likewise, synaptic connections present a "Shallow network" that has an input layer, an output layer and at least one hidden layer without recurrent connection. Similarly, it should be noted that as the number of layers increases, the complexity of the network also increases. Therefore, when a greater number of layers or recurring connections are connected, the depth of the network often increases and this allows providing multiple levels of data representation and resource extraction. These networks are usually made up of simple nonlinear units, with upper layers providing a more abstract representation of the data and removing particular unwanted variability. (Salehinejad et al. 2017).

It is important to consider the architecture of the artificial neural network.

The architecture is thus called the topology that is the structure or pattern of connection in a neural network. This topology consists of the organization and arrangement of neurons in the network, forming layers or groups of neurons. An artificial neural network comprises three types of layers: input layers, output layers, and hidden layers. Therefore, taking into consideration certain concepts, different architectures of neural networks can be established. When it is composed of a single layer, it is called monolayer networks, if it is organized in several layers, it is called multilayer networks. In addition, if we consider the data flow, we can distinguish unidirectional networks (feedforward) and recurrent networks (feedback) (Larranaga, Inza, and Moujahid 1997). Likewise, Sazli 2006, points out that there are two main categories of network architectures according to the type of connections between neurons, "feed-forward neural networks (FFNN)" and "recurrent neural networks (RNN)". Before presenting the recurrent neural networks, we present first the backpropagation training method.

In a neural network, the architecture is represented by the connection weight matrix $W = [w_{ij}]$, where w_{ij} indicates the weight of node i connected to node j. When $w_{ij} = 0$, there is no connection between nodes i and j. However, by defining some weights w_{ij} as zero, different network types are described than those presented in the previous paragraph (Du and Swamy 2013).

2.2 Feedforward Neural Networks (FFNN)

The feedforward neural networks (FFNN), also called multilayer perceptrons (MLP) consist of a group of neurons properly structured in one or several layers. In addition, a neuron in a particular layer is connected to all or a subset of neurons in the subsequent layer. The neurons in the input layer do not perform any processing

on the received data, but simply pass it on to the next layer. When input data is placed in the input layer of an FFNN, it is passed to connected neurons that process the inputs and generate the outputs. These networks allow only one directional signal flow. (Iba and Noman 2020).

Feedforward neural networks do not have free loops and are fully connected. This means that each neuron contribute as an input to each neuron in the following layer. (Staudemeyer and Morris 2019).

2.3 Backpropagation Neural Networks (BPNN)

Backpropagation refers to a set of artificial neural networks (ANN), whose architecture is composed of several interconnected layers (Buscema 1998), with bidirectional connections. The most common learning technique for neural networks is the error backpropagation algorithm, which relies on the gradient descent method to learn weights in multilayer networks. This algorithm works in small iterative steps, starting from the output layer back to the input layer. An important condition is that the activation function of the neuron is differentiable.(Staudemeyer and Morris 2019)

In feed-forward neural network, sets of neurons are ordered in layers, where each neuron is calculated as a weighted sum of its inputs. This means that each neuron contribute as an input to each neuron of the next layer, and that none of the weights give an input to a neuron in a previous layer. The notation used is from Staudemeyer and Morris 2019, who has unified the notation from the primary concepts of neural networks to LSTM.

To model the external input received by the neural network, the external input vector is the vector $x = (x_1, x_2, ...x_n)$. For each component of the external input vector, we find a corresponding input unit that models it, so the output of the i^{th} input unit should be equal to the i^{th} component of the input to the network. For a non-input unit $u \in U$, the output of u (also called the activation of u), written as y_u, is a single value and is defined using the sigmoid activation function given by the equation (2.1):

$$y_u = \frac{1}{1 + e^{-s_u}} \tag{2.1}$$

where s_u is the state of the unit u and it is defined by the equation (2.2):

$$s_u = z_u + b_u \tag{2.2}$$

where b_u is called the bias of u, and z_u is the weighted input of u, defined by the equation (2.3):

$$z_u = \sum_v W_{[v,u]} y_v \quad with \quad v \in Pre(u) \tag{2.3}$$

where $y_v = X_{[u,v]}$ is the information that v passes as input to u, and $Pre(u)$ is the set of units v that precede u, containing all input units and hidden units that provide their outputs y_v.

Starting at the input layer, the inputs forward and backward propagate through the network until they reach the output units at the output layer. So, the output units produce an observable output (the output of the network) y. More precisely, for the output units $o \in O$, its output y_o corresponds to the o^{th} component of

y. Then, the backpropagation learning algorithm propagates the error backwards, and the weights and biases are updated in such a way to reduce the error in the training sample. Starting from the output layer, the algorithm compares the output of the network y_o with the output of the desired target value d_o. Then the error e_o is calculated for each output neuron using some function that minimizes the error. The error for each output neuron e_o is calculated as $e_o = (d_o - y_o)$ and the overall network error is given by the equation (2.4):

$$E = \frac{1}{2} \sum_{o \in O} e_o^2. \tag{2.4}$$

The update of the weights is given by the equation (2.5):

$$\Delta W_{[}u, v] = -\eta \frac{\partial E}{\partial W_{[}u, v]}, \tag{2.5}$$

where η is the learning rate. Then we apply an artifice to calculate the update of the weight by deriving the error with respect to the activation, and the activation with respect to the state and the state with respect to the weights is

$$\Delta W_{[}u, v] = -\eta \frac{\partial E}{\partial y_u} \cdot \frac{\partial y_u}{\partial s_u} \cdot \frac{\partial s_u}{\partial W_{[}u, v]}.$$

For the output unit o, the error signal is defined by the equation (2.6):

$$\vartheta_o = \frac{\partial E}{\partial y_o} \frac{\partial y_o}{\partial s_o}. \tag{2.6}$$

Therefore, the error of the ouputs units is represented by the equation (2.7):

$$\vartheta_o = (d_o - y_o) y_o (1 - y_o). \tag{2.7}$$

We can update the weight between the hidden unit h and the output unit o by

the equation (2.8):

$$\Delta W_[u, v] = \eta \vartheta_o y_h. \tag{2.8}$$

When we are in the hidden unit h, if we consider to what extent the production error of a defective output contributed, we can backpropagate the error of the output units to which h sends signals, with greater precision for an input unit i. Therefore, we see that we can update the weight between the input unit i and the hidden unit h by the equation (2.9):

$$\Delta W_[i, h] = -\eta \frac{\partial E}{\partial W_[i, h]}. \tag{2.9}$$

This expression can be represented by the equation (2.10):

$$\Delta W_[i, h] = \eta \sum_o (\vartheta_o W_[h, o]) y_h (1 - y_h) y_i. \tag{2.10}$$

Therefore, the standardized expression for the change in each weight is the equation (2.11):

$$\Delta W_[v, u] = \eta \vartheta_o y_v \tag{2.11}$$

2.4 Recurrent neural networks

Recurrent Neural Networks (RNN) are a type of neural network architecture that is primarily used to detect patterns in a sequence of data.(Schmidt 2019).The most basic way of a simple neural network is one that has three layers, one input layer, a recurrent hidden layer and an output layer, which is shown in Figure 2.3 (Salehinejad et al. 2017).

In Figure 2.3, the input layer has N input units. The inputs to this layer

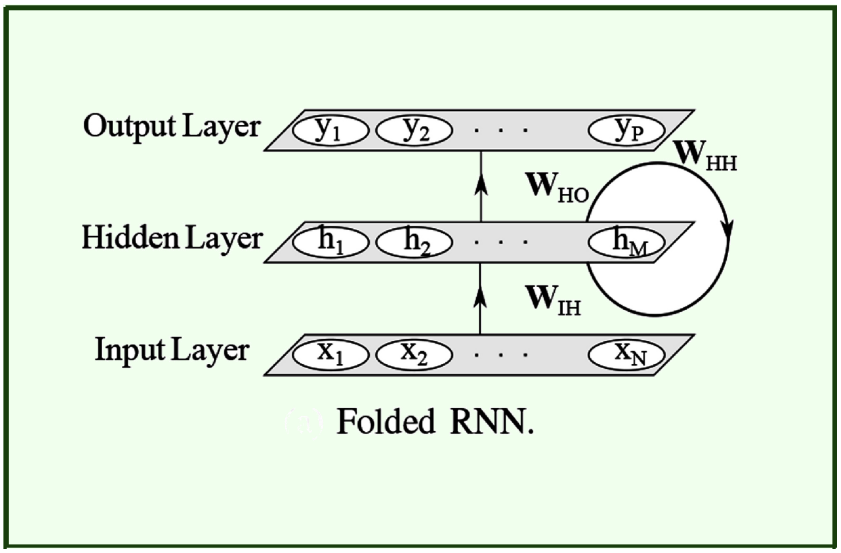

Figure 2.3: Folded RNN.Figure of Salehinejad et al. 2017

is a sequence of vectors through time t such as $..., \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, ...$, where $\mathbf{x}_t = (x_1, x_2, ..., x_N)$. These RNN units are fully connected to the hidden units in the hidden layer, where the connections are defined with a weight matrix $\mathbf{W}_{IH}$. The hidden layer has M hidden units $\mathbf{h}_t = (h_1, h_2, ..., h_M)$. These recurring connections shown in Figure 2.4 present their structure unfolded through time T, where each arrow shows the connection of the units between the layers. The hidden layer defines the state space or "memory" of the system. Finally the output layer has P units, where the output layer with weighted connections $\mathbf{W}_{HO}$ are connected to the hidden units (Salehinejad et al. 2017).

2.5 Training of recurrent neural networks

There are several approaches to train RNNs in order to optimize an algorithm that accelerates convergence and decrease complexity in training algorithms (Sutskever 2013).

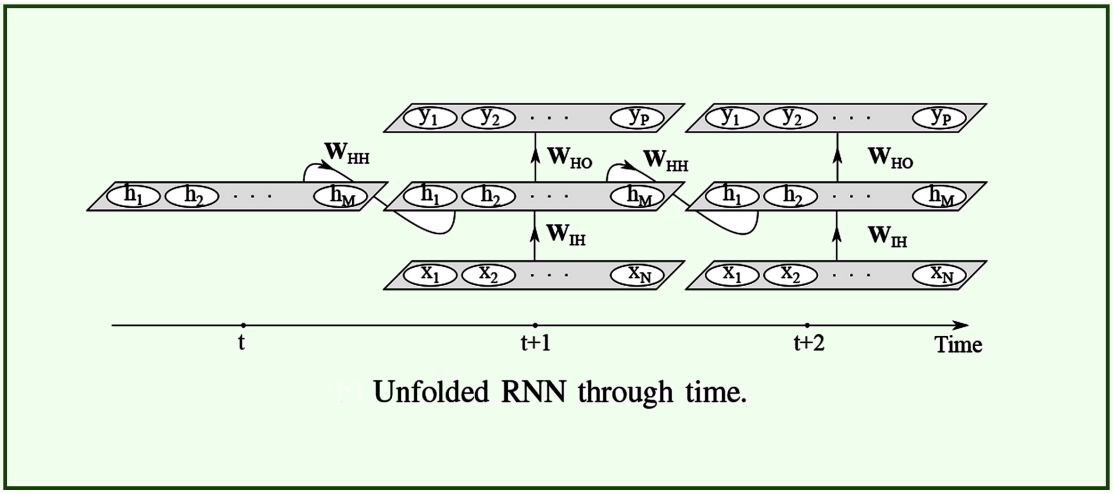

Figure 2.4: Unfolded RNN through time. Figure of Salehinejad et al. 2017

The most common methods to train recurrent neural networks are Backpropagation Through Time (BPTT) and Real-Time Recurrent Learning (RTRL) (Staudemeyer and Morris 2019).

2.5.1 Backpropagation through time (BPTT)

In procedures where learning converts a network that evolves through time into a network whose activation flows through a series of layers, translating time into space is called backpropagation through time or BPTT (Pearlmutter 1995).

The BPTT algorithm considers the fact that, during a finite period of time, there is an FFNN with identical behavior for each RNN. To obtain this FFNN, the RNN needs to be displayed in time. The unfolded network can be trained using the backpropagation algorithm where at the end of a training sequence, the network is deployed over time and the error is calculated for the output units with existing target values using some error measure or cost function. The most used error measures are the mean square error (MSE), the mean absolute error (MAE)

and the crossed entropy. This measure of the error propagates backward through the network and the weight is updated for all calculated time steps. Likewise, the error signal is then calculated for one unit for all time steps in a single pass, using the following iterative backpropagation algorithm (Staudemeyer and Morris 2019).

Subsequently, it is considered discrete time steps $1, 2, 3...$ which is indexed by the variable τ. The network starts at time t' and run until one last time t. This period of time between t' and t is called an **epoch**.

In equation (2.3) we have the weights of the input of u and applying a function that is normally not linear we can define the following.. Let U be the set of units that are not input, and let f_u be the differentiable nonlinear squashing function of the unit $u \in U$, the output $y_u(\tau)$ of u at the unit time τ is given by the equation (2.12):

$$y_u(\tau) = f_u(z_u(\tau)), \tag{2.12}$$

with weighted input in the equation (2.13):

$$z_u(\tau + 1) = \sum_v W_{[u,v]} y_v(\tau) + \sum_i W_{[u,i]} y_i(\tau + 1). \tag{2.13}$$

In equation (2.13) the inputs for u at time $\tau + 1$ are of two types: the environmental input that arrives at time $\tau + 1$ through the input units, and the recurring output of all the non-units of entry into the network produced at that time τ. The cost function is the summed error $E_{total}(t', t)$ for the epoch $t', t'+1, ..., t$ that we want to minimize using the learning algorithm and is defined by the equation (2.14):

$$E_{total}(t', t) = \sum_{\tau=t'}^{t} E(\tau), \tag{2.14}$$

where $E(\tau)$ is given in the equation (2.15):

$$E(\tau) = \frac{1}{2} \sum_{u \in U} (e_u(\tau))^2 \tag{2.15}$$

and the error $e_u(\tau)$ of the non-input unit u at time τ is defined by the equation (2.16):

$$e_u(\tau) = \begin{cases} d_u(\tau) - y_u(\tau) & \text{, if} \quad u \in T(\tau) \\ 0 & \text{, if} \quad \text{otherwise.} \end{cases} \tag{2.16}$$

.

In a similar way to equation (2.6) to adjust the weights we can define the signal error $\vartheta_u(\tau)$ of a non-input unit u at a time τ as follows in equation (2.17):

$$\vartheta_u(\tau) = \frac{\partial E(\tau)}{\partial z_u(\tau)} \tag{2.17}$$

after unrolling $\vartheta_u(\tau)$ in time, we obtain the following equality shown in equation (2.18):

$$\vartheta_u(\tau) = \begin{cases} f'_u(z_u(\tau)) e_u(\tau) & \text{, if} \quad \tau = t \\ f'_u(z_u(\tau)) (\sum_{k \in U} W_{[k,u]} \vartheta_k(\tau + 1)) & \text{, if} \quad t' \leq \tau < t. \end{cases} \tag{2.18}$$

Then, the calculation of the update of the weights $\Delta W_{[u,v]}$ of a recurring network is done by adding the corresponding weight updates for all time steps shown in equation (2.19):

$$\Delta W_{[u,v]} = -\eta \frac{\partial E_{total}(t', t)}{\partial W_{[u,v]}}, \tag{2.19}$$

where

$$\frac{\partial E_{total}(t', t)}{\partial W_{[u,v]}} = \sum_{\tau=t'}^{t} \vartheta_u(\tau) \frac{\partial z(\tau)}{\partial W_{[u,v]}},$$

$$\frac{\partial E_{total}(t', t)}{\partial W_{[u,v]}} = \sum_{\tau=t'}^{t} \vartheta_u(\tau) X_{[u,v]}(\tau).$$

2.5.2 Recurring real-time learning algorithm (RTRL)

In the RTRL algorithm, it considers the assumption that the weights do not remain fixed along the trajectory, it allows the training in real time of behaviors of indefinite duration in order to make weight changes while the network is in operation (Williams and Zipser 1989).

The RTRL algorithm does not require error propagation. This algorithm has a significant computational cost per update cycle considering that the information stored is not local; that is, we consider an additional idea called sensitivity of the output of a unit of the network $k \in U$ and the time steps $t' \leq \tau \leq t$. (Staudemeyer and Morris 2019)

In RTRL we assume the existence of a label $d_k(\tau)$ at every time τ for every non-input unit k, whose objective is to minimize the general error of the network, which occurs in the time step τ by the equation (2.20):

$$E(\tau) = \frac{1}{2} \sum_{k \in U} (d_k(\tau) - y_k(\tau))^2. \tag{2.20}$$

Therefore, in equation (2.14) the gradient of the total error is also the sum of the gradient for all previous time steps and the current time step which is shown in equation (2.21):

$$\nabla_W E_{total}(t', t+1) = \nabla_W E_{total}(t', t) + \nabla_W E_{total}(t+1). \tag{2.21}$$

In the network time series, it is necessary to accumulate the gradient values at each time step to follow the change of weights $\Delta W_{[u,v]}(\tau)$ where the overall weight change is given by the equation (2.22):

$$\Delta W_{[u,v]} = \sum_{\tau=t'+1}^{t} \Delta W_{[u,v]}(\tau). \tag{2.22}$$

For the change of weights it is necessary to calculate the expression (2.23):

$$\Delta W_{[u,v]}(\tau) = -\eta \frac{\partial E(\tau)}{\partial W_{[u,v]}}. \tag{2.23}$$

For each time step t and after expanding through the gradient descent and applying equation (2.14), we find that the overall change in weights is given by the equation (2.24):

$$\Delta W_{[u,v]}(\tau) = -\eta \sum_{k=U} (d_k(\tau) - y_k(\tau)) \left(\frac{\partial y_k(\tau)}{\partial W_{[u,v]}} \right). \tag{2.24}$$

In RTRL, the gradient information is forward-propagated. Using Equations (2.12) and (2.13), the output $y_k(t+1)$ at time step $t+1$ is given by the equation (2.25):

$$y_k(t+1) = f_k(z_k(t+1)), \tag{2.25}$$

with the weighted input is given by the equation (2.26):

$$z_k(t+1) = \sum_{v \in U} W_{[k,v]} y_v(t) + \sum_{i \in I} W_{[k,i]} y_i(t+1). \tag{2.26}$$

2.6 Solving the Vanishing Error Problem

In the theoretical analysis of the flow of errors with methods of recurrent learning based on gradients, he had to overcome difficulties related to problems of long delays by finding the problem of the disappearing gradient. Several models were presented where the Long Short Term Memory(LSTM) model performed well on long time delay problems involving 1000 step time delays (S. Hochreiter 1998).

As mentioned, the problem exists over many time steps, where the error usually explodes or vanishing. Increased error signals lead directly to oscillating weights, whereas with a vanishing error, learning takes an unacceptable amount of time or does not work at all (Staudemeyer and Morris 2019).

The calculation of the gradients using the back-propagation algorithm for the analysis of the fading error when we update the weights after the network has been trained from time t' to time t is given by the equation (2.27):

$$\Delta W_{[u,v]} = -\eta \frac{\partial E_{total}(t', t)}{\partial W_{[u,v]}}, \tag{2.27}$$

with

$$\frac{\partial E_{total}(t', t)}{\partial W_{[u,v]}} = \sum_{\tau=t'}^{t} \vartheta_v(\tau) X_{[u,v]}(\tau). \tag{2.28}$$

At an arbitrary time $\tau < t$, where the back-propagated error signal of the unit u is

$$\vartheta_v(\tau) = f'_k(z_u(\tau)) \left(\sum_{u \in U} W_{vu} \vartheta_v(\tau + 1) \right). \tag{2.29}$$

In consequence the Error flow scaling factor backpropagating is an error occurring at an unit u at time step t to an unit v for $t - t'$ time steps, and it is given by the equation (2.30):

$$\frac{\partial \vartheta_v(t')}{\partial \vartheta_o(t)} = \begin{cases} f'_k(z_u(\tau)) W_{[o,v]} & , \text{if} \quad t - t' = 1 \\ f'_k(z_u(\tau)) \left(\sum_{u \in U} \frac{\partial \vartheta_v(t'+1)}{\partial \vartheta_o(t)} W_{[u,v]} \right) & , \text{if} \quad t - t' > 1. \end{cases} \tag{2.30}$$

In this scheme of analysis by (J. Hochreiter 1991), it is assumed that you have a fully connected network whose non-input unit indices range from 1 to n (S. Hochreiter and Schmidhuber 1997).

2.7 Bidirectional Recurrent Neural Networks

Recurrent bidirectional neural networks (or bidirectional RNNs) were invented to overcome the limitations of RNNs (Schuster and Paliwal 1997).

In a bidirectional recurrent neural network (BRNN) can be trained without limitation using all the input information available in the past and future of a specific time frame.In BRNN, the main idea is to divide the state neurons of a regular RNN into a part that is responsible for the positive time direction (forward states) and a part for the negative time direction (backward states). The outputs of the forward states are not connected to the inputs of the backward states and vice versa. In BRNN it can be trained mainly with the same algorithms as a unidirectional RNN because there are no interaction between the two types of

state neurons, and therefore it can be deployed in a general feedback network. However, if the BPTT is used, the backward and forward step procedure is a bit more complicated since updating the state and output neurons can no longer be done one at a time (Schuster and Paliwal 1997).

Chapter 3

ARIMA Models

In the 1970s, George Box and Gwilym Jenkins, developed the well known Box-Jenkins methodology (George E. P. Box and Liung 2016; Morettin and Toloi 2006) to fit the ARIMA (Autoregressive Integrated Moving Average) time series models. A time series is a sequence of observations taken sequentially in time. A stochastic process is a sequence of random variables, that can be indexed in time. A stochastic process is known as weakly stationary, or stationary of second order, if it presents constant mean, a finite variance, which is also constant for all time points, and its covariance function may depend only on the lag. This latter property means that the covariance of Z_t and Z_{t-k}, depends only on k.

For a stationary process $Z_t, t = 1, 2, \ldots$, the autoregressive and moving average model (ARMA(p,q)) is defined as:

$$Z_t = \phi_0 + \sum_{i=1}^{p} \phi_i Z_{t-i} + \sum_{j=0}^{q} \theta_j a_{t-j} + a_t, \tag{3.1}$$

where a_t are non-correlated, zero mean, homoscedastic random variables, known as white noise, ϕ_i is an autoregressive parameter, and θ_j is a moving average parameter. In general, this model is fitted by the maximum likelihood method,

assuming that the errors a_t are Gaussian. The methods to identify the autoregressive and moving average orders (p and q) are well described in Morettin and Toloi 2006, along all the diagnostic analysis to check all the model assumptions and the methods to calculate the forecasts and their corresponding confidence intervals. For this model, the expected value $E(Z_t) = \frac{\phi_0}{1-\sum_{i=1}^{p}\phi_i}$. From now on, we will consider the centered variable $\widetilde{Z}_t = Z_t - E(Z_t)$.

It is simpler to define the ARMA(p,q) model using the backshift operator B that may be applied to any random variable Y_t, where $BY_t = Y_{t-1}$, $B^kY_t = Y_{t-k}$. Using this notation and sending the autoregressive terms to the left side, the ARMA(p,q) model may be written as

$$\begin{aligned}
\widetilde{Z}_t - \sum_{i=1}^{p}\phi_i\widetilde{Z}_{t-i} &= a_t + \sum_{j=0}^{q}\theta_j a_{t-j} \\
\widetilde{Z}_t - \sum_{i=1}^{p}\phi_i B^i\widetilde{Z}_t &= a_t + \sum_{j=0}^{q}\theta_j B^j a_t, \\
(1 - \sum_{i=1}^{p}\phi_i B^i)\widetilde{Z}_t &= (1 + \sum_{j=0}^{q}\theta_j B^j)a_t,
\end{aligned}$$

Some time series are not stationary and may present some stochastic trend. An extension of the ARMA model is the ARIMA(p,d,q) model, which consists of calculating d differences of the original time series Z_t until the series become stationary. We may calculate one difference ΔZ_t or two differences $\Delta^2 Z_t$ using the polynomial notation as

$$\begin{aligned}
\Delta Z_t &= (1-B)Z_t = Z_t - Z_{t-1} \\
\Delta^2 Z_t &= (1-B)^2 Z_t = (1-B)(Z_t - Z_{t-1}) = Z_t - Z_{t-1} - (Z_{t-1} - Z_{t-2}) = \\
&= Z_t - 2Z_{t-1} + Z_{t-2}.
\end{aligned}$$

In general, it is necessary only one difference to obtain a stationary series, but sometimes it is necessary to calculate the second difference.

After calculating d differences to achieve the stationarity, the ARIMA(p,d,q) model may be defined as

$$\left(1 - \sum_{i=1}^{p} \phi_i B^i\right) (1 - B)^d \widetilde{Z}_t \;\; = \;\; \left(1 + \sum_{j=0}^{q} \theta_j B^j\right) a_t,$$

where again a_t is a white noise sequence with constant variance σ_a^2. This means that after calculating d differences, the differenced process $(1-B)^d Z_t$ is stationary and follows an ARMA(p,q) model. The maximum likelihood method is used to estimate all parameters $\{\phi_0, \phi_1, \ldots, \phi_p, \theta_1, \ldots, \theta_q, \sigma_a^2\}$.

This ARIMA model is considered appropriate only after a complete residual analysis, including:

- a residual plot to verify if the residual mean and variance are constant over time;
- a residual correlogram and a Ljung-Box test (George E. P. Box and Ljung 2016) to check if the residuals are not correlated ; and
- a qq plot and a normality test to evaluate if the residuals follow a Gaussian distribution.

The asymptotic variance of the maximum likelihood estimators may be obtained using the information matrix, based on the hessian matrix of the conditional log-likelihood function (details in Morettin and Toloi 2006).

For the process Z_t, the ARIMA model may be used to calculate forecasts of horizont h, ie for Z_{t+h}. The forecasts may be obtained writing the model for Z_t as function of all lagged Z_{t-k}, a_t and lagged errors a_{t-1}. To calculate the variance of

forecasts, it is necessary to write the model depending only on a_t and the previous errors and the variance depends on the coefficients of these errors. Assuming that the errors follow a Gaussian distribution, it is possible to calculate the forecasts' intervals. Details on obtaining forecasts, their variance and intervals may be found in (Morettin and Toloi 2006).

An natural extension of the ARIMA model is the SARIMA model, that introduces the seasonality including lagged observations and errors. For example, for monthly data and seasonality of 12 months, the effects of Z_{t-12} and a_{t-12} are also included in the seasonal autoregressive polynomial and in the moving average polynomial.

Another important extension includes explanatory variables in the model. It is like a regression model with errors ARMA(p,q), what can be called ARMAX model. Details about this last model may be found for example in (Hyndman and Athanasopoulos 2018) and (Shumway and Stoffer 2017).

Chapter 4

Long Short-Term Memory

The problem of the BPTT or RTRL algorithms when the error signals flow backwards tend to explode or disappear. In the first case the explosion can lead to oscillating weights and in the second case learning to overcome long delays takes a prohibitive amount of time or does not work at all. According to (Hochreiter 1991) the temporal evolution of the backpropagated error depends exponentially on the size of the weights. (S. Hochreiter and Schmidhuber 1997)

One solution to dealing with the disappearing error problem is the gradient-based method called long short-term memory (LSTM). The LSTM can learn to overcome minimal time delays of more than 1000 discrete time steps. This solution method uses constant error carousels (CEC), which enforce a constant error stream within special cells (Staudemeyer and Morris 2019).

4.1 Architecture LSTM

The architecture of LSTM networks consists of the grouping of subnets connected on a recurring basis, called memory blocks. Each memory block contains one or

more self-connected cells and three multiplicative units: the entry gates (admits new elements to memory), exit (allows creating the updated hidden state) and forgetting (permit deleting elements from memory). Multiplicative gates allow LSTM memory cells to store and access information for long periods of time, thus mitigating the vanishing gradient problem (Graves 2012).

4.2 Constant Error Carousel (CEC)

When we want to preserve the error, meaning that it does not explode or vanish in time, it is necessary to create an identity function in the squashing function f_u equating the weights that connect u with itself (Staudemeyer and Morris 2019). The return flow of the local error of u in a single time step follows from equation (2.19) and is given by:

$$\vartheta_v(\tau) = f_k'(z_u(\tau)) \left(\sum_{u \in U} W_{[vu]} \vartheta_v(\tau + 1) \right). \tag{4.1}$$

To ensure a constant error flow through the u unit, we need to have

$$f_u'(z_u(\tau)) W_{[u,u]} = 1. \tag{4.2}$$

Now integrating this expression, we obtain

$$f_u(z_u(\tau)) = \frac{z_u(\tau)}{W_{[u,u]}}. \tag{4.3}$$

From this, it follows that f_u must be linear and that the activation of u must remain constant over time; therefore we have

$$y_u(\tau + 1) = f_u(z_u(\tau + 1)) = f_u(y_u(\tau + 1)) W_{[u,u]} = y_u(\tau). \tag{4.4}$$

4.3 Memory Blocks

In defining the architecture of an LSTM network, the idea of memory block cells was conceptualized. This block contains a memory cell and three multiplicative gate units: the gate of input, output and forget. The purpose of the gates is to prevent the rest of the network from changing the memory cell value in multiple time stages. This is what allows the model to retain information for much longer than in an RNN. In practice, memory blocks can have even more gates where the input to the memory block is multiplied by the activation of the input gate, then the output is multiplied by the exit gate, and the previous cell values are multiplied. through the door of oblivion. It should be noted that the gates control the flow of information into and out of the memory cell(Lewis 2017).Figure. 4.1

4.4 Backpropagation LSTM

4.4.1 The Forward Pass

The Forward pass is referred to the process of calculating the values of the output layers of the input data, that is, it crosses all the neurons from the first to the last layer. For this process, M is defined as the set of memory blocks, m_c the $c-th$ memory cell in the memory block m and $W_{[u;v]}$ is a connection of the LSTM models, each memory block m is associated with an input port i_{nm} and an output port out_m. The internal state of a memory cell m_c in time $\tau+1$ is updated according to its state $s_{m_c}(\tau)$ and according to the weighted input $z_{m_c}(\tau+1)$ multiplied by the activation of the input gate $y_{in_m}(\tau+1)$ then we use the activation of the output gate $z_{out_m}(\tau+1)$ to calculate the cell activation $y_{m_c}(\tau+1)$ (Staudemeyer and Morris 2019).

To calculate the activation y_{in_m} of the imput gate i_{nm} we have the following expression:

$$y_{in_m}(\tau + 1) = f_{in_m}(z_{in_m}(\tau + 1)) \tag{4.5}$$

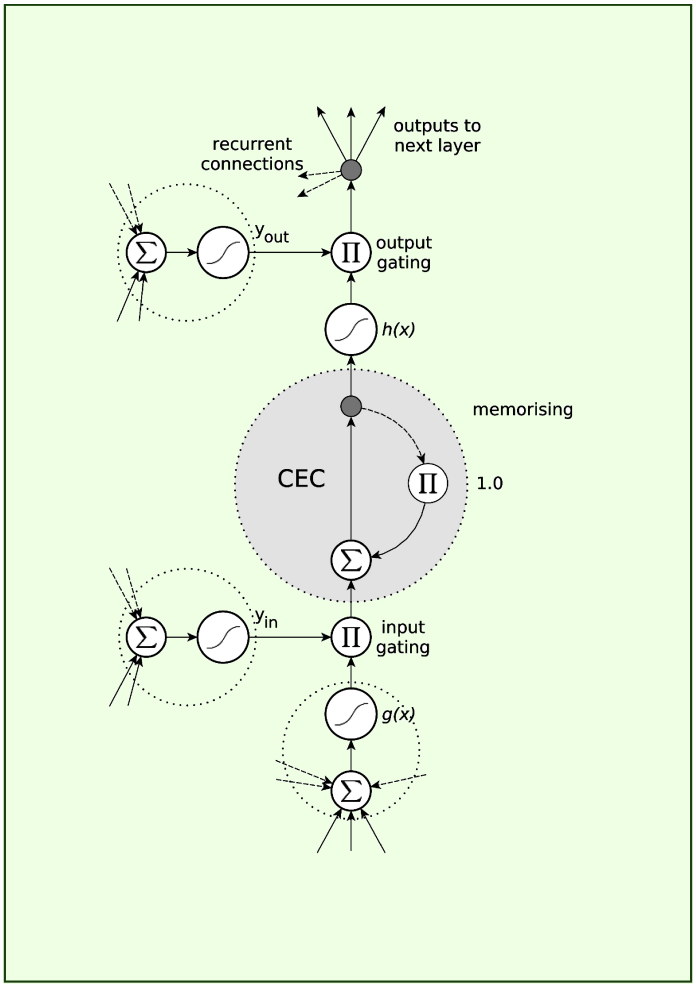

Figure 4.1: A standard LSTM memory block. The block contains (at least) one cell with a recurrent self-connection (CEC) and weight of '1'. The state of the cell is denoted as s_c. The internal cell state is calculated by multiplying the result of the squashed input, g, by the result of the input gate, y_{in}, and then adding the state of the last time step, $s_c(t-1)$ Figure of Staudemeyer and Morris 2019

4.4.2 Forget Gates

Unfortunately, the cell states s_m tend to grow linearly during the progression of a time series presented in a continuous input sequence. The main negative effect is that the entire memory cell loses its memorising capability, and begins to function like an ordinary RNN network neuron. To address this problem, proposed that an adaptive forget gate could be attached to the self-connection. Forget gates can learn to reset the internal state of the memory cell when the stored information is no longer needed (Staudemeyer and Morris 2019).

To this end, we replace the weight '1.0' of the self-connection from the CEC with a multiplicative, forget gate activation y_φ which is computed using a similar method as for the other gates:

$$y_{\varphi_m}(\tau+1) = f_{\varphi_m}(z_{\varphi_m}(\tau+1) + (b_{\varphi_m}), \tag{4.6}$$

where f is the squashing function from $f(c) = \frac{1}{1+e^(-s)}$ with a range $[0,1]$, b_{φ_m} is the bias of the forget gate, and

$$\begin{aligned} z_{\varphi_m}(\tau+1) &= \sum_u W_{[\varphi_m,u]} X_{[u,\varphi_m]}(\tau+1) \quad with \quad u \in Pre(\varphi_m) \\ &= \sum_{v \in U} W_{[\varphi_m,v]} y_v(\tau) + \sum_{i \in I} W_{[\varphi_m,i]} y_i(\tau+1) \end{aligned} \tag{4.7}$$

4.4.3 Backward Pass

LSTM networks incorporate elements from both BPTT and RTRL. Thus, we split units into two types: those units whose weight changes are calculated using a variation of BPTT and those whose weight changes are calculated using a variation of RTRL.(Staudemeyer and Morris 2019).

Using Equations (2.14) and (2.16), the overall network error at time step is

$$E = \frac{1}{2}\sum_{o \in O}(d_o(\tau) - y_o(\tau))^2. \tag{4.8}$$

4.5 Bidirectional LSTM

Bidirectional LSTM (BLSTM) networks have two independent recursive hidden layers, these layers connect to the same input and output layers. This type of networks use past and future information, this being an essential requirement (Fernández, Graves, and Schmidhuber 2008). It must be kept in mind that these networks have strict communication in the entire data sequence in any time step t, and at the same time provide access in relation to the long-range bidirectional aspect (Graves et al. 2007).

When we train bidirectional LSTM neural networks, they have architectural superiority under unidirectional training if it is used to classify phonemes.Bidirectional LSTM removes the one-step truncation originally present in LSTM, and implements a full error gradient computing. (Staudemeyer and Morris 2019).

4.6 Gated Recurrent Unit (GRU)

As mentioned, the LSTM model is composed of the entrance gate, the exit gate and the forgetting gate. This design allows processing the data of a time series. The GRU model is a variant of the LSTM neural network where the structure of the LSTM neurons is optimized and it combines the three activation units of the LSTM into two activation units, which are the update gate and the reset

gate.(Wang et al. 2019). The GRU is a variant of the LSTM and reduces the activation signals to those of the LSTM model (Dey and Salem 2017).

The Gated Recurrent Unit (GRU) architecture for RNN as an alternative to LSTM. GRU has practically been found to outperform LSTM in almost all tasks, with the exception of language modelling with naive initialization. Staudemeyer and Morris 2019).

According to (Staudemeyer and Morris 2019), GRU units do not have a memory cell, although they do have a reset gate and an update gate. Specifically, the H makes up the set of GRU units,if $u \in H$, then we define the activation $y_{res_u}(\tau+1)$ of the reset gate res_u at time $\tau+1$ by

$$y_{res_u}(\tau+1) = f_{res_u}(s_{res_u}(\tau+1)), \tag{4.9}$$

where f_{res_u} is the squashing function of the reset gate (usually a sigmoid function), and $s_{res_u}(\tau+1)$ is the state of the reset gate res_u at time $\tau+1$, which is defined by

$$s_{res_u}(\tau+1) = z_{res_u}(\tau+1) + b_{res_u}, \tag{4.10}$$

with b_{res_u} is the bias of the reset gate, and $z_{res_u}(\tau+1)$ is the weighted input of the reset gate at time $\tau+1$.

Similarly, we define the activation $y_{upd_u}(\tau+1)$ of the update gate updu at time $(\tau+1)$ by the equation (4.11)

$$y_{upd_u}(\tau+1) = f_{upd_u}(s_{upd_u}(\tau+1)), \tag{4.11}$$

where f_{upd_u} is the squashing function of the update gate (again, usually a sigmoid function), and $s_{upd_u}(\tau + 1)$ is the state of the update gate updu at time $\tau + 1$, defined by

$$s_{upd_u}(\tau + 1) = z_{upd_u}(\tau + 1) + b_{upd_u} \tag{4.12}$$

where b_{upd_u} is the bias of the update gate, and $z_{upd_u}(\tau + 1)$ is the weighted input of the update gate at time $\tau + 1$.

In the GRU model the reset and input ports work as normal units in a recurrent network. The most important feature of the GRU model is the way the activation of the GRU units is defined (Staudemeyer and Morris 2019).

Chapter 5

Real data analysis

5.1 Introduction

There is an extensive literature on the analysis of time series of financial data and its application with a technical approach that allows analyzing the behavior of the evolution of the data in a certain period of time, for future forecasts related to an investment. The most used model for time series analysis is the Autoregressive Integrated Moving Average (ARIMA).

In the financial market, it is well known that stock prices behave as a random walk and it is not possible to forecast prices due to the non-arbitrage hypothesis. But in our study, the ARIMA model is adopted to obtain naive forecasts for the prices as a reference to evaluate the performance of all other methods. Likewise, in recent decades other techniques such as machine learning and artificial intelligence have been developed, particularly recurrent neural network models. In this section, a comparative analysis of the ARIMA, LSTM (Unidirectional, Bidirectional) and GRU models is carried out, which were developed in chapter 3 and chapter 4.

From the perspective of deep learning, the computer learns from a program in several steps and deep learning can be safely considered as the study of models that involve a larger number of learned functions or learned concepts than traditional machine learning. (Goodfellow, Bengio, and Courville 2016). In the case of neural networks, there are several important issues associated with neural network setup, preprocessing, and initialization. (Aggarwal 2018).

We are going to compare the results of the ARIMA, LSTM (Unidirectional, Bidirectional) and GRU models considering different configurations of the hyperparameters. For the construction of these configurations, it begins with the design of the network topology and then the multiple hyperparameters that are included in the training which established appropriately. Also, it should be considered that these hyperparameters interact in a complex way (Galiano 2018)

- **Neurons.** From the computational point of view, the neuron is considered as a node. The connections between neurons form a neural network that implicitly represents knowledge, through the weights with which the connections between neurons are modeled. The choice of the number of neurons is taken into account minimizing the number of these to reduce the computational load of training and use of additional neurons. Each additional neuron adds unnecessary load to the CPU (or GPU).

- **Learning rate.** The learning rate allows you to control the size of the updates that are made on the parameters of the model that are being trained. A small value is usually chosen so that it does not cause convergence problems. Likewise, the learning rate does not affect the computational time.

- **Batch size.** It is defined as batch training which allows defining how often the weights are updated within each epoch. It should be noted that batch learning guarantees the convergence of the learning algorithm as long as the

learning rate $\eta < 2$, as long as the mean square error is used. Also, batch training allows a formal analysis of the dynamic properties of the learning process and its convergence.

- **Number of epochs.** It is the number of times the model is exposed to the entire training dataset. This means that each sample in the training dataset updates the parameters in the model internally. Therefore, this hyperparameter defines the number of times the learning algorithm will run on the training data.

- **Optimization Algorithms.** These algorithms involve an optimization process whose purpose is to gradually decrease the learning rate used by the descending gradient, that is, instead of using a fixed learning rate, η, a variable learning rate is used, $\eta(t)$, which decreases as the optimization process based on the descending gradient progresses. Over the years, various techniques for adaptive adjustment of learning rates have been developed. The most common optimizers are presented below.

  - **AdaGrad** (Adaptive Gradient). The AdaGrad algorithm accumulates the sum of the squares of the gradients of the error function, for each parameter of the network throughout the algorithm. This optimizer may have certain desirable properties from a theoretical point of view, however, the accumulation of squared gradients in the denominator may give rise to an early decrease in effective learning rates.
  - **RMSPro.** The Root Mean Square Propagation is a variant of the adaptive learning rate gradient descent method, which is obtained if the gradient is divided by a moving average of its magnitude. The magnitude of the gradient can be very different for different parameters of a neural network. The RMSprop algorithm only changes

the AdaGrad gradient buildup to an exponentially smoothed moving average.

– **Adam** (Adaptive Moment estimation). This optimizer is inspired by the combination of AdaGrad and RMSprop. The method uses the estimation of the first and second moments of the gradient using exponential moving averages and then applies some bias correction. This algorithm preserves two moving averages of the gradients:

$$\begin{aligned} m(t) &= \beta_1 m(t-1) + (1-\beta_1) g_t \\ v(t) &= \beta_2 v(t-1) + (1-\beta_2)(g_t)^2 \end{aligned}$$

where $m(t)$ is an estimate of the first moment of the gradient (its mean) and $v(t)$ is an estimate of the second moment of the gradient (its variance). The parameters that control the moving averages, β_1 and β_2 , have values close to 1, eg $\beta_1 = 0.9$ and $\beta_2 = 0.999$. Adam is usually considered quite robust with respect to the values of its hyperparameters $(\eta, \beta_1 \ and \ \beta_2)$, although sometimes we will have to manually adjust the reference learning rate, η.

– **AdaMax**. The Adaptive Maximum method is a variant of the Adam algorithm based on the infinite norm. Norms with large values tend to be numerically unstable, hence we almost always use the L^1 or L^2 norms. However, the norm L^∞, is also numerically stable. If we use this norm, we can calculate the moving average.

The algorithm is defined by the following set of iterations:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ u_t &= max(\beta_2 m_{t-1}, |g_t|) \\ x_t &= x_{t-1} - \frac{\eta}{1-{\beta_1}^t} \frac{m_t}{u_{t-1}} \end{aligned}$$

– **Nadam** (Nesterov-accelerated adaptive moment estimation). This optimizer uses the concept of the Nesterov moment. This type of moment is based on correcting an error after it has been made and helps to accelerate the convergence of the optimization algorithm based on moments, offering a better rate of convergence in the optimization of convex functions. Nesterov moments perform slightly better than standard moments , although without providing asymptotic improvements in the rate of convergence of the descending gradient.

$$\Delta x = -\eta \frac{1}{\sqrt{\hat{v}} + \epsilon} \left(\beta_1 \hat{m}_t + \frac{(1-\beta_1) g_t}{1-\beta_1^t} \right)$$

Where:

$\hat{m}_t = \frac{m_t}{1-{\beta_1}^t}$ estimate of the first moment of the gradient (mean)

$\hat{v}_t = \frac{m_t}{1-{\beta_2}^t}$ estimation of the second moment of the gradient (variance)

After having established the hyperparameters, it is necessary to decide which is the best model under the pre-established configurations and for this it is necessary to consider one or more metrics to determine the best model. There is a set of metrics that allow evaluating the quality of the model in time series forecasting models. The three most used metrics according to (Botchkarev 2018) are the mean square error (MSE) (or root MSE (RMSE)), mean absolute error (MAE) and mean absolute percentage error (MAPE).

From this group of metrics, the MSE has been considered as it better explains the differences in the model's performance. Also, the errors follow a normal distribution and are not biased (Chai and Draxler 2014).

The MSE that will be presented after the application corresponds to the forecast data set. Of equal importance, when executing the program code, the error of the model is obtained for both the training set and the validation set, they are called training loss and validation loss respectively. Also, we should mention that the training loss is calculated during each epoch, but the validation loss is calculated at the end of each epoch (Brownlee 2017).

It is important to define the loss function which is calculated by means of a cost function and this should be a value as low as possible. This loss function seeks to minimize the error and is calculated to the validation and test training group.

- **Training Loss (loss).-** The loss function on the training set is used to evaluate how well a learning model fits the training data. That is, this function evaluates the model error on this set.
- **Validation Loss (val loss).-**The loss function on the validation set is used to evaluate the performance of a learning model on the validation set. The validation set is a part of the data set reserved for validating model performance.

It is important to indicate that the loss function is used to evaluate and diagnose the optimization of the model and the metric is used to evaluate and choose a certain model.

After presenting the concepts related to hyperparametrics, metrics and loss functions, we evaluate different methods and configurations for the forecast of the main performance indicator of the shares traded on B3 (IBOVESPA). The study of these models is to determine which of them has a better performance when investing in shares in the financial market. The choice of the model will contribute to the projections of the aforementioned index and at the same time minimize or reduce the risk in investments.

To obtain the results of the different models, the code was developed in the Python programming language using the pmdarima, Numpy, Pandas, Seaborn, Matplotlib, Scikit-lear, Keras and Tensorflow libraries.

In most deep learning problems, computational algorithms that work at various levels are used. Deep Learning models have a complex architecture and require a large amount of data to train. For the execution of the simulations and working only with a CPU it is unlikely that certain results will be obtained. Therefore, it is convenient to use a hardware architecture with GPUs because if we work with a complex network, it can take too long to train the networks, so it is necessary to use a state-of-the-art computer. In this investigation we used a personal laptop with relatively modern features, a Ryzen 7 5800H 8 cores and 16 threads with 32 GB of RAM and an NVIDIA RTX 3050Ti video card. This general consumption card optimizes time when performing simulations.

For the adjustment in the learning of the neural network, it is necessary to consider different architectures for the training of the network in search of an optimal structure. (Lara-Benıtez, Carranza-Garcıa, and Riquelme 2021). For the prediction of our target in the LSTM, BLSTM and GRU models, the following

hyperparameters are considered:

- learning rate, which was fixed as lr=0.001 for all configurations;
- number of neurons can be 30 or 50 neurons;
- batch size, that considered as 5, 10, 20 and 60;
- number of epochs, which were 250, 500, 1000, 2000, and 6000.

Considering all combinations of these hyperparameters and optimizer, the number of all configurations were 720, taking into account that 5 replications of each configuration were made and the average of these was presented as the final result of each determined configuration.

5.2 Financial index in the Brazilian market (IBOVESPA)

Futures contracts are versatile and dynamic tools that have been used with some effectiveness by investors seeking to benefit from fluctuations in asset prices. These are basically traded in four main segments: interest rates, currencies, stock indices and commodities. All contracts are standardized and listed in the stock market.

The IBOVESPA is the most important stock market index and tracks the performance of around the 50 most liquid stocks traded on the São Paulo stock market in Brazil, called B3. In short, this index is calculated based on the average performance of the most traded shares on the official Brazilian Stock Exchange. The IBOVESPA series analyzed here was obtained from the yahoo finance site and corresponds to the daily index from January 2010 to December 2019.

Figure 5.1 shows the IBOVESPA stock market index from January 2010 to December 2019. In the period between January 2010 and the beginning of 2016, the Ibovespa index shows a downward trend. In that period the prices were quite irregular and the smallest drop was registered on December 26, 2016 with 37 947 points. As of that date, the index has exhibited a growing trend. However, from February 26, with 87 653 points, the index was in decline until June 18 of the same year, when it had its biggest drop with 69 815 points. Since then, the IBOVESPA index has shown a growing trend. This analysis has been carried out until the end of 2019 since with the appearance of COVID-19 it generated many changes and we can find non-real results. However, we must point out that these models showed quite satisfactory results.

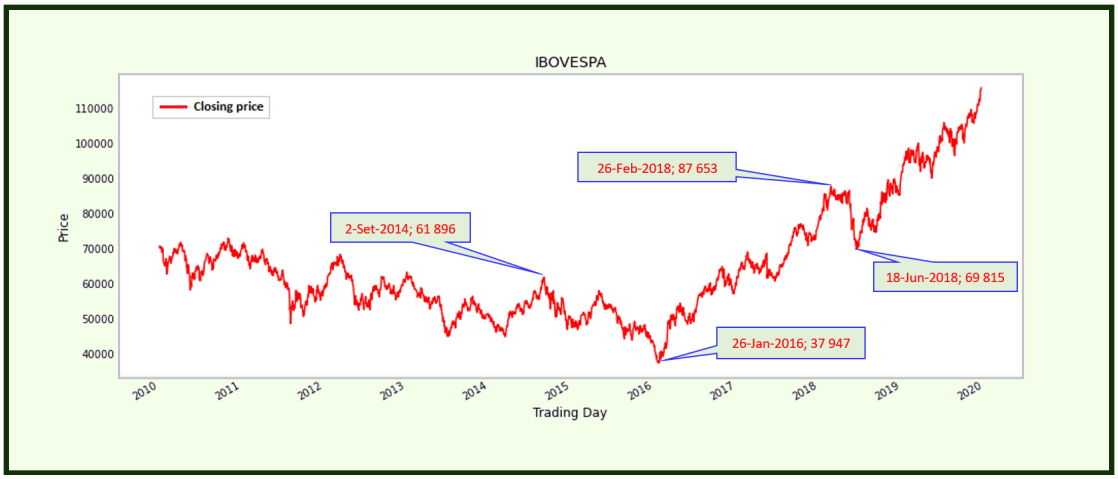

Figure 5.1: Daily IBOVESPA index from 2010 to 2019.

Before starting the simulations under the different configurations, the database must be prepared, taking into account the elimination of the missing data and the partition of the data base. This partition is made into three groups, the first training group that corresponds to 80% of the first observations (1972 observations) from January 4, 2010 to December 20, 2017, this data for validation is used to train the algorithm and the adjustment of the weights. The second, the validation

data, which corresponds to 10% (246), starts on December 21, 2017 and ends on December 20, 2018 and this second group is used to adjust the hyperparameters, evaluate the accuracy of the algorithm with unknown data during training, and check its performance. If the results are not satisfactory we can adjust the hyperparameters of the algorithm and repeat the process again. Finally, the last 10% (248 observations) whose period is between December 21, 2018 and December 23, 2019 and corresponds to the Test group (see Figure 5.2). This last set is similar to the validation set since the algorithm has not used it during training. However, when we strive to improve the algorithm on the validation data, we may unexpectedly introduce a bias that distorts the results in favor of the validation group and does not reflect actual performance, since we use the test alone. However, this procedure provides a more objective measurement, since this data set is used to test the model after the training is finished. Therefore, it provides a better performance of the metric in terms of accuracy and precision, that is, a better performance of the model.

Figure 5.2: Daily IBOVESPA index from 2010 to 2019.

In the application of recurrent neural network models, it is essential to prepare

the data before implementing the time series model (LSTM, BLSTM and GRU). For the correct training of the networks it is necessary to normalize the three groups of data in the interval between 0 and 1, where the minimum value is subtracted and divided by the range of the series, this is achieved using the minmax scale function of the scikit-learn library. Carrying out this procedure guarantees better results in network training under the different configurations in the different models. (Aggarwal 2018).

According to (Bao, Yue, and Rao 2017), there is no rule to select the number of delays and hidden layers. Therefore, to obtain the least mean squared error in this first configuration, we initialize with the same values considered in his investigation where the learning rate, the batch size and the number of epochs are 0.001, 50 and 6000 respectively.

It is important to point out that the comparison of the mean square error of both the ARIMA model and the LSTM, BLSTM and GRU models will be carried out. The Python package Pdarima, allowed to calculate the estimation of the parameters of the ARIMA(5,2,0) model. In the case of the neural network models (LSTM, BLSTM and GRU), the combination of the ADAM, NADAM and ADAMAX optimizers with 30 and 50 neurons has been considered. In the table 5.1, it shows the result of the average of the simulations of the neural network models and in the case of the ARIMA(5,2,0) model, it only shows the results of a single output.

In this first simulation under the mentioned configuration, the best average of the MSE obtained in the GRU model was with the ADAMAX optimizer and 30 neurons, which is $406.1 * 10^{-6}$.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	10154	726	**222**	**730**	4645	**1339**
	sd	800	11275	121	5400	293	1008
BLSTM	mean	79284	**248**	743	749	60320	2963
	sd	176513	151	806	118023	129387	1080
GRU	mean	**1304**	1234	406	666	**1018**	1461
	sd	775	712	496	155	514	719
ARIMA	mean	9186	9186	9186	9186	9186	9186

Table 5.1: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =50, and 6000 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

In the MSE estimation of the GRU model with 50 neurons, there were no vanishing or blow-up problems, however, the MSE values are higher compared to the same model with 30 neurons.

In the LSTM model, the lowest mean square error was with the ADAMAX optimizer and 30 neurons whose value is $221.8 * 10^{-6}$. However, in the first and second simulations with the ADAM optimizer and 30 neurons, the gradients blows-up, that is, became extremely large. Likewise, in the fourth simulation of this configuration, the problem of gradient vanishing was presented. These scenarios generated problems in the calculation of the MSE. Therefore it is not possible to make a comparison, this configuration. In the case of the ADAMAX optimizer with 50 neurons, no blows-up or vanishing problems were present, however, the average MSE presented high values.

Finally, in the BLSTM model with 30 neurons and the NADAM optimizer, there

were no problem in calculating the average of the MSE. However when we use the ADAM optimizer one of the simulations presented the gradient blows-up problem and the ADAMAX optimizer presented high values. Likewise, when 50 neurons were considered in the configuration, it presented various problems with the three optimizers, either due to fading problems, and blows-up or high values in the MSE.

This configuration has presented various problems in the neural network models, therefore it is necessary to establish a new configuration in order to obtain a better performance of the MSE.

Table 5.2 shows the second configuration for learning the networks where the batch size is 20 and the number of epochs is 2000. This configuration did not present any problems in any of the neural network models. The GRU model performed better with the ADAMAX optimizer and 30 neurons with an average MSE of $32.6 * 10^{-6}$ and a standard deviation of $38 * 10^{-6}$.

The LSTM model with the best performance was with the ADAMAX optimizer and 30 neurons, with an average MSE of $64 * 10^{-6}$ and a standard deviation of the MSE of $34.6 * 10^{-6}$. This model was superior to the LSTM model with 50 neurons.

Figure 5.3 shows that in this second configuration the LSTM BLSTM and GRU models improve significantly and follow the historical behavior of the series, likewise the ARIMA model does not compete in its performance with these models. The graph shown shows that the models were trained with 30 neurons and a learning rate of 0.001. These models have a good fit in the Test group, however, another configuration will be considered to evaluate if the mean square error is reduced.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	228	402	**64**	611	874	221
	sd	96	77	34	548	26	91
BLSTM	mean	434	**346**	155	663	936	303
	sd	161	219	75	526	279	138
GRU	mean	**218**	467	**33**	886	910	**211**
	sd	293	153	37	193	425	108
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.2: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =20, and 2000 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

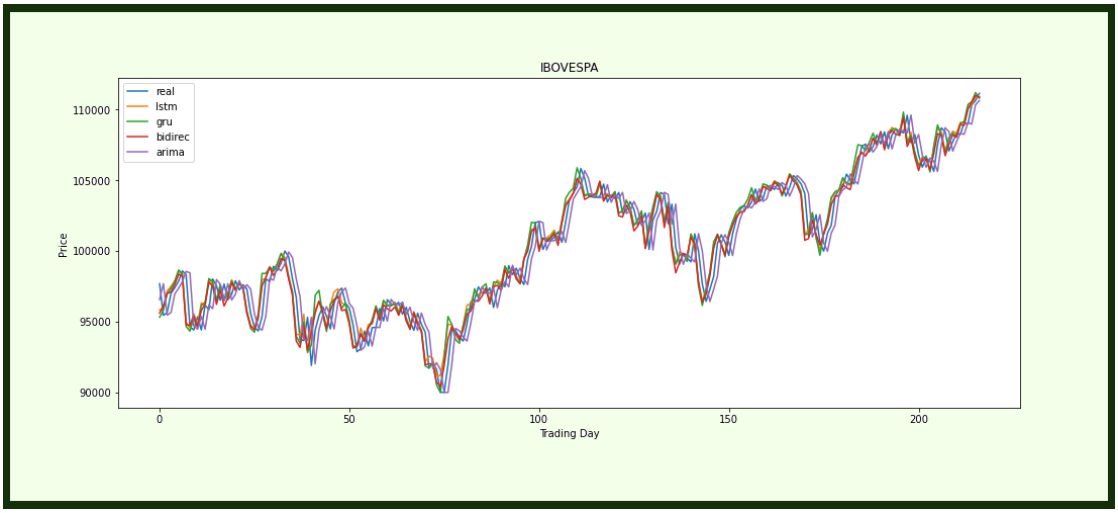

Figure 5.3: Sequence of real data and test data of the ARIMA, LSTM, BLSTM and GRU models, with lr=0.001, batch size=20, number of epochs=2000 with 30 neurons of the Ibovespa index in the period of 21-12-2018 to 23-12-2019.

Table 5.3 shows the results of the third configuration where the mean and standard deviation of the MSE of all the neural network models showed an evident decrease in the mean of the MSE compared to the previous configurations.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	21	41	22	99	96	67
	sd	20	38	16	37	32	52
BLSTM	mean	71	100	59	112	116	38
	sd	14	39	42	33	53	18
GRU	mean	**18**	**11**	**12**	**19**	**28**	**14**
	sd	12	4	9	9	39	7
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.3: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =10, and 1000 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

The lowest average of the MSE $10.5 * 10^{-6}$ corresponds to the GRU model with 30 neurons and the NADAM optimizer. Likewise, the results of the simulations were more homogeneous with a standard deviation of the MSE of $3.8. * 10^{-6}$. On the other hand, the LSTM networks had a better performance with the ADAMAX optimizer and 30 neurons.

Finally, in this third configuration, the BLSTM networks also showed better performance with the use of the ADAMAX optimizer and 50 neurons. The application of this third configuration leads us to think that it is necessary to establish other configurations to achieve better network performance. To achieve better network performance, it is necessary to vary one or more hyperparameters in order to optimize the low cost function in the shortest time possible under some other configuration.

Figure 5.4 shows the history of the IBOVESPA series, the predictions of the ARIMA model and the predictions of the neural network models (LSTM, BLSTM and GRU). It is observed that in this configuration, all the models of neural networks follow the same behavior of the original series. In the same way, it is observed that in the LSTM, BLTSM and GRU models they present a better behavior and this is evidenced by superimposing the original series. However, it is necessary to establish other configurations with a lower cost function.

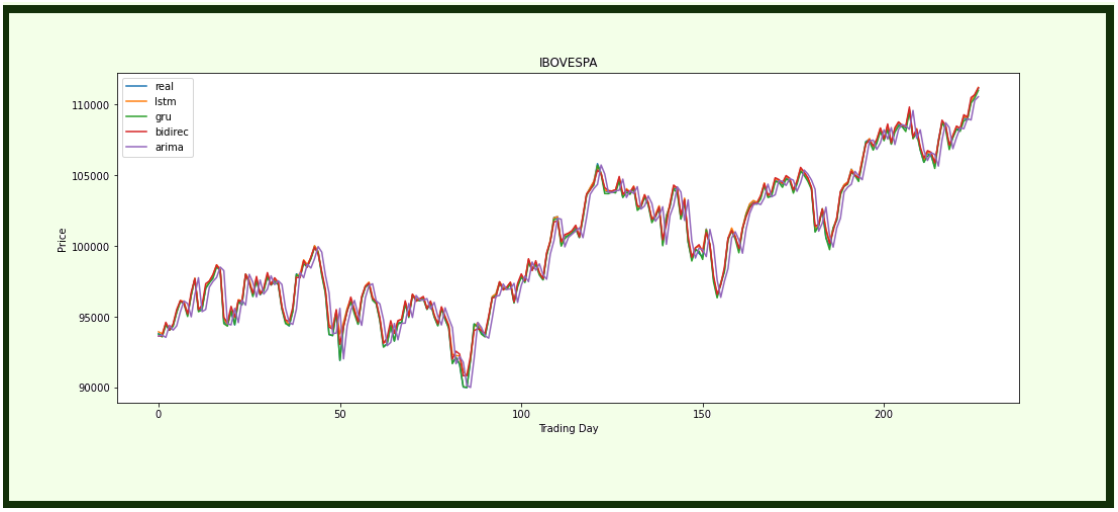

Figure 5.4: Sequence of real data and test data of the ARIMA, LSTM, BLSTM and GRU models, with lr=0.001, batch size=10, number of epochs = 1000 and 30 neurons of the four models for the Ibovespa index in the period from 21-12-2018 to 23-12-2019.

Table 5.4 presents the fourth configuration where the batch size, the number of epochs and the learning rate were 5, 1000 and 0.001 respectively. The GRU model with 30 neurons and the ADAMAX optimizer had the lowest average mean square error ($7.2 * 10^{-6}$) and the lowest MSE standard deviation ($5.4 * 10^{-6}$). Likewise, in the LSTM model with the ADAMAX optimizer and 30 neurons, the average mean square error and standard deviation of the MSE also improved with $13.9 * 10^{-6}$

and $6.7 * 10^{-6}$, respectively. Finally, the BLSTM model also improved, however, it is still inferior to the other neural network models. In the BLSTM models, the average root mean square error ($16.9 * 10^{-6}$) and standard deviation of the MSE ($2.5 * 10^{-6}$) obtained are smaller when works with 50 neurons.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	27	21	14	46	38	20
	sd	8	9	7	23	15	23
BLSTM	mean	40	30	42	64	50	17
	sd	19	15	49	33	21	3
GRU	mean	**11**	**18**	**8**	**8**	**17**	**15**
	sd	6	18	6	9	4	16
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.4: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =5, and 1000 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

Figure 5.5 shows the IBOVESPA series together with the predictions of the LSTM, BLSTM and GRU and ARIMA models. According to the image shown, it is observed that the ARIMA model shows a lower performance compared to the recurrent neural network models. However, it is necessary to make other configurations to establish if it is possible to obtain a better model for the calculation of the forecasts.

It is important to evaluate how the modification of the hyperparameters generates a decrease in the mean square error, Figure 5.6 shows the results that the loss function of the training group converges in $444.84 * 10^{-6}$. and the validation group

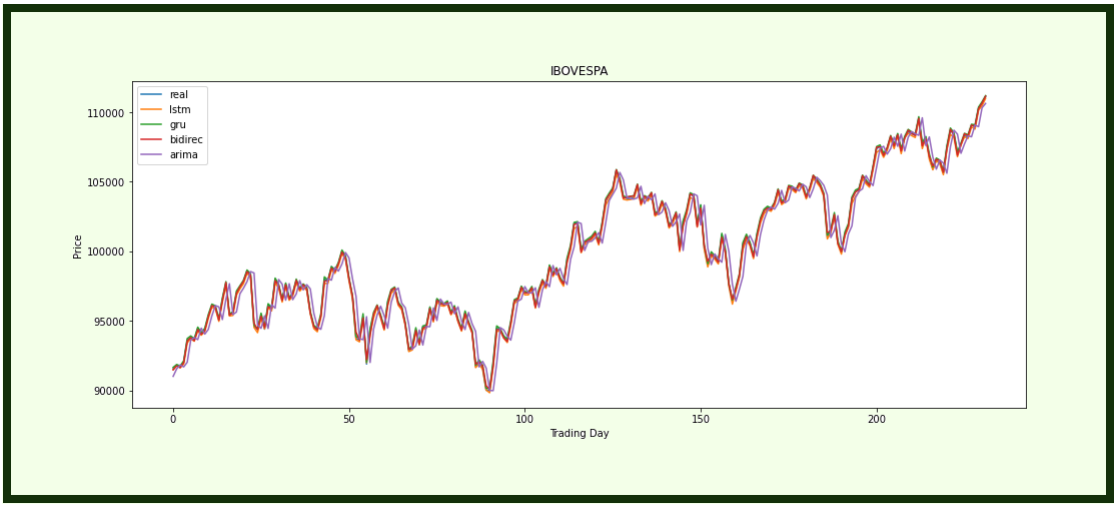

Figure 5.5: Sequence of real data and test data of the LSTM, BLSTM and GRU models, with lr=0.001, batch size=5, number of epochs = 1000 and 30 neurons of the four models for the Ibovespa index in the period from 21-12-2018 to 23-12-2019.

converges to $3300 * 10^{-6}$.

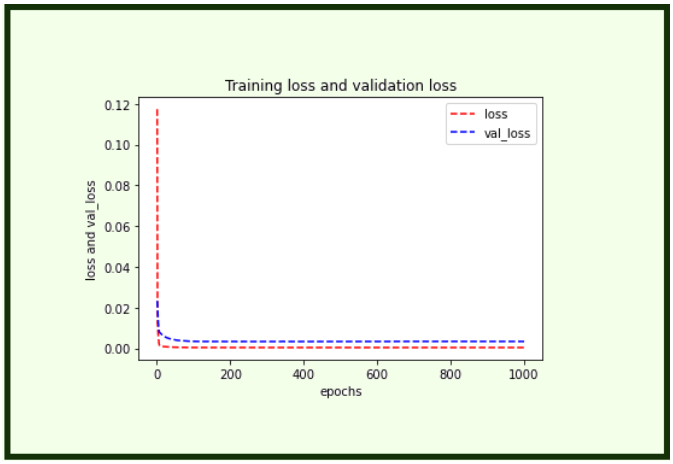

Figure 5.6: Sequence of the loss function of the training and validation group according to the number of epochs

The evaluation of a fifth configuration considers reducing the number of epochs while keeping the same the learning rate, the batch size of the optimizers and

the number of neurons as indicated above. Table 5.5 shows the results of this configuration. The GRU model had little improvement in 30-neuron configurations as well as 50-neuron configurations with the ADAMAX optimizer. However, the model performed better with 50 neurons calculating an average root mean square error of $6.1 * 10^{-6}$ and a standard deviation of the MSE of $3.9 * 10^{-6}$, values slightly lower than the model with 30 neurons. The BLSTM model performed better when 50 neurons and the NADAM optimizer were considered with an average mean square error of $22.9 * 10^{-6}$ and a standard deviation of the MSE of $6 * 10^{-6}$.. Considering that the results of the fifth configuration and fourth configuration were similar, it is necessary to search for a new configuration to minimize the cost function.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	25	29	98	42	23	40
	sd	13	38	53	41	12	23
BLSTM	mean	32	36	31	30	23	31
	sd	24	25	30	18	6	18
GRU	mean	**21**	**19**	**7**	**19**	**9**	**6**
	sd	20	20	4	15	6	4
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.5: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =5, and 500 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

The series in Figure 5.5 and 5.7 are similar, however, this difference can be seen in Tables 5.4 and5.5. Both models follow the same pattern of the historical series.

The sixth configuration shown in table 5.6 was made with a minimum variation

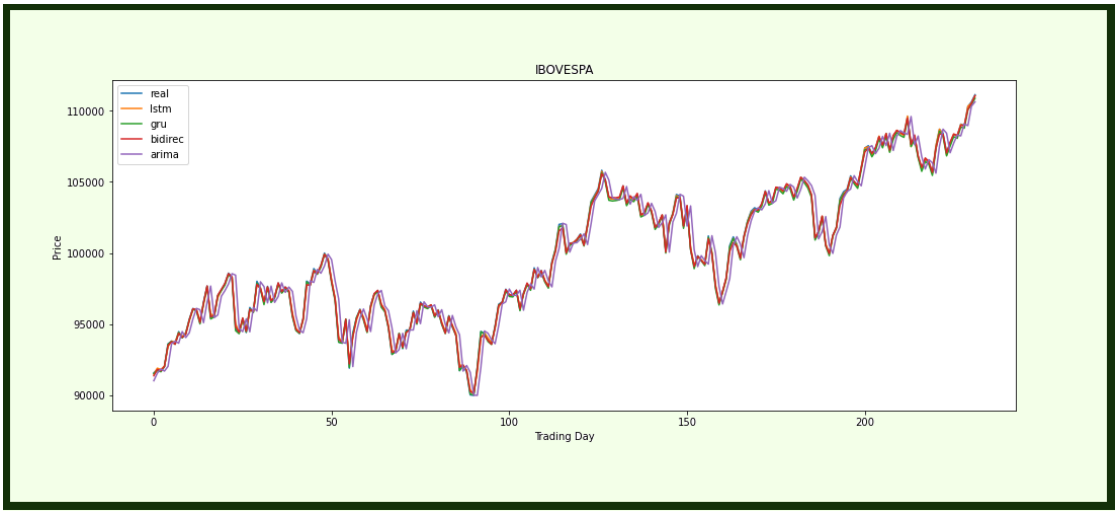

Figure 5.7: Sequence of real data and test data of the LSTM, BLSTM and GRU models, with lr=0.001, batch size=5, number of epochs = 500 and 30 neurons of the four models for the Ibovespa index in the period from 21-12-2018 to 23-12-2019.

where the batch size = 10 and the same number of epochs = 500 was considered, preserving the same hyperparameters established in the previous configurations. The GRU model with 30 neurons and the ADAMAX optimizer presented a better performance obtaining the average value of the mean square error of $5.8 * 10^{-6}$ and the standard deviation of the MSE of $4.8 * 10^{-6}$. This model has calculated on average the smallest mean squared error compared to the previous configurations. The LSTM model did not exceed previous configurations. However, when we consider 30 neurons and the ADAMAX optimizer, the average of the mean square error was $16.7 * 10^{-6}$ and the MSE standard deviation was $10.2 * 10^{-6}$. Likewise, the BLSTM model did not outperform previous configurations. However, when considering 50 neurons and the ADAMAX optimizer, the average root mean square error is $43.2 * 10^{-6}$. and the standard deviation of the MSE is $11.1 * 10^{-6}$.

Figure 5.8 presents the ARIMA, LSTM, BLSTM and GRU models and it is observed that there is no clear difference in the LSTM, BLSTM and GRU

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	45	28	17	45	43	33
	sd	44	25	10	25	26	17
BLSTM	mean	61	72	46	46	50	43
	sd	66	86	46	43	33	43
GRU	mean	**11**	**20**	**6**	**14**	**11**	**15**
	sd	4	20	5	8	7	7
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.6: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =10, and 500 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

models and they follow the same historical behavior of the IBOVESPA series. These configurations make a good fit and we can only distinguish these differences through the mean square error.

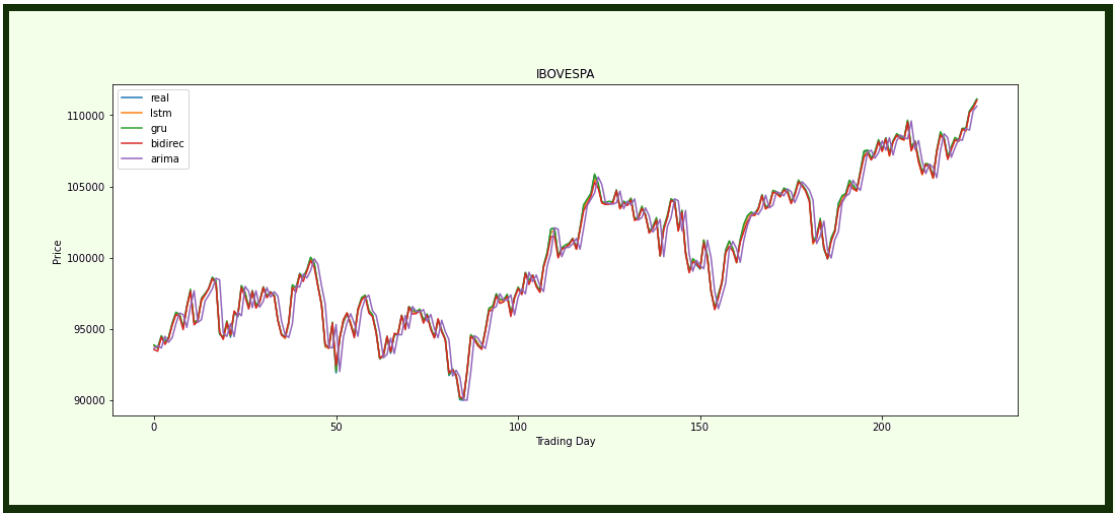

Figure 5.8: Sequence of real data and test data of the LSTM, BLSTM and GRU models, with lr=0.001, batch size=10, number of epochs = 500 and 30 neurons of the four models for the Ibovespa index in the period from 21-12-2018 to 23-12-2019.

This seventh configuration shown in table 5.7 search to reduce the execution

time and considers the same batch size of number = 10 but with a number of epochs = 250, preserving the same hyperparameters established in the previous configurations. . The GRU model in none of the configurations had a better performance considering the designed combinations. The LSTM model obtained the same mean square error 16.7*10-6 but a lower standard deviation of the MSE with a value of $3.5 * 10^{-6}$ and the NADAM optimizer. In the BLSTM model, it was inferior to the other recurrent neural network models.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	21	40	22	66	16	17
	sd	20	38	15	40	11	4
BLSTM	mean	19	73	44	80	39	25
	sd	9	59	19	46	19	8
GRU	mean	**15**	**69**	**16**	**43**	**12**	**19**
	sd	15	95	12	53	11	31
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.7: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =10, and 250 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

In the execution of this seventh configuration in figure 5.9, the historical or real data of the IBOVESPA series is observed together with the predictions of the four models (ARIMA; LSTM, BLSTM and GRU) under study, where It is observed that the predictions of the proposed models follow the same behavior and, as mentioned, who shows the greatest displacement in the ARIMA model. Therefore, as in figures 5.7 and 5.8, we state that to distinguish these differences and determine the best model, it is necessary to see the smallest mean square error (table 5.7).

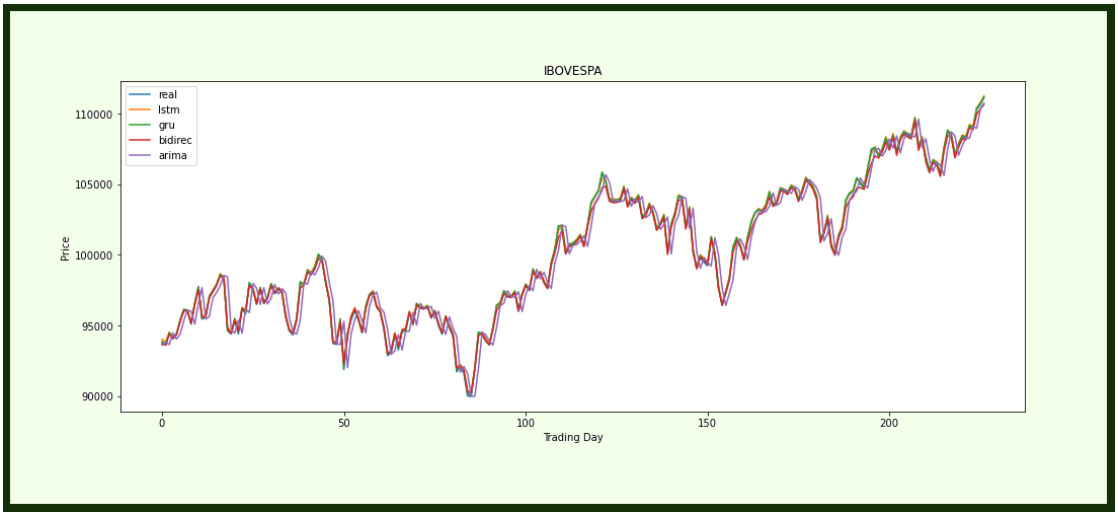

Figure 5.9: Sequence of real data and test data of the LSTM, BLSTM and GRU models, with lr=0.001, batch size=10, number of epochs = 250 and 30 neurons of the four models for the Ibovespa index in the period from 21-12-2018 to 23-12-2019.

According to the results found in the last two configurations, see table 5.6,and table 5.7, there is not much difference in the values of the mean squared error average. However, a final configuration will be calculated considering a smaller number of epochs and batch size to evaluate if by reducing the execution time and preserving the same hyperparameters established in previous configurations, the performance of the considered cost function can be improved.

This last configuration shown in table 5.8, has considered reducing the number of epochs to 250 and a batch size equal to 5. The GRU model with the ADAM optimizer and 30 neurons improved its performance in comparison to the other models of recurrent neural networks. However, it was slightly lower in mean square error at the sixth configuration calculating the mean square error value at $6.2*10^{-6}$ and a standard deviation of the MSE of $3.4*10^{-6}$. The LSTM and BLSTM models did not show an improvement in any of the preset configurations, on the contrary, in some of the configurations they had a lower performance.

		30 Neurons			50 Neurons		
	MSE	Adam	Nadam	Adamax	Adam	Nadam	Adamax
LSTM	mean	24	29	97	13	29	37
	sd	13	38	53	7	39	9
BLSTM	mean	19	73	44	80	39	25
	sd	9	59	19	46	19	8
GRU	mean	**15**	**69**	**16**	**43**	**12**	**19**
	sd	15	95	12	53	11	31
ARIMA	mean	9186	9186	9186	9186	9186	9186
	sd	0	0	0	0	0	0

Table 5.8: Estimation of the average of the MSE ($\times 10^{-6}$) and SD of the MSE ($\times 10^{-6}$) of the LSTM, BLSTM and GRU models with batch size =5, and 250 epochs with 30 and 50 neurons and the estimated MSE ($\times 10^{-6}$) of the ARIMA model (5,2,0).

Figure 5.10 presents the ARIMA, LSTM, BLSTM and GRU models and, as in figure 5.9, it is observed that there is no clear difference in these models. Likewise, they maintain the same historical behavior of the IBOVESPA series. These neural network configurations have a good fit and can only be distinguished through the mean square error.

In the study of these models there are 3 configurations where the value of the mean square error was smaller. The choice of these models corresponds to the GRU model where there is a minimal difference between them. Table 5.9 shows the 3 most relevant models of the research.

In one of the three resulting models, when we modify the dimensionality of the output space, similar results are obtained. In this configuration with 50 neurons, it presented a mean square error similar to the previous ones. However, we must choose one of these models and consider the convergence, dispersion and

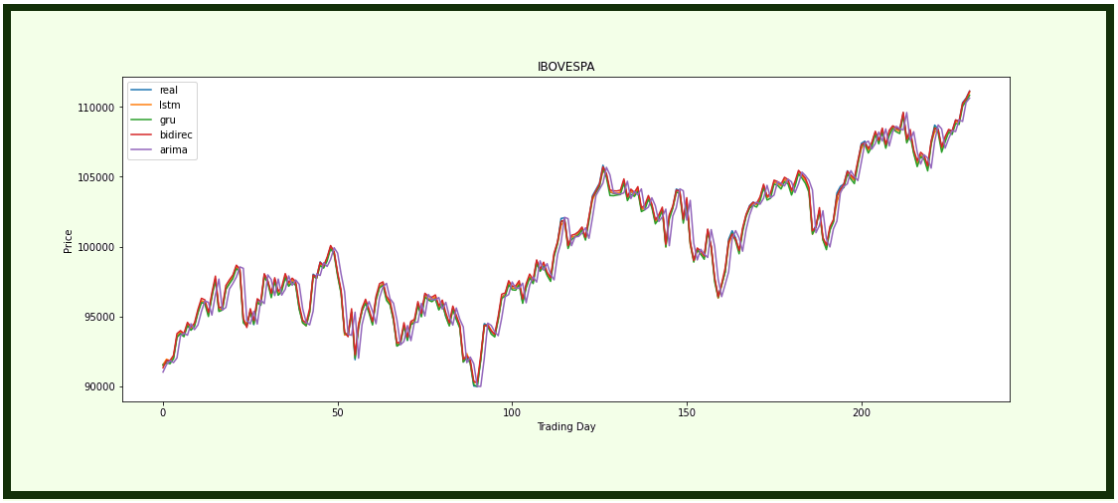

Figure 5.10: Sequence of real data and test data of the LSTM, BLSTM and GRU models, with lr=0.001, batch size=5, number of epochs = 250 and 30 neurons of the four models for the Ibovespa index in the period from 21-12-2018 to 23-12-2019.

execution time obtained in the simulations. The convergence of RNNs is an initial and essential step towards their successful applications. In the choice of the 3 final models, they show the same convergence when executing the replicas in the different groups where the loss function of the validation group in each of them was 0.0033, where approximately 64 weight updates were needed to optimize The learning. However, it can be thought that we would only need at most 100 epochs but this is not entirely true since the number of epochs is related to the shape of the data and when running 100, 150 and 200 epochs, the average error mean square and standard deviation were higher in these configurations.

Below we show in Table 5.10 the percentage difference between the actual data and the forecast data.

Taking into consideration the predictive capacity of these models, it is necessary to remember that under these configurations it was possible to minimize the mean

	GRU Model		
	CFG 1	CFG 2	CFG 3
Opt. Adamax	0.001	0.001	0.001
Batch Size	5	5	10
Epochs	250	500	500
Neurons	30	50	30
Average MSE ($\times 10^{-6}$)	6.2	6.1	**5.8**
SD MSE ($\times 10^{-6}$)	**3.4**	3.9	4.8
Average time (secs.)	29.7	83.3	110.3

Table 5.9: Performance of the 3 best GRU models

square error in the predictors. Likewise, these configurations will minimize the mean square error in the predictions or forecasts at any horizon.

Table 5.10 shows that the first forecast value is not always the one with the most precision. It should be reminded that these methods, even though they provide better forecasts than the ARIMA model or other forecasting techniques, they are quite limited, for this reason new models are being explored to achieve a better forecast in the results of a given problem.

	GRU Model		
	Real	Forcasts	Difference %
Day 1	118573.00	115749.83	2.38
Day 2	117707.00	115652.14	1.75
Day 3	116878.00	115540.22	1.14
Day 4	116662.00	115434.55	1.05
Day 5	116247.00	115340.57	0.78

Table 5.10: Forecasts of the GRU Model for the next 5 days

The proposed final model and its prediction results generated certain concerns, since we expect the first forecast value to have a lower percentage error between the real value and the predicted value. However, this result did not occur and we

found a slight decrease in the following days. These results did not improve with increasing or decreasing the number of data.

Chapter 6

Conclusions

The present investigation seeks to evaluate different models of recurrent neural networks (LSTM, BLSTM and GRU) in comparison to the ARIMA model, in order to determine which model is capable of better forecasting the closing price of 5 steps forward of the IBOVESPA stock market index. In the search to optimize the weights of the parameters minimizing the loss function, we can affirm that the ADAMAX optimizer has worked better compared to the other optimizers, since in approximately 80% of the simulations it has presented a lower mean square error. However, in the literature reviewed, most authors work with the ADAM optimizer. This first conclusion leads us to think that it is necessary to work simultaneously with other optimizers to reduce the risk and make better estimates in the forecasts.

In the review of articles and books, it was found that they mostly work with the ADAM optimizer. This optimizer offers good results when it comes to improving learning, however, when contrasting with other optimizers, it was found that there is a difference between them. Therefore, we can conclude that of the group of optimizers evaluated, the ADAMAX optimizer offers more satisfactory results in the different configurations. These results were more stable and it is because when

working with embedded models, this optimizer is superior.

It is important to keep in mind that from the fourth configuration the root mean square error has decreased a minimal amount and it has been found that there is not much difference in the following configurations. Likewise, it is possible to evaluate other configurations where the proportionality between the batch size and the number of neurons can be evaluated with the purpose of a faster training of the neural network.

When estimating the adequate number of neurons, we try to avoid difficulties and limitations in training, since considering a greater number of neurons increases time and cost, therefore it is necessary to reduce the number of neurons to achieve better performance in the training final model. The investigation showed that in all configurations of the GRU model with 30 neurons they presented better results than the GRU model with 50 neurons. On the other hand, the learning algorithm of the training data runs less than 1000 epochs, this model has a better performance than the LSTM and BLSTM models.

In the LSTM model of the different simulated configurations, the best performance was shown with a batch size of 5 and 1000 epochs and 30 neurons. Likewise, the BLSTM model presented better performance when considering a batch size of 5 and 500 epochs and 30 neurons. In conclusion, in all neural network models it performed better when 30 neurons were considered, and the GRU model is better than the LSTM and BLSTM models. However, models with better performance have been obtained where the configuration simulation considers 50 neurons.

In neural network models we fix the learning rate, since the lower the learning

rate, the more training epochs will be required. Initially, 0.01, 0.001 and 0.0001 were considered and after carrying out some simulations it was decided to set it at 0.001 since it did not affect the computational time.

As previously mentioned, there are multiple hyperparameters for network training and the batch size is the number of samples between updates of the model weights, that is, it is used to balance the learning speed and computational efficiency. From the different configurations executed, we can conclude that when the batch size was reduced, performance improved and this is seen in the three GRU models selected at the end of the simulations.

The backpropagation algorithm requires the training of the network for a suitable and specific number of epochs in the training data set. The proper number of epochs ensures that too many input patterns are not loaded into memory at once in the optimization process. We can conclude that the GRU model does not require a large number of epochs to achieve good performance in an attained model.

Bibliography

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Ed. by Springer. 2018, 2018.

[2] C Alves. "B3 atinge 4,2 milhões de investidores pessoas físicas em renda variável". In: *Brazilian Review of Finance* 17.3 (2019), pp. 47–65.

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

[4] Alexei Botchkarev. "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology". In: *arXiv preprint arXiv:1809.03006* (2018).

[5] Jason Brownlee. *Redes Neuronales Deep Learning*. Ed. by Independently published. 2017, 2017.

[6] Massimo Buscema. "Back propagation neural networks". In: *Substance use & misuse* 33.2 (1998), pp. 233–270.

[7] F Henrique Castro et al. "Fifty-year History of the Ibovespa". In: *Brazilian Review of Finance* 17.3 (2019), pp. 47–65.

[8] Tianfeng Chai and Roland R Draxler. "Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature". In: *Geoscientific model development* 7.3 (2014), pp. 1247–1250.

[9] Daming Shi Daniel S. Yeung Ian Cloete and Wing W.Y. Ng. *Sensitivity Analysis for Neural Network*. Ed. by Springer. 2009.

[10] Rahul Dey and Fathi M Salem. "Gate-variants of gated recurrent unit (GRU) neural networks". In: *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE. 2017, pp. 1597–1600.

[11] Ke-Lin Du and Madisetti NS Swamy. *Neural networks and statistical learning*. Springer Science & Business Media, 2013.

[12] Santiago Fernández, Alex Graves, and Jürgen Schmidhuber. "Phoneme recognition in TIMIT with BLSTM-CTC". In: *arXiv preprint arXiv:0804.3269* (2008).

[13] Fernando Berzal Galiano. *Long Short-Term Memory Networks With Python*. Ed. by Machine Learning Mastery. 2018, 2018.

[14] Gregory C. Reinsel George E. P. Box Gwilym M. Jenkins and Greta M. Liung. *Time series analysis : forecasting and control*. Ed. by John Wiley Sons. Fifth edition. 2016, 2016.

[15] Gregory C. Reinsel George E. P. Box Gwilym M. Jenkins and Greta M. Ljung. *Time series Analysis*. Ed. by Wiley. Fifth edition. 2016, 2016.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning [pre-pub version]*. Ed. by MIT Press. 2016, 2016.

[17] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Ed. by Springer. 2012, 2012.

[18] Alex Graves et al. "Unconstrained on-line handwriting recognition with recurrent neural networks". In: *Advances in neural information processing systems* 20 (2007).

[19] Simon Haykin. *Redes neurais: princıpios e prática*. Bookman Editora, 2001.

[20] J. Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". Diploma thesis,Institut fur Informatik, Lehrstuhl Prof.Brauer,Technische Universitat Munchen. www7.informatik.tu-muenchen.de/ hochreit., 1991.

[21] Sepp Hochreiter. "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[22] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[23] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. Ed. by Amazon. Second edition. 2018, 2018.

[24] Hitoshi Iba and Nasimul Noman. *Deep Neural Evolution: Deep Learning with Evolutionary Computation*. Ed. by Springer. 2020, 2020.

[25] Pedro Lara-Benıtez, Manuel Carranza-Garcıa, and José C Riquelme. "An experimental review on deep learning architectures for time series forecasting". In: *International journal of neural systems* 31.03 (2021), p. 2130001.

[26] Pedro Larranaga, Inaki Inza, and Abdelmalik Moujahid. "Tema 8. redes neuronales". In: *Redes Neuronales, U. del P. Vasco* 12 (1997), p. 17.

[27] N.D Lewis. *Deep Time Series Forecating with Python: An Intuitive Introduction to Deep Learning for Applied Time Series Modeling*. Ed. by Springer. 2016, 2017.

[28] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity. Bulletin of mathematical biophysics, vol. 5, pp. 115–133". In: *Journal of Symbolic Logic* 9.2 (1943).

[29] Pedro A. Morettin and Clélia M. C. Toloi. *Análise de Séries Temporais*. Ed. by Blucher. Segunda edição. 2006, 2006.

[30] John Murphy. *Trading Strategies - John Murphy'S Ten Laws Of Technical Trading*. Ed. by Gestion 2000. Primera edição. 2000, 2000.

[31] Barak A Pearlmutter. "Gradient calculations for dynamic recurrent neural networks: A survey". In: *IEEE Transactions on Neural networks* 6.5 (1995), pp. 1212–1228.

[32] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[33] Hojjat Salehinejad et al. "Recent advances in recurrent neural networks". In: *arXiv preprint arXiv:1801.01078* (2017).

[34] Murat H Sazli. "A brief review of feed-forward neural networks". In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01 (2006).

[35] Robin M Schmidt. "Recurrent neural networks (rnns): A gentle introduction and overview". In: *arXiv preprint arXiv:1912.05911* (2019).

[36] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[37] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications*. Ed. by Springer. Fourth Edition. 2017, 2017.

[38] Ralf C Staudemeyer and Eric Rothstein Morris. "Understanding LSTM–a tutorial into Long Short-Term Memory Recurrent Neural Networks". In: *arXiv preprint arXiv:1909.09586* (2019).

[39] Ilya Sutskever. *Training recurrent neural networks*. University of Toronto Toronto, Canada, 2013.

[40] Xin Wang et al. "OGRU: An optimized gated recurrent unit neural network". In: *Journal of Physics: Conference Series*. Vol. 1325. 1. IOP Publishing. 2019, p. 012089.

[41] Ronald J. Williams and David Zipser. "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks". In: *Neural Computation* 1.2 (June 1989), pp. 270–280. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.2.270. eprint: https://direct.mit.edu/neco/article-pdf/1/2/270/811849/neco.1989.1.2.270.pdf. URL: https://doi.org/10.1162/neco.1989.1.2.270.