Application of reinforcement learning in quantitative assets trading

Alphonse EBROTIE*

6 février 2024

Introduction

Selon l'hypothèse d'efficience des marchés, il n'est pas possible de faire du profit systématiquement sur les marchés financiers. En effet, cette théorie stipule que les agents économiques sont rationnels et que le marché s'autorégule en termes d'offre et de demande et il n'est pas possible de faire de l'arbitrage (à partir d'un investissement nul, réaliser un gain avec une probabilité non nulle). Toutefois, en pratique, il y a des inefficiences de marché qu'il est possible d'exploiter, à condition de les identifier de façon efficace et être capable d'exécuter des stratégies de trading afin d'en profiter.

Avec les évolutions technologiques récentes , quasiment toutes les firmes de trading quantitatifs ont connus une véritable mutation dans leurs méthodes et techniques . En matière de stratégies , les méthodes basées sur le machine learning sont de plus en plus explorées et donnent des résultats intéressants [1] . C'est dans le contexte de ces techniques que s'inscrit ce projet . L'objectif est de présenter les aspects théoriques des algorithmes de Q-learning , State-Action-Reward-State (SARSA) et *recurrent reinforcement learning* puis d'explorer une application au trading d'actifs financiers .

*alphonseebrotie@gmail.com

Table des matières

1 Aspects théoriques

1.1 Concepts de base de l'apprentissage par renforcement (RL)

L'idée du RL consiste en un *agent* qui obtient des récompenses à partir d'une séquence d'*actions* à partir d'un état initial dans un certain *environnement*.

L'agent utilise ensuite ces récompenses et ces observations pour apprendre une politique optimale afin de maximiser ses gains dans cet environnement. Le RL s'intéresse essentiellement à la modélisation de la distribution parfois complexe entre la fonction de récompense et les différentes paires action-état.

Ci-dessous résumé le processus en apprentissage par renforcement :

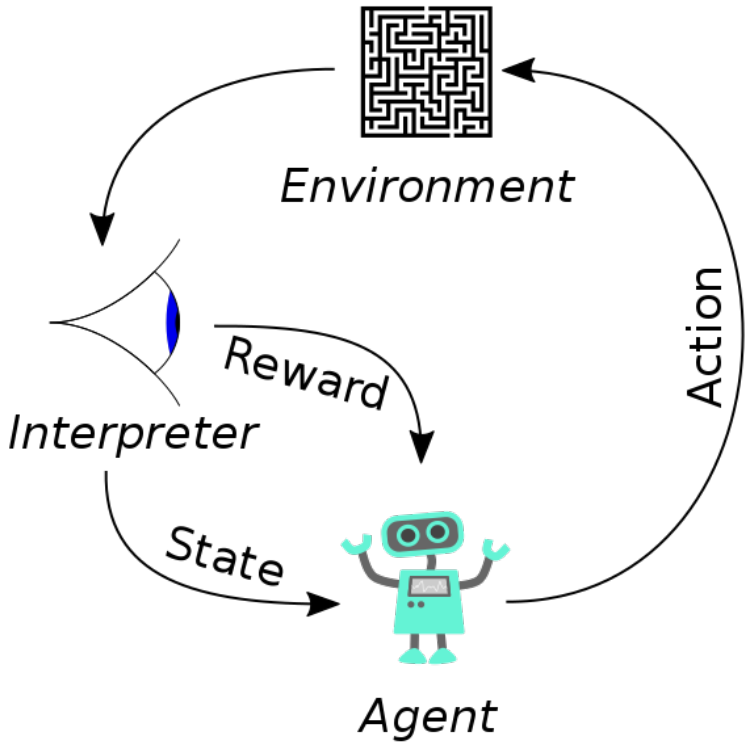

FIGURE 1 – Schéma synthétique résumant l'apprentissage par renforcement , source : en.m.wikipedia.org .

Il y a de nombreux concepts, toutefois les plus importantes et usuelles sont ceux qui seront présentés dans la suite .

Les actions (a)

Les actions correspondent aux décisions de l'agent . À l'instant t, avant de mener une quelconque action , l'agent dispose de t observations (x_1,...,x_t) et des actions passées (a_1,...,a_{t-1}) à partir desquelles il ajuste une politique.

Les états (s)

L'état à chaque pas de temps est caractérisé par les observations et actions passées ainsi que de leur influence . En fait , les états peuvent changer en réponse aux actions de l'agent. Les états fournissent des informations cruciales à l'agent afin qu'il puisse apprendre et s'adapter.
Pour l'instant t , le vecteur des états s'écrit : $s_t = (x_1,a_1,x_2,a_2,...,x_{t-1},a_{t-1},x_t)$.

L'environnement (E)

L'environnement est le système dans lequel l'agent apprend à naviguer .Il correspond à un univers de décisions Markoviennes définit entre l'espace des états $\mathcal{S}$ et celui des actions $\mathcal{A}$ avec comme distribution initiale des états $\mathcal{P}(S_1)$ et la dynamique de transition donnée par : $\mathcal{P}(S_{t+1}|s_t,a_{t-1})$.

Dans notre contexte par exemple , l'environnement contient les informations liées aux investissements (les positions, les prix de marchés, etc.) . Comme entrée , l'environnement prend les différentes allocations d'actifs conformément à notre politique d'investissement et nous retourne la récompense et le prochain état.

La récompense (r)

La récompense correspond au*feed-back* de l'action de l'agent et de l'état dans notre environnement .

L'environnement fournit des récompenses à l'agent en fonction de ses actions. Ces récompenses sont fondamentales dans l'apprentissage par renforcement, car elles permettent à l'agent d'évaluer l'efficacité de ses actions et de s'orienter vers des comportements plus avantageux.

Dans notre contexte , il s'agit du rendement de notre portefeuille. Le rendement futur R_t d'une séquence d'actions et d'états est donné par :

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

où $\gamma \in [0,1]$ est le facteur d'actualisation (*discount factor*) des récompenses futures.

La politique (π)

Une politique, généralement notée π, est une règle ou une stratégie utilisée par l'agent pour décider de l'action à prendre dans un état donné.

$$\pi : \mathcal{S} \rightarrow \ P(\mathcal{A})$$

On distingue généralement deux types de politiques :

- **Politique déterministe :** Dans ce cas, pour un état donné, la politique prescrit toujours la même action. La politique est une fonction directe de l'état vers l'action.

- **Politique stochastique :** Ici, la politique assigne une probabilité à chaque action possible dans un état donné. Cela signifie que pour un état donné, l'action est choisie en fonction d'une distribution de probabilités.

La fonction valeur

C'est une fonction qui attribue une value à chaque état proportionnellement aux récompenses possible pour cet état , à un instant t . En d'autres termes , elle mesure à quel point il est avantageux pour l'agent d'être dans un état donné .

En particulier , la valeur de l'état $s_t = s$ suivant la politique $\pi(.)$ est donnée par la fonction valeur : $V^{\pi}(s) = \mathbb{E}(R^{\pi}(s_t)|s_t = s)$.

De même , la valeur de prendre l'action $a_t = a$ en étant dans l'état $s_t = s$ selon la politique $\pi(.)$ est donnée par la fonction valeur :

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

avec la propriété fondamentale satisfaite par la fonction valeur :

$$V^{\pi}(s) = \mathbb{E}[r(s_t,\pi(s_t),s_{t+1}) + \gamma V^{\pi}(s_{t+1})|s_t = s](1)$$

L'équation (1) correspond à l'équation de Bellman pour $V^{\pi}(s_t)$ (il existe dans la littérature scientifique des manières de prouver que $V^{\pi}(s)$ est l'unique solution de (1) au sens de Bellman.
De même , il y a une équivalence pour $\mathcal{Q}^{\pi}(s_t,a_t)$.
On a également les résultats suivants :

$$V^{*}(s) = \max_{\pi} V^{\pi}(s), Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

,

$$V^{*}(s) = \max_{\pi} Q^{*}(s,a) = \mathbb{E}[R^{*}(s_t)|s_t = s,a_t = a](2)$$

L'équation (2) traduit le fait que la valeur d'un état dans le cadre d'une politique optimale doit correspondre à la récompense globale attendue pour la meilleure action à partir de cet état.

Il est possible d'estimer de manière itérative la valeur optimale de la fonction valeur pour un état étant donné une politique :

- Soit $V_0^{\pi}(s_t)$, l'état initial choisi arbitrairement ;
- Pour tout $V_k^{\pi}(s_t)$, avec k=0,1,.. N ($\in \mathbb{N}$) , on peut approcher la valeur en k+1 par :

$$\hat{V}_{k+1}^{\pi}(s_t) = \mathbb{E}[r(s_t,a_t,s_{t+1}) + \gamma \hat{V}_k^{\pi}(s_{t+1})]$$

, avec $\hat{V}_k^{\pi}(s_t)$ désignant l'estimation de $V_k^{\pi}(s_t)$.

Si l'espérance $\hat{V}^{\pi}_{k+1}(s_t)$ existe , alors on a : $\lim_{k\to\infty} \hat{V}^{\pi}_{k}(s) = V^{\pi}(s)$

Cette manière de calculer permet de trouver des politique de mieux mieux éfficace en mettant en place des techniques d'amélioration de stratégies . Une importante classe de ces techniques concerne à améliorer la politique à chaque itération selon un critère .

De manière général , le processus itératif pour estimer les incréments s'écrit :

$$\hat{V}^{\pi_{new}}_{k+1}(s_t) = \max_{a} \mathbb{E}[r(s_t,a,s_{t+1}) + \gamma \hat{V}^{\pi}_{k}(s_{t+1})] \quad (3)$$

$\hat{V}^{\pi_{new}}_{k+1}(s_t)$ correspond à la fonction de valeur mise à jour en tenant compte de la nouvelle politique d'amélioration déterminée à l'étape k+1 , tenant compte de l'estimation précedente de la fonction de valeur selon la politique adoptée à l'étape k .

Model-free & Model-based

Le model-based a pour objectif de simuler la dynamique de l'environnement en apprenant la probabilité de transition $\mathbb{P}(s_1|s_0,a)$. Si cette probabilité est apprise avec succès , l'agent saura avec quelle probabilité il pourra entrer dans un état spécifique en fonction de l'état actuel et de l'action .

Alors que le model-free s'appuyent sur le concept d' essai-erreur (*trial and error*) pour la mise à jour de leur connaissance .Les algorithmes que nous allons explorer sont de cette catégorie.

On-policy & Off-policy

Un agent est dit "on-policy" si son action courante dérive de la politique actuelle . En fait, l'agent apprend la valeur d'une politique tout en suivant cette même politique. Autrement dit, l'agent apprend sur la base de l'expérience acquise à partir des actions qu'il choisit selon sa politique actuelle. L'algorithme SARSA est un exemple classique d'apprentissage on-policy. Ici, l'agent met à jour ses estimations de la politique en se basant sur les actions effectuées et les récompenses obtenues en suivant cette politique. Alors le "off-policy" fait référence à un agent dont la politique prend en compte d'autres stratégies.

L'agent "off-policy" apprend la valeur d'une politique (la politique cible) indépendamment de la politique qu'il suit actuellement (la politique de comportement). Cela permet à l'agent d'apprendre à partir d'une politique tout en explorant l'environnement en utilisant une autre politique. L'algorithme Q-learning est un exemple typique d'apprentissage off-policy. Dans le Q-learning, l'agent peut suivre une politique exploratoire (comme epsilon-greedy) pour interagir avec l'environnement, mais il apprend une politique différente qui est dérivée de la maximisation de la fonction de valeur Q.

Policy-base & value-based

Les méthodes basées sur la politique apprennent directement la politique optimale. Pour rappel , ine politique est une fonction qui mappe les états de l'environnement aux actions que l'agent doit prendre. On construit explicitement une politique qui est gardée en mémoire tout au long

de l'entrainement. Elles fonctionnent bien dans des environnements avec des espaces d'action continus ou de grande taille. Comme exemple, on a : l'algorithme *Policy Gradient* et *Actor-Critic*

Les méthodes basées sur la valeur apprennent la valeur des actions ou des états, et la politique est dérivée indirectement de ces valeurs. Elles sont souvent plus efficaces dans des environnements avec des espaces d'action discrets et de taille limitée. Comme exemple : le Q-learning, le *Deep Q-Networks (DQN)* .

1.2 Le Q-learning

Le Q-learning est un algorithme off-policy et de type model-free dans lequel il n'y a pas besoin d'initialiser l'environnement.

Dans cet algorithme , deux différentes politiques sont utilisés afin de trouver la politique la plus optimale : la prmeière est utilisée pour calculer l'espérance de la fonction valeur et la seconde afin d'améliorer la stratégie.

Comme sus-mentionné, l'environnement est modélisé par un processus de décision de Markov mais l'agent ne le connaît pas et il n'est pas utilisé dans l'algorithme. L'algorithme est basé sur l'équation de Bellman.

En reprenant les notations introduites plus haut , on re-écrit (3) , l'estimation de la fonction valeur pour chaque état $\hat{V}_{k+1}(s_t)$ de cette façon :

$$\begin{aligned}\hat{V}_{k+1}(s_t) = \alpha \sum_{j=1}^{k+1} R_j(s_t) = \alpha \left[R_{k+1}(s_t) + \sum_{j=1}^{k} R_j(s_t)\right] = \cdots = \\ = \hat{V}_k(s_t) + \alpha \left[R_{k+1}(s_t) - \hat{V}_k(s_t)\right],\end{aligned} \tag{4}$$

Avec $\alpha \in [0,1]$ est appelé le *learning rate*.

L'algorithme de Q-learning est capable de mettre à jour l'estimation $\hat{V}_{k+1}(s_t)$ dès lors que la quantité :

$$d_k = R_{k+1}(s_t) - \hat{V}_k(s_t) = r(s_t,s_{t+1}) + \gamma \hat{V}_k(s_{t+1}) - \hat{V}_k(s_t),$$

est disponible .

Ainsi (4) s'écrit à nouveau :

$$\tilde{V}_{k+1}(s_t) = \tilde{V}_k(s_t) + \alpha \left[r(s_t,s_{t+1}) + \gamma \tilde{V}_k(s_{t+1}) - \tilde{V}_k(s_t)\right].$$

Notons que α représente "le pourcentage" de d_k, c'est-à-dire de l'erreur à ajouter à l'estimation de la fonction de valeur de l'état à la k-ième étape pour obtenir l'estimation de la valeur de l'état au (k+1)-ème pas.

De manière analogue , on a pour la fonction valeur état-action :

$$\widehat{Q}_{k+1}(s_t,a_t) = \widehat{Q}_k(s_t,a_t) + \alpha \left[r(s_t,a_t,s_{t+1}) + \gamma \max_a \widehat{Q}_k(s_{t+1},a_{t+1}) - \widehat{Q}_k(s_t,a_t)\right].$$

Ci-dessous le pseudo-code qui y est associé :

Algorithm 1 Pseudo code de l'algorithme Q-learning

Initialiser $Q(s, a)$ arbitrairement
repeat(pour chaque épisode)
    Initialiser s
    repeat(pour chaque étape dans l'épisode)
        Choisir a à partir de s en utilisant la politique dérivée de Q (ex. ϵ-greedy)
        Poser l'action a, observer r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    until s arrive en phase terminale
until

1.3 L'algorithme SARSA

L'apprentissage par renforcement basée sur la valeur est un concept fondamental en RL comme les couches fully connected le sont pour le deep learning. Comme sus-mentionné,cela permet à l'agent de savoir pour un état donné , quel action est optimale même si cette seule information ne suffit pas à resoudre des problèmes complexes de manières robuste.

L'algorithme SARSA est un algorithme *on-policy* , *value-based* qui permet de prendre des décisions dans un environnement marqué par un processus markovien.

La différence principale entre les algorithmes SARSA (State-Action-Reward-State-Action) et Q-learning, réside dans la manière dont ils mettent à jour leurs valeurs Q, qui sont des estimations des retours futurs attendus pour une action donnée dans un état donné.

- Comme l'algorithme SARSA est un algorithme *on-policy*, cela signifie qu'il apprend la valeur Q en fonction de l'action effectuée par la politique actuelle. La mise à jour de la valeur Q dans SARSA se fait comme suit :

  $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)]$, où $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ représente l'état actuel, l'action actuelle, la récompense reçue, le nouvel état, et la nouvelle action choisie selon la politique actuelle .En fait , il prend en compte l'action que la politique actuelle suggère de faire dans l'état suivant. c'est de l'acronyme du quintuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ que provient le nom de l'algorithme.

- Le Q-learning, en revanche, est un algorithme *off-policy*. c'est-à-dire qu'il apprend la valeur Q indépendamment de l'action choisie par la politique actuelle, en utilisant la meilleure valeur Q disponible pour l'état suivant. Sa formule de mise à jour comme susmentionné est : $Q(s_t,a_t) \leftarrow Q(s_t,a_t)+\alpha[r_{t+1}+\gamma \max_a Q(s_{t+1},a)-Q(s_t,a_t)]$. Dans cette formule , $max_a Q(s_{t+1},a)$ représente la meilleure estimation de la valeur Q pour le prochain état, indépendamment de l'action que la politique actuelle pourrait choisir.

L'agent entraîné avec l'algorithme SARSA aura une tendance plus prudente puisqu'il prend en compte la politique qui sera réellement utilisée, tandis que l'agent sous Q-learning sera plus

optimiste, visant toujours la meilleure récompense possible.

L'algorithme SARSA que nous avons implémenté est basé sur Livre de Sutton et Barto . Afin que l'algorithme converge il faut une politique qui permette de générer des actions . On peut soit générer les actions de façon aléatoire ou calculer une action optimale en se basant sur l'action actuelle.

Le fait d'utiliser des actions aléatoires consiste à explorer les effets des actions inconnues, ce qui peut permettre de mettre à jour la Q-value dans une certaine mesure . L'utilisation d'une politique un peu naïve ,*greedypolicy* , consiste en une exploitation qui nous permet déjà de tester si en phase de test , l'algorithme sera éfficace même si la mise à jour de la Q-value est relativement difficile.
Pour tirer partie de la force de ces pratiques , on combine les combine . De cette combinaison résulte la politique appelée dans la littérature $\epsilon - greedypolicy$.ϵ correspond à la probabilité de sélectionner des actions de manière aléatoire .

La politique '$\epsilon - greedy$ est un moyen de gérer le dilemme exploration-exploitation dans l'apprentissage par renforcement :

- L'exploitation : La plupart du temps, l'algorithme choisit l'action qui a la plus haute valeur Q estimée pour l'état courant, ce qui représente l'exploitation des connaissances actuelles pour obtenir la meilleure récompense immédiate.

- L'exploration : Avec une probabilité , une action aléatoire est choisie, indépendamment de la valeur Q. Cela permet à l'algorithme d'explorer de nouvelles actions qui pourraient conduire à de meilleures récompenses à long terme.

En utilisant -greedy, SARSA explore efficacement l'espace des actions tout en exploitant les connaissances acquises, ce qui est crucial pour une bonne performance dans de nombreux environnements d'apprentissage par renforcement.
De plus , sous certaines conditions, comme une politique d'exploration continue et certains paramètres d'apprentissage, SARSA avec $\epsilon - greedy$ est garantie de converger vers une politique optimale.

Ci-dessous le pseudo-code qui y est associé :

Algorithm 2 Pseudo code de l'algorithme SARSA

Initialiser $Q(s, a)$ arbitrairement
repeat(pour chaque épisode)
    Initialiser s
Choisir a à partir de s en utilisant la politique dérivée de Q (ex. ϵ-greedy)
    repeat(pour chaque étape dans l'épisode)
        Poser l'action a, observer r, s'
        Choisir a' à partir de s' en utilisant la politique dérivée de Q (ex. ϵ-greedy)
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$; $a \leftarrow a'$
    until s arrive en phase terminale
until

1.4 Le Recurrent reinforcement learning

Le *recurrent reinforcement learning* tel qu'introduit par [2] est un modèle hybride qui allie un réseau de neurones récurrent (RNNs) qui apprend la répresentation des états pour le *reinforcement learning* .Comme il existe des états cachés dans l'environnement, à savoir les marchés financiers, nous ne pouvons pas déduire la fonction Q facilement. C'est pour cette raison qu'il faut un modèle hybride qui combine un RNN pour apprendre la distribution des états cachés et un Deep Q-Network (DQN) afin d'apprendre la politique sous-jacente.

Brève introduction aux RNNs

Les RNNs sont des types de réseaux de neurones artificiels conçu pour traiter des données séquentielles ou temporelles. Contrairement aux réseaux de neurones classiques (perceptron,etc.), les RNN possèdent des connexions récurrentes qui leur permettent de mémoriser des informations sur des séquences de données.

Les réseaux RNN sont des types de réseaux pour lesquels les inputs incluent les outputs précédents. Par conséquent , les *output* ne sont pas seulement influencées par les poids appliqués aux entrées mais aussi , par des états cachés représentant le contexte basé sur des observations antérieures.

Le RNN peut être utilisé pour apprendre la représentation des états cachés car nous pouvons l'alimenter en données historiques et le réseau apprendra des caractéristiques internes utiles. Le RNNs apprend donc les états cachés à partir de signaux comprenant les prochaines observations et les récompenses.

Deep Q-Network (DQN)

L'algorithme Q-learning apprend la fonction valeur $Q(s,a)$ qui dépend de l'état s et de l'action a .Par conséquent , on peut approximer cette fonction par un *deep neural network* afin d'optimiser cette fonction pour des séquences de portefeuilles. En effet,dans son article [3] , Cybenko a démontré que les réseaux de neurones à une seule couche cachée et avec une fonction d'activation

non linéaire (comme les sigmoïdes) peuvent approximer toute fonction continue à l'intérieur d'une région compacte avec une précision arbitraire, tant que la fonction d'activation soit choisie de manière appropriée.

L'objectif de l'agent est de choisir les actions qui maximisent la fonction valeur.L'architecture d'apprentissage par renforcement profond traite l'allocation d'actifs comme un problème de contrôle contrôle continu de la politique de portefeuille qui tente de maximiser la récompense . Il permet au processus d'apprendre les meilleures actions possibles dans l'environnement virtuel afin d'atteindre leurs objectifs. Il unifie donc l'approximation de la fonction et l'optimisation de l'objectif. La formule de mise à jour de la valeur Q est la suivante :

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

Où α correspond au learning rate et a' , l'une des actions permettant de maximiser la fonction valeur pour l'état suivant.

De plus amples explications sont disponibles sur ces pages : A Beginner's Guide to Deep Reinforcement Learning and et DQN Deep Q-Network .

Par conséquent, le DQN utilise ici les états cachés appris par le RNN en tant qu'entrées afin d'apprendre la politique optimale.

De plus,l'entrainement des deux réseaux de neurones n'est pas dissocié . De façon pratique , voici comment cela se fait :

A chaque *epoch* de l'algorithme :

- On entraîne un RNNs qui apprend les états cachés ;
- Les états cachés appris deviennent l'entrée pour le DQN, qui apprend la fonction Q .

De manière sommaire , voici comment se présente le process :

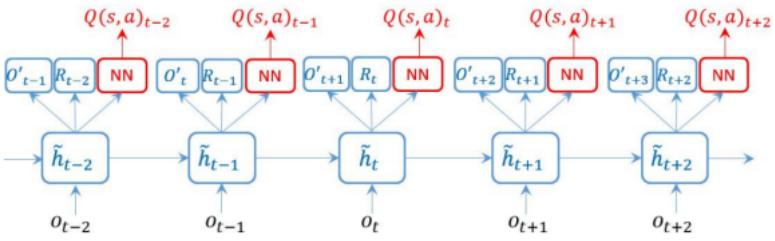

FIGURE 2 – résumé de la méthodologie

Avec ,

- Le RNN est en bleu et le DQN en rouge ;
- o_t correspond à l'observation ;
- $\tilde{h}_t$ est l'état caché issu du RNN
- o'_{t+1} correspond à l'observation prédite en t+1 ;
- R_t est la récompense prédite ;
- $Q(s,a)_t$ est la valeur de Q prédite à l'instant t .

Une autre approche qui n'utilise pas directement les RNNs est exposée par [4] , c'est celle là qui sera explorée pour nos implémentations.

2 Implémentation

Ce chapitre vise à exposer la mise en place concrète des algorithmes introduits ci-dessus dans le cadre de notre problème d'*asset trading* .

Données

Les données utilisées pour construire les signaux sont les prix quotidiens des indices d'actifs . Nous nous sommes intéressés aux 4 principaux indices de marchés à savoir : le *CAC 40*, le *Dow Jones Industrial Average*, le *S&P 500* et le *NASDAQ Composite*. Les données s'étendent du 1^{er} Janvier 1990 au 30 décembre 2023 .

2.1 Implémentation des algorithmes Q-learning et SARSA

La fonction récompense (*reward*) utilisée dans nos algorithmes est le rendement du portefeuille.

2.1.1 Contextualisation du problème

Nous implémentons deux agents (Q-learning et SARSA) qui sont chargés de faire du trading optimal sur un seul actif .

Les agents doivent générer des rendements totaux aussi proches que possible d'une limite théorique déterminée par la performance de l'actif sous-jacent.

De manière général , cela signifie que l'agent génère des rendements maximum sur un horizon quasi-infini (dans le sens où nos agents de trading pourraient continuer à trader indéfiniment et donc indéfiniment un retour sur un investissement initial) ; en phase de test pour l'agent , il est question de générer des rendements proches des rendements historiques , en phase d'entrainement.

Dans le papier original , les auteurs ont pris en compte le rendement des actifs dans les états. Toutefois, cela conduit à des tables Q de grandes dimensions . Pour contourner ce problème , certains chercheurs optent pour un *clustering* des rendements plutôt que les vrais rendements afin de réduire la taille de la matrice Q , pour notre part , nous allons considérer des rendements arrondis.

Pour notre mise en œuvre, nous avons défini le problème comme le choix de l'action qui maximise le rendement pour chaque jour de trading et dans un état particulier parmi toutes les paires état-action correspondantes, où l'ensemble d'actions est un élément de ['buy', 'sell' , 'hold'].

Afin de choisir , l'action qui maximise le rendement , il est intuitif d'anticiper les retours pour l'ensemble des différentes actions pour des états donnés .

Cela signifie que notre problème consiste à la fois à apprendre les valeurs état-action (observation) et à sélectionner des actions (action) en conséquence.

En particulier, étant donné une estimation initiale de la valeur d'une paire d'états-actions, il faut mettre à jour l'estimation de la valeur d'une paire à l'aide d'un échantillon de transitions et de récompenses afin de déterminer le rendement qui maxime l'action.

Ainsi, l'objectif de nos agents de trading est d'apprendre une politique , *online* , qui permet de maximiser leur rendement.

Nous faisons l'hypothèse de générer des États "successeurs". Notre environnement excluant les couts de transactions , l'état actuel ne peut être en relation avec l'état instant qu'au travers des positions issus des actions de notre agent .En fait, cela veut dire que si l'on sait que l'agent vend à l'instant actuel , sa position future à l'instant suivant sera forcément 0 .

Toutefois, étant donné que les décisions d'achat et de vente de notre agent n'ont pas d'incidence sur la variation ultérieure en pourcentage du prix de l'actif négocié , l'état ultérieur, en tant que *tuple* de la position du portefeuille de notre agent et de la variation en pourcentage du prix de l'actif au jour le jour, n'est pas entièrement spécifié par l'action de notre agent dans l'état actuel.

En guise de solution simple, nous avons donc décidé de laisser nos agents supposé que l'état successeur est égal à la position du portefeuille induit par une action particulière dans l'état présent, et au pourcentage de variation du prix de l'actif au jour le jour projeté dans l'état suivant. l'état actuel, et le pourcentage de variation du prix de l'actif d'un jour à l'autre projeté dans le jour suivant,ainsi on projette une tendance.

2.1.2 Tuning des hyper-paramètres

En termes de paramétrages, pour un portefeuille d'actifs donné, nous choisissons les hyper-paramètres, le couple (α,γ) , où :

- α est le *learning rate* , correspondant au paramètre qui détermine de combien notre algorithme ajuste au fur et à mesure ses estimations de la valeur *state-action* en utilisant l'échantillon le plus récent de transition et de récompence .
- γ est le facteur d'actualisation (*discount factor*) appliqué aux rendements futurs et détermine la valeur que notre agent accorde aux rendements futurs par rapport aux rendements immédiats.

Pour les implémentations , on choisit :

$\alpha \in 0.05 \leq \alpha \leq 0.35$; $\beta \in 0.85 \leq \alpha \leq 1$; ϵ dans [0.01, 0.05, 0.10, 0.15] .

Pour cette tranche de valeurs , on effectue plusieurs combinaisons de valeurs de α et β par pas de 0.05.

On stocke pour chaque paire, les rendements totaux ainsi que les écart-types de chaque paramètre , pour rester cohérent ,dans la mesure où les mêmes trades ne peuvent pas être exécuté plusieurs fois.

Pour finir , on sélectionne les valeurs qui maximisent la différence entre les rendements totaux moyens et l'écart-type des rendements totaux afin d'avoir un compromis entre cohérence et performance. Et pour ϵ , on utilise une valeur plus grande dans la période d'entraînement que dans la période de test afin d'exploiter le plus grand nombre d'états états possibles pendant le train et d'assurer la stabilité des résultats lors du test.

2.2 Implémentation du recurrent reinforcement learning (RRL)

La méthodologie implémentée s'inspire du papier de recherche [4].Ce papier utilise la méthode *reciprocal of the returns weighted direction symmetry index* (RRWDS) plutôt que le Sharpe ratio.

Comme évoqué précédemment, un modèle RRL comprend deux composantes : la composante de supervised learning (ici un réseau de neurones récurrents) qui apprend les états cachés et la composante d'apprentissage par renforcement (ici un réseau Q profond) qui apprend la politique en utilisant cet état caché l'état caché en tant qu'input.L'entrainement se fait de se fait conjointement.

Dans les sous-sections suivantes, nous allons expliquer la mise en œuvre du ANN et du DQN,et comment ils fonctionnent ensemble.

2.2.1 Eléments de contexte

Cette section décrit l'apprentissage d'un agent qui interagit de manière dynamique dans un environnement et y prend des décisions. Dans le language courant , on dit qu'un système financier de trading (l'agent) met en place une stratégie (politique) afin d'émettre des ordres sur le marché financier (environnement).

La méthodologie générale est :

- Identifier les variables d'intérêts (prix des actifs , les volumes , etc .) ;
- Extraire des informations des valeurs passées et actuelles pour prédire leur valeur future ;
- Utiliser les informations et les prédictions pour mettre en œuvre une stratégie.

L'environnement avec lequel l'agent interagit comporte une part d'incertitude , il y a donc des états cachés.

2.2.2 L'état caché du système (y_t)

Dans l'idée , comme on travaille avec des prix d'instruments financiers, il y a des états cachés dans le système . En fait , les prix sont influencés par l'existence de facteurs ou de dynamiques sous-jacents qui ne sont pas directement observables ou facilement mesurables sur le marché. C'est l'une des raisons pour lesquelles un algorithme vanille de reinforcement learning n'est pas adapté, nous introduisons donc un modèle neuronal capable d'apprendre ces états cachés à partir de signaux , en utilisant les données historiques.

Dans le papier , les auteurs ont considéré un perceptron simple sans couche cachée avec la fonction tanh comme fonction d'activation , avec l'objectif que les valeurs soient dans l'intervalle [-1 ;1] de sorte à ce que les poids du réseau converge vers une solution stable :

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

Avec :

- $w_{0,\tau}$,...,$w_{M+1,\tau}$ correspond aux poids du réseau de neurones à l'instant τ
- x_τ,...,$x_{\tau-N}$ correspond aux valeurs présentes et passées du sont les valeurs actuelles et passées du taux de rendement logarithmique de l'indice à l'instant τ.

- $w_{M+2,\tau}$ est le seuil du réseau . En fait , dans un réseau de neurones, le seuil est une valeur qui détermine à quel moment un neurone s'active ou se "déclenche".

2.2.3 La politique

La politique est juste la stratégie de trading à mettre en place.Dans notre cas , c'est déterminé par la fonction :

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

Dans l'aspect code , Cette fonction correspond à la fonction POSITIONS qui prend en inputs x_τ,...,$x_{\tau-N}$ et $w_{0,\tau}$,...,$w_{M+1,\tau}$ puis retourne les valeurs de $y_{0,\tau}$,...,$y_{M+1,\tau}$ et $F_{0,\tau}$,...,$F_{M+1,\tau}$.

Pour une implémentation plus propre , nous utilisons des matrices pour stocké y_t. Ainsi , la formule de y_t devient :

$$y_t = \tanh\left(\theta^T x_t\right)$$

, avec :

- θ représente l'ensemble des paramètres à optimiser ;
- $x_t = [1,r_{t-M},...,r_t,F_{t-1}]$, où r_t est le changement de valeur de l'actif entre t et t-1 , M est le nombre d'inputs de la série temporelle.
-

Cela signifie qu'à chaque pas de temps, le modèle sera alimenté par sa dernière position et une série de prix historiques qu'il peut utiliser pour calculer sa prochaine position.

2.2.4 La récompense

La récompense nette à la période générique (t - 1, t] est définie de cette façon (en accord avec le papier qui nous sert de référence) :

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

Avec :

- μ le montant de capital à investir ;
- r_t est le taux de rendement géométrique au temps τ du portefeuille .
- δ correspond au coût de transaction en pourcentage lié au quota de portefeuille à trader.

La fonction récompense est le rendement du portefeuille pénalisé du cout des transactions . Elle correspond dans l'implémentation à la fonction REWARDS qui prend en inputs à chaque pas de temps les positions (et donc les états cachés) , les rendements , les coûts de transactions et le capital à investir.

2.2.5 L'espérance de récompense totale

La récompense totale espérée est donnée par :

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

,

Où R_i correspond à la récompense à l'instant i . Cette espérance est calculée avec la fonction GRADIENT.

2.2.6 Indice de gain des investisseurs

cet indice est utilisé afin de trouver les poids du système.Etant donné que le sharpe ratio est largemet utilisé dans la littérature , les auteurs ont voulu utilisés une autre métrique comme indice de gain de l'investisseur : "la mesure de la symétrie directionnelle pondérée par la réciproque des rendements". En d'autres termes, il s'agit d'une évaluation qui prend en compte la symétrie des rendements dans différentes directions, tout en accordant plus de poids à certaines valeurs, et en utilisant l'inverse (la réciproque) de ces rendements dans le processus de calcul.

Il s'agit du rapport entre les récompenses positives cumulées et les récompenses négatives cumulées à chaque instant t, donné par la formule suivante :

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

, avec bien entendu $\sum_{i=1}^{t} b_i \mid R_i \mid \neq 0$, avec :

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

I_t est utilisée pour calculer la variation marginale de la fonction d'utilité de l'agent : $D_t = U_t - U_{t-1} = I_t - I_{t-1}$. Comme I_t est d'autant plus grand que t augmente, on utilise une formulation exponentielle définie par :

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

, avec :

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

Dans la partie implémentation , cet index est representé par la fonction RRWDS.

2.2.7 Mise à jour des poids

A chaque itération , il y'a besoin d'une mise à jour des poids du réseau de neurones afin d'affiner ses prédictions . Comme notre réseau de neurones est un perceptron simple (et comme

notre objectif est de maximiser le rendement) , la mise à jour des poids se fait par l'ascension de gradient suivante :

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

, avec :

- $w_{i,t}$ les poids du réseau à l'instant t ;
- ρ_t est le learning rate à l'instant t ;

U_t est la fonction d'utilité de l'investisseur à l'instant t .

Afin de réaliser ce gradient ascendant , il y a besoin de calculer les dérivées de la fonction d'utilité (RRWDS) par rapport à $\theta(\frac{dU_t}{d\theta})$. En utilisant les formules de dérivations en chaîne , on peut écrire :

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

Dans l'implémentation , les gradients sont calculés via la fonction GRADIENT. Ainsi , la fonction retourne $\frac{dU_t}{dw_{i,t}}$, ce qui permet de calculer la valeur de $w_{i,t}$ et par conséquent d'ajuster les poids lors de l'entraînement dans la fonction TRAIN.

2.2.8 Le phénomène de "drawdown"

Le terme "Drawdown" est un terme souvent utilisé dans la littérature financière , en particulier dans la gestion de portefeuilles d'investissement. Il se réfère à la mesure de la baisse d'un investissement ou d'un fonds d'investissement depuis son pic jusqu'à son point le plus bas avant de remonter. Par exemple, si un portefeuille atteint un pic de valeur puis perd une certain pourcentage avant de se redresser, ce pourcentage de perte est le *drawdown*.

L'un des objectifs des auteurs du papier [4] est que le système soit en mesure de minimiser les pertes importantes et d'assurer la continuité du trading lorsque ces pertes surviennent.

C'est pourquoi ils mettent en place une méthode afin de gérer le phénomène du *drawdown*.

Pour se faire , ils utilisent comme capital à investir : $\mu + \mu_0$ où μ_0 est la valeur absolue de la plus grande perte de la partie initiale de la première sous-période temporelle. En d'autres termes :

$$\mu_0 = min(min_{0<t<t_{off}}(R_t : R_t < 0 \cap CR_t < 0 \cap sign(F_t F_{t-1}) = -1), 0)$$

Pour réaliser cette gestion du drawdown, nous articulons l'ensemble de la période de trading (de 0 à T) en une séquence de sous-périodes temporelles qui se chevauchent :

$$[0,\Delta t_{off} + \Delta t_{on}], [t_{on},t_{off} + 2\Delta t_{om}],...,[T - \Delta t_{off} - 2\Delta t_{on},T - \Delta t_{on}],[T - \Delta t_{off} - \Delta t_{on},T]$$

, avec :

- Δt_{off} : La durée de la partie initiale de chaque sous-période de temps pendant laquelle le système travaille en mode hors ligne afin d'effectuer les réglages optimaux des paramètres considérés.

- Δt_{on} : La taille de la partie finale pour chaque t.

2.2.9 Tuning des paramètres

Afin de rendre notre système de trading opérationnel , il nous faut déterminer les valeurs optimaux des différents paramètres :

- M : le nombre de séries temporelles en entrée ;
- δ : le pourcentage du coût de transaction lié au quota de portefeuille à trader ;
- ρ : le *learning rate* ;
- η : un coefficient d'adaptation .

Pour effectuer ce paramétrage optimal, nous considérons les rendements du S &P 500 et nous divisons l'ensemble des données en deux catégories : un ensemble de formation et un ensemble de test. Pour chaque paramètre à optimiser, nous fixons les autres paramètres et nous entraînons le modèle en utilisant 100 époques. Une fois l'entraînement terminé, nous appliquons notre fonction de test et nous conservons les paramètres dont la valeur a atteint le plus grand CR_T.

3 Résultats et Backtesting

3.1 Q-learning et SARSA

Pour montrer les résultats de nos agents, nous étudions les prix quotidiens du S &P 500 (pour les autres indices , on peut les trouver dans le code) pour la période allant de sa création à décembre 2018. Et nous effectuons des tests sur la période du 01/01/2019 - 01/01/2020.Le benchmark est juste la performance réel du portefeuille sur la même période. Les données sont extraites de yahoo finance. Les résultats sont présentés ci-dessous :

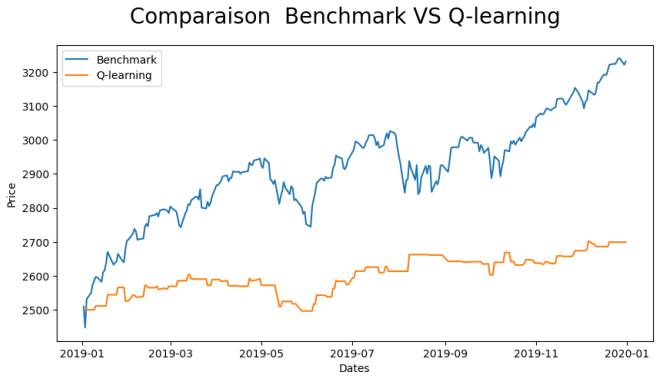

FIGURE 3 – Performance du Q-learning

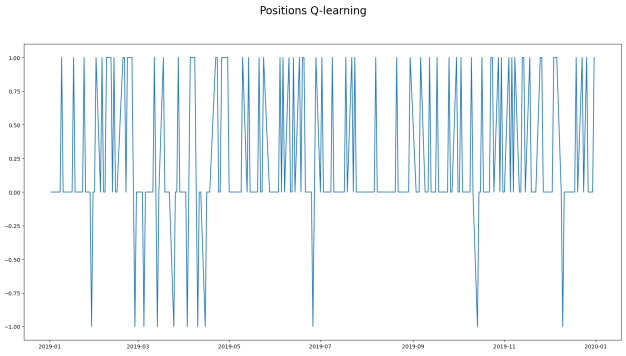

FIGURE 4 – Positions de l'agent Q

On constate que l'algorithme SARSA a une meilleure performance que le Q-learning même si aucun des deux ne fait mieux que le benchmark.Nous remarquons également que le test n'est pas stable car nous ne disposons pas de suffisamment de données pour entraîner notre modèle.

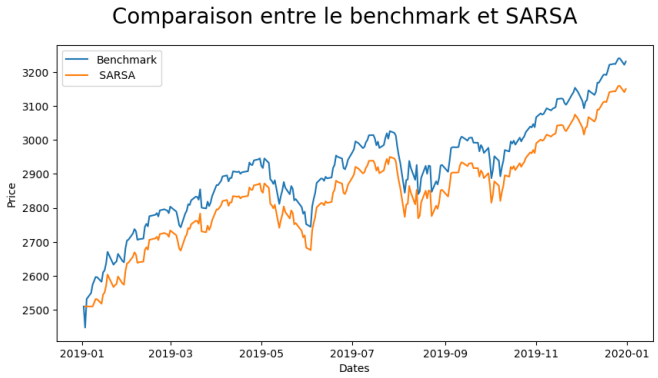

FIGURE 5 – Performance SARSA

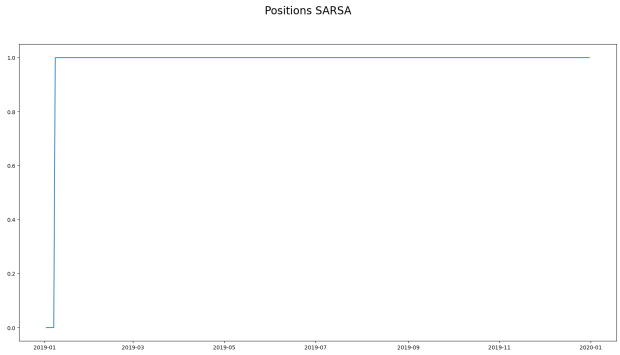

FIGURE 6 – Positions avec SARSA

Indicateur	**Benchmark**	**Q-learning**	**SARSA**
Rendement annualisé	0.261	0.079	0.236
Volatilité annualisée	0.125	0.061	0.117
Sharpe Ratio	2.096	1.295	2.014

TABLE 1 – Comparaisons d'indicateurs

3.2 Recurrent reinforcement learning

Tout d'abord, nous commençons par les hyperparamètres, puis nous avons entraînons et testons le modèle sans oublier de le comparer aux modèles SARSA et Q-learning.

3.2.1 Choix des paramètres

Comme expliqué plus haut , on essaye d'avoir des paramètres optimaux en prenant les paramètres qui maximisent le rendement cumulé du S & P 500 (on peut le faire pour chacun des indices) . On obtient comme paramètres optimaux :

$\delta = 5\ \%$, $M = 7$, $\rho = 0.1$, $\eta = 1$

Les paramètres sont très proches de ceux trouvés dans l'article .

3.2.2 résultats pour le RRL

Dans cette section, nous avons distingué deux cas : une modélisation sans utiliser la gestion des phénomènes de *drawdown* et une autre avec gestion du *drawdown*.

Comme indiqué dans le paragraphe 3.1, les données que nous avons utilisées pour construire nos signaux sont les prix quotidiens des indices d'actifs. Nous avons utilisé les prix de 4 des principaux indices des marchés financiers mondiaux : le CAC 40, le Dow Jones Industrial Average, le SP 500 et le NASDAQ Composite, du 1er janvier 1990 au 16 mai 2020.

On utilise 70 % du dataset en train et 30 % en test , puis le modèle s'ajuste 100 fois (*epoch*).
Ci-dessous les différents résultats pour les différents datasets :

S & P 500

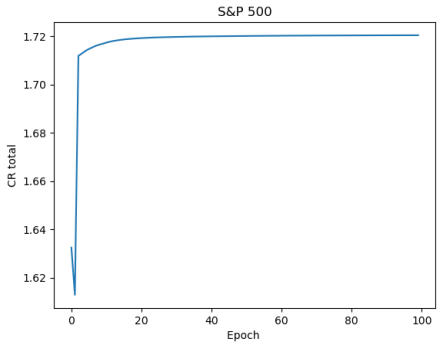

FIGURE 7 – rendement cumulé avant la gestion du drawdown

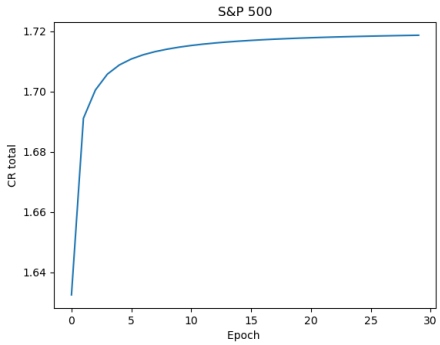

FIGURE 8 – rendement cumulé après la gestion du drawdown

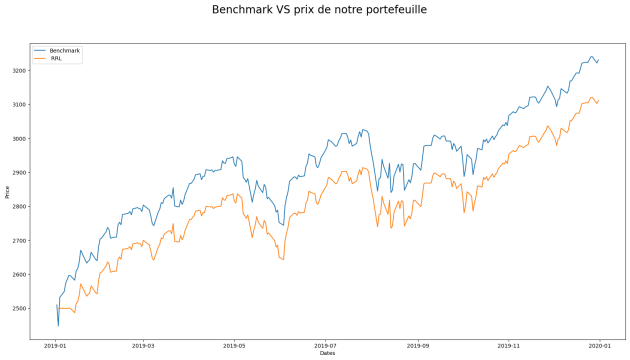

FIGURE 9 – comparaison benchmark et RRL sur le S&P500

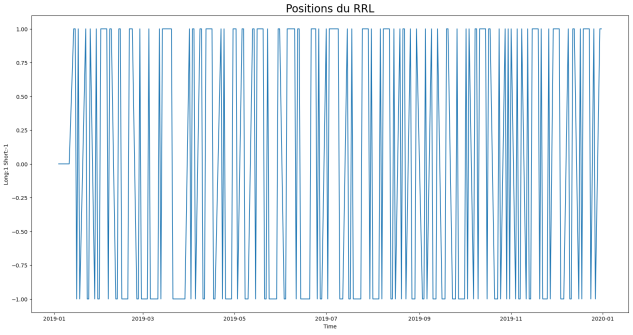

FIGURE 10 – positions du RRL pour le S&P500

Indicateur	**Benchmark**	**RRL**
Rendement annualisé	0.261	0.227
Volatilité annualisée	0.125	0.117
Sharpe Ratio	2.096	1.944

TABLE 2 – Comparaisons d'indicateurs

En comparant la performance de notre algorithme au benchmark , il revient que même si les résultats sont proches , il ne parvient pas à faire mieux , tout comme le Q-learning et le SARSA.

CAC 40

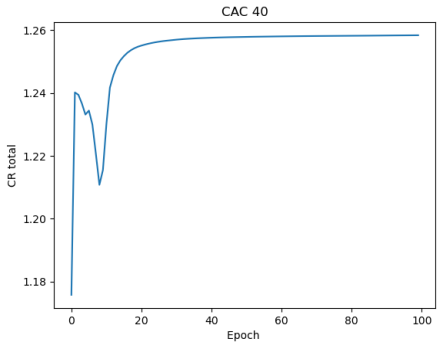

FIGURE 11 – rendement cumulé avant la gestion du drawdown

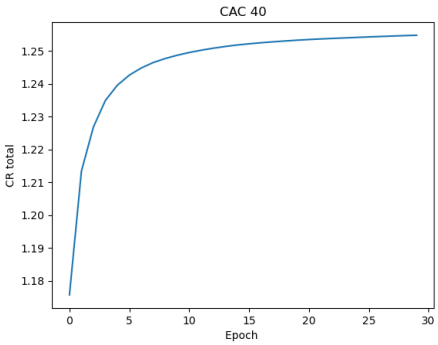

FIGURE 12 – rendement cumulé après la gestion du drawdown

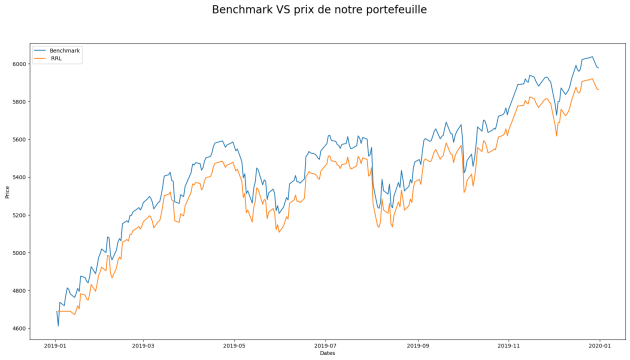

FIGURE 13 – comparaison benchmark et RRL sur le CAC 40

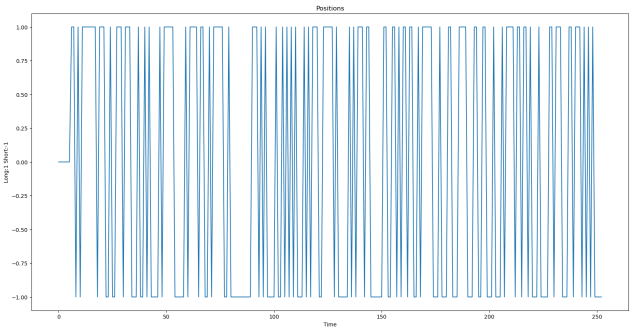

FIGURE 14 – positions du RRL pour le CAC 40

Indicateur	**Benchmark**	**RRL**
Rendement annualisé	0.250	0.231
Volatilité annualisée	0.132	0.128
Sharpe Ratio	1.889	1.807

TABLE 3 – Comparaisons d'indicateurs pour le CAC 40

De la même manière que précédemment on constate que le RRL ne fait pas mieux que le Q-learning et le SARSA , même si les résultats obtenus sont proches du benchmark.

Nasdaq Composite

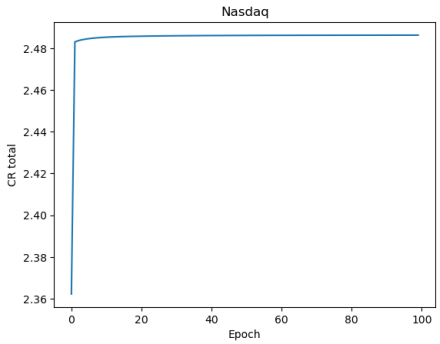

FIGURE 15 – rendement cumulé avant la gestion du drawdown

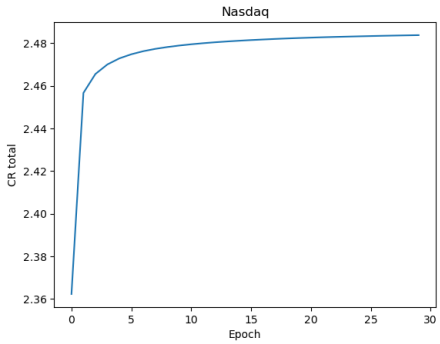

FIGURE 16 – rendement cumulé après la gestion du drawdown

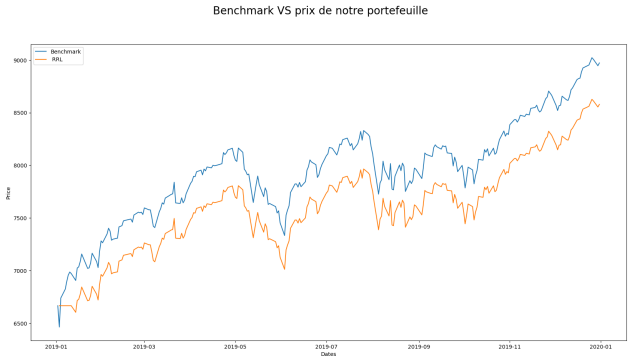

FIGURE 17 – comparaison benchmark et RRL sur le Nasdaq

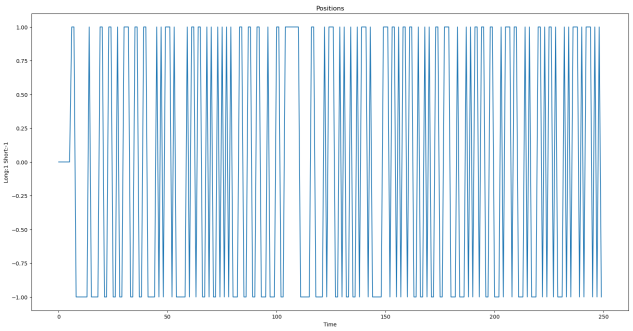

FIGURE 18 – positions du RRL pour le Nasdaq

Indicateur	**Benchmark**	**RRL**
Rendement annualisé	0.311	0.265
Volatilité annualisée	0.157	0.147
Sharpe Ratio	1.985	1.806

TABLE 4 – Comparaisons d'indicateurs pour le Nasdaq composite

Dow Jones

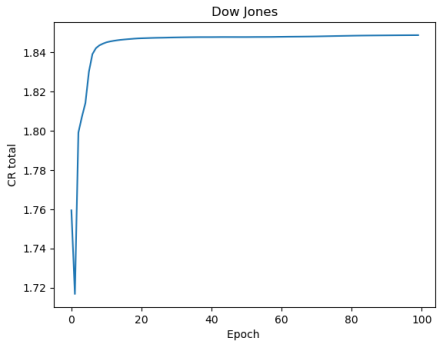

FIGURE 19 – rendement cumulé avant la gestion du drawdown

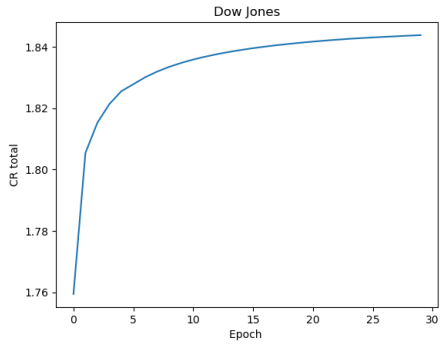

FIGURE 20 – rendement cumulé après la gestion du drawdown

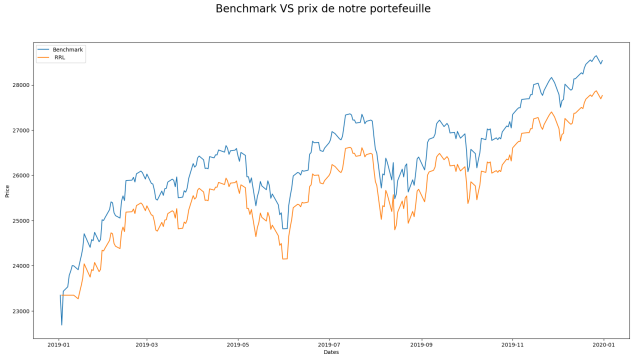

FIGURE 21 – comparaison benchmark et RRL sur le Dow Jones

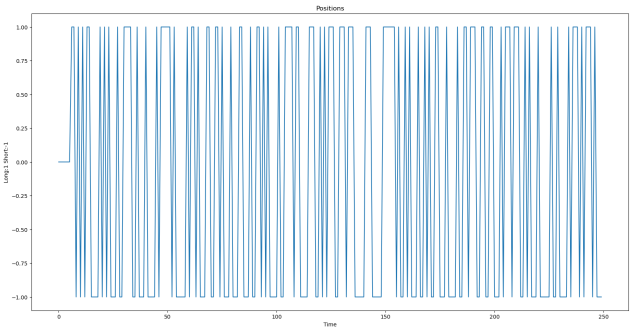

FIGURE 22 – positions du RRL pour le Dow Jones

Indicateur	**Benchmark**	**RRL**
Rendement annualisé	0.209	0.182
Volatilité annualisée	0.124	0.116
Sharpe Ratio	1.684	1.565

TABLE 5 – Comparaisons d'indicateurs pour le Dow Jones

4 Conclusion

L'objectif de ce projet était d'explorer les algorithmes Q-leaning , SARSA et recurrent reinforcement learning basés sur l'article [4].

Les différents résultats montrent qu'il n'est pas viable d'utiliser des algorithmes d'apprentissage par renforcement en mode model-free à des fins de trading automatisé.En fait, il se trouve que les différents modèles n'ont pas surpassé le marché et ont même montré des performances négatives (*Q-learning*) dans un marché haussier. Cependant, les rendements restent positifs et sont proches de ceux du benchmark, surtout pour le RRL. C'est un début positif qui nous donne de l'espoir pour améliorer les modèle .

Les perspectives pour améliorer les modèles SARSA et Q-learning sont les suivantes :

- Trouver une fonction de récompense plus optimale ; dans notre cas, nous avons utilisé les rendements quotidiens. Cependant, d'autres récompenses alternatives telles que le ratio de Sharpe, le ratio de Calmar, ont d'autres propriétés et l'effet de différentes fonctions de récompense devrait également être examiné plus en détail ;
- Introduire une modification dynamique des hyperparamètres $\alpha, \gamma et \epsilon$
- Mise en place de techniques (clusters d'états) de réduction de la dimension de la table Q .

En ce qui concerne le RRL , une utilisation directe de réseaux récurrents (Long Short Term Memory [LSTM] ou Gated Recurrent Unit [GRU]) en lieu pourrait être plus robuste.

Références

[1] S. K. Sahu, A. Mokhade et N. D. Bokde. « An Overview of Machine Learning, Deep Learning, and Reinforcement Learning-Based Techniques in Quantitative Finance : Recent Progress and Challenges ». In : *Applied Sciences* 13.3 (2023), p. 1956.

[2] X. Li et al. « Recurrent reinforcement learning : a hybrid approach ». In : *arXiv preprint arXiv :1509.03044* (2015).

[3] G. Cybenko. « Approximation by superpositions of a sigmoidal function ». In : *Mathematics of control, signals and systems* 2.4 (1989), p. 303-314.

[4] F. Bertoluzzo et M. Corazza. « Making financial trading by recurrent reinforcement learning ». In : *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 11*. Springer. 2007, p. 619-626.