UPEC – M2 SDTS

EU – Deep Learning

Academic year : *2021/2022*

Names: BOUTOUDJ, TLEMCANI

First names: Mustapha, Ramy

Course Instructor : Dr. Amir Nakib

Deep learning project :

Stock Market prediction:

Short-term trend in stock prices using deep learning

Summary :

A time series is a collection of data points that are stored with respect to their time. Mathematical and statistical analysis performed on this kind of data to find hidden patterns and meaningful insight is called timeseries analysis. Time-series modeling techniques are used to understand past patterns from the data and try to forecast future horizons. These techniques and methodologies have been evolving for decades.

The change of stock price trend has always been identified as a very important problem in the economic field. Stock prices are influenced by various internal and external factors, such as domestic and foreign economic environment, international situation (ex : covid, war …), industry outlook, financial data of listed companies and stock market operation. Thus, the forecasting method also has a different emphasis.

The traditional analysis method is based on economics and finance, which mainly uses the fundamental analysis method and the technical analysis method. On the one hand, the fundamental analysis method pays more attention to the intrinsic value of stocks and qualitatively analyzes the external factors that affect the stock, such as interest rate, exchange rate, inflation, industrial policy, financing of listed companies, international relations, and other economic and political factors. On the other hand, the technical analysis method mainly focuses on the direction of stock prices, trading volume and psychological expectations of investors, which mainly focuses on analyzing the trajectory of the stock index of individual stocks or the whole market by using curve charts and other tools. Currently, traditional fundamental analysis and technical analysis are still the most common methods used by many organizations and individual investors.

The accuracy of the traditional fundamental analysis method is difficult to convince. The reason is not only that the influencing factors are in a long-term cycle, but also that the forecasting results depend more on the professional quality of the analysts. As a financial time series, stock market data has the characteristics of a random walk. Based on statistics and probability theory, some researchers use a linear time series forecasting model to predict short-term stock prices with a large amount of long-term data, such as vector autoregression (VAR), Bayesian vector autoregression (BVAR) model,

autoregressive integrated moving average mode (ARIMA), and generalized autoregressive conditional heteroscedasticity (GARCH) model.

However, the accuracy of using the time series model alone is questionable due to the uncertainty and high noise characteristics of financial time series, and the relationship between the independent and dependent variables is subject to dynamic changes over time, which limits its application and expansion.

t has some limitations to predict the stock price trend with one simply using the linear time series forecasting model or neural network model. At present, combining the advantages of various methods and using various better algorithms to improve the hybrid method is the development trend of financial time series deep learning. Therefore, in order to make the most of the time series characteristics of data series, deepen the data characteristics and improve the accuracy of stock price forecasting, the LSTM-based stock price forecasting method for closing stock price is proposed. Long Short Term Memory (LSTM) which can not only find the interdependence of data in time series data, but also automatically detect the best mode suitable for the relevant data, this method can effectively improve the accuracy of stock price forecasting.

Table of contents :

I- Introduction :

A lot of people are trying to imagine when a stock will go up or down in the future (in two days, next week...), then with the remaining money they have, they invest or sell that stock, or take short or long position. After playing the stock market knowing if the stock will go up or down in value, we can always take profit by selling those shares at the right time.

Unfortunately, this is impossible because nobody can know the future. However, we can make educated guesses and forecasts based on the information we have in the present and the past about the entire stock. An estimated guess based on past stock price movements and patterns is called technical analysis . We can use technical analysis (TA) to predict the direction of a stock's price, however, it is not 100% accurate. In fact, some traders criticize TA and have stated that it is just as effective at predicting the future as astrology. But there are other traders who swear by it and have established long successful trading careers.

In our case, being students in the field of artificial intelligence, we will try to use our knowledge to try to predict these future data from the data we have (the present data), and for this and after several researches we came to the conclusion that the most suitable way for our project is the neural network that uses MT to help it to make informed predictions. The specific neural network that we will implement is called LSTM (Long short-term memory), which is a type of recurrent neural network.

Various machine learning algorithms such as neural networks, genetic algorithms, support vector machine, and others are used to predict stock prices. Recurrent neural networks (RNNs) are a powerful model for processing sequential data such as sound, time series data or written natural language .Some designs of RNNs were used to predict stock market . Long Short-Term Memory (LSTM) is one of the most successful RNNs architectures. LSTM introduces the memory cell, a unit of computation that replaces traditional artificial neurons in the hidden layer of a network. With these memory cells, networks are able to effectively associate memories and input remote in time, hence suit to grasp the structure of data dynamically over time with high prediction capacity.

II- Neural network :

Neural networks are one of the most popular machine learning algorithms and also outperform other algorithms in terms of accuracy and speed.

A neural network consists of different layers connected to each other, working on the structure and function of a human brain. It learns from huge volumes of data and uses complex algorithms to train a neural network.

Several neural networks can help solve different business problems :

- Feed-Forward Neural Network: used for general regression and classification problems.

- Convolutional Neural Network: used for object detection and image classification.

- Deep Belief Network: used in the health sector for cancer detection.

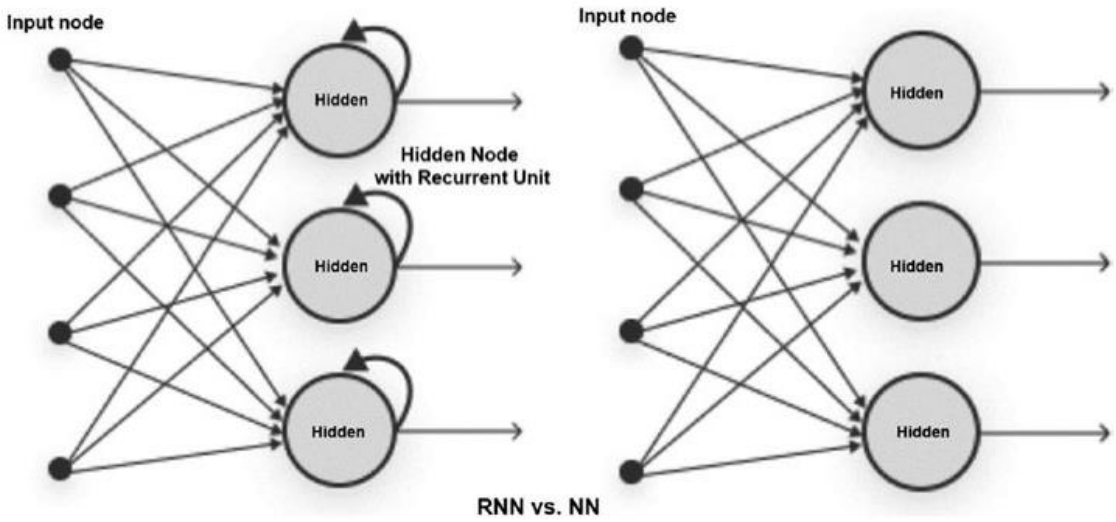
- RNN : used for speech recognition, time series prediction and natural language processing.

In our case we will use the RNN, which is more relevant to our data, which is about short term prediction of stock prices, which means time series prediction, predicting future data from present data.

1- Recurrent Neuronal Network (RNN) :

A recurrent neural network (RNN) is a sequential class of neural network where the output from the previous state is served as input to the current state. In a neural network, all inputs and outputs are also independent of each other, but when it is required to predict things in sequence, the RNN works well. The RNN handles data that is sequential in fashion. So, when order matters in the data, an RNN should be used. Examples of such data are natural language data, speech data, time-series data, video data, music data, and stock market data, they are embedded in popular applications such as Siri, voice search, and Google Translate.

Like convolutional neural networks (CNN), recurrent neural networks use training data to learn. They are distinguished by their "memory" because they take information from previous inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depends on previous elements of the sequence.



Regular RNN versus NN

Operation of the RNN :

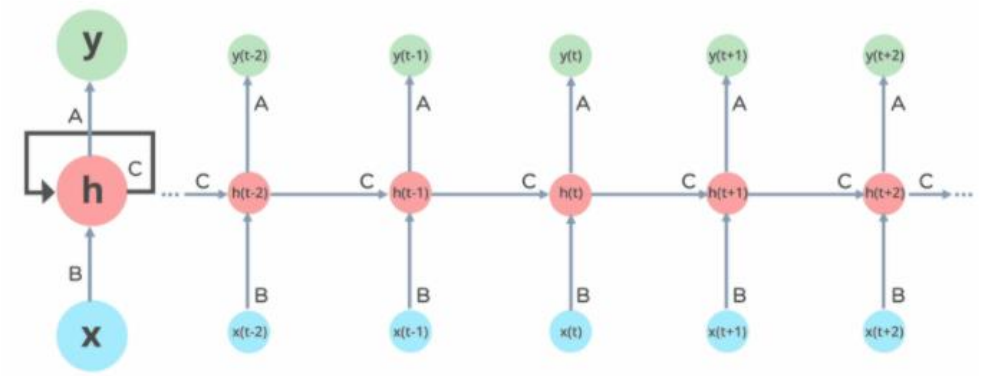


Figure : operation of the recurrent neural network

The input layer "x" receives the input from the neural network, processes it and passes it to the intermediate layer.

The intermediate layer "h" can consist of several hidden layers, each with its own activation functions, weights and bias. If we have a neural network where the different parameters of the different hidden layers are not affected by the previous layer, i.e. the neural network has no memory, then we can use a recurrent neural network.

The recurrent neural network will normalize the different activation functions, weights and biases so that each hidden layer has the same parameters. Then, instead of creating several hidden layers, it will create one and loop through it as many times as necessary.

Types of RNNs:

- One to One: This type of neural network is known as Vanilla Neural Network. It is used for general machine learning problems, which have a single input and a single output.
- One to Many: This type of neural network has a single input and multiple outputs.
- Many to One: takes a sequence of inputs and generates a single output. For example: Sentiment analysis, this type of network where a given sentence can be classified as expressing positive or negative feelings.
- Many to Many: takes a sequence of inputs and generates a sequence of outputs, for example: Machine translation.

2 – Problems with the use of standard RNN :

- Vanishing Gradient Problem :

Recurrent neural networks allow us to model time-dependent and sequential data problems, such as stock market forecasting, machine translation and text generation. However, we will find that RNNs are difficult to train due to the gradient problem.

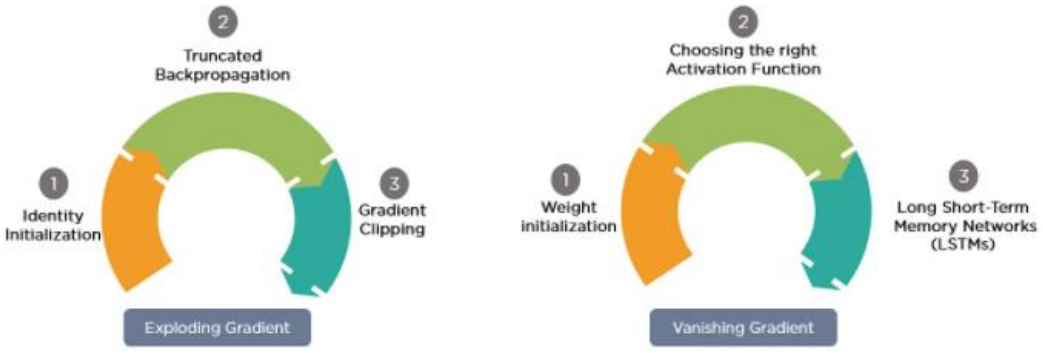
RNNs suffer from the leaky gradient problem. Gradients carry information used in RNN, and when the gradient becomes too small, parameter updates become trivial. This makes it difficult to learn long sequences of data.

- Exploding Gradient Problem :

When training a neural network, if the slope tends to grow exponentially instead of decreasing, it is called an explosive gradient. This problem occurs when large error gradients accumulate, resulting in very large updates to the neural network model weights during the training process.

Long training time, poor performance, and poor accuracy are the main problems with gradient problems.

3 - Solutions to gradient problems :



The most popular and effective way to deal with gradient problems is the Long Short Term Memory (LSTM) network.

4 - Long Short-Term Memory Networks :

Long short-term memory (LSTM) was first mentioned in 1997 by *Sepp Hoch Reiter* and *Jürgen Schmid Huber*. By using constant error carousel (CEC) units, LSTM treats the exploding and vanishing gradient problems. The preliminary version of the LSTM block incorporated cells, input, and output gates. It is famous for its design, which is a specific type of recurrent neural network, utilizing many different real-world applications that the standard version doesn't. An RNN's biggest problem is that it only holds the previous state information, causing the vanishing gradient problem. This problem has been solved by LSTM. LSTM was constructed to avoid the issue of long-term dependencies. Remembering information for a long duration is basically its default behavior. All RNNs have repeating chain modules of the

neural network. This repeating module is a simple structure, such as the tanH layer in an RNN. LSTM also has an identical chaining structure instead of having a single neural network layer

LSTMs are a special type of RNN, capable of learning long-term dependencies by remembering information for long periods of time is the default behavior.

All RNNs take the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh.

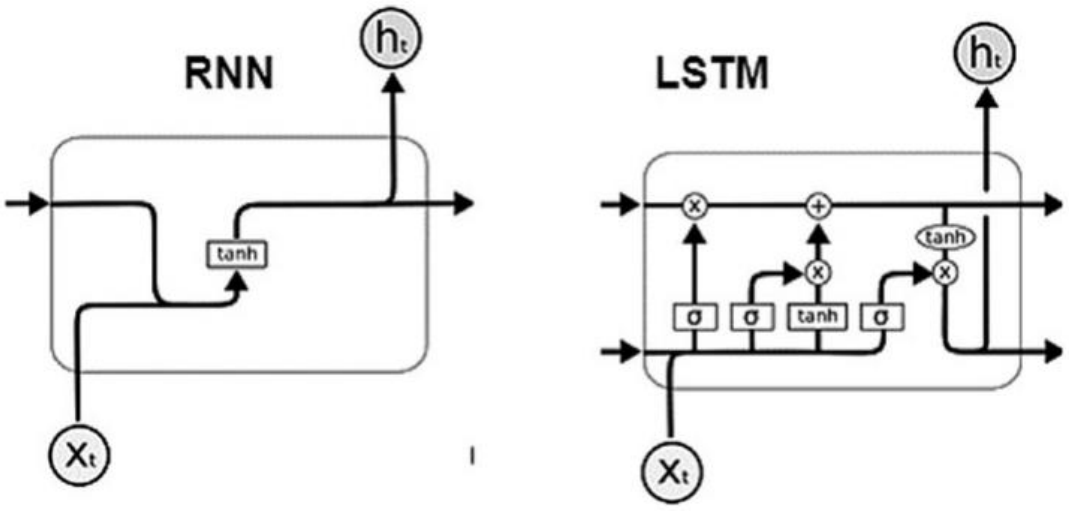


Figure : RNN versus LSTM

LSTM has an extra feature as compared to RNNs, which is called memory.

- Forget gate f (a neural network with sigmoid activation)
- Candidate layer C (a neural network with tanH activation)
- Input gate I (a neural network with sigmoid activation)
- Output gate O (a neural network with sigmoid activation)
- Hidden state h
- Memory cell C

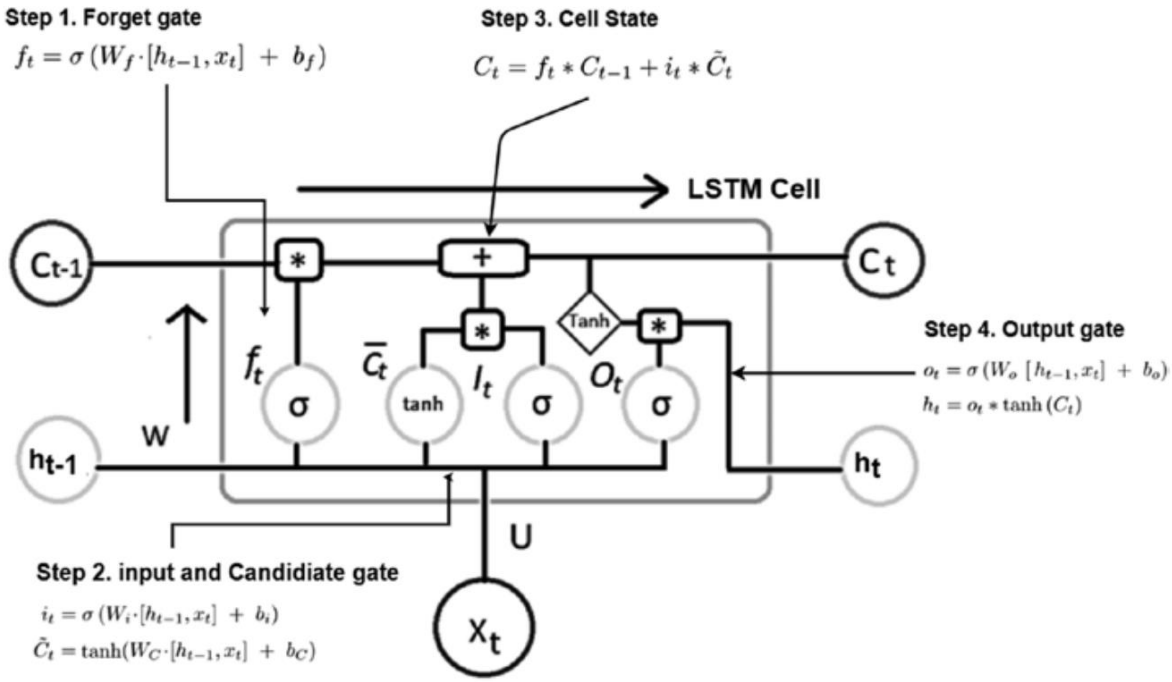


Figure X illustrates that it has four different gates: forget gate, candidate cate, input gate, and output gate. All gates are single-layer neural networks with the sigmoid activation function, excluding the candidate gate, which is using the tanH activation function

Figure shows LSTM at the T time step.

• The input cell contains x (input vector), ht-1 (previous hidden state), and C (previous memory state).

• The output cell contains h (current hidden cell) and C (current memory cell).

• W and U are weight vectors.

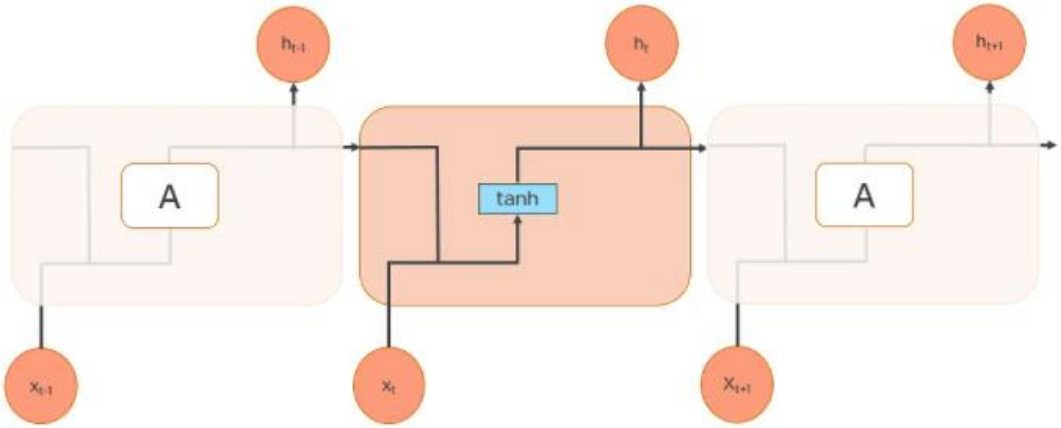
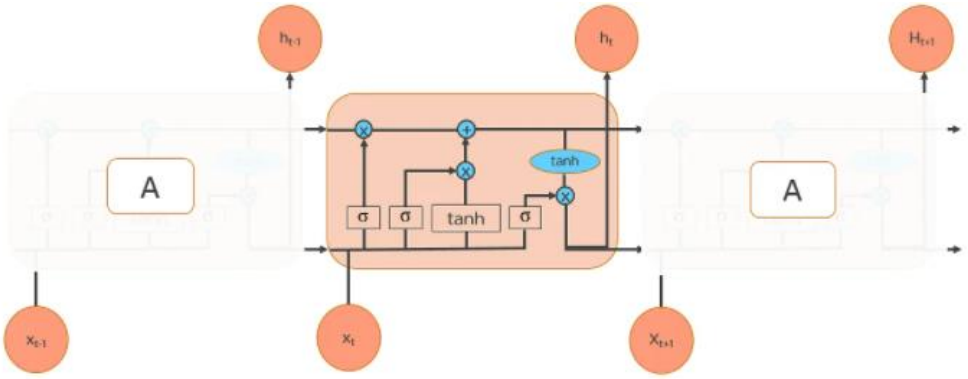
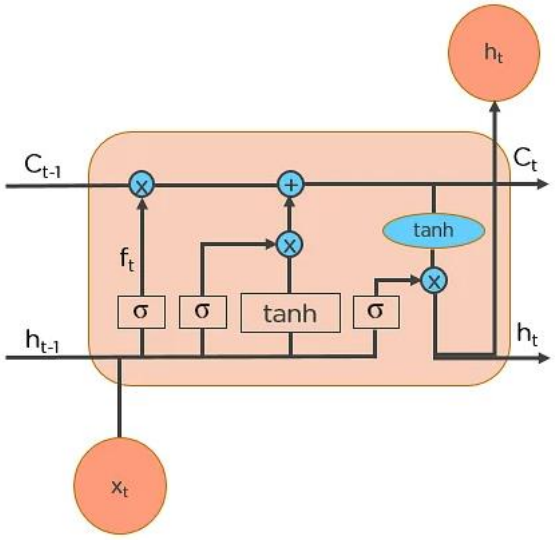


Figure : Long Short-Term Memory Networks

LSTMs also have a chain-like structure, but the repeating module is a slightly different structure. Instead of having a single neural network layer, four interacting layers communicate in extraordinary ways :



5 - How LSTMs work ? :



- 5 – 1- Decide how much past data to retain :

The first step in LSTM is to decide what information should be omitted from the cell at that particular time step. The sigmoid function determines this. It looks at the previous state (ht-1) with the current input xt and calculates the function.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

f_t = forget gate
Decides which information to delete that is not important from previous time step

Example :

Consider the following two sentences :

- Let's say the output of h(t-1): *"Bob is good in Math. Alice, on the other hand, is good in Mechanics".*

- Or the current entry at x(t): *"Alice plays soccer well, she told me on the phone that he was captain of his team".*

The forgotten door realizes that there might be a change of context after meeting the first point. It compares itself to the current input sentence at x(t). The next sentence is about Alice, so the information about Bob is removed. The subject position is released and assigned to Alice.

- 5- 2- Decide how much this unit adds to the current state :

In the second layer, there are two parts. One is the sigmoid function and the other is the tanh function. In the sigmoid function, it decides which values to pass (0 or 1). The tanh function gives a weight to the values that are passed, deciding their level of importance (-1 to 1).

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

i_t = input gate
Determines which information to let through based on its significance in the current time step

With the current entry at x(t), the gateway parses the important information- Alice plays soccer, and the fact that she is the captain of his team is important.

"He told me on the phone" is less important; so it is forgotten. This process of adding new information can be done via the front door.

- 5- 3- Decide which part of the current state of the cell reaches the output :

The third step is to decide what the output will be. First, we run a sigmoid layer, which decides which parts of the cell state reach the output. Then we put the cell state through tanh to push values between -1 and 1 and multiply it by the output of the sigmoid gate :

$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

o_t = output gate
Allows the passed in information to impact the output in the current time step

6- PROPHET

Prophet is an open source framework from Facebook used for framing and forecasting time series. It focuses on an additive model where nonlinear trends fit with daily, weekly, and yearly seasonality and additional holiday effects. Prophet is powerful at handling missing data and shifts within the trends and generally handles outliers well. It also allows you to accumulate exogenous variables to the model.

There are numerous techniques for time series forecasting. MAPE can be used for precise prophecy. This equips us with the expertise to make time-series predictions with good accuracy utilizing simple inherent parameters and has provision for incorporating the impression of inheritance seasonality and holidays. Facebook Prophet is seeking to fit various linear and nonlinear functions of time as elements. Modeling seasonality as an additive element is an identical strategy taken by exponential smoothing. This library is so significant that it has the potentiality of handling stationary within the data and also seasonality related elements.

But Facebook Prophet has some limitations like it presumes to be input columns with names ‘ds’ and ‘y’ where ‘ds’ is Date and ‘y’ is the target variable. Here, a trend can either be positive or negative and may be increasing or decreasing. The sample of this data is as follows:

The Prophet Model :

Prophet uses a decomposable time-series model.

$$y(t) = g(t) + s(t) + h(t) + \epsilon t$$

where

g(t) = Trend (linear/logistic)

s(t) = Periodic change/seasonality

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right)$$

Here, P is the consistent period we anticipate the time series to have (e.g., P = 365.25 for yearly data or P = 7 for weekly data, when we are scaling our time variable in days). Fitting seasonality requires calculating the 2N parameters for β = [a1, b1,…., aN, bN] T by developing a matrix of seasonality vectors for each and every value of t in our historical data and future data

h(t) = Effect of holiday

For each holiday i, let D i be the set of past and future dates for that holiday, as shown here:

$$Z(t) = [1(t \in D1),\ldots,1(t \in DL)]$$

and taking the following:

$$h(t) = Z(t)\kappa$$

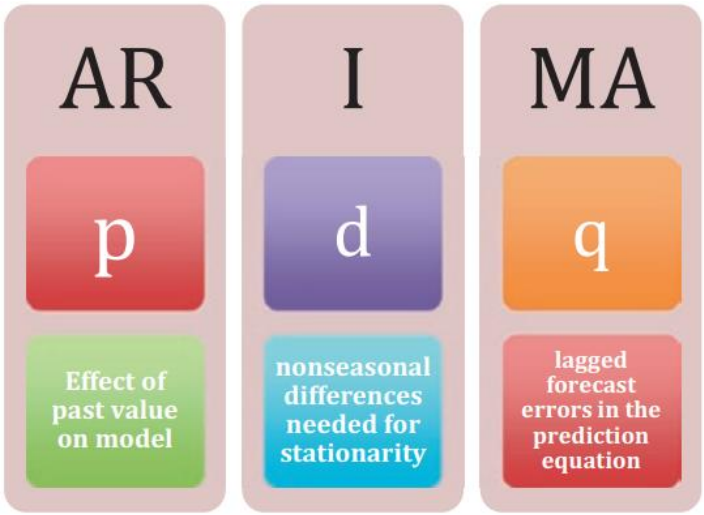
As with seasonality, we use a prior $\kappa \sim \text{Normal}(0, v^2)$.

ϵt = Changes that are not adopted by the model

Prophet's core procedure is implemented using Stan (a probabilistic programming language). Stan performs map optimization to find parameters and facilitates estimating parameter uncertainty using the Hamiltonian Monte Carlo algorithm.

7) ARIMA

Autoregressive integrated moving average—also called ARIMA(p,d,q)—is a forecasting equation that can make time series stationary with the help of differencing and log techniques when required. A time series that should be differentiated to be stationary is an integrated (d) (I) series. Lags of the stationary series are classified as autoregressive (p), which is designated in (AR) terms. Lags of the forecast errors are classified as moving averages (q), which are identified in (MA) terms



A nonseasonal ARIMA model is called an ARIMA(p,d,q) model, where:

• p is the number of autoregressive terms.

• d is the number of nonseasonal differences needed for stationarity.

• q is the number of lagged forecast errors in the prediction equation

ARIMA (0, 0, 0)	0	0	0	$y_t = Y_t$	White noise
ARIMA (0, 1, 0)	0	1	0	$y_t = Y_t - Y_{t-1}$	Random walk
ARIMA (0, 2, 0)	0	2	0	$y_t = Yt - 2Yt\text{-}1 + Yt\text{-}2$	Constant
ARIMA (1, 0, 0)	1	0	0	$\hat{Y}t = \mu + \phi_1 Y_{t-1} + \varepsilon$	AR(1): First-order regression model
ARIMA (2, 0, 0)	2	0	0	$\hat{Y}t = \phi_0 + \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \varepsilon$	AR(2): Second-order regression model
ARIMA (1, 1, 0)	1	1	0	$\hat{Y}t = \mu + Yt\text{-}1 + \phi 1\ (Yt\text{-}1 - Yt\text{-}2)$	Differenced first-order autoregressive model
ARIMA (0, 1, 1)	0	1	1	$\hat{Y}t = Yt\text{-}1 - \phi 1 et\text{-}1$	Simple exponential smoothing
ARIMA (0, 0, 1)	0	0	1	$\hat{Y}t = \mu_0 + \varepsilon t - \omega_1 \varepsilon_{t-1}$	MA(1): First-order regression model
ARIMA (0, 0, 2	0	0	2	$\hat{Y}t = \mu_0 + \varepsilon t - \omega_1 \varepsilon_{t-1} - \omega_2 \varepsilon_{t-2}$	MA(1): Second-order regression model
ARIMA (1, 0, 1)	1	0	1	$\hat{Y}t = \phi_0 + \phi_1 Y_{t-1} + \varepsilon t - \omega_1 \varepsilon_{t-1}$	ARMA model
ARIMA (1, 1, 1)	1	1	1	$\Delta Y_t = \phi_1 Y_{t-1} + \varepsilon t - \omega_1 \varepsilon_{t-1}$	ARIMA model

ARIMA (1, 1, 2)	1	1	2	$\hat{Y}_t = Y_{t-1} + \phi_1 (Y_{t-1} - Y_{t-2}) - \theta_1 e_{t-1} - \theta_1 e_{t-1}$	Damped-trend linear Exponential smoothing
ARIMA (0, 2, 1) OR (0,2,2)	0	2	1	$\hat{Y}_t = 2\,Y_{t-1} - Y_{t-2} - \theta_1 e_{t-1} - \theta_2 e_{t-2}$	Linear exponential smoothing

ARIMA is a method among several used for forecasting univariate variables, which uses information obtained from the variable itself to predict its trend. The variables are regressed on its own past values.

AR(p) is where p equals the order of autocorrelation (designates weighted moving average over past observations) z I (d), where d is the order of integration (differencing), which indicates linear trend or polynomial trend z.

MA(q) is where q equals the order of moving averages (designates weighted moving average over past errors).

ARIMA is made up of two models: AR and MA.

Time-series data is often nonstationary, and to make time-series stationary, the series needs to be differentiated. This process is known as the integration part (, and the order of differencing is signified as d.

Differencing eradicates signals with time, which contains trends and seasonality, so this series contains noise and an irregular component, which will be modeled only

Integral d Value	**Formula(yt)**
d = 0	Y_t
d = 1	$Y_t - Y_{t-1}$
d = 2	$Y_t - 2Y_{t-1} + Y_{t-2}$

III- Data Cleaning :

Now that we have a good understanding of how LSTMs work, let's proceed to a practical implementation to predict short-term stock prices, using the historical data of the company APPLE imported through yfinance (yahoo api)

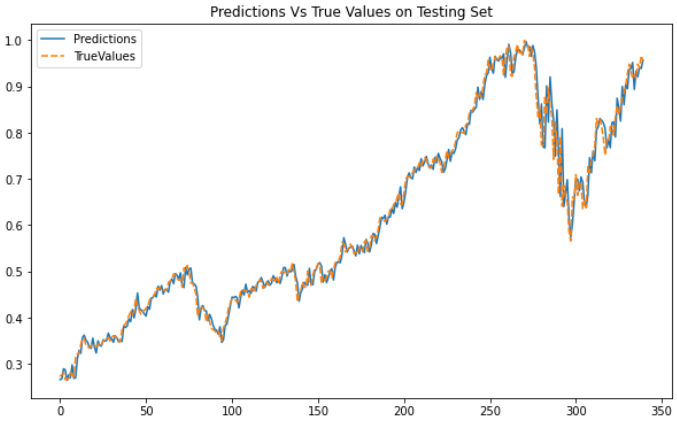
Based on the historical data (from 2017-01-01 to 2022-01-01), we will predict the stock price of the company AMAZON :

IV- Use of other algorithms for comparison :

1- Linear regression :

It could seem trivial to predict stock prices using linear regression. We will train a regression model using our database to make future predictions :

Our model worked well to predict the Apple stock price using a linear regression model.




```
print("model Accuracy on training data:",model.score(X_train, y_train))
```

```
model Accuracy on training data: 0.9970342320018716
```

```
# Model accuracy on Testing data
print("model Accuracy is on training data:",model.score(X_test, y_test))
```

```
model Accuracy is on training data: 0.9847722212152704
```

Notice : This prediction is only for the short term. So it is not recommended to use this model for medium and long term forecasting periods, because it depreciates in terms of performance. Not because our linear model is bad, but because stock markets are very volatile.

Example : This time we will use linear regression to predict the TESLA stock price :

```
#charger le data et le stocker dans un data frame
df= pd.read_csv('TSLA.csv')

df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2010-07-01	5.000	5.184	4.054	4.392	4.392	41094000
1	2010-07-02	4.600	4.620	3.742	3.840	3.840	25699000
2	2010-07-06	4.000	4.000	3.166	3.222	3.222	34334500
3	2010-07-07	3.280	3.326	2.996	3.160	3.160	34608500
4	2010-07-08	3.228	3.504	3.114	3.492	3.492	38557000

After training the algorithm, we obtain the following predictions :

Metric	Train	Test
r2_score	0.41044228972678165	0.4131373098429034
MSE	21136.840352516032	20635.915166425984

We notice that for the TESLA data, we don't get a good accuracy score for our predictions.

2 – Random forest :

Random forest is a supervised learning algorithm that is used for both classification and regression. However, it is mainly used for classification problems. As we know, a forest is composed of trees and more trees means a more robust forest. Similarly, the random forest algorithm creates decision trees on data samples, then gets the prediction of each of them and finally selects the best solution by voting. It is an ensemble method that is better than a single decision tree because it reduces overfitting by averaging the result.

Operation of the random forest algorithm :

We can understand how the Random Forest algorithm works with the following steps :

- Let's start by selecting random samples from a data set.
- Then, this algorithm will build a decision tree for each sample. Then, it will obtain the result of the prediction of each decision tree.

- In this step, voting will be done for each expected result.
- Finally, let's select the most voted prediction result as the final prediction result.

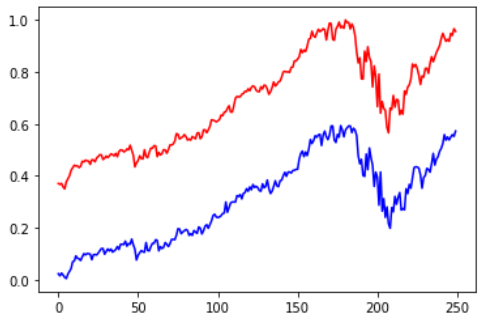
We will use this algorithm to try to see the Stock_Market prediction (using APPLE data) to check the final prediction obtained by this algorithm :

```
from sklearn.metrics import mean_squared_error
math.sqrt(mean_squared_error(test,testY))

#mean_squared_error(test,testY)
```

```
0.38008588075457683
```

```
plt.plot(test,color="blue")
plt.plot(testY,color='red')
plt.show()
```



Notice :

We notice that when we use the random forest algorithm, we get an interesting mean square error (high), and a poor accuracy score, which leads us to conclude that Random forest algorithm, is not suitable for this type of use.

3 - Autoregressive models (AR) :

An autoregressive model occurs when a value of a time series is regressed on previous values of the same time series. for example,yt to yt-1 :

$$y_t = \beta_0 + \beta_1 y_{t-1} + \epsilon_t$$

In this regression model, the response variable from the previous period has become the predictor and the errors have our usual assumptions about the errors in a simple linear regression model. The order of an autoregression is the number of immediately preceding values in the series that are used to predict the value at the present time. Thus, the previous model is a first order autoregression, denoted AR(1).

If we want to predict y this year (yt) using the global temperature measurements in the previous two years (yt-1,yt-2), then the autoregressive model to do this would be :

$$y_t = \beta_0 + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \epsilon_t$$

This model is a second-order autoregression, denoted AR(2), since the value at time t is predicted from the values at times t-1 and t-2. More generally, a Kth Order autoregression, written as AR(k), is a multiple linear regression in which the value of the series at any time t is a (linear) function of the values at times t-1,t-2,...,t-k.

Autocorrelation and partial autocorrelation :

The correlation coefficient between two values in a time series is called the autocorrelation function (ACF) :

$$\mathrm{Corr}(y_t, y_{t-k})$$

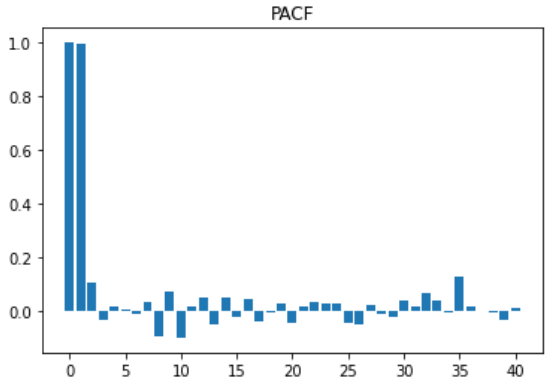
This value of k is the time interval considered and is called the lag .

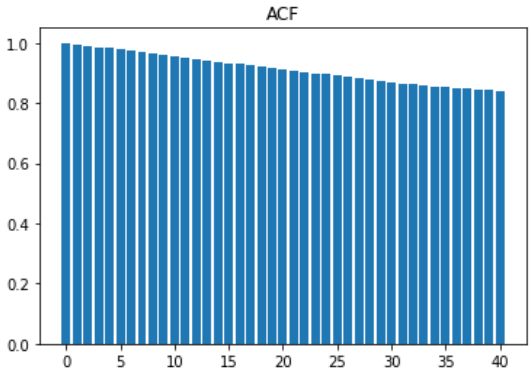
An autocorrelation with a lag of 1 (i.e. k = 1 in the above) is the correlation between values separated by one time period. More generally, an autocorrelation with lag k is the correlation between values separated by k time periods.

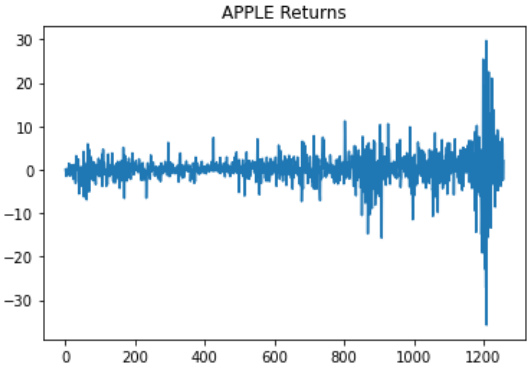
The ACF is a way to measure the linear relationship between an observation at time t and observations at previous times. If we assume an AR(k) model, then we may wish to measure only the association between yt and yt-k and filter out the linear influence of random variables that lie in between (i.e.,yt-1,yt-2,...,yt-(k-1)), which requires a transformation on the time series. Then, by computing the correlation of the transformed time series, we obtain the partial autocorrelation function (PACF).

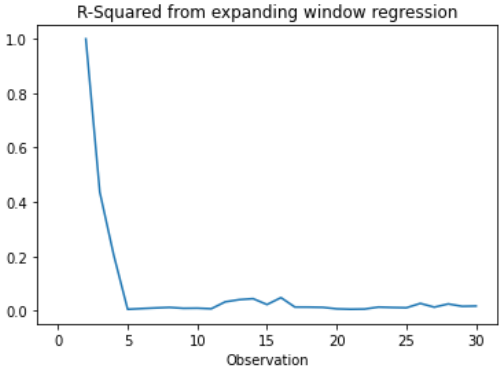
The PACF is most useful for identifying the order of an autoregressive model. Specifically, sample partial autocorrelations that are significantly different from 0 indicate lagged terms of y that are useful predictors of yt. To help differentiate between ACF and PACF, they are considered analogs of R2 and partial R2 values as discussed above.

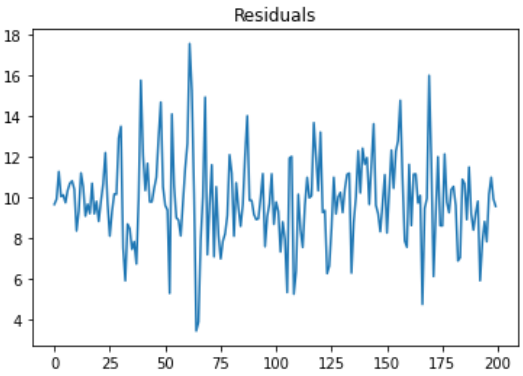
Graphical approaches to evaluating the lag of an autoregressive model include examining the ACF and PACF values against the lag. In a plot of ACF versus lag, if one sees large ACF values and a non-random pattern, then the values are likely serially correlated. In a PACF versus lag plot, the model will generally appear random, but large PACF values at a given lag indicate that this value is a possible choice for the order of an autoregressive model :

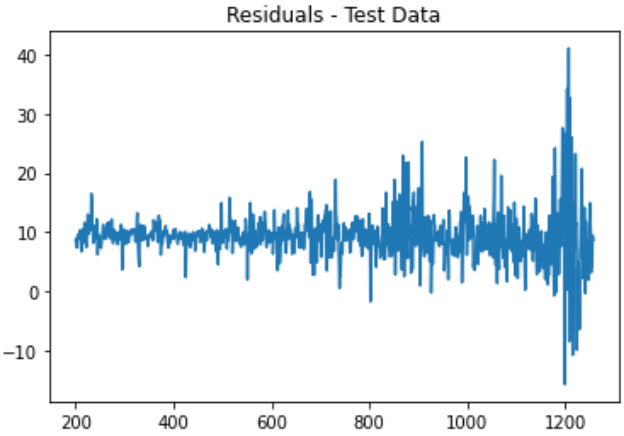












This gives us the following result :

```
# Calculer le taux de réussite (données de test)
true_neg_test = np.sum((df_2_test["Fitted Value"] <0) & (df_2_test["Actual"] <0))
true_pos_test = np.sum((df_2_test["Fitted Value"] >0) & (df_2_test["Actual"] >0))

accuracy = (true_neg_test + true_pos_test)/len(df_2_test)
accuracy
```

```
0.5406427221172023
```

Our precision is : 0.54, which confirms our choice of algorithm

Conclusion

We presented and compared three different algorithms for time series prediction. As expected, there is no clear winner and each algorithm has its own advantages and limitations. Below we summarize our observations for each algorithm:

1. **ARIMA** is a powerful model and as we saw it achieved the best result for the stock data. A challenge is that it might need careful hyperparameter tuning and a good understanding of the data.
2. **Prophet** is specifically designed for business time series prediction. It achieves very good results for the stock data but, speaking from anecdotes, it can fail spectacularly on time series datasets from other domains. In particular, this holds for time series where the notion of *calendar date* is not applicable and we cannot learn any seasonal patterns. Prophet's advantage is that it requires less hyperparameter tuning as it is specifically designed to detect patterns in business time series.
3. **LSTM-based recurrent neural networks** are probably the most powerful approach to learning from sequential data and time series are only a special case. The potential of LSTM based models is fully revealed when learning from massive datasets where we can detect complex patterns. Unlike ARIMA or Prophet, they do not rely on specific assumptions about the data such as time series stationarity or the existence of a Date field. A disadvantage is that LSTM based RNNs are difficult to interpret and it is challenging to gain intuition into their behaviour. Also, careful hyperparameter tuning is required in order to achieve good results.