Tahar Adam AMAIRI (amat0601), Victor CHAU (chav0401), Omar CHIDA (chim2708), Céline ZHANG (zhac3201)

Gestion d'un portfolio par un BOT de trading entraîné par Reinforcement learning

PROJET - Planification en intelligence artificielle - IFT 702 - *Supervised by Froduald Kabanza* - Université de Sherbrooke - 08/03/2023

Abstract. Ce projet concerne la réalisation d'algorithmes de *trading*, basés sur des méthodes d'apprentissage par renforcement (RL), ayant pour finalité de réaliser le plus de profit possible dans les placements de fonds réalisés. Les données d'exploitation du projet proviennent des cours boursières du marché. Le but est de programmer un *bot* qui se sert des algorithmes de prédiction pour gérer un *portfolio*. Le projet se décompose en trois étapes, la première est consacrée à l'exploration des données existantes, étudier les métriques d'évaluation du système, à entraîner un modèle de prédiction (DQN) déjà existant et à l'appliquer sur les cours boursiers. La seconde étape se concentrera sur les améliorations du *bot* en essayant d'autres méthodes d'apprentissages (par exemple PG, LSTM, GRU, etc.). La troisième étape consistera à ajouter une fonctionnalité au *bot* (par exemple louer ou *trade* sur plusieurs cours), puis éventuellement agrandir le rayon des courbes à étudier. Le projet se base sur l'article *An application of deep reinforcement learning to algorithmic trading* de T. Théate et D. Ernst dont le code source sont publiés dans leur répertoire GitHub.

Introduction. La particularité de ce projet est que les données d'apprentissage sont publiques, produites en continu et disponibles en ligne. Cependant, un grand nombre de méta-données, d'informations environnementales externes (*news*, médias, etc.) peuvent influencer les cours boursiers. De plus, les stratégies utilisées par les compagnies privées de *trading* impactent directement leurs activités économiques, donc les recherches concernant les méthodes de *trading* ne sont pas publiques. De ce fait, le système du problème de *trading* reste assez complexe à définir. L'algorithme de *trading* étudié se base sur un modèle entraîné par DQN adapté au marché boursier, il est capable de gérer un *portfolio* à partir des actions de base (acheter, vendre et conserver). La difficulté des actions de l'agent repose sur quel *stock* je réalise mon action, quand est-ce que je le fais, comment je le réalise, à quel prix et pour quelle quantité ? Pour simplifier le problème, un *stock* est géré à la fois par l'agent, une action est réalisée à chaque instant, à partir de cette base, le contexte de *trading* sera défini, une présentation des résultats existants et des applications suivront, puis enfin les améliorations seront abordées.

L'état de l'art. Le *trading* algorithmique est un domaine qui a attiré une attention croissante ces dernières années. Les avancées dans l'apprentissage par renforcement profond ont permis de développer des algorithmes de trading plus efficaces. L'objectif de cet article est d'étudier l'application de l'apprentissage par renforcement profond au problème du trading algorithmique, en proposant un nouvel algorithme de *deep reinforcement learning* appelé *Trading Deep Q-Network* (TDQN), inspiré de l'algorithme DQN.

Le *trading* peut être considéré comme la gestion d'un *portfolio* d'actifs, qui peut inclure des actions, des obligations, des matières premières, des devises, etc. Dans cette étude, le *portfolio* considéré ne contient qu'une seule action et de l'argent en liquide. La valeur du *portfolio* évolue en fonction de la valeur de l'action de de l'argent en liquide détenue par l'agent de trading. Les opérations d'achat et de vente consistent simplement en des échanges d'argent et d'actions. L'agent de *trading* interagit avec le marché boursier à travers un carnet d'ordres, qui contient l'ensemble des ordres d'achat et de vente. Un ordre représente la volonté d'un participant du marché de trader et est composé d'un prix, d'une quantité et d'un côté (achat ou vente). Pour qu'un échange ait lieu, une correspondance entre les ordres d'achat et de vente est requise, un événement qui ne peut se produire que si le prix d'achat est supérieur ou égal au prix de vente. Ensuite, l'agent de *trading* doit décider quoi, quand, comment, à quel prix et quelle quantité trader afin de générer un profit, ce qui représente le problème complexe de prise de décision séquentielle étudié dans cette recherche.

Le *trading* algorithmique est formalisé en un problème de *reinforcement learning* en considérant le marché financier comme environnement dans lequel l'agent doit apprendre à prendre des décisions en fonction des informations disponibles afin de maximiser une récompense. La prise de décision séquentielle est un problème complexe étudié dans cet article. Les décisions de *trading* sont basées sur une stratégie, qui peut être considérée comme une politique programmée qui prend en compte les informations du marché et donne une action de *trading* en retour. Cette stratégie est appliquée de manière séquentielle à chaque pas de temps, l'agent se base sur des données temporelles des *stocks* qu'il divise en périodes Δt pour l'entraînement. Le nombre de jours, c'est-à-dire l'information journalière du marché, pris en compte dans l'ensemble des états peut être ajusté par l'utilisateur.

L'exécution de l'algorithme consiste en quatre étapes : mise à jour des informations de marché, exécution de la stratégie de *trading* pour obtenir l'action de *trading*, application de cette action, et passage à l'instant suivant. Le but est de maximiser une fonction de récompense qui dépend des décisions de *trading* prises par l'agent. La décision de l'algorithme est prise à partir des observations et des récompenses selon les états dans l'environnement. L'observation de l'environnement est composée de plusieurs éléments : les informations d'état de l'agent RL, l'information recueillie par l'agent concernant les données OHLCV caractérisant le marché boursier, l'information de l'agent concernant l'instant de trading (date, jour, heure), l'information de l'agent concernant plusieurs indicateurs techniques sur le marché boursier, les informations macroéconomiques dont dispose l'agent, les informations d'actualité recueillies par l'agent, et toute information supplémentaire utile à la disposition de l'agent trader.

Concernant l'objectif principal d'une stratégie de *trading*, bien que la génération de profits soit un objectif important, il est insuffisant pour évaluer la performance globale d'une stratégie. L'objectif est de maximiser le ratio de *Sharpe*, un indicateur de performance largement utilisé dans les domaines de la finance et du *trading* algorithmique. Le ratio de *Sharpe* est bien adapté à la tâche d'évaluation des performances car il considère à la fois le profit généré et le risque associé à l'activité de *trading*.

Le TDQN est une nouvelle approche de DRL pour le *trading* algorithmique. Il diffère du DQN classique par la manière dont la récompense (ou le signal de retour) est définie. Dans le TDQN, la récompense est définie comme le rendement quotidien, qui est calculé en fonction des prix des actions du marché. En revanche, dans le DQN classique, la récompense est généralement définie

comme une fonction de la différence entre la fonction de valeur état-action pour l'état actuel et le suivant. En outre, le TDQN utilise une architecture de réseau de neurones différente de celle du DQN classique. Il utilise un réseau de neurones composé de plusieurs couches convolutionnelles et de couches entièrement connectées pour modéliser les prix des actions et les tendances du marché.

Les auteurs soulignent de l'importance d'une méthodologie précise d'évaluation des performances dans le domaine du *trading* algorithmique et présentent une nouvelle méthodologie fiable pour évaluer les performances des stratégies de *trading* algorithmique. Ils expliquent que les résultats obtenus avec une seule action peuvent ne pas être représentatifs de la performance globale de l'algorithme. Ainsi, ils présentent une analyse des performances de TDQN sur un *portfolio* d'actions, montrant que l'algorithme est capable de s'adapter à des environnements complexes et volatiles. La méthodologie comprend un *testbench* composé de 30 actions présentant diverses caractéristiques (secteurs, régions, volatilité, liquidité). L'horizon de test est de huit ans, divisé en ensembles d'entraînement et de test. Des stratégies de trading de référence, notamment *Buy and Hold*, *Sell and Hold*, *Trend Following* et *Mean Reversion*, sont sélectionnées à des fins de comparaison. Ces stratégies sont divisées en stratégies passives et actives, avec un risque plus élevé pour les stratégies actives. Les performances sont évaluées sur plusieurs appareils afin d'éliminer les biais, et les hyperparamètres de l'algorithme restent inchangés sur l'ensemble du banc d'essai.

Les métriques caractéristiques du *portfolio* sont les valeurs de la liquidité (*cash*) et la valeur du capital global des actions (*shares*). L'algorithme prend en considération les coûts de transactions. Les stratégies possibles de l'algorithme RL sont le *long trading* : transforme les liquidités en actions (achat d'actions) et le *short trading* : transforme les actions en liquidités (vente d'actions). La rentabilité du *portfolio* est évaluée grâce au *ratio de Sharpe*, qui est une mesure de performance financière qui permet d'évaluer le rendement ajusté au risque d'un investissement ou d'un *portfolio* d'investissement.

L'article présente une analyse détaillée des performances de l'algorithme TDQN sur l'action Apple, qui a donné de bons résultats, puis sur l'action Tesla, qui présente des caractéristiques différentes de celles de l'action Apple, telles qu'une forte volatilité (dispersion de ses rendements par rapport à une moyenne sur une période de temps donnée). L'analyse montre que l'algorithme TDQN a obtenu des résultats mitigés sur l'action Tesla, ce qui met en évidence les limites de l'algorithme. La section aborde également le paramètre du facteur d'actualisation, l'influence des coûts de trading et les principaux défis rencontrés par l'algorithme TDQN.

Finalement, l'algorithme TDQN obtient des résultats surpassant en moyenne les stratégies de trading de référence. TDQN présente de nombreux avantages par rapport aux approches plus classiques, telles que la polyvalence et la résistance aux différents coûts de trading.

Méthodologie. Pour adapter le modèle au problème de *trading*, plusieurs modifications et améliorations ont été introduites par les autres pour réaliser leur TDQN sur la base de DQN :

- **Réseau de neurones profonds (DNN)** : La différence clé entre l'algorithme DQN classique et l'algorithme TDQN se trouve dans l'architecture du **DNN** utilisé pour estimer la fonction d'action-valeur. Étant donné que l'entrée est une série temporelle et non des images brutes, le réseau de neurones convolutionnel (CNN) a été remplacé par un DNN avec des fonctions d'activation LReLU.
- **Double DQN** : Le problème de l'algorithme DQN est qu'il surestime fortement les valeurs, ce qui nuit à ses performances. L'article de van Hasselt et al. (2015) propose une solution appelée **Double DQN**, qui consiste à décomposer l'opération max de la cible en deux parties distinctes : la sélection de l'action et l'évaluation de l'action. Cette technique permet de réduire l'impact de la surévaluation des valeurs et d'améliorer les performances de l'algorithme.
- **ADAM *Optimizer*** : Le DQN classique utilise l'optimiseur RMSProp. Toutefois, l'**optimiseur ADAM**, introduit par Kingma et Ba (2015), s'est avéré améliorer à la fois la stabilité de l'entraînement et la vitesse de convergence de l'algorithme DRL.
- **Huber *Loss*** : Le DQN classique utilise une perte quadratique moyenne (MSE), mais la perte de Huber a été expérimentalement démontrée comme améliorant la stabilité de la phase d'entraînement. La perte MSE pénalise fortement les grandes erreurs, ce qui est généralement souhaité, mais a un effet négatif pour le DQN car le DNN doit prédire des valeurs qui dépendent de ses propres entrées. Idéalement, la mise à jour du DNN devrait être effectuée de manière plus lente et plus stable. En revanche, l'erreur moyenne absolue (MAE) n'est pas différentiable en 0. La perte de **Huber** offre un compromis intéressant entre ces deux pertes.
- ***Gradient clipping*** : La technique de **gradient clipping** est utilisée dans l'algorithme TDQN pour résoudre le problème de l'explosion de gradient qui induit des instabilités significatives pendant l'entraînement du réseau de neurones profond. Cette technique consiste à limiter la magnitude du gradient en le projetant sur un seuil prédéfini afin d'empêcher les gradients de devenir trop grands pendant l'apprentissage. Cela peut aider à améliorer la stabilité et la convergence de l'apprentissage profond.
- **Xavier *initialisation*** : L'**initialisation Xavier** est une méthode utilisée dans les réseaux de neurones pour améliorer la convergence de l'algorithme. Contrairement à l'initialisation aléatoire des poids du réseau de neurones utilisée dans le DQN classique, l'**initialisation Xavier** consiste à définir les poids initiaux de manière à ce que la variance des gradients reste constante à travers les différentes couches du réseau de neurones.
- ***Batch Normalization*** : La technique de normalisation par lot en français, introduite par Ioffe et Szegedy en 2015, consiste à normaliser les couches d'entrée d'un réseau de neurones en ajustant et en mettant à l'échelle les fonctions d'activation. Les avantages à cette méthode sont : une phase d'entraînement plus rapide et plus robuste, ainsi qu'une meilleure généralisation.
- **Techniques de régularisation** : Les techniques de régularisation sont mises en place dans la stratégie de trading DRL pour éviter un sur-apprentissage qui avait été observé lors des premiers essais. Trois techniques sont utilisées : le dropout, la régularisation L2 et l'arrêt prématuré de l'apprentissage (*early stopping*).
- **Pré-traitement et normalisation** : Avant de commencer la boucle d'entraînement de l'algorithme TDQN, les observations sont pré-traitées et normalisées. Comme l'application d'un filtre pour réduire le bruit, mais il y a un risque de perte d'information

pertinente, transformation de l'information en format plus pertinent, comme montrer les mouvements des courbes boursiers plutôt que le prix des *stocks*.

- ***Data augmentation*** : Les **techniques d'augmentation de données** sont utilisées dans l'algorithme de trading TDQN afin de compenser la limitation de la quantité de données disponibles et leur qualité souvent faible. Ces techniques comprennent le décalage de signal, le filtrage de signal et l'ajout de bruit artificiel. Ces méthodes permettent de générer de nouvelles données de *trading* artificielles qui sont légèrement différentes, mais qui représentent les mêmes phénomènes financiers. Cela permet d'augmenter la quantité et la diversité des données d'entraînement, améliorant ainsi les performances de l'algorithme.

Un Jupyter Notebook avec des affichages dynamiques a été créé pour simplifier la visualisation et l'analyse des résultats. Plusieurs fonctionnalités de *rendering* ont été implémentées pour illustrer l'avancement de l'apprentissage de l'agent, tels que la visualisation dynamique de l'évolution des prix en fonction du temps, les Q-values selon les stratégies de *trading*, le capital, la liquidité, etc. Des fonctionnalités d'évaluation ont été enrichies pour rendre la comparaison des résultats plus simple, tels que le tableau des métriques de performance pour l'analyse, voir la figure 1 pour une appréciation du rendu d'un entraînement du modèle sur les actions d'Apple. D'autres améliorations ont été apportées au code source, comme l'ajout de *trading* en temps réel sur le marché ouvert, la sauvegarde du modèle pour sauver le temps d'entraînement. Une analyse des limites du code et une exploration du comportement de TDQN a permis de connaître les faiblesses de l'algorithme actuelle, tels que la limite de réalisation des actions, la restriction sur le *trading* d'un cours boursier, les écarts de résultats entre les phases d'entraînement et de test.

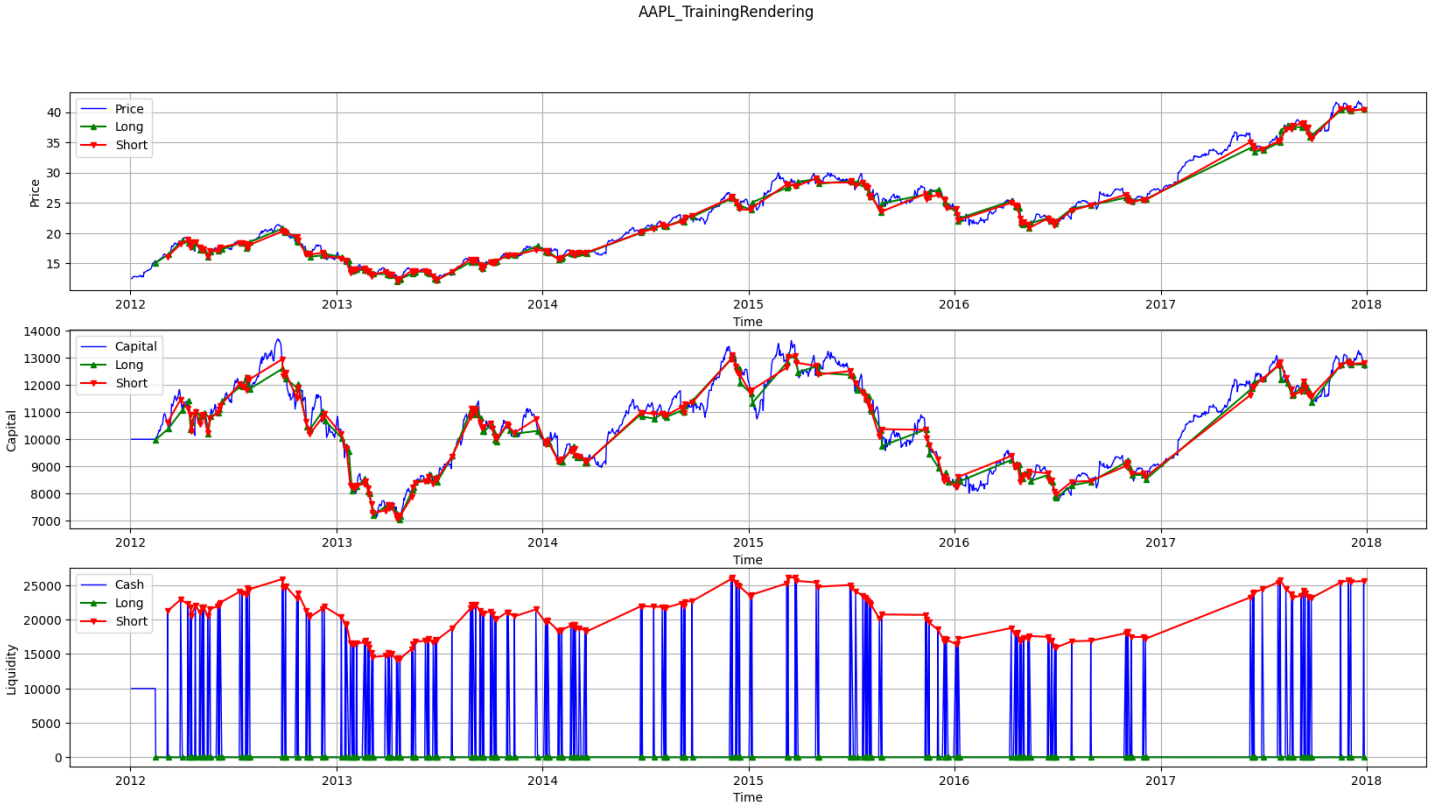


Figure 1 - Courbes des valeurs en prix, capitaux, liquidités du *bot* sur le cours boursier d'Apple

Expérimentations. Les données composées des bourses sont divisées en **ensemble d'entraînement** (à 75% de 2012 à 2017) et de **test** (à 25% de 2018 à 2019). Le *bot* a été lancé sur les actions d'Apple et de Tesla pour une période de 2 ans réparti en 50 épisodes, avec une fréquence de *trade* quotidienne basée sur des états de 30 jours (information courante ainsi que ceux des 29 derniers jours).

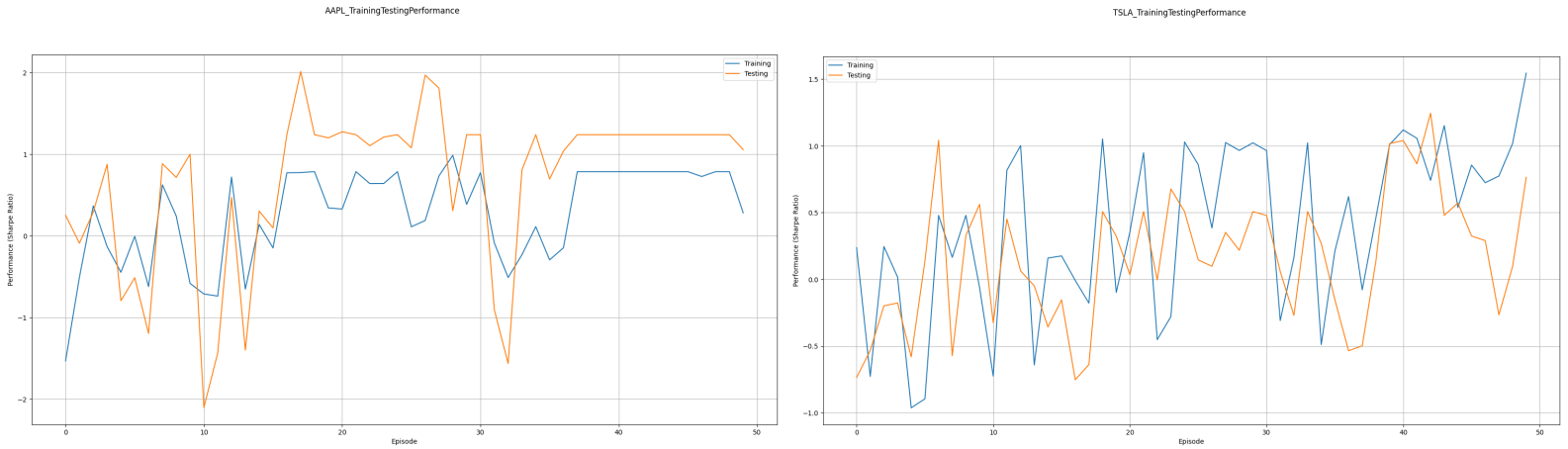
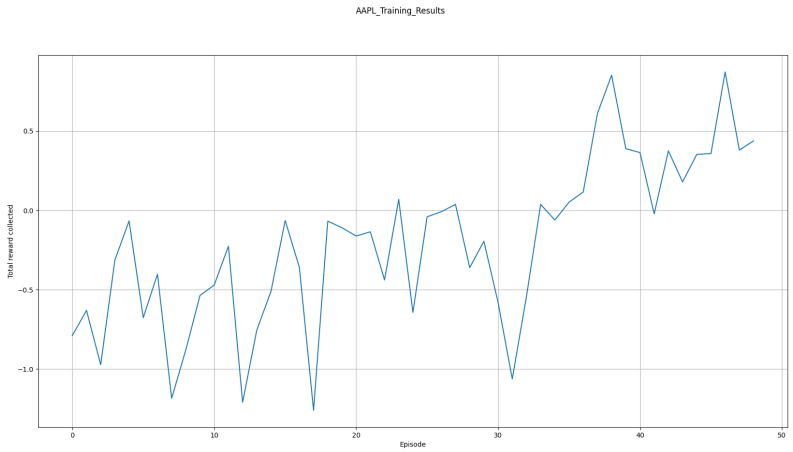


Figure 2 - Courbes d'apprentissage de l'agent sur les *stocks* d'Apple (gauche) et de Tesla (droite), *training* (bleu) et *testing* (jaune)



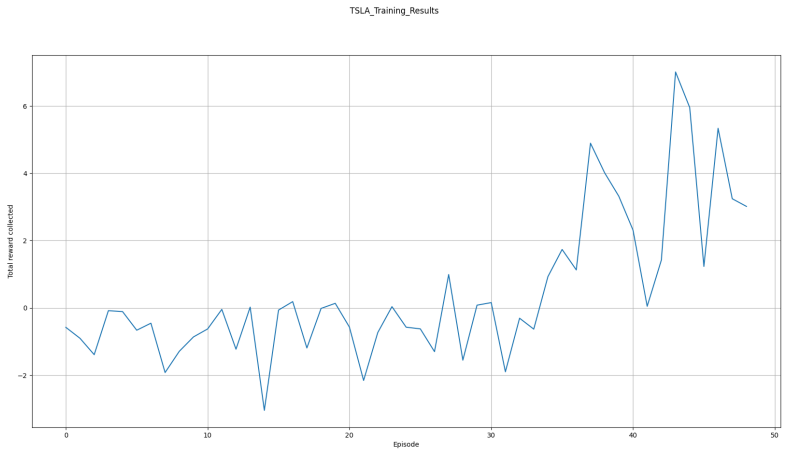


Figure 3 - Courbes de récompenses du système sur les *stocks* d'Apple (gauche) et de Tesla (droite)

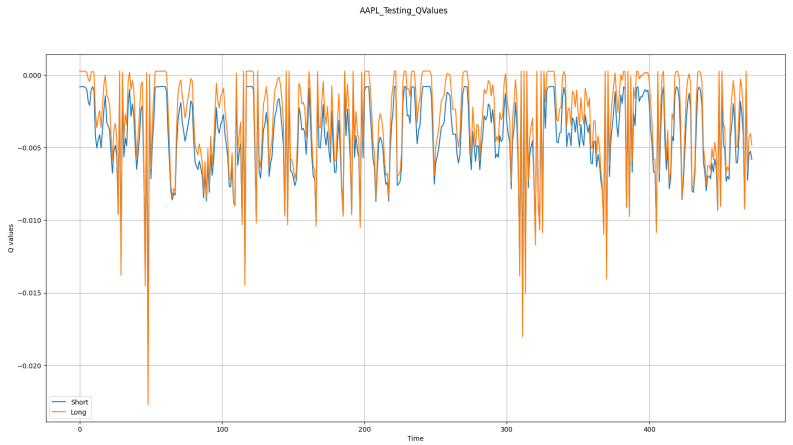
En figures 2 et 3, nous avons respectivement les courbes d'apprentissage et de récompenses du *bot* sur les actions d'Apple et de Tesla. Pour cette simulation, l'agent performe légèrement mieux en test sur Apple que sur Tesla, en remarquant une instabilité légèrement plus élevée sur cette dernière. Sur les courbes de récompenses, la conclusion est inversée, les récompenses sont moins stables pour Apple. En comparant avec les tableaux en figure 4, les résultats de l'agent sur Tesla sont beaucoup plus intéressants que ceux sur Apple, 378334 contre 2609 de profit. Cela est certainement dû à la longue période de chute des actions d'Apple (en regardant les métriques de *Drawdown*) sur les chiffres de ce tableau Apple est clairement plus risqué (en termes de *Sharpe* et *Sortino Ratio*).

Performance Indicator	TDQN (Training)
Profit & Loss (P&L)	2609
Annualized Return	5.96%
Annualized Volatility	24.68%
Sharpe Ratio	0.281
Sortino Ratio	0.373
Maximum Drawdown	48.51%
Maximum Drawdown Duration	145 days
Profitability	47.54%
Ratio Average Profit/Loss	1.187
Skewness	-0.335

Performance Indicator	TDQN (Training)
Profit & Loss (P&L)	378334
Annualized Return	32.10%
Annualized Volatility	46.50%
Sharpe Ratio	1.545
Sortino Ratio	2.419
Maximum Drawdown	24.16%
Maximum Drawdown Duration	15 days
Profitability	47.86%
Ratio Average Profit/Loss	2.750
Skewness	0.818

Figure 4 - Tableaux des métriques de performance en phase entraînement du *bot* sur les *stocks* d'Apple (gauche) et de Tesla (droite)

Les figures 5 et 6 donnent un aperçu des Q-*values* ainsi que les valeurs du *portfolio* du *bot* selon les actions de celui-ci sur les actions boursières d'Apple et de Tesla. Les Q-*values* pour Apple sont en moyenne négatives alors que ceux de Tesla sont en moyenne positives. De plus, en comparant la fréquence de *trading*, en phase de test, le *bot* a tendance à réaliser moins d'action effective par rapport à la phase d'entraînement, figure 4. De ce fait d'autres applications ont été réalisées et la conclusion reste inchangée, ce qui pourrait amener au questionnement sur le sous ou sur-apprentissage, voire le manque de généralisation du modèle. La figure 7 présente l'évaluation des performances pour la phase de test, en comparaison aux tableaux des métriques de la phase d'entraînement, en figure 4, le *bot* génère des profits semblables sur les deux *stocks*, de plus les valeurs comme le *Drawdown*, *Ratio Average Profit/Loss*, le *Sortino Ratio* sont plutôt proches. Sur cette période de test, Apple a un *Drawdown* relativement inférieur à celui de la période d'entraînement et les profits sont plus importants, pour Tesla le résultat est contraire. Cela laisse croire que le *bot* est sensible au *Drawdown*, ce facteur peut grandement influencer les performances du *bot*, en particulier les profits, le *Sharpe* et *Sortino Ratio*.



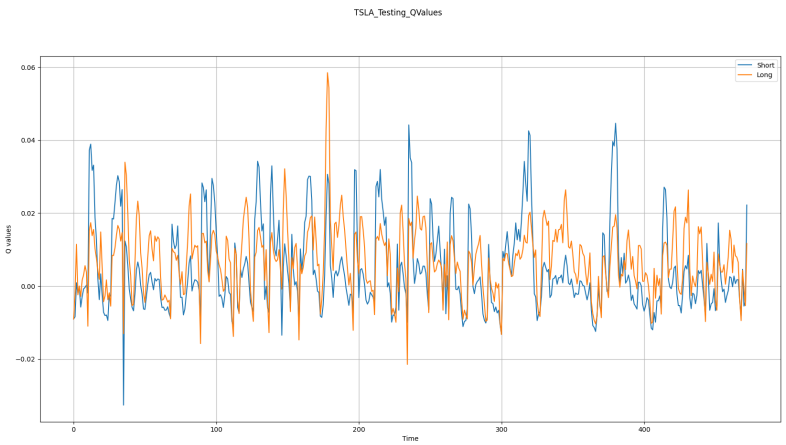


Figure 5 - Q-*values* selon les actions de l'agent sur les actions boursières d'Apple (gauche) et de Tesla (droite)

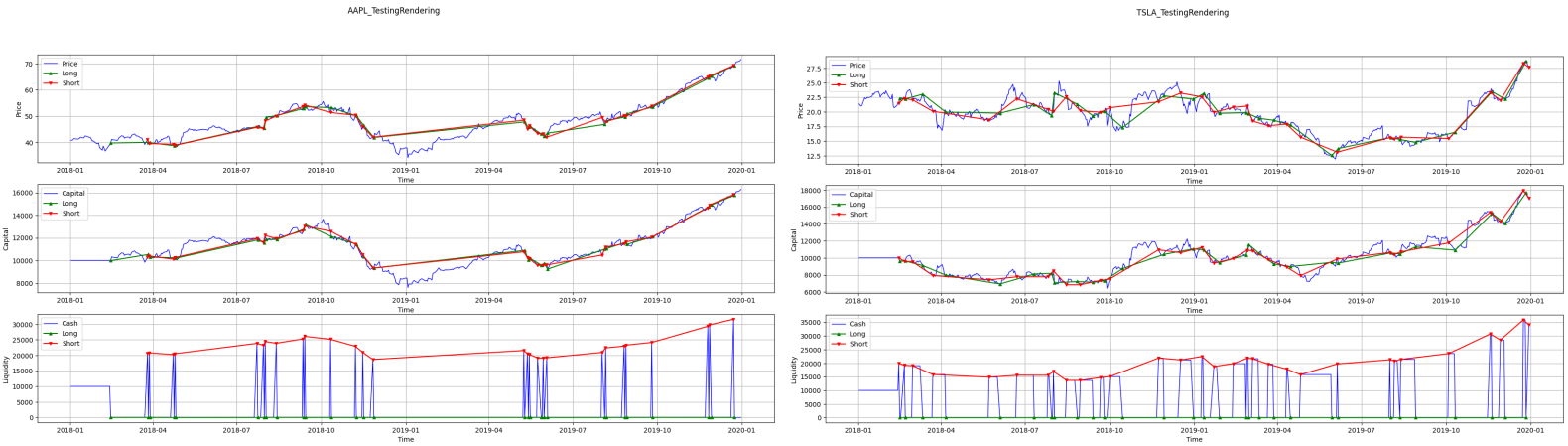


Figure 6 - Application de l'agent RL entre 2018 et 2020 sur les *stocks* d'Apple (gauche) et de Tesla (droite)

Performance Indicator	TDQN (Testing)
Profit & Loss (P&L)	6288
Annualized Return	24.91%
Annualized Volatility	26.50%
Sharpe Ratio	1.056
Sortino Ratio	1.323
Maximum Drawdown	44.20%
Maximum Drawdown Duration	62 days
Profitability	57.89%
Ratio Average Profit/Loss	1.338
Skewness	-0.463

Performance Indicator	TDQN (Testing)
Profit & Loss (P&L)	6873
Annualized Return	33.89%
Annualized Volatility	51.77%
Sharpe Ratio	0.764
Sortino Ratio	1.084
Maximum Drawdown	40.84%
Maximum Drawdown Duration	90 days
Profitability	50.82%
Ratio Average Profit/Loss	1.396
Skewness	0.252

Figure 7- Evaluation de performance en phase de test du *bot* pour Apple (à gauche) et Tesla (à droite)

Les résultats introduits précédemment par les auteurs ne sont pas tous validés par cette expérimentation. En effet, en phase de test, Tesla vis-à-vis de la métrique *Annualized Volatility* et *Sharpe Ratio*, Apple semble être plus stable que Tesla, avec un *Sharpe ratio* de 1.056 (Apple) par rapport à 0.764 (Tesla) et en volatilité annuelle 26.50% (Apple) contre 51.77% (Tesla). Tesla est clairement plus volatile sur cette période de simulation comme expliqué par les auteurs. De plus, selon le *Sortino ratio*, Apple (avec 1.323) semble avoir moins de chance de perdre par chute de sa valeur que Tesla (0.764). Les valeurs des actions d'Apple semblent bien être plus stables que celles de Tesla. Cependant, dans cette simulation le *bot* a produit légèrement moins de profit sur les actions d'Apple en comparaison avec ceux de Tesla, bien que la différence n'est pas énorme. Cela peut-être expliqué par la volatilité des actions de Tesla, en effet la volatilité importante n'est pas un indicateur désavantageux pour la gestion du *portfolio*, si l'agent est entraîné à profiter des volatilités, il peut générer un plus grand profit global dû aux explosions de valeurs boursières et appliquer de manière plus optimale les stratégies en situation de *Drawdown*, dont une faiblesse a été noté précédemment.

Conclusion. Par les analyses précédentes, nous pouvons conclure que le *bot* actuel a une faiblesse sur les cours boursiers a longue période de *Drawdown*, ceci pourrait être expliqué par le manque d'entraînement ou de généralisation lors de l'apprentissage. Les métriques à considérer ne sont pas que les valeurs du profit, mais également les *Sharpe* et *Sortino Ratio* indiquant le rendement en prenant en compte le facteur de volatilité, qui n'est pas nécessairement une mauvaise caractéristique pour un cours boursier. Les fonctionnalités présentées en méthodologie ont permis de produire ces analyses et de valider certaines des hypothèses des auteurs, cependant, pour plus de rigueur, il aurait fallu moyenner sur plusieurs simulations et augmenter l'ensemble des données pour éliminer les hypothèses de manque de données, de sous-apprentissage ou de manque de généralisation. Malheureusement, le coût d'apprentissage et la limitation en temps et en ressources restreignent l'exploration de ces solutions à grande échelle.

Pour la suite, nous pouvons réaliser plusieurs simulations en parallèle sur différentes données boursières à petit échelle et comparer la moyenne des résultats obtenus afin d'essayer de valider ou réfuter les hypothèses précédentes, puis principalement introduire une autre technique d'apprentissage par renforcement, comme une méthode de *Policy-Gradient*, le *Proximal Policy Optimization* (PPO), ou le *Long Short-Time Memory* (LSTM) qui est répandu en séries temporelles, et les comparer à TDQN ainsi que les méthodes classiques. Dans la limite du temps disponible, essayer d'amorcer une analyse des différents modèles selon les caractéristiques des différentes courbes boursières pour discuter et améliorer l'agent à l'étape 3.