MINES Saint-Étienne

Une école de l'IMT

서울대학교 SEOUL NATIONAL UNIVERSITY

TECHNICAL REPORT

Research on Dynamic Data Driven Forecasting for Financial Time Series Data

Auteur :

AKRIM Anass, Engineering Student ICM 2016, Mines Saint-Etienne

Supervisors :

BAY Xavier, Professor at Ecole des Mines de Saint-Etienne

KIM Tae-wan, Professor at Seoul National University

Saint-Étienne, France, September 2018

Contents

1 Introduction

1.1 Context

My internship consisted of a 3-month research internship in Deep Learning, within the engineering department of Seoul National University in South Korea. Working in pair, our supervisor of this department worked on problems related to the use of Deep Learning, and he asked us to learn and develop various algorithms using machine learning techniques in order to develop a robot that would perform automated trading on the stock market.

Our goal, was to learn and code various algorithms using machine learning and deep learning techniques (Moving Average, HMM and LSTM especially) in order to develop a robot that would automatically perform trading, using reinforcement learning on the stock market. It is interesting to know that these methods, deep learning in neural networks, are very recent and have been popularized thanks to Google's AlphaGo algorithm, namely the first robot able to beat the world champion at the go game. In fact, there are very few publications and articles on the subject since it is a recent field of research.

I was working in a research lab with 8 other people (Korean PhD researchers mainly), who were working on other Deep Learning's domains of application, while my colleague and I were the only ones working on Deep Learning applied to Finance. Even if it was in Korea and our colleagues were speaking mainly in Korean, the atmosphere at work was very friendly, not at all stressful and I did not encounter any particular difficulties during my work.

We had an appointment with our teacher every week to show him our work of the week (code or researches) and in return he gave us advices and other methods so that we could use them. In order to carry out our research, we mainly relied on publications and articles about deep reinforcement learning and LSTM, and we were able to code a trading algorithm based on techniques of prediction of like Moving Average and of 'Deep Learning' such as Neural Networks (LSTM), more complex and models difficult to tune and adapt to the sequential data used. The main work focus here is on **Deep Learning** (LSTM initially used only for **prediction**) and then combined with **Reinforcement Learning**.

1.2 Objectives and Methodology

In section III), we started exploring the Singular Spectrum Analysis decomposition of time series, seeing how we can filter our data.
In section IV), we study the moving average crossover method, forecasting and trading strategy, building an algorithm which can allow us making automatic trading, and evaluating our trading strategy.
In section V), we study the Hidden Markov Model technique and theory.
In section VI), we explore Deep Leaning in Neural Networks, studying Neural Networks, RNN and LSTM. Then we implement LSTM in Python, and see how we can tune the hyperparameters of our LSTM model.
In the last section, we talk about reinforcement learning and deep reinforcement learning, which we implement in Python with LSTM, building a trading strategy.

2 Filtering the Data: SSA decomposition

Singular Spectral Analysis (SSA) is a technique introduced by Vautard et al. which allows to obtain in a simple way a decomposition of the signal in components of tendency, seasonality (harmonics) and noise.

SSA decomposes a time series into a set of summable components that are grouped together and interpreted as trend, periodicity and noise.

SSA can be used to analyze and reconstruct a time series with or without different components as desired. For example, you could apply SSA to:

- construct a smoothed version of a time series using a small subset of its components; - investigate a time series' periodic components to understand the underlying processes that generated the time series; - reconstruct the original time series without its periodic components; - remove all trend and periodic components from the series, leaving just the 'noise', which may be meaningful in and of itself...

Unlike the commonly used autoregressive integrated moving average (ARIMA) method, SSA makes no assumptions about the nature of the time series, and has just a single adjustable (and easily interpretable) parameter.

2.1 Theory: Singular Spectrum Analysis Method

We can decompose the SSA method in X steps :

1) We first map the time series $F = \{f_0, f_1, \ldots, f_{N-1}\}$, to a sequence of multi-dimensional lagged vectors. Let us define an integer L, the window length such as $2LN/2$. Then we form a 'window' $X_i = \{f_i,\ f_{i+1}, \ldots,\ f_{i+L-1}\}$, for $i = 0, \ldots, NL$. We slide this window along the time series, forming our L-trajectory matrix $X = \{X_0, \ldots, X_{N-L}\}$ such as :

$$\mathbf{X} = \begin{bmatrix} f_0 & f_1 & f_2 & f_3 & \cdots & f_{N-L} \\ f_1 & f_2 & f_3 & f_4 & \cdots & f_{N-L+1} \\ f_2 & f_3 & f_4 & f_5 & \cdots & f_{N-L+2} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{L-1} & f_L & f_{L+1} & f_{L+2} & \cdots & f_{N-1} \end{bmatrix}$$

Figure 1: L-trajectory matrix.

2) Then we decompose the L-trajectory matrix X with a singular-value decomposition (SVD):

$$X = U\Sigma V^T$$

where

- U an L L unitary matrix containing the orthonormal set of left singular vectors of X as columns;

- Σ an L K rectangular diagonal matrix containing L singular values of X, ordered from largest to smallest;

- V is a K K unitary matrix containing the orthonormal set of right singular vectors of X as columns.

We have :

$$X = \sum_{i=0}^{d-i} \sigma_i U_i V_i^T$$

where σ_i is the i-th singular value, U_i and V_i are vectors representing the i-th columns of U and V, respectively, $d \leq L$ is the rank of the trajectory matrix. The collection U_i, σ_i, V_i will be denoted the i-th eigentriple of the SVD.

The rank of X, $d = \text{rank}X$ is the maximum value of i such that $\sigma_i > 0$. For noisy real-world time series data (especially in stock market), the trajectory space is likely to have $d = L$ dimensions.

3) Then we put it all together. We focus on reconstructing the components of a time series from its elementary matrices: we decompose the trajectory matrix and form its elementary matrices.

2.2 Application :

The general method of SSA is fine for a short and simple time series. However, for longer and more complicated time series, we seek a method that quantifies whether a reconstructed component $\tilde{F}_i$ can be considered separate from another component $\tilde{F}_j$, so we don't need to make grouping decisions by visually inspecting each $\tilde{F}_i$.

For two reconstructed time series, $\tilde{F}_i$ and $\tilde{F}_j$, of length N, and a window length L, we define the weighted inner product, $(\tilde{F}_i, \tilde{F}_j)_w$ as:

$$(\tilde{F}_i, \tilde{F}_j)_w = \sum_{k=0}^{N-1} w_k \tilde{f}_{i,k} \tilde{f}_{j,k}$$

where $\tilde{f}_{i,k}$ and $\tilde{f}_{j,k}$ are the k-th values of $\tilde{F}_i$ and $\tilde{F}_j$, respectively, and w_k is given by :

$$w_k = \begin{cases} k+1 & 0 \leq k \leq L-1 \\ L & L \leq k \leq K-1 \\ N-k & K \leq k \leq N-1 \end{cases}$$

The weight w_k simply reflects the number of times $\tilde{f}_{i,k}$ and $\tilde{f}_{j,k}$ appear in the Hankelised matrices $\tilde{X}_i$ and $\tilde{X}_j$, from which the time series $\tilde{F}_i$ and $\tilde{F}_j$ have been obtained.

The elements of W_corr (weighted correlation matrix d x d) are given by :

$$W_{i,j} = \frac{(\tilde{F}_i, \tilde{F}_j)_w}{||\tilde{F}_i||_w ||\tilde{F}_j||_w}$$

- If $W_{i,j} = 0$, $\tilde{F}_i$ and $\tilde{F}_j$ are w-orthogonal and the time series components are separable.
- If $W_{i,j} \geq 0.3$, it indicate that the components may need to be grouped together.

We apply our SSA decomposition. We choose one of the common benchmarks for the U.S. stock market, the Standard Poor's 500 index (SP 500) to implement our model. The SP 500 was monthly data from January 1980 to August 2018 and was taken from finance.yahoo.com

We first fix L= 0.7*($\frac{N}{2}$) = 2853 here.

Then we plot the W-correlation of the time series components (only the first 30 ones here):

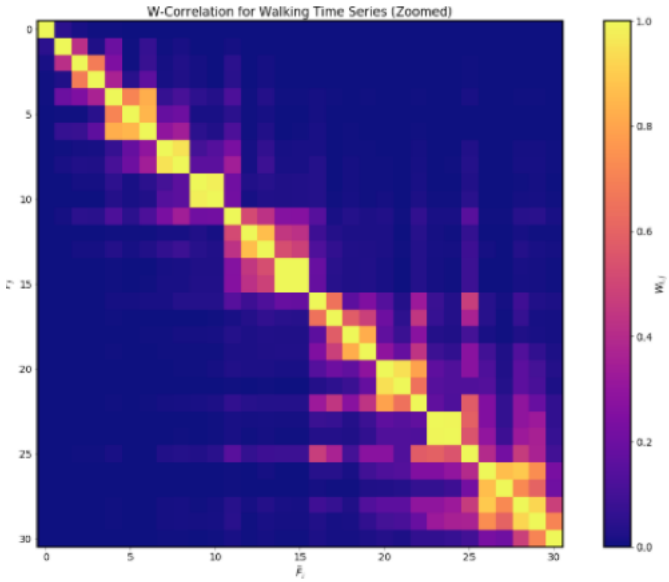


Figure 2: w-correlation.

We can see that the first component has no correlation with the other ones, while we can identify 2 others 'blocks' : 2nd component to the 14th one, and the 15th component all the way up to the 2852 (last) component.

After grouping them, we plot our grouped components (only the $trend+periodicity = signal$ and the noise):

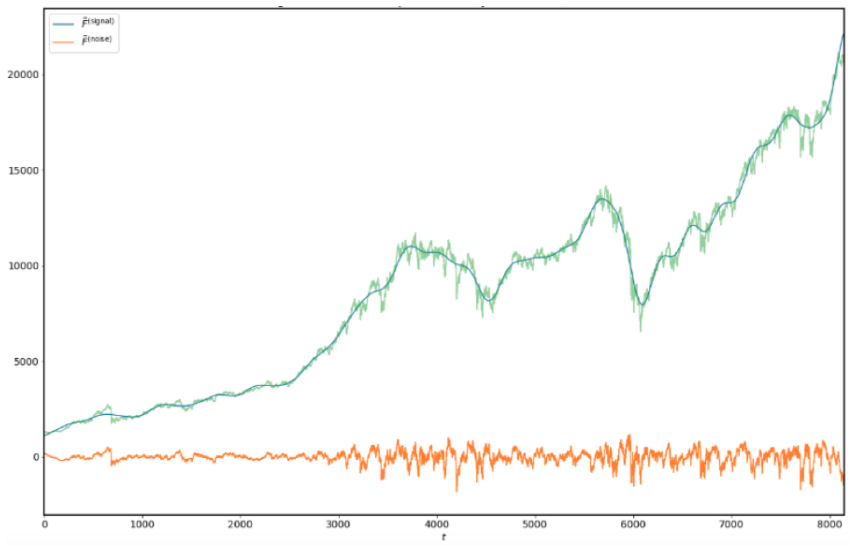


Figure 3: Signal and noise components of the time series.

Using the SSA decomposition, we've been able to decompose our time series: extract the signal and the noise, separately.

Singular spectrum analysis, relatively new non-parametric data-driven technique in time series analysis, is a very powerful tool for detecting patterns in long time series (raw data for example, with no other model assumptions), and can also be used in forecasting.

3 Moving Average Crossover

We used one of the oldest and simplest methods to get an overall idea of the trend in a data set. It is useful when we want to approximate the recent trend of the price and useful for forecasting long-term trends.

3.1 Definition

SMA :

SMA, Simple Moving Average is the mean, or average, of the stock price values over a specified period. The SMA is simply the total sum of the stock price over a specified period of time, divided by the number of days in that stretch.

EMA :

The exponential moving average, EMA is a weighted moving average that gives more weigh or importance to recent price data than the SMA does. The EMA responds more quickly to recent price changes than the SMA. The EMA us defined as :

$$EMA(t) = (1-\alpha)EMA(t \quad 1) + \alpha p(t)$$
$$EMA(t_0) = p(t_0)$$

With $p(t)$ the price at time t and is called the decay parameter for the EMA. α is related to the lag and the length of the window M as $\alpha = \frac{1}{L+1} = \frac{2}{M+1}$

3.2 Forecasting

Methodology :

The idea is simple and powerful. For example, if we use 100-day moving average of our data set (stock prices here), then a significant portion of the daily price noise will have been "averaged-out". Thus, we can observe more accurately the longer-term behavior of the asset. If we use 20-day moving average of our data set, the prediction time series will be more accurate and more 'noisy', and we'll use it to observe short-term stock prices movements. We first compute the rolling Simple moving averages (SMA) of the data set. We need to choose accurately, because calculating the M days with simple MA, the first M1 days are not valid, and we need the first M prices for the first moving average data point.

Application :

We chose SP 500 stock prices and imported our dataset from Yahoo Finance (from 1980-01-31 until today). We first compute the short-rolling (20 days) and long-rolling (100 days) SMA for Close Price of the dataset SP500, then we plot the last 21 months to get a feeling about how these behave.

First we can see that short-rolling and long-rolling time series (SMA) are much less noisy than the original Close Price data. However, this comes at a cost: SMA time series lag the original Close Price time series, which means that the changes in the trend are only seen with a delay (lag) of L days.

How much is this lag L? For a SMA moving average calculated using M days, the lag is roughly M/2 days. Thus, if we are using a 100 days SMA, this means we may be late by almost

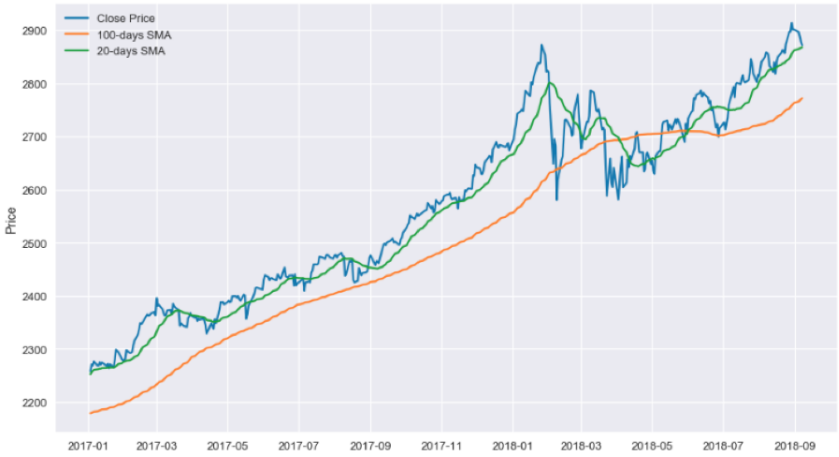


Figure 4: Time series forecasting using Simple Moving Average Crossover.

50 days, which can significantly affect our forecasting, then our trading strategy.

We can reduce this lag using the Exponential Moving Average (EMA) :

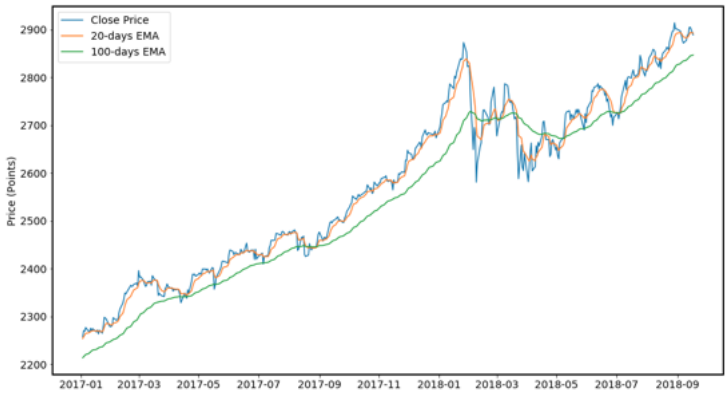


Figure 5: Time series forecasting using Exponential Moving Average Crossover.

The reason why EMA reduces the lag is that it puts more weight on more recent observations, whereas the SMA weights all observations equally by 1/M.

We can see that the predictive EMA time series seems more accurate than SMA one for the small window (20 days here), and EMA keeps the original trend more accurately than SMA for long window (100 days here). Thus the EMA seems to be more adapted for our trading strategy.

3.3 Trading Strategy

As we said Moving Average technique is one of the oldest and simplest trading strategies that exist, used to keep the recent trend of the price.

Methodology :

Now we know how to determine the trend by plotting on some moving averages on our Close Price time series. It also helps us to determine when the trend is about to end and reverse.

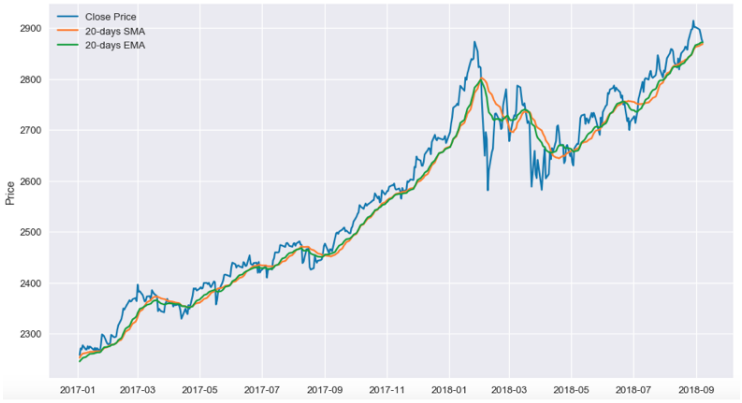


Figure 6: EMA and SMA prediction with the window M=20 days .

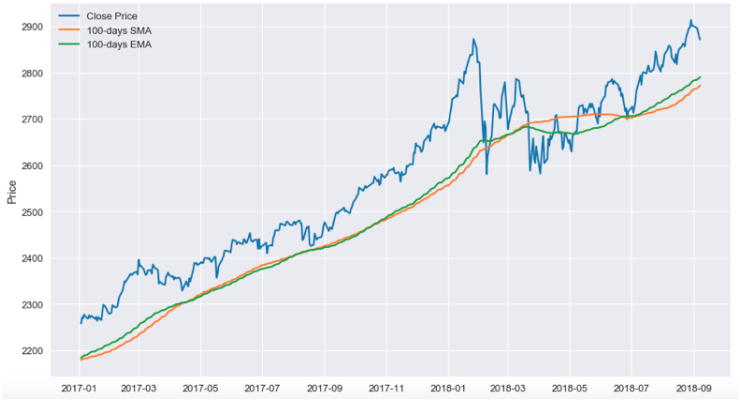
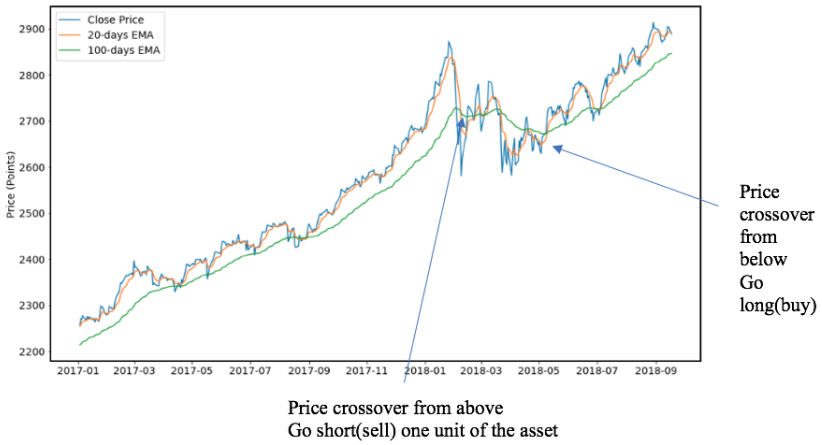


Figure 7: EMA and SMA prediction with the window M=100 days .

Using the moving averages calculated above, we are looking to build a trading strategy. We are also going to take advantage of the fact that a moving average timeseries (whether SMA or EMA) lags the actual price behavior. If the moving averages cross over one another, it could signal that the trend is about to change. Therefore, we will consider the crossing of the two as potential trading signals.



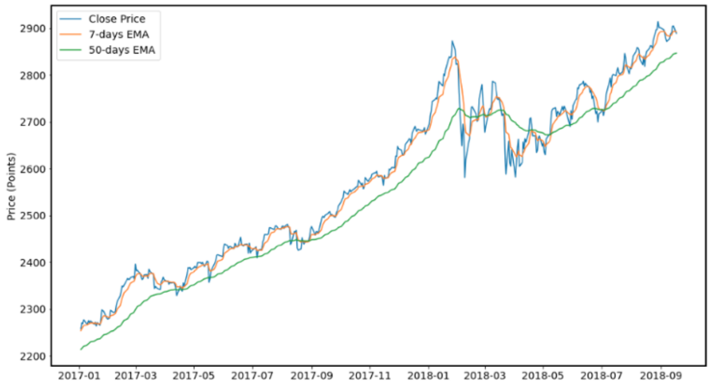
1. When the 20-days EMA crosses under the 100-days EMA, we will close any existing long

position and go short (sell) one unit of the asset. It means that the close price is going down.

2. When the 20-days EMA crosses over the 100-days EMA, we will close any existing short position and go long (buy) one unit of the asset. It means that the close price is going up.

3. Our algorithm is simple : First we compute the 7-days EMA (one week) and 50-days EMA (about two months).

Then we compute the difference between 7-days EMA and 50-days EMA. Given that the time series are continuous data, thus we may not have values equal to 0, we divided the difference by the 50-days EMA, so we will have all the values very small (else we would have values for the difference between 0 and 100 before 1990, and values higher than 400 after 2010, so we can have all the values in the same range). This helps us identifying a crossover.



Then we fix a threshold equal to 0.005, and we create a new 'array' :

- If the value is higher to 0. 005, then we assign 1.
- If the value is small than -0. 005, then we assign -1.
- If the value is between -0. 005,and 0.01, then we assign 0.

Then we get this array :

```
difference_regime
```

```
array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  0,  0, -1, -1, -1, -1,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  0,  0,  0,  0,  0,
        0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  0,  0,  0,  0,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1])
```

We trade like this :

- If value[t] = 0 and value[t-1]=1, then 7-days EMA>50-days EMA at day t-1, which is signal of a crossover from above. Therefor we sell an asset.
- If value[t] = 0 and value[t-1]=-1, then 7-days EMA<50-days EMA at day t-1, which is signal of a crossover from below. Therefor we buy an asset.

With an initial cash equal to 10.000$, we make a profit of 1244 Points with 9 transactions (without counting the transaction cost), which seems to be good.

4 Hidden Markov Models

The hidden Markov model has been widely used in the financial mathematics area to predict economic regimes or predict stock prices.

4.1 Theory

Markov Model

A Markov Model is a stochastic model which is used for modelling systems based on stochastic processes. It provides a way to model the dependencies of current information (e.g. weather) with previous information. A Markov chain is useful when we need to compute a probability for a sequence of events that we can observe in the world. Markov Models are a powerful abstraction for time series data. The system is uncertain, thus we need to take on a probability distribution to describe the set of outcomes.

In many cases, however, the events we are interested in may not be directly observable in the world.

Hidden Markov Model

The Hidden Markov Model, or HMM, is a signal detection model, introduced in 1966 by Baum and Petrie (Baum and Petrie 1966). The applications of HMM were expanded to many areas such as speech recognition, biomathematics, and financial mathematics.

A Hidden Markov Model, is a stochastic model where the states of the model are hidden. Each state can emit an output which is observed.

The observations can be discrete or continuous but the states are always discrete.

A HMM is specified by the following components :

- Sequence of Hidden states $Q = \{q_t | t = 1, 2, ..., T\}$,
- Observation data, $O = \{{O_t}^l | t = 1, 2, .., T; l = 1, 2, ..., L\}$, where l is the number of independent observation sequences and T is the length of each sequence,
- Possible values of each state, $\{S_i, i = 1, 2, ..., N\}$,
- Possible observations per state, $\{v_k, k = 1, 2, ..., M\}$,
- Transition matrix, $A = (a_{ij})_{i,j=1,2,...,N}$, where $a_{ij} = P(q_t = Sj | q_{t1} = S_i)$,
- Initial probability distribution $\pi = (P(q_1 = S_1), \ldots, P(q_1 = S_M))$
- Observation probability matrix, also called emission probabilities, each expressing the probability of an observation v_k being generated from a state S_i, $B = b_i(O)$, where $b_i(O_t) = b_i(O_t = v_k) = P(O_t = v_k | q_t = S_i), i = 1, 2, ..., N, k = 1, 2, ..., M$.

We use a compact notation for the parameters, given by $\lambda \equiv \{A, B, p\}$. The HMM system can be graphically represented using a Trellis diagram :

After modelling our system as a HMM, we need to initialize the parameters, where they can first be estimated. A good HMM model is established after training the model on historical data and finding the right parameters adjusted to the specific system and its patterns.

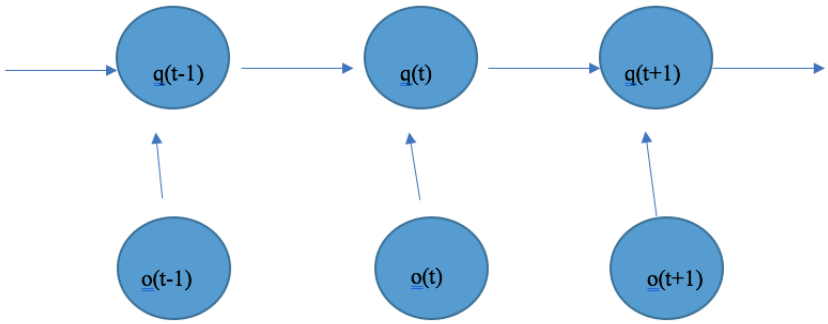


Figure 8: Trellis diagram representing Q as the hidden sequence of states and O the sequence of observations.

4.2 Three Fundamental Problems

An influential tutorial by Rabiner (1989), based on tutorials by Jack Ferguson in the 1960s, introduced the idea that hidden Markov models should be characterized by three fundamental problems. There are three basic problems of interest that must be solved for the model to be useful in real world applications :

1. Given an observation data O and the model parameters λ, what is the probability $P(O|\lambda)$ of this observed sequence ?
2. Given the observation data O and the model parameters λ, what is the most likely series of states to generate the observations ?
3. Given the observation O, can we find the model's parameters λ?

The first problem can be solved by using forward algorithms Baum and Egon (1967); the second problem was solved by using Viterbi algorithm Forney (1973); the Baum–Welch algorithm Rabiner (1989) was developed to solve the last problem.

In Stock Prices Prediction, most of the time we use the Baum–Welch algorithm to calibrate parameters for the model and the forward algorithm to calculate the probability of observation to predict trending signals for stocks.

Forward Algorithm :

We define the joint probability function as $\alpha_t(i) = P(O_1, O_2, ..., O_t, q_t = S_i|\lambda)$, then calculate $\alpha_t(i)$ recursively. The probability of observation $P(O|\lambda)$ is just the sum of the $\alpha_T(i)$s.

The forward algorithm.

1: Initialization: for i=1,2,..., N

$$\alpha_{t=1}(i) = p_i b_i(O_1).$$

2: Recursion: for $t = 2, 3, \ldots, T$, and for $j = 1, 2, \ldots, N$, compute

$$\alpha_t(j) = \left[\sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] b_j(O_t).$$

3: Output:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Figure 9: Forward Algorithm.

The Viterbi Algorithm :

The goal here is to find the best sequence of states Q when (O, λ) is given. We define :

$$\delta_t(i) = max_{q_1,q_2,...,q_{t1}} P(q_1, q_2, ..., q_t = S_i, O_1, ..., O_t|\lambda)$$

By induction, we have:

$$\delta_{t+1}(j) = max_i[\delta_(i)a_{ij}]b_j(O_{t+1})$$

Using $\delta_t(i)$, we can solve for the most likely state q_t , at time t, as :

$$q_t = argmax_{1\leq i\leq N}\delta_t(i), 1 \leq t \leq T$$

The Viterbi algorithm.

1: Initialization:

$$\delta_1(j) = p_j b_j(O_1), \quad j = 1, 2, ..., N;$$

$$\phi_1(j) = 0.$$

2: Recursion: for $2 \leq t \leq T$, and $1 \leq j \leq N$

$$\delta_t(j) = \max_i[\delta_{t-1}(i)a_{ij}]b_j(O_{t+1})$$

$$\phi_t(j) = arg\max_i[\delta_{i1}(i)a_{ij}]$$

3: Output:

$$q_T^* = \text{argmax}_i[\delta_T(i)]$$

$$q_t^* = \phi_{t+1}(q_{t+1}^*), \; t = T-1, \ldots, 1$$

Figure 10: Viterbi Algorithm.

The Baum-Welch algorithm

Given the third problem, we have to find the parameters $\lambda = A, B, p$ to maximize the probability $P(O, \lambda)$ of observation data $O = \{O_1, O_2, ..., O_T\}$.

We can choose the parameters, such that $P(O|\lambda)$ is locally maximized using the Baum–Welch iterative method, which used the maximum likelihood estimator (MLE) to train the model parameters. In order to describe the procedure, we defined $\gamma_t(i)$, the probability of being in state S_i at time t, as:

$$\gamma_t(i) = P(q_t = S_i|O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(O, \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N} \alpha_t(j)\beta_t(j)}$$

The probability of being in state S_i at time t and state S_j at time $t + 1$, $\xi_t(i, j)$ is defined as:

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) = \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O, \lambda)}$$

And we have :

The Baum–Welch algorithm

1: Initialization: input parameters λ, the tolerance tol, and a real number $\triangle$

2: Repeat until $\triangle < tol$

- Calculate $P(O, \lambda)$ using forward algorithm in Section 2.1
- Calculate new parameters λ^*: for $1 \leq i \leq N$

$$p_i^* = \gamma_1(i)$$

$$a_{ij}^* = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)},\ 1 \leq j \leq N$$

$$b_{ik}^* = \frac{\sum_{t=1, O_t=v_k}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)},\ 1 \leq k \leq M$$

- Calculate $\triangle = |P(O, \lambda^*) - P(O, \lambda)|$
- Update $\lambda = \lambda^*$

3: Output: parameters λ.

Figure 11: Baum-Welch Algorithm.

These steps are repeated iteratively until a desired level of convergence (tol : threshold fixed or certain number of reiterations instead).

4.3 Application of the HMM on stock market

Besides the match between the properties of the HMM and the time series data, the Expectation Maximization(EM) algorithm provides an efficient class of training methods.

Given plenty of data that are generated by some hidden power, we can create an HMM architecture and the EM algorithm allows us to find out the best model parameters that account for the observed training data. Especially in the case of financial time series analysis and prediction, where the data can be thought of being generated by some underlying stochastic process that is not known to the public.

If the professional's strategies and the number of his strategies are unknown, we can take the imaginary professional as the underlying power that drives the financial market up and down. As shown in the example we can see the HMM has more representative power than the Markov chain.

The transition probabilities as well as the observation generation probability density function are both adjustable.

The hidden Markov model has been widely used in the financial mathematics area to predict economic regimes or predict stock prices.

In Stock Market, observations are continuous vector- Open Price, Closing Price, High and Low. The states are discrete : stock market movements like rise, drop, no change; or bull and bear.

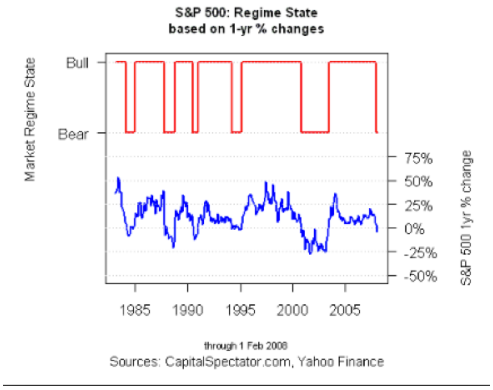


Figure 12: S&P 500 regime detection.

5 Deep Learning

In recent years, deep artificial neural networks (including recurrent ones) have won numerous contests in pattern recognition and machine learning.

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Deep learning, a powerful set of techniques for learning in neural networks.

Deep learning is the application of artificial neural networks(neural networks for short) to learning tasks using networks of multiple layers. It can exploit much more learning (representation) power of neural networks, which once were deemed to be practical only with one or two layers and a small amount of data.

Neural networks and deep learning are big topics in Computer Science and in the technology industry, they currently provide the best solutions to many problems in image recognition, speech recognition and natural language processing.

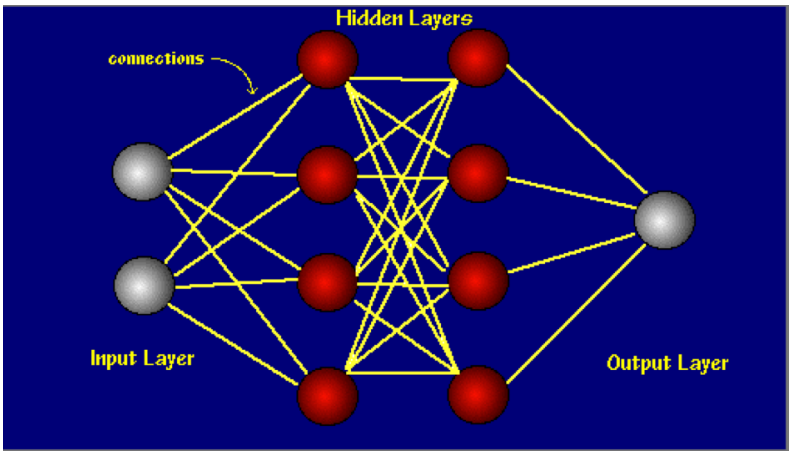
5.1 Neural Networks

Neural networks, developed in the 1950s not long after the dawn of AI research, looked promising because they attempted to simulate the way the brain worked, though in greatly simplified form. A program maps out a set of virtual neurons and then assigns random numerical values, or "weights," to connections between them. These weights determine how each simulated neuron responds—with a mathematical output between 0 and 1.

Neural neworks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'.

Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. The hidden layers then link to an 'output layer' where the answer is output as shown in the graphic below.

The hidden layer is composed of many cells with a weight associated to each. Firstly, the weights of the network are initialized randomly and then they are adjusted during the training



phase.

At the end of the training phase, we obtain a neural network which can predict with a good approximation the output value according to the close values given as inputs.

Why using Neural Networks in Prediction :

The advantage of the usage of neural networks for prediction is that they are able to learn from examples only and that after their learning is finished, they are able to catch hidden and strongly non-linear dependencies, even when there is a significant noise in the training set.

The disadvantage is that Neural Nets can learn the dependency valid in a certain period only. The error of prediction cannot be generally estimated.

We have time series, i.e. a variable x changing in time $x_t(t = 1, 2, ...)$ and we would like to predict the value of x in time $t + h$.

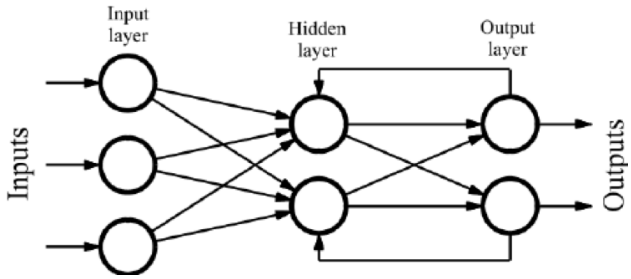
The prediction of time series using neural network consists of teaching the net the history of the variable in a selected limited time and applying the taught information to the future. Data from past are provided to the inputs of neural network and we expect data from future from the outputs of the network.

There are many classes of neural networks.

5.2 Recurrent Neural Networks

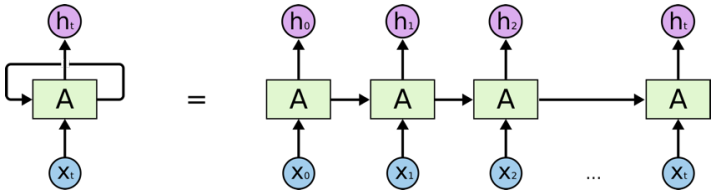
Recurrent neural networks were developed in the 1980s. Recurrent Neural Networks is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for Machine Learning problems that involve sequential data. (time series in particular).

A recurrent neural network (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence.



There have been incredible success applying RNNs to a variety of problems: speech recognition, language modeling, image captioning...

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. They are networks with loops in them, allowing information to persist.



In the above diagram, A looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next. A Recurrent Neural Network produces output, copies that output and loops it back into the network. This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists.

The "internal memory" means that RNN performs the same task for every element of a sequence with each output being dependent on all previous computations, which is like "remembering" information about what has been processed so far. Therefore a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

5.2.1 Theory

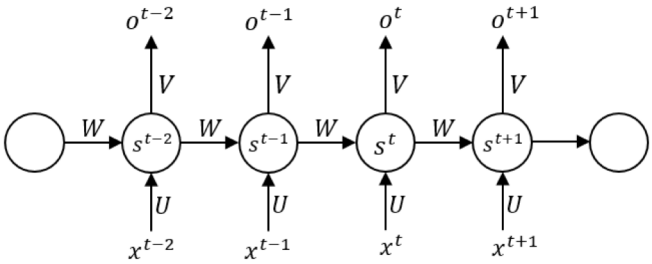


Figure 13: RNN model.

The formulas that govern the computation happening in a RNN are as follows:

• x_t is the input at time step t. For example, x_1 could be a one-hot vector corresponding to the Close Price value at time 1.

• U, V are weight vectors, different for every time step : U for the hidden layer, V for the output layer. Weight vectors are random initially.

• The weight vector W is the same for every time step.

• s_t is the hidden state at time step t. It's the "memory" of the network. s_t is calculated based on the previous hidden state and the input at the current step :

$$s_t = f(Ux_t + Ws_{t-1})$$

The function f usually is a nonlinearity such as 'tanh' or 'ReLU' (the rectifier, an activation function such as : $f(x) = x^+ = max(0, x)$).

s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.

• o_t is the output at step t, $o_t = Vs_t$. The basic idea is to keep in memory the sequence of inputs $[x_0, ..., x_t]$ and its order because we are using the last following close values to predict the new one. As a rule, algorithms exposed to more data produce more accurate results.

5.2.2 RNN Problems

There are two major obstacles RNN's have or had to deal with. But to understand them, you first need to know what a gradient is. A gradient is a partial derivative with respect to its inputs. If you don't know what that means, just think of it like this: A gradient measures how much the output of a function changes, if you change the inputs a little bit.

You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops to learning. A gradient simply measures the change in all weights with regard to the change in error.

We speak of 'Vanishing Gradients' when the values of a gradient are too small and the model stops learning or takes way too long because of that. This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber, which we will discuss now.

Theoretically, RNN can make use of the information in arbitrarily long sequences, but in practice, the standard RNN is limited to looking back only a few steps due to the vanishing gradient or exploding gradient problem.

Thankfully, LSTMs don't have this problem!

5.3 Long Short-Term Memory

The Long Short-Term Memory, or LSTM, network is a type of Recurrent Neural Network (RNN) designed for sequence problems and capable of learning long-term dependencies.

LSTMs are explicitly designed to avoid the long-term dependency problem. They were introduced by Hochreiter Schmidhuber (1997), and were refined and popularized by many

people in following work. They work tremendously well on a large variety of problems, and are now widely used.

It basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.

5.3.1 Theory

LSTMs were designed to combat vanishing gradients through a gating mechanism.

We first define the sigmoid function S such as, here denoted the function σ :

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

An LSTM module (or cell) has 5 essential components which allows it to model both long-term and short-term data :

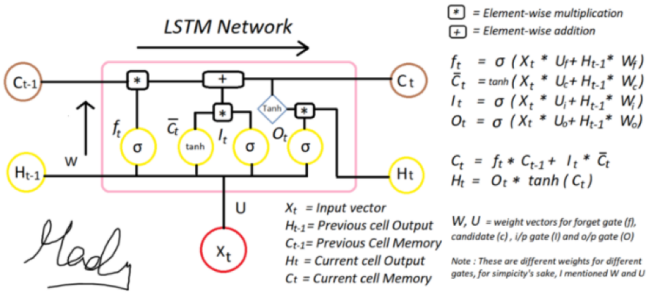


Figure 14: Logistic sigmoid $\frac{1}{1+e^{-x}}$, or σ is used for activation functions of the gates and hyperbolic tangent tanh is used as the block input and out activation functions.

-Cell state (C_t) : This represents the internal memory of the cell which stores both short term memory and long-term memories

-Hidden state (H_t) : This is output state information calculated w.r.t. current input, previous hidden state and current cell input which you eventually use to predict the future stock market prices. Additionally, the hidden state can decide to only retrive the short or long-term or both types of memory stored in the cell state to make the next prediction.

-Input gate (I_t) : Decides how much information from current input flows to the cell state.

-Forget gate (f_t) : Decides how much information from the current input and the previous cell state flows into the current cell state.

-Output gate (O_t) : Decides how much information from the current cell state flows into the hidden state, so that if needed LSTM can only pick the long-term memories or short-term memories and longterm memories.

Long Short-Term Memory models are extremely powerful time-series models. They can predict an arbitrary number of steps into the future.

5.3.2 Application

In this application, we use the stock value of the Dow Jones Index, one of the oldest U.S. market index and best-known measuring stick for the American stock market, imported from Yahoo

Finance historical data.

We can phrase the problem as a regression problem :

Given the Close Price of the shares (the input) at time T, $(T-1), ..., (T-h+1)$, what is the Close Price of the shares at time $T+1$?

Where h (denoted as the variable lookback) is the number of previous time steps to use as input variables to predict the next time period – defaulted to lookback = 25 here.

We normalize the Closing Price, split the ordered dataset into train and test datasets (we fix the train set as the first $k = 70\%$ data of the original dataset) and then run a LSTM network on it in Python using keras with Tensorflow.

Each day, we are going to predict and plot the following day based on the last 'lookback = 25' days :

We can see that the predictions tends to fit the real close price value, the Mean Squared Errors tends to be really low, which means that the prediction is really close to the actual value for a day-by-day prediction.

As preconized by Christopher Krauss (2016), "careful hyper parameter optimization may still yield advantageous results for the tuning-intensive deep neural networks". This is the reason why we are looking for the best parameters of the neural network to obtain the best forecasting model.

Tuning the Hyperparameters :

We built out LSTM model in Python with Keras (Tensorflow environment) :

-The previous network has a visible layer with 1 input, a hidden layer with 20 LSTM blocks or neurons, and an output layer that makes a single value prediction.

-The default sigmoid activation function is used for the LSTM blocks. We can change the activation function, choosing the 'tanh' or 'linear' function.

-A loss function (or optimization score function) is one of the two parameters (with the optimizer) required to compile a model.

-We have to choose a good optimizer, or optimization algorithm for the deep learning model, which can mean the difference between good results in minutes, hours and days.

-We also need to specify the number of epochs (20 for the previous one) and the size of batches.

Epochs, size of Batches and Dropout :

1) The first LSTM parameter we look at tuning is the number of training epochs. One epoch is when an entire dataset is passed forward and backward through the neural network only once. We are using a limited dataset and to optimize the learning, we use Gradient Descent which is an iterative process. Thus, updating the weights with single pass (one epoch) is not enough, which leads to underfitting.

We first calculate the RMSE of train and test data for each epoch with different number of maximum epochs (for example if epoch = 200, we run RMSE 200 times). This prevents us to overfit and gives an aproximated range of epochs to start with. It's important that the RMSE test curve must not be convex as it denotes overfitting. We tried 50, 100, 150, 200, 300 and 500 epochs, and evaluated the Mean Squared Error Value : we kept 200 epochs.

2) Moreover, we can't pass the entire dataset into the neural network at one. So we need to divide the data into number of batches or parts : the batch size is the total number of training examples present in a single batch. Each batch trains network in a successive order.

For example, if we have 100 training examples, and our batch size is 50, then it will take only 2 iterations to complete 1 epoch. If we have 1024 training examples, and our batch size is 64 it means that your model will receive blocks of 64 samples, compute each output, average the gradients and propagate it to update the parameters vector.

3) A dropout is an argument for our LSTM model. It is a float between 0 and 1, a parameter which indicates the number of cells that the neural net is going to forget every pass, in order to avoid overfitting. If the number is 1 then all the cells are forgotten each pass and if it's 0 we keep all the cells.

We try different Dropout values between 0 and 1, and evaluate the Mean Squared Error of each dropout value (lowest difference between the prediction and the real value). We found that 0.3 is the optimal Dropout value for our LSTM model.

Number of hidden layers and neurons :

Configuring neural networks is difficult because there is no good theory on how to do it. The optimum parametrization depends on the problem.

After choosing the best number of epochs, we repeat the method but maintaining the epochs constant and testing with different neuron number. With this method we can tune this parameters obtaining a good trade off between accuracy and generalization.

Optimizer :

An optimizer is one of the two arguments (with loss function) required for compiling a keras model.

- The optimizer Root Mean Square Propagation, 'RMSprop', an adaptive learning rate method, is usually a good choice for recurrent neural networks. It maintains per-parameter learning rates that are adapted, based on the average of recent magnitudes of the gradients for the weight.

- The Adam optimization algorithm is an extension to stochastic gradient descent, which is different to classical ones. Adaptive Moment Estimation (Adam) combines the advantages of two other extensions of stochastic gradient descent (Adaptive Gradient Algorithm and RMSprop), realizing the benefits of both.

- Nadam (Nesterov-accelerated Adaptive Moment Estimation) is an extension of Adam, incorporation the algorithm NAG (Nesterov accelerated gradient) into Adam, which allows to perform a more accurate step in the gradient direction.

Diagnoze overfitting and underfitting of LSTM models :

You may be getting a good model skill score, but it is important to know whether your model is a good fit for your data or if it is underfit or overfit and could do better with a different configuration.

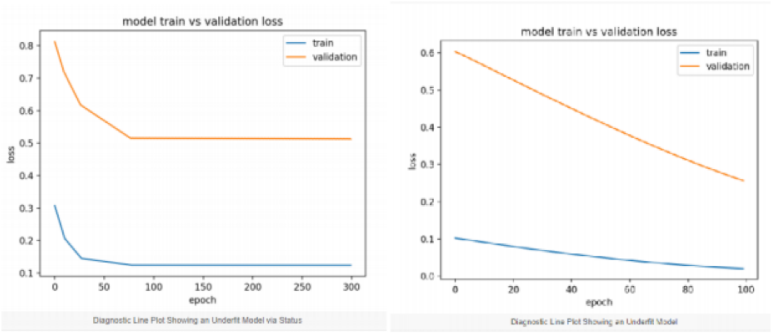
The training history of your LSTM models can be used to diagnose the behavior of your model. You can plot training loss vs test loss.

LSTMs are stochastic, meaning that you will get a different diagnostic plot each run.

It can be useful to repeat the diagnostic run multiple times (e.g. 5, 10, or 30). The train and validation traces from each run can then be plotted to give a more robust idea of the behavior of the model over time. The example below runs the same experiment a number of times before plotting the trace of train and validation loss for each run.

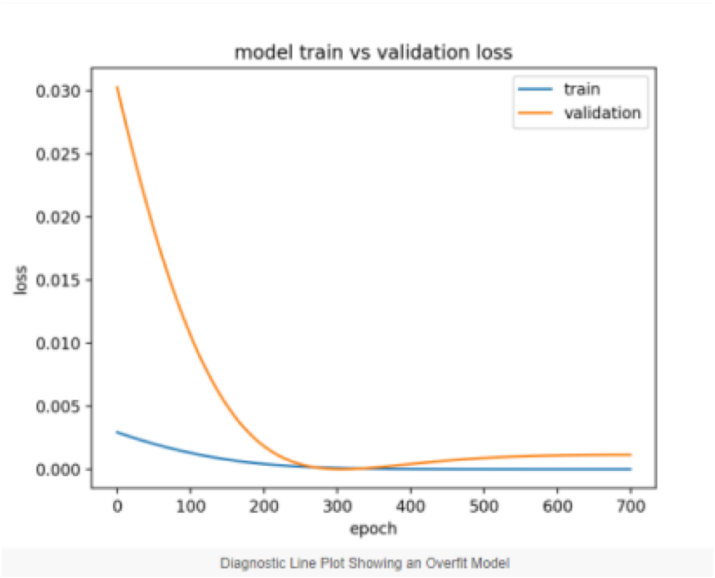
a) Under Fit :

The training history of your LSTM models can be used to diagnose the behavior of your model. You can plot the performance of your model using the Matplotlib library. For example, you can plot training loss vs test loss as follows.



b) Over Fit :

An overfit model is one where performance on the train set is good and continues to improve, whereas performance on the validation set improves to a point and then begins to degrade. This can be diagnosed from a plot where the train loss slopes down and the validation loss slopes down, hits an inflection point, and starts to slope up again.

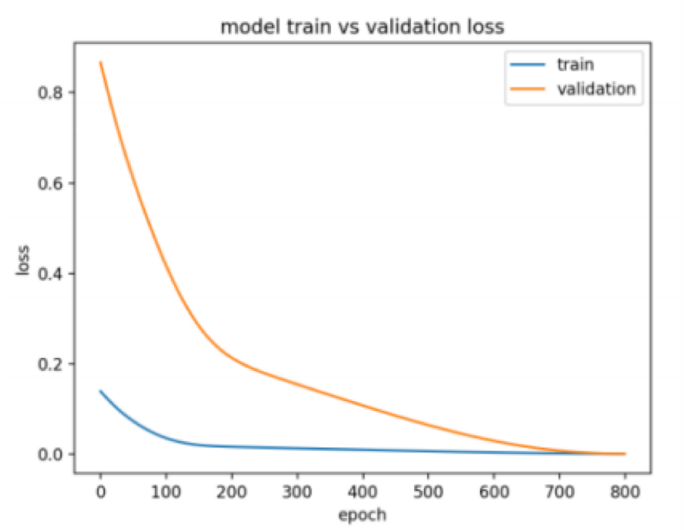


Running this example creates a plot showing the characteristic inflection point in validation loss of an overfit model.

This may be a sign of too many training epochs. In this case, the model training could be stopped at the inflection point. Alternately, the number of training examples could be increased.

c) Good Fit :

A good fit is a case where the performance of the model is good on both the train and validation sets. This can be diagnosed from a plot where the train and validation loss decrease and stabilize around the same point.



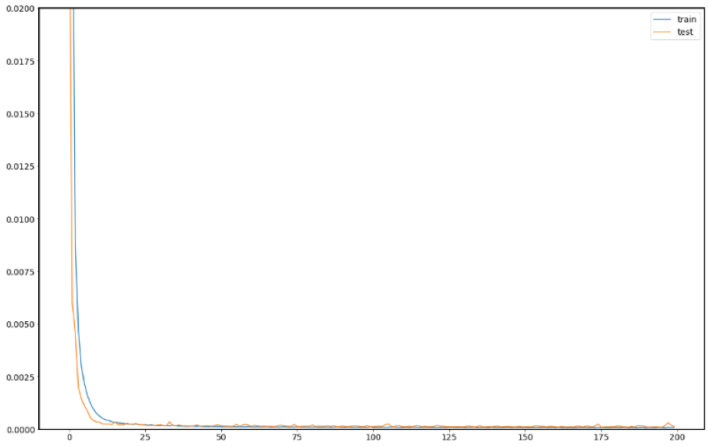
Diagnostic Line Plot Showing a Good Fit for a Model

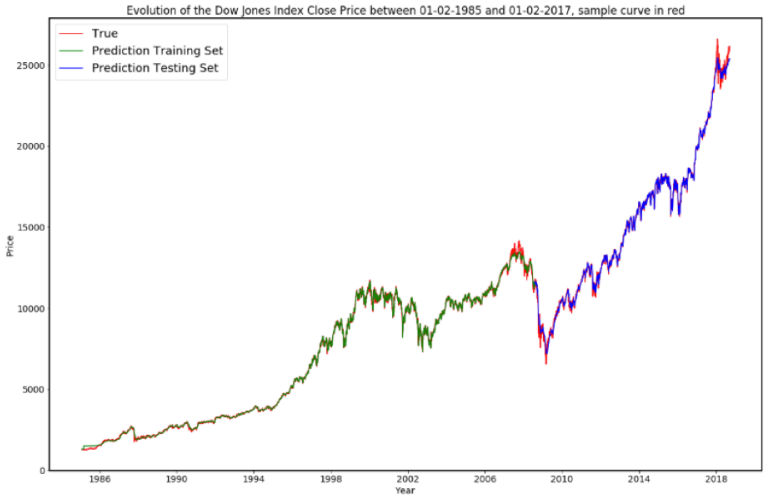
Visualisation

Once the model is fit, and the other parameters optimized, we can estimate the performance of the model on the train and test datasets.

This will give us a point of comparison for new models.

Note that we invert the predictions before calculating error scores to ensure that performance is reported in the same units as the original data.






```
Train Score: 117.77 RMSE
Test Score: 216.38 RMSE
```

It is getting better and more accurate, so we keep this model for other forecastings.

5.3.3 Natural Language Processing : Sentiment Analysis

We want to build a predictive model that will be able to buy and sell stock based on profitable prediction, without any human interactions.
We can use Natural Language Processing (NLP) to make smart "decisions" based on current affairs, articles, tweets... Our goal is to increases the accuracy of the stock predictions.

Natural Language Processing is a technique used by a computer to understand and manipulate natural languages. By natural languages, we mean all human derived languages. NLP is used to analyze text and let machines derive meaning from the input.
Methodology :

In our project, we make use of some established NLP techniques to evaluate past data of the stock, in order to make predictions in stock trends. With the raw data, we cannot proceed much further untilwe manipulate the data to suit our analysis and convert the data into vectors that are much easier to work on.

Sentimental Analysis is an analytical method that the computer uses to understand a natural language and deduce if the message is positive, neutral or negative. In our case, Sentimental Analysis refers to the deduction of the news headlines (here the last tweets) : if the stock prices are going up or down.

Our goal is to predict the tendency of the stock of a specific company or financial index (here Dow Jones Index), the data that lead the stock's price of the next day to decline or stay the same are labelled or rise are labelled.

We imported our articles from The New York Times, because the archives of NY times gives us access to articles from September 18, 1851 to today, retrieving headlines, and helpful metadata (subject terms, abstract, date) which can gives us data from articles about the stock market.

We first explore the first 100 pages of the media, looking for the articles containing the word 'Dow Jones', since 2010.

	Unnamed: 0	date	headline	snippet	source	subjects	url
1992	3743	2018-07-26	b'Facebook\xe2\x80\x99s Stock Plunge Shatters ...	b'Roughly $120 billion in wealth was wiped out...	The New York Times	['Social Media', 'Advertising and Marketing', ...	https://www.nytimes.com/2018/07/26/business/fa...
1993	3744	2018-08-01	b'Apple\xe2\x80\x99s Stock Buybacks Continue t...	b'The iPhone maker has repurchased almost $220...	The New York Times	['Stocks and Bonds']	https://www.nytimes.com/2018/08/01/business/de...
1994	3745	2018-08-02	b'Stocks Fall as Trade Tensions Cause Jitters'	b'Benchmark stock indexes in Asia fell by more...	The New York Times	['Stocks and Bonds', 'International Trade and ...	https://www.nytimes.com/2018/08/02/business/st...
1995	3746	2018-08-06	b'The Stock Market\xe2\x80\x99s Next $1 Trilli...	b'Corporate boards could authorize companies t...	The New York Times	['Corporate Taxes', 'Stocks and Bonds', 'Board...	https://www.nytimes.com/2018/08/06/business/de...
1996	3748	2018-08-13	b'Plunge in Lira, Turkey\xe2\x80\x99s Currency...	b'The country\xe2\x80\x99s economic crisis has...	The New York Times	['Economic Conditions and Trends', 'Internatio...	https://www.nytimes.com/2018/08/13/business/tu...

After getting our data, we use the package NLTK in Python to analyze 'snippets' and gives us data about the headlines.

Thus we get this dataframe :

Date	close	compound	neg	neu	pos
2018-07-26	25527.070312	0.4215	0.000	0.741	0.259
2018-07-27	25451.060547	0.0000	0.000	0.000	0.000
2018-07-30	25306.830078	0.0000	0.000	0.000	0.000
2018-07-31	25415.189453	0.0000	0.000	0.000	0.000
2018-08-01	25333.820312	0.0000	0.000	1.000	0.000
2018-08-02	25326.160156	-0.4019	0.310	0.690	0.000
2018-08-03	25462.580078	0.0000	0.000	0.000	0.000
2018-08-06	25502.179688	0.0000	0.000	1.000	0.000

With the close price and the sentiment analysis for the data from NY times (compound, negative, neutral and positive are the polarity, compound is just the aggregated score of neg, neu and pos).

The rows containing 0 are the days where we're missing data from NY Times. Then we apply the LSTM network on our data, with 5 inputs.

On the first try, the predictions seem to be quite messy at the end. But it's normal, because we don't have all the correct inputs : not all the articles we are looking for, not every day articles etc... However we are sure that our predictions could be more accurate if we change the source of data for example (change the source NY times by 'MarketWatch' for example or maybe twitter). Thus we can say that this technique seems to be really interesting for our forecasting, and should be more improved so we can get better results.

We can see finally that tuning the hyperparameters of our LSTM model and adding inputs can give us a better model and more accurate predictions. We kept our tuned LSTM model (without NLP Sentiment analysis).

6 Reinforcement Learning

« Reinforcement Learning - An Introduction », by Richard S. Sutton and Andrew G. Barto published in 1998, is one of the major references about reinforcement learning.

6.1 Theory

Reinforcement Learning is a branch of Machine Learning which can help us find an optimal strategy for a sequential decision problem, with direct interaction with the environment.

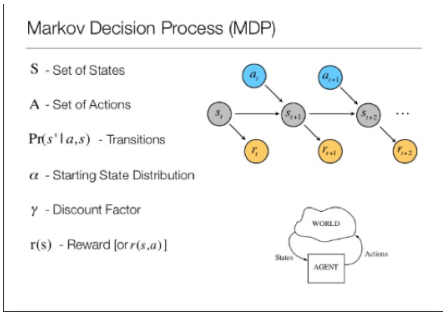
It has recently been in the spotlight during the last decade for being at the core of the system who beat the Go world champion in 5-match series.

Reinforcement Learning can be modeled as a Markov Decision Process (MDP). An MDP is a directed graph which has states for its nodes and edges, which describe transitions between Markovian states.

The agent seeks, through iterated experiments, an optimal decision-making behavior (called strategy or policy, which is a function associating the current state with the action to be executed) in the sense that it maximizes the sum of Rewards over time.

A **Markov Decision Process** consists of a tuple of 5 elements:

- S: Set of states of the agent in the environment.
- A : Set of actions that the agent can perform.
- $P(s(t+1)|s(t), a(t))$: State transition model that describes how the environment state changes when the user performs an action a, depending on the action a and the current state $s(t)$.
- $P(r(t+1)|s(t), a(t))$: Reward model that describes the real-valued reward value that the agent receives from the environment after performing an action, depending on the current state and the action performed.
- γ : discount factor that controls the importance of future rewards.



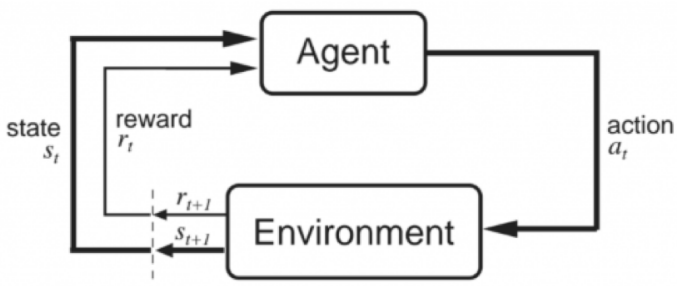
The most important components are the agent and the environment.

« Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. » (Sutton Barto (1998), Reinforcement Learning - An Introduction).

The goal in reinforcement learning is to learn how to act to spend more time in more valuable states. To have a valuable state we need more information in our MDP.

The agent can choose which action to take in a given state, which has a significant effect on the states it sees. This MDP has the addition of rewards. Each time you make a transition into a state s, you receive a reward r.

However, the agent does not control the dynamics of the environment completely, which adds some randomness. The environment, upon receiving these actions, returns the new state and the reward.



At some time step t, the agent is in state $s(t)$ and takes an action $a(t)$. The environment then responds with a new state $S(t+1)$ and a reward $r(t+1)$. The reason that the reward is at $(t+1)$ is because it is returned with the environment with the state at $(t+1)$, so it makes sense to keep them together.

Reinforcement Learning is a general class of algorithms in the field of Machine Learning that allows an agent to learn how to behave in a stochastic and possibly unknown environment, where the only feedback consists of a numerical score (reward).

Reward :

The goal of the agent is to learn by trial-and-error which actions maximize his long-run rewards and maximize its total future reward (adding the maximum reward attainable for each state). This potential reward (or future cumulative weighted reward) is a weighted sum of the expected values of the rewards of all future steps starting from the current state t. In mathematical notation, we have :

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

with $0 < \gamma < 1$ the discount factor.

Adding gives us two benefits :

-The return (the sum of expected values of the rewards of all future steps) is well defined for infinite series. -It gives a greater weight to sooner rewards, meaning that we care more about imminent rewards and less about rewards we will receive further in the future.

- If $\gamma = 1$, we have :

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... = \sum_{k=0}^{\infty} r_{t+k+1}$$

we might end up with an infinite return.

- If $\gamma = 0$, we care only about the immediate reward and not about the future rewards.

Policy :

Based on these interactions, the reinforcement learning algorithm should allow the agent to develop a policy $\pi(s, a)$ which describes a way of acting. It is a function that takes in a state and an action and returns the probability of taking that action in that state. Therefore for each state, we have $\sum_a \pi(s, a) = 1$, and the policy is supposed to describe how to act in each state s.

In reinforcement learning, our goal is to learn an optimal policy π^*, which tells us how to act to maximize return in every state.

However, since the environment evolves stochastically and may be influenced by the actions chosen, the agent must balance his desire to obtain a large immediate reward by acting greedily and the opportunities that will be available in the future : compromise between the quest for short-term rewards and long-term rewards.

Value Functions :

We use two types of value functions in reinforcement learning: the state value function, denoted $V(s)$, and the action value function, denoted $Q(s, a)$.

The value function represent how good is a state for an agent to be in. It is equal to expected total reward for an agent starting from state s. The value function describes the value of a state when following a policy :

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t | s_t = s]$$

The other value function is the action value function, the expected return given the state and action under the policy π, and tells us the value of taking an action in some state s under the policy π :

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a]$$

The agent's goal is to select a policy π that maximizes his expected return in all possible states, and the corresponding value functions are called :

- Optimal State-Value Function :

$$V_*(s) = \sup_{\pi} V_{\pi}(s)$$

- Optimal Action-Value Function :

$$Q_*(s, a) = \sup_{\pi} Q_{\pi}(s, a)$$

The optimal policy corresponds to Optimal Action-Value function such that :

$$\pi_* = argmax_\pi V^\pi(s) \quad \forall s \in \mathbb{S}$$

And the optimal policy can be obtained by selecting in each state the action a^* which maximizes Q^* such that :

$$\pi_*(s) = argmax_a Q^*(s,a) \quad \forall s \in \mathbb{S}$$

with $a^* = \pi^{(s)}$.

The next algorithm could be useful to compute the Optimal State-Value :

Initialize a policy π' arbitrarily
Repeat
  $\pi \leftarrow \pi'$
  Compute the values using π by
    solving the linear equations
      $V^\pi(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s)) V^\pi(s')$
  Improve the policy at each state
    $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V^\pi(s'))$
Until $\pi = \pi'$

Figure 15: Pseudo-code for value-iteration algorithm.

This one could be useful to extract the optimal policy π :

Initialize $V(s)$ to arbitrary values
Repeat
  For all $s \in S$
    For all $a \in \mathcal{A}$
      $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a) V(s')$
    $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

Figure 16: Pseudo-code for policy-iteration algorithm.

Bellman Equation :

The Bellman's equation for a policy π :

$$Q(s,a) = R(s,a) + \gamma * max_a(Q(s',a'))$$

with $R(s,a)$ the expected reward obtained when action a is taken in state s such that :

$$\mathfrak{R}(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

Since $V(s) = max_a(Q(s',a'))$, We have the relation :

$$Q(s,a) = R(s,a) + *V(s')$$

Which gives us :

$$Q^*(s,a) = R(s,a) + \gamma * V^*(s')$$

Which makes it easier to compute the optimal policy π.

Q-Learning :
Q-learning is a reinforcement learning technique used in machine learning.

The Q-Learning is used when the agent does not know apriori what are the effects of its actions on the environment (state transition and reward models unknown). The goal of Q-Learning is to learn a policy , which tells an agent what action to take under what circumstances. The agent only knows what are the set of possible states and actions, and can observe the environment current state.

It does not require a model of the environment : the agent has to actively learn through the experience of interactions with the environment, and can handle problems with stochastic transitions and rewards, without requiring adaptations.

Q-learning finds a policy that is optimal in the sense that it maximizes the expected value of the total potential reward (starting from the current state).
The idea of the Q-Learning is to approximate the state-action paires Q-function, $Q : S$ x A -> $\mathbb{R}$ from the samples $Q(s, a)$ that we observe during interaction with the environment.

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, Q_{obs}(s,a)$$
$$\text{where, } Q_{obs}(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

Q-Learning algorithm

where α is the learning rate.
We first initialize the table Q(s,a) arbitrarily, then the agent starts interacting with the environment, receiving a reward of its action r(s,a) at each interaction. Then the agent computes the observed Q-Value Qobs, which is used to update Q(s,a).
With this equation is it possible to find the best actions to perform at each time in order to obtain the best rewards in the future, therefore obtain the best trading strategy in our case.

6.2 Deep Reinforcement Learning

Let us suppose that we created an agent in reinforcement learning to build a robust trading strategy using the algorithm of Q-Learning. We first implement the Q-learning function to create and update a Q-table.

However if we want to deal with huge amount of data and create a robot who could perform Trading High Frequency, it would be difficult creating and updating a Q-table for that environment : the (state, action) space can be very big.

The best idea in this case is to create a neural network to approximate that will approximate, given a state, the different Q-values for each action; this is the essence of the Deep Q-Learning or Deep Reinforcement Learning. Therefore we use the Neural Network to predict the Q-value, based on the input (state,action), which is more tractable than storing every possible value like

we can do in a simple Reinforcement Learning.

6.2.1 Methodology

Now we are looking to apply Deep Reinforcement Learning to a stock value dataset SP 500 imported from Yahoo Finance, and used raw data from 2010-01-01 to today. (training set from 2010-01-01 to 2017-12-31 and the test set from 2018-01-01 to today).

Then we initialize the first state S0 composed of many financial indicators such that :

S(t) = [Close Price(t), Close Price(t) – Close Price(t-1), SMA 20-days(t), Close Price(t) – SMA 20-days(t), SMA 20-days(t)- SMA 100-days(t), EMA 20-days(t), Close Price(t) – EMA 20-days(t), EMA 20-days(t)- EMA 100-days(t), OBV 14-days]

On Balanca Volume :
OBV is a momentum indicator that uses volume flow to predict changes in stock price, measuring buying and selling pressure.

Formula :

If the Close Price(t) > Close Price(t-1) then:
Current OBV = Previous OBV + Current Volume

If the Close Price(t) < Close Price(t-1) then:
Current OBV = Previous OBV - Current Volume

If Close Price(t) = Close Price(t-1) then:
Current OBV = Previous OBV (no change)

We chose those indicators as input because they seem to be the most relevant variables in our Neural Network Model.

Moreover, we work with a sequence prediction problem (a sequence of multiple steps as input mapped to quantity prediction), therefore RNN and especially LSTM, is perhaps the most adapted neural network for our problem (for time series data).

Then we initialize our LSTM model.//
We use 3 layers of 64 neurons, a dropout equal to 0.3 (the optimal one), lookback = 7 number of time steps, a linear activation function (so we can have range of real-valued outputs), and the optimizer 'Nadam'.

We also set the number of actions to 3 (buy, sell and hold), and the reward are only the money obtained (loss or gain) after performing each action.
For example, if the agent buys when the stock prices are rising, then the agent has performed the best action to do, so he gets a reward (money), and will probably do the same on

a similar situation.

If the trend was going up and the action was selling then the reward would be negative : the agent will learn to forget this action and next time it will perform another one.

First we trained our agent on 5 episodes here : our agent trains on the training set during 4 episodes (learning best actions to perform), and applies it all in final on the test set.

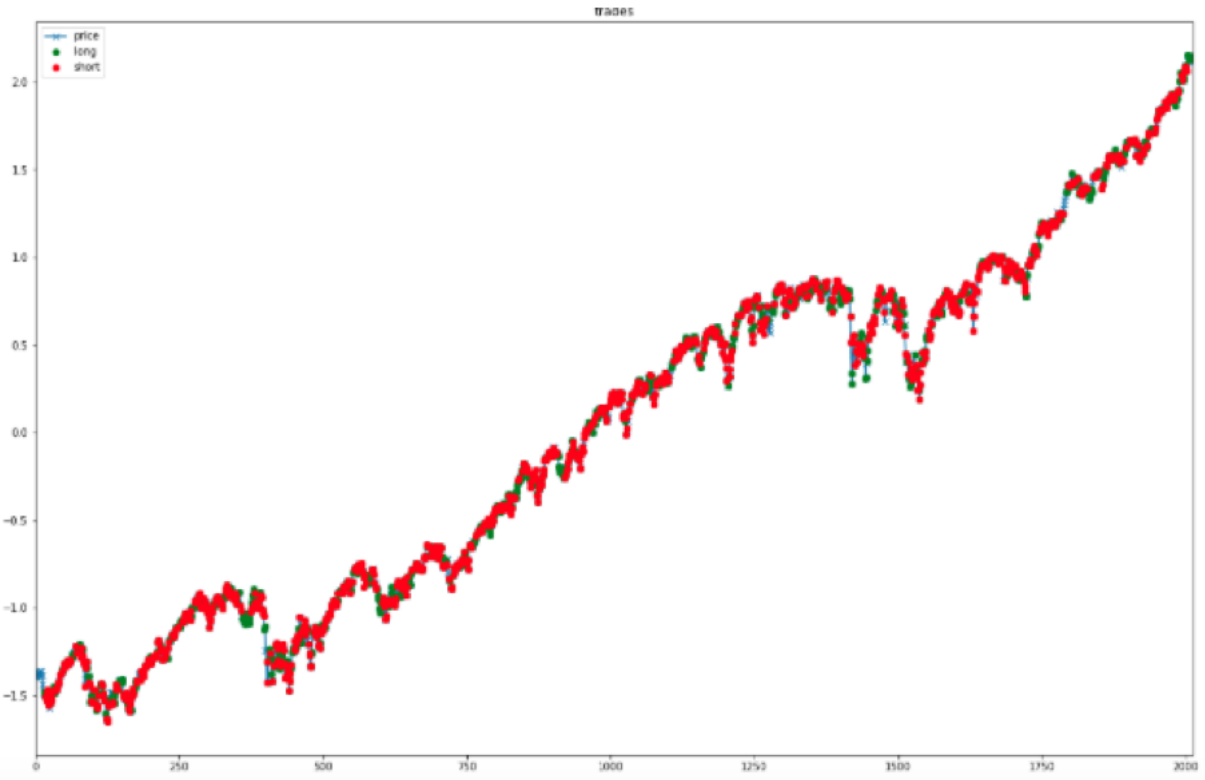


Figure 17: Trading actions after 1 episode.

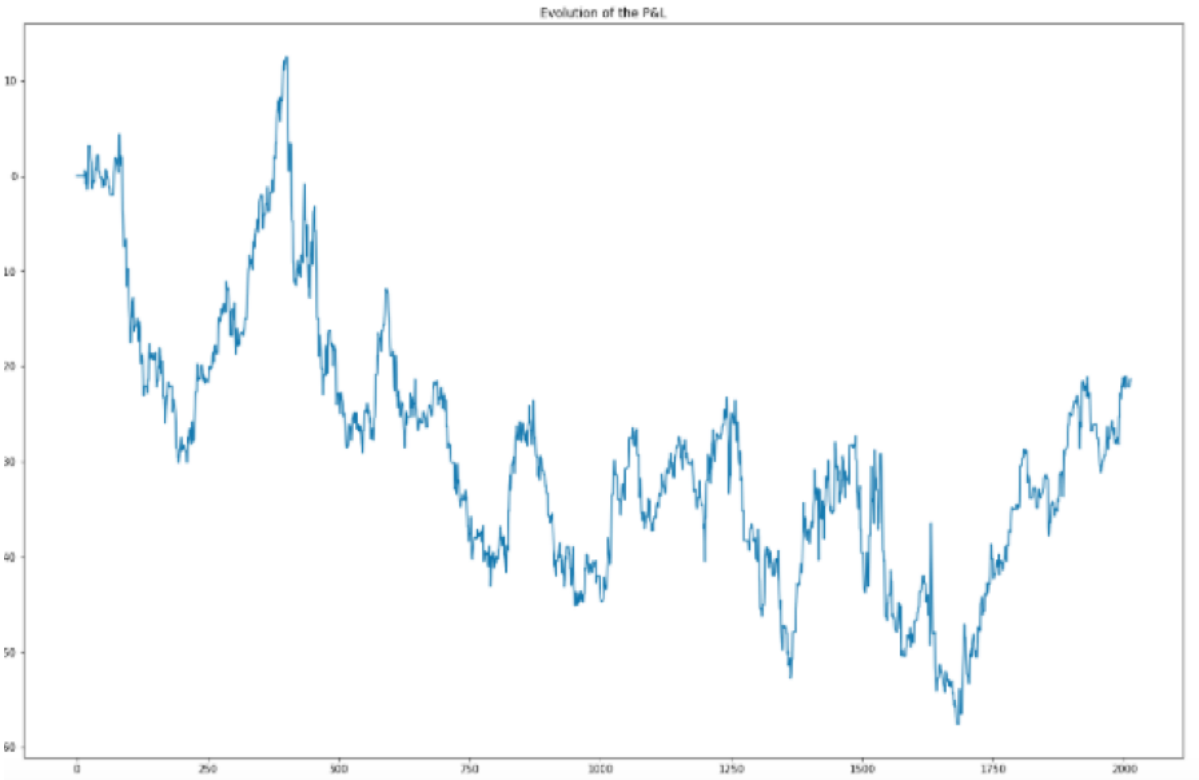


Figure 18: Evolution of PnL after 1 episode.

In the beginning, the agent does really badly (loss of 20 points after the 1st episode here). But over time, it begins to associate frames (states) with best actions to do. The agent starts trading from the 15th day (because we only have NA values for the first fourteenth OBV 14-days data).

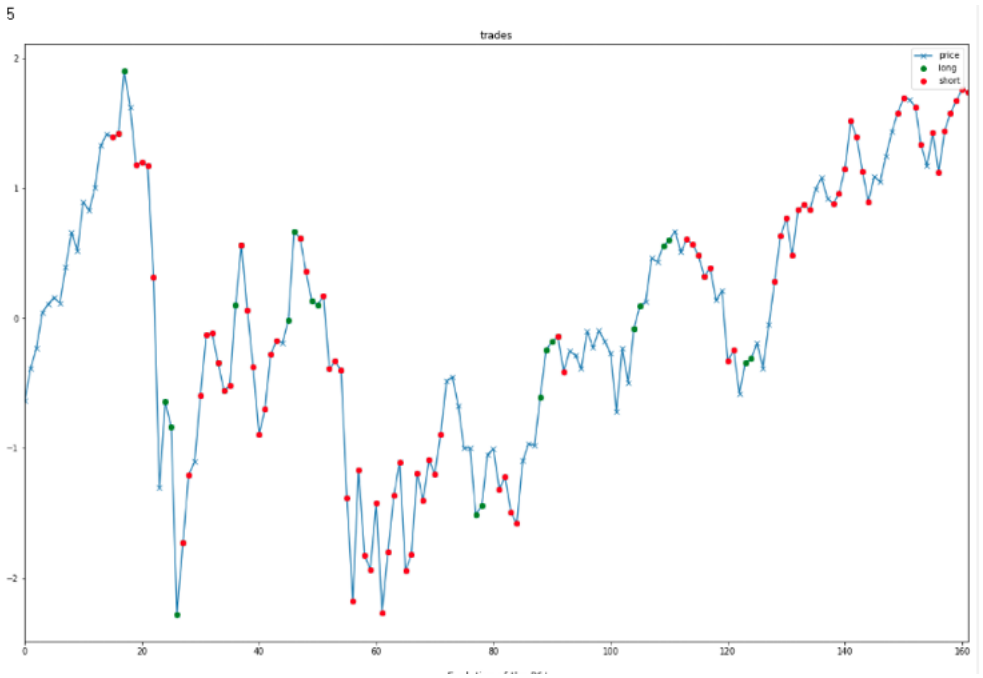


Figure 19: Trading actions after 5 episode.

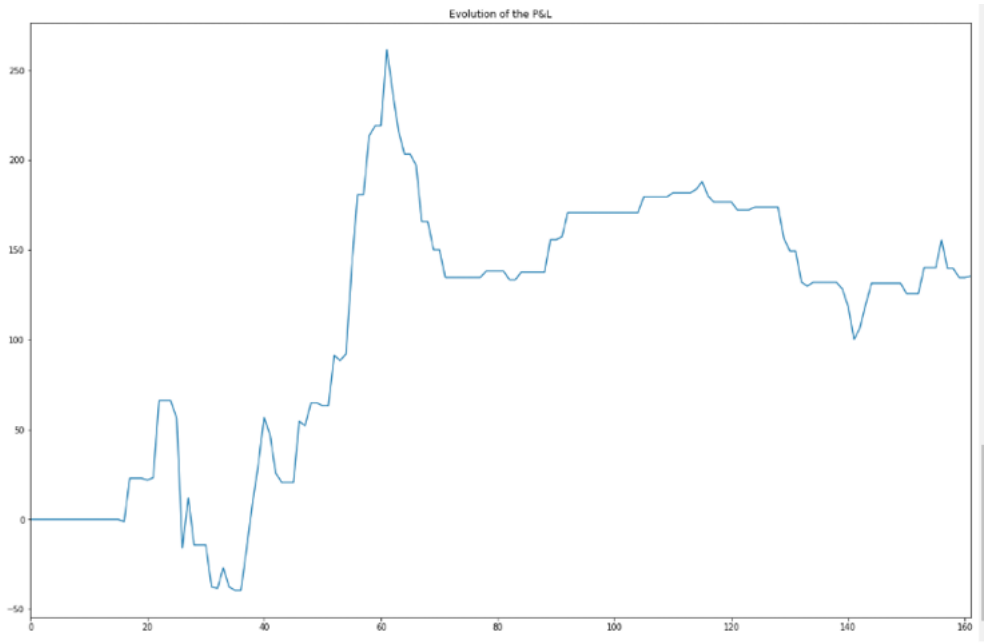


Figure 20: Evolution of PnL after 5 episode.

We can see that it is getting better on the test set (positive profit : +148 Points).

Moreover, if we add the number of episodes, we can see that the profit is getting better, thus we can assume that the agent learns how to perform better and forget actions during learning episodes.

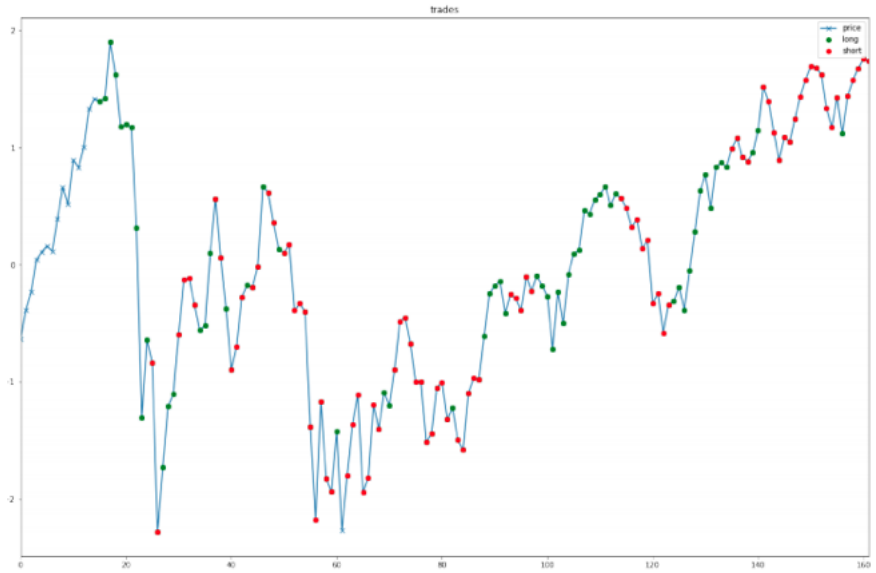


Figure 21: Trading actions after 15 episode.

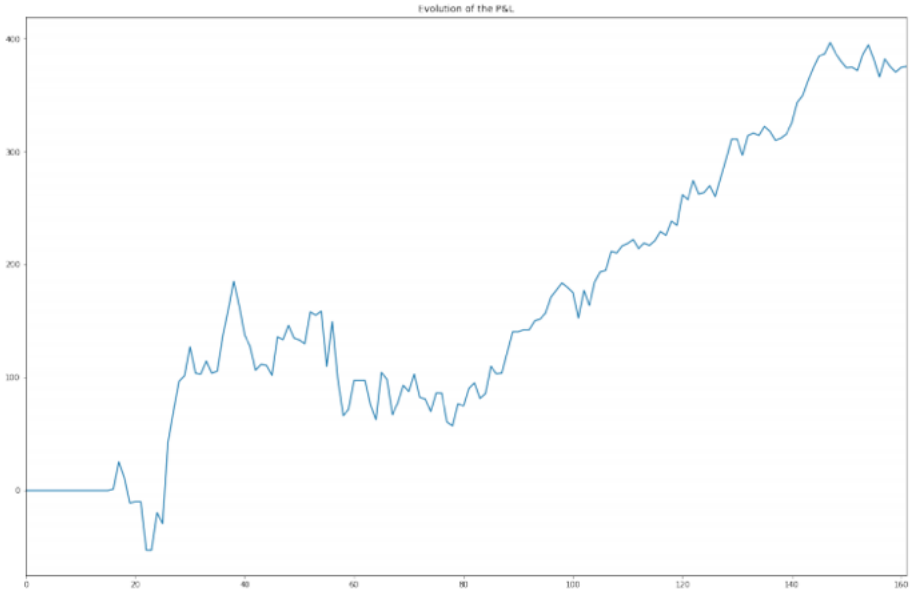


Figure 22: Evolution of PnL after 15 episode.

This one with a profit of 400 Points with 'only' 15 episodes, thus maybe we can have a better profit if we add the number of episodes (training the agent can take a lot of time, around 3 minutes the episode if we have a performant CPU), and we will be able to build a good trading strategy. We can compare our strategy to a benchmark trading strategy, using a Monte Carlo simulation :

-We first create a benchmark algorithm, with an initial amount of cash (we chose 10000$ here), where every day sort a random value (0, 1 or 2). If the value is 0, we buy 1 share, if the value is 1 we sell 1 share, if the value is 2 we hold and do nothing. We repeat the procedure every trading day during 8 months, then we collect the profit (in points) and we store the value

of the profit (profit in points + the initial cash) in a table.

- We repeat this procedure X times (here we chose 50 000 reiterations)

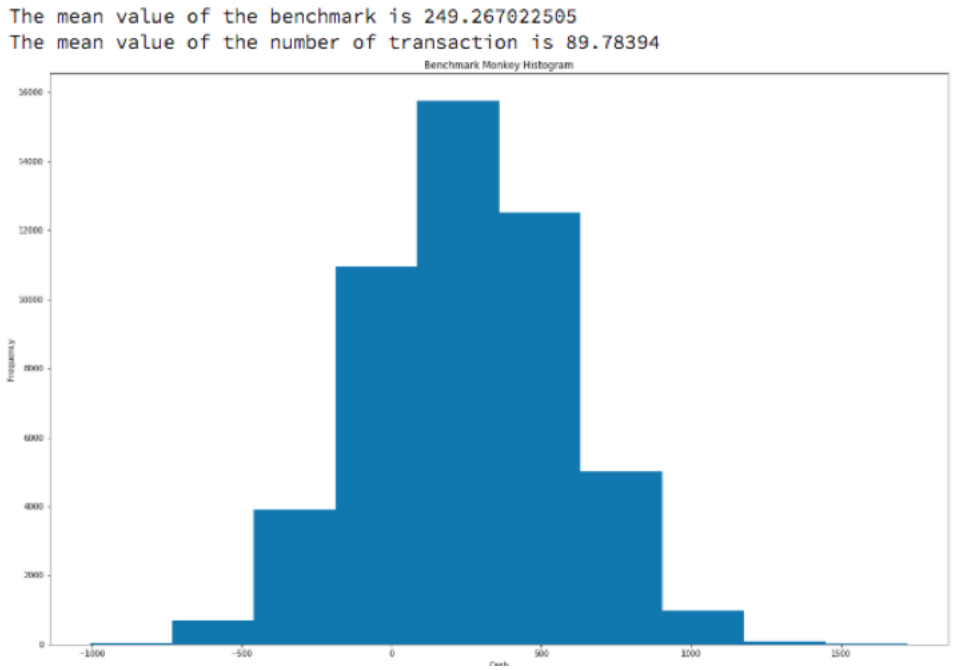
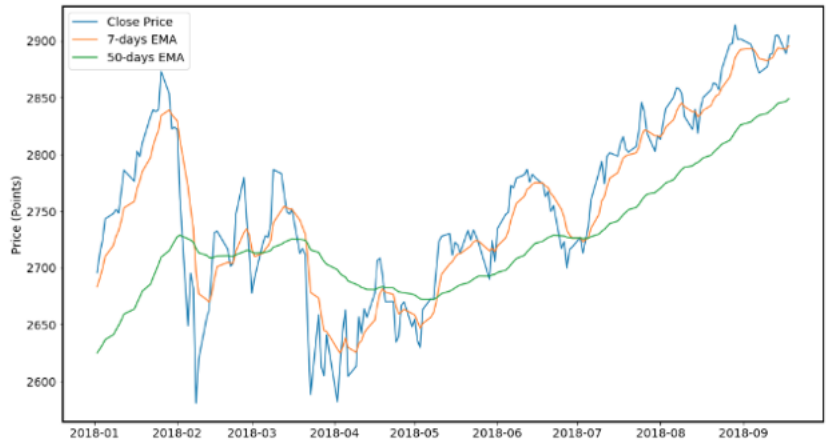


Figure 23: Monte Carlo Simulation.

We can see that the histogram tends to be a Gaussian distribution histogram (when the number of iterations tends to larger number), with a mean around 250 Points.

Thus, we can say that our Deep Reinforcement Learning strategy outperforms a normal random trading strategy.

We can also compare our strategy to a Moving Average Crossover strategy (starting from 2018-01-01 this time) :




```
count_transactions_ma = 0
strategy_EMA(data, ema_short, ema_long , 10000, count_transactions_ma)
```

```
 1    118
-1     33
 0     29
Name: 0, dtype: int64
(225.68017578125, 7)
```

Figure 24: Moving Average Crossover Trading Strategy.

Thus we earned 226 Points with 7 transactions. Given that one SP 500 point is equivalent to 50$ and the transaction cost is around 40$, we can say that :

$FinalProfit_{MVC}$ = 226 Points*50 – 40*7 = 11.020 $

For our Deep Reinforcement Learning's, given that the agent entered transactions every day since the 15th day, our final profit is equal to :

$FinalProfit_{DRL}$ = 400 Points*50 – 40*(160-14) = 14.160$

So we made 14.160$ in profit trading every day during 8 months, 3000$ dollars more than a Moving Average Crossover strategy (which is quite a robust trading strategy), using Deep Reinforcement Learning ! Of course the profits seems to be 'so good to be true' : we know that our methods has been used upon existing data, but it's a good start because we found that Deep Reinforcement Learning outperforms most of trading strategies and might be useful for making good trade decisions.

References

[1] Bakhache, B. and Nikiforov, I. (2000) Reliable Detection of Faults in Measurement Systems, International Journal of Adaptive Control and Signal Processing, 14, 683-700.

[2] Richard S Sutton and Andrew G Barto. Introduction to reinforcement learning, volume 135. MIT Press Cambridge, 1998.

[3] Csaba Szepesvari. Algorithms for reinforcement learning. Synthesis lectures on artificial intelligence and machine learning, 4(1):1–103, 2010.

[4] Broomhead, D.S. A and King, G.P. (1986) Extracting Qualitative Dynamics from Experimental Data. Physica D, 20, 217-236

[5] C.J. Watkins, P.Dayan, Technical note: Q-Learning, Machine Learning,vol.9, 1992

[6] L. Kaelbling, M. Littman, A. Moore, Reinforcment Learning" A Survey, Journal of Artificial Intelligence Research, vol.4, 1996

[7] Elsner, J. and Tsonis, A. (1996) Singular Spectrum Analysis: a New Tool in Time Series Analysis, Plenum Press, New York.

[8] Fraedrich, K. (1986) Estimating the Dimension of Weather and Climate Attractors, J. Atmos. Sci., 43, 419-432.

[9] Golyandina, N., Nekrutkin, V. and Zhigljavsky, A. (2000) Analysis of Time Series Structure: SSA and Related Techniques, London: Chapman And Hall.

[10] Lai T.L. (1995) Sequential Changepoint Detection in Quality Control and Dynamical Systems, Journal of Royal Statistical Society, B, 613658.

[11] Leadbetter, M.R., Lindgren, G., And Rootzen, H. (1983) Extremes and Related Properties of Random Sequences and Processes, Springer Series in Statistics, Springer-Verlag.

[12] Mathai, A.M., and Provost, S.B. (1992) Quadratic Forms in Random Variables: Theory and Applications, Marcel Dekker.

[13] Mehr, C.B., Mcfadden, J.A., (1965) Certain Properties Of Gaussian Processes and Their First Passage Times, Journal of Royal Statistical Society, B, 505-522.