CVPR #01

ILLINOIS INSTITUTE OF TECHNOLOGY

CS 584 MACHINE LEARNING

FINAL SUBJECT PROJECT

Machine Learning model for stock prediction

Authors

Ferro Bañales, Julen ferrojulen@gmail.com
González Vergara, Eneko egonzalez30@hawk.iit.edu

December 3, 2022

CVPR #01

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

CVPR #01

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:

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

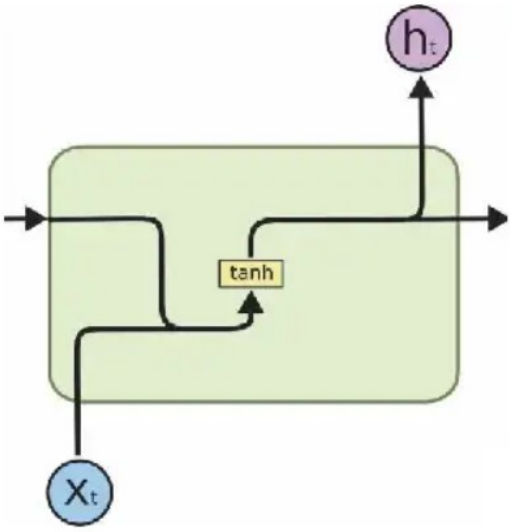
2.3. Simple RNN

A SimpleRNN is a type of recurrent neural network that processes sequential data one step at a time, with a single hidden layer. It can be useful for modeling simple sequential data, but it has a limitation known as the vanishing gradient problem, which can make it difficult to train for longer sequences.

In a SimpleRNN, the hidden layer processes the input sequence one element at a time, producing a new output at each time step. The output at each time step is then fed back into the network as input for the next time step, allowing the network to retain information about the sequence as it processes it.

One of the key advantages of SimpleRNNs is that they are relatively simple to train and use, making them a good choice for simple sequential data. However, they are limited in their ability to retain long-term dependencies in the data, which can make them less effective for more complex tasks.

This approach can be better understand in the following representation:



2.4. Gated Recurrent Unit (GRU)

A GRU, or gated recurrent unit, is a type of recurrent neural network that has been designed to overcome the vanishing gradient problem. It is similar to a SimpleRNN, but it uses gating mechanisms, which allow the network to selectively retain or forget information from previous steps in the sequence, helping to prevent the gradient from vanishing.

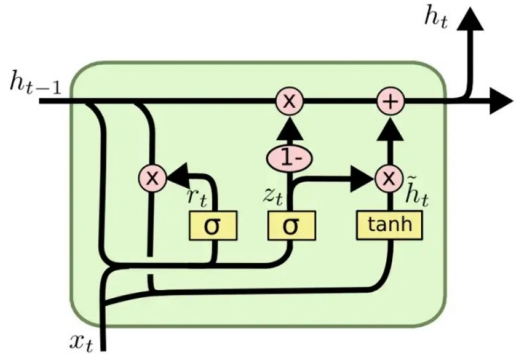
In a GRU, the hidden layer processes the input sequence one element at a time, just like in a SimpleRNN. However, instead of simply passing the output at each time step to the next time step as input, the GRU uses gating mechanisms to

CVPR #01

control the flow of information through the network. These gating mechanisms allow the network to selectively retain or forget information from previous steps in the sequence, depending on the input at each time step.

One of the main advantages of using a GRU instead of a SimpleRNN is that it is less susceptible to the vanishing gradient problem. This makes it possible to train a GRU for longer sequences, allowing it to better capture long-term dependencies in the data. However, GRUs are still relatively simple and efficient to train and use, making them a good choice for many sequential data modeling tasks.

It can be represented as follows:



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

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

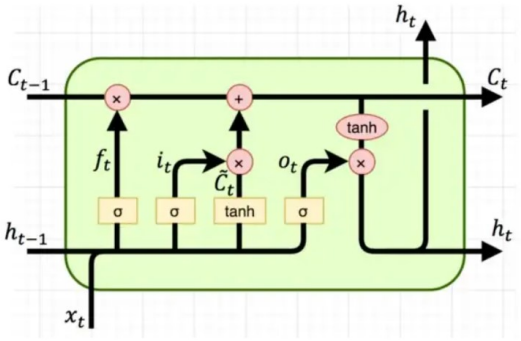
2.5. Long Short Term Memory (LSTM)

A LSTM, or long short-term memory network, is another type of recurrent neural network that has been designed to overcome the vanishing gradient problem. Like the GRU, it uses gating mechanisms to control the flow of information through the network, but it has a more complex structure, with multiple layers and additional gating mechanisms, which allows it to better retain long-term dependencies in the data.

In a LSTM, the hidden layer is made up of a series of interconnected ”cells,” each of which contains multiple gates that control the flow of information into and out of the cell. These gates allow the LSTM to selectively retain or forget information from previous time steps in the sequence, similar to the way that the gates in a GRU work.

One of the main advantages of using a LSTM instead of a GRU is that it is better able to retain long-term dependencies in the data. This makes it a good choice for tasks that require the network to remember and make use of information from earlier in the sequence. However, LSTMs are more complex and computationally expensive than GRUs, so they may not be the best choice for all tasks.

It can be represented as follows:



The equations for the LSTM are as follows:

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

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.

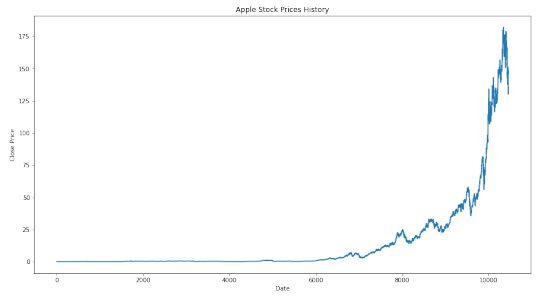
The data set in matter is composed of more or less ten thousands of rows, that is to say, ten thousands of samples. It is quite a lot of information but the fact that almost forty years of stock price evolution is covered must be taken into account.

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:

CVPR #01



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:

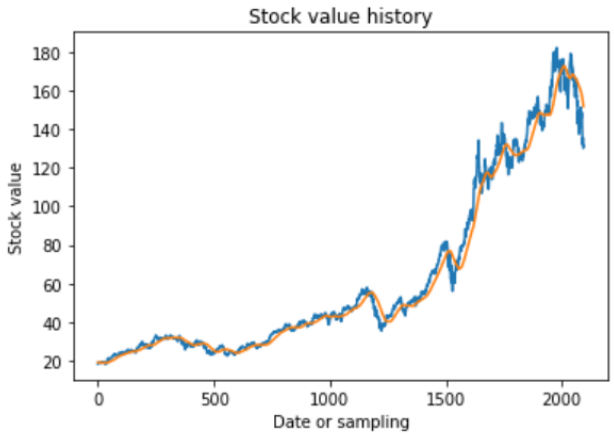
- Numpy. It provides tools for performing operations on arrays of numbers, including mathematical functions, linear algebra operations, and random number generation, and allows for efficient manipulation of large datasets.
- Matplotlib. It provides a range of tools for creating static, animated, and interactive visualizations in Python, allowing users to create a wide variety of plots, charts, and other visualizations to help understand and analyze data.
- Pandas. It provides tools for reading and writing data to and from a variety of formats, including CSV and Excel files, as well as tools for manipulating and analyzing data in the form of dataframes, which are rectangular arrays of data with labeled rows and columns.
- Scikit-Learn. It provides a range of tools for building, training, and evaluating machine learning models, including tools for preprocessing data, selecting and tuning model hyper-parameters, and evaluating model performance.
- Tensor Flow. TensorFlow is an open-source library for building and training machine learning models.
- Keras. Keras is a high-level neural networks API for building and training deep learning models. It provides a user-friendly interface for defining and training neural networks. Keras is designed to make it easy to experiment with different models and to quickly prototype new ideas, and is widely used in both academia and industry.

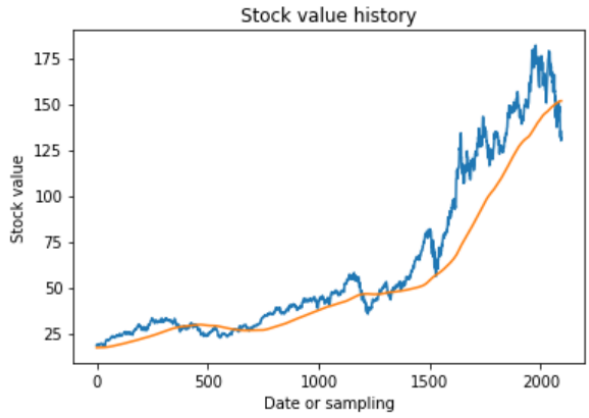
4. Results

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

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

CVPR #01

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. The better accuracy of the lowest window size model, can be appreciated in terms of both, the ***Medium Average Error*** and the ***Root Mean Square Error***, due to the fact that both statistical accuracy quality measuring indicators have lower values for the lowest window size model.

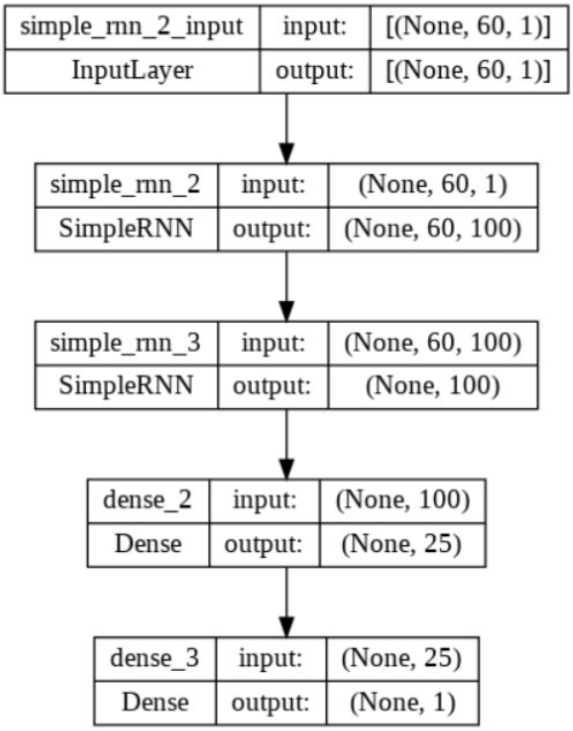
```
THE MOVING AVERAGE MODEL with window size = 50
MAPE: 5.8
RMSE: 5.82


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

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:



The model has been trained with the following hyperparameters values:

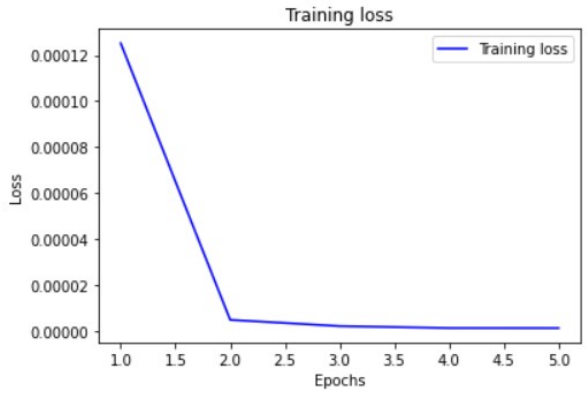
- Optimizer: Adam
- Learning Rate:0.0001
- Loss: Mean Squared Error
- Epochs: 5
- Batch Size: 1

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
```

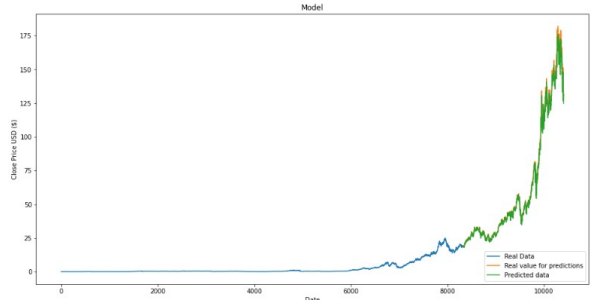
With a training loss that can be represented as follows:



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:

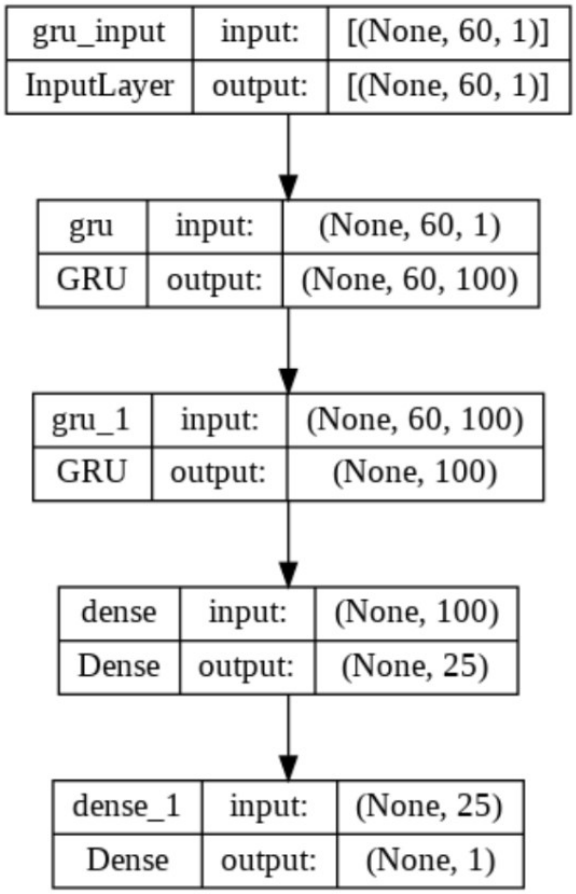


4.3. Gated Recurrent Unit (GRU)

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 hyperparameters values:

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

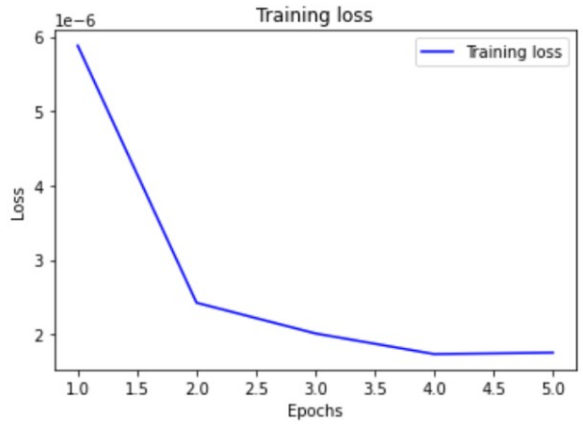
With all the conditions described above the obtained results are as follows.

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:

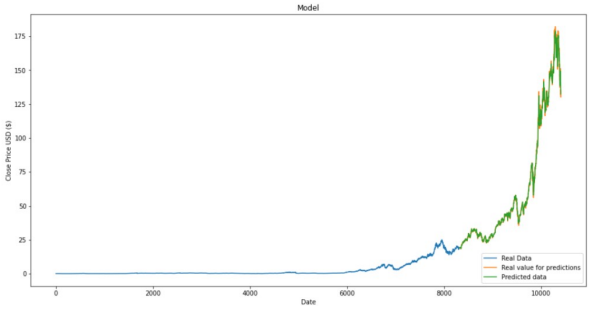
CVPR #01



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:

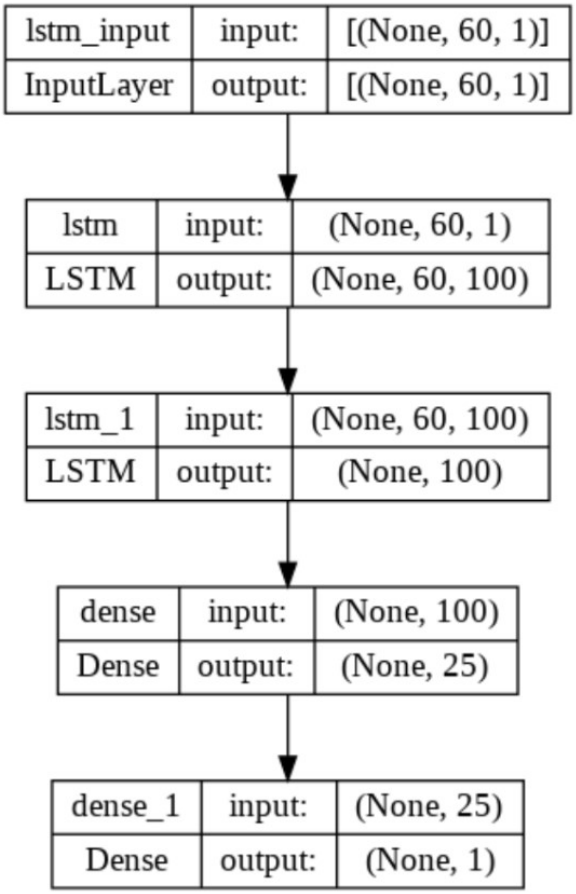


4.4. Long Short Term Memory (LSTM)

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:



The model has been trained with the following hyperparameters values:

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

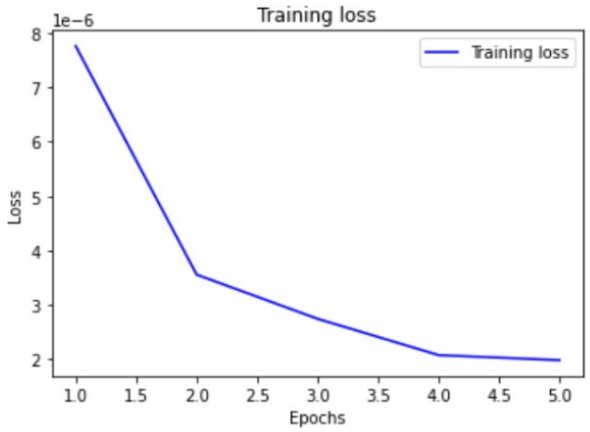
With all the conditions described above the obtained results are as follows.

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:

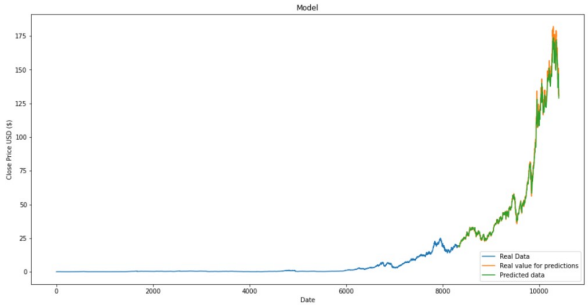
CVPR #01



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 of the results

In the table underneath, it can be seen the different values of the accuracy indicators after having run the simulations. Taking into account the results, the order of the most accurate algorithms for ***Stock Value prediction*** can be performed. Therefore, the algorithms follow this order from the most accurate to the least accurate: GRU, LSTM, simple RNN and SMA; being the ***Gated Recurrent Unit*** the most accurate algorithm.

	SMA		*RNN*	*GRU*	*LSTM*
	W. Size = 50	*W. Size = 300*			
MAE	5.8	16.19	-	-	-
Loss	5.82	15.94	1.37	0.06	0.91

5. Deriving theoretical properties of the algorithm

About the theoretical properties of the algorithm, the authors have concluded that the best information that could fit into this part of the report could be to mention the time complexity of the different algorithms. In the first place, the fastest one would be the SMA taking into account that it only need to average an array of numbers whose length would be the window size. That said, the time complexity of this algorithm would be WS = Window Size and it is helpful to know that the most accurate algorithm would also be the fastest one only taking into account the SMA methods, because the ones with the lowest WS values will be the most accurate ones and the ones which needs to compute a smaller amount of numebrs for averaging.

About the time complexity of the other algorithms it can only be said that it will be for sure greater than WS or N, due to the fact that this kind of ***Artificial Intelligence*** algorithm could be as complex as the developer wants. Therefore, it can be only concluded that the computing time for these other algorithms has been the greatest one, mainly the LSTM algorithm, due to the fact that is the fanciest and the most accurate one. It also must be taken into account that the time complexity should encompass both stages, the training and testing phases.

6. Future Work

The future work of a project of this characteristics may vary depending on the goals of the project. For that reason here are some possible directions for future work could include:

- Improving the model's performance on the stock prediction task, by exploring different model architectures, training algorithms, and hyperparameter settings, and by incorporating additional data sources or features.
- Evaluating the model's performance on a wider range of stocks, to see if it is able to generalize well to different stocks and market conditions.
- Incorporating additional information about the stock market, such as news articles, analyst reports, or sentiment data, to see if this can improve the model's predictions.
- Developing methods for interpreting the model's predictions, to understand the factors that are driving the model's decisions and to provide insights into the stock market.
- Integrating the model into a larger stock trading system, to see how it can be used to make real-time predictions and decisions.
- Conducting a more thorough analysis of the model's performance, including evaluating its accuracy, precision, and recall, and comparing its performance to other models or benchmarks.

7. Conclusions

After having analysed how to predict the stock price values for the Apple company by using three different

CVPR #01

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. Even though the results of the LSTM seem to be worst than the GRU in terms of quality metrics, it must be taken into account that a well programmed LSTM model should give better results due to the higher complexity of the algorithm. Nevertheless, this is a double-edged sword if junior programmers are in charge of implementing the model, as it is the case of this report. Hence, the LSTM has not been hundred percent optimized in this report, so the work done for this report gives room for further improvements in next stages of the author's education.

Compared to the SMA and the RNN, the LSTM model 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, being a good practice trying to fit a model with the greatest values of a range of small window size values, bounded by some constrains determined experimentally when running the model. 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. Additionally, for flatter time series profiles, larger window size numbers can be used.

Finally, it has been made clear that the algorithms involving any kind of ***Machine Learning optimization*** technique, lead the simulations to more accurate results as it helps the model to learn from previous data and adopt the curve by taking up the patters that relate the inputs of the ***Neural Networks*** with the outputs. In fact, the simple RNN model and the LSTM model (the complex version of the simple RNN) should be highlighted, due to the well-known potential of the algorithms based on these kind of mathematical foundation. The implementation made so far for this report is far enough accurate, but it is well-known within the industry that there is much more work to be done in terms of optimizing the model for reaching almost null-error results.

8. GitHub

This is the repository used for the project: https://github.com/ferriitoo/CS584-Machine-Learning/tree/main/project

References

[1] Stock price prediction in finance. 2

[2] Stock price prediction with python. 2