Project 3: UiO: FYS-STK4155: Financial time series forecasting using Recurrent Neural Networks (RNN) and Long Short-Term Memory Cells (LSTM) and application

Fábio Rodrigues Pereira
fabior@uio.no - github: @fabiorod

December 16, 2020

Abstract

Predicting the stock market always is a topic of significant interest for scholars and professionals. However, the chaotic dynamics of the markets make this kind of study complex and challenging. In order to overcome these obstacles, this project proposes a day-trading tool based on outputs of Recurrent Neural Networks (RNN) and Long-Short Term Memory Networks (LSTM) for the decision-making process. We perform an RNN and an LSTM, with various combinations of parameters, to predict the next day's highest and lowest prices for the stock PETR4 (B3 Sao Paulo's Stock Exchange) and employ the best-trained network to an algorithm trading problem. Our best scenario, under RNN model, achieved training MSE accuracy of 0.38 and 0.36, Cross-Validated testing MSE of 0.48 and 0.45, training MAE of 0.43 and 0.40, and CV testing MAE of 0.45 and 0.50, for predicting the next day's highest and lowest, respectively. Besides, we compare our best network's performance in real-data with a traditional trading strategy (5-day Bollinger Bands) bench-marked. From 14.10.200 to 03.12.2020, based on our best RNN, our day-trading strategy makes a promising cumulative daily return of BRL10,76 per share (cumulative daily percentage of 39%), overcoming the traditional strategy that achieved a cumulative daily return of BRL1,95 per share (cumulative daily percentage of 7%).

Contents

1 Introduction **3**
1.1 Source code . . . 3

2 Theory **4**
2.1 Recurrent Neural Network . . . 4
2.1.1 RNN's backpropagation . . . 6
2.1.2 Challenges of training the RNN model . . . 6
2.2 Long Short-Term Memory Cells (LSTM) . . . 7
2.3 Technical Analysis . . . 8
2.3.1 Simple Moving Average (SMA) . . . 8
2.3.2 Exponential Moving Average (EMA) . . . 9
2.3.3 Bollinger Bands (BB) . . . 9

3 Discussion and Results **10**
3.1 Handling the raw data and choosing a security . . . 10
3.2 Engineering the features and deciding the prediction's targets . . . 10
3.3 Time series Cross-Validation . . . 11
3.4 Training RNN model and optimizing hyper-parameters . . . 12
3.4.1 Predicting the highest price of the next day using RNN . . . 12
3.4.2 Predicting the lowest price of the next day using RNN . . . 14
3.5 Training LSTM model and optimizing hyper-parameters . . . 16
3.5.1 Predicting the highest price of the next day using LSTM . . . 17
3.5.2 Predicting the lowest price of the next day using LSTM . . . 18
3.6 Our optimal model . . . 20
3.7 Applying the predictions to an algorithm trading strategy . . . 22

4 Conclusions and Future Work **24**

5 Bibliography **25**

1 Introduction

This scientific report's idea consists of approaches in the area of Quantitative Finance Analysis field using computational mathematical models directly inspired by the human brain structure, so-called Artificial Neural Network (ANN) methods, for instance, Recurrnt Neural Network (RNN) and Long-Short Term Memory Network (LSTM).

Nowadays, professionals and quantitative researchers rely on many traditional and contemporary computational strategies for decision-making based on speed and accuracy. Therefore, in the face of big data analytic advances, the challenge is to find a reliable tool suitable for interpreting large amounts of different unstructured data available in the market today. Indeed, the dynamics, complexities, evolutive, and chaotic nature of the markets make this task unmanageable and sometimes ineffective.

For that, this report will explore computational learning process techniques to generalize the information obtained from interactions with the financial market environment. Today uncountable many promising researchers of ANN techniques highlight their capability to resolve complex sequential decision-making dilemmas, and this should not be a problem when applied in the financial time series forecasting field.

The assignment starts collecting and handling the various available data from B3 Sao Paulo's Stock Exchange [8], filtering, creating, and labeling the best features. In parallel, RNN and LSTM theories desire to be included and addressed. Next, it will be necessary to find feasible ways of feeding the ANN systems with the processed data. Then, assessment of the results, adjustments, new inquiries, and ideas could be analyzed. In the end, the research aims to propose a application of the optimized models to allow experts to enhance returns, reduce risk, and increase efficiency by systematically incorporating ANN outcomes for algorithm trading decision-making.

1.1 Source code

All codes utilized in this project is written in Python v.3.8, and found in the GitHub repository at https://github.com/fabiorodp/UiO-FYS-STK4155/tree/master/Project3. The repository contains:

- data/ - Directory containing the data utilized in this project.
- package/ - Directory containing the collection of python's methods utilized in the project.
- report/ - Directory containing the written report in latex and pdf.
- handling_day.py - Test script for handling the raw data to samples and features organized object.
- RNN.py - Test script for training the various SimpleRNN model with CV and parameters' combination.
- LSTM.py - Test script for training the various LSTM model with CV and parameters' combination.
- application.py - Test script containing all the processes of creation of an algorithm trading system.

2 Theory

This project will discuss predictions using two different Artificial Neural Networks (ANN) subclasses called Recurrent Neural Network (RNN) and Long Short-Term memory cells (LSTM).

Typically, ANNs, like MLPs or CNNs, are not proficient in handling ordered input data because they do not have a memory of the past seen data. For instance, the input data passed through the feedforward (FF), backpropagation steps, and only update the weights so that the order of the data is not relevant. Conversely, ANN's recurrent classes present ways to model sequences and can remember past information, as we will study in more detail in the next sub-topics.

2.1 Recurrent Neural Network

As we know from the FeedForward Neural Network (FFNN), the activation flow occurs only in one direction, from the input, passing through hidden-layers, until the output. The Recurrent Neural Network (RNN) keeps this principle but also add connections backward. In other words, in each time step (also called a frame), the RNN receives the input $x^{(t)}$ as well as the hidden-layer's activation from the previous time step $h_u^{(t-1)}$. This flow, called *unrolling the network through time* [3][p. 504] or *recurrent edge in graph notation* [2][p. 744], is illustrated by [2][p. 744] as follows:

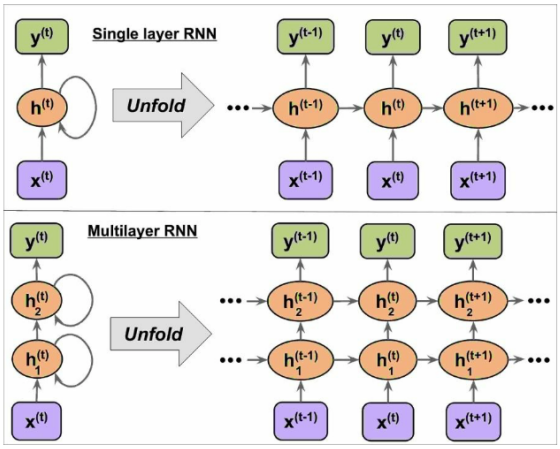

Figure 1: RNN's recurrent edge in graph notation for one hidden-layer and two hidden-layers.

Therefore the recurrent neuron has three sets of weights, $W_{(xh)}$ for the inputs $x^{(t)}$ and hidden-layer h, $W_{(hh)}$ associated with the recurrent edge, and $W_{(hy)}$ for the hidden-layer h and the output layer $y^{(t)}$. The activations of the hidden units at time step t is calculated by:

$$\begin{aligned} h^{(t)} &= \phi_h(z_h^{(t)}) = \phi_h(W_{xh}x^{(t)} + W_{hh}h^{(t-1)} + b_h) \\ &= \phi_h(W_h \begin{bmatrix} x^{(t)} \\ h^{(t-1)} \end{bmatrix} + b_h) \end{aligned}$$

where b_h is the bias vector for the hidden-unit; $z_h^{(t)}$ is the net-input; $\phi_h(.)$ is the activation function for the hidden-unit; $x^{(t)}$ is the input containing the features of the instances; W_xh and W_hh weights containing the connections between input and hidden-layer, and hidden-layer and hidden-layer, respectively; W_h is a concatenation of $[W_{(xh)}; W_{(hh)}]$ containing the hidden-layer's weights at time t for each instance m of the mini-batch.

Once the activation of hidden units $h^{(t)}$ is computed, the activation of output units will be:

$$y^{(t)} = \phi_y(W_{hy}h^{(t)} + b_y)$$

where b_y is the bias vector for the hidden unit and output-layer; W_{hy} is the weight for the hidden-layer and output-layer.

Everything is summarized by the following picture from [2][p. 748]:

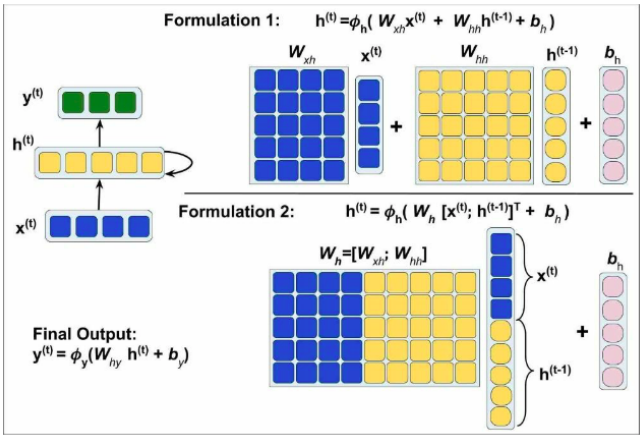

Figure 2: The process of computing the activations.

Hence, these recurrent neural network cells acquire a kind of memory so-called memory cells, in which there are many different applications, translated by the picture [3][p. 509] next:

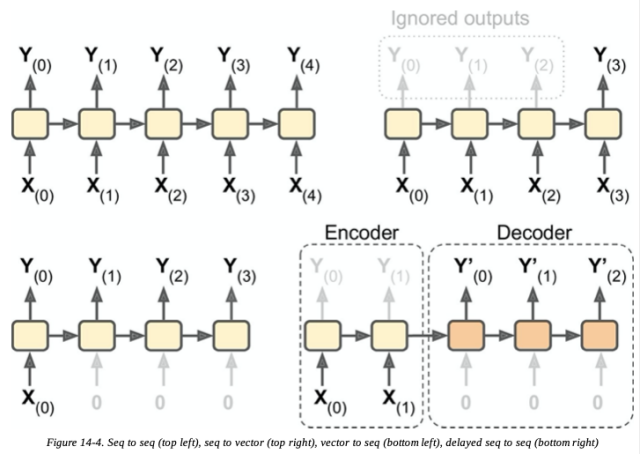

Figure 3: Different applications of RNN method.

The picture above shows four different applicabilities of the RNN. The first (top left) is a time series system where the inputs are values of the last N days, and the outputs are the predictions of the value shifted by one day into the future. Alternatively (top right), one could feed the RNN with a sequence of inputs and ignore all outputs except for the last one, expressing a prediction based on all previous inputs and memory cells. Conversely (bottom left), you can give a single input, and the

series predictions are a function of this first input and the series of memory cells. Finally (bottom right), you can have an encoder (sequence-to-vector network) and a decoder (vector-to-sequence network) [3][p. 508] system, in which the predictions are a function of the entire encoder and memory cells.

This report will apply the top left system, also called *many-to-many* method [2][p. 740-1], where the input will be many different time series quotations (multi-features) of a given financial asset. The aim here is to predict a series of outputs to be used as a tool in a decision-making process (trading strategy).

2.1.1 RNN's backpropagation

The gradients' main idea is that the overall loss of L is the sum of all loss functions from time $t = 1$ to $t = T$:

$$L = \sum_{t=1}^{T} L^{(t)}$$

and the gradients:

$$\frac{\partial L^{(t)}}{\partial W_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \times \frac{\partial y^{(t)}}{\partial h^{(t)}} \times (\sum_{k=1}^{t} \frac{\partial h^{(t)}}{\partial h^{(k)}} \times \frac{\partial h^{(k)}}{\partial W_{hh}})$$

and $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ can be calculated by:

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}}$$

2.1.2 Challenges of training the RNN model

The typical RNN creates common challenges in its training process, so-called vanishing or exploding the gradients' computations, which can be explained in the figure from [2][p. 750] below:

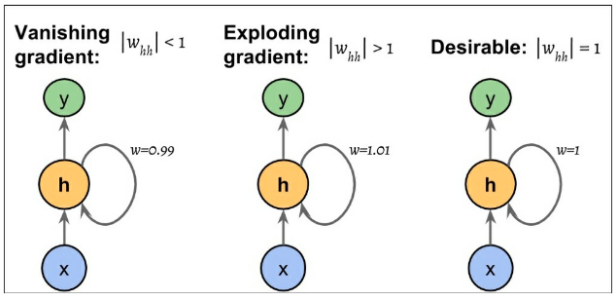

Figure 4: Vanishing or exploding the gradients' computations.

The multiplicative factor $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ in the computation of the gradients of the loss function has $t - k$ multiplications. As a result, when $t - k$ is small, the factor $W^{(t-k)}$ becomes very small, vanishing

the gradient. On the other hand, when the long-range dependencies $t-k$ is large, the factor $W^{(t-k)}$ becomes very large, exploding the gradient.

As we will see, the LSTM class explores some modern approaches and solutions to these challenges.

2.2 Long Short-Term Memory Cells (LSTM)

The LSTMs were introduced by *Long Short-Term Memory, S. Hochreiter and J. Schmidhuber, Neural Computation, 9(8): 1735-1780, 1997*, where it was proposed a solution for the vanishing gradient problem. This method creates a building block (memory cell), giving a new representation for the hidden layer, illustrated by [2][p. 752] as follows:

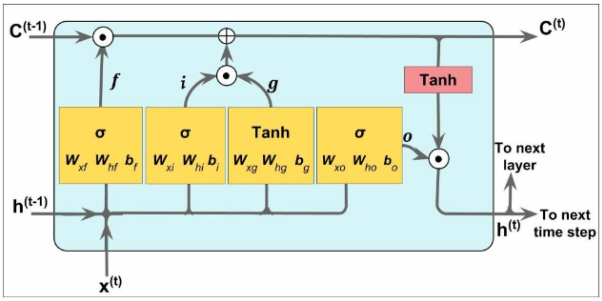

Figure 5: LSTM's memory cell to avoid vanishing the gradient's computations.

First, the LSTM's cell contains four yellow boxes indicated with an activation function in the top and weights in the bottom. Each box performs a linear combination by matrix-vector multiplications on their input and weights, resulting in their output.

Next, this output passes through gates represented by a circle with a point in the middle ($\odot$:= element-wise product/multiplication [2][p. 752]). Each of the three gates in the cell has a specific function, as follows:

1st.: The **forget gate** (f_t) resets the cell state and decides which information passes through or suppresses. This function is defined by:

$$f_t = \sigma(W_{xf}x^{(t)} + W_{hf}h^{(t-1)} + b_f)$$

2nd.: The **input gate** (i_t) and **input node** (g_t) updates the cell state and is defined by:

$$i_t = \sigma(W_{xi}x^{(t)} + W_{hi}h^{(t-1)} + b_i)$$

and:

$$g_t = tanh(W_{xg}x^{(t)} + W_{hg}h^{(t-1)} + b_g)$$

3rd.: The **output gate** (o_t) decides how the hidden-units updates:

$$o_t = \sigma(W_{xo}x^{(t)} + W_{ho}h^{(t-1)} + b_o)$$

Subsequently, the previous cell state $C^{(t-1)}$ is adjusted to return the current cell state $C^{(t)}$ without being directly multiplied by any weighting factor. Remark that the value compared to the RNN's recurrent edge is called cell state $C^{(t)}$ for the LSTM's models. The cell state $C^{(t)}$ is calculated as follows:

$$c^{(t)} = (c^{(t-1)} \bigodot f_t) \bigoplus (i_t \bigodot g_t)$$

where $\bigoplus$ is the element-wise-summation or addition [2][p. 753].

Hence, the hidden-units at the current time step t is measured by:

$$h^{(t)} = o_t \bigodot tanh(c^{(t)})$$

2.3 Technical Analysis

Technical analysis (TA) is a tool or trading policy employed in the decision-making process in investments. Typically, TA is focused on identifying opportunities by analyzing the market's statistical metrics, including prices, volumes, and more. Besides, TA is regularly used to generate short-term trading signals, evaluating the security's strength and weakness from historical trading data.

Importantly, we can not mistake Technical analysis (TA) from Fundamental analysis (FA), which the latter aims to assess a security's price based on business results.

In this project, we will use as features or bench-mark three different TA's tools, the so-called Simple Moving Average (SMA), Exponential Moving Average (EMA), and Bollinger Bands (BB).

2.3.1 Simple Moving Average (SMA)

Simple Moving Average (SMA) computes the arithmetic/simple moving average of a given range of data, which could be prices, volumes, and more. In TA, SMA is widely used to indicate if financial security will continue an upward (bull) or downward (bear) trend. The formula for SMA is given by:

$$SMA_t = \frac{A_1 + A_2 + ... + A_T}{T} = \frac{1}{T}\sum_{t=0}^{T} A_t$$

where $t = 1, 2, ..., T$ and $t \in N$; A_t is the value of the data stored in time series; T is the number of total periods of the time series range.

In our GitHub repository for Project3, directory package/ there is a file technical_analysis.py containing a python script to return the SMA.

2.3.2 Exponential Moving Average (EMA)

Exponential Moving Average (EMA) is a weighted moving average type that attributes a greater weight on the most recent data points. Thus, the EMA's reaction is more significant to recent data points than for SMA, which applies the same weight for the entire range. Although that, likely the SMA, EMA is also widely used to indicate upward (bull) or downward (bear) trends, and its formula is given by:

$$EMA_t = \alpha A_t + (1 - \alpha) EMA_{(t-1)}$$

where $t = 1, 2, ..., T$ and $t \in N$; A_t is the value of the data stored in time series; T is the number of total periods of the time series range; α is a smoothing factor given by $\alpha = \frac{2}{(T+1)}$.

A detail that must be stated is that the EMA tool must start from some value, and it is a protocol that its first value is an SMA.

In our GitHub repository for Project3, directory package/ there is a file technical_analysis.py containing a python script to return the EMA.

2.3.3 Bollinger Bands (BB)

Bollinger Bands (BB) was created in the early 80s and represents a popular TA tool for finance traders. It is a measure of two standard deviations (std) away from a simple moving average (SMA) price representing overbought and oversold areas. Three lines represent the bands, the **upper band** is two std above the SMA, the **middle band** is the SMA, and the **lower band** is two std below the SMA. Typically, traders use this tool to spot sell/buy signals when the security price is above (overbought) the upper band or below (oversold) the lower bands, respectively.

In our GitHub repository for Project3, directory package/ there is a file technical_analysis.py containing a python script to return the BBs.

3 Discussion and Results

3.1 Handling the raw data and choosing a security

Our raw data came from B3 Brazilian's stock exchange, which can be download freely from *its website*. The files contain all negotiations (historical series of data) for all securities from a trading section (day). We have downloaded the files containing the data-series from 22.06.2020 to 04.12.2020, and the chosen security that our studies will focus on will be PETR4. For that, a python function called get_instrument() in the Project3/package/read_data.py file does the job of extracting all negotiations for PETR4.

It is essential to state that PETR4 is the biggest specialized company in Brazil's oil, natural gas, and energy industry. Besides, in the B3 market, this stock has one of the highest liquidity flow that guarantee us to avoid higher price gaps, slippage, or any other trading issues. These reasons justify our choice for PETR4, but any additional security can perform the following experiments.

3.2 Engineering the features and deciding the prediction's targets

Most of the studies using ANN for time series forecasting uses input data holding the stock daily (OLHC) opening, lowest, highest, and closing prices [5][6]. The input may include many other technical analysis (TA) or fundamental analysis (FA) indicators.

As this report considers short-term trading structures, there is no reason to use fundamentalists' indicators published quarterly. However, technical indicators are very relevant and useful to bring more information to our ANN's models. Hence, as introduced in the theory section of this work, we will use three technical tools, such that, Simple Moving Average (SMA), Exponential Moving Average (EMA), and Bollinger Bands (BBs).

Therefore, the 16 variables/features used as input to RNN and LSTM are described in the following table:

Features for training data	
Qtd	Description
4	The opening, lowest, highest, closing prices and volume (OLHCV);
9	The upper, middle and lower BBs for 5, 10 and 20 periods;
3	The EMAs for 5, 10 and 20 periods.

Table 1: Variables/features used as input to RNN and LSTM models.

Remark that these features test the ability of the RNN and LSTM to learn from previous information of the time series data. Besides, these features were chosen according to the author's trading practices, but it could have been chosen in any other preferable periods. As a future research objective, it would be a relevant idea to study the effect of several indicators in the learning and the appropriate amount of historical data that should be considered.

Regarding the prediction's targets, we want to design a valuable tool for algorithm tradings and decision-making processes. The idea is to predict the highest and lowest of the next trading section (next day). This information would bring a rouge advance and benefit for day-traders because they will start the day already knowing the probable highest and lowest and, consequently, avoid trading operations in the wrong direction against the oversold (lowest of the day) overbought (highest of the day) areas.

3.3 Time series Cross-Validation

The Machine Learning system, in general, learns the data's structure so that we can predict the future. When we fit and train the learning algorithm, it does not surprise the spectacular results that can be found. Still, it typically has zero forecasting power if not tested with unseen data [4][p. 131]. Therefore, one solution is to perform Cross-Validation (CV) splits, where we separate the entire data on training and testing sets.

Regarding time-series data, the typical CV, which handles independent identically distributed (IID) sets, fails in its proposes. Indeed, we are dealing with ordered, recurrent, and serially-correlated data, which can not be split into unordered blocks or even shuffled. Besides, it is also common to have leakage [4][p. 131] when pieces of information from the training set also arise in the testing sets. One widespread example of leakage is when one wants to predict a period's closing price giving the highest and lowest of the same time step. The ideal in this scenario is to provide the highest and lowest of the previous time step. Thinking on leakage, Professor Marcos Lopes de Prado elaborated a policy to reduce the likelihood of leakage, *ipsis verbis* [4][p. 132]:

1st. *Drop from the training set any observation i where Y_i is a function of information used to determine Y_j, and j belongs to the testing set.*

> ***(a)*** *For examples, Y_i and Y_j should not span overlapping periods.*

2nd. *Avoid over-fitting the classifier. In this way, even if some leakage occurs, the classifier will not be able to profit from it. Use:*

> ***(a)*** *Early stopping of the base estimators;*
>
> ***(b)*** *Bagging of classifiers, while controlling for oversampling on redundant examples, so that the individual classifiers are as diverse as possible.*
>
> > ***(b.i)*** *Set max_samples to the average uniqueness.*
> >
> > ***(b.ii)*** *Apply sequential bootstrap.*

In this project, we have followed Prado's policies for avoiding leakages.

For the CV technique, we have used the proposed methodology given by [9][Ch.3.4], which describes a procedure for testing unseen data with series of test sets containing a single observation, and the corresponding training set consists of rolled samples that occurred before. The diagram that illustrates the series of training and testing sets follows [9][Ch.3.4]:

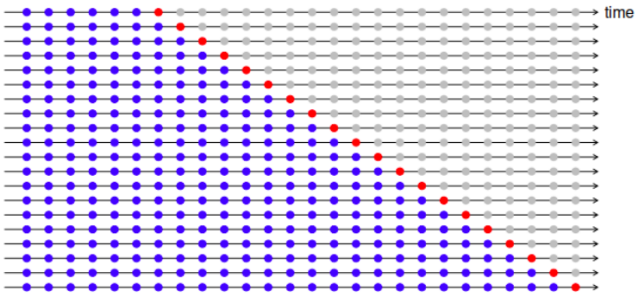

Figure 6: Time series CV using *"evaluation on a rolling forecasting origin"*[9][Ch.3.4]. Blue data represents the training set. Red data represents the testing set.

The accuracy of the predictions will be calculated by averaging over the Mean Squared Error (MSE) and Mean Absolute Error (MAE) resulted from all training and validations. This method is called *"evaluation on a rolling forecasting origin"* by the authors [9][Ch.3.4] because of the training sets based on rolls ahead in time.

Our script called studies.py in Project3/package/ directory performs a Time series Cross-Validation with parameter combination of Units and Epochs. The beginning's rolling value for training is defined by default as 60-time steps, which means that our ANN model will start the training with the first 60-days of data, and then it validates the 61's day and measure the testing metrics. The CV algorithm keeps doing the same procedure for the next steps, training 61-days and validating the 62's day, until all data have been gone. In the end, the average testing metrics are returned for an assessment of the best model and parameters.

3.4 Training RNN model and optimizing hyper-parameters

This section will feed and fit a SimpleRNN model from Keras with our features and targets using the *Time Series Cross-Validation* discussed before. A combination of parameters is applied with different numbers for units, several activation functions and various numbers of hidden-layers for epoch equals 50 and mini batch equals 1 avoiding costly computations, as follow:

Hyper-parameters	
Name	Values
Act. Functions	Sigmoid, Tanh, ReLu;
Hidden-layers	For Sigmoid and Tanh: 1, 2; For ReLu: 2, 3;
Units	50, 100, 150, 200, 400, 600, 800, 1000, 1200;
Epoch	50;
Mini-batch	1;
Loss-Function	Mean Squared Error (MSE);

Table 2: Hyper-parameters used for optimization of the SimpleRNN model.

Our aim here is to find the most promising model (lowest testing MSE and MAE) with the most efficient parameters (lowest number of units as function of hidden layers and activation function). For that, we need to investigate which parameter (activation function, hidden-layers, and units) combination is the best. Further, the experiments will be based on the loss-curve, the number of epochs and any other possible way of optimization.

The test file RNN.py stored in our GitHub repository replicates all the following results in the next sub-sections.

3.4.1 Predicting the highest price of the next day using RNN

We start experimenting the input data presented in Table 1 on the SimpleRNN model with various combinations of parameters as delineated in Table 2. The goal is to optimize the SimpleRNN model for predicting the highest price of the next day with the most promising and effective parameters as explained in *Taining RNN model and optimizing hyper-parameters* section.

Starting with Sigmoid's activation function and hidden-layers 1 and 2, the results follow:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Sigmoid	1	15	2.78	7.54	1.43	2.37
Sigmoid	1	50	2.06	7.93	1.25	2.45
Sigmoid	1	100	1.95	7.92	1.16	2.33
Sigmoid	1	150	1.76	7.57	1.04	2.21
Sigmoid	1	200	1.48	6.61	0.89	2.02
Sigmoid	1	400	0.73	2.77	0.57	1.33
Sigmoid	1	600	0.51	1.33	0.47	0.88
Sigmoid	1	800	0.42	0.82	0.42	0.63
Sigmoid	1	1000	0.38	0.65	0.40	0.52
Sigmoid	**1**	**1200**	**0.37**	**0.55**	**0.40**	**0.49**
Sigmoid	2	15	2.08	7.82	1.27	2.47
Sigmoid	2	50	0.54	1.63	0.50	0.99
Sigmoid	2	100	0.40	0.89	0.42	0.65
Sigmoid	2	150	0.41	0.90	0.42	0.66
Sigmoid	2	200	0.42	0.94	0.43	0.68
Sigmoid	2	400	0.43	0.81	0.43	0.62
Sigmoid	2	600	0.37	0.56	0.40	0.49
Sigmoid	**2**	**800**	**0.38**	**0.48**	**0.43**	**0.45**
Sigmoid	2	1000	0.44	0.49	0.47	0.48
Sigmoid	2	1200	0.52	0.56	0.51	0.52

Table 3: Parameters' combinations for predicting the next day's highest price.

Table 3 shows that the Testing MSE and MAE achieve their best-validated accuracy values of 0.55 and 0.49 for one hidden-layer and 0.48 and 0.45 for two hidden-layers. Note that the metrics for one hidden-layer have not hit the lowest by 1200 units, although it might have happened if we have increased the number of units even more. However, increasing the number of units above 1200 is very costly for computations, and it would be better practice to increase the number of hidden-layers instead. Indeed, the best accuracy metrics for two hidden-layers were by 800 units where is the optimal point for the model because, above that, the metrics worsen again.

Next, we perform Tanh's activation function for hidden-layers 1 and 2, the outcomes go:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Tanh	1	15	2.08	7.82	1.27	2.47
Tanh	1	50	1.96	7.97	1.17	2.34
Tanh	1	100	1.46	6.52	0.88	2.01
Tanh	1	150	0.98	4.32	0.67	1.65
Tanh	1	200	0.73	2.76	0.57	1.34
Tanh	1	400	0.42	0.83	0.42	0.63
Tanh	1	600	0.37	0.56	0.40	0.49
Tanh	**1**	**800**	**0.38**	**0.48**	**0.43**	**0.45**
Tanh	1	1000	0.44	0.49	0.47	0.48
Tanh	1	1200	0.52	0.56	0.51	0.52
Tanh	2	15	2.08	7.82	1.27	2.47
Tanh	2	50	0.42	1.00	0.43	0.71
Tanh	2	100	0.42	1.04	0.43	0.73
Tanh	2	150	0.46	1.19	0.45	0.82
Tanh	2	200	0.50	1.27	0.47	0.85
Tanh	2	400	0.45	0.80	0.43	0.62
Tanh	2	600	0.37	0.56	0.41	0.48
Tanh	**2**	**800**	**0.39**	**0.51**	**0.44**	**0.47**
Tanh	2	1000	0.44	0.49	0.47	0.48
Tanh	2	1200	0.52	0.56	0.51	0.52

Table 4: Parameters' combinations for predicting the next day's highest price.

Table 4 exhibits that the Testing MSE and MAE obtain their best-validated accuracy values of 0.48 and 0.45 for one hidden-layer and 0.51 and 0.47 for two hidden-layers. Remark that the

optimal point happened at units equals 800 for both one or two hidden-layers. Thus, we would choose one-hidden layer as the best model because it is costly less-expensive and got slightly better metrics than two hidden-layers.

Finally, ReLu's activation function is run for hidden-layers 2 and 3 with the measures:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
ReLu	2	15	2.13	7.55	1.29	2.43
ReLu	2	50	1.96	7.97	1.17	2.34
ReLu	2	100	1.46	6.52	0.88	2.01
ReLu	2	150	0.98	4.32	0.67	1.65
ReLu	2	200	0.73	2.76	0.57	1.34
ReLu	2	400	0.42	0.83	0.42	0.63
ReLu	2	600	0.37	0.56	0.40	0.49
ReLu	**2**	**800**	**0.38**	**0.48**	**0.43**	**0.45**
ReLu	2	1000	0.44	0.49	0.47	0.48
ReLu	2	1500	0.70	0.60	0.55	0.55
ReLu	3	15	2.08	7.82	1.27	2.47
ReLu	3	50	0.42	0.99	0.43	0.71
ReLu	3	100	0.42	1.05	0.43	0.73
ReLu	3	150	0.46	1.19	0.45	0.82
ReLu	3	200	0.50	1.29	0.47	0.86
ReLu	3	400	0.44	0.80	0.43	0.62
ReLu	3	600	0.37	0.56	0.41	0.48
ReLu	**3**	**800**	**0.39**	**0.48**	**0.44**	**0.45**
ReLu	3	1000	0.44	0.49	0.47	0.48
ReLu	3	1200	0.52	0.56	0.51	0.52

Table 5: Parameters' combinations for predicting the next day's highest price.

Table 5 reveals that the Testing MSE and MAE reach their best-validated accuracy values of 0.48 and 0.45 for two hidden-layer and 0.48 and 0.45 for three hidden-layers. Important to state that ReLu with one hidden-layer makes the computations explode to a very high value, then it was not considered in this report. On the other hand, there is no relevant difference between training ReLu with two or three hidden-layers because both got the same best-validated accuracy metrics by 800 units.

3.4.2 Predicting the lowest price of the next day using RNN

Now, we apply again the input data presented in Table 1 on the SimpleRNN model with various combinations of parameters as delineated in Table 2. The goal is to optimize the SimpleRNN model for predicting the lowest price of the next day with the most promising and effective parameters as explained in *Taining RNN model and optimizing hyper-parameters* section.

From Sigmoid's activation function and hidden-layers 1 and 2, the results follow:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Sigmoid	1	15	2.47	7.03	1.36	2.23
Sigmoid	1	50	2.12	7.39	1.26	2.33
Sigmoid	1	100	1.95	7.92	1.16	2.33
Sigmoid	1	150	1.76	7.57	1.04	2.21
Sigmoid	1	200	1.48	6.00	0.91	1.93
Sigmoid	1	400	0.73	2.44	0.59	1.29
Sigmoid	1	600	0.49	0.97	0.47	0.81
Sigmoid	1	800	0.40	0.61	0.42	0.61
Sigmoid	1	1000	0.37	0.49	0.40	0.53
Sigmoid	**1**	**1200**	**0.36**	**0.46**	**0.40**	**0.51**
Sigmoid	2	15	2.15	7.37	1.28	2.37
Sigmoid	2	50	0.54	1.35	0.50	0.96
Sigmoid	2	100	0.39	0.60	0.42	0.59
Sigmoid	2	150	0.39	0.64	0.43	0.61
Sigmoid	2	200	0.40	0.67	0.43	0.62
Sigmoid	2	400	0.42	0.63	0.42	0.60
Sigmoid	**2**	**600**	**0.36**	**0.44**	**0.40**	**0.50**
Sigmoid	2	800	0.38	0.42	0.42	0.51
Sigmoid	2	1000	0.43	0.44	0.45	0.53
Sigmoid	2	1200	0.48	0.47	0.50	0.57

Table 6: Parameters' combinations for predicting the next day's lowest price.

Table 6 demonstrates that the Testing MSE and MAE achieve their best-validated accuracy values of 0.46 and 0.51 for one hidden-layer and 0.44 and 0.50 for two hidden-layers. As occurred before, the metrics for one hidden-layer have not hit the lowest by 1200 units. The convergence to the minimum would happen if the number of units or number of epoch increases, but it would increase the computations' cost. In this report, we restricted these numbers, but for future experiments, it would be relevant to investigate these topics.

Now, we perform Tanh's activation function for hidden-layers 1 and 2, and the outcomes go:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Tanh	1	15	2.15	7.37	1.28	2.37
Tanh	1	50	1.99	7.38	1.18	2.25
Tanh	1	100	1.46	5.87	0.90	1.92
Tanh	1	150	0.98	3.76	0.69	1.56
Tanh	1	200	0.72	2.30	0.58	1.25
Tanh	1	400	0.40	0.60	0.42	0.60
Tanh	**1**	**600**	**0.36**	**0.45**	**0.40**	**0.50**
Tanh	1	800	0.38	0.42	0.42	0.51
Tanh	1	1000	0.43	0.44	0.45	0.53
Tanh	1	1200	0.48	0.47	0.50	0.57
Tanh	2	15	2.15	7.37	1.28	2.37
Tanh	2	50	0.40	0.69	0.43	0.64
Tanh	2	100	0.40	0.77	0.43	0.67
Tanh	2	150	0.46	0.90	0.45	0.73
Tanh	2	200	0.47	1.00	0.46	0.79
Tanh	2	400	0.45	0.83	0.44	0.67
Tanh	**2**	**600**	**0.36**	**0.45**	**0.41**	**0.50**
Tanh	2	800	0.38	0.42	0.43	0.51
Tanh	2	1000	0.43	0.44	0.45	0.53
Tanh	2	1200	0.48	0.47	0.50	0.57

Table 7: Parameters' combinations for predicting the next day's lowest price.

Table 7 presents that the Testing MSE and MAE reach their best-validated accuracy rates of 0.45 and 0.50 for one hidden-layer and 0.45 and 0.50 for two hidden-layers. Both scores that happened at units equals 600, evidence that one or two-hidden layers have not much difference in terms of

learning and predicting. Thus, we would choose one-hidden layer as the best model because it is costly less-expensive.

Lastly, ReLu's activation function is run for two or three hidden-layers:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
ReLu	2	15	1.99	7.38	1.18	2.25
ReLu	2	50	0.97	3.76	0.69	1.56
ReLu	2	100	1.46	5.87	0.90	1.92
ReLu	2	150	0.97	3.76	0.69	1.56
ReLu	2	200	0.72	2.30	0.58	1.25
ReLu	2	400	0.40	0.60	0.42	0.60
ReLu	**2**	**600**	**0.36**	**0.45**	**0.40**	**0.50**
ReLu	2	800	0.38	0.42	0.42	0.51
ReLu	2	1000	0.43	0.44	0.45	0.53
ReLu	2	1200	0.48	0.47	0.50	0.57
ReLu	3	15	2.15	7.37	1.28	2.37
ReLu	3	50	0.40	0.69	0.43	0.64
ReLu	3	100	0.40	0.77	0.44	0.67
ReLu	3	150	0.43	0.90	0.45	0.74
ReLu	3	200	0.47	1.01	0.47	0.78
ReLu	3	400	0.45	0.82	0.44	0.67
ReLu	**3**	**600**	**0.36**	**0.45**	**0.40**	**0.50**
ReLu	3	800	0.38	0.42	0.43	0.51
ReLu	3	1000	0.43	0.44	0.45	0.53
ReLu	3	1200	0.48	0.47	0.51	0.57

Table 8: Parameters' combinations for predicting the next day's lowest price.

Table 8 reports that the Testing MSE and MAE accomplish their best-validated accuracy of 0.45 and 0.50 for two hidden-layer and 0.45 and 0.50 for three hidden-layers. There is no relevant difference between training ReLu with two or three hidden-layers because both got the same best-validated accuracy metrics by 600 units.

3.5 Training LSTM model and optimizing hyper-parameters

Here, we will feed and fit a LSTM model from Keras with our features and targets using the *Time Series Cross-Validation* discussed before. A combination of parameters is applied with different numbers for units, several activation functions, and various numbers of hidden-layers, avoiding costly computations, as follow:

Hyper-parameters	
Name	Values
Act. Functions	Hard-Sigmoid, Tanh, ReLu;
Hidden-layers	For Hard-Sigmoid and Tanh: 1, 2; For ReLu: 2, 3;
Rec. Act. Funct.	Tanh;
Units	50, 100, 150, 200, 400, 600, 800, 1000, 1200;
Epoch	50;
Mini-batch	1;
Loss-Function	Mean Squared Error (MSE);

Table 9: Hyper-parameters used for optimization of the LSTM model.

Our aim is to find the most promising model (lowest testing MSE and MAE) with the most efficient parameters (lowest number of units as function of hidden layers and activation function). For that, we need to investigate which parameter (activation function, hidden-layers, and units)

combination is the best. Further, the experiments will be based on the loss-curve, the number of epochs and any other possible way of optimization.

The test file LSTM.py stored in our GitHub repository replicates all the following results in the next sub-sections.

3.5.1 Predicting the highest price of the next day using LSTM

We keep exploring the input data presented in Table 1, but now on the LSTM model with various combinations of parameters as outlined in Table 9. The purpose is to optimize the LSTM model for predicting the highest price of the next day with the most promising and effective parameters as disclosed in *Training LSTM model and optimizing hyper-parameters* section.

The followed results contain the training and validation results with Hard-Sigmoid's activation function and hidden-layers 1 and 2:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Hard-Sigmoid	1	50	3.38	8.21	1.44	2.53
Hard-Sigmoid	1	100	2.08	7.85	1.26	2.46
Hard-Sigmoid	1	150	2.05	7.92	1.24	2.42
Hard-Sigmoid	1	200	2.02	7.97	1.21	2.39
Hard-Sigmoid	1	400	1.74	7.49	1.03	2.18
Hard-Sigmoid	1	600	1.30	5.88	0.80	1.91
Hard-Sigmoid	1	800	0.99	4.33	0.67	1.66
Hard-Sigmoid	1	1000	0.77	3.04	0.59	1.40
Hard-Sigmoid	**1**	**1200**	**0.65**	**2.19**	**0.54**	**1.19**
Hard-Sigmoid	2	50	1.81	7.57	1.08	2.25
Hard-Sigmoid	2	100	0.54	1.51	0.49	0.95
Hard-Sigmoid	2	150	0.42	0.94	0.43	0.68
Hard-Sigmoid	2	200	0.40	0.84	0.42	0.62
Hard-Sigmoid	2	400	0.38	0.73	0.41	0.57
Hard-Sigmoid	2	600	0.38	0.66	0.40	0.53
Hard-Sigmoid	2	800	0.37	0.61	0.40	0.51
Hard-Sigmoid	**2**	**1000**	**0.36**	**0.58**	**0.39**	**0.50**
Hard-Sigmoid	2	1200	0.36	0.62	0.40	0.52

Table 10: Parameters' combinations for predicting the next day's highest price.

Table 10 registers that the Testing MSE and MAE achieve their best-validated precision of 2.19 and 1.19 for one hidden-layer and 1200 units, and 0.58 and 0.50 for two hidden-layers and 1000 units. Both best scenarios appear not to have converged to their minimum validated score, and a higher number of units and epochs would be relevant for futures tests.

Note that the LSTM model appears to need more units or epochs to reach better scores than the RNN model, and might be the reason that a higher number of units is necessary for converging to its minimum accuracy.

Tanh's activation function for one and two hidden-layers is implemented, and the outcomes go:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Tanh	1	50	2.02	7.98	1.21	2.40
Tanh	1	100	1.74	7.49	1.03	2.19
Tanh	1	150	1.30	5.85	0.80	1.90
Tanh	1	200	0.97	4.23	0.66	1.64
Tanh	1	400	0.50	1.25	0.47	0.85
Tanh	1	600	0.39	0.71	0.41	0.55
Tanh	1	800	0.37	0.55	0.41	0.48
Tanh	**1**	**1000**	**0.38**	**0.49**	**0.42**	**0.45**
Tanh	1	1200	0.41	0.48	0.45	0.46
Tanh	2	50	0.98	3.78	0.70	1.58
Tanh	2	100	0.43	1.02	0.44	0.72
Tanh	2	150	0.41	0.90	0.43	0.66
Tanh	2	200	0.40	0.86	0.42	0.63
Tanh	2	400	0.39	0.77	0.41	0.59
Tanh	2	600	0.38	0.68	0.40	0.54
Tanh	**2**	**800**	**0.39**	**0.61**	**0.41**	**0.53**
Tanh	2	1000	0.43	0.97	0.43	0.63
Tanh	2	1200	0.43	0.83	0.43	0.57

Table 11: Parameters' combinations for predicting the next day's highest price.

Table 11 presents that the Testing MSE and MAE get their best-validated accuracy of 0.49 and 0.45 for one hidden-layer and 1000 units, and 0.61 and 0.53 for two hidden-layers and 800 units. The first scenario with one hidden-layer and 1000 units was the most promising (better-validated metrics) and efficient (less time for training and predicting).

Subsequently, ReLu's activation function is run for two and three hidden-layers 2:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
ReLu	2	50	2.10	7.72	1.28	2.46
ReLu	2	100	2.08	7.90	1.26	2.47
ReLu	2	150	2.06	7.93	1.24	2.43
ReLu	2	200	2.01	8.05	1.21	2.41
ReLu	2	400	1.74	7.51	1.03	2.19
ReLu	2	600	1.31	5.86	0.81	1.90
ReLu	2	800	0.99	4.30	0.67	1.65
ReLu	2	1000	0.77	3.05	0.59	1.40
ReLu	**2**	**1200**	**0.66**	**2.26**	**0.54**	**1.21**

Table 12: Parameters' combinations for predicting the next day's highest price.

Table 12 displays that the Testing MSE and MAE obtain their best-validated accuracy values of 2.26 and 1.21 for one hidden-layer and 1200 units. Again, it seem not to have converged to its minimum validated score, and a higher number of units and epochs would be suitable for futures tests.

3.5.2 Predicting the lowest price of the next day using LSTM

The input data presented in Table 1 is performed on the LSTM model with various combinations of parameters as delineated in Table 9. The aim is to optimize the LSTM model for predicting the lowest price of the next day with the most promising and effective parameters as explained in *Training LSTM model and optimizing hyper-parameters* section.

The results for Hard-Sigmoid's activation function with one and two hidden-layers follow:

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Hard-Sigmoid	1	50	2.19	7.52	1.29	2.39
Hard-Sigmoid	1	100	2.14	7.40	1.27	2.36
Hard-Sigmoid	1	150	2.11	7.41	1.26	2.33
Hard-Sigmoid	1	200	2.06	7.44	1.23	2.31
Hard-Sigmoid	1	400	1.75	6.81	1.05	2.08
Hard-Sigmoid	1	600	1.31	5.29	0.83	1.82
Hard-Sigmoid	1	800	0.97	3.64	0.69	1.54
Hard-Sigmoid	1	1000	0.76	2.58	0.60	1.31
Hard-Sigmoid	**1**	**1200**	**0.65**	**1.84**	**0.55**	**1.12**
Hard-Sigmoid	2	50	1.71	6.34	1.05	2.11
Hard-Sigmoid	2	100	0.51	1.19	0.49	0.89
Hard-Sigmoid	2	150	0.40	0.62	0.42	0.60
Hard-Sigmoid	2	200	0.38	0.56	0.41	0.57
Hard-Sigmoid	2	400	0.37	0.50	0.40	0.53
Hard-Sigmoid	2	600	0.36	0.48	0.40	0.52
Hard-Sigmoid	**2**	**800**	**0.36**	**0.45**	**0.39**	**0.51**
Hard-Sigmoid	2	1000	0.36	0.46	0.39	0.52
Hard-Sigmoid	2	1200	0.35	0.45	0.39	0.52

Table 13: Parameters' combinations for predicting the next day's lowest price.

Table 13 lists that the Testing MSE and MAE achieve their best-validated precision of 1.84 and 1.12 for one hidden-layer and 1200 units, and 0.45 and 0.51 for two hidden-layers and 800 units. Both best scenarios appear not to have converged to their minimum validated score, and a higher number of units and epochs would be relevant for futures tests.

Act. Funct.	H-layers	Units	Train MSE	Test MSE	Train MAE	Test MAE
Tanh	1	50	2.06	7.44	1.23	2.30
Tanh	1	100	1.75	6.84	1.05	2.08
Tanh	1	150	1.29	5.22	0.82	1.81
Tanh	1	200	0.96	3.68	0.69	1.55
Tanh	1	400	0.48	0.92	0.47	0.78
Tanh	1	600	0.38	0.53	0.40	0.56
Tanh	**1**	**800**	**0.36**	**0.45**	**0.40**	**0.50**
Tanh	1	1000	0.37	0.42	0.42	0.50
Tanh	1	1200	0.40	0.43	0.44	0.52
Tanh	2	50	0.81	2.54	0.63	1.23
Tanh	2	100	0.41	0.77	0.44	0.67
Tanh	2	150	0.40	0.66	0.43	0.63
Tanh	2	200	0.39	0.61	0.43	0.59
Tanh	2	400	0.40	0.65	0.42	0.63
Tanh	**2**	**600**	**0.39**	**0.56**	**0.41**	**0.58**
Tanh	2	800	0.40	0.54	0.41	0.58
Tanh	2	1000	0.42	0.75	0.42	0.63
Tanh	2	1200	0.42	0.57	0.42	0.58

Table 14: Parameters' combinations for predicting the next day's lowest price.

Table 14 shows that the Testing MSE and MAE achieve their best-validated precision of 0.45 and 0.50 for one hidden-layer and 800 units, and 0.56 and 0.58 for two hidden-layers and 600 units. However, these accuracies are not better than the ones achieved from RNN training and validation with same parameters and activation function.

3.6 Our optimal model

After an exhaustive investigation, we can now select the best model and parameters for further analysis.

We chose the SimpleRNN model for further experiments because it got better-validated accuracy metrics than LSTM. Besides, SimpleRNN model had lower time for fitting and training than LSTM. As stated before, we were looking for the most promising and effective model and the SimpleRNN demonstrated to be the prefect match in this analysis.

Hence, the best parameters and metrics for each activation function and hidden-layers were gotten from *Table 3*, *Table 4*, *Table 5* for predicting the next day's highest, and *Table 6*, *Table 7*, *Table 8* for predicting the next day's lowest.

The comparative plots follow:

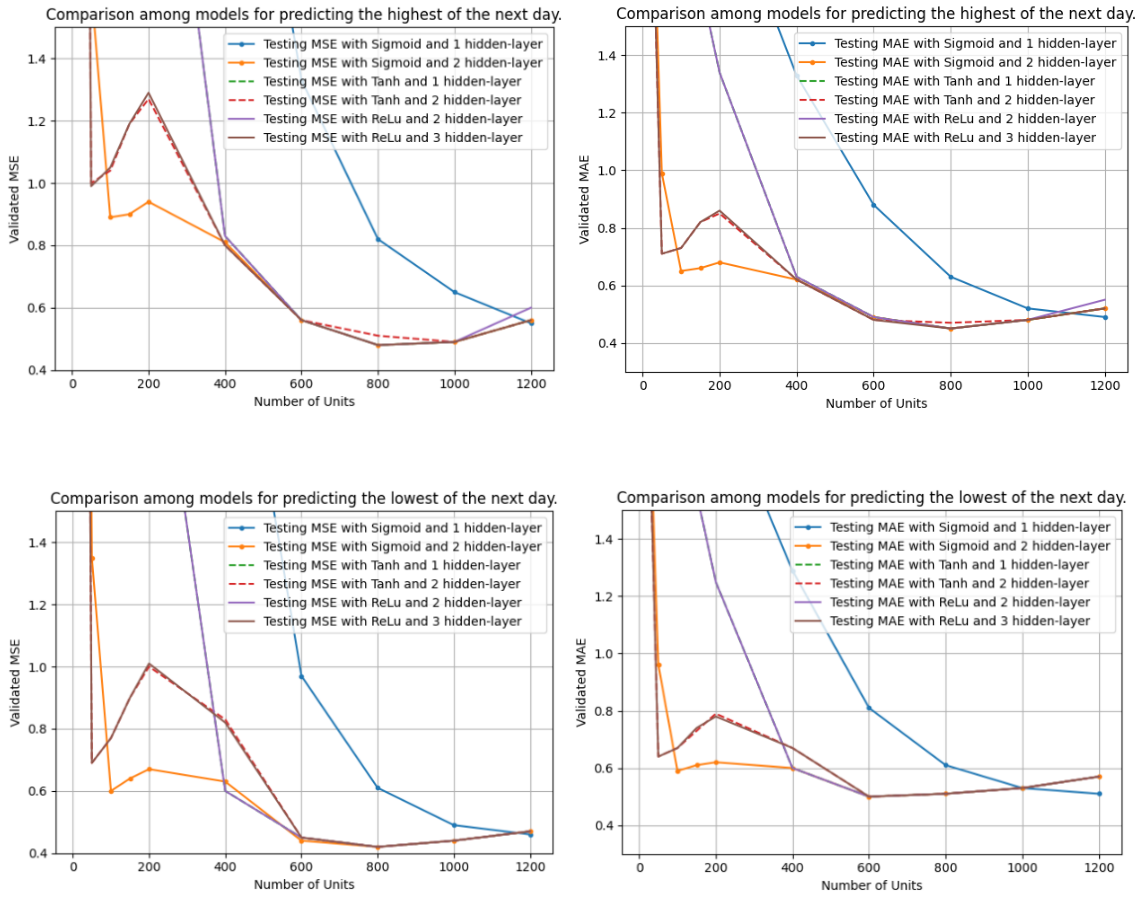

Figure 7: Comparison between all validated MSE (left) and MAE (right) when predicting the highest (top) and lowest (bottom) prices of the next day.

The activation function in which the best testing MSE and MAE scores converged to their minimum was Tanh. Besides, Tanh got the lowest scores with the lowest number of hidden-layers (1) and units (800), confirming to be the most promising and efficient model among the others. Therefore, the summary of parameters for this best model is:

Target	Act.Funct.	H-Layers	Units	Epochs	Mini-batch	Loss-Funct.	Test MSE	Test MAE
Highest	Tanh	1	800	50	1	MSE	0.48	0.45
Lowest	Tanh	1	800	50	1	MSE	0.42	0.51

Table 15: Summary of parameters for our best model.

It would be interesting to visualize the *Train x Test* plots in order to analyse the bias and variance of the learning process:

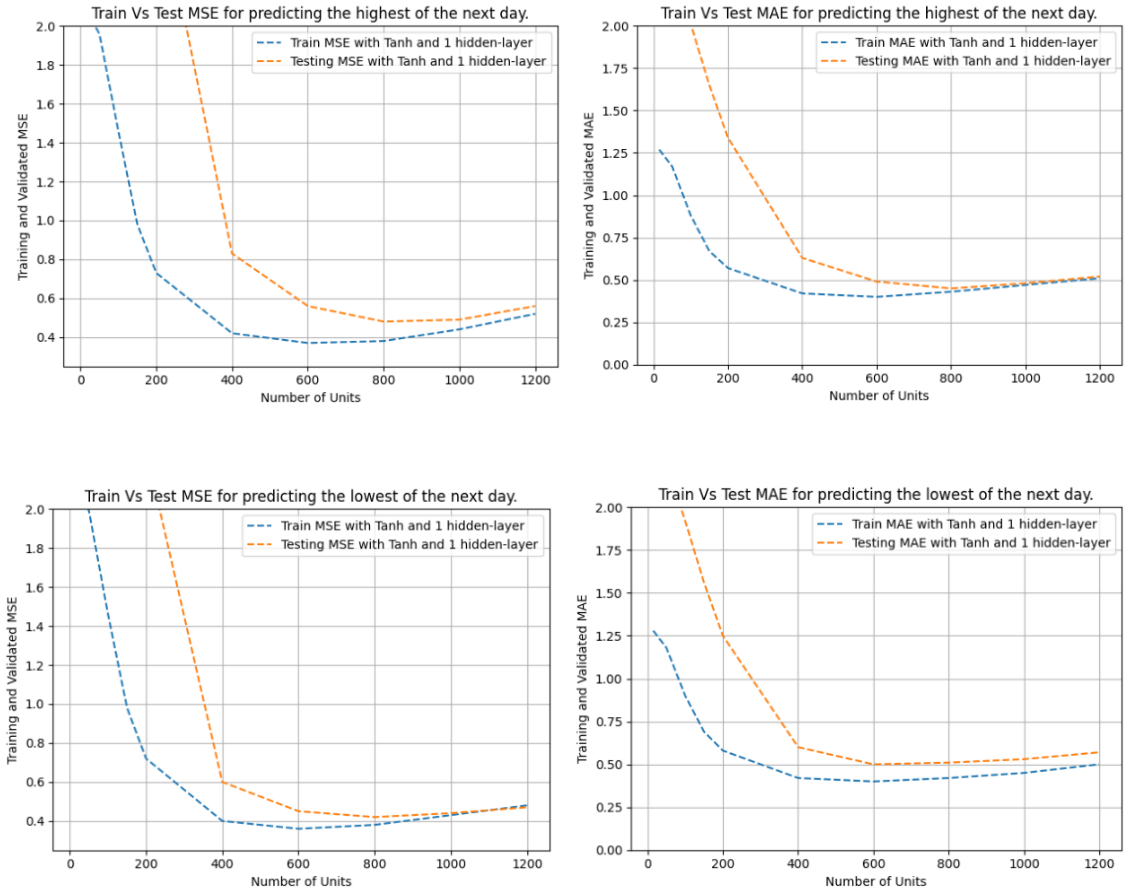

Figure 8: Train Vs Test MSE (left) and MAE (right) plots for the most promising and efficient model when predicting the highest (top) lowest (bottom) prices of the next day.

As expected, as the number of units increases, the metrics become more reliable until a certain point and then worsen again. We can visualize the same pattern in all plots, where the biases are very high for units before 200. As long the units grow around 800, it seems to reach the optimal point with very low biases and variance. After 800 units, we can notice that the metrics worsen, manifesting apparent over-fitting of the model, thus increasing the biases and variances.

Next, we need to check if the number of epochs can improve the model's metrics or efficiency. For that, we run a combination of epochs as a function of the optimized parameters:

Target	Epochs	Train MSE	Test MSE	Train MAE	Test MAE
Highest	**10**	**0.3884**	**0.4856**	**0.4364**	**0.4559**
Highest	30	0.3884	0.4856	0.4364	0.4559
Highest	50	0.3884	0.4856	0.4364	0.4559
Highest	80	0.3884	0.4856	0.4364	0.4559
Lowest	**10**	**0.3808**	**0.4278**	**0.4284**	**0.5114**
Lowest	30	0.3808	0.4278	0.4284	0.5114
Lowest	50	0.3808	0.4278	0.4284	0.5114
Lowest	80	0.3808	0.4278	0.4284	0.5114

Table 16: RNN model with parameters' combinations for predicting the next day's highest/lowest prices as a function of epochs.

Remark that the metrics are precisely the same for 10, 30, 50, or 80 epochs, confirming that the SimpleRNN model converged to its minimum at ten epochs or before. Hence, we have improved our model's efficiency by decreasing from 50 to 10 epochs in the training process, and the final summary of parameters for our best model is:

Target	Act.Funct.	H-Layers	Units	Epochs	Mini-batch	Loss-Funct.	Test MSE	Test MAE
Highest	Tanh	1	800	10	1	MSE	0.48	0.45
Lowest	Tanh	1	800	10	1	MSE	0.42	0.51

Table 17: Final summary of parameters for our best model.

3.7 Applying the predictions to an algorithm trading strategy

Merely measuring the accuracy of our RNN does not provide a reliable source for helping investors make decisions. Indeed, the predictions from RNN can be incredibly promising, but sometimes not applicable in the real world. Therefore, a better way to assess our model's performance is through back-testing real-data, in other words, using the RNN's outputs as inputs to a trading system.

As stated before, we are interested in finding an application for the RNN's outputs translating into algorithm day-trading decision-making process. For that, we will follow the stock market in real-time, taking into account fixed intervals of 15 minutes, that is, every 15 minutes, the system consults our trade rules and recommends entering or exiting from the market.

As bench-mark, we will apply a typical trading system using Bollinger Bands. This traditional trading system consists of entering or exiting the market when the day-session prices' oscillation reaches the five-day SMA Bollinger Upper or Lower-Band. It is vital to remark that more than one trade is expected but only allowed if they are alternated, that is, the system can not buy or sell twice in a row. Besides, since this is a day-trading system, all opened trades will be closed by the end of the last 15 minutes of the market session. In summary, the rules of the Bollinger Bands trading system are:

- Buy when the 15 min session's highest price is larger than the 5day-SMA-Upper-Band;
- Sell when the 15 min session's lowest price is lower than the 5day-SMA-Lower-Band;
- Close all positions by the end of the last 15 min session.
- It is allowed only one trade per time in each direction.

The python script named application.py, in our GitHub directory Project3/, performs the back-testing of this strategy from 14.10.2020 until 03.12.2020, and the results follow:

Strategy:	5-days Bollinger Bands
Financial security:	PETR4
The period analyzed:	from 14.10.2020 to 03.12.2020
Number of trades:	04
Cumulative daily return:	BRL1,95 per share
Cumulative daily percentage:	approx. 7 percent of a share

Table 18: Results for 5-day Bollinger Bands trading strategy.

Now, it is time to introduce or trading strategy. From our best RNN model, we have predicted the next day's highest and lowest prices. Therefore, we use these predicted highest and lowest during the day-trading session for algorithm trading decision-making, likely the 5-day-SMA Upper and Lower Bollinger bands. In summary, the rules of our trading system are:

- Buy when the 15 min session's highest price is larger than the predicted highest for the day;
- Sell when the 15 min session's lowest price is lower than the predicted lowest for the day;
- Close all positions by the end of the last 15 min session.
- It is allowed only one trade per time in each direction.

The python script named application.py, in our GitHub directory Project3/, performs the back-testing of this strategy from 14.10.2020 until 03.12.2020, and the results follow:

Strategy:	5-days Bollinger Bands
Financial security:	PETR4
The period analyzed:	from 14.10.2020 to 03.12.2020
Number of trades:	35
Cumulative daily return:	BRL10,76 per share
Cumulative daily percentage:	approx. 39 percent of a share

Table 19: Results for our trading strategy.

One can see that our trading system overcomes the profitability of the traditional trading system, confirming that the combination of RNN and algorithm trading is a promising and reliable tool for decision-making process in stock market trading.

4 Conclusions and Future Work

This project introduces a day-trading algorithm tool for the decision-making process that uses the outputs from RNN and LSTM. The ANNs sub-classes predict the next day's highest and lowest prices as outputs and then use them as inputs for a day-trading algorithm system. The metrics used for assessing the predictions were MSE and MAE values. It is essential to say that the data's scaling is not applied because we want to visualize the numbers (as it is) and check possible improvements. Regarding our suggested trading system, it uses specific rules as triggers for entering and exiting the market. Back-tests are performed on PETR4 stock data from B3 Sao Paulo's stock exchange, which has the highest liquidity flow (financial volume) in the Brazilian market. Results are compared with a traditional trading system using a TA instrument called Bollinger Bands, our bench-mark.

Our trading system's best result (BRL10.76 per share of cumulative profit) operated approx. five times as better as the bench-mark (BRL1.95 per share of cumulative profit), confirming that ANN usage, especially the RNN, has outstanding profitability and great advance against traditional techniques. Indeed, the targets' validated predictions get accuracy metrics as low as 0.5, which are very good, using both RNN and LSTM. Although LSTM is extremely expensive for training computations and might not be adequate for high-speed markets nowadays.

As a future research objective, it would be relevant to explore other ANN techniques like Gated Recurrent Unit (GRU), as well as many additional TA indicators as features for training the ANN. Besides, scaling data, a larger number of units, hidden-layers, epochs, and mini-batch could be considered in the future, but it will significantly increase the computation costs avoided in this project. Moreover, the quantity of historical data could also be examined to have a long-term image of performance and profits. Other securities could similarly be experimented with for checking the strategy's feasibility in different types of stocks, markets, countries, and others. Finally, trading policies for risk management and other tools for avoiding order issues like slippages could be analyzed in future works as additional protection against turnovers or unexpected market situations.

5 Bibliography

[1] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning data mining, inference, and prediction*; 2nd edition; Springer; New York, USA. 2017.

[2] Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning*; 2nd edition; Packt; Birmingham, UK. 2017.

[3] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*; O'Reilly Media; Sebastopol, CA. 2017.

[4] Lopez de Prado, Marcos. *Advances in Financial Machine Learning*; Wiley; New Jersey; 2018.

[5] *From an Artificial Neural Network to a Stock Market Day-Trading System: A Case Study on the BMF BOVESPA*; Leonardo C. Martinez, Diego N. da Hora, Joao R. de M. Palotti, Wagner Meira Jr. and Gisele L. Pappa (2009).

[6] *Portfolio Optimization-Based Stock Prediction Using Long-Short Term Memory Network in Quantitative Trading*; Van-Dai Ta, Chuan-Ming Liu, Direselign Addis Tadesse (2020).

[7] University of Oslo. *Week 42 Convolutional (CNN) and Recurrent (RNN) Neural Networks.* Department of Physics, University of Oslo. https://compphysics.github.io/MachineLearning/doc/pub/week42/html/week42.html. (Oct 17, 2020)

[8] B3 - Sao Paulo's stock exchange. http://www.b3.com.br/en_us/market-data-and-indices/data-services/market-data/quotes/quotes/

[9] Hyndman, R.J., and Athanasopoulos, G. (2018) Forecasting: principles and practice, 2nd edition, OTexts: Melbourne, Australia. otexts.com/fpp2/. Accessed on 07.12.2020.