INSTITUT DE SCIENCE FINANCIÈRE ET D'ASSURANCES

UNIVERSITY LYON 1, CLAUDE BERNARD

Interest of deep learning classifier for finance

MASTER 1 ECONOMETRIC AND STATISTICS

Claude Derrick N'GORAN
Marcelle Sarah KOUAMELAN
Gon'Yassouet Alain OUIN

Directed by: Aurélien Couloumy

Table of contents

*

Introduction

Several invention of engineers and researchers are inspired by real life: this is the case for example of the plane which hovers in the air without flapping its wings but which undoubtedly resembles a bird. The best known center or medium for learning in living things is the brain, which contains a multitude of neurons which transmit information to each other in the form of an electrical pulse. That said, it would be very laborious to want to reproduce an exact brain because of its extreme complexity. However, the engineers have programmed an abstraction of its functioning that is called "neural networks" which constitute the basis of "deep learning". “Deep learning” is a branch of artificial intelligence, it is a vast field allowing through certain methods to make robotic programming, predictions on a data set (forcasting), linguistic translation, simulation and many others. In the case of forcasting, for example, one method used is “supervised learning” which consists of separate the data into two parts, one for learning the model and the other to test the validity of the model. Still within the framework of forcasting, there are areas today where correct prediction remains a boon for the protagonists. One of these remains the stock market, which represents the financial market on which securities and commodities are traded. It is the place where shares and bonds issued by various publicly traded companies are sold and bought by investors. For example, securities have a share price that defines their purchase and sale price. We find on the stock market retail investors, that is to say they invest for their personal account, but also professionals (banks, management company, financial advisors and many others) who intervene either for clients or on behalf of the company in question. Therefore, any technique, any method, predicting with precision and subject to validation of these by statistical criteria the future prices of a financial product would constitute a rare pearl for any investor. These have as sole objective to grow their investments. With this in mind, it is legitimate to ask ourselves how neural network models could help decision makers in the financial market. Thus, the purpose of our Study and Research Work would be to answer this question. To do this, we will first present the neural networks, their compositions and their functions. Then some peculiarities of these neural networks such as Recurrent Neural Networks, and LSTM. Then we will conclude with an application of a neural network-based model to a stock index (the CAC40) to predict its future prices.

Chapter 1

Presentation of neural network

In this first part of our work shall present the neural network, present how are this neural networks made up or implemented and how they work.

1.1 The Architecture of a Neural Network

Neural networks are interconnected neurons belonging to different functional groups that we will call families here. Figure 1.1 show that we have three types of families: inputs, hiddens layers and outputs. The input family receives the data which it transmits as it stands to the first layer of the hiddens layers. The first layer of the family of Hiddens layers modifies these values and sends them to the next layer of neurons and the output layer modifies the values lastly to provide the final answer, it is the prediction of our neural network based on the data it received. The Hiddens Layers family can contain a large number of layers of neurons, which is where the "deep" designation of deep learning comes from.

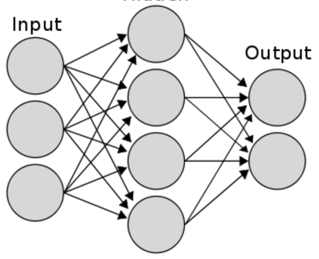


Figure 1.1: Types of neural layers

1.2 The FEEDFORWARD process

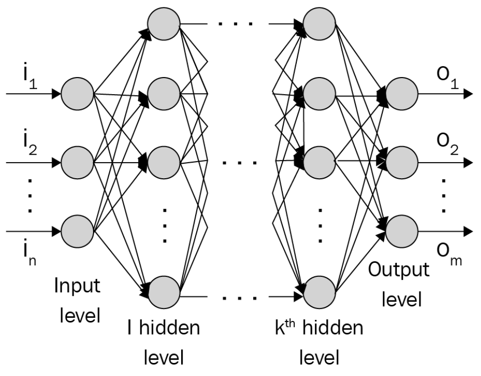


Figure 1.2: Feedforward process

Going through these three different operating groups presented in Figure 1.2, is called the "feedforward" process. It's going forward this way that gives the name feedforward to this process. It is therefore a question of knowing in more detail how these values are modified and brought to the next layer of neurons. Very simply, values are put into the input layer, they pass through the Hiddens layers which identify the data and the required transformations. Then, a result is provided as output which will be a first result. In more detail, information is given to our neural network as input according to the type of data chosen. Each neuron of the input family is linked by a cable or a synapse to each of the neurons of the first layer of the hiddens layers so that all the information is taken into account.

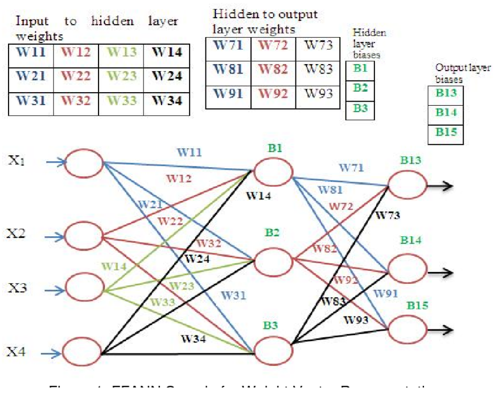


Figure 1.3: Highlighting weight and bias

In Figure 1.3, we assign to the neurons weights (w) which are originally randomly generated. These weights may or may not increase the importance of the information given by the neuron depending on whether the weight is relatively large or small. So this is the weight w of the connection. It is thanks to the biases of the neurons, the values carried by the neurons and the different weights assigned to them that the pre-activations of the neurons of the next layer will be calculated in a linear fashion. These are values included in the set of reals R. We use the term pre-activation because thereafter this pre-activation value will be put into calculation in an activation function before delivering the information to the neuron of the following layer . These actions are repeated until the last layer of the Hiddens Layers as well as that of the outputs. This will produce our results at the output layer. It is obviously important to know why these activation functions are used and how they work.

1.3 Some Activation functions

Neurons send signals that are signal intensities to neurons in the next layer. In general a neuron receives activating and inhibitory contributions and adds its own bias. Then it calculates an activation level based on its preactivation value and an activation function. The activation function is used to add a non-linearity in the functioning of the neuron, it is also called thresholding or transfer function. What you need to know is that it is imperative to add non-linearity to the hidden layers because data classification is not always done linearly. There are several types of activation functions. Here we present some of the most common activation functions.

1.3.1 Sigmoid function

$$g(x) = \frac{1}{e^{-x}}$$

It represents the distribution function of the logistic law and it is often used in neural networks because it is derivable at all points. The sigmoid activation function reduces the preactivation value between 0 and 1. For example, if we want to predict that an individual is ill or healthy compared to certain characteristics represented by the input data, we can look at how well our model is sure to have the right prediction when it states "good health". The sigmoid function competent to produce us probability will reduce the preactivation value in the interval $[0,1]$ as shown in the graph below.

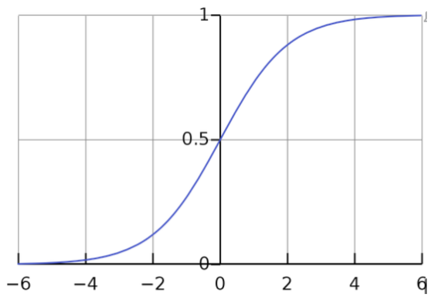


Figure 1.4: Sigmoid function

Obviously, the probability value 0.5, in Figure 1.4, corresponds to the pre-activation value 0. This means that the more the outgoing value of the sigmoid activation function approaches 1, the more we are sure of the quality of the information produced. In the case of our health example, a probability very close to 1 would mean that the prediction that the individual is in good health is almost certain. However, this function encounters limits when we have several outputs with different and mutually exclusive assigned roles. For example, recognizing a banana at the exit is not recognizing an orange or even a pineapple. Putting a sigmoid in activation at each of the output neurons in a model which is especially not sufficiently trained will give real meaningless results because the role of the neurons and implicitly the preactivation values are taken into account individually.

1.3.2 Softmax function

This function comes in some way to overcome the deficiencies of the sigmoid stated above when we have several different outputs. The softmax function does not only take into account the preactivation of a single neuron, but it simultaneously takes into account all the pre-activations so that the sum of all its activation values is equal to 1 to achieve mutually exclusive results.

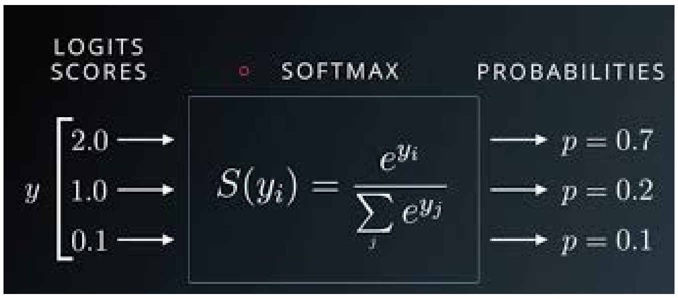


Figure 1.5: Softmax Function

In Figure 1.5,we can see that we would obtain, as in the previous example, a banana with probability a, an orange with probability b and a pineapple with probability $1 - a \cdot b$, a and b belonging to the interval $[0,1]$.

1.3.3 Hyperbolic tangent function (Tanh function)

$$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The function reduces the values in the interval $[-1,1]$

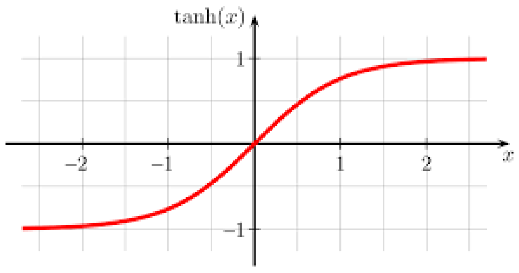


Figure 1.6: Hyperbolic tangent Function (tanh)

The advantage of using the Tanh function (show in Figure 1.6) is that it suffers less from the vanish gradient problem which we will explain later in the backpropagation process.

1.3.4 RELU function

$$g(x) = \max(0,x)$$

For this function, when the activation value is below 0 it provides 0 as the activation value and when it is above 0 then the activation is linear. However, it is well regarded as a non-linear function. The RELU function is most often used for activation of Hiddens Layers neurons.

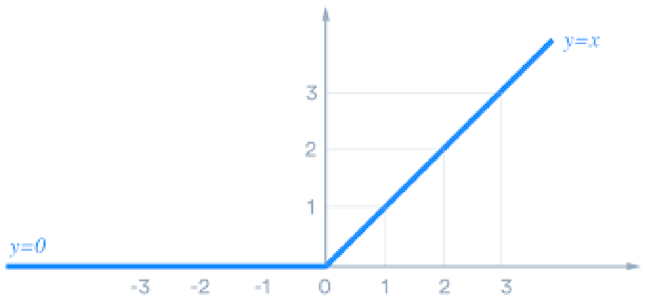


Figure 1.7: RELU Function

The RELU function, presented on Figure 1.7, has a very good convergence properties due to the gradient which does not vanish. In short, the choice of a type of activation function simply depends on the type of signals the neuron receives, the type of layer in which it is located, the range of values that one would like to have in output, or if you are in Hidden Layers you must introduce non-linearity by choosing the activation value that will have the best convergence properties or keep a linear activation function if it is for example an amount to be predicted and etc. it is about essentially to choose the most suitable and effective activation functions to train our model. We remember that all of the above is done for learning on examples of which we know the answer, it is indeed a training set. And, it is necessary to check if the answers given in outputs correspond to reality.

1.4 Error Function

The error function plays a very important role in that it allows to evaluate in some way the performance of the model more than it judges the veracity of the information provided at the exit of our neural network. More explicitly it evaluates the gap between the true value, that is, the desired value at the exit of the neural network and the value provided at the exit of our neural network. Note that there are several types of error function but this is the one we present, probably the most known of all:

$$E = \frac{1}{2} \cdot (y - t)^2$$

where:

y =prediction of our model;

t = the true value or desired value

This everything is squared so as to have a positive value and so that the most important errors are much more highlighted. Indeed there is more potential in improving what works less well than what works almost well. It is noted that the multiplication by 0.5 of this value is made to simplify the calculations later.

It is important to have the smallest possible error value to be sure to get as close as possible to the truth in our prediction, so we try to minimize the outgoing value of our error function: the closer it is to zero, the better our model is. It is almost certain that the output does not give us in the first place the desired value, it can have a not insignificant error. When the error is great it is necessary to make a return and try to modify the weights of the network to minimize the error and this is what we will call the “backpropagation” all this being part of the learning process of our prediction model.

1.5 The backpropagation

Backpropagation consists of going back to repair errors related to the bad or not very good prediction of our model. The idea is to know the impact of the modification of a neuron's weight on the error while always seeking to minimize the value of the error to have the most adequate result at the output.

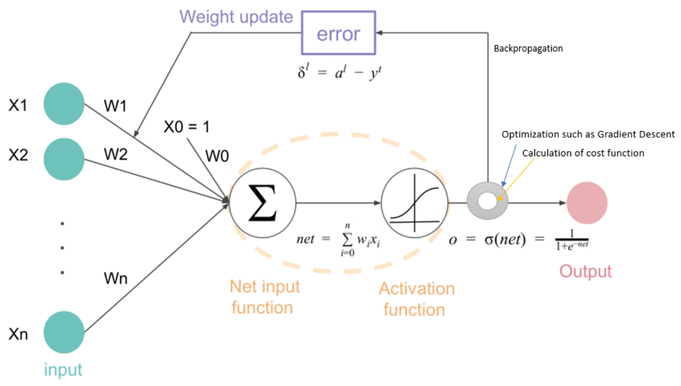

Weight update
error
$\delta^l = a^l - y^l$
Backpropagation
X1
X2
Xn
W1
W2
Wn
X0 = 1
W0
Optimization such as Gradient Descent
Calculation of cost function
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Net input function
Activation function
Output
input

Figure 1.8: Backpropagation

As we can see on Figure 1.8, Gradient backward propagation is a method for calculating the error gradient for each neuron in a neural network, from the last layer to the first. This gradient is obtained by the partial derivative of the error function and it allows us to know how to modify the weights of the neuron to be able to minimize the error. The algorithm is iterative and the correction applies as many times as necessary. By applying this step several times, the error tends to decrease and the network offers better prediction.

Everything that we have described previously is part of a particular case of networks called unidirectional networks, that is to say whose signals go in one direction, from left to right. However there are some that loop this is what we will call recurrent neural networks (RNN)

Chapter 2

Theory, literature review

This part will focus the theory on neural networks in general , the recurrent neural network (RNN) and the long short time memory (LSTM) theory, will briefly presents how they are used and will end with a short presentation of some limits.

2.1 Recurrent Neural Networks

Recurrent neural networks are a class or type of neural network that allows past predictions to be used as input, hence the word loop or recurrence. In order to better understand this type of neurons it would be good to understand how they are implemented, especially within the hidden layers.

2.1.1 Configuration and implementation

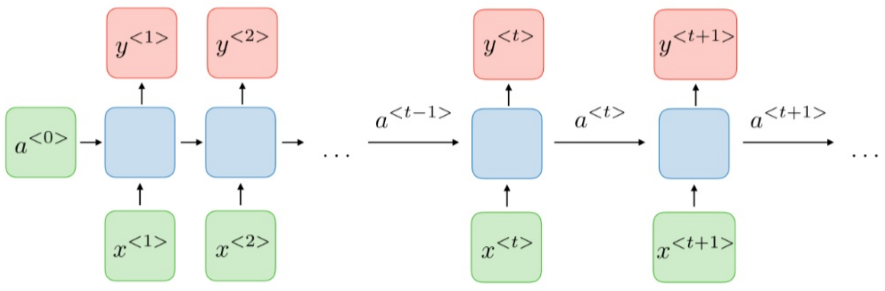


Figure 2.1: Configuration of Recurrent Neural Networks

Its structure,on Figure 2.1, is different from that of the simple neural networks described above. The following figure shows how the cells of the Hiddens Layers are implemented, this will lead us to better understand the functioning of the RNNs.

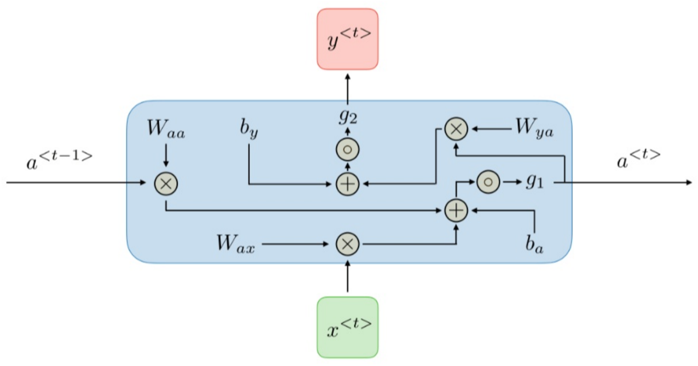


Figure 2.2: Implementation Recurrent Neural Networks

Indeed, Figure 2.2 shows that the a^{t-1} information of the activation of the previous neuron arrives, the neuron assigns it a W_{aa} weight, it is the same for the x^t data entering at the moment t, the neuron assigns it a W_{ax} weight then it adds these two elements each of them multiplied by its weight and adds a b_a bias. This is what constitutes the preactivation value that is transformed by the activation function g_1 according to the chosen type. This first value a^t of activation outgoing is transmitted directly to the next neuron and then, as in a simple neural network, the neuron multiplies this value of activation by a weight W_{ya} , adds its bias by to constitute the preactivation value of the neuron which is also transformed b_y a g_2 activation function and provides y^t output.

The activation process does not fundamentally change from that of simple neural networks but the structure of the network is not the same the pre-activation at time t is calculated by adding the information of the activation of the prior time $t - 1$. However, the activation functions remain the same. Recurrent neural networks are mainly used in the fields of automatic natural language processing and speech recognition. Let's take a look at the types of RNN we have.

2.1.2 Types of Recurrent Neural networks

We are going to see that we can have many types of Recurrent neural networks and each of these types are used for different purpose.

- ONE TO ONE

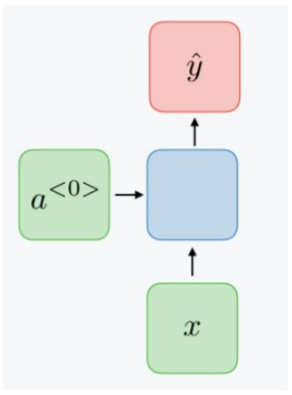


Figure 2.3: Type One-to-One

These, on Figure 2.3 do not differ from the traditional neural networks presented previously.

- ONE TO MANY

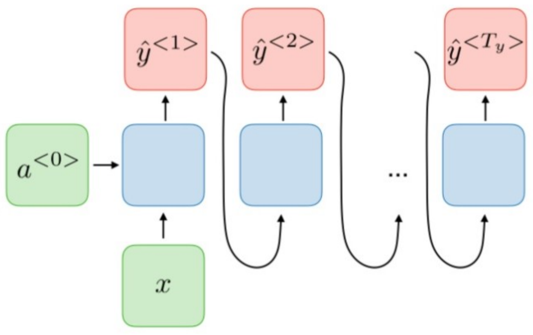


Figure 2.4: Type One-to-Many

The particularity of this type of network (Figure 2.4) is that from an input, it is capable of generating an output series. This is the case, for example, with the proposal for a "series of words" given by our telephone when we start a sentence. Obviously more and more these propositions agree with our expression already written in the past.

- MANY TO ONE

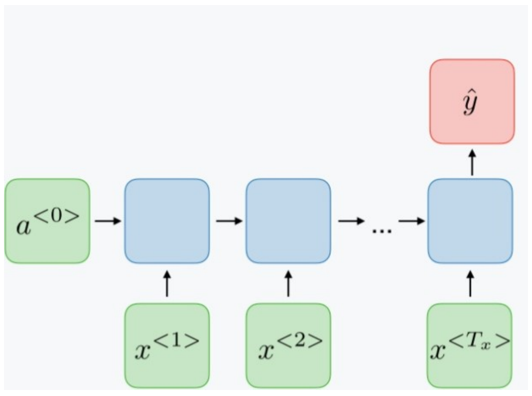


Figure 2.5: Type Many-to-One

As presented above, on Figure 2.5 from multiple inputs the network is able to predict a single output. It allows for example to classify feelings, to make a deduction from a set of elements.

- MANY TO MANY

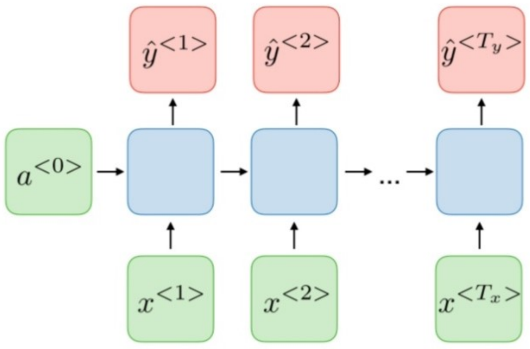


Figure 2.6: Type Many-to-Many($T_x = T_y$)

This first category of the many to many type presented above (Figure 2.6) is obviously with as many inputs as outputs. It can express recognition of an entity.

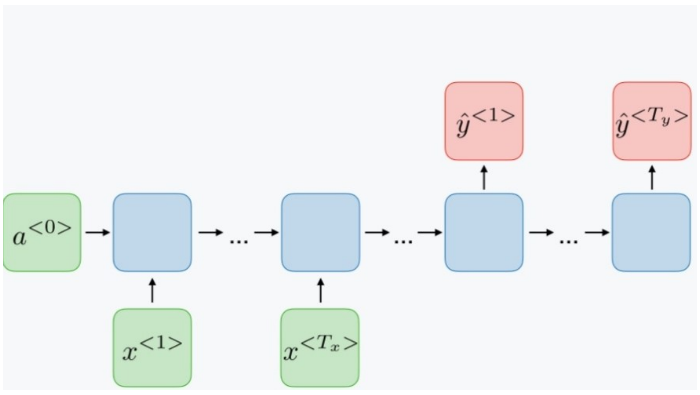


Figure 2.7: Type Many-to-Many($T_x \neq T_y$)

This second category of many to many types illustrated on Figure 2.7, shows that there can also be a difference in number between the inputs and the outputs. It can express a machine translation. For example, in a publication of BFM.TV by Cecile Bolesse, “Google translation” offers the merit of its improvement to a neural network system more precisely to Neural Machine Translation (NMT) which “uses context to determine the most accurate translation relevant, which it rearranges and then adjusts to get closer to human language with correct grammar, "says Google. Concretely, this amounts to feeding the machine with words, sentence segments, text already translated, etc.

2.1.3 Temporal loss function and backpropagation

Just like in simple neural networks, we need to quantify our deviation from the truth by the time-dependent error function formulated like This :

$$L(\hat{y}, y) = \sum$$

$^{T_y} L(\hat{y}^t, y^t)$

$$\frac{\partial L^{(T)}}{\partial W} = \sum^{T} \frac{\partial L^{(T)}}{\partial W}(t)$$

$$W \longleftarrow W - \alpha \frac{\partial L(x,y)}{\partial \omega}$$

As Shown in the two previous functions, the backpropagation process remains the same and the gradient is always obtained by making the first derivation of the error function applied to the time dimension. However we decided to expose here the problems of Vanish and Exploding Gradient because it is the recurrent neural networks which suffer the most.

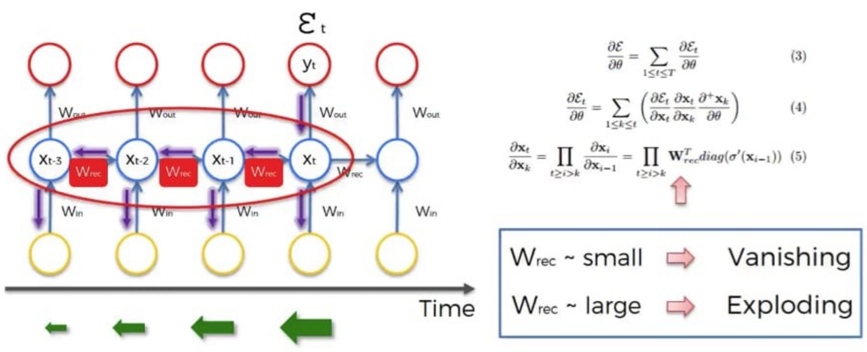

$\mathcal{E}_t$
y_t
W_{out}
X_{t-3}
X_{t-2}
X_{t-1}
X_t
W_{rec}
W_{in}
Time
$\frac{\partial \mathcal{E}}{\partial \theta} = \sum_{1 \leq t \leq T} \frac{\partial \mathcal{E}_t}{\partial \theta}$ (3)
$\frac{\partial \mathcal{E}_t}{\partial \theta} = \sum_{1 \leq k \leq t} \left(\frac{\partial \mathcal{E}_t}{\partial \mathbf{x}_t} \frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} \frac{\partial^+ \mathbf{x}_k}{\partial \theta} \right)$ (4)
$\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{x}_i}{\partial \mathbf{x}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{rec}^T diag(\sigma'(\mathbf{x}_{i-1}))$ (5)
W_{rec} ~ small ⇨ Vanishing
W_{rec} ~ large ⇨ Exploding

Figure 2.8: Vanishing and exploding Gradient problem

The weights of the neural network receive, on Figure 2.8 an update proportional to the partial derivative of the error function with respect to the current weight at each iteration of the training. However, the gradient can be very small and prevent the weight from changing its value effectively. It is these very small gradient values that multiply between them will induce insignificant changes in the weight of the neurons and cause the vanish gradient problem. It can happen in extreme cases that this prevents the neuron network from following its formation. Similarly, when gradient values are very high, this can cause unreasonable changes in the weights of our neurons resulting from an exponential increase in the gradient. This is called the gradient exploding problem.

2.1.4 Some solutions to vanish and exploding gradient problems

One technique used to mitigate the exploding gradient problem is the "gradient clipping". This technique is to cap the value that can be taken by the gradient so that it does not exceed a certain threshold.

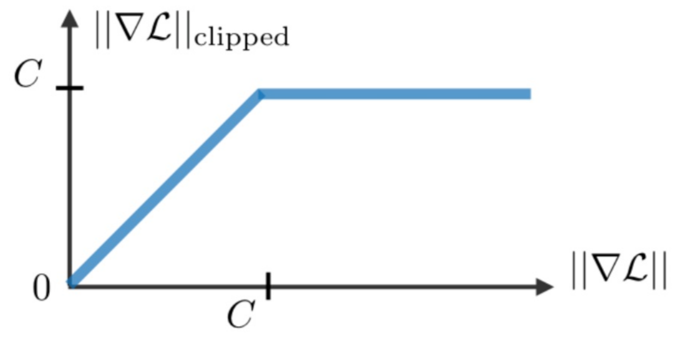


Figure 2.9: Gradient Clipping

The activation functions, as we can see on Figure 2.9, do not change from those seen previously. The bottom line is that it is imperative to take the vanish gradient problem into account when choosing the activation function depending on the nature and type of data that we are trying to predict. For example when you don't necessarily need to have a probability in the output you can very well use the tanh function which suffers less from the Vanish Gradient problem. Also in practice, the problem is not seen with a low number of sequences but the higher the number of sequences the greater the problem of vanish gradient becomes important. For example with our study database it seems obvious from theory that using a recurrent neural network would lead to a weighty vanish gradient problem. we present below a solution to overcome the vanish Gradient problem: Long short time memory (LSTM).

2.2 LSTM

2.2.1 A Little story

Since machine learning has powerful mathematical tools, and high-performance machines, the famous problems of backward propagation (Vanish gradient problem), which are very important to fix to get better results, are more likely to be solved. Indeed, it was in 1997 that the neural network, Long Short Term Memory LSTM, was conceptualized by Sepp Hoch Hochreiter and Jurgen Schmidhuber, two German computer scientists, in order to be able to solve or rather control this problem. Thus, LSTM are generally used on time-sorted data. This allows to have accurate informations about the past, and the present, in order to predict the future.

2.2.2 Définition: LSTM

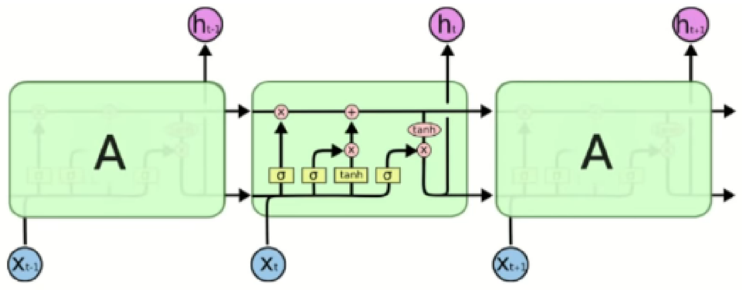


Figure 2.10: LSTM

LSTM, which means Long Short-Term Memory, is a cell composed of three «doors»: these are calculation areas that manipulate a certain flow of information (by carrying out specific actions). Then, compared to a cell of a classical neural network, a LSTM cell includes one more state, or used one more vector that corresponds to the memory (hidden state). It is noted C_t. Thanks to this new state the cell will be able to forget information that would not be necessary for prediction.The middle cell on Figure 2.10 is a complete LSTM cell. Let's decompose her.

2.2.3 LSTM cell components:

A LSTM cell presents 3 main operations:

- Forget gate
- Input gate
- Output gate

2.2.3.1 Forget gate

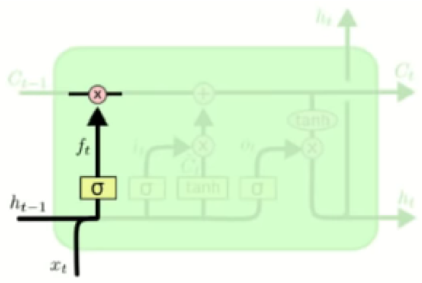


Figure 2.11: Forget Gate

The forget gate will filter the information contained in the previous memory cell. The purpose of this gate is to keep only memory cell' s relevant information. To sort these information, we will use different mathematical calculations as we can see on Figure 2.11. We can detail the different functions showed on this Figure:

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

where:

σ =Sigmoid function;

x_t = incoming data in the neural network,

h_{t-1} = output value, predicted by the previous LSTM layer.

$[h_{t-1} - x_t]$ = concatenation of the x_t and h_{t-1} vectors.

W_f = weight of neurons

b_f = bias (degree of additional freedom) of a neural network

f_t = set of values between 0 and 1.

To determine the values of the previous memory cell that will be retained, simply multiply f_t and C_{t-1}. The values contained in C_{t-1} retained, will be those whose multiplication quoted before, will give a value (output value) as close as possible to 1. Otherwise, the information contained in C_{t-1} will be ignored. Then, all values whose multiplication with f_t will be close to 1, will be stored.

$$C_t = f \times C_{t-1}$$

where: C_{t-1}=old Cell memory (time $t-1$);

2.2.3.2 Input gate

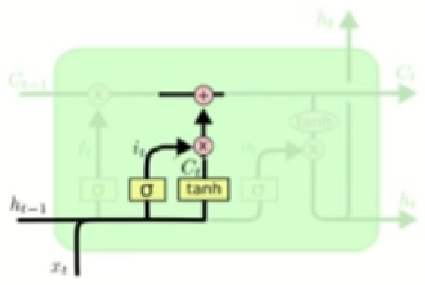


Figure 2.12: Input Gate

Now, it's the time to collect new information. The first step completed, the second would allow to sort, according to their relevance, the new information collected (decide to store or not a new information in the memory cell), corresponding to the x_t (incoming values of the LSTM). These will be used for decision making for time t and maybe for time $t + 1$, if, obviously, the condition given by the forget gate is fulfilled. For this purpose, we will need mathematical formulas as for the previous door and as we can see on Figure 2.12:

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

$$\tilde{C} = \tanh(W_c \cdot [h_{t-1} - x_t] + b_c)$$

i_t corresponds to a set/vector of values always between 0 and 1. The sigmoide function is used here again because it will decide which new entries are likely to be stored in the memory cell.

$\tilde{C}$ contains all new values that can be stored in the memory cell. The principle is then practically the same as forget gate's. The multiplication between i_t and $\tilde{C}$ is performed. The values of $\tilde{C}$, whose multiplication with their corresponding in the array i_t will be close to 1, will be stored as a new value in the memory cell. But Otherwise, corresponding values will be blocked. This causes the memory cell update.

The new memory (which corresponds to (+) on the picture) will be:

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_{t-1}$$

2.2.3.3 Output gate

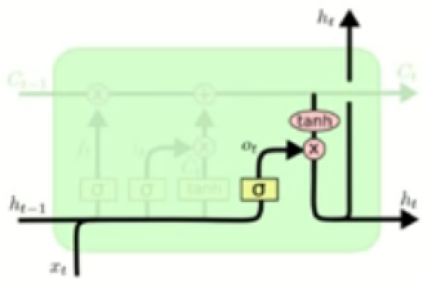


Figure 2.13: Output Gate

The output gate is the time when a h_t decision, as we can see on Figure 2.13 is provided to the present LSTM block and the next. It defines the state of the cell at time t:

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

$$h_t = \tanh(o_t \times o_t)$$

It depends on the information present in the C_t memory cell and the input information x_t in the LSTM layer.

These are the neural networks in general, the RNN and the LSTM that we felt necessary to understand in order to better understand our empirical work. However, it must be remembered that these are not the only types of neural networks existing and that we have among other things, for example, the convolutional neural networks (CNN) that are more used in image classification and therefore take images as input.

In practice, setting up these models can be problematic. It is therefore these limits that we try to enumerate in the next point.

2.3 Limits of Neural Networks

Neural networks have been subject to several rejections over the "decades" which have led researchers to improve their functioning better and to develop methods that could overcome the limits that were once the cause of rejection. However, these models still have limits that we set out below:

- We need a real case database which serves as a learning base however in many cases, there is not enough data to get to learn our model and optimally predict or collect data sufficient quantity is very expensive.
- Unreasonable computing times.
- Impossibility of taking into account important information from the future which could help us to get closer to the truth in our predictions.

However, thanks to the work of other individuals, we have had the opportunity or we would even say the chance to be able to access more or less substantial data that cover a long period in order to build our learning database. This is what we present to you in the rest of the work.

Chapter 3

Methodology and presentation of our data

3.1 Presentation of the dataset

The dataset on which we will work contains data relating to the stock price CAC40 on the period from 16-04-1998 to 15-04-2020 harvested on ***abc bourse***. CAC40 is the main stock market index of France. It's determined on the basis of continuous prices on the market of forty stocks among the hundred companies whose exchanges are most abundant on Euronext Paris. It is a representative index of the national economy. So, doing a technical analysis of past prices in order to predict future prices would be a very interesting study. And even more so in the context in which we are where the crisis which originally caused health has really impacted all areas on a global scale.

3.2 Description of the approach

Forcasting stock prices is a complex problem in financial domain and it need particular methods to solve it. Our approach will be based on the using of **Recurrent Neural Networks (RNN)** with basic cells **LSTM** in order to make forcasting. Indeed, Neural networks are well known for their efficiency with multiple input variables. It's a slightly different approach to technical analysis, usually applied which consists in analyzing the graphs as well as certain indicators to predict if we will have price hause or decline in the future. Application will make with **Keras**, popular python library for deep learning

3.3 Librairies

We, here, are Loading dispensable libraries

[1]:
```
# Importing used libraries

import numpy as np
import pandas as pd
import math
import sklearn.metrics
import sklearn.preprocessing
import matplotlib.pyplot as plt
import seaborn as sns
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
```

```
Using TensorFlow backend.
```

3.4 Analyses, descriptives statistics

[2]:
```
# Loading data set
data = pd.read_csv("cac40.txt", sep = ";", names = ["symbol", "date", "open",␣
 ↪"high", "low","close", "volum"])
```

[3]:
```
# Sanity check to ensure non null values
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5616 entries, 0 to 5615
Data columns (total 7 columns):
symbol    5616 non-null object
date      5616 non-null object
open      5616 non-null float64
high      5616 non-null float64
low       5616 non-null float64
close     5616 non-null float64
volum     5616 non-null int64
dtypes: float64(4), int64(1), object(2)
memory usage: 307.2+ KB
```

Interpretation : We haven't missing values. It's good we can continue.

[4]:
```
# five first columns
display(data.head(5))
# five last columns
display(data.tail(5))
```

```
         symbol      date     open     high      low    close  volum
0  FR0003500008  16/04/98  3895.44  3895.44  3821.92  3845.94      0
1  FR0003500008  17/04/98  3820.36  3874.03  3807.76  3861.58      0
2  FR0003500008  20/04/98  3914.38  3916.07  3872.59  3885.69      0
3  FR0003500008  21/04/98  3838.29  3862.09  3822.54  3860.40      0
4  FR0003500008  22/04/98  3874.52  3886.09  3816.20  3835.07      0
```

```
            symbol      date     open     high      low    close    volum
5611  FR0003500008  07/04/20  4489.40  4527.60  4379.27  4438.27  4639126
5612  FR0003500008  08/04/20  4397.36  4442.75  4333.09  4442.75  3165581
5613  FR0003500008  09/04/20  4512.30  4543.69  4409.04  4506.85  4195923
5614  FR0003500008  14/04/20  4553.55  4577.84  4497.56  4523.91  3575705
```

```
5615  FR0003500008  15/04/20  4511.87  4525.50  4335.59  4353.72  3727813
```

The data set contains:

- **symbol :** It's the numerical identifiant of CAC40 on the financial market
- **date :** The date where the price was quatified.
- **open :** Price at openning of stock market outstandings
- **high :** Maximum price that we could get that day
- **low :** Minimum price that we could get that day
- **close :** Price at closing of stock market outstandings

The main variables that we will use are the variables which identify the differents steps(4) of CAC40

We can notice that the data are already ordered with respect to time

[5]:
```python
# Some descriptives statistics
data.describe()
```

[5]:
```
              open         high          low        close         volum
count  5616.000000  5616.000000  5616.000000  5616.000000  5.616000e+03
mean   4436.405881  4467.919268  4400.833583  4435.102612  3.327348e+06
std     904.683450   905.758938   904.000347   904.815494  1.831248e+06
min    2453.050000  2518.290000  2401.150000  2403.040000  0.000000e+00
25%    3731.637500  3760.600000  3699.812500  3729.597500  2.580918e+06
50%    4374.905000  4404.150000  4339.230000  4375.695000  3.274608e+06
75%    5149.067500  5182.890000  5116.270000  5145.295000  4.105511e+06
max    6929.050000  6944.770000  6838.700000  6922.330000  1.585276e+07
```

3.5 Data Visualization

Then, we will see, how the data are grouped.

[6]:
```python
df = data.iloc[:, 2:-1]
plt.figure(figsize=(15, 8))
ax = sns.boxplot(data = df, orient="h", palette="Set2", linewidth= 2.5)
```

Figure 3.1: Box plot of different variables of the database

In **Figure 3.1** we can see that, globally, we have the same values repartition for our main variables.

[7]:
```python
# Grouping plot

plt.figure(figsize=(18, 6))
colors = ["windows blue", "amber", "greyish", "faded green"]
```

```
ax = sns.lineplot(data = df,style="event", color = colors)
xlabel_names = [list(data.date.values)[i*500] for i in range(7)]
tmp = ax.set_xticklabels(xlabel_names)
plt.show()
```

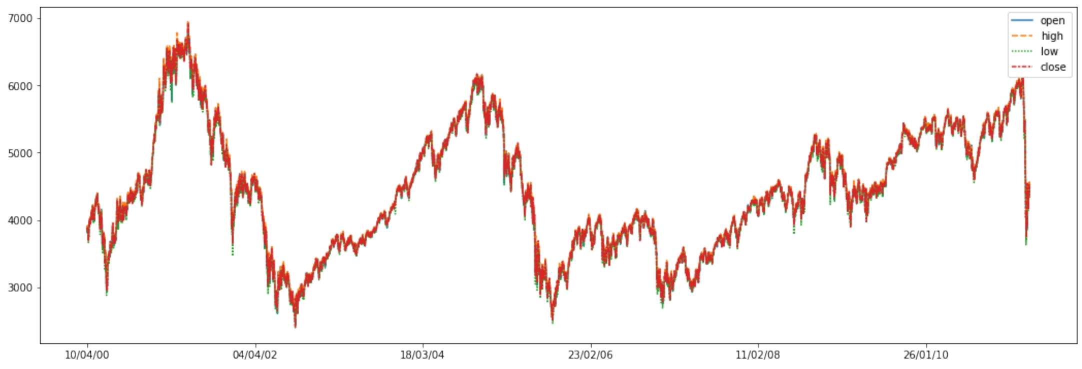


Figure 3.2: Plot of variables expressed in terms of dates

[8]:
```
# Some individuals plot
df2 = df.copy()
df2["date"] = data["date"]

plt.figure(figsize = (10, 4))
ax1 = sns.lineplot(data = df2["open"],color = "coral", label = "open")
xlabel_names = [list(df2.date.values)[i*500] for i in range(7)]
tmp = ax1.set_xticklabels(xlabel_names)
```

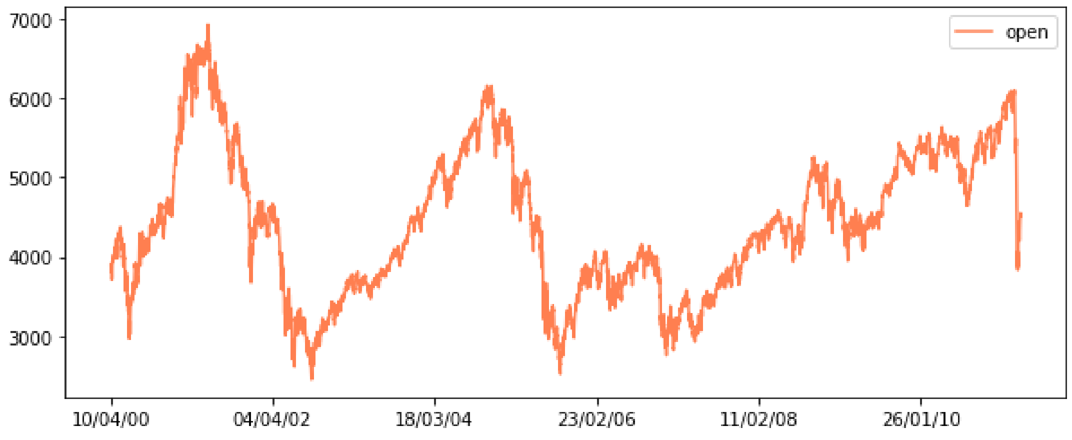


Figure 3.3: Plot of "Open" values expressed in terms of dates

[9]:
```
plt.figure(figsize = (10, 4))
ax2 = sns.lineplot(data = df2["close"],color = "green", label = "close")
xlabel_names = [list(df2.date.values)[i*500] for i in range(7)]
```

```
tmp = ax2.set_xticklabels(xlabel_names)
```

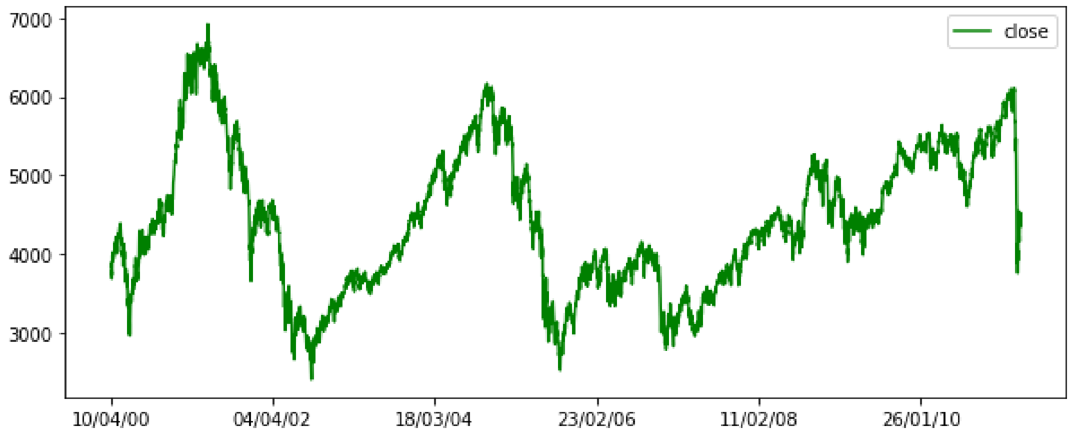


Figure 3.4: Plot of "Close" values expressed in terms of dates

Interpretations about above plots

- We can clearly see that, that structure in the data that is dependent on the time and not stationary. However it's stationaries data are easier to model and provide forcasting so we must stationarize the data set. One way is to differentiate the data. That is to say that the observation of the previous time step (t-1) is subtracted from the current observation (t). It gives a differentiated series, or changes to observations from one time step to the next. Differentiation can be done on the data set before splitting.
- Open, high, low and close prices have similar variation
- Previous plots show us a large price irregularity over time. So, we cannot envisage a conjecture of the price of future prices from a simple observation of the graph. Hence the importance of an LSTM model.
- Also the irregularity of the prices passed over time helps us because it makes learning robust. Indeed, if our model, once trained, it predicts with good precision future values knowing those that are earlier, we cannot hypothesize that the price variation has something to do with it. The single reason will be that our model is good.
- In 2020, due to the Covid19 crisis , we observe a decline until reaching a lower price than 22 years ago in 1998
- In 2008 we observe the sharply drop due to the subprime crisis.

3.6 Data preparation

[10]:
```
# Fonction pour normaliser les valeurs et les ramener toutes entre 0 et 1
min_max_scaler = sklearn.preprocessing.MinMaxScaler(feature_range=(0, 1))
def normalize_data(df):
    global min_max_scaler
    df['open'] = min_max_scaler.fit_transform(df.open.values.reshape(-1,1))
    df['high'] = min_max_scaler.fit_transform(df.high.values.reshape(-1,1))
    df['low'] = min_max_scaler.fit_transform(df.low.values.reshape(-1,1))
    df['close'] = min_max_scaler.fit_transform(df['close'].values.
↪reshape(-1,1))
    return df
```

The following function allows to create training, validation and test data sets. The following cutting will be done:

- 50% of original data set for training set
- 30% of original data set for validation set
- 20% du of original data set for test set

NB : Other splitting is possible (like 80% train set, 10% validation set and 10% test set)

Important : To make the LSTM model, we must devided our data set into inputs (X) and output (y) components. In fact the observation from the last time step (t-1) are the inputs and the obsrvation at the current time step (t) is the output.

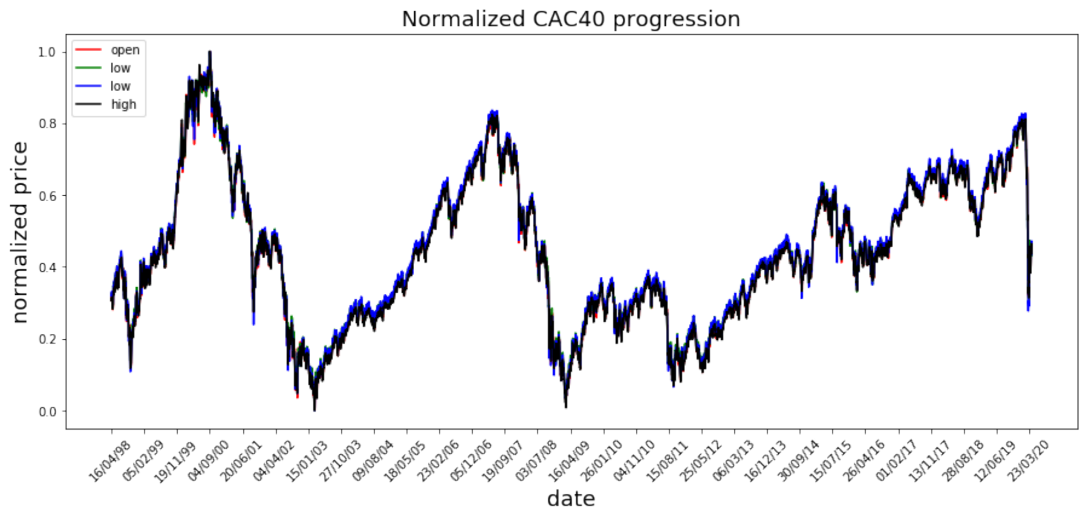
[11]:
```
def load_data(df, time_steps):
    dataset_np = df.values # convert to numpy array
    dataset = []

    # create all possible sequences of length time_steps
    for index in range(len(dataset_np) - time_steps):
        dataset.append(dataset_np[index: index + seq_len])

    dataset = np.array(dataset)
    validSetSize = int(np.round(0.3*dataset.shape[0]))
    testSetSize = int(np.round(0.2*dataset.shape[0]))
    trainSetSize = dataset.shape[0] - (validSetSize + testSetSize)

    X_train = dataset[:trainSetSize,:-1,:]
    Y_train = dataset[:trainSetSize,-1,:]

    X_valid = dataset[trainSetSize:trainSetSize+validSetSize,:-1,:]
    Y_valid = dataset[trainSetSize:trainSetSize+validSetSize,-1,:]

    X_test = dataset[trainSetSize+validSetSize:,:-1,:]
    Y_test = dataset[trainSetSize+validSetSize:,-1,:]


    #Reshaping data st with 3D array with sample size, time_steps and features
    Y_train = Y_train.reshape(Y_train.shape[0], 1, Y_train.shape[1])
    Y_valid = Y_valid.reshape(Y_valid.shape[0], 1, Y_valid.shape[1])
    Y_test = Y_test.reshape(Y_test.shape[0], 1, Y_test.shape[1])

    return [X_train, Y_train, X_valid, Y_valid, X_test, Y_test]


df_norm = df.copy()
columns = list(df.columns)
```

```
#Ensure all data is float
df_norm = df_norm.values
df_norm = df_norm.astype('float32')
df_norm = pd.DataFrame(df_norm)
df_norm.columns=columns

df_norm = normalize_data(df_norm)
# create train, test data
seq_len = 65 # choose time steps
X_train, Y_train, X_valid, Y_valid, X_test, Y_test = load_data(df_norm,␣
 ↪seq_len)
print('X_train.shape = ', X_train.shape)
print('Y_train.shape = ', Y_train.shape)
print('X_valid.shape = ', X_valid.shape)
print('Y_valid.shape = ', Y_valid.shape)
print('X_test.shape = ', X_test.shape)
print('Y_test.shape = ', Y_test.shape)
```

```
X_train.shape =  (2776, 64, 4)
Y_train.shape =  (2776, 1, 4)
X_valid.shape =  (1665, 64, 4)
Y_valid.shape =  (1665, 1, 4)
X_test.shape =  (1110, 64, 4)
Y_test.shape =  (1110, 1, 4)
```

Besides, it's important to scale and normalize the data set. Because: * the activation function's of the network's neurons such as **relu** are defined on the [0, 1] interval * But when you see above the describe of the data set we don't have values on this interval. So it's necessary to scale and normalize data to allow of neural networks to have good performance

[12]:
```
# Normalizing the data and representation

plt.figure(figsize=(15, 6))
plt.plot(df_norm.open.values, color='red', label='open')
plt.plot(df_norm.close.values, color='green', label='low')
plt.plot(df_norm.low.values, color='blue', label='low')
plt.plot(df_norm.high.values, color='black', label='high')
plt.title('Normalized CAC40 progression', fontsize=18)
plt.xticks(range(0,data.shape[0],200),data['date'].loc[::200],rotation=45)
plt.xlabel('date',fontsize=18)
plt.ylabel('normalized price',fontsize=18)
plt.legend(loc='best')
plt.show()
```


Normalized CAC40 progression
open
low
low
high
normalized price
date
1.0
0.8
0.6
0.4
0.2
0.0
16/04/98
05/02/99
19/11/99
04/09/00
20/06/01
04/04/02
15/01/03
27/10/03
09/08/04
18/05/05
23/02/06
05/12/06
19/09/07
03/07/08
16/04/09
26/01/10
04/11/10
15/08/11
25/05/12
06/03/13
16/12/13
30/09/14
15/07/15
26/04/16
01/02/17
13/11/17
28/08/18
12/06/19
23/03/20

Chapter 4

Presentation of our results, analysis and interpretation of our results

We use an encoder-decoder LSTM because it can be used for prediction problems where there are both input and output sequences. Also called sequence-to-sequence, this model is used for multi-step time series forcasting

First we define the encoder. Then the ouput of the encoder is used by the decoder as an input.

The output of the encoder is repeated for each time step in the output. But in our case we want one time step in the output Hyperparameters defined like:

- the number of neurons
- the time steps in the inputs
- return_sequences = True, which determines whether to return the last output in the output sequence, or the full sequence
- the features which are the steps of our stock price

The final output is a vector with a number of columns equal to the stock price features. During compilation, we need to specify a loss function and optimization algorithm. We used **mean_squared_error** as the loss function and the efficient **adam** optimization algorithm

[13]:
```
# Architecture of the model

# hyperparameters
n_steps = X_train.shape[1]
n_features = X_train.shape[2]
n_neurons = 50

#Model implemention

model = Sequential()
model.add(LSTM(n_neurons, activation='relu', input_shape=(n_steps,␣
 ↪n_features))) #encoder
model.add(RepeatVector(1)) # make repatition
model.add(LSTM(n_neurons, activation = 'relu', return_sequences = True))␣
 ↪#decoder
model.add(TimeDistributed(Dense(n_features))) # fixing final output dimension
model.compile(optimizer = 'adam', loss = 'mae')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 50)                11000
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 1, 50)             0
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 50)             20200
_________________________________________________________________
time_distributed_1 (TimeDist (None, 1, 4)              204
=================================================================
Total params: 31,404
Trainable params: 31,404
Non-trainable params: 0
_________________________________________________________________
```

Important : On this following, we made the choice of epochs and batch size after some readjusments, given that epochs represent the number of times the data set is scanned and batch size, the size of observations group scanned in only one time.

[14]:
```
# Model fitting

n_epochs = 50
batchSize = 64

fit_model = model.fit(X_train, Y_train, epochs=n_epochs,␣
 ↪batch_size=batchSize, validation_data=(X_valid, Y_valid),
                     shuffle=False)
```

```
Train on 2776 samples, validate on 1665 samples
Epoch 1/50
2776/2776 [==============================] - 13s 5ms/step - loss: 0.2991 -
val_loss: 0.1185
Epoch 2/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.3815 -
val_loss: 0.1434
Epoch 3/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.1343 -
val_loss: 0.0657
Epoch 4/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0753 -
val_loss: 0.0457
Epoch 5/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0655 -
val_loss: 0.0446
Epoch 6/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0545 -
```

```
val_loss: 0.0462
Epoch 7/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0462 -
val_loss: 0.0443
Epoch 8/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0410 -
val_loss: 0.0382
Epoch 9/50
2776/2776 [==============================] - 8s 3ms/step - loss: 0.0368 -
val_loss: 0.0338
Epoch 10/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0345 -
val_loss: 0.0289
Epoch 11/50
2776/2776 [==============================] - 8s 3ms/step - loss: 0.0335 -
val_loss: 0.0261
Epoch 12/50
2776/2776 [==============================] - 10s 3ms/step - loss: 0.0361 -
val_loss: 0.0322
Epoch 13/50
2776/2776 [==============================] - 10s 3ms/step - loss: 0.0281 -
val_loss: 0.0246
Epoch 14/50
2776/2776 [==============================] - 11s 4ms/step - loss: 0.0249 -
val_loss: 0.0195
Epoch 15/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0284 -
val_loss: 0.0223
Epoch 16/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0284 -
val_loss: 0.0219
Epoch 17/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0256 -
val_loss: 0.0158
Epoch 18/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0232 -
val_loss: 0.0185
Epoch 19/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0243 -
val_loss: 0.0206
Epoch 20/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0299 -
val_loss: 0.0162
Epoch 21/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0231 -
val_loss: 0.0174
Epoch 22/50
2776/2776 [==============================] - 11s 4ms/step - loss: 0.0244 -
val_loss: 0.0183
Epoch 23/50
2776/2776 [==============================] - 13s 5ms/step - loss: 0.0250 -
```

```
val_loss: 0.0162
Epoch 24/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0237 -
val_loss: 0.0186
Epoch 25/50
2776/2776 [==============================] - 10s 3ms/step - loss: 0.0238 -
val_loss: 0.0162
Epoch 26/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0241 -
val_loss: 0.0214
Epoch 27/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0221 -
val_loss: 0.0180
Epoch 28/50
2776/2776 [==============================] - 11s 4ms/step - loss: 0.0260 -
val_loss: 0.0161
Epoch 29/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0206 -
val_loss: 0.0160
Epoch 30/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0231 -
val_loss: 0.0154
Epoch 31/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0202 -
val_loss: 0.0198
Epoch 32/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0298 -
val_loss: 0.0182
Epoch 33/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0226 -
val_loss: 0.0160
Epoch 34/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0239 -
val_loss: 0.0163
Epoch 35/50
2776/2776 [==============================] - 11s 4ms/step - loss: 0.0200 -
val_loss: 0.0188
Epoch 36/50
2776/2776 [==============================] - 10s 3ms/step - loss: 0.0304 -
val_loss: 0.0212
Epoch 37/50
2776/2776 [==============================] - 10s 3ms/step - loss: 0.0222 -
val_loss: 0.0139
Epoch 38/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0215 -
val_loss: 0.0158
Epoch 39/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0189 -
val_loss: 0.0198
Epoch 40/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0326 -
```

```
val_loss: 0.0226
Epoch 41/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0210 -
val_loss: 0.0140
Epoch 42/50
2776/2776 [==============================] - 10s 4ms/step - loss: 0.0203 -
val_loss: 0.0150
Epoch 43/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0226 -
val_loss: 0.0172
Epoch 44/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0214 -
val_loss: 0.0181
Epoch 45/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0285 -
val_loss: 0.0264
Epoch 46/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0195 -
val_loss: 0.0133
Epoch 47/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0203 -
val_loss: 0.0144
Epoch 48/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0206 -
val_loss: 0.0148
Epoch 49/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0229 -
val_loss: 0.0177
Epoch 50/50
2776/2776 [==============================] - 9s 3ms/step - loss: 0.0212 -
val_loss: 0.0184
```

The following line plot the test set(blue) and the predicted values(orange) providing the model skill:

[15]:
```
#Plotting the trainning progression
plt.figure(figsize=(18, 5))
plt.plot(fit_model.history['loss'], label = 'real test evolution')
plt.plot(fit_model.history['val_loss'], label = 'predicted test evolution')
plt.legend()
plt.title("Trainning progression")
plt.show()
```

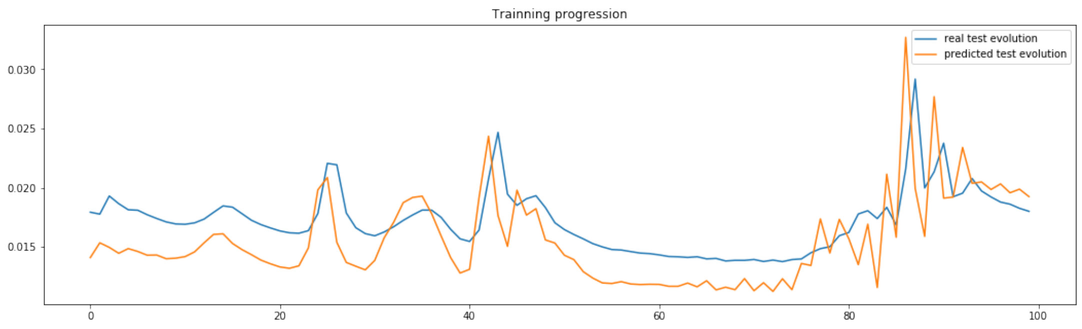


Figure 4.1: Training progression

Interpretation: At the start of fitting, the model doesn't predict precisly. But it is graduately converging in time.

Once the model is fit to the trainning data, we will use it to make forcasts. Indeed, we can predict the values of the validation data set and after that we will predict the values of test data set to confirm powerful and robutness of our model. This approch is called **fixed approch**

[16]:
```
#Make validation prediction
Y_hat = model.predict(X_valid)
```

Important : We must invert the scale on forecasts to return the values back to the original scale so that the results can be interpreted and a comparable error score can be calculated

[17]:
```
def reconstructSRData(X, y):
    inv_Y_hat = np.concatenate((X[0], y[0]), axis = 0)
    if len(X) > 1:
        for i in range(1, len(X)):
            tmp = np.concatenate((X[i], y[i]), axis = 0)
            inv_Y_hat = np.concatenate((inv_Y_hat, tmp), axis = 0)
    inv_Y_hat = min_max_scaler.inverse_transform(inv_Y_hat) # end of␣
 ↪reconstruction
    # But we must select only the values of the output y in one vector
    idx = X.shape[1] # to select the line corresponding of the output (y)
    inv_Yhat = list()
    for i in range(len(y)):
        inv_Yhat.append(list(inv_Y_hat[idx]))
        idx = idx + X.shape[1] + 1
    return np.array(inv_Yhat)

#invert scalling for forcast
inv_Yhat = reconstructSRData(X_valid, Y_hat)

#invert scaling for actual
inv_Y = reconstructSRData(X_valid, Y_valid)
```

```
print("Mean absolute error (MAE):  %f"% sklearn.metrics.
 ↪mean_absolute_error(inv_Y, inv_Yhat))
print("Mean squared error (MSE):  %f"% sklearn.metrics.
 ↪mean_squared_error(inv_Y, inv_Yhat))
print("Root mean squared error (RMSE):  %f"% math.sqrt(sklearn.metrics.
 ↪mean_squared_error(inv_Y, inv_Yhat)))
print("R square (R²):  %f"% sklearn.metrics.r2_score(inv_Y, inv_Yhat))
```

```
Mean absolute error (MAE):  83.130959
Mean squared error (MSE):  10974.038086
Root mean squared error (RMSE):  104.757043
R square (R²):  0.963560
```

To judge if our model is good we use statistical criteria such as R square, mean error, mean absolute error (MAE), or root mean squared error (RMSE). Therefore the model is pretty well trained based on the results(R square = 94%). But we will verify it on the test dataset.

[18]:
```
# Results representation

plt.figure(figsize=(18, 5))
plt.plot(inv_Y, label = "Actual")
plt.plot(inv_Yhat, label = "Predicted")
plt.legend()
plt.show()
```

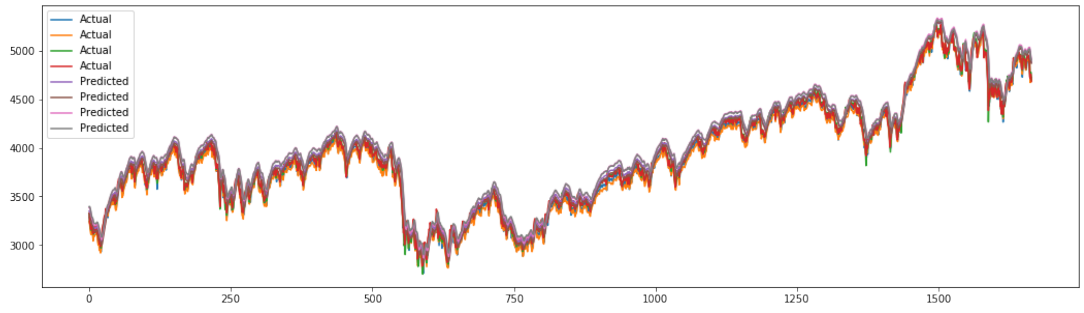


Figure 4.2: Results of prediction

Interpretation : Difficult to give clearly explanations because on this plot we have forcast and actual values of steps(4) of our stock index. but, we can still see that the model gives generally good answers.

[19]:
```
# More clarity with one step of the stock index
plt.figure(figsize=(18, 5))
plt.plot(inv_Y[:, 0], label = "Actual open value", color = "black")
plt.plot(inv_Yhat[:, 0], label = "Predicted open value", color = "red")
plt.legend()
plt.show()
```

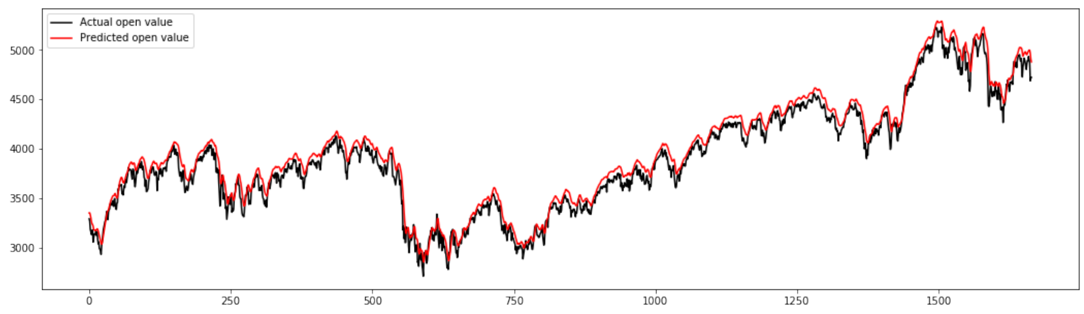


Figure 4.3: More clarity with one step of stock index

Interpretation :

With the openning value, we can notice that our model make effectively a good job. But it's now the truth moment, that is to say try our model with an unknow dataset (test dataset) for it.

[20]:
```
# Make test prediction
final_Yhat = model.predict(X_test)

#invert scalling for forcast
inv_finalYhat = reconstructSRData(X_test, final_Yhat)

#invert scaling for actual
inv_finalY = reconstructSRData(X_test, Y_test)

print("Mean absolute error (MAE):  %f"% sklearn.metrics.
 ↪mean_absolute_error(inv_finalY, inv_finalYhat))
print("Mean squared error (MSE):  %f"% sklearn.metrics.
 ↪mean_squared_error(inv_finalY, inv_finalYhat))
print("Root mean squared error (RMSE):  %f"% math.sqrt(sklearn.metrics.
 ↪mean_squared_error(inv_finalY, inv_finalYhat)))
print("R square (R²):  %f"% sklearn.metrics.r2_score(inv_finalY,␣
 ↪inv_finalYhat))
```

```
Mean absolute error (MAE):  152.733231
Mean squared error (MSE):  30170.271484
Root mean squared error (RMSE):  173.695917
R square (R²):  0.878997
```

Interpretation : According the results(almost 95 percent for the R square) our model is good and we can used it.

4.1 Simulated production

We take the last sixty four (64) values of the entire dataset as the input and we make the forcasting.

```
[21]: X_input = df_norm.iloc[(df_norm.shape[0]-64):df_norm.shape[0], :].values
      X_input = X_input.reshape((1, X_input.shape[0], X_input.shape[1]))

      # Prediction
      y_output = model.predict(X_input) # don't forget that this value is scale. We␣
       ↪must inverse it

      inv_y_output = reconstructSRData(X_input, y_output)
      print(inv_y_output[0])
```

```
[4403.816  4336.3    4425.4336 4419.9116]
```

Result : According our model, it's about the predicted values of 16 April 2020 for CAC40 (In this order open, heigh, low, close).

4.2 Results Conclusion

To finish, we can say that forcasting stock prices, a complex problem of financial domain can be solve by the neural networks. Precisly, for our caste study, we used the Encoder-Decoder LSTM model on CAC40 data set to make previsions. Generally we got good results and it could have been better if the model was more trained on powerfull and adapted computer for machine or deep learning. This application show us that, the proper use of neural networks on financial time series, allows to predict economic scenarios.

Conclusion

The goal of our research work was to understand how neural network models participate in the decisions of the financial market decision makers. To make that, first we wanted to seek to know more about neural networks and their specificities to know and understand the prediction model that would be most suitable for stock market data which are time series. Secondly, being equipped with the necessary tools we decide to apply LSTM on CAC40 data. These are daily opening, rising, falling and closing data from 1998 to 2020. The results obtained are very satisfactory with an adjustment coefficient $R - square$ of 95%. Thus they confirm the fact that neural networks are important for financial market players. In the sense that they allow them to make decisions that can turn out to be crucial. Although these methods are undoubtedly of great help in terms of prediction, there are still many areas for improvement for deep learning. If there is an improvement, wouldn't it be interesting to take it into account in our model and know its effects?

Summary

Artificial intelligence and especially deep learning is one of the most popular fields of study today. Supervised learning techniques have been improved considerably over the course of this century and offer increasingly satisfactory prediction results thanks to the different types of neural networks we have at our disposal. What is pleasing about this approach is that it is a process of continuous improvement. Its performance and its openness to all types of data justify its galloping solicitation in diverse areas. Among them, one of the subjects that we are focusing on is finance. The data used are those of CAC40, collected over a 22-year period, to which we apply a Long Short Time Memory (LSTM) model. For the implementation of our model, we use the Python programming language providing dedicated libraries for such a task. The results obtained following this application supported by certain statistical criteria are quite convincing. Thus showing the importance of the use of neural networks in finance and foreshadowing a future less concerned with certain economic scenarios sometimes unforeseen.

References

Related to the theoric part (Chapter 1 and 2) we were inspired by these articles and videos

https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-recurrents

https://stanford.edu/~shervine/l/fr/teaching/cs-229/pense-bete-apprentissage-profond

http://www.scilogs.fr/intelligence-mecanique/architecture-des-reseaux-de-neurones-reseaux-de-

https://www.youtube.com/watch?v=gPVVsw2OWdM&list=RDCMUCFMQG2aYndcIPHGNfmkTo_w&
index=6

https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels

https://hightech.bfmtv.com/logiciel/comment-google-traduction-est-devenu-presque-aussi-bon-q
html

https://blog.octo.com/les-reseaux-de-neurones-recurrents-des-rnn-simples-aux-lstm/

https://www.youtube.com/watch?v=VuamhbEWEWA

And for the application (on CAC 40 values), we were inspered by:

https://towardsdatascience.com/recurrent-neural-network-to-predict-multivariate-commodity-pr

https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

https://www.kaggle.com/raoulma/ny-stock-price-prediction-rnn-lstm-gru

https://www.kaggle.com/pablocastilla/predict-stock-prices-with-lstm

List of Figures

Appendices: How did we organize ourselves despite the health crisis?

We had the interest from the start to work on a subject related to data science and we therefore approach our current TER supervisor Mr Couloumy Aurelien. The first meeting went well and among these proposals we chose an area of interest which we then refined to retain our subject of TER. Concerning Project Management, the difficulties became apparent over time. First we set ourselves goals, and planned every step of the project from the start. Unfortunately, the health crisis having kept us away, we could not see ourselves as expected. That's when you realize that obviously this was not the best way to work. In the course of "Project Management", followed in the first semester, we learned that planning all the stages before developing it is completely bad for the success of the project, not being without surprises. We therefore decided to operate step by step while basing ourselves on the directives given in the month of January by Mr Denis Clos during the meeting-Review (presenting the specifications) and also by our tutor, Mr Aurélien Couloumy who can both be considered as potential customers to be satisfied, while remaining in our context of university project. We also learned that regular communication between the client and the project team is still necessary for the smooth running of the project, client satisfaction being paramount. In this sense we most often sought to contact Mr Couloumy, but the context surely of the health crisis, of confinement, did not facilitate contact with us. Most often there was no response from our supervisor, probably because he had to be overwhelmed by the situation. We had to adapt, knowing that at the very beginning of the project we had received recommendations from him about what he expected from us. The major part of our project being without contact with him we are not sure to be in complete phase with his request. Communication within the project team itself went pretty well, we worked in harmony and respect for each other's ideas despite controversial ideas. We used virtual means of communication such as Skype, Zoom and Whatsapps for scheduled meetings and to manage our urgent contingencies. In priority, in addition to the dropbox shared by the professor, we create a folder on google drive so that we can work together in real time according to predefined slots. We were therefore able to achieve our own objectives, the challenge being to satisfy the client.