Q-learning and Recurrent Reinforcement Learning for Asset Trading
Machine Learning for Finance

Supervised by Romuald Elie

Eléonore Blanchard
ENSAE Paris
eleonore_blanchard@ensae.fr

Yannick Ly
ENSAE Paris
yannick.ly@ensae.fr

Ines Deguy
ENSAE Paris
ines.deguy@ensae.fr

Jing Tan
ENSAE Paris
jing.tan@ensae.fr

Abstract

In recent years, artificial intelligence has become more and more popular in quantitative finance. One main subject is the applications of machine learning algorithms into trading. In this project, we focus on asset trading to implement reinforced algorithms (recurrent reinforcement learning, Q-learning and SARSA) and compare how well they deliver to produce trading strategies.

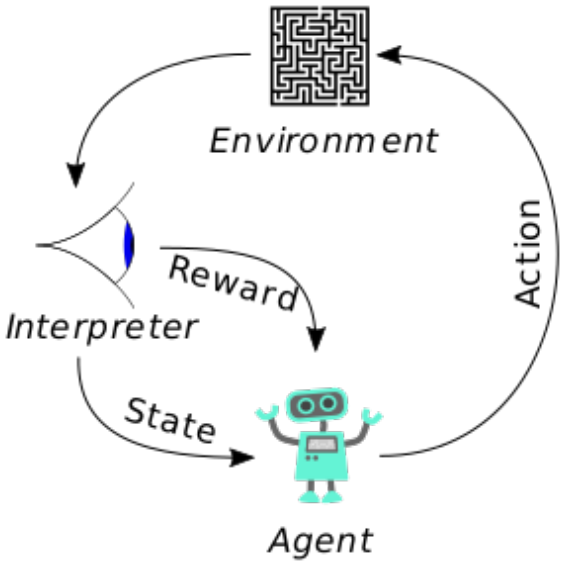
1 Introduction

Artificial intelligence has been trending lately in financial trading. This is why it is not surprising to see that relatively new subjects in AI such as reinforcement learning and deep learning are now being experimented in the financial industry. In this project we will try to implement algorithms using those notions in order to predict the rise and fall of asset prices before they occur and then to build a trading strategy that beats the market.

2 Theory

2.1 Reinforcement learning basic concepts

An agent gets some rewards from a sequence of actions starting from a given state in an environment. Then the agent uses these rewards and observations to learn an optional policy for the environment in order to maximise its expected total reward. Reinforcement learning tries to model the complex probability distribution of rewards in relation to a very large number of state-action pairs.



There are several key concepts, among which the most important are reward, action, state and environment.

2.1.0.1 Actions (a) : Actions are moves of agents. At time t before we decide which move to take, we have t observations $\{x_1, x_2, ..., x_t\}$, and $t-1$ past actions $\{a_1, a_2, ..., a_{t-1}$. The action a_t is then generated from the policy network.

2.1.0.2 States (s) : The state at each time step is composed of the past observations and actions $s_t = (x_1, a_1, ..., x_{t-1}, a_{t-1}, x_t)$. To simplify, we suppose that the states are fully observed. In our case the prices of our assets can fully reflect market conditions, which coincides with the efficient market hypothesis.

2.1.0.3 Environment (E) : The environment E is a Markov decision process between state space $\mathcal{S}$ and action space $\mathcal{A}$ with an initial state distribution of $P(s_1)$ and transition dynamic $P(s_{t+1}|s_t, a_{t-1})$. In other words, the environment contains information about the investment rules. It takes into account the current positions in our port-

folio, market prices. In the input, the environment takes the re-balanced weights according to our investment policy then returns as output our reward and the next state.

2.1.0.4 Reward (r) : The reward is a feedback of our action and state from the environment. In our problem, this represents the return of our portfolio. The future returns R_t of a sequence of actions and states is thus:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_i, a_i)$$

where γ is the discount factor of future rewards.

2.1.0.5 Policy (π) : The policy maps states to actions. It determines the strategy that the agent employs to determine the next action based on the current state : $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. We discount rewards, or lower their estimated value, the further into the future they occur.

2.1.0.6 Q-value (Q) : Q-value is the expected long-term return given the current state and current action.

$$Q^{\pi}(s_t, a_t) = E_{r_{i>t}, s_{i>t} \sim E, a_{i>t} \sim \pi}[R_t | s_t, a_t]$$

It refers to the long-term return of taking action a under policy π from the current state.

More explications on reinforcement learning can be found on this website: Reinforcement Learning Definitions.

2.1.0.7 Model-free vs Model-based : The model stands for the simulation of the dynamics of the environment. That is, the model learns the transition probability $P(s_1|s_0, a)$ from the pair of current state s_0 and action a to the next state s_1. If the transition probability is successfully learned, the agent will know how likely to enter a specific state given current state and action. However, model-based algorithms become impractical as the state space and action space grows.
On the other hand, model-free algorithms rely on trial-and-error to update its knowledge. As a result, it does not require space to store all the combination of states and actions. All the algorithms discussed in the next section fall into this category.

2.1.0.8 On-policy vs Off-policy : An on-policy agent learns the value based on its current action a derived from the current policy, whereas its off-policy counter part learns it based on the action a^* obtained from another policy. In Q-learning, such policy is the greedy policy. (We will talk more on that in Q-learning and SARSA)

2.1.0.9 Policy-based vs Value-based In Policy-based methods we explicitly build a representation of a policy (mapping $\pi : s \rightarrow a$) and keep it in memory during learning.
In Value-based we don't store any explicit policy, only a value function. The policy is here implicit and can be derived directly from the value function (pick the action with the best value).

2.2 SARSA

Value learning or value based reinforcement learning is a fundamental concept in reinforcement learning. It is the entry point to learn RL and as basic as the fully connected network in Deep Learning. It estimates how good to reach certain states or to take certain actions. While it may not be sufficient to use value-learning alone to solve complex problems, it is a key building block for many RL methods.
The value function $V(s)$ measures how good to be in a specific state. By definition, it is the expected discounted rewards that collect totally following the specific policy.
On-policy Value based reinforcement learning such as State–action–reward–state–action (SARSA) is an algorithm for learning a Markov decision process policy, used in the reinforcement learning area of machine learning. SARSA learns the Q-value based on the action performed by the current policy.
This name reflects the fact that the main function for updating the Q-value depends on the current state of the agent "s_1", the action the agent chooses "a_1", the reward "R" the agent gets for choosing this action, the state "s_2" that the agent enters after taking that action, and finally the next action "a_2" the agent chooses in its new state. The acronym for the quintuple $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ is SARSA.
Our tabular SARSA agent is based on the description of the SARSA Q-update rule in Sutton and Barto's book.
In the algorithm, we can see that a certain policy needs to be used to generate actions. There are two approaches that can be used as policy to generate the action:

1. Randomly generate an action

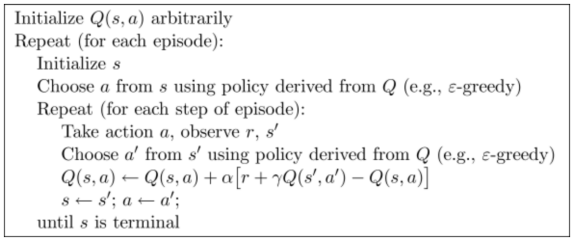


Figure 1: SARSA on policy TD control

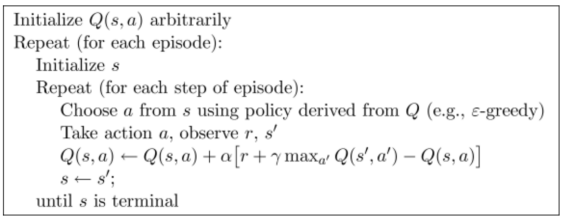


Figure 2: Q Learning off policy TD control

2. Calculate an optimal action based on the current Q value.

This policy is called $\epsilon - greedy$ policy. The use of random actions is exploration, which is to explore the effects of unknown actions, which is beneficial to update the Q value and obtain a better policy. The use of greedy policy, which is the target policy, is an exploitation. Using policy, it is relatively difficult to update a better Q value, but we can get better test results for judging whether the algorithm is effective.
Combining the two is the so-called $\epsilon - greedy$ strategy. ϵ is generally a very small value, which is used as the probability of selecting random actions. You can change the value of ϵ to get different ratios of exploration and exploitation.
One point to note here is that using the $\epsilon - greedy$ strategy is an extremely simple and crude method, and it is not advisable to use this method to explore unknown spaces for some complex tasks. Therefore, there are more and more methods to improve this exploration mechanism recently. $\epsilon - greedy$ policies ensure that agents continue to take random actions even once the true Q-values have been learned. The SARSA algorithm will learn the optimal $\epsilon - greedy$ policy, i.e, the Q-value function will converge to a optimal Q-value function but in the space of $\epsilon - greedy$ policy only (as long as each state action pair will be visited infinitely). We expect that in the limit of ϵ decaying to 0, SARSA will converge to the overall optimal policy.

2.3 Q-learning

Q-learning is an off-policy, model-free reinforcement learning algorithm where the environment does not need any initial model. As we have seen previously, the environment is modeled by a Markov decision process, but the agent does not know it, and it is not used in the algorithm. This algorithm is based on the Bellman equation.
Here we introduce a value iteration method for the update of the Q function. In this case, the algorithm also require exploration to converge. And the Q-learning algorithm will converge to q^* which is the optimal Q-value.

2.4 Recurrent Reinforcement Learning

Recurrent reinforcement learning (RRL) is a hybrid model (Li, 2015) that uses recurrent neural networks (RNN) models to learn the representation of states for reinforcement learning. Because there is some hidden states in the environment, being the financial markets, we can not infer the Q-function easily. This is why we require a hybrid model that combines a RNN for hidden-state representation learning and a a DQN for policy learning.

2.4.0.1 Recurrent Neural Networks (RNN): A RNN is a neural networks where the inputs include previous outputs. Therefore, the outputs are not only influenced by weighs applied on inputs, but also by hidden states representing the context based on prior observations. Therefore, such algorithm can be employed with sequential data, including time series.
The RNN can be used to learn hidden-state representation because we can feed it supervised historical data and the network will learn useful internal features. Thus the RNN learns hidden states from signals including next observations and rewards.

2.4.0.2 Deep Q-Network (DQN): Q-learning learns the Q-value function ($Q(s,a)$), which depends on state s and action a. Therefore, applications of deep neural network can help approximate this function thus optimize the criteria for a sequence of portfolio. The goal of the agent is to choose actions that maximize the Q-value.
The deep reinforcement learning architecture treats asset allocation as a problem of continuous control of portfolio policy which attempts to maximize the delayed reward. It enables the process to learn the best actions possible in virtual environment in order to attain their goals. Thus it unifies

function approximation and target optimization. The Q-value updating formula is as below:

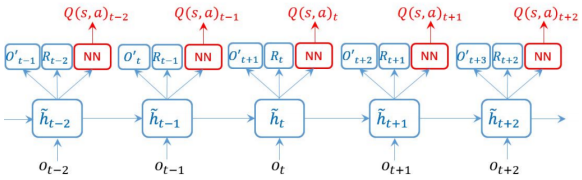
$$Q(s',a) \leftarrow (1-\alpha)Q(s,a)+\alpha(r+\gamma \max_{a'} Q(s',a'))$$

where α is the learning rate and the next action a' is the one that maximize the next state's Q-value. For more details, please check the following web pages: A Beginner's Guide to Deep Reinforcement Learning and RL — DQN Deep Q-network. Therefore, here the DQN uses the hidden states learned by the RNN as inputs in order to learn the Q-function of a near-optimal policy.

Moreover, we must not separate the training of the two networks. At each epoch of the algorithm, we iterate the two training steps:

1. Train a RNN which learns hidden states
2. The learned hidden states become the input for the DQN, which learns the Q-function

The algorithm can grossly be seen as follow:



where:

- the RNN is in blue
- the DQN is in red
- o_t is the observation
- $\tilde{h}_t$ is the hidden state for RNN
- o'_{t+1} is the predicted observation for time $t+1$
- R_t is the predicted reward
- $Q(s,a)_t$ is the predicted Q-value at time t

We can see that the hidden layer ($\tilde{h}_t$) predicted by the RNN are the DQN's input at time t. And the previous hidden layers ($\tilde{h}_{t-1}, \tilde{h}_{t-2}$...) are the inputs for the RNN at time t.

3 Implementation

In this section, we are going to take a more profound look on how the algorithms that we have previously presented can be implemented for asset trading. First, we will present our data, then we will talk about the joint implementation of Q-learning and SARSA algorithms, and finally we will see how we have implemented an algorithm using Recurrent Reinforcement Learning,

3.1 Data

The data we have used to build our signals are daily prices of asset indices. We have decided to first implement our models on the prices of 4 of the major world financial market indices: CAC 40, Dow Jones Industrial Average, SP 500 and NASDAQ Composite, from January 1, 1990, to May 16, 2020. This time frame allows us to have sufficient data to build our artificial neural networks (7653 rows) and to have a history of the behaviour of the financial markets during some crisis (the internet bubble in 2000, the subprime crisis in 2008...). This is a way to make sure that we are going to take into account different economic cycles while training our model.

3.2 Implementation of SARSA and Q-learning based Reinforcement Learning

As reviews, we consulted previous algorithmic trading implementations. For instance, some researchers use multiple Q-Learning agents to trade stocks, and assign the tasks of stock pricing and stock selection to different agents. In a different paper they stress the central difficulty in algorithmic trading as being the representation of states; the paper (Corazza and Bertoluzzo, 2014) mentioned this, noting the importance of their discretization of the state space for the success of their trading agent.
In constructing our reward function, we decided to use the portfolio return as the reward. We make our agents' rewards the returns implied by the product of their portfolio position and the daily percentage change in the asset's price.

3.2.1 Problem Framing

Our goal was to implement two agents based on SARSA algorithm and Q-learning algorithm that trades a single asset optimally. The agents should generate total returns as close as possible to some theoretical limit determined by the performance of the underlying asset. In testing, this meant generating total returns as close as possible to the limit determined by an asset's historical performance over the testing period, while in general it means generating maximal returns over an unbounded

horizon (in the sense that our trading agents could continue trading indefinitely and thus indefinitely generate a return on some initial investment).
In the original paper, the author take the asset return in the states. However, this leads to a large Q table and thus a high dimension. Some researchers use clusters of returns instead of real returns, we round the asset returns to thousands thus reduce the dimension of our Q table.
For our implementation, we specified the problem as choosing the return-maximizing action on every trading day and in a particular state over all corresponding state-action pairs, where the action set contained the actions ['buy', 'sell' and 'hold']. To be able to choose the return-maximizing action we naturally also needed to determine the anticipated returns for taking an action in a particular state, meaning that our problem constituted both learning state-action values (observation) and subsequently selecting actions (action). In particular, given an initial estimate of the value of a state-action pair, we needed to update our estimate of a pair's value using a transition and reward sample, and then determine the return-maximizing action. Thus the objective of our trading agents was to learn a policy, online, that maximized our agents' returns.
A decision was made to generate successor states. Since our simulation environment does not involve transaction costs, our current state can only be related to the successor state through the portfolio position implied by our agent's actions (that is, we know that if our agent sells in the current time period our agent's portfolio position in the next time period will necessarily be 0). However, because our agent's purchasing and selling decisions do not impact the traded asset's subsequent percentage change in price (according to economic theory, unless our agent's starting capital is artificially inflated) the successor state, as a tuple of our agents' portfolio position and the day-on-day percentage change in asset price, is not fully specified by our agent's action in the present state. As a simple solution we therefore decided to let our agents assume that a successor state equals the portfolio position implied by a particular action in the present state, and the current day-on-day percentage change in asset price projected into the next day (that is, we decided to project a trend).

3.2.2 Tuning of hyperparameters

We first tune our trading agents by finding the optimal tuple of hyperparameters (α, γ) for a specified asset, where α is the learning rate and determines by how much our algorithms update their current estimation of the value of a state-action pair using the most recent transition and reward sample, and γ is the discount factor applied to future returns and determines how much our agent values future returns relative to immediate returns. We execute several test runs for every combination of α and γ at 0.05 increments, where α and γ are bounded such that $0.05 \leq \alpha \leq 0.35$ and $0.85 \leq \gamma \leq 1$ (these bounds were determined through quick experimentation). We record the total returns generated as well as the standard deviation of the total returns for each parameter pair, because we wish to value consistency (this is particularly important in a trading context where real funds are at stake and trades cannot be executed multiple times in simulation). We ultimately selected values of and that maximized the difference average total returns standard deviation o f total returns (where this criterion was selected to trade off high average performance against consistency).
In terms of ϵ we test ϵ in 0.01, 0.05, 0.10, 0.15. And we decided to use a larger ϵ in the training period than in the testing period to exploit most possible states possible during the training and to ensure the stability of results in the test.

3.3 Implementation of Recurrent Reinforcement Learning

In order to implement the Recurrent Reinforcement Learning (RRL) algorithm on our data, we have followed a paper (Bertoluzzo, 2007) which uses the reciprocal of the returns weighted direction symmetry index instead (RRWDS) of the Sharpe ratio.
As we have previously seen, a RRL model is made up of two components: the supervised learning component(a recurrent neural network here) which learns the hidden-state and the reinforced learning component (a deep q-network here) which learns the policy using this hidden state as an input. We also have seen that the two models need to be trained jointly. Therefore we cannot implement them separately.
In the following subsections, we are going to show how we have implemented the RNN and the DQN, and how they work together.

3.3.1 The Framework

This section depicts the learning of an agent that interacts dynamically with an environment and take actions. More specifically, we can say that a financial trading system (the agent) is creating a strategy (the policy) to to place orders (the actions) on the financial market (environment).
The financial trading system (FTS) is a tool for investors who are trying to make decisions under uncertainty. It works as follow:

1. identify the interesting variables (asset prices, volume...)
2. extract information from their past and current values to predict their future value
3. use the information and prediction to implement a trading strategy

Therefore, thanks to the characteristics of the FTS, we can use it as an agent that will try to build a trading strategy (follow a policy) in order to increase its expected final gain (expected final reward).
The environment that the agent is interacting with has some uncertainty, and therefore has hidden states.

3.3.2 The Hidden State (y_t)

Because we are working with financial prices, there is a hidden-state, and this is the reason why we cannot implement a vanilla reinforcement learning algorithm. We need a neural networks (NN) that will learn the hidden state from signals, using historical data for its training (hence the need for a broad data-set). The author consider a simple perceptron with no hidden-layer, simply using a $\tanh$ function as the squashing function, in order to obtain values included in $[-1, 1]$, so that the weights in the network converge to a stable solution:

$$y_t = \tanh\left(\sum_{i=0}^{M} w_{i,t} x_{t-i} + w_{M+1,t} F_{t-1} + w_{M+2,t}\right)$$

where:

- $w_{0,\tau}, ..., w_{M+1,\tau}$ are the weights of the NN at time τ
- $x_\tau, ..., x_{\tau-N}$ are the current and past values of the logarithmic rate of return of the index at time τ
- $w_{M+2,\tau}$ is the threshold of the NN at time τ

3.3.3 The policy

The policy is the trading strategy of the FTS. It is determined by the following function:

$$F_t = \begin{cases} -1 & \text{if } y_t < 0 \\ F_{t-1} & \text{if } y_t = 0 \\ 1 & \text{if } y_t > 0 \end{cases}$$

We have implemented it in the function `positions` which takes as inputs $x_\tau, ..., x_{\tau-N}$ (the current and past values of the logarithmic rate of return of the index at time τ) and $w_{0,\tau}, ..., w_{M+1,\tau}$ (the weights of the NN at time τ) and returns the values of $y_{0,\tau}, ..., y_{M+1,\tau}$ and $F_{0,\tau}, ..., F_{M+1,\tau}$.
In our implementation, you will see that we have used matrix to measure y_t. Hence, the formula becomes: $y_t = \tanh\left(\theta^T x_t\right)$, where:

- θ will be the parameters we will optimize using gradient ascent
- $x_t = [1, r_{t-M}, \ldots, r_t, F_{t-1}]$, where :
  - r_t is the change in value between the asset at time t and $t-1$
  - M is the number of time series inputs

This means that at every time step, the model will be fed its last position and a series of historical price changes that it can use to calculate its next position.

3.3.4 The reward

The net reward at generic time period $(t-1, t]$ is defined in the paper as follow:

$$R_t = \mu[F_{t-1} r_t - \delta \mid F_t - F_{t-1} \mid]$$

where:

- μ is the amount of capital to invest
- r_τ is the geometric rate of return at time τ of the portfolio to trade
- δ is the per cent transaction cost related to the portfolio quota to trade

Thus the reward is the return of the portfolio, penalised by the cost of transaction.
We implemented the function `rewards` to compute the reward at each time step with inputs the positions (and thus the hidden state), the returns, the transaction cost and the amount of capital to invest.

3.3.5 The expected total reward

The expected total reward is calculated with the following formula:

$$CR_t = \sum_{i=1}^{t} R_i$$

where R_i is the reward at time i.
This formula is computed in the function `gradient` once the rewards have been computed for each time step. It then takes the sum of these values.

3.4 Investor's gain index

The investor's gain index is used to find the optimal weights in the RNN. Since the use of the Sharpe ratio as profitability measure is wide spread in the literature, the authors wanted to try to use another investor's gain index: the reciprocal of the returns weighted directional symmetry measure. It is the the ratio between the cumulative positive rewards and the cumulative negative rewards at each time t, given by the following formula:

$$I_t = \frac{\sum_{i=1}^{t} g_i \mid R_i \mid}{\sum_{i=1}^{t} b_i \mid R_i \mid}, with \sum_{i=1}^{t} b_i \mid R_i \mid \neq 0$$

where:

- $G_\tau = \begin{cases} 0 & \text{if } R_\tau \leq 0 \\ 1 & \text{if } R_\tau > 0 \end{cases}$
- $b_\tau = \begin{cases} 1 & \text{if } R_\tau \leq 0 \\ 0 & \text{if } R_\tau > 0 \end{cases}$

I_t is used to calculate the marginal variation of the agent's utility function : $D_t = U_t - U_{t-1} = I_t - I_{t-1}$. We can see that as t increases, computing I_t becomes harder. Therefore, we use an exponential moving formulation for I_t:

$$\tilde{I}_t = \frac{A_t}{B_t}$$

where:

- $A_\tau = \begin{cases} A_{\tau-1} & \text{if } R_\tau \leq 0 \\ \eta R_\tau + (1-\eta) A_{\tau-1} & \text{if } R_\tau > 0 \end{cases}$
- $B_\tau = \begin{cases} -\eta R_\tau + (1-\eta) B_{\tau-1} & \text{if } R_\tau \leq 0 \\ B_{\tau-1} & \text{if } R_\tau > 0 \end{cases}$

In our code, it is implemented by the function `rrwds` which uses the latter formulation.

3.4.1 Weights Updating

For every training loop, we need to update the weights of the recurrent neural network to improve its prediction improve. Since the neural network is a simple perceptron, the updating of the weight is defined by the following gradient ascent technique:

$$w_{i,t} = w_{i,t-1} + \rho_t \frac{dU_t}{dw_{i,t}}$$

with $i = 0, ..., M+2$ and where:

- $w_{i,t}$ are the weights of the RNN at time t
- ρ_t is the learning rate at time t
- U_t is the investor's utility function at time t

In order to perform gradient ascent, we must compute the derivative of the utility function (i.e `rrwds`) with respect to θ ($\frac{dU_t}{d\theta}$). Using the chain rule and the above formulas we can write it as:

$$\frac{dU_t}{d\theta} = \frac{dU_t}{dR_t}\left(\frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta}\right)$$

We have computed the gradient ascent in the function `gradient` using the latter function. The function then returns the gradient $\frac{dU_t}{dw_{i,t}}$ and we can finally calculate the value of $w_{i,t}$ and therefore update the weights in the neural network in the main function `train`.

3.4.2 The Drawdown-like Phenomenon Management

The authors of the paper (Bertoluzzo, 2007) want the FTS to be able to minimize large losses, and to guarantee the continuation of the trading when they occur. This is why they need to manage the drawdown-like phenomenon and guarantee the continuation of the financial trading in case of a large loss.
Therefore,they utilize as amount of capital to invest $\mu + \mu_0$ where μ_0 is the absolute value of the largest loss of the initial part of the first time sub-period. $\mu_0 = min[min_{0<t<t_{off}}[R_t : R_t < 0 \cap CR_t < 0 \cap sign(F_t F_{t-1}) = -1], 0]$ For carrying out this drawdown management, we articulate the whole trading time period (from 0 to T) in the sequence of overlapping time sub-period: [0,$\Delta t_{off} + \Delta t_{on}$], [$t_{on}$,$t_{off} + 2\Delta t_{on}$], .., [$T - \Delta t_{off} - 2\Delta t_{on}$, $T - \Delta t_{on}$], [$T - \Delta t_{off}$ - Δt_{on}, T]

- Δt_{off}: The length of the initial part of each time sub-period during which the FTA works in an off-line modality for performing the optimal setting of the considered parameters.
- Δt_{on}: The length of the final part of each t.

3.4.3 Tuning of the parameters

In order to make operative our financial trading system, we need to determine the optimal values of the parameters:

- M: the number of time series inputs
- δ: the per cent transaction cost related to the portfolio quota to trade
- ρ: the learning rate
- η: an adaptation coefficient

For carrying out this optimal setting, we consider the returns of S&P 500 and we split the dataset in a training and a test sets. For each parameters to optimize, we fix the others parameters and we train the model using 100 epochs. After the training is done, we apply our testing function and we keep the parameters whose value achieved the biggest CRT.

4 Results/Backtest

Different scenario (crisis, bull trends, bear trends, reverting, fluctuating, etc)

4.1 SARSA and Q-learning based Reinforcement Learning

To show the results of our agents, for example we investigate the daily prices of SP 500 for the period from its inception to December 2018. And we tested on the period from 2019/01/01- 2020/01/01 Data are retrieved from yahoo finance.
The results are shown in the following figures.

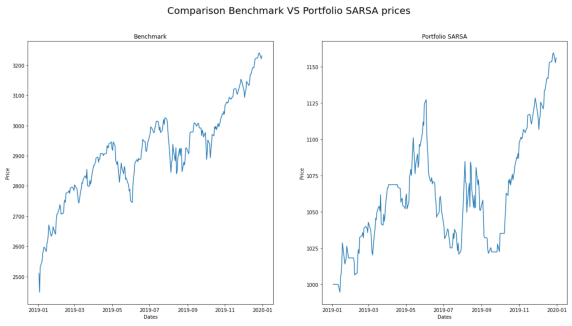


Figure 3: Performance of SARSA agent

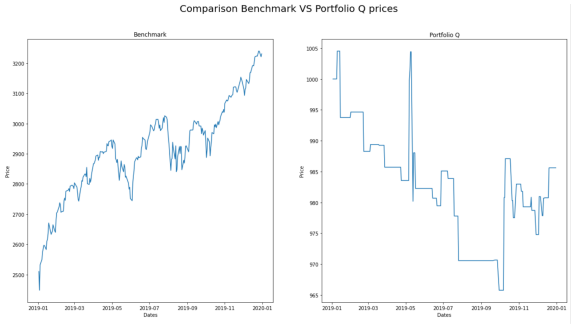


Figure 4: Performance of Q learning agent

The results are summarized in the following table. For reference, we allow short sellings and the SARSA agent leads to a performance of 15.2% gain per annum and the Q-learning agent has a negative annual return of -1.37%. However neither of the agent beats the benchmark.
The positions of the two agents are shown in the following figures. We also notice that the test is not stable since we don dispose enough data to train our model. When there is a ϵ in the SARSA algorithm, the two agents don't converge.

Indicator	Benchmark	SARSA	Q Learning
Annualized return	0.261	0.152	-0.014
Annualized volatility	0.125	0.1119	0.043
Sharpe Ratio	2.096	1.372	-0.322

Table 1: Performance of SARSA and Q-Learning

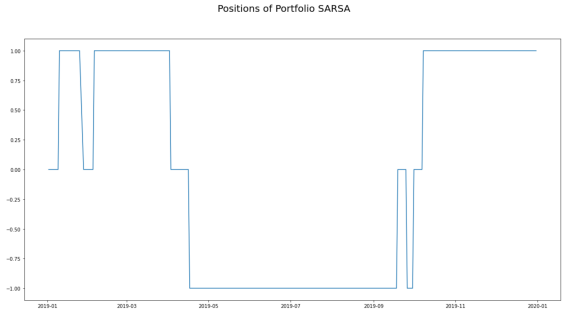


Figure 5: Positions of SARSA agent

4.2 Recurrent Reinforcement Learning

First of all, we started by tuning the hyperparameters, then we have trained and tested our model and finally we have computed different ratios in order to compare this model with SARSA and Q-learning models.

4.2.1 Choice of parameters

As stated in the paragraph 3.3.3, we have carrying out our optimal setting by considering the return

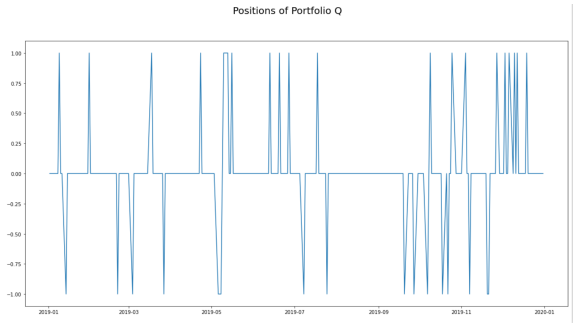


Figure 6: Positions of Q learning agent

of S&P 500. We have set 10% of our dataset to the training set and 90% to the test set. As far as regards the optimal setting of the parameters, we have determined that, in general:

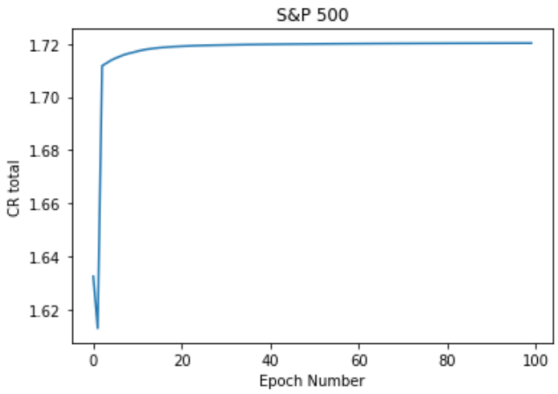
- $M = 7$
- $\delta = 0.05$
- $\rho = 0.1$
- $\eta = 0.01$

Our parameter results are very close to those found in the article.

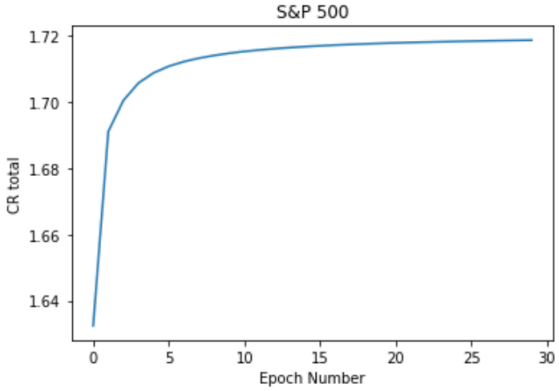
4.2.2 Results of the model

In this section, we have first of all try our model without using the management for drawdown-like phenomenon and then we have train the model considering it.
As stated in paragraph 3.1, the data we have used to build our signals are daily prices of asset indices. We used the prices of 4 of the major world financial market indices: CAC 40, Dow Jones Industrial Average, S&P 500 and NASDAQ Composite, from January 1, 1990, to May 16, 2020.
We have set 70% of our dataset to the training set and 30% to the test set and we are doing 100 epochs.

4.2.2.1 S&P500 We find that the Cumulative Reward Total S&P 500 for the test dataset is 0.913.



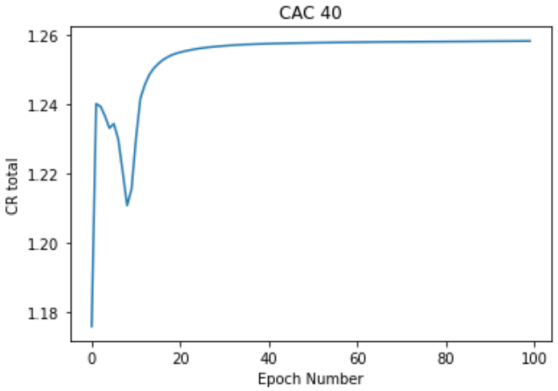
We find that the Cumulative Reward Total S&P 500 for the test dataset with using the management for drawdown-like phenomenon is 0.912.



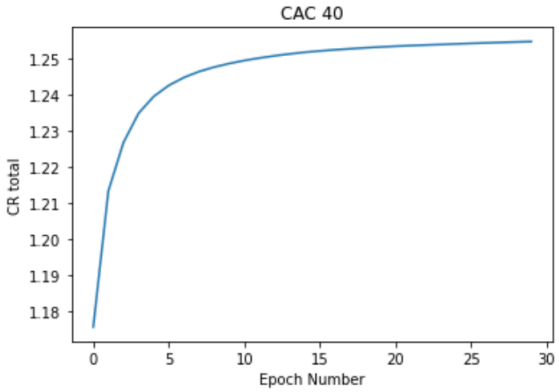
When the comparing our strategy with the benchmark on the SP500, we can see that our performance is not as good, even if our Sharpe ratio and annualized returns are good in the absolute.

	Benchmark	Portfolio
Ret annualized	0.261357	0.227370
Vol annualized	0.124720	0.117032
Sharpe Ratio	2.095539	1.942811

4.2.2.2 CAC40 We find that Cumulative Reward Total CAC 40 for the test is dataset 0.274.



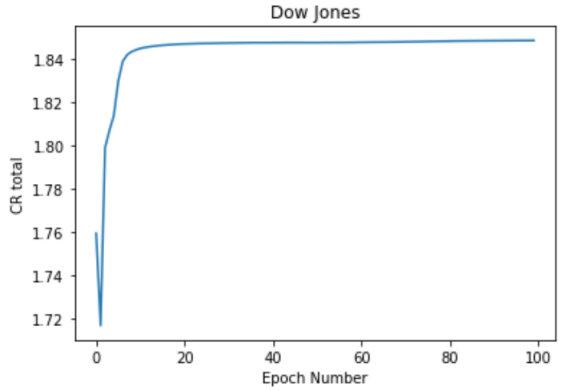
We find that the Cumulative Reward Total CAC 40 for the test dataset with using the management for drawdown-like phenomenon is 0.273.



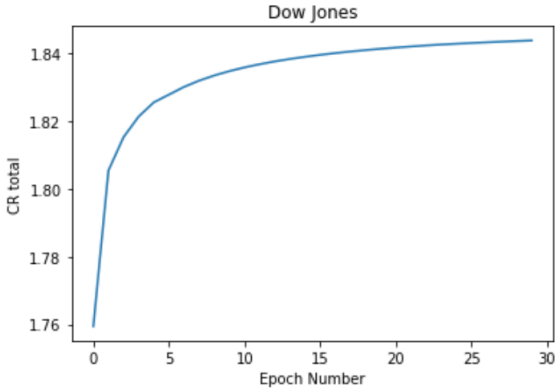
When the comparing our strategy with the benchmark on the CAC40, we see that the difference is not as broad as the one on the SP500. We show a good return with lower volatility which is very interesting.

	Benchmark	Portfolio
Ret annualized	0.249766	0.230707
Vol annualized	0.132235	0.127661
Sharpe Ratio	1.888803	1.807181

4.2.2.3 DowJones We find Cumulative Reward Total Dow Jones for the test dataset is 0.791.



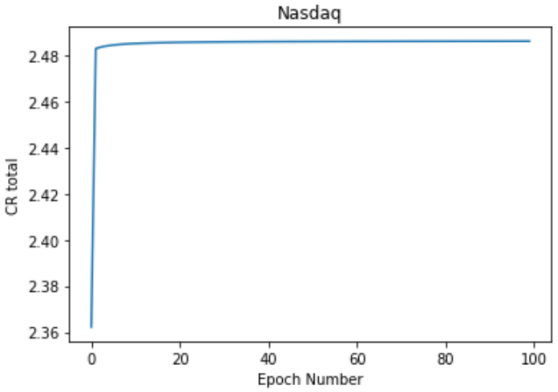
We find that the Cumulative Reward Total Dow Jones for the test dataset with using the management for drawdown-like phenomenon is 0.789.



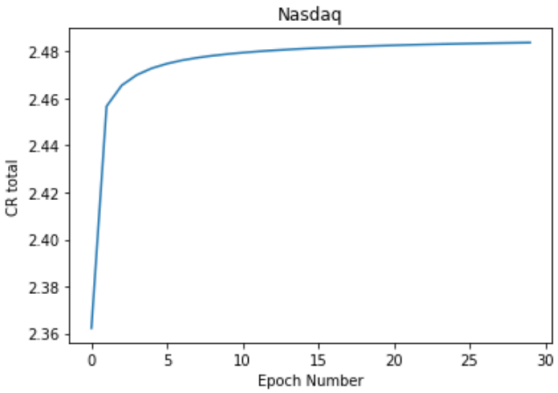
When comparing the performance of our portfolio against its benchmark, the Dow Jones, we can see that we are not sur-performing because we have lower returns and Sharpe ratio.

	Benchmark	Portfolio
Ret annualized	0.209449	0.181576
Vol annualized	0.124363	0.116065
Sharpe Ratio	1.684174	1.564431

4.2.2.4 Nasdaq We find that the Cumulative Reward Total Nasdaq for the test dataset is 1.35.



We find that the Cumulative Reward Total Nasdaq for the test with using the management for drawdown-like phenomenon dataset is 1.35.



When comparing the results on our portfolio with its benchmark, the NASDAQ, we can see that the portfolio fails at beating the market, even though it has lowered the volatility.

	Benchmark	Portfolio
Ret annualized	0.310805	0.265324
Vol annualized	0.156566	0.146911
Sharpe Ratio	1.985136	1.806013

4.2.2.5 Conclusion of the Results We found conclusive results with good cumulative reward. We can also notice that using a drawdown-like phenomenon management allows for no downward spike in the training curve.

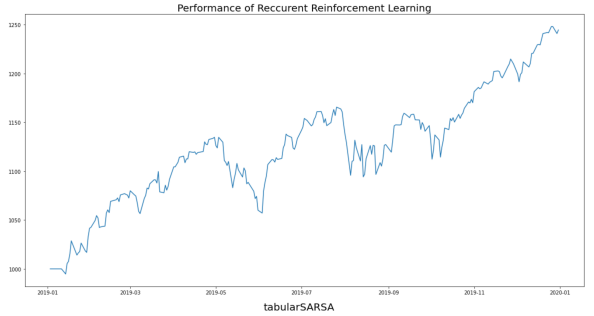
The results in the test dataset with using the drawdown management are close to the results without using it.

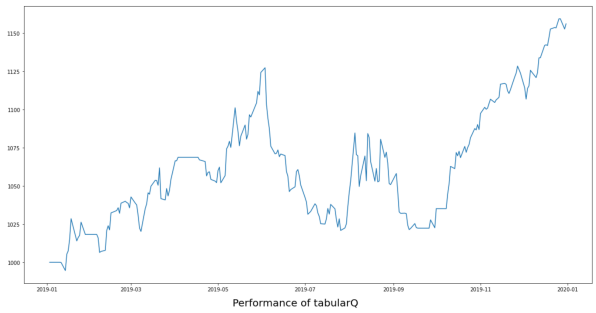
4.2.3 Other ratios

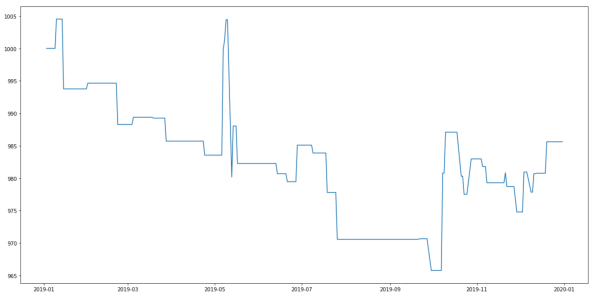
4.3 Comparisons

The following graphics give us an idea of how the three different algorithms worked with respect to the others. We can see that the best algorithm is the one using Recurrent Reinforcement Learning, with a final value around 1200 for an initial capital of 1000. The second best algorithm is the one using SARSA. Even though its final value is approximately the same as the one built with RRL, we can see that the second strategy is much more volatile. Finally the worst algorithm is the one using the Q-learning algorithm, which records a loss...

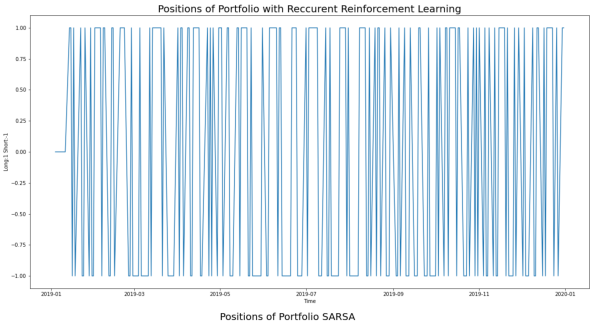
This differences are pretty much expected since the first algorithm (RRL) is much more complex and uses a neural network to predict the hidden states of the financial market.

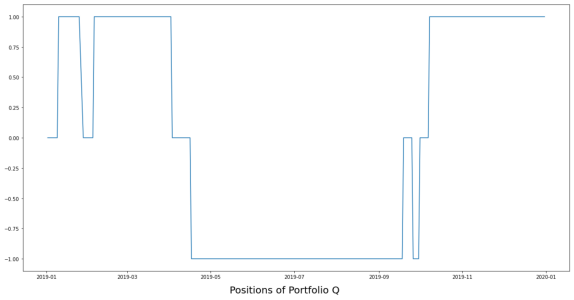


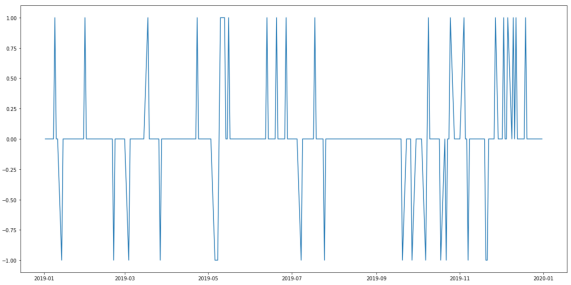




The following graphics show us how the algorithms hold the positions in the portfolio. We can see that the two last algorithms do not take a lot of actions, in comparision with the first one (RRL). This might be because the algorithms were not trained on enough data, and therefore it has not seen enough states during the training. Besides, the parameter ϵ in the ϵ-greedy is not high enough and thus it does not go enough outside of its confort-zone during the training ($\epsilon = 10\%$), and after during the testing ($\epsilon = 1\%$) (hence, holds its position, and does not explore other horizons).







5 Conclusion/Discussion

To conclude, our results show that it is not viable to use model-free reinforcement learning algorithms for automated trading trading purpose. Our models did not beat the market, and have even shown negative performance (Q-learning) in a bullish market. However, the returns are still positive and are close to the benchmark's, especially for the RRL. This is a positive start, which gives us hope to improve the model and do better. The possible improvements for the SARSA and Q-learning models are the following:

- Find a more optimal reward function, in our case we have used the daily returns. However, alternative rewards like sharp ratio, calmar ratio have other properties and the effect of different reward functions should also be further investigated.

- Try to find clusters of states in order to be able to cover more states and thus to update more often the Q-value of the actions. This is a method to reduce the dimension of the Q table.

- Dynamically modify the hyper-parameters α, γ and ϵ over time

- Find better features to define the state. It corresponds to the definition of states. Rather than using the last position and return like what we did in this project we may also use other financial indicators as lagged prices and financial ratios, and macro-economic indicators.

- There are also other extensions to the reinforcement learning model. For example, instead of learning and on values, we may apply policy based learning and Actor critic network which proved to be more efficient thatn value based reinforcement learning method.

The possible improvements for the RLL model are the following:

- Add more features to the FTS, such as macroeconomics and fundamentals variables

- Add layers in the RNN in order to improve its predictions of the states

For now we used reinforcement learning on a single asset and built trading strategies with SARSA, Q-learning and Recurrent Reinforcement Learning. With some simple modifications in the data inputs and changing on some concepts, the problem can be formulate into a larger framework of portfolio allocation, the reinforcement learning methods can also be used to trade multi-assets or to construct a multi-asset portfolio.

References

Corazza Bertoluzzo. 2007. Making financial trading by recurrent reinforcement learning. *Knowledge-Based Intelligent Information and Engineering Systems: 11th International Conference, KES 2007, XVII Italian Workshop on Neural Networks, Vietri sul Mare, Italy, September 12-14, 2007. Proceedings, Part II*.

Marco Corazza and Francesco Bertoluzzo. 2014. Q-learning-based financial trading systems with applications. *SSRN Electronic Journal*.

Gao He Chen Deng He Li, Li. 2015. Recurrent reinforcement learning: A hybrid approach. *arXiv preprint arXiv:1509.03044v2*.