Deep learning pour les marchés financiers

Al-Romhein Jean-Marc 1588498
Gélinas Olivia 1784417
Lelièvre Philippe 1695014
Proulx-Meunier Vincent 1399037

Abstrait

La prédiction du prix d'une action à la bourse est une tâche extrêmement complexe. Dans ce travail nous expérimentons une technique et une architecture d'intelligence artificielle capables de prendre des actions par rapport au prix de l'or.L'architecture de l'agent proposé est composée de trois modules : un Encodeur, un GRU et un module A2C. Nous utilisons notre implémentation pour faire plusieurs expériences, afin de comparer les résultats selon les différents hyper-paramètres utilisés. Après des expérimentations, nous n'avons pas réussi à implémenter un agent capable d'interagir avec le marché boursier de façon stable. En effet, bien qu'on observe quand même un certain apprentissage au fil des itération, la variance dans les résultats est trop grande. Lien vers le projet Github : https://github.com/Lap1n/inf8225

Introduction

Dans ce travail, nous nous sommes penchés sur l'étude du marché boursier et l'application de méthodes d'apprentissage machine et d'intelligence artificielle à ce sujet. Notre objectif est de créer un système pouvant faire des prédictions sur les fluctuations de la bourse, une tâche difficile considérant les facteurs pouvant influencer le domaine financier. Puisque nous voulions essayer de réduire ces facteurs, nous avons choisi d'examiner le prix de l'or. L'or est considéré une valeur refuge, et nous considérons que celle-ci mènerait donc vers un apprentissage plus stable.

Nous avons donc approché ce problème en essayant de prédire à travers différents moments donnés dans le temps s'il serait mieux de vendre ou d'acheter de l'or, ou bien de rien faire. Dans les prochaines sections, nous décrirons en plus de détail notre approche théorique et nos résultats, que nous décrirons par la suite, tout en faisant une analyse critique du travail effectué.

Travaux antérieurs

Plusieurs travaux antérieurs pouvant s'appliquer à la prédiction du marché boursier existent. Ces travaux peuvent soit traiter de l'application de méthodes d'intelligence artificielle à ce domaine, ou bien parler en termes plus généraux de méthodes pouvant s'appliquer pour résoudre notre problème. Nous avons relevé une manière principale de procéder: construire un agent qui fera des actions (long, short, neutre) dans l'environnement, utilisant des techniques décrites ci-bas. Dans le contexte des marchés de type « futures », « long » signifie d'acheter des actions pour les vendre par la suite, tandis que « short » signifie de vendre d'abord (même si on a pas encore en possession l'or) pour ensuite acheter pour conclure la transaction.

Agents agissant dans un environnement Certains travaux semblables aux nôtre existent, dont nous nous sommes inspirés. Une méthode implique une combinaison de techniques de Deep Learning (DL) et de Reinforcement Learning (RL) (Deng et al. 2017). Selon cette méthode, on peut utiliser une partie de DL pour obtenir les informations sur le marché nécessaires pour l'apprentissage à partir de *features*. Puis, on utilise une partie de RL pour interagir avec ces représentations profondes pour faire des décisions financières et tenter d'accumuler des récompenses dans un environnement inconnu. Dans l'ouvrage proposant cette méthode, on fait l'implémentation avec un réseau de neurones profond et récurrent. Ce travail représente donc une façon de combiner différentes méthodes d'apprentissage pour faire des prédictions dans des marchés financiers.

De plus, Si et al. tentent de faire des prédictions sur le marché financier avec une méthode semblable (Si et al. 2017). Dans leur étude, les chercheurs développent un réseau de neurones profond pour découvrir les features du marché financier. Puis, ils utilisent un LSTM pour faire des décisions en continu sur les actions à prendre dans le marché financier. Ils formulent leur problème selon deux objectifs, soit de maximiser le profit et minimiser le risque, tout en considérant le coût de chaque transaction. Le travail était donc assez semblable au précédent, mais considère plus de facteurs pouvant influencer nos décisions dans le marché financier. Ces travaux ont donc inspiré notre travail actuel.

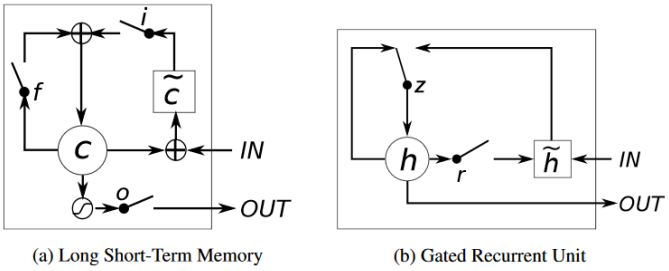
Pour ce qui est de la façon d'entraîner notre réseau pour que l'agent prenne les décisions optimales, plusieurs méthodes existent. Une de ces méthodes est discutée par Mnih et al. dans leur travail sur les méthodes asynchrones d'acteur critique (Mnih et al. 2016). En particulier, ils discutent des méthodes A3C et A2C. En effet, A2C est une version synchrone d'A3C. Nous discuterons davantage de la méthode A2C dans le restant de ce travail, puisque nous avons utilisé cette méthode spécifiquement pour faire nos apprentissages.

Approche théorique

Cette section présente l'approche théorique et l'architecture utilisé ainsi que ses composantes. Notre projet se démarque des travaux précédents du fait que nous utilisons un module GRU au lieu d'un LSTM ainsi que du fait que nous utiliseront la technique d'entraînement par renforcement A2C.

Gated Recurrent Unit (GRU)

Notre architecture utilise un Gated Recurrent Unit. Il s'agit en fait d'une version améliorée d'un réseau neuronal récurrent standard (RNN). Contrairement au RNN, il peut résoudre le problème du gradient qui s'évanouit. Le GRU peut également être considéré comme une variante du "Long short-term Memory" (LSTM) car les deux sont conçus de manière similaire et, dans certains cas, produisent des résultats tout aussi excellents. La figure 1 illustre la différence entre un LSTM et un GRU (Chung et al. 2014).



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Figure 1: LSTM vs GRU selon l'article de Chung et al.

Les GRUs peuvent stocker et filtrer des informations en utilisant leur mise à jour et en réinitialisant les portes. Cela élimine le problème du gradient qui s'évanouit puisque le modèle ne nettoie pas la nouvelle entrée à chaque fois, mais conserve les informations pertinentes et les transmet aux prochaines étapes du réseau.

Architecture

L'architecture est composé de 3 principaux éléments: l'encodeur, le Gated Recurrent Unit (GRU) et le Advantage-Actor-Critic (A2C). La figure 2 représente l'architecture "déroulée" de notre système récurent. Nous avons en réalité qu'un seul module GRU qui prend ses valeurs de sortie précédentes.

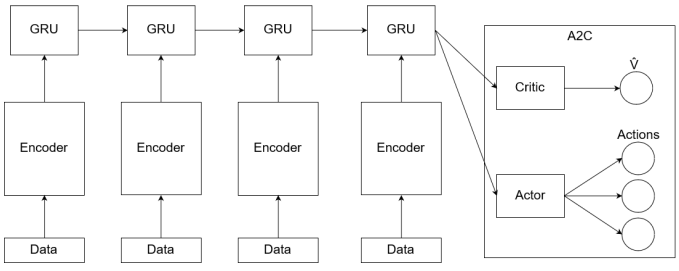


Figure 2: Aperçu du réseau et de l'architecture avec la récurrence "déroulé"

Comme l'on peut le voir sur la figure 3, l'architecture débute avec un encodeur. Il est lui même composé de 3 couches de neurones. Il prend en entrés 50 données et produit 20 valeurs de sortie. Toutes les neurones de ce réseau sont implémentées avec une fonction d'activation rectify (ReLU).

$$h(x) = max(0, x) \tag{1}$$

Ensuite, les 20 valeurs de sortie de l'encodeur entrent dans un GRU. Comme expliqué précédemment, le GRU est un module récurent et il prend donc aussi en entrée la sortie du GRU au temps précédent t-1. Cette récurrence est faite sur les 3 dernières sorties du GRU. À la sortie du GRU, 128 valeurs sont fournies au module A2C.

Le module A2C est composé de 2 éléments : le critic et l'actor. Le but de l'actor est de mapper les 128 valeurs d'entrés vers 3 valeurs de sortie qui correspondent aux 3 actions possibles à prendre soit Long, Neutral et Short. Le critic, lui, sort une seule valeur de sortie qui correspond à l'estimation des récompenses. Le fonctionnement de A2C sera détaillé plus tard dans le rapport, à la section Méthode d'entraînement.

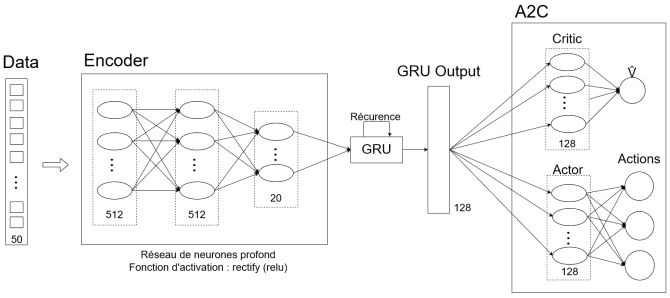


Figure 3: Architecture détaillé du réseau de neurone profond

Données d'entraînement

Notre méthode de collecte de données s'inspire fortement de la méthode de l'article *A Multi-objective Deep Reinforcement Learning Approach for Stock Index Future* (Si et al. 2017).

Tout d'abord, au niveau des d'entraînements de training et de test, nous disposons en tout du cours de l'or de la précédente année et 20% de ces données serviront pour tester notre agent alors que le reste servira à l'entraînement de notre agent.

Notre échantillon de données est composée de 450 données par jours (bars) sur le prix de l'or sur une période d'une année. Les données sont échantillonnées à la minute durant le jour uniquement (9h30 à 17h). Étant donné que notre modèle tente de faire des actions sur de courtes durées de temps, il n'est pas pertinent de considérer la journée entière à la fois. Le modèle va donc considérer 45 minutes à la fois pour chaque itération. Cette période de 45 minutes, soit 45 données, est appelée la fenêtre. Une journée pourra donc fournir 400 itérations au modèle. La journée boursière commence à 9h30, cependant au temps *t* il faut avoir les 45 données précédentes. C'est pourquoi le modèle commence

ses prédiction seulement à 10h15. Cette fenêtre de 45 données va glisser d'une minute à chaque itération.

Nous avons effectué certaines étapes de prétraitement sur nos données d'entraînement étant le cours de l'or de la dernière année.

Tout d'abord, il faut transformer les données selon la fonction suivante :

$$z_t = x_t - x_{t-1} \tag{2}$$

z_t : variation du prix de l'or
x_t : prix de l'or au temps t

En d'autre mot, il ne faut pas représenter les données comme la valeur de l'or, mais plutôt comme la variation de l'or à chaque unité de temps.

Contrairement à l'article de référence (Si et al. 2017), nous avons décidé de normaliser la variation de l'or. Une fois les variations de l'or obtenue pour chaque instant, ces données sont normalisées pour obtenir une valeur allant de 0 à 1. Les données sont normalisées à l'aide de l'équation suivante:

$$z_{t_N} = \frac{z_t - z_{t_{min}}}{z_{t_{max}} - z_{t_{min}}} \tag{3}$$

Le vecteur d'entrée de notre modèle est composé de 50 éléments. Les 45 premiers sont les valeurs de l'or normalisées pour chaque instant de la fenêtre glissante. Les 5 dernières données représentent la trajectoire générale dans laquelle se dirige la variation de l'or par rapport à diverse périodes de temps. Cette valeur est le momentum, qui représente une variation grossière du prix de l'or sur un période donnée. On calcule ce momentum sur une durée de 3 heures, 5 heures, 1 jour, 3 jours et 10 jours. Le momentum est calculé à l'aide de l'équation suivante:

$$m_p = x_{t_N} - x_{t_N - p} \tag{4}$$

m : momentum
p : période sur laquelle le momentum est évalué

Ceci conclut la phase de prétraitement des données. Le modèle reçoit donc en entrée un vecteur de taille 50 composé de 45 valeurs de variations d'or normalisées et de 5 valeurs de momentum. À chaque itération le modèle va recevoir ce vecteur à nouveau pour sa fenêtre glissante correspondante.

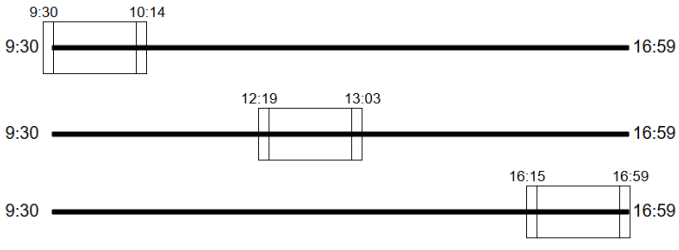


Figure 4: Principe le la fenêtre glissante

Représentation de l'environnement financier

Pour représenter notre environnement, nous avons implémenté l'interface GYM de Facebook pour faciliter l'implémentation de la méthode d'apprentissage A2C qui nécessite plusieurs agents à la fois.

La fonction reset prend une journée aléatoire de l'ensemble de donnée dans un tampons et initialise l'index représentant le moment dans la journée au début. Une journée aléatoire est nécessaire puisqu'il y a plusieurs agents en parallèle et il faut éviter que ces agents agissent sur la même journée. De plus, le choix d'une journée aléatoire a été fait pour essayer de construire un environnement favorisant des données d'entraînements indépendantes d'un épisode (journée) à l'autre.

La fonction step prend en entrée une action (Long, Short et Neutre) et retourne la prochaine observation de l'environnement qui constituera le prochain input à notre réseau de neurone pour déterminer la prochaine action à effectuer. Cette fonction retourne aussi la récompense immédiate et un booléen déterminant si la journée est terminé (ce qui nécessitera d'effectuer un reset pour passer à une autre journée).

Équation de récompense

La récompense est donnée par l'équation suivante:

$$R_t = \delta_{t-1} \times z_t - c \times ceil(|\frac{\delta_t - \delta_{t-1}}{2}|) \tag{5}$$

R_t : récompense au temps t
δ_t : action choisie au temps t
c : coût de transaction

L'objectif de notre réseau de neurones est de maximiser la somme des récompenses pour chaque instant.

Méthode d'entraînement

La méthode d'entraînement utilisée est la méthode Actor Advantage Critic (A2C). Dans un modèle d'apprentissage par renforcement, un agent parcours différents états **S** dans un environnement en prenant des actions, tout en essayant de maximiser ses gains **R**. Le *critic* va estimer des valeurs d'états **Ve(s)**, soit combien de gains sont attendus de l'état actuel jusqu'à l'état de fin. Cette valeur est obtenue grâce au réseau de neurones du *critic*. L'acteur va recommander une action basé sur la politique **A**. Dans notre cas nous avons trois actions. Généralement l'action avec la plus forte recommandation est sélectionnée. L'agent va donc faire cette action et noter le gain obtenu. Ce processus est répété un nombre déterminé de fois jusqu'à ce que l'agent décide de prendre une pause et de réfléchir sur ses observations. L'agent va supposer que la dernière valeur d'état correspond à la valeur d'état réelle de l'état de fin. Ceci introduit un biais, mais permet de diminuer la variance par rapport à une approche Monte Carlo.

Les valeurs d'états **V(s)** réelles avec biais sont calculées à l'aide de l'équation suivante:

$$V(s) = V_e(s+1) + R(s) \tag{6}$$

$V(s)$: valeur d'état réelle biaisée
$V_e(s+1)$: valeur d'état estimée à l'état suivant (*ground truth*)

Les erreurs par rapport aux valeurs d'états estimées sont exprimés à l'aide de l'équation suivante:

$$E(s) = V(s) - V_e(s) \tag{7}$$

$E(s)$: erreur de la valeur d'état estimée à l'état s

Il est maintenant possible de faire une mise à jour des paramètres du réseau de neurone qui a généré **Ve(s)** et **A(s)** en minimisant la fonction de perte pour la *mini-batch*. La perte est composée de trois parties principales.

1. La perte totale pour la *mini-batch* est donnée par l'équation suivante :

$$Perte = Perte_{action} + Perte_{tat} - F \times entropie \tag{8}$$

F : facteur d'entropie

2. La perte correspondant à la valeur d'état estimé tient en compte de l'erreur d'estimation. Cette perte est données par l'équation suivante:

$$Perte_{etat} = \frac{1}{n}\sum_{s=1}^{n} E(s)^2 \tag{9}$$

n : taille de la mini-batch (nombre d'étapes avant que l'agent reflète)

3. La perte correspondant à l'action choisie tien en compte du choix d'action effectuée par l'*actor*. Cette perte est donnée par l'équaton suivante:

$$Perte_{action} = -\frac{1}{n}\sum_{s=1}^{n}(E(s) - logPr(s)) \tag{10}$$

$logPr(s)$: log probabilité de chaque action à l'état s

Finalement, l'entropie est un facteur qui sert à favoriser l'exploration. Ceci fait en sorte que l'acteur ne va pas toujours prendre l'action suggérée qui a la meilleur probabilité. Prendre l'action immédiate qui semble la meilleure ne garanti pas un meilleur gain à long terme.

À chaque journée, l'acteur va répéter ce processus et mettre à jour son réseau de neurones jusqu'à ce que la fin de la journée boursière soit atteinte.

Algorithme pour mise à jour des paramètres

Pour ce qui est de l'optimisateur, nous avons choisi celui nommé ADAM (Kingma and Ba 2014) puisqu'il semble être efficace dans les environnements qui contiennent beaucoup de bruit comme le notre .

Discussion

Pour les résultats, on a décidé de se limiter qu'à quelques hyper-paramètres, soit le *learning rate*, le coefficient d'entropie, le coefficient du *loss* ainsi que le nombre de step entre chaque backward propagation. De plus, nous allons aussi grandement limité le nombre d'épisode par expérimentation. La raison pour ces limitations est le fait que nos ordinateurs ne sont pas très puissants, alors difficile de faire plusieurs expériences sur un grand nombre d'épisodes.

Meilleurs résultats obtenus

Voici les résultats obtenus avec un learning rate de 0.0001, un coefficient d'entropie de 0.0001, un coefficient de loss de 0.001 ainsi que des mises à jour aux 100 step. On remarque ici que l'agent débute avec des performances négatives et se stabilise autour du 0$ en moyenne par jour. On observe aussi beaucoup de variance dans les résultats : autant à certains épisode, l'agent fait 400$ en moyenne, autant à celui d'après il peut faire -400$.

La figure 6 présente les performances de l'agent sur des données qui n'ont pas servi a l'entrainement, ce qui fait que les résultats ici sont plus représentatif de la réalité. On peut voir ici que le score moyen ici passe de -600 et s'améliore jusqu'à atteindre un score moyen de -34$ par journées. Ce résultat relativement décevant montre que notre agent n'est pas encore au point pour être utilisé dans un environnement réel. Cependant, on observe quand même une nette amélioration au fil de l'entraînement.

Pour le prochain résultat décrit dans le tableau 1, nous avons sélectionné le modèle ayant donné le meilleur résultat dans l'entraînement précédant. On peut remarquer avec le tableau 1 que l'agent s'en sort avec un profit moyen de 31.21 $ par jour. Cependant, on observe le même problème que celui mentionné dans le graphe du score obtenu avec les données d'entraînement qui est la présence d'une grande variance dans nos résultats. En effet, certaines journées, on peut perdre jusqu'à -1434.80$ alors que d'autre journée on peut gagner jusqu'à 1725.19 $. On peut donc se dire qu'il y a place à de l'amélioration si l'on veut dans le future utiliser notre agent dans un environnement réel, parce qu'il va être difficile pour une personne d'enchaîner des journées avec des grandes pertes successives.

Effet du nombre d'étape entre chaque mise à jour des poids

Tout d'abord, on a tenté de mesurer l'influence du nombre d'étape entre chaque mise à jour des poids de notre modèle. On observant les figures 7, 8 et 9, on remarque que pour un nombre de 50 pas entre chaque mise à jour que le modèle tombe dans un minimum local de 0 $ par jour, ce qui implique que l'agent ne produit aucune transaction, ce qui est problématique. En augmentant cet hyper-paramètre, passé un certain seuil, on remarque qu'il est difficile de voir l'influence de cet hyper-paramètre dans le cadre de notre sujet d'apprentissage.

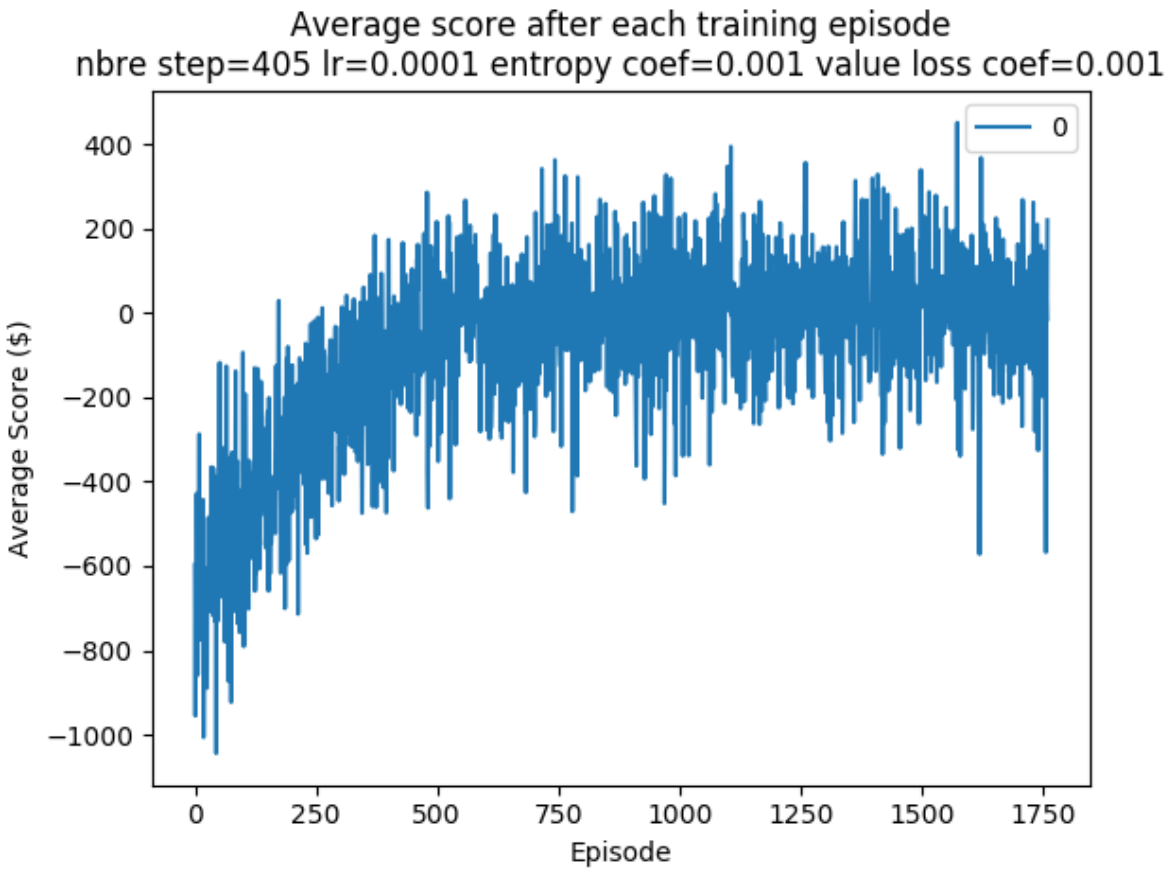


Figure 5: Profit moyen par jour selon le nombre d'épisodes d'entraînements

Effet de la valeur du taux d'apprentissage

Pour ce qui est de l'influence du taux d'apprentissage, on remarque aisément sur les figures 10, 11 et 12 que plus celui-ci est petit, plus ça prend du temps avant de converger. On peut donc conclure qu'un taux de 0.0001 est suffisant pour nos expérimentations.

Effet du coefficient de perte (*loss*)

Nous avons aussi tenté de mesurer l'influence du coefficient de *loss* dans l'algorithme A2C dans la performance de notre système de transactions automatisées. Les résultats qui sont représentés par les graphes 13, 14 et 15 sont difficiles à comparer, mais on semble voir que plus le coefficient de *loss* est petit, moins il semble de variance d'un épisode à l'autre.

Analyse critique

Il y aurait plusieurs éléments à critiquer et à explorer dans des expérimentations futures.

Limites

Tout d'abord, il y le fait que nos données d'entraînements ne se concentrent que sur la dernière année du cours de l'or. Idéalement, il faudrait avoir plusieurs années de données pour éviter que nos prédiction ne soit trop corrélé avec le cours récent de l'or. Plus de données rendraient nos observations plus généralisables.

De plus, on doit considérer que les transactions ne sont pas instantanées dans le monde réel, mais que notre approche les traite comme si elles l'étaient. Il serait donc intéressant de reconsidérer ce travail en prenant en compte l'effet du délai qui a lieu lorsqu'on fait une transaction. De cette façon, nos prédictions refléteraient mieux le monde réel et s'appliqueraient mieux à celui-ci.

Semblablement, nous n'avons pas considérer le volume de transactions ayant lieu à un moment donné dans notre analyse. Un nombre faible ou élevé de transactions à un certain moment indiqueraient différentes interprétations du marché, et donc pourraient mener à choisir différentes actions. Prendre en considération le volume de transactions supporterait alors aussi une meilleure représentation du monde réel, permettant ainsi de prendre de meilleures décisions boursières.

Éléments à explorer

Évidemment, on peut remarque que dans nos expérimentations, nous n'avons pas essayer de modifier les hyperparamètres liés à notre architecture comme le nombre de

Figure 6: Profit moyen par jour obtenu sur des données de tests selon le nombre d'épisodes d'entraînements

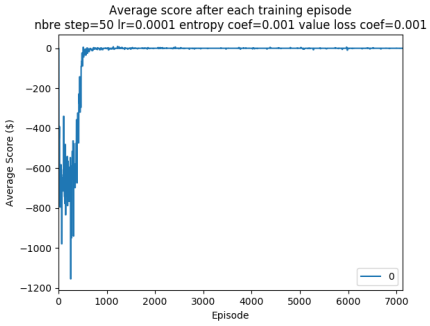


Figure 7: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec rétro-propagation aux 50 steps

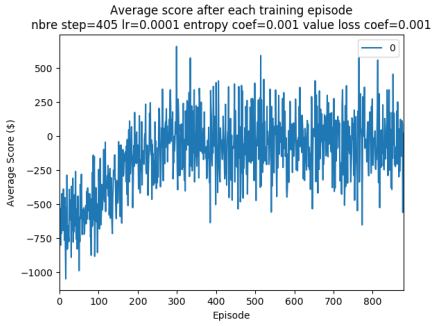


Figure 8: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec rétro-propagation aux 405 steps

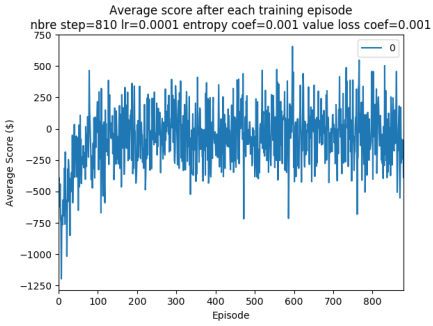


Figure 9: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec rétro-propagation aux 810 steps

couches dans notre encodeur ou bien la grosseur du *hidden size* dans le GRU. La raison pour laquelle nous avons laissé notre architecture intact est le fait que nous ne possédons pas les capacités matériels pour tester différentes configuration. En effet, les entraînements ont été particulièrement long avec nos machines. Il serait bien sûr très intéressant d'évaluer l'impact des modifications de l'architecture sur les performances de notre agent.

Dans le futur, il existe plusieurs éléments que l'on pourrait explorer. En premier, il serait pertinent de continuer ce travail en y apportant certains correctifs vis-à-vis les limites discutées dans la section précédente. Par exemple, étant donné l'origine des données que nous avons utilisé, il serait possible d'obtenir des informations sur le volume de trans-

Score moyen par jour ($)	Variance	Nombre de transaction moyen par journée	Récompense totale sur l'ensemble de test($)	Meilleur score ($)	Pire score ($)
31.21	453.66	2.39	5492.99	1725.19	-1434.80

Table 1: Solutions aux problèmes

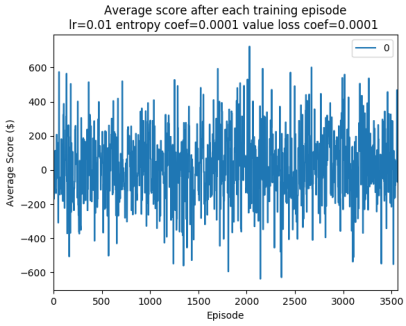


Figure 10: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec lr = 0.01

Figure 11: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec lr = 0.001

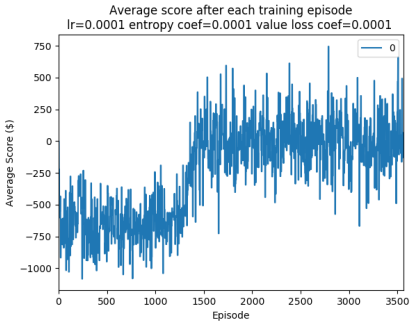


Figure 12: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec lr = 0.0001

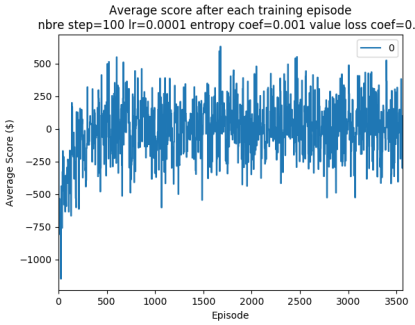


Figure 13: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec un coefficient de *loss* de 0.1

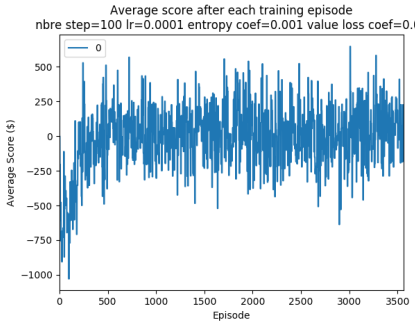


Figure 14: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec un coefficient de *loss* de 0.01

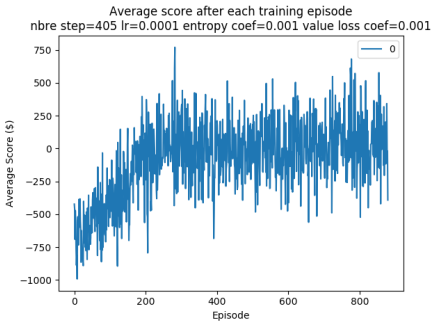


Figure 15: Profit moyen par jour selon le nombre d'épisodes d'entraînements avec un coefficient de loss de 0.001

actions à un temps donné, nous permettant donc d'inclure cette variable dans notre analyse. On pourrait aussi ajouter des indicateurs dans les données d'entrés comme le MA (moving average), l'indicateur RSI (Relative Strength Index) pour nommer que quelques possibilités.

Aussi, il serait possible d'améliorer notre implémentation de notre solution. Par exemple, une option potentielle serait d'entraîner un deuxième réseau de neurones qui analyserait le marché à plus long terme. L'objectif serait qu'il aide l'autre réseau à plus court terme à prendre des décisions. Ceci pourrait donc nous permettre d'obtenir de meilleurs résultats ainsi qu'explorer des pistes de solutions alternatives.

Conclusion

En conclusion, dans ce travail nous avons exploré comment les méthodes d'intelligence artificielle s'appliquent aux prédictions dans le marché boursier. Pour ce faire, nous avons implémenté un réseau de neurones, dont l'architecture est composée d'un Encodeur, d'un GRU et d'un module A2C. Nous avons utilisé cette architecture pour faire différents essais sur des données traitant du prix de l'or et sa fluctuation à chaque minute. Nos résultats, bien que décevant, indiquent une bonne possibilité qu'on puisse appliquer des méthodes d'intelligence artificielle efficacement à des problèmes tels que le nôtre dans le futur. Pour l'instant, il reste du travail à faire pour trouver des implémentations menant vers des résultats financièrement profitables.

References

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Deng, Y.; Bao, F.; Kong, Y.; Ren, Z.; and Dai, Q. 2017. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems* 28(3):653–664.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 1928–1937.

Si, W.; Li, J.; Ding, P.; and Rao, R. 2017. A multi-objective deep reinforcement learning approach for stock index future's intraday trading. In *Computational Intelligence and Design (ISCID), 2017 10th International Symposium on*, volume 2, 431–436. IEEE.