Final Project: Machine Learning algorithm for Stock value prediction

CS 584 - Machine Learning - Fall 2022

Final Project Report

1. Introduction

1.1. Team members

The team in charge of the development of the project will be made up of two members. On the one hand, ***Julen Ferro Bañales*** CWID:A20512110, currently studying the M.Sc. in Computational Decision Science & Operations Research. On the other hand, ***Eneko Gonzalez*** CWID: A20520157, currently studying the M.Eng. Computer Science.

1.2. Description of the problem

This project will deal with a really well-known problem in Finance called **'Stock prediction'**. Nowadays, the business environment is quite unstable due to different types of crisis that the world is suffering and the new context that has been given by the globalization. Therefore, the enterprises are forced to find different methods to fight against this uncertainty, in which there are not so used to work compared to the previous ***'blue oceans'*** of economic competitiveness in which they have lived so far.

1.3. Project scope

This project will try to develop a tool valid for stock prices analysis and prediction. This way, the software that will be developed will try to train an **Artificial Intelligence** algorithm which tries to give an stock price output, based on past stock prices inputs. The insights given by the output of the software may be used as guidance for the consultants of the finance enterprises, due to the fact that the information got by the software cannot be hundred percent taken for granted, as the algorithm's output will be a prediction or an approximation, not an exact event to occur. Therefore, as in nowadays world nothing could be assured, it is important to make clear that the output of this model should only serve as some kind of insight of guidance to be used by any expert in the financial industry.

In addition to this, different algorithms will be compared, some of which will use ***Machine Learning Optimization*** techniques in order to learn from the data, such as ***gradient descent*** for example. Some other algorithms will be based on simple statistical regression models, which will give results to be compared to the other ***Machine Learning*** algorithms' results, in order to make clear whether if the ***Machine Learning*** techniques are as accurate and fast as they seem, or if traditional statistical techniques still work faster and more accurately than the new ones.

1.4. Timeline & Milestones

In order to achieve the goals and scope of the project, the project has been segmented into different development phases. This phases cover the complete development of the project, from the initial research to the final project report.

Each of these phases have had a initial and final meeting to determine the goals and to evaluate the achievements. Also, in each of the iterations or also known as 'sprints' in the 'Agile' methodology, the team tried to achieve the goal of reaching the previously established milestone.

The different phases and milestones are shown below:

- Topic Research and project scope definition.
  Milestone: Initial Proposal Report (Oct 1, 2022)
- Definition of the algorithms and methodologies used for the achievements of the project goals.
  Milestone: Theoretical bases of the project. (Nov 1, 2022)
- Implementation of the different algorithms and actual development of the project.
  Milestone: Final Project Code (Nov 10, 2022)
- Summary of the obtained results and presentation of the project
  Milestone: Project presentation, 5-8 minutes (Nov 16, 2022)
- Analysis and documentation of the work done and accomplished achievements.
  Milestone: Final project report (Dec 2, 2022)

2. Proposed Solution

In order to reach the goals of the project, different models and algorithms have been trained to be able to find the solution that best fits the expected results. To do so, both ***mathematical algorithms*** and **Artificial Intelligence** models have been implemented.

The following section describes the theoretical base of each of them to allow the proper understanding of the potential and the fundamentals they follow.

2.1. Simple Moving Average model

In the first place, the ***Simple Moving Average (SMA)*** algorithm, which is the simplest one and the most intuitive one. It gets a prediction of the stock price based on the weighting that has been given to the previous stock prices. The importance or weighting of each the chosen stock price values will be the equal, and the window size concept will be defined as the N number of previous data samples taken into account for the average computing. [1] [**?**].

The predicted output will look like this equation.

$$SMA = \frac{P1 + P2 + ... + Pn}{N} \tag{1}$$

P1...Pn will be the values of the previous stock prices and N will be the window size. It must be taken into account that by varying the window size, more of the previous stock prices could be taken into account for the prediction calculation leading to a increase in the stiffness of the curve, therefore, losing its ability to fit into data defined by a high-variance curve.

2.2. Exponential Moving Average model

In addition to the ***Simple Moving Average (SMA)*** model, it also exists the ***Exponential Moving Average (EMA)***. This model is based on the same mathematics as the ***Simple Moving Average***, but it will be more fancy due to the fact that in this model, the closest samples to the recent dates are weighted to a greater extent. That is to say, the events that occurred long time ago, will gain importance in this model, compared to the ***Simple Moving Average*** model, which does not take into account data before the last data point within the window size. While in the ***Simple Moving Average*** model, only the previous N (window size) stock price values where taken into account at the same weight, in the ***Exponential Moving Average*** model, all the previous stock prices are taken into account but being weighted to a greater extent as the closer the data gets to the predicted element [2].

In this model, the formula that gives the predicted value of the model is the next one.

$$EMA = P_t * k + EMA_(t-1) * (1-k) \tag{2}$$

Where Pt is the price at time point t and N is the number of time points in the EMA.

The weighting factor is calculated as follows:

$$k = \frac{2}{N+1} \tag{3}$$

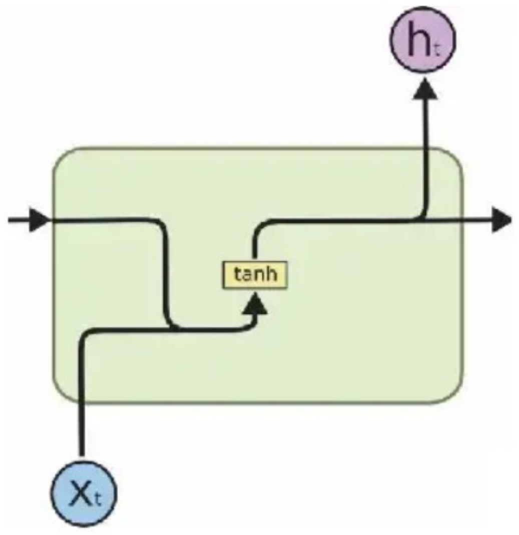
2.3. Simple RNN

The second implemented model uses the Keras Simple RNN layers for its implementation.

The Simple RNN is the most simple approach of the RNN idea. The Recurrent Neural Networks (RNN) are neural networks developed and used for sequential and time dependant data-sets. They basic idea is that instead of treating each of the given input as an independent sample of the data, they create relations between the new input and the last given output.

Concretely, the Simple RNN layer uses a simple multiplication of Input (xt) and Previous Output (ht-1). Passed through 'Tanh' activation function. Without using any gates.

This approach can be better understand in the following representation:

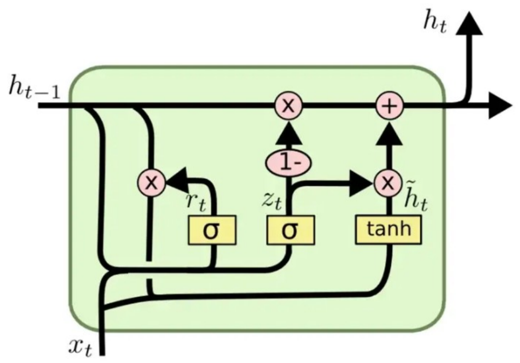


2.4. Gated Recurrent Unit (GRU)

In addition to the Simple RNN layer, the Gated Recurrent Units (GRU) has been implemented too. This RNN layer increases a bit the complexity and usually allows better performance.

To do so, the GRU includes an update gate. In this way, it gains the possibility to decide whether to pass the last output to the new input or no.

It can be represented as follows:



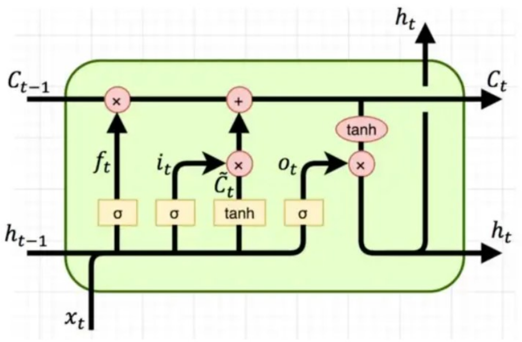
Also, for this particular case, the equations are the following ones:

$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)\\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)\\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)\\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

2.5. Long Short Term Memory (LSTM)

The third of the models implemented makes use of the Long Sort Term Memory (LSTM) Neural Networks. This type of layer adds one step more in complexity compare to the previous one.

In this case, more gates are included, a forget and an output gate. These gates help to decide whether a value is used of not for the generated relations between the inputs and outputs. Additionally, in this case, apart from the tanh activation function the sigmoid is used too.



The equations for the LSTM are as follows:

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)\\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)\\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)\\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t\\
o_t &= \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)\\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

3. Data-set & Libraries

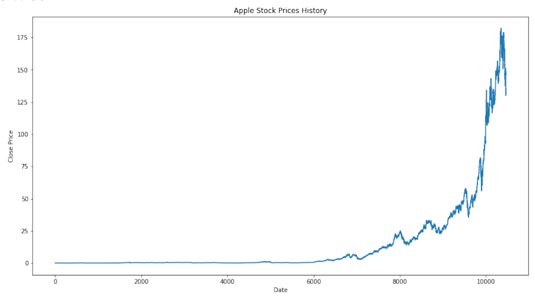
First of all, the data that has been used for the project must be described. The data set has been imported from Kaggle and it is comprised of an evolution of the stock price of the well-known enterprise Apple from the year 1980 to the year 2020.

The data set in matter is composed of more or less ten thousands of rows, that is to say, ten thousands of samples. It is quite a lot of information but the fact that almost forty years of stock price evolution is covered must be taken into account.

As attributes, the data set has eight different attributes described as follows: Date (the day in which the stock price is taken), Open (the price at which the first trade is made), High(the maximum price at which a trade is made in that period of time), Low(the minimum price at which a trade is made in the given period of time), Close(the price at which the last stock is trade in the given period of time) and the Adj Close (the same price as close, but after having paid the dividends).

As the most important price of the stock is the ***'Close Price'***, the Machine Learning model will be trained and tested with the ***'Close Price'*** data.

The data can be represented as the following graph shows:



For the data preparation the following steps have been followed:

1. Clean data. Delete the unnecessary information of the data-set.

2. Split the data between Data and Labels. The inputs and the expected outputs of the model

3. Apply MinMax Scalar in range 0 to 1 to reduce the impact of the outliers in the whole data set and model.

4. Split the data into different subsets to allow training and testing.

Finally, to implement all the required methods and functionalities, different libraries have been implemented:

- Numpy
- Matplotlib

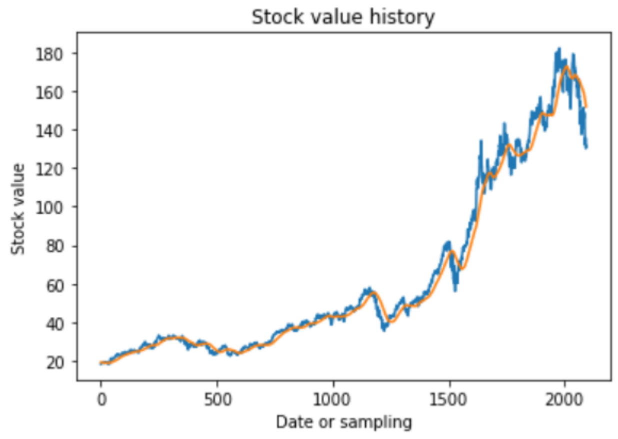
- Pandas
- Scikit-Learn
- Tensor Flow Keras

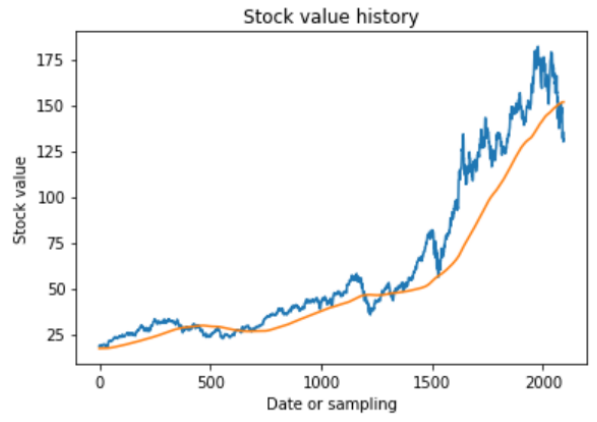
4. Results

After implementing the different models, different results have been obtained. The following section describes how the different results have been obtained by simulating each model by feeding the same data set.

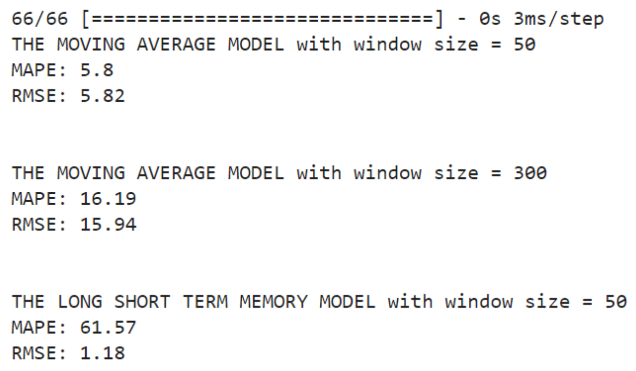
4.1. Simple Moving Average model

The simple average model has been implemented in a python program and has been simulated with the twenty percent of the data set. During this testing the next figures have been plotted respectively for the cases of window size = 50 and window size = 300.





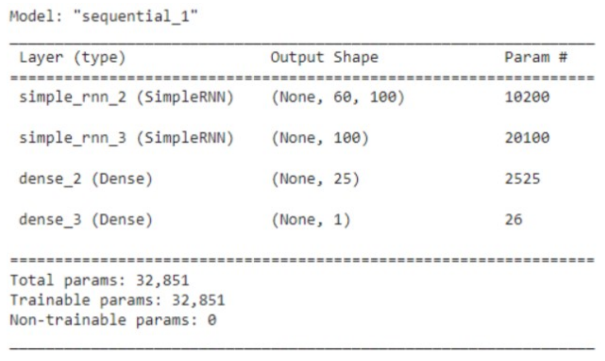
By comparing to each other, it can be seen that the Simple Moving Average model with the lowest window number has fitted better into the data than the one with a window size of 300. Hence, it can be seen that with a lower number of window size the model becomes more flexible and it adapts better to the high variance scenario of the stock prediction issue. Finally, large numbers of window sizes clearly make the prediction curve smoother because averages a longer array of numbers leading this way to a flat profile.



```
66/66 [==============================] - 0s 3ms/step
THE MOVING AVERAGE MODEL with window size = 50
MAPE: 5.8
RMSE: 5.82


THE MOVING AVERAGE MODEL with window size = 300
MAPE: 16.19
RMSE: 15.94


THE LONG SHORT TERM MEMORY MODEL with window size = 50
MAPE: 61.57
RMSE: 1.18
```

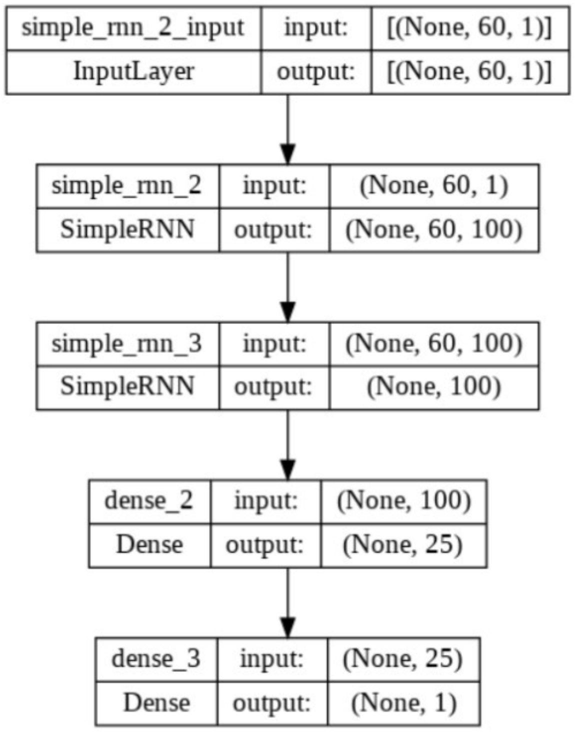
4.2. Simple RNN

The model generated for the Simple RNN layer implementation is the following one:



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 simple_rnn_2 (SimpleRNN)    (None, 60, 100)           10200

 simple_rnn_3 (SimpleRNN)    (None, 100)               20100

 dense_2 (Dense)             (None, 25)                2525

 dense_3 (Dense)             (None, 1)                 26

=================================================================
Total params: 32,851
Trainable params: 32,851
Non-trainable params: 0
_________________________________________________________________
```

This model can be graphically represented too with the following structure:

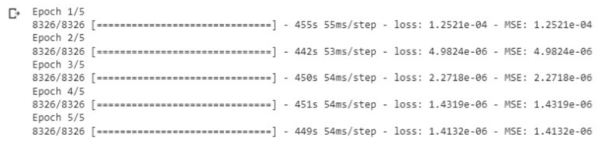

simple_rnn_2_input | input: | [(None, 60, 1)]
InputLayer | output: | [(None, 60, 1)]
simple_rnn_2 | input: | (None, 60, 1)
SimpleRNN | output: | (None, 60, 100)
simple_rnn_3 | input: | (None, 60, 100)
SimpleRNN | output: | (None, 100)
dense_2 | input: | (None, 100)
Dense | output: | (None, 25)
dense_3 | input: | (None, 25)
Dense | output: | (None, 1)

The model has been trained with the following hyper-parameters values:

- Optimizer: Adam
- Learning Rate:0.0001
- Loss: Mean Squared Error
- Epochs: 5
- Batch Size: 1

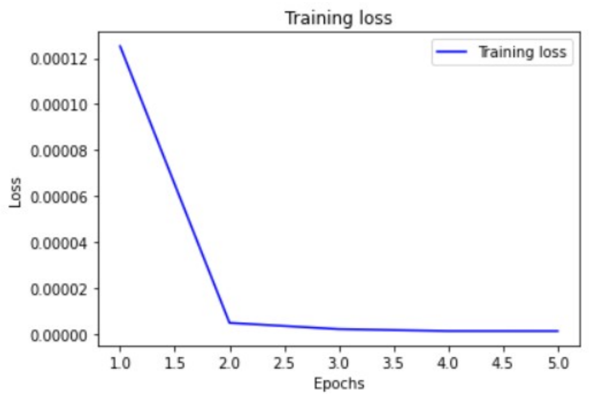
With all the conditions described above the obtained results are as follows.

Firstly, in the training process the model shows the following results for each of the epoch:

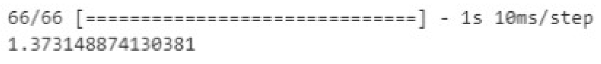


```
Epoch 1/5
8326/8326 [==============================] - 455s 55ms/step - loss: 1.2521e-04 - MSE: 1.2521e-04
Epoch 2/5
8326/8326 [==============================] - 442s 53ms/step - loss: 4.9824e-06 - MSE: 4.9824e-06
Epoch 3/5
8326/8326 [==============================] - 450s 54ms/step - loss: 2.2718e-06 - MSE: 2.2718e-06
Epoch 4/5
8326/8326 [==============================] - 451s 54ms/step - loss: 1.4319e-06 - MSE: 1.4319e-06
Epoch 5/5
8326/8326 [==============================] - 449s 54ms/step - loss: 1.4132e-06 - MSE: 1.4132e-06
```

With a training loss that can be represented as follows:

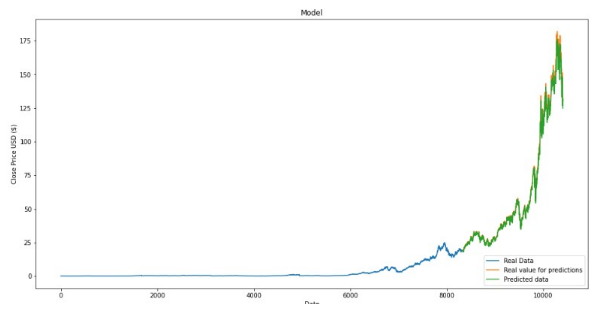


Finally, the model shows the following testing loss:



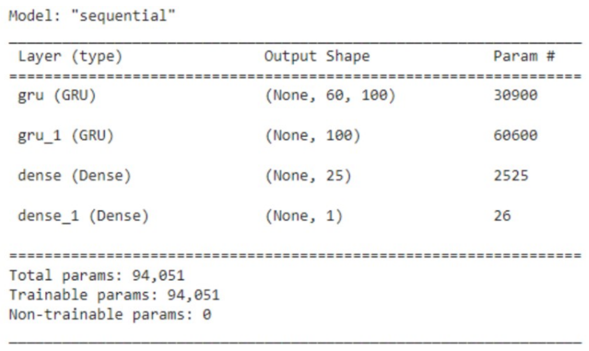
```
66/66 [==============================] - 1s 10ms/step
1.373148874130381
```

The final results can also be represented in the predicted plot:



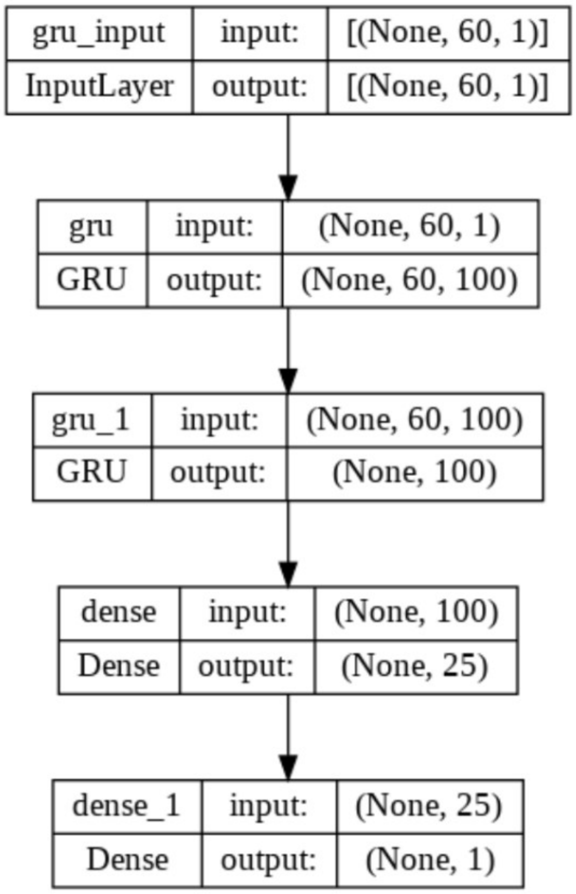
4.3. Gated Recurrent Unit (GRU)

The model generated for the Simple RNN layer implementation is the following one:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (None, 60, 100)           30900

 gru_1 (GRU)                 (None, 100)               60600

 dense (Dense)               (None, 25)                2525

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 94,051
Trainable params: 94,051
Non-trainable params: 0
_________________________________________________________________
```

This model can be graphically represented too with the following structure:

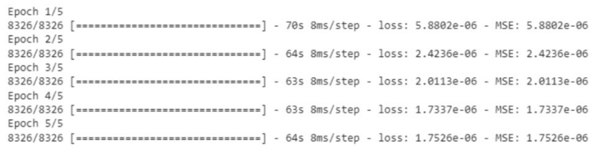


The model has been trained with the following hyper-parameters values:

- Optimizer: Adam
- Learning Rate:0.0001
- Loss: Mean Squared Error
- Epochs: 5
- Batch Size: 1

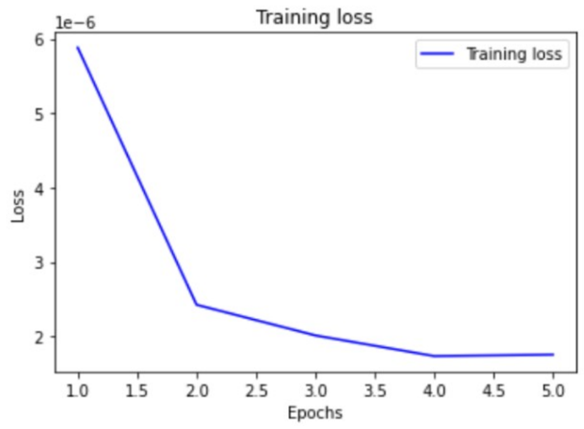
With all the conditions described above the obtained results are as follows.

Firstly, in the training process the model shows the following results for each of the epoch:

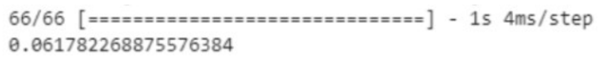


```
Epoch 1/5
8326/8326 [==============================] - 70s 8ms/step - loss: 5.8802e-06 - MSE: 5.8802e-06
Epoch 2/5
8326/8326 [==============================] - 64s 8ms/step - loss: 2.4236e-06 - MSE: 2.4236e-06
Epoch 3/5
8326/8326 [==============================] - 63s 8ms/step - loss: 2.0113e-06 - MSE: 2.0113e-06
Epoch 4/5
8326/8326 [==============================] - 63s 8ms/step - loss: 1.7337e-06 - MSE: 1.7337e-06
Epoch 5/5
8326/8326 [==============================] - 64s 8ms/step - loss: 1.7526e-06 - MSE: 1.7526e-06
```

With a training loss that can be represented as follows:

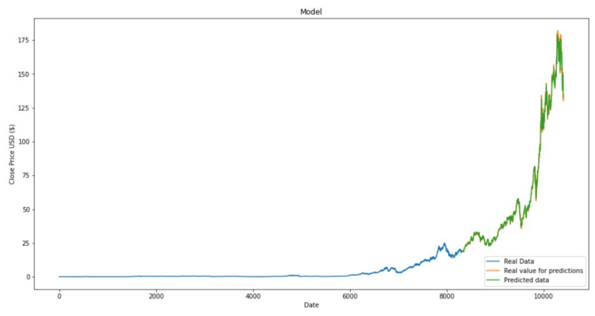


Finally, the model shows the following testing loss:



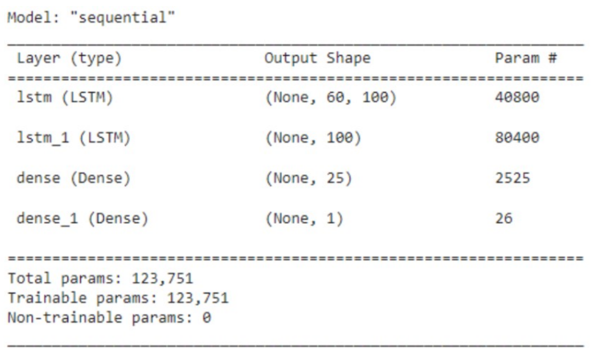
```
66/66 [==============================] - 1s 4ms/step
0.061782268875576384
```

The final results can also be represented in the predicted plot:



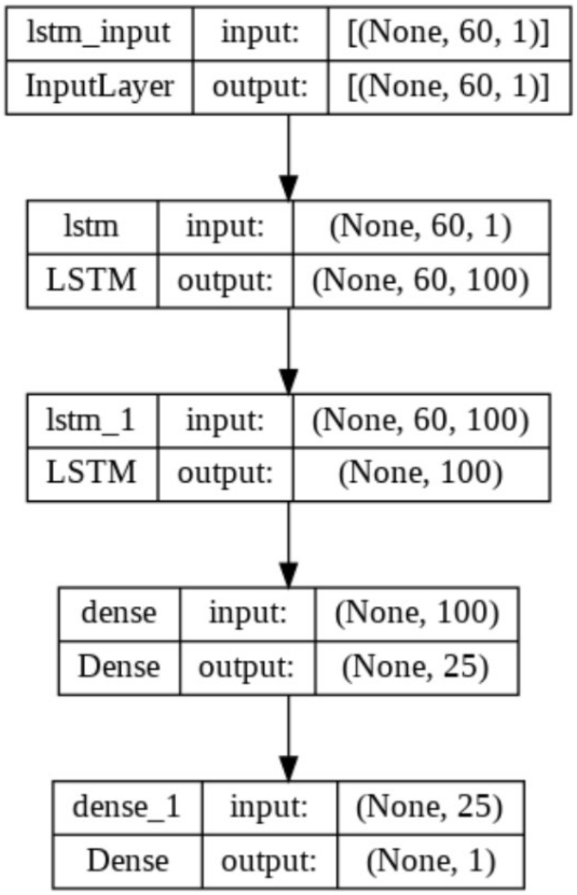
4.4. Long Short Term Memory (LSTM)

The model generated for the Simple RNN layer implementation is the following one:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 100)           40800

 lstm_1 (LSTM)               (None, 100)               80400

 dense (Dense)               (None, 25)                2525

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 123,751
Trainable params: 123,751
Non-trainable params: 0
_________________________________________________________________
```

This model can be graphically represented too with the following structure:

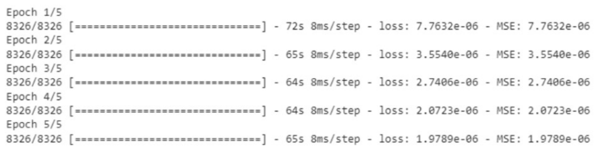

lstm_input | InputLayer | input: [(None, 60, 1)] | output: [(None, 60, 1)]
lstm | LSTM | input: (None, 60, 1) | output: (None, 60, 100)
lstm_1 | LSTM | input: (None, 60, 100) | output: (None, 100)
dense | Dense | input: (None, 100) | output: (None, 25)
dense_1 | Dense | input: (None, 25) | output: (None, 1)

The model has been trained with the following hyper-parameters values:

- Optimizer: Adam
- Learning Rate:0.0001
- Loss: Mean Squared Error
- Epochs: 5
- Batch Size: 1

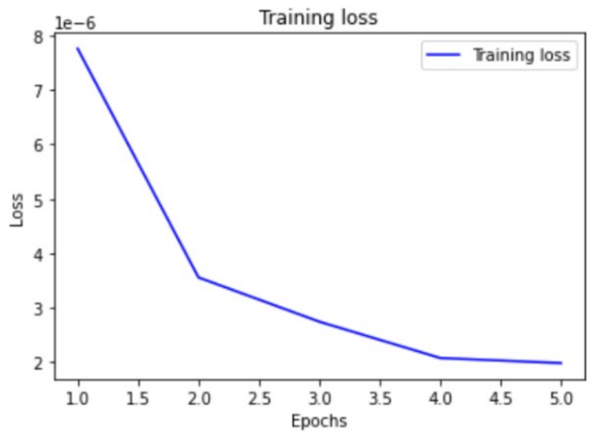
With all the conditions described above the obtained results are as follows.

Firstly, in the training process the model shows the following results for each of the epoch:

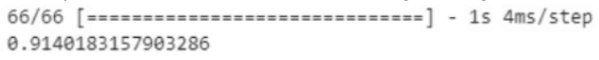


```
Epoch 1/5
8326/8326 [==============================] - 72s 8ms/step - loss: 7.7632e-06 - MSE: 7.7632e-06
Epoch 2/5
8326/8326 [==============================] - 65s 8ms/step - loss: 3.5540e-06 - MSE: 3.5540e-06
Epoch 3/5
8326/8326 [==============================] - 64s 8ms/step - loss: 2.7406e-06 - MSE: 2.7406e-06
Epoch 4/5
8326/8326 [==============================] - 64s 8ms/step - loss: 2.0723e-06 - MSE: 2.0723e-06
Epoch 5/5
8326/8326 [==============================] - 65s 8ms/step - loss: 1.9789e-06 - MSE: 1.9789e-06
```

With a training loss that can be represented as follows:

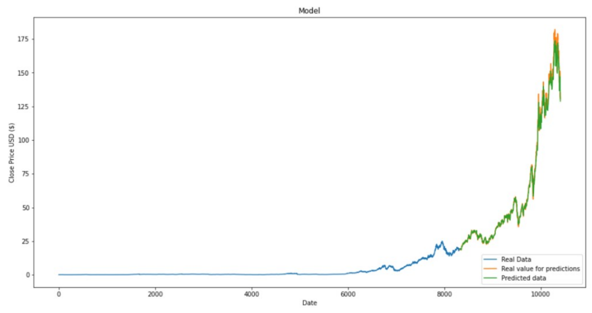


Finally, the model shows the following testing loss:



```
66/66 [==============================] - 1s 4ms/step
0.9140183157903286
```

The final results can also be represented in the predicted plot:



4.5. Summary

METER UNA TABLA CON LOS RESULTADOS

5. Conclusions

After having analysed how to predict the stock price values for the Apple company by using three different algorithms, several conclusions may be drawn.

In the first place, it has been demonstrated that the flexibility of the Long Short Term Memory Machine Learning algorithm has been the winner. It is able to fit the data better than other two methods, probably due to the fact that this method takes more into account the whole experience given by the data from the past. In the Moving Average methods, not the whole past events are taken into account, and even thought if sometimes are taken into account, the weighting gives more importance to the most recent ones. Therefore, one of the reasons of the Moving Average methods resulting in a worst solution, could be the seasonal character of the stock prices, due to the fact that Moving Average methods do not take this into account whereas the LSTM method represents better this kind of influence due to its flexibility.

In the second place, the Exponential Moving Average fits into the data better than the Simple Moving Average due to the fact that weights the used input data for the calculations by giving greater importance to the most recent samples. Anyway, if the Simple Average Method must be chosen, the window size number is a crucial parameter to be taken into account. From the analysis aforementioned, it has been drawn that smaller window size's adapt in a easier way to the curve, therefore withstanding high variance scenarios such as stock prediction. Finally, for flatter time series profiles, larger window size numbers can be used.

6. Future Work

LUGAR PARA CONTAR LA PELICULA DE INVERSOR FINANCIERO

7. Deriving theoretical properties of the algorithm

you may introduce an academic conversation along with your practical problem. Or, you may want to point out the challenges of the algorithm of your choice from the theoretical perspective. From what I understand, this should be elaborate, not just your own personal opinion.

Complexity of algorithms (function of the neuron number)

References

[1] Stock price prediction in finance. 2

[2] Stock price prediction with python. 2