UCLouvain

epl

École polytechnique de Louvain

Prédiction du rendement des crypto-monnaies à l'aide de réseaux LSTM améliorés

Auteur: **Jérémy Degraeuwe**
Promoteur: **Charles Pecheur**
Lecteurs: **Alexander Gerniers, Sébastien Jodogne**
Année académique 2022–2023
Master [120] : ingénieur civil en informatique

Table des matières

Remerciements

Tout d'abord, je tiens à remercier le professeur Charles Pecheur pour son accompagnement rigoureux et qualitatif en tant que promoteur de ce mémoire. Ses conseils et retours ont tiré ce travail vers le haut, et son approbation continue a représenté une grande source de motivation.

Je remercie également le chercheur Harold Kiossou pour le temps qu'il m'a accordé. Sa validation de mes protocoles techniques m'a permis d'avancer plus sereinement dans la suite de mon travail.

Ensuite, je souhaite également remercier mes lecteurs Sébastien Jodogne et Alexander Gerniers pour avoir accepté de jouer ce rôle clé. J'espère qu'ils apprécieront lire le fruit de mon travail de recherche et d'expérimentation.

Par ailleurs, je tiens à remercier tous les professeurs que j'ai croisés au cours mes années d'études à l'École Polytechnique de Louvain. La qualité de leur enseignement me laisse confiant quant à mon futur professionnel en tant que jeune ingénieur.

Finalement, je remercie ma famille et mes amis pour leur soutien sans faille, même lors des années plus difficiles au début de mon parcours universitaire.

Introduction

Les crypto-monnaies

Les crypto-monnaies, produits de l'ère numérique, ont eu un impact significatif sur l'écosystème financier mondial depuis la création du Bitcoin en 2009. Fondées sur une technologie décentralisée, les crypto-monnaies offrent un système de *peer-to-peer* pour les transactions financières, sans intervention gouvernementale ni contrôle central. Fin 2021, le marché des crypto-monnaies comprenait plus de 10000 actifs numériques différents et culminait à une capitalisation boursière globale de plus de 2500 milliards de dollars (sa capitalisation s'élevant aujourd'hui à environ 1000 milliards de dollars).

Résultant de la fusion de la cryptographie, de l'économie et de l'informatique, les crypto-monnaies ont présenté de nombreuses opportunités et défis aux investisseurs, aux décideurs politiques et aux chercheurs. L'un des aspects les plus particuliers et les plus complexes des crypto-monnaies est leur nature hautement volatile. La promesse de rendements importants a attiré une pléthore de participants au marché, mais l'absence de réglementation et la dynamique imprévisible du marché rendent l'aventure fortement spéculative et risquée.

Le challenge de la prédiction financière

Les marchés financiers régissent le paysage économique mondial depuis des siècles. Leur impact profond sur la création de richesse et la stabilité économique a été au cœur des stratégies financières historiques et modernes. Le défi de prédire les mouvements de prix au sein de ces marchés a été et reste un objectif alléchant, poursuivi sans relâche tant par les investisseurs particuliers que par les institutions financières.

L'utilisation de l'informatique à cette fin a débuté dès les années 1970. Un article publié dans *The New York Times* en 1986 évoquait déjà la montée en puissance des

solutions automatisées, renversant les capacités humaines d'absorption de données [1]. Ceci n'était pourtant que la base de ce qui allait devenir un champ de recherche tentaculaire. Au cours des décennies suivantes, l'investissement dans les méthodes informatiques s'est rapidement développé, donnant naissance à l'élaboration de modèles mathématiques et algorithmiques de plus en plus complexes.

BlackRock, le fond de gestion d'actifs le plus capitalisé au monde à ce jour (près de 9500 milliards d'actifs sous gestion [2]), confirme l'importance de l'intelligence artificielle dans le domaine de la prédiction des mouvements financiers. En juin 2023, BlackRock a par ailleurs publié un article illustrant brièvement la performance d'un de leurs nouveaux modèles, spécialisé dans la prédiction des mouvements de marché, suite à une annonce d'*earning calls* d'entreprises (trad. résultats financiers) [3].

En 2020, 10% des fonds d'investissements d'Europe et des États-Unis impliquaient le *trading* algorithmique dans plus de 70% de leur volume d'échange. La même année, parmi tous les échanges réalisés sur le marché du Forex, 92% étaient automatisés [4].

Sans surprise, cette tendance ne s'est pas limitée aux marchés traditionnels. En 2019, une étude réalisée auprès de plus de 10000 particuliers a montré que 86% du capital échangé sur les marchés de crypto-monnaies étaient réalisés par des programmes automatisés [5].

Objectif

L'objectif principal de ce mémoire est de prédire le rendement de plusieurs crypto-monnaies différentes à l'aide de l'intelligence artificielle.

Plan du mémoire

Tout d'abord, nous commencerons par situer ce mémoire dans le contexte des compétitions Kaggle.

Ensuite, une analyse de la littérature nous permettra de décrire l'état actuel de la recherche autour de la prédiction de séries temporelles, et principalement dans le contexte de la finance.

Il est important de préciser que l'analyse sera tournée vers des modèles à échelle et complexité raisonnables. La nature du sujet de ce mémoire engendre en effet que des modèles sont aujourd'hui développés par de grandes entreprises. Ces technologies, associées à leur importante complexité et leurs grands besoins en puissance de calcul, placent ces solutions hors de portée du cadre de ce mémoire.

Puis, nous décrirons de façon théorique et détaillée des concepts et modèles utilisés dans ce travail.

Ces différents éléments nous permettrons d'ensuite entamer notre partie pratique.

Cette partie sera composée de l'analyse des données mises à disposition et d'une explication détaillée des architectures des différents modèles prédictifs choisis pour ce mémoire.

Enfin, une étude comparative visera à mettre en lumière les différences de performance parmi une succession de variantes, basées sur le principe des réseaux de neurones artificiels de type *Long Short-Term Memory* (LSTM).

Le modèle le plus abouti présenté dans ce mémoire, à savoir le réseau LSTM empilés, avec système d'attention et de *skip connections*, le tout optimisé par la méta-heuristique PSO, n'a pour l'heure, à la connaissance de l'auteur, pas encore fait l'objet de publication scientifique et représentera la contribution principale de ce mémoire.

1 Contexte – Compétition Kaggle

Ce mémoire s'appuie sur des données mises à disposition dans le cadre d'une compétition proposée sur le site Kaggle, qui héberge diverses compétitions dans le domaine de l'intelligence artificielle.

Chaque compétition est ouverte pour une période de temps donnée, durant laquelle tout le monde peut participer afin de tenter de remporter des prix en cas de bons résultats. Une fois cette période clôturée, la compétition et les données associées restent à disposition pour toute personne souhaitant s'y mesurer *a posteriori*. Il en résulte qu'à ce jour, la grande majorité des compétitions disponibles sur Kaggle sont clôturées. En effet, en date du 18 août 2023, sur les plus de 600 compétitions disponibles, seules 17 sont encore en période de participation officielle (dont seulement cinq décernent un prix). La compétition sur laquelle ce mémoire repose est elle aussi clôturée, depuis le 4 mai 2022.

Les descriptions détaillées du critère d'évaluation et des données mises à disposition seront effectuées à la section *4 : Analyse et préparation des données*.

2 Résumé de l'analyse de la littérature

Avant de tenter de réaliser des prédictions sur l'évolution de la valeur des crypto-monnaies, il convient de s'interroger sur le caractère prédictible de ce type de données.

2.1 Hypothèse des marchés efficients

Il existe, dans le domaine de la finance et de l'économie, une théorie appelée *l'hypothèse des marchés efficients* (HME) [6]. Selon cette théorie, le prix actuel d'un actif reflète l'entièreté de l'information liée à cet actif. Dès lors, il serait impossible de prédire son évolution, puisque aucun actif ne pourrait être sous- ou sur-évalué.

Cette hypothèse peut prendre plusieurs formes :

- **Faible** : Sous cette forme, il est considéré que toute l'information contenue dans l'historique complet du prix de l'actif est traduite dans le prix actuel. Dès lors, cela rend impossible la prédiction à l'aide d'indicateurs techniques orientés sur les prix antérieurs. La prédiction reste toutefois possible, à l'aide d'informations externes, trouvées par exemple dans la santé financière d'entreprises, dans la situation géo-politique d'un pays, etc.
- **Semi-forte** : Sous cette forme, l'information reflétée dans le prix représente toute l'information publiquement disponible. Ceci rend l'analyse fondamentale elle aussi inutile dans un but de prédiction. Seules des informations privées pourraient encore permettre de prédire le mouvement de l'actif.
- **Forte** : Dans cette version extrême, il n'existe aucune information, publique ou privée, qui ne soit pas reflétée dans le prix de l'actif, ce qui empêche indiscutablement toute prédiction fiable.

Au regard de ces éléments, nous pouvons nous interroger quant à la nature du marché des crypto-monnaies, pour déterminer s'il s'agit ou non d'un marché

efficient.

Heureusement pour les investisseurs, il a été prouvé, par Urquhart A. en 2016, que le marché du Bitcoin n'était pas un marché efficient [6]. Il est cependant précisé dans cette publication que les tests montrent une tendance pour le marché du Bitcoin vers l'efficience. Nous reviendrons sur ce point à la section *7 : Résultats*.

En guise de preuve de l'inefficience du marché du Bitcoin, Devavrat S., en 2014, a proposé une méthode qui lui a permis de réaliser un rendement de 89% en seulement 50 jours sur le marché du Bitcoin [7].

Bien qu'il soit possible de penser que de tels résultats aient été obtenus par un facteur chance, la valeur du *sharpe ratio* de ses échanges prouve une réelle capacité de prédiction de l'évolution du prix. En effet, le *sharpe ratio* permet d'évaluer le niveau de risque d'un investissement par rapport à ce qu'il peut rapporter. Une valeur inférieure à 1 est considérée mauvaise (trop de risque par rapport au gain potentiel). Pour une valeur entre 1 et 2, l'investissement est considéré correct. Entre 2 et 3, l'investissement est jugé très bon, et au-delà, il est jugé excellent [8][9]. La méthode ayant rapporté 89% de rendement en 50 jours avait un *sharpe ratio* de 4.1.

2.2 Prédiction de séries temporelles financières

Dans une publication de 2020 [10], les auteurs ont comparé les performances de deux modèles statistiques (LR et *Linear Discriminant Analysis*) et de cinq modèles de *machine learning* (*Random Forest*, XGBoost, *Quadratic Discriminant Analysis*, *Support Vector Machine* et LSTM) dans l'exercice de la prédiction du prix du Bitcoin. Les modèles statistiques se sont montrés plus performants quand il était question de prédire le prix journalier du Bitcoin. En revanche, lorsqu'il fallait prédire l'évolution du prix sur des pas de temps de cinq minutes, les modèles de *machine learning* ont été plus précis. Parmi les cinq modèles, c'est le modèle LSTM qui a le mieux performé.

Aggarwal *et al.* [11] ont comparé les performances d'un réseau de neurones à convolution (CNN) avec celles d'un réseau LSTM et GRU. Parmi les trois, le modèle LSTM est celui qui a le mieux performé, avec la plus faible valeur RMSE.

Felizardo *et al.* [12] ont comparé les modèles ARIMA, *Random Forest*, LSTM, SVM et WaveNets sur plusieurs longueurs de séquences. Pour de la prédiction à court terme, le modèle LSTM est celui qui a apporté les meilleurs résultats.

En janvier 2023, Murray *et al.* [13] ont réalisé une grande étude comparative

entre une multitude de modèles statistiques, et surtout de *machine learning*, sur la prédiction du prix de plusieurs crypto-monnaies. Le modèle LSTM a obtenu la meilleure performance sur les cinq crypto-monnaies testées. Le second modèle est le GRU. Yamak *et al.* [14] ont quant à eux montré que le modèle GRU a mieux performé que le modèle LSTM sur la prédiction du prix journalier du Bitcoin.

Un certain nombre d'autres études ([15],[16],...) tendent à montrer la supériorité des modèles LSTM en prédiction de séries temporelles financières face à d'autres modèles de même ordre de complexité.

Variantes LSTM

L'utilisation de méta-heuristiques permet d'accélérer et d'améliorer l'optimisation des hyperparamètres des réseaux basés sur les LSTM. Ceci est illustré dans de nombreuses publications, comme par exemple dans celle de Sharifi [17], évoquant la méta-heuristique *Chaotic Dolphin Swarm Algorithm*. Kumar & Heider [18] ont comparé quatre méta-heuristiques, dont PSO, et ont montré que cette dernière était celle ayant abouti au modèle le plus optimisé pour la prédiction du marché des actions sur le court terme.

3 Théorie des modèles proposés

Cette section a pour objectif de retracer les fondements théoriques qui permettront d'aboutir aux modèles utilisés à la Section *5 : Implémentation des modèles.*

Notons toutefois que certains concepts de base ne seront abordés que brièvement. En effet, les détails mathématiques de certains de ces principes seront omis, car le but de la Section 3.1 est essentiellement de permettre aux lecteurs d'en avoir une compréhension suffisante, pour pouvoir s'approprier les concepts plus complexes qui suivront.

3.1 Réseaux de neurones artificiels (ANN)

3.1.1 Architecture

Les réseaux de neurones artificiels ont été inspiré des structures cérébrales [19]. L'élément principal qui compose ces réseaux est le neurone artificiel (aussi appelé nœud), qui est la représentation simplifiée d'un neurone biologique. Seul, il n'est capable que d'un traitement basique de l'information. C'est lorsqu'il est combiné à une multitude d'autres neurones artificiels, tous indépendants les uns des autres, que peuvent apparaître des propriétés intéressantes. Cette interconnexion de plusieurs nœuds est qualifiée de *réseau de neurones artificiels.*

La composition d'un réseau de neurones artificiels est constituée de couches, chacune composée de plusieurs nœuds. L'information va se propager à travers le réseau, de l'entrée à la sortie, en subissant de multiples opérations. L'architecture la plus simple d'un réseau de neurones artificiels est illustrée à la Figure 3.1 [1].

Ces couches de neurones peuvent être catégorisées en trois classes principales :

1. Il s'agit ici d'un réseau de type *feedforward* (trad. propagation avant). C'est-à-dire que l'information va linéairement passer de la couche d'entrée à la couche de sortie.

— *Input layers* (trad. couches d'entrée) : Les neurones de ces couches contiennent les données brutes de l'entrée du réseau.
— *Hidden layers* (trad. couches cachées) : Ces couches ont pour but d'appliquer des transformations aux données qui sont propagées à travers le réseau. Elles caractérisent toutes les couches situées entre la couche d'entrée et celle de sortie dans un réseau.
— *Output layer* (trad. couche de sortie) : Les neurones de ces couches contiennent le résultat final de l'application du réseau aux données d'entrée.

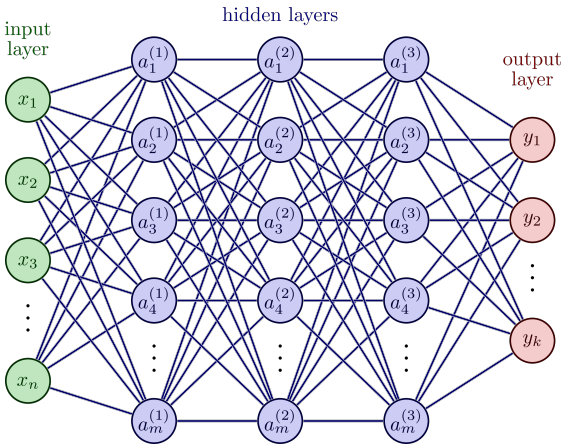


FIGURE 3.1 – Représentation d'un réseau de neurones artificiels simple

Le nombre de neurones par couche peut varier, il n'est notamment pas nécessaire que toutes les *hidden layers* soient de même dimension. Lorsqu'un réseau contient plus de deux *hidden layers*, il est appelé *deep neural network* (trad. réseau de neurones profond).

3.1.2 Fonctionnement

Telle une fonction mathématique, un réseau de neurones n'est autre qu'un outil qui transforme une entrée en une sortie. Nous pouvons donc concevoir un réseau de neurones artificiels comme une tentative de reproduire une fonction inconnue. Pour ce faire, l'entrée va être manipulée lors de sa propagation dans le réseau.

Comme dit précédemment, un neurone seul n'effectue qu'une opération simple. En effet, en notant y la sortie du neurone et x son entrée, l'action du neurone peut se noter comme :

$$y = \sum (x * w) + b$$

où w est le poids associé au neurone et b est son biais.

Les poids w et biais b de tous les neurones qui composent le réseau représentent les paramètres de ce réseau. Lors de l'entraînement du réseau, ce sont ces paramètres que nous allons faire varier afin d'améliorer le résultat final.

Ce traitement élémentaire effectué par les neurones est linéaire, ce qui rend le réseau incapable de reproduire une fonction comportant des non-linéarités, et ce peu importe le nombre de neurones et de couches qui composent le modèle. Pour remédier à cela, la sortie de chaque neurone peut être reliée à une fonction d'activation.

Ces fonctions sont non-linéaires. Les plus couramment utilisées sont décrites ci-dessous :

***Linear Rectified Unit* (ReLU)**

Cette fonction d'activation est la plus simple. La fonction ReLU est définie comme $ReLU(x) = max(0, x)$ et est représentée à la Figure 3.2.

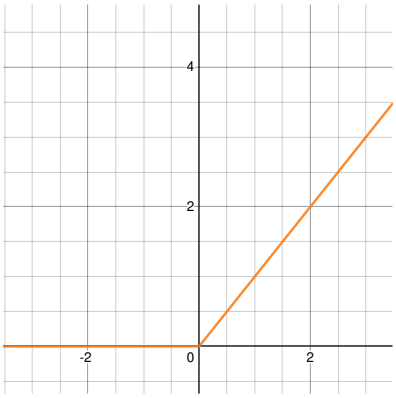


FIGURE 3.2 – Représentation de la fonction ReLU

Ses avantages principaux sont les suivants :

— **Simplicité** : La ReLu est facile à implémenter et nécessite peu de puissance de calcul.
— **Efficacité** : Elle a tendance à améliorer la convergence du gradient stochastique (expliqué à la section 3.2.1) par rapport aux autres fonctions d'activation [20].
— **Sparsité** : Puisque chaque sortie de neurone négative est ramenée à 0, le réseau a tendance à être plus creux, ce qui réduit ses besoins de puissance

de calcul.

En revanche, l'apport de cette sparsité peut aussi engendrer le phénomène des neurones "morts". En effet, tout neurone produisant un résultat négatif voit sa sortie ramenée à 0, ce qui revient donc à désactiver le neurone.

Leaky Rectified Linear Unit **(LeakyReLU)**

Pour remédier au phénomène des neurones "morts" introduit avec la ReLU, la LeakyReLU est définie comme :

$$LeakyReLU(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{else} \end{cases}$$

où a est un paramètre à déterminer, situé sur l'intervalle $]0, 1[$. Si a est trop grand, la LeakyReLU peut ne pas apporter suffisamment de non-linéarité. En revanche, si a est trop petit, cela peut ne pas corriger le phénomène des neurones qui stagnent. Généralement, la valeur de a est choisie aux alentours de 0.01, ce qui donne la représentation de la Figure 3.3.

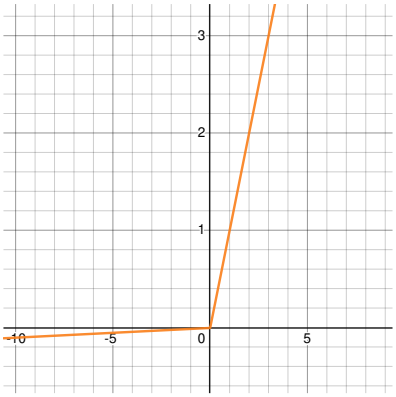


FIGURE 3.3 – Représentation de la fonction LeakyReLU pour $a = 0.01$

Sigmoid (fonction logistique)

Autre fonction d'activation fréquemment utilisée, la fonction sigmoïde est définie comme :

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

Bien que plus gourmande en puissance de calcul, cette fonction d'activation a l'avantage de transformer l'entrée en une sortie comprise entre 0 et 1. Cette fonction est alors souvent utilisée lorsque l'on souhaite interpréter le résultat comme étant une probabilité.

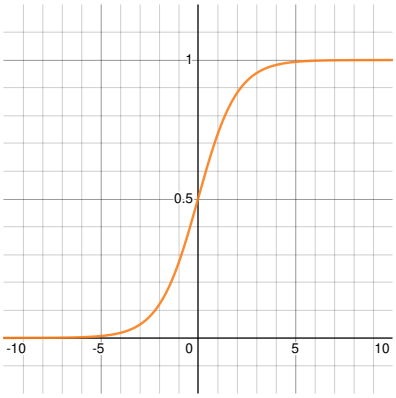


FIGURE 3.4 – Représentation de la fonction sigmoïde

Cependant, le fait que la sortie ne soit pas centrée autour de 0 peut engendrer un décalage des valeurs d'activation, pouvant ralentir l'apprentissage.

Tangente hyperbolique (tanh)

La tangente hyperbolique permet justement de centrer les sorties autour de 0, en renvoyant toute entrée sur la plage $]-1, 1[$. Elle est définie comme :

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Tout comme la sigmoïde, la tangente hyperbolique demande plus de ressources de calcul.

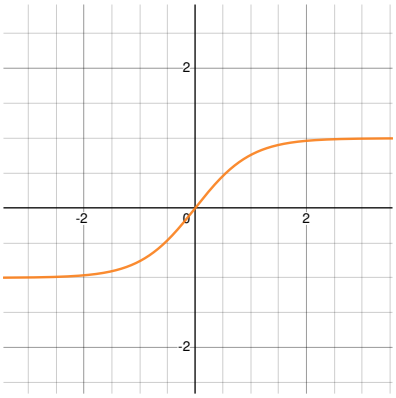


FIGURE 3.5 – Représentation de la fonction tangente hyperbolique

Softmax

La fonction *softmax* a pour effet de transformer un vecteur de valeurs réelles en entrée en un vecteur de probabilités, dont la somme vaut 1. Avec un vecteur d'entrée $\mathbf{x}$ de K valeurs réelles, l'élément j du vecteur de probabilités de sortie de la fonction *softmax* s'écrit comme :

$$softmax(\mathbf{x})_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, \qquad \forall j \in [1, ..., K]$$

Par exemple, sur le vecteur $\mathbf{x} = \{2, 3.5, 2, 1.5, 4, 3\}$, l'application de *softmax* donne le vecteur (valeurs arrondies à 10^{-2}) : $\{0.06, 0.26, 0.06, 0.04, 0.42, 0.16\}$. La somme vaut bien 1, et les valeurs illustrent le poids relatif de chaque valeur dans l'espace de son vecteur.

3.1.3 Entraînement

Après avoir déterminé l'architecture d'un modèle et obtenu une sortie $\hat{y}$ suite à la propagation avant, il convient de l'entraîner. L'objectif de la période d'apprentissage d'un réseau de neurones est l'ajustement du réseau, en agissant de manière itérative sur tous ses paramètres.

Les fonctions d'erreur

Dans le domaine des réseaux de neurones, l'entraînement peut être décrit comme la minimisation d'une fonction objectif, ici appelée "fonction d'erreur".

Parmi les nombreuses fonctions de mesure de l'erreur existantes, les plus utilisées sont reprises à la Table 3.1. Dans cette table, $\hat{\mathbf{y}}$ représente les prédictions, $\mathbf{y}$ représente les vraies valeurs et n est le nombre de prédictions.

Nom	Définition	Formule
MAE	Moyenne des différences absolues	$\frac{1}{n}\sum_{i=1}^{n}\lvert\hat{y}_i - y_i\rvert$
MSE	Erreur quadratique moyenne	$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$
RMSE	Erreur quadratique moyenne racine	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$

TABLE 3.1 – Fonctions d'erreur les plus utilisées [14]

L'avantage de la valeur du RMSE est sa plus grande facilité d'interprétation, du fait que cette valeur est de la même unité que les prédictions. C'est généralement la valeur la plus utilisée pour qualifier l'efficacité d'un modèle. Cette valeur sera donc majoritairement utilisée dans ce mémoire.

3.1.4 *Backpropagation*

L'erreur associée à une prédiction $\hat{y}$ (issue de la propagation avant dans le réseau) et à une valeur cible y permet ensuite de pratiquer la propagation arrière (ang. *backpropagation*). C'est lors de ce processus que tous les paramètres qui composent le réseau (les poids et les biais) vont être modifiés. Le but de la *backpropagation* est de calculer la direction dans laquelle ces paramètres doivent évoluer, afin que l'erreur diminue.

Pour ce faire, le gradient de l'erreur va être calculé et les paramètres de la dernière couche vont être modifiés en conséquence. Le calcul du gradient va ainsi être propagé dans les couches précédentes, depuis la fin (d'où le terme de *backpropagation*).

À chaque étape, le but est donc de décrire mathématiquement l'impact de chaque paramètre sur la fonction d'erreur. Ceci est réalisé à l'aide de la dérivée partielle du terme d'erreur en fonction du paramètre que l'on souhaite modifier. C'est donc pour cela que la *backpropagation* se réalise de la sortie vers l'entrée, car il est nécessaire d'appliquer la *chain rule* (trad. règle de la chaîne) pour définir les dérivées partielles par rapport aux paramètres plus éloignés du terme d'erreur de la sortie du réseau.

En procédant de la sorte de manière itérative à chaque nouvelle prédiction, il est espéré que les poids et les biais du modèle se dirigent peu à peu vers la valeur qui permettra au réseau de minimiser le terme d'erreur.

3.2 *Recurrent Neural Network* (RNN)

Le modèle de réseau de neurones illustré à la Figure 3.1 ne tient compte d'aucune notion de temps, ce qui le rend non pertinent dans le cadre des prédictions de séries temporelles. C'est pour faire face à cette limite que les réseaux de neurones récurrents (RNN) ont été créés. La nouveauté de ces modèles est leur capacité de mémoire, nécessaire dans le traitement de données séquentielles. Ce type de modèle est aussi largement utilisé dans des tâches comme le traitement du langage naturel.

En effet, pour pouvoir prédire des séries temporelles ou du langage, le réseau de neurones doit disposer d'informations sur les éléments passés, c'est-à-dire posséder une forme de mémoire. Ainsi, l'intérêt du modèle RNN est de permettre un apprentissage adapté en cas de données d'entrée séquentielles.

La Figure 3.6 illustre l'architecture de base d'un RNN [21], où x_t est l'entrée du modèle au pas de temps t, h_t l'état caché (ang. *hidden state*) et $\hat{y}_t$ la sortie.

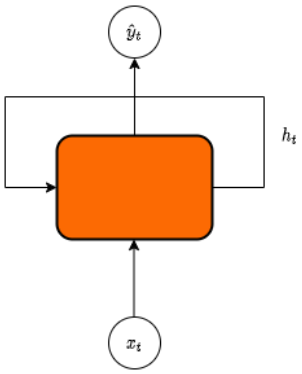


FIGURE 3.6 – Architecture d'un réseau RNN de base, version repliée

À l'intérieur de chaque cellule, la sortie du pas de temps actuel (l'*hidden state*, h_t) est calculée sur base des données d'entrée actuelles (x_t) et de l'*hidden state* du pas de temps précédent (h_{t-1}). Ce processus est visible sur la version dépliée du réseau (Figure 3.7).

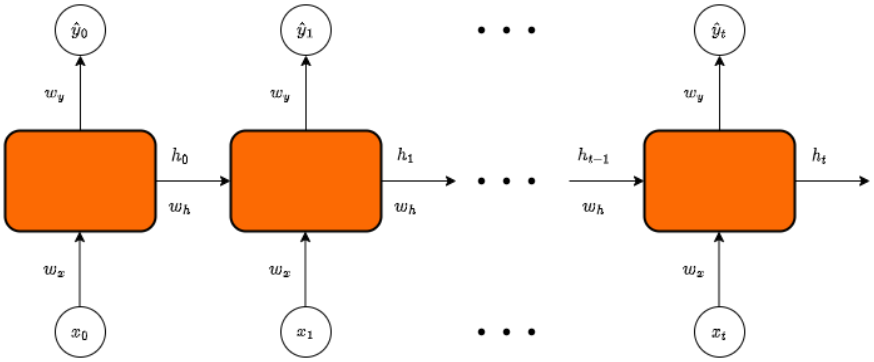


FIGURE 3.7 – Architecture d'un réseau RNN de base, version dépliée

Deux équations mathématiques permettent la propagation à travers le RNN (dont l'architecture interne d'une cellule est illustrée à la Figure 3.8) :

$$h_t = tanh(w_h h_{t-1} + w_x x_t)$$
$$\hat{y}_t = w_y h_t$$

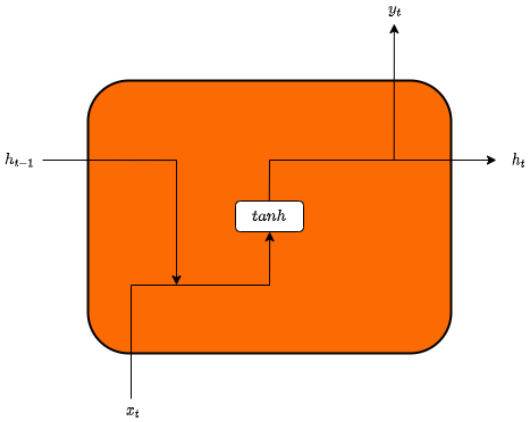


FIGURE 3.8 – Architecture interne d'une cellule RNN de base

Les paramètres w_h , w_x et w_y sont les poids du RNN. Leur apprentissage est réalisé via une *backpropagation* sur la version déroulée du réseau. Une autre particularité des réseaux RNN est que les mêmes poids sont utilisés à chaque pas de temps, c'est-à-dire dans chaque cellule. Seuls x_t, h_t et y_t évoluent au fur et à mesure du temps.

Un problème potentiel dans les réseaux appliquant la *backpropagation* est le risque de disparition/explosion du gradient. Cette notion est abordée à la section suivante.

3.2.1 Disparition/Explosion du gradient

La disparition du gradient est un problème qui peut survenir durant la période d'entraînement d'un modèle [22].

Lors du calcul du gradient, les termes calculés peuvent devenir petits au point d'empêcher les poids de changer de valeur, voire d'interrompre le processus d'apprentissage. Ceci engendre en effet une stagnation des paramètres.

Cet impact n'est néanmoins pas le même pour toutes les fonctions d'activation. Par exemple, les valeurs peuvent devenir très petites dans des fonctions d'activation qui génèrent un résultat entre 0 et 1, comme la fonction logistique ou la tangente hyperbolique. Ainsi, les différentes valeurs à travers les couches peuvent être fortement diminuées en multipliant de manière successive par ces petits facteurs. Un exemple contraire est la ReLU, qui peut donner des valeurs plus élevées et est donc moins sensible au phénomène de disparition du gradient. Dans le cas présent, c'est la multiplication successive par des facteurs supérieurs à 1 qui pose problème et qui engendrer l'explosion du gradient.

C'est dans le but de limiter cet effet de disparition/explosion du gradient que les cellules GRU et LSTM ont été créées.

3.3 *Long Short-Term Memory* (LSTM)

Les réseaux Long Short-Term Memory sont une forme de RNN capables d'apprendre et de se remémorer de longues séquences d'entrées, via des portes contrôlant l'affluence d'information qui parcoure le réseau.

Les modèles LSTM permettent de solutionner divers problèmes survenant dans les RNN :

— La mémoire à court terme : En avançant dans la séquence d'entrée, correspondant à l'avancement dans le temps, certaines informations, parfois importantes, se perdent par diminution successive. Les LSTM, grâce à leurs portes qui pondèrent les différents signaux, sont capables de conserver de l'information relative à des pas de temps plus anciens.

— La disparition du gradient : Comme nous l'avons évoqué dans la section à ce sujet, le problème de disparition du gradient survient quand, au cours de la progression dans les couches en *backpropagation*, le gradient devient trop petit, au point de ne plus permettre la moindre modification des poids.
— L'explosion du gradient : Ce problème peut se produire quand les poids reçoivent une importance trop élevée durant la phase d'apprentissage.

L'architecture d'un LSTM est semblable à celle d'un RNN, hormis le fait que différentes portes vont cette fois permettre à la cellule LSTM de mieux contrôler le passage des différents signaux d'information [23]. Ces portes sont composées d'une multitude de paramètres, ce qui offre la possibilité aux LSTM d'apprendre plus de complexités qu'un RNN classique. La Figure 3.9 représente l'architecture d'une cellule LSTM.

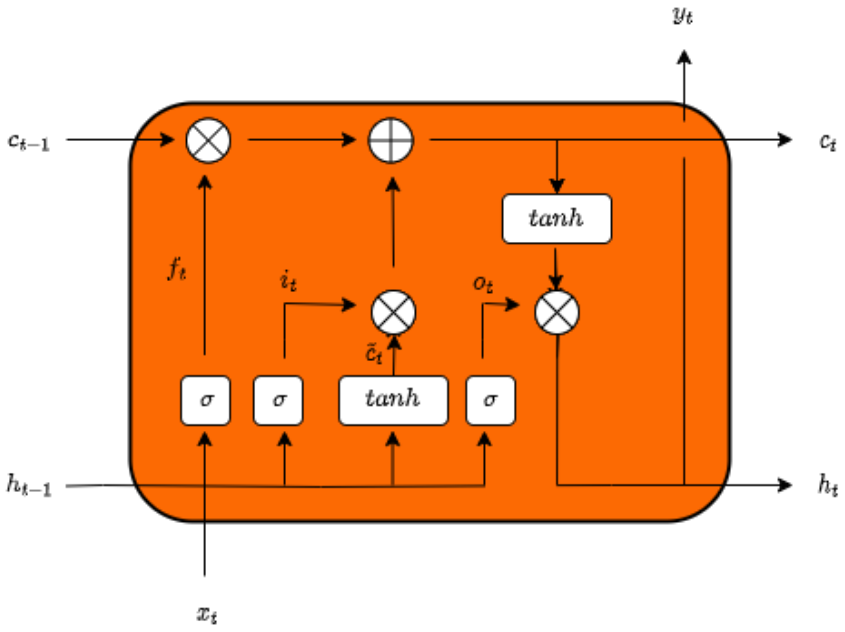


FIGURE 3.9 – Architecture interne d'une cellule LSTM

Dans cette cellule LSTM, nous pouvons retrouver des *hidden states* d'entrée et sortie (h_{t-1} et h_t), ainsi que des *cell states* d'entrée et sortie (c_{t-1} et c_t). Les données correspondant au pas de temps actuel dans la séquence reçue par le LSTM sont notées x_t. Ensemble, h_{t-1}, c_{t-1} et x_t servent à calculer la sortie prédite ($\hat{y}_t$).

Le flux d'information à travers une cellule LSTM est régi par ces équations :

$$i_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + b_i) \tag{3.1}$$
$$o_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + b_o) \tag{3.2}$$
$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t-1} + b_f) \tag{3.3}$$
$$\tilde{c}_t = tanh(w_{xc}x_t + w_{hc}h_{t-1} + b_c) \tag{3.4}$$
$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \tag{3.5}$$
$$h_t = o_t \cdot tanh(c_t) \tag{3.6}$$

où tous les w sont les poids associés aux différentes liaisons dans la cellule, et les b sont les biais de chaque porte.

Afin de mieux comprendre le fonctionnement d'une cellule LSTM, il nous semble judicieux de décomposer les différentes portes qu'elle contient :

3.3.1 Porte d'oubli

La première étape est de déterminer quelle proportion de la mémoire à court terme (c_{t-1}) est à oublier (d'où le nom de *forget gate*). L'équation 3.3 permet, à l'aide de la sigmoïde, de fournir un coefficient f_t situé entre 0 et 1, déterminant cette proportion à oublier. Le terme $f_t \cdot c_{t-1}$ de l'équation 3.5 permet par la suite d'appliquer ce coefficient.

3.3.2 Porte d'entrée

Ensuite, il est question de décider quelle nouvelle information est à ajouter au *cell state* (c_t). Ceci se déroule dans l'*input gate*. Les équations 3.1 et 3.4 permettent de calculer cette nouvelle information ($\tilde{c}_t$) et, ensuite, le terme $i_t \cdot \tilde{c}_t$ de l'équation 3.5 permet d'ajouter l'information au *cell state*.

3.3.3 Porte de sortie

Finalement, sur base des *cell* et *hidden states* précédents (c_{t-1} et h_{t-1}), ainsi que sur base de l'entrée au pas de temps actuel (x_t), il est temps de construire la sortie de la cellule ($h_t = y_t$). Ce sont les équations 3.2 et 3.6 qui déterminent finalement cette valeur.

3.4 *Gated Recurrent Unit* (GRU)

Bien qu'elles soient semblables aux cellules LSTM, les cellules de type GRU disposent de moins de portes, et par conséquent de moins de paramètres. La

structure interne d'une cellule GRU est illustrée à la Figure 3.10.

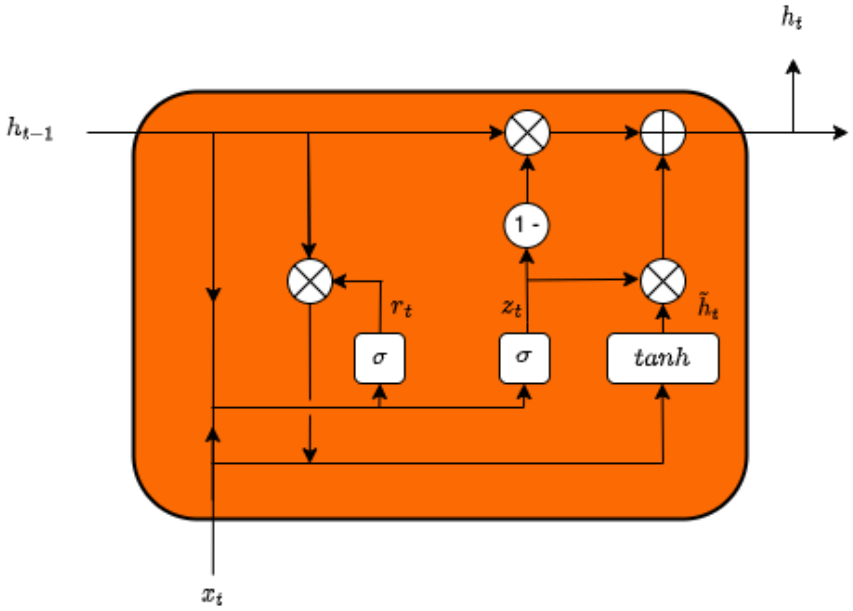


FIGURE 3.10 – Architecture interne d'une cellule GRU

Les équations qui régissent le passage de l'information à travers la cellule sont :

$$r_t = \sigma(w_{xr}x_t + w_{hr}h_{t-1} + b_r) \tag{3.7}$$
$$z_t = \sigma(w_{xz}x_t + w_{hz}h_{t-1} + b_z) \tag{3.8}$$
$$\tilde{h}_t = tanh(w_{xh}x_t + w_{rh}(r_t \cdot h_{t-1})) \tag{3.9}$$
$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \tag{3.10}$$

Par rapport à une cellule LSTM, les portes d'oubli et d'entrée de la GRU ont fusionné en une unique porte de mise à jour (ang. *update gate*). Les *cell states* et *hidden states* des LSTM ont également fusionné.

3.5 Mécanisme d'attention

Le principe de l'attention consiste en l'attribution d'un degré d'importance différent selon les éléments. Le mécanisme d'attention peut être temporel. Dans ce cas, ce sont les différents pas de temps d'une séquence qui ne seront pas traités

avec la même importance.

Toutefois, l'attention peut aussi servir à pondérer les *features* d'entrée [24]. C'est de cette manière que va être utilisé le mécanisme d'attention dans ce mémoire.

Comme il sera expliqué à la Section 4.2, le modèle reçoit en entrée 25 *features* différentes par pas de temps. Lors de l'entraînement, le réseau pourra apprendre comment pondérer ces *features* afin de minimiser l'erreur finale de prédiction.

Concrètement, les données d'entrée (x_t), l'*hidden state* d'entrée (h_t) et le *cell state* d'entrée, vont être concaténées ensemble. Ensuite, ce nouveau vecteur passera dans une couche linéaire dense (la description précise des couches qui composent un réseau de neurones se trouve à la Section 5.1.1).

La partie du vecteur concaténé représentant les x_t initiaux va ensuite être passée dans la fonction *softmax*.

L'interprétation de ce processus est que les *features* de l'entrée x_t vont être gardées en tenant compte de leur importance face aux autres *features* et face à h_t et c_t.

3.6 *Skip connections*

Malgré que les cellules de type LSTM soient plus robustes face aux problèmes de disparition/explosion du gradient, elles n'en sont pas pour autant immunisées. Pour tenter de contrer encore un peu plus ce phénomène, il est possible d'avoir recours à une architecture implémentant le mécanisme de *skip connections* (trad. sauter des connexions).

Le but est de fournir aux cellules LSTM en entrée non pas uniquement le *hidden state* du pas de temps précédent (h_{t-1}), mais bien la combinaison linéaire :

$$\tilde{h}_{t-1} = \lambda h_{t-1} + (\lambda - 1) h_{t-k} \tag{3.11}$$

où λ est un réel compris entre 0 et 1, et h_{t-k} est le *hidden state* de k pas de temps précédents, $k \in \mathbb{N}$.

L'illustration de la cellule modifiée se trouve à la Figure 3.11.

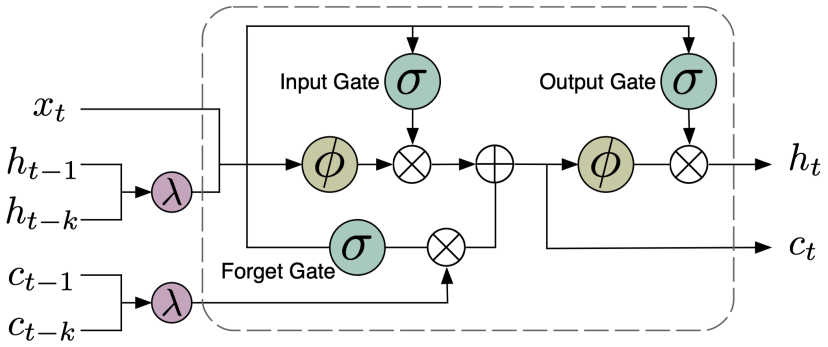


FIGURE 3.11 – Architecture de la cellule modifiée pour implémenter le mécanisme de *skip connections*. Illustration tirée de [25].

Ceci offre donc un chemin plus direct vers les pas de temps plus anciens, ce qui permet au gradient d'être moins impacté dans son amplitude lors de la *backpropagation* [25]. Au niveau des équations, il suffit alors de remplacer h_{t-1}, dans les équations 3.1 à 3.6, par $\tilde{h}_{t-1}$, défini à la Section 3.11.

Ce mécanisme a notamment permis d'améliorer la capacité de généralisation [2] d'un modèle prédictif en modélisation du langage [3] (ang. *language modeling*) [25].

2. La capacité de généralisation d'un modèle peut être calculée via le score de perplexité.

3. Le *language modeling* est la discipline dans laquelle le but est de prédire les prochains mots (ou plus généralement *tokens*) sur base de ceux précédents.

4 Analyse et préparation des données

Les données fournies par Kaggle concernent 14 crypto-monnaies différentes. L'objectif de la compétition qui nous occupe est de prédire, pour chaque pas de temps, le retour potentiel individuel de ces 14 crypto-monnaies sur un horizon de 15 minutes. Pour tenter d'obtenir des résultats intéressants, il est conseillé d'entraîner séparément chaque modèle, personnalisé pour chaque crypto-monnaie individuellement.

Dans notre jeu de données, il y a en moyenne près de 1.75 millions de données par crypto-monnaie, pour un total de 24.3 lignes de données, chacune composée de 10 valeurs différentes (les *features*).

Dupliquer 14 fois la même procédure de traitement des données et d'entraînement aurait demandé un temps excessivement long et des ressources en calcul trop importantes. Dès lors, puisque, pour rappel, ce mémoire ne s'inscrit pas dans le cadre d'une participation officielle à la compétition (celle-ci étant clôturée, cfr. Section *1 : Contexte – Compétition Kaggle*), nous avons choisi de nous concentrer sur trois de ces crypto-monnaies. En effet, cette décision permet d'accorder ce temps pour augmenter la qualité des modèles, en travaillant sur des architectures plus innovantes et complexes qu'il n'aurait pas été possible d'expérimenter en ayant à entraîner 14 modèles différents, plutôt que la quantité.

En ce qui concerne le choix de ces crypto-monnaies, deux d'entre elles ont été sélectionnées par défaut : le Bitcoin (BTC) et l'Ethereum (ETH). En effet, ces deux crypto-monnaies dominent le marché, en représentant à elles seules respectivement 499 et 191 milliard de dollars de capitalisation [1]. En comparaison, la troisième crypto-monnaie volatile dans le classement est le Binance Coin (BNB), avec seulement 32 milliards de dollars de capitalisation.

1. Données actualisées pour la dernière fois le 18 août 2023, via `https://coinmarketcap.com/fr/`.

Comme troisième crypto-monnaie, il a été décidé d'utiliser le Litecoin (LTC). En effet, le Litecoin fait partie des plus anciennes crypto-monnaies (lancée en octobre 2011 [26]) et a été dans le top 10 des crypto-monnaies les plus capitalisées durant la majorité de son existence. Cependant, en date du 18 août 2023, elle est descendue à la 15ème position. De plus, le Litecoin est moins volatil que la plupart des autres crypto-monnaies proposées dans les données, pour la plupart bien plus récentes.

Ce lien important avec les deux autres crypto-monnaies choisies (le Bitcoin et l'Ethereum) se traduit d'ailleurs dans le matrice de corrélation présentée à la Figure 4.1. La corrélation illustrée est la corrélation de Pearson, qui sera expliquée en détails à la Section 6.2.3.

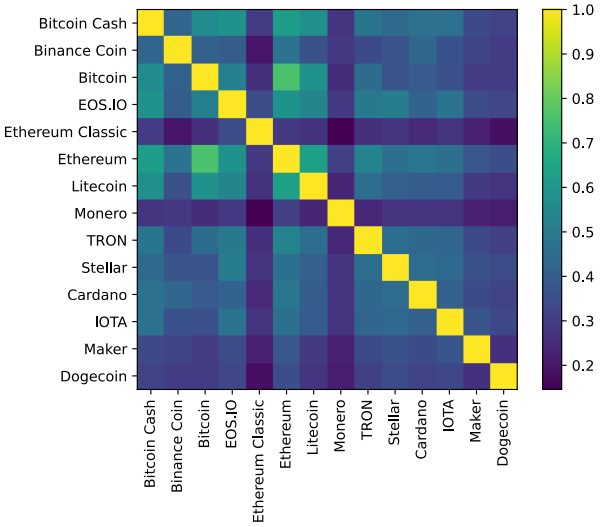


FIGURE 4.1 – Matrice de corrélation entre les 14 crypto-monnaies

Toutes ces raisons nous ont conduit à sélectionner le Litecoin (LTC) comme troisième crypto-monnaie analysée.

4.1 Exploration

Une ligne de données pour une crypto-monnaie sur un intervalle d'une minute est composée des données suivantes (les prix étant tous en dollars américains) :

— `timestamp` : *timestamp* en secondes représentant le début de la minute concernée par la ligne de données
— `Asset_ID` : l'identifiant de la crypto-monnaie concernée par cette ligne (allant de 0 à 13, 1 pour le Bitcoin, 6 pour l'Ethereum et 9 pour le Litecoin)

— `Count` : le nombre de d'échanges qu'il y a eu sur le marché de la crypto-monnaie face au dollar sur l'intervalle d'une minute
— `Open` : le prix au début de la minute
— `High` : le plus haut prix atteint lors de la minute
— `Low` : le plus bas prix atteint lors de la minute
— `Close` : le prix à la fin de la minute
— `Volume` : le nombre d'unités de la crypto-monnaie échangées lors de la minute
— `VWAP` : le prix moyen pondéré en fonction du volume échangé lors de la minute
— `Target` : représente la rentabilité résiduelle sur un horizon de 15 minutes

L'objectif décrit dans cette compétition Kaggle est de prédire correctement la valeur `Target`. La rentabilité résiduelle d'un actif financier peut être vue comme la rentabilité de l'actif qui n'est pas décrite par l'évolution d'un *benchmark*. Ici, le *benchmark* utilisé est l'évolution globale des crypto-monnaies. En effet, si sur un horizon de 15 minutes, l'entièreté du marché des crypto-monnaies prend en moyenne 0.2% de valeur, alors, un actif qui a pris exactement 0.2% de valeur aura une rentabilité résiduelle de 0. En revanche, même s'il a pris 0.1% de valeur, sa rentabilité résiduelle sera bien négative et vaudra -0.1, car l'actif aura performé moins bien que ce que l'on aurait pu en attendre en connaissant la situation globale du marché.

Les données sont construites de sorte à ce que la valeur `Target` à la ligne i corresponde à la valeur que le modèle doit prédire lorsqu'il a connaissance des données de la ligne i et des lignes précédentes.

Procédons à quelques observations préliminaires. Les données débutent le 01/01/2018 et s'étendent jusqu'au 21/09/2021. Les données financières *Open*, *High*, *Low* et *Close*, souvent notées OHLC, peuvent facilement être observées via un graphe en bougies japonaises. Voici, à la Figure 4.2, l'exemple sur un échantillon des données du Bitcoin de ce type de graphe.

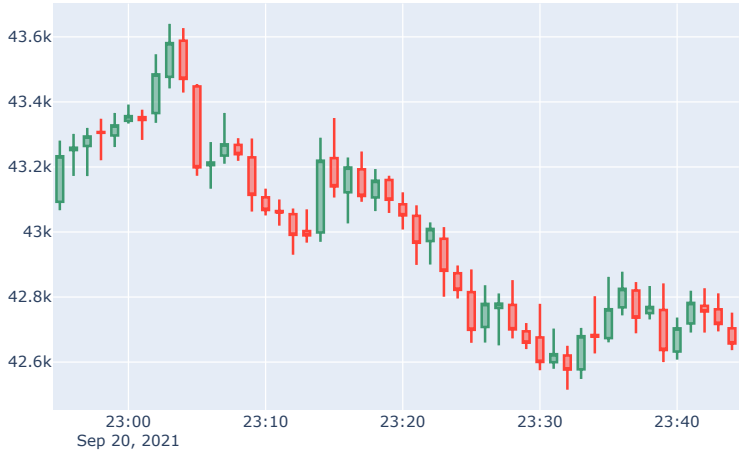


FIGURE 4.2 – Échantillon du prix du Bitcoin représenté sur un graphe en bougies japonaises

Une bougie japonaise s'interprète comme ceci [27] :

- **Le corps** : Le corps de la bougie (partie plus large) indique la différence entre le prix d'ouverture et le prix de clôture.
- **La couleur** : Si la bougie est rouge, cela signifie que le prix d'ouverture était plus élevé que le prix de clôture du pas de temps. Le prix d'ouverture peut donc être retrouvé en regardant la limite haute du corps de la bougie. Si la bougie est verte, le raisonnement est inverse.
- **Les mèches** : Les mèches de la bougie (parties fines) représentent les *extrema* atteints sur le pas de temps.

Ce type de graphes n'est pas lisible sur un large spectre temporel avec des pas de temps si petits. Voici donc, à la Figure 4.3, l'évolution des prix de clôture des trois crypto-monnaies sélectionnées sur toute la durée contenue dans les données.

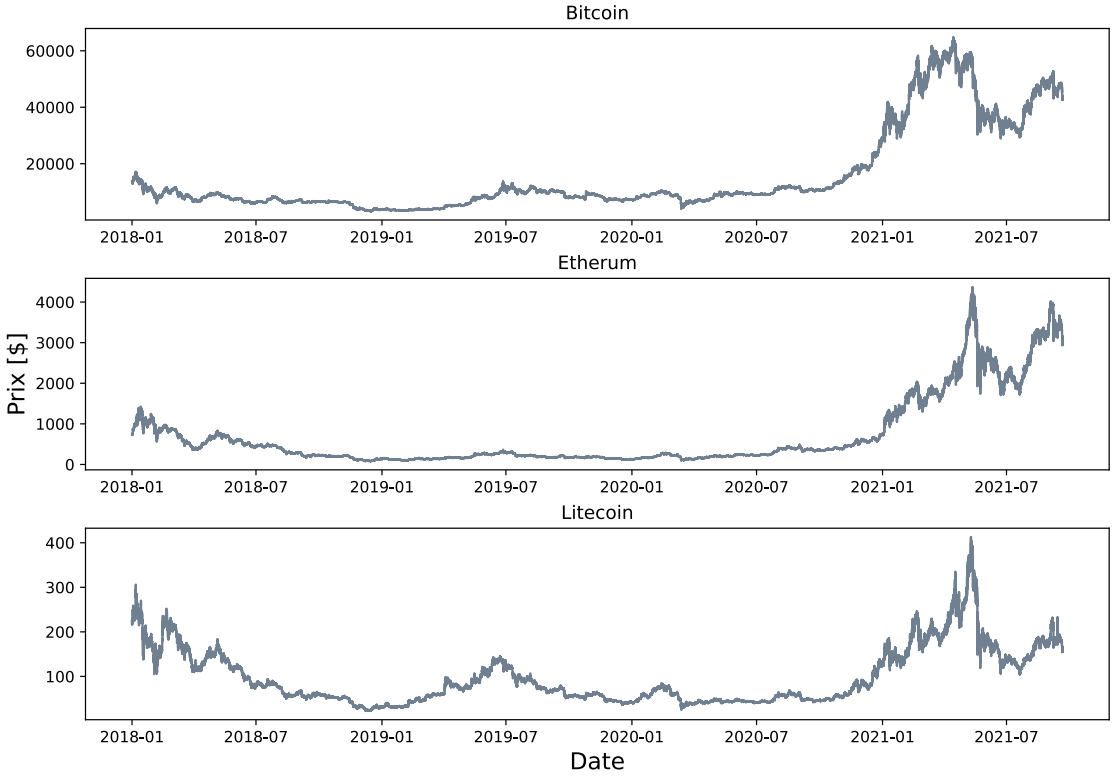


FIGURE 4.3 – Évolution des prix du Bitcoin, de l'Ethereum et du Litecoin, du 01/01/2018 au 21/09/2021

4.1.1 Stationnarité

Une série temporelle est dite stationnaire si [28] :

— l'espérance est constante au cours du temps ;
— la variance est constante au cours du temps et est finie ;
— l'auto-corrélation ne dépend que de la valeur du *lag* (trad. décalage), et non du temps.

Ces conditions peuvent s'interpréter comme le besoin que la structure de fond de la série temporelle ne change pas au cours du temps. Par exemple, le bruit blanc est une série temporelle purement stationnaire.

Un modèle de prédiction peut souffrir du manque de stationnarité dans les données. En effet, lors de l'entraînement sur une période de temps, on attend d'un modèle qu'il puisse comprendre la structure de fond des séries temporelles qu'il

traite en entrée. Si cette structure est dépendante du temps, elle sera différente sur la période d'application/de test de celle apprise sur la période d'entraînement.

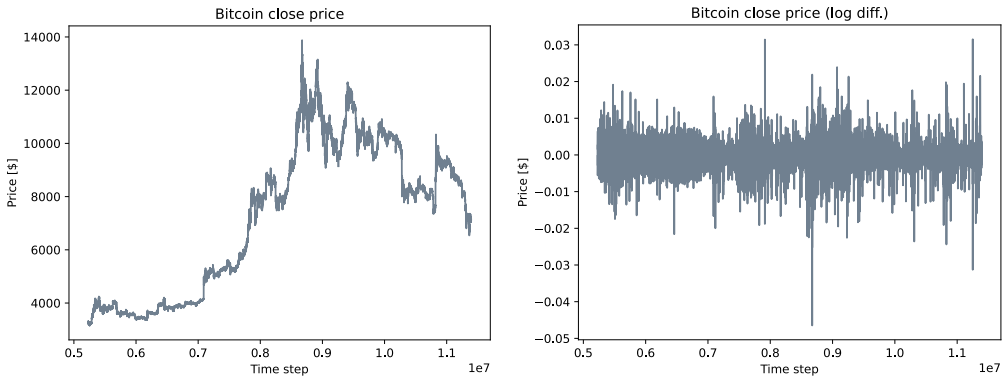
Il est néanmoins possible d'appliquer des transformations à une série temporelle non-stationnaire afin de la rendre stationnaire. En finance, la différenciation est souvent utilisée, à laquelle on applique le logarithme.

Il est assez évident, en voyant la figure 4.3, que l'évolution du prix des crypto-monnaies n'est pas stationnaire. Il existe un test statistique, appelé test augmenté de Dickey-Fuller (ADF), qui permet de tester cette propriété. Les résultats de ce test sont visibles à la Table 4.1.

Feature	**Données brutes**	**Données log-différenciées**
`Count`	Stationnaire	-
`Open`	Non-stationnaire	Stationnaire
`High`	Non-stationnaire	Stationnaire
`Low`	Non-stationnaire	Stationnaire
`Close`	Non-stationnaire	Stationnaire
`Volume`	Stationnaire	-
`VWAP`	Non-stationnaire	Stationnaire

TABLE 4.1 – Résultat du test ADF avant et après transformation des données, pour un seuil de 5%

Il est possible d'appliquer la méthode du logarithme des différences sur les *features* non-stationnaires et de refaire le test ADF pour vérifier que les données transformées sont bien stationnaires. Les résultats sont résumés à la Table 4.1. La visualisation de l'effet de cette transformation sur le prix de clôture du Bitcoin est visible à la Figure 4.4.



(a) Données brutes (b) Différence des logarithmes

FIGURE 4.4 – Illustration de la transformation d'une série temporelle pour la rendre stationnaire

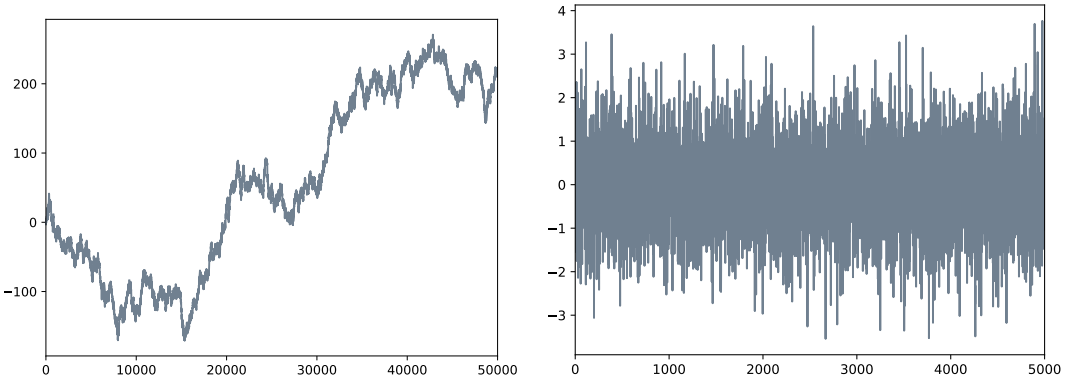
4.1.2 Marche aléatoire

En voyant la Figure 4.5b, il est tout à fait justifié de se demander si le prix ne répond pas à l'hypothèse de la marche aléatoire. La *théorie de la marche aléatoire* décrit une évolution temporelle dans laquelle la série évolue de manière tout à fait aléatoire, en partance de la valeur précédente. L'évolution est décrite comme :

$$x_{t+1} = x_t + \epsilon_t$$

où ϵ_t est tiré d'une probabilité normale centrée en 0.

La visualisation de la réalisation d'une marche aléatoire et sa transformation par log. diff. est visible à la Figure 4.5



(a) Marche aléatoire (b) Différence des logarithmes, bruit blanc

FIGURE 4.5 – Illustration d'une marche aléatoire avec le bruit blanc obtenu par transformation log. diff.

Pour tester si le prix de clôture du Bitcoin est une marche purement aléatoire ou non, il est possible d'observer les graphes d'auto-corrélation. Ces graphes ACF (*Autocorrelation Function*) et PACF (*Partial Autocorrelation Function*) peuvent informer sur la corrélation observée entre une valeur et sa version décalée [29]. Les graphes ACF et PACF du Bitcoin, ainsi que celui du bruit blanc, sont visibles à la Figure 4.6.

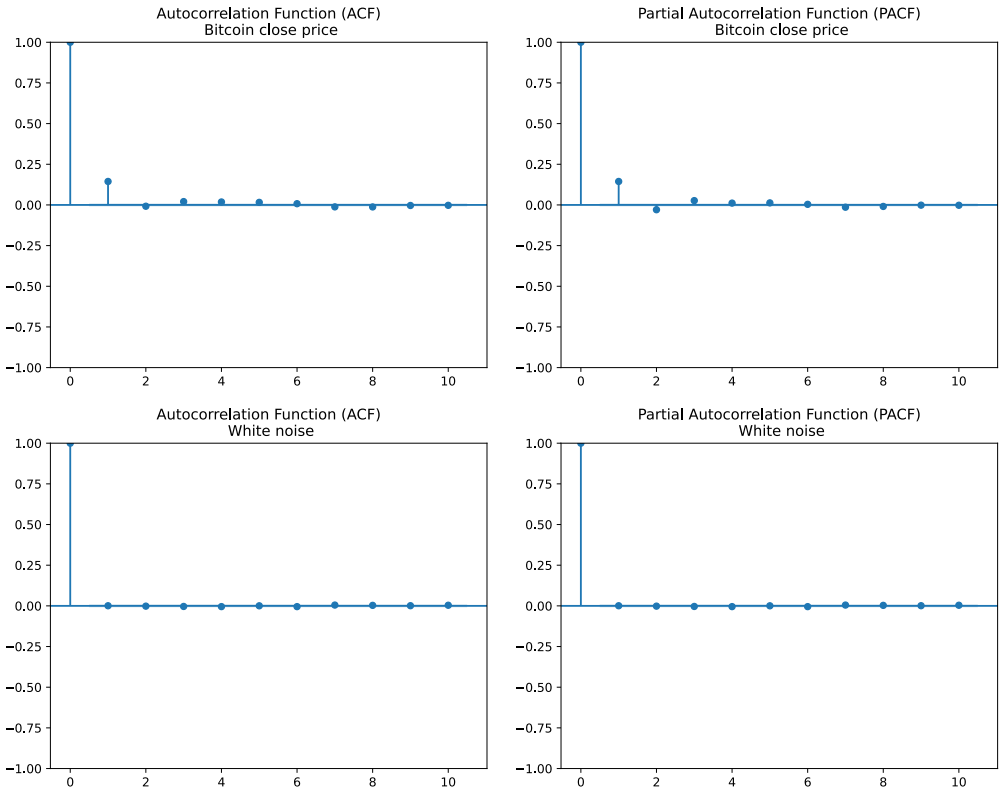


FIGURE 4.6 – Comparaison des graphes ACF et PACF entre le prix de clôture du Bitcoin et une marche aléatoire

L'élément permettant d'affirmer que le prix de clôture n'est pas une marche purement aléatoire est qu'il existe des décalages donnant lieu à des valeurs significatives, à la fois sur le graphe ACF et sur le graphe PACF (comme par exemple le décalage de 1, 2 et 3). Ceci n'est pas le cas pour la marche aléatoire.

4.2 Extraction de données

Une des particularités de cette compétition Kaggle est qu'elle requiert que le *notebook* utilisé ait la connexion internet désactivée. Dès lors, seules les données fournies à la base sont accessibles, et une stratégie reposant sur des sources d'informations externes (comme par exemple Mohanty *et al.*, [15] qui utilisent le sentiment de marché via les réseaux sociaux) est impossible.

Inspiré par Pan [30], ce sont alors des indicateurs techniques qui ont été extraits. En revanche, chaque indicateur pouvant être appliqué aux quatre données *Open*,

High, *Low* et *Close*, en ajouter trop rendrait la dimension des entrées du modèle trop importante quant au phénomène du *curse of dimensionality* (trad. fléau de le dimension) [31].

Finalement, quatre indicateurs financiers ont été extraits (ajoutant donc 16 valeurs au vecteur d'entrée), et deux indicateurs temporels, pour un total de 18 *features* supplémentaires.

- Indicateurs financiers (appliqués aux 4 valeurs OHLC)
  - ***Log return*** : pourcentage de changement d'un pas de temps par rapport au précédent, auquel est appliqué le logarithme
  - **MA5** et **MA15** : moyenne mobile sur une fenêtre de 5 et 15 pas de temps, respectivement
  - **EMA15** : moyenne mobile exponentielle sur une fenêtre de 15 pas de temps
- Indicateurs temporels
  - `is_weekend` : binaire permettant d'indiquer si le pas de temps considéré a eu lieu un week-end ou non
  - `day_of_week` : nombre compris entre 0 et 7 permettant d'indiquer le jour exact durant lequel le pas de temps a eu lieu

L'illustration des indicateurs financiers se trouve à la Figure 4.7. Utiliser des indicateurs mobiles (reprenant de l'information d'une fenêtre de pas de temps) permet de diffuser l'information contenue dans les données d'un pas de temps [32], tout en respectant un principe primordial qui est de ne pas faire fuiter l'information du futur (ang. *data leakage*).

Lorsque l'objectif est de construire un modèle doté de capacités de prédiction, il faut veiller à ne pas fournir, à un pas de temps t, de l'information contenue dans des pas de temps futurs à t. En effet, cela biaiserait le modèle qui ne pourrait jamais, en conditions réelles, obtenir ces informations pour sa prédiction.

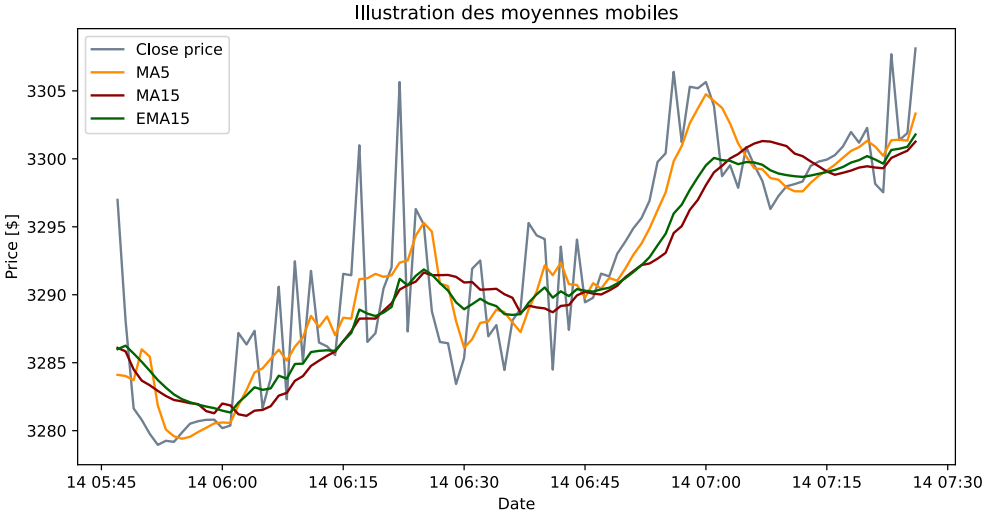


FIGURE 4.7 – Illustration des indicateurs financiers sur un échantillon du prix de clôture du Bitcoin

Puisque les indicateurs financiers ajoutés ne sont pas stationnaires, il a été appliqué la même transformation que décrite précédemment, afin d'obtenir des séries temporelles stationnaires.

4.3 Mise à l'échelle des données

La dernière étape de préparation des données avant l'entraînement des modèles est la mise à l'échelle. En effet, les réseaux de neurones, dont les réseaux LSTM, souffrent de la différence d'échelle entre les *features*. Il est alors primordial d'appliquer une mise à l'échelle à toutes les données.

Pour ce faire, il existe deux solutions principales :

— ***MinMaxScaler*** : permet de mettre à l'échelle toutes les valeurs d'une *feature* en fonction de ses valeurs minimale et maximale. Après la mise à l'échelle, toutes les données sont comprises entre 0 et 1.

— ***StandardScaler*** : transforme une *feature* afin de lui apporter les propriétés d'une moyenne de 0 et d'un écart-type de 1. Cette mise à l'échelle suppose une distribution gaussienne.

L'avantage du *MinMaxScaler* est qu'il ne modifie pas la distribution des données. Mais il a le grand désavantage d'être très sensible aux valeurs aberrantes présentes dans le jeu de données, là où le *StandardScaler* est plus robuste. Puisque les données ne contiennent aucune donnée aberrante, il a été décidé d'appliquer le

MinMaxScaler afin de conserver les distributions initiales, tout en mettant sur la même échelle (entre 0 et 1) chaque *feature*.

5 Implémentation des modèles

5.1 Détails des architectures

Pour chacun des modèles présentés ci-dessous, un certain nombre d'hyperparamètres sont à définir. Un *hyperparamètre* est un paramètre qui va influencer l'architecture d'un modèle ou sa méthode d'entraînement (et donc finalement sa performance). Les hyperparamètres influençant l'architecture spécifique des modèles seront présentés à leur section respective, ci-dessous. Nous allons commencer par aborder les principaux hyperparamètres communs à tous les modèles :

- ***Optimizer*** : Un *optimizer* permet de ne pas avoir à calculer l'entièreté de la fonction de perte lors de l'entraînement. Il en existe deux principaux, Adam et RMSProp, qui se basent tous les deux sur le principe de la descente de gradient avec une impulsion (ang. *momentum*), la différence entre les deux se trouve dans la manière de calculer le gradient.
- ***Learning rate*** : Le *learning rate* est un réel positif et généralement très petit (entre 10^{-5} et 10^{-2}). Il est donné en paramètre à l'*opimizer* et détermine à quel point le modèle doit être modifié en réponse à une valeur de perte.
- ***Epochs*** : Il détermine le nombre d'itérations d'entraînement à effectuer.
- **Fonction de perte** : Comme évoqué précédemment, il existe plusieurs fonctions de perte, qui peuvent entraîner des résultats significativement différents en termes de performance. Certaines vont par exemple plus sanctionner les valeurs extrêmes, là où d'autres les toléreront.
- ***Lookback*** : Il s'agit de la valeur qui détermine le nombre de pas de temps fournis au modèle pour qu'il fasse une prédiction.
- ***Training size*** : Pourcentage de l'entièreté des données qui va être utilisé pour l'entraînement.
- ***batch size*** : Lors de l'entraînement du modèle, une descente de gradient (qui va modifier les paramètre du réseau) ne se fait pas à chaque prédiction. Ce paramètre détermine combien de prédictions vont être prises en compte dans chaque itération de modification des paramètres du modèle.

Les valeurs que vont prendre tous ces hyperparamètres seront décrites à la Section *7 : Résultats*.

5.1.1 Différents types de couches

La construction des modèles se fait via l'assemblage de couches, de manipulations et de redirections de leurs entrées et sorties. Ces couches peuvent être de différents types. Cette section vise à décrire le rôle de chaque couches afin de rendre ensuite la description des modèles plus succincte.

Linéaire (Dense)

Une couche linaire est composée de deux rangées de nœuds (au nombre variable) qui sont tous inter-connectés. La première rangée représente l'entrée et la seconde la sortie. Cette couche permet une modification de dimension par opération linéaire. Elle est généralement utilisée pour réduire la sortie en une dimension attendue pour la prédiction.

Dans notre cas, les modèles doivent prédire une valeur unique pour le pas de temps suivant, la derrière couche sera donc toujours une couche linéaire ayant une sortie de dimension 1.

Batch Normalization

Une couche de type *batch normalization* fait, comme son nom l'indique, la normalisation des données. Placée en sortie d'une autre couche, elle permet de de recentrer cette sortie autour de 0 avec un écart-type de 1. Cela a pour effet de rendre l'apprentissage du réseau plus rapide et plus stable [33].

Cependant la raison pour laquelle cela fonctionne est encore incertaine. Il est avancé que l'effet de la *batch normalization* pourrait lisser la fonction objectif [34].

Dropout

Une couche de type *dropout* (trad. abandon) permet aléatoirement de désactiver certains neurones lors de l'entraînement (lors de la phase de test ou d'utilisation du réseau, aucun neurone n'est désactivé). Cela permet de lutter contre l'*overfitting,* en empêchant au modèle de programmer par cœur des chemins dans le réseau.

L'hyperparamètre associé à cette couche est la probabilité d'abandon. Ce paramètre est généralement situé entre 0 et 0.3.

LSTM et GRU

Une couche LSTM (resp. GRU) est composée de cellules LSTM (resp. GRU), comme présentées à la Section 3.3. Les paramètres de ces couches sont :

- **Dimension de l'entrée** : détermine le nombre de *features* qui composent chaque pas de temps.
- **Dimension cachée** : détermine le nombre de cellules mises en parallèle sur la couche LSTM. Chaque cellule aura son propre jeu de poids.

Activation

L'application des fonctions d'activation présentées à la Section 3.1.2 se fait via des couches de fonction d'activation. Ces couches appliquent à l'entrée la fonction d'activation à laquelle elles sont associées, et n'en changent pas sa dimension.

5.1.2 Modèle de base

Afin d'évaluer la performance des différents modèles plus évolués présentés dans ce mémoire, il est utile d'avoir un modèle "naïf" de base, pouvant déterminer le niveau initial de performance à battre.

En traitement de séries temporelles, et plus particulièrement lors de séries temporelles réelles (ce modèle de base n'a par exemple aucune pertinence dans le traitement du langage naturel), un modèle de base couramment utilisé est le *repeat last*. Ce modèle fournit en prédiction pour le pas de temps suivant la dernière valeur connue.

En se rappelant la signification de la valeur *target*, le fonctionnement de ce modèle peut être décrit de la façon suivante : si le rendement résiduel sur la dernière période de 15 minutes est de $x\%$, alors il va prédire que le rendement résiduel sur les prochaines 15 minutes sera de $x\%$.

Dans la section des résultats, ce modèle sera nommé "base".

5.1.3 GRU

Le modèle GRU simple est composé d'une couche de type GRU (composée de plusieurs cellules comme expliqué précédemment). La sortie de la couche GRU, de dimension égale au nombre de cellules qui composent la couche, est ensuite passée

à travers une couche de *batch normalization* avant d'être filtrée par une couche de *dropout*.

La sortie de cette couche de *dropout* est ensuite dirigée vers une couche d'activation, afin d'apporter de la non-linéarité au réseau.

Enfin, le signal est passé à travers une couche linéaire, afin de réduire le résultat à la dimension 1, représentant ainsi la prédiction du modèle.

Les hyperparamètres supplémentaires pour ce modèle sont donc le type de fonction d'activation utilisée et le nombre de cellules qui composent la couche GRU.

Dans la section des résultats, ce modèle sera nommé "GRU".

5.1.4 LSTM

Le modèle LSTM simple est exactement le même que le modèle GRU présenté ci-dessus, dans lequel la couche GRU a été remplacée par une couche LSTM. Les hyperparamètres sont donc identiques et le flux du signal à travers les couches est lui aussi identique.

Dans la section des résultats, ce modèle sera nommé "LSTM".

5.1.5 LSTM empilés

Ce modèle va d'abord être composé d'une succession de deux blocs, chaque bloc étant décrit par :

— Couche LSTM
— Couche de *batch normalization*
— Couche de *dropout*

C'est-à-dire que pour chaque pas de temps, la sortie du premier bloc va servir d'entrée au second. Les deux couches LSTM (une par bloc) peuvent disposer d'un nombre de cellules différent. Cela crée deux hyperparamètres supplémentaires.

Ensuite, la sortie du second bloc (dont la dimension est égale au nombre de cellules qui composent la seconde couche LSTM) va passer à travers une succession de deux couches linéaires, entre lesquelles une couche d'activation est placée afin d'apporter de la non-linéarité. Cela engendre 2 hyperparamètres supplémentaires, qui sont le choix de la fonction d'activation et la valeur de la dimension intermédiaire entre les deux couches linéaires. Comme pour chaque modèle, la sortie de la

dernière couche est de dimension 1.

Dans la section des résultats, ce modèle sera nommé "*Stacked* LSTM".

5.1.6 LSTM empilés avec mécanisme d'attention

Ce modèle reprend la même architecture que le modèle précédent, auquel on a greffé le mécanisme d'attention décrit à la Section 3.5. Voici la description architecturale de ce mécanisme d'attention :

— Pour chaque pas de temps, un signal, résultat de la concaténation entre le signal d'entrée (x_t), l'*hidden state* du pas de temps précédent (h_{t-1}) et le *cell state* du pas de temps précédent (c_{t-1}) sont calculés.
— Ce signal est fourni en entrée à une couche linéaire, suivi d'une fonction *softmax*. Cela détermine une distribution de probabilités dont la somme vaut 1 sur les entrées, c'est-à-dire un poids pouvant réguler l'importance de chaque entrée.
— Ces poids multiplient ensuite l'entrée initiale (x_t) afin de pondérer les différentes *features* qui la compose.

La sortie de ce mécanisme d'attention est ensuite donnée en entrée à un réseau ayant l'architecture du modèle *staked* LSTM présenté ci-dessus.

Dans la section des résultats, ce modèle sera nommé "*Stacked* Att-LSTM".

5.1.7 LSTM empilé avec mécanismes d'attention et de *skip connections*

À nouveau, la base de l'architecture de ce modèle est celle du réseau décrite ci-dessus, à laquelle a été ajouté le principe des *skip connections* (cfr. Section 3.6).

Afin de mettre en place ce mécanisme, il a fallu changer le fonctionnement interne d'une couche LSTM. En effet, d'un point de vue implémentation, il a fallu rediriger la sortie d'une cellule LSTM d'un pas de temps t, non pas uniquement vers celle du pas de temps suivant, mais aussi vers celle située k pas de temps plus loin, temporellement parlant. Cela fait de k un hyperparamètre supplémentaire pour ce modèle.
Dans la section des résultats, ce modèle sera nommé "*Stacked* Att-Skip-LSTM".

5.1.8 LSTM empilé avec mécanismes d'attention et de *skip connections*, optimisé avec PSO

D'un point de vue architectural, ce modèle est identique au modèle *Stacked* Att-Skip-LSTM présenté ci-dessus, mais ses hyperparamètres vont être optimisés à l'aide de la méta-heuristique *Particle Swarm Optimization*. Cette méthode sera présentée à la Section 6.1.3, lors de la description des différents moyens d'optimiser les hyperparamètres d'un modèle.

Dans la section des résultats, ce modèle sera nommé "*Stacked* Att-Skip-LSTM-PSO".

5.2 Choix de la librairie d'implémentation

Le format de la compétition Kaggle engendre que le code est implémenté en Python dans un *notebook*. En revanche, les participants ont le choix en ce qui concerne les librairies utilisées pour implémenter leurs modèles.

Dans le cadre des réseaux de neurones récurrents, trois grandes librairies dominent : Keras, TensorFlow et PyTorch.

Keras est à différencier des deux autres. Cette librairie a la réputation d'être d'assez haut niveau, facile d'utilisation, avec des entraînements totalement automatisés. Cependant, cette librairie souffre du manque de performance, surtout lors de l'utilisation de grandes bases de données. Le second point critique de Keras est la possibilité de personnaliser les différents éléments constitutifs d'un modèle. Utiliser des couches LSTM simples est faisable, mais y appliquer des modifications telles que décrites à la section 5.1.1, par exemple, n'est pas possible.

Ensuite, TensorFlow et PyTorch sont très similaires, tant au niveau des performances qu'au niveau des possibilités qu'elles offrent. Il est cependant possible d'utiliser TensorFlow sans entrer dans le bas niveau de compréhension des réseaux de neurones, de leur architecture et de leur entraînement, là où PyTorch ne propose pas de simplifications de la sorte, mais un peu plus de moyens de personnalisation des modèles. D'ailleurs, comme il est possible de le voir à la Figure 5.1, PyTorch est aujourd'hui la librairie la plus utilisée dans les publications scientifiques. Puisque PyTorch possède aussi de meilleures capacités de *debugging*, il a été décidé d'utiliser cette librairie pour l'implémentation des concepts et modèles présentés dans ce mémoire.

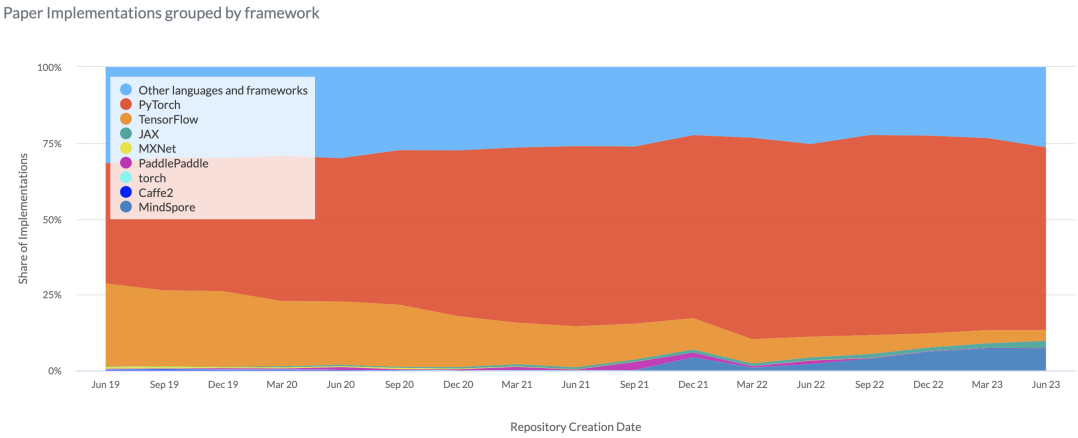


FIGURE 5.1 – Évolution de l'utilisation des différentes librairies de *machine learning* dans les publications scientifiques. Source : `https://paperswithcode.com/trends`

Le code informatique a été déposé sur un répertoire GitHub privé. Les lecteurs sont invités à transmettre à l'auteur leur adresse mail afin d'obtenir l'accès. Un document README explique la mise en place nécessaire pour toute personne souhaitant reproduire les résultats. Il s'agit d'un *notebook* écrit sur et pour l'environnement de Kaggle. Il est donc nécessaire de l'importer au sein de cet environnement pour être capable de le faire fonctionner correctement.

Cependant, l'implémentation des modèles est elle totalement indépendante de l'environnement Kaggle et peut être réutilisée dans d'autres contextes.

6 Entraînement

6.1 Optimisation des hyperparamètres

Lorsqu'il en vient à entraîner les modèles, se pose la question de l'optimisation des hyperparamètres. Comme vu précédemment, ils sont nombreux et peuvent avoir des plages de valeurs assez grandes, sans qu'il ne soit possible de prédire quelles valeurs particulières pourraient être plus prometteuses que d'autres.

Dans ce genre de situation, les méthodes les plus couramment utilisées sont le *Grid Search* et le *Random Search* (trad. recherche par quadrillage et recherche aléatoire respectivement).

6.1.1 *Grid Search*

Le *Grid Search* est une recherche exhaustive parmi tous les jeux d'hyperparamètres définis. Comme dans toute exploration exhaustive, cela veut dire que la solution optimale sera forcément trouvée. Mais évidemment, cela veut aussi dire qu'il faut restreindre le domaine de recherche, sans quoi le nombre de candidats rendrait impossible l'exploration complète. En effet, en considérant par exemple uniquement ces quatre hyperparamètres :

- `batch_size` : un entier situé entre 32 et 256 inclus (225 valeurs possibles)
- `learning_rate` : un réel entre 10^{-5} et 10^{-2} inclus, en ne considérant que des pas de temps de 10^{-5} (1000 valeurs possibles
- `lookback` : un entier situé entre 1 et 60 inclus (60 valeurs possibles)
- `dropout_probability` : un réel compris entre 0 et 0.3 inclus, qu'on peut restreindre aux pas de temps de 0.01 (30 valeurs possibles)

Cela donne déjà, uniquement avec quatre hyperparamètres, un nombre de $225 * 1000 * 60 * 30 = 405$ millions de candidats à tester !

Pour rendre la recherche faisable, le domaine est donc restreint, et les valeurs possibles pour ces hyperparamètres pourraient devenir par exemple :

— `batch_size` $\in \{32, 64, 128, 256\}$
— `learning_rate` $\in \{10^{-5}, 10^{-4}, 10^{-3}\}$
— `lookback` $\in \{1, 5, 10, 20, 30, 40, 50, 60\}$
— `dropout_probability` $\in \{0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3\}$

Ce qui rapporte désormais le nombre de candidats de jeux d'hyperparamètres à $4 * 3 * 8 * 7 = 672$.

Cependant, il est évident que si la solution optimale du problème était obtenue via un `batch_size` de 85 par exemple, cette nouvelle définition du domaine rend impossible sa découverte.

6.1.2 *Random Search*

Si le nombre de candidats obtenus en restreignant le domaine est toujours trop important pour les explorer tous, il y a la possibilité d'avoir recourt au *Random Search*.

Pour cette technique, le domaine est lui aussi restreint, et le jeu des valeurs des hyperparamètres est construit aléatoirement en sélectionnant parmi les valeurs possibles de chacun d'eux indépendemment les uns des autres. Il est alors possible d'imposer un nombre limite de tentatives. Le côté aléatoire du choix des valeurs testées engendre une exploration uniforme du domaine décrit, mais aussi une non-garantie sur la rapidité d'obtenir de bons résultats. Il est aussi possible d'arrêter la recherche lorsqu'un niveau d'optimisation suffisant (défini à l'avance) est atteint.

6.1.3 *Particle Swarm Optimization* (PSO)

Le gros point faible de la technique du *Random Search* est qu'il n'y a aucune forme d'intelligence sur le choix des valeurs des hyperparamètres. Ce choix n'est que purement aléatoire. Il est dès lors possible que la recherche explore plusieurs fois des jeux d'hyperparamètres très semblables et donnant lieu à de mauvais résultats, ou au contraire n'explorer que très rarement une zone du domaine fournissant pourtant de bons résultats.

Le principe des méta-heuristiques dans le domaine de l'optimisation des hyperparamètres vient en réalité s'attaquer à ce point faible du *Random Seaarch*.

En effet, l'idée derrière les méta-heuristiques comme *Particle Swarm Optimization* (trad. optimisation par essaims particulaires) est d'orienter le choix des futurs jeux de valeurs d'hyperparamètres à explorer en fonction de la performance de ceux testés précédemment.

Description de la méthode

La méta-heuristique PSO s'inspire du mouvement des troupeaux d'oiseaux et des bancs de poissons [35]. En effet, les premières modélisations de ces mouvements ont servi de base pour la création de l'algorithme de mise à jour des vitesses et positions des particules. Dans le cas des bancs de poisson, une particule représente un seul poisson.

Le but initial de cette méthode est de converger vers un candidat qui minimise une fonction non-linéaire. La valeur du candidat est sa position, et sa vitesse représente la modification qui sera apportée à ce candidat pour la prochaine itération.

Lorsque la méthode PSO est appliquée à l'optimisation des hyperparamètres d'un modèle, une particule est un jeu d'hyperparamètres appartenant au domaine. La position d'une particule représente le vecteur des valeurs de ces hyperparamètres. La mise à jour de la position d'une particule sera influencée à la fois par la meilleure solution que la particule ait vue, et à la fois par la meilleure solution trouvée sur l'entièreté du domaine (c'est-à-dire parmi toutes les solutions rencontrées, par toutes les particules).

Avec $v_{i,d}$ la vitesse de la particule i à l'itération d, $x_{i,d}$ sa position, $p_{i,d}$ la position qui a entraîné la meilleure solution pour cette particule jusqu'à présent et g_d la position associée à la meilleure solution trouvée sur l'ensemble du modèle, la mise à jour des positions se fait à chaque itération selon les relations suivantes :

$$v_{i,d} = wv_{i,d} + \phi_p r_p(p_{i,d} - x_{i,d}) + \phi_g r_g(g_d - x_{i,d})$$
$$x_{i,d} = x_{i,d} + v_{i,d}$$

Dans ces équations, ϕ_p et ϕ_g sont les deux paramètres permettant de moduler l'importance accordée à la meilleure solution individuelle et à la meilleure solution collective, respectivement. Le poids w est le troisième paramètre de cette méthode, qui permet d'ajuster l'inertie d'un mouvement précédent. Pour un w proche de 0, la particule aura tendance a rapidement changer de direction et de vitesse lorsqu'une meilleurs solution a été trouvée ailleurs (peu d'inertie). En revanche, avec un w proche de 1, la particule aura tendance à continuer dans sa direction et être peu

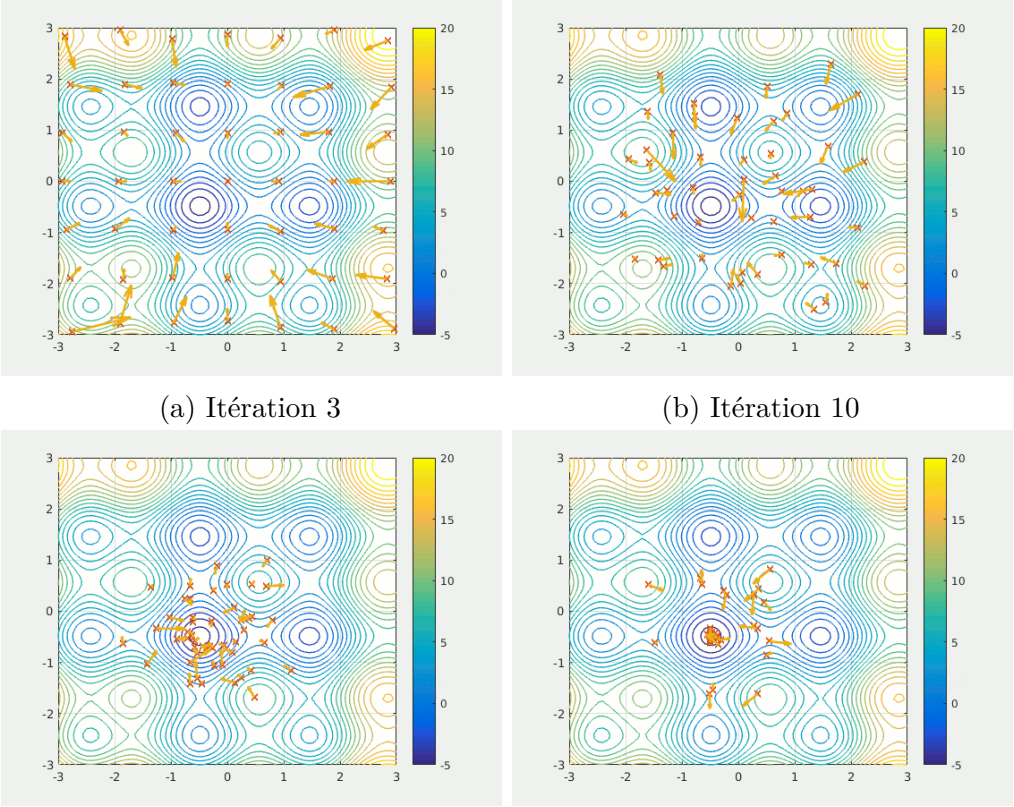
influencée par les nouvelles solution trouvées par l'essaim (beaucoup d'inertie). Les variables r_p et r_p sont des valeurs aléatoires, tirées depuis la distribution uniforme $\mathcal{U}(0, 1)$.

À l'initialisation, une position aléatoire est sélectionnée pour chaque particule, indépendemment les unes des autres (ce qui établi la première ressemblance avec *Random Search*). Il en est de même pour les vitesses initiales. L'algorithme PSO se termine lorsqu'un nombre maximal d'itérations ou un niveau suffisant de l'objectif a été atteint, comme pour *Random Search*.

Le dernier paramètre de cette méthode est le nombre de particules. En effet, plus celui-ci est grand, plus une partie vaste du domaine pourra être explorée. Plus la dimension du domaine (c'est-à-dire le nombre d'hyperparamètres à optimiser) est importante, plus il faudra initialiser de particules. En effet, c'est encore dû à l'effet du principe de *curse of dimensionality* [31].

Une illustration visuelle du fonctionnement de cette méthode itération après itération est présentée à la Figure 6.1 [1].

1. Il s'agit de visualisations tirées d'un GIF. L'auteur invite tout lecteur de la version électronique à visualiser la version animée via la source de la Figure 6.1



(a) Itération 3 (b) Itération 10

(c) Itération 25 (d) Itération 55

FIGURE 6.1 – Illustration de l'évolution des positions et vitesses de particules durant l'algorithme PSO, pour une fonction en 2 dimensions. Source : `https://commons.wikimedia.org/wiki/File:ParticleSwarmArrowsAnimation.gif#/media/File:ParticleSwarmArrowsAnimation.gif`

6.2 Procédure d'entraînement

Comme il a été mentionné précédemment, les réseaux de neurones récurrents, notamment les LSTM, sont entraîné sur des données séquentielles. La construction de ce jeu de données est illustrée à la Figure 6.2.

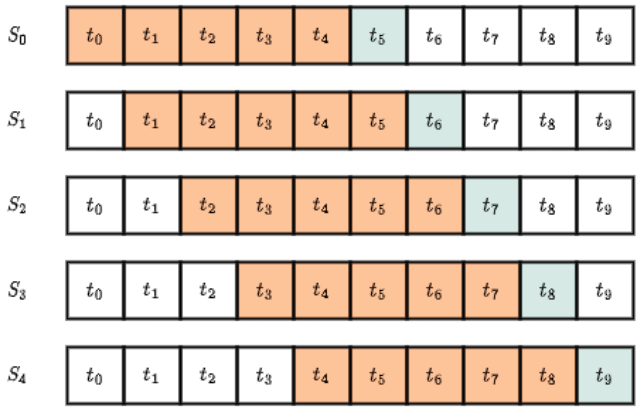


FIGURE 6.2 – Représentation de la construction des séquences d'entraînement d'un RNN

Les pas de temps colorés en rouge représentent les pas de temps pour lesquels le modèle recevra les *features* d'entrée. Le pas de temps en vert est celui qui contient la valeur cible (vraie valeur) que doit prédire le modèle.

6.2.1 Environnement d'entraînement

Tous les entraînements des modèles ont été réalisés via le *notebook* hébergé sur Kaggle. Cet environnement permet l'accès à du matériel plus performant que ce que propose un ordinateur portable classique. Les accélérateurs disponibles sont un GPU P100 et un double GPU T4. De plus, il est possible d'utiliser jusqu'à 30 Gb de RAM.

Malgré l'accélération que permet ce matériel, l'entraînement reste une étape très longue et lente. Afin d'accélérer encore ce processus, des mécanismes d'ajustement dynamique du *learning rate* et d'*early stopping* ont été implémentés.

6.2.2 Techniques d'accélération de l'entraînement

Le *package* PyTorch fourni le planificateur dynamique du *learning rate* appelé `ReduceLROnPlateau`. Ce planificateur prend comme paramètre le nombre d'itérations d'entraînement consécutives sans amélioration à attendre avant de réduire le taux d'apprentissage. Un second paramètre est le facteur de réduction à appliquer à chaque déclenchement.

Pour ce qui est du mécanisme d'arrêt anticipé (ang. *eaarly stopping*), il prend également en paramètre le nombre d'itérations tolérées sans amélioration, après

quoi l'entraînement du modèle s'arrête.

Ces techniques permettent d'éviter de continuer l'entraînement d'un modèle qui semble ne pas s'améliorer au fil des itérations, et ainsi gagner un temps considérable sur la recherche des hyperparamètres optimaux.

6.2.3 Métriques de performance

Afin de correctement déterminer la qualité des modèles entraînés, il est important de fixer des métriques pertinentes. En plus du RMSE (présenté à la section 3.1.3), deux autres mesures ont été utilisées.

Le coefficient de Pearson

Proposé comme méthode d'évaluation par Kaggle pour cette compétition, ce coefficient mesure la corrélation linéaire entre deux vecteurs de réels. Calculer le coefficient de Pearson entre les valeurs prédites et les valeurs réelles permet donc d'évaluer la ressemblance linéaire entre ces deux séries de valeurs.

Le calcul de ce coefficient de Pearson entre deux vecteurs $\mathbf{x}$ et $\mathbf{y}$ est défini comme :

$$r_{\mathbf{xy}} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$

où n est le nombre d'éléments qui composent les vecteurs $\mathbf{x}$ et $\mathbf{y}$. Il prend une valeur entre -1 et 1 (1 signifie une corrélation linéaire parfaite, 0 signifie l'absence de corrélation linéaire, et -1 la présence d'une corrélation linaire négative).

Dans le cadre de la prédiction d'évolution d'une série temporelle, un coefficient de Pearson le plus élevé possible est espéré.

La précision pondérée

Lorsqu'il est question d'évolution financière, il est utile d'être capable de prédire la direction dans laquelle va évoluer un actif. Les *traders*, pariant à la baisse ou à la hausse sur les actifs, sont en effet attentifs à ce facteur.

Il est donc intéressant d'observer la capacité des modèles vis-à-vis de la prédiction de la direction du rendement des crypto-monnaies.

Une précision pondérée, basée sur le nombre de vrais positifs (TP), faux positifs (FP), vrais négatifs (TN) et faux négatifs (FN), permet de déterminer le pourcentage de prédictions allant dans la bonne direction. La version pondérée de la précision est utilisée afin de ne pas subir le biais de la probabilité *a priori* d'une des classes.

En effet, sans cette pondération, le résultat de la précision peut ne pas refléter la capacité réelle de prédiction d'un modèle. Imaginons que le modèle doivent prédire les valeurs $\{1, 1, 1, 1, 1, 1, -1\}$, il suffirait de toujours prédire la valeur 1 pour obtenir une précision de plus de 85% via la formule [17] :

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} = 0.857$$

À l'aide de la précision pondérée, la précision tomberait à 50%, traduisant bien, pour une prédiction parmi deux classes distinctes (positive et négative) l'équivalence à une prédiction aléatoire. La formule de la précision pondérée s'écrit comme :

$$Acc = \frac{TPR + TNR}{2} = 0.5$$

où $TPR = \frac{TP}{P}$, P étant le nombre de positifs contenus dans les données, et $TNR = \frac{TN}{N}$, N étant le nombre de négatifs dans les données. Dans les résultats, cette valeur sera nommée "Acc".

L'objectif est d'observer pour les modèles une valeur de précision la plus éloignée possible de 50% [2].

2. Bien qu'il soit tentant de vouloir une valeur supérieure à 50% de précision, un résultat montrant une précision de par exemple 40% serait un très bon résultat. En effet, il suffirait alors de prendre l'opposé de la prédiction du modèle pour prédire dans 60% des cas la bonne direction.

7 Résultats

Malgré l'utilisation des GPUs de Kaggle et des mécanismes d'*early stopping* et de *learning rate scheduling*, le facteur limitant est resté le temps d'entraînement des modèles.

En effet, lorsque les modèles étaient entraînés sur l'entièreté des données, ils présentaient des signes d'*underfitting*. L'*underfitting*, au contraire de l'*overfitting*, est le phénomène durant lequel le terme d'erreur (calculé sur les données d'entraînement) ne diminue pas malgré les itérations.

Pour remédier à cela, il aurait fallu augmenter le nombre de paramètres entraînables qui composent les modèles (c'est-à-dire augmenter leur complexité). Cependant, certains modèles avec plus de 1.5 millions de paramètres ont eux aussi souffert de l'*underfitting* malgré les nombreuses heures d'entraînement. Continuer d'augmenter la complexité des modèles en terme de nombres de paramètre n'était malheureusement plus envisageable, étant donné les ressources demandées pour les entraîner.

Comme évoqué en tout début de document (lors de la présentation de l'hypothèse des marchés efficients) le marché des crypto-monnaies tend à renforcer de plus en plus son caractère efficient. Dès lors, il n'est pas surprenant que des modèles de cette échelle de complexité ne puisse conclure d'aucun résultat probant.

La Table A.1, située en annexes pour la lisibilité du document, reprend les métriques de performance pour les différents modèles présentés à la section *Implémentation des modèles*, entraînés sur l'entièreté des données disponibles, pour le Bitcoin, l'Ethereum et le Litecoin.

Bien qu'une tendance semble se dessiner, aucun résultat n'est convaincant. En terme de précision (cfr. Section 6.2.3), la meilleure valeur atteinte est de 50.17%, ce qui n'est pas significativement meilleur qu'une prédiction aléatoire obtenant 50% de

précision. Le RSME semble tout de même légèrement diminuer avec l'augmentation de la complexité du modèle. Pour ce qui est de la corrélation de Pearson, il ne semble pas y avoir de tendance pour ces jeux de données, et le modèle de base (cfr. 5) fait aussi bien que les certains autres modèles.

Avec l'hypothèse que la maturité du marché des crypto-monnaies est positivement corrélée à son efficience, il a été testé d'entraîner les modèles sur les 400 000 (par crypto-monnaie) données les plus anciennes disponibles dans les données fournies par Kaggle (pour rappel, chaque crypto-monnaie possède initialement environ 1.75 millions de données).

Quelques dizaines d'heures d'entraînement plus tard, il était possible d'observer les résultats reportés à la Table A.2 (située en annexes).

Déjà, nous pouvons remarquer que le modèle LSTM simple donne de meilleurs résultats que le modèle GRU. C'est la raison pour laquelle les modèles plus élaborés qui ont suivi ont été basés sur les cellules LSTM.

Ensuite, nous pouvons distinguer une nette domination du modèle optimisé par PSO. En visualisant les hyperparamètres finaux des modèles entraînés (visibles à l'Annexe B), on peut remarquer que les hyperparamètres du modèle optimisé par PSO sont assez éloignés des valeurs testées pour les autres modèles.

Ceci peut être l'explication de cette différence significative des performances. En effet, les autres modèles, en explorant des jeux d'hyperparamètres fixés, ne peuvent pas explorer toutes les zones du domaine, comme le peut la méthode PSO. Il semblerait donc que les hyperparamètres choisis ne permettent pas d'atteindre la combinaison optimale.

En conclusion, le modèle le plus abouti permet, sur ces données plus anciennes, d'atteindre une prédiction significativement meilleure que ce que celle du modèle naïf de base. En effet, le modèle *Stacked*-Att-Skip-LSTM-PSO atteint une précision de prédiction du mouvement du rendement du Litecoin de près de 58%. La correlation linéaire de Pearson augmente elle aussi fortement avec ce modèle comparé au autres en atteignant 0.0616.

8 Pistes futures

Les modèles présentés dans ce mémoire nécessitent un temps conséquent d'entraînement et de prédiction. Entraînés sur des données représentant des pas de temps de 60 secondes, il serait pertinent d'explorer des modèles plus adaptés à la prédiction en direct.

Une piste principale serait aussi d'augmenter la complexité des modèles, afin d'éviter l'*underfitting* rencontré. Pour cela, il faudrait alors avoir recourt à des ressources matérielles plus importantes.

Conclusion

Après avoir découvert dans la littérature scientifique que les réseaux LSTM étaient des candidats prometteurs à explorer, une revue de la théorie fondamentale des concepts associés à été faite.

Certaines notions avancées, nécessitant un langage de programmation de bas niveau pour les implémenter, ont été décrites et ajoutée à l'architecture de base des réseaux LSTM. Cela a abouti à une succession de réseaux de plus en plus complexes, et de plus en plus performants.

Comme disait Niels Bohr [36], "*Prediction is very difficult, especially if it's about the future !*" (trad. "Prédire est très difficile, surtout quand il s'agit du futur !").

Il est certains que les modèles présentés dans ce mémoires ne permettront à personne de vaincre les marchés. Cependant leurs prédictions peuvent servir d'indicateur. En effet, malgré la pauvre performance sur le jeu de données complet, il a été montré que certains modèles (sur des données plus anciennes), surtout le réseau LSTM empilés, avec mécanismes d'attention et de *skip connections*, entraîné via PSO, peut réellement prédire la direction de la valeur du retour d'une crypto-monnaie, et ce significativement mieux que l'aléatoire ne peut le faire.

Ces architectures avancées n'ont donc pas été vaines et auront au moins permis de mettre en évidence le renforcement de la caractéristique efficiente du marché des crypto-monnaies.

Bibliographie

[1] D. E. Sanger, "WALL STREET'S TOMORROW MACHINE," en-US, *The New York Times*, oct. 1986, ISSN : 0362-4331. adresse : https://www.nytimes.com/1986/10/19/business/wall-street-s-tomorrow-machine.html (visité le 18/08/2023).

[2] *BlackRock, Inc. - Financials - Quarterly Results*, en-US. adresse : https://ir.blackrock.com/financials/quarterly-results/default.aspx (visité le 18/08/2023).

[3] systematic-investing, *How AI is Transforming Investing*, en-US. adresse : https://www.blackrock.com/us/individual/insights/ai-investing (visité le 18/08/2023).

[4] *79+ Amazing Algorithmic Trading Statistics (2023) - Analyzing Alpha*, en-us, Section : Algorithmic Trading, mai 2021. adresse : https://analyzingalpha.com/algorithmic-trading-statistics (visité le 18/08/2023).

[5] AdamSCochran, *Adam Cochran | Substack*, en. adresse : https://adamcochran.substack.com/embed (visité le 18/08/2023).

[6] A. Urquhart, "The inefficiency of Bitcoin," en, *Economics Letters*, t. 148, p. 80-82, nov. 2016, ISSN : 01651765. DOI : 10.1016/j.econlet.2016.09.019. adresse : https://linkinghub.elsevier.com/retrieve/pii/S0165176516303640 (visité le 18/08/2023).

[7] D. Shah et K. Zhang, "Bayesian regression and Bitcoin," 2014, Publisher : arXiv Version Number : 1. DOI : 10.48550/ARXIV.1410.1231. adresse : https://arxiv.org/abs/1410.1231 (visité le 18/08/2023).

[8] *What Is a Sharpe Ratio ? Understanding Its Use in Investing*, en. adresse : https://www.investopedia.com/ask/answers/010815/what-good-sharpe-ratio.asp (visité le 18/08/2023).

[9] *Sharpe Ratio : Formula & Calculation in Trading | CMC Markets*. adresse : https://www.cmcmarkets.com/en/trading-guides/the-sharpe-ratio-definition (visité le 18/08/2023).

[10] Z. Chen, C. Li et W. Sun, "Bitcoin price prediction using machine learning : An approach to sample dimension engineering," en, *Journal of Computational and Applied Mathematics*, t. 365, p. 112 395, fév. 2020, issn : 03770427. doi : 10.1016/j.cam.2019.112395. adresse : https://linkinghub.elsevier.com/retrieve/pii/S037704271930398X (visité le 18/08/2023).

[11] A. Aggarwal, I. Gupta, N. Garg et A. Goel, "Deep Learning Approach to Determine the Impact of Socio Economic Factors on Bitcoin Price Prediction," in *2019 Twelfth International Conference on Contemporary Computing (IC3)*, Noida, India : IEEE, août 2019, p. 1-5, isbn : 978-1-72813-591-5. doi : 10.1109/IC3.2019.8844928. adresse : https://ieeexplore.ieee.org/document/8844928/ (visité le 18/08/2023).

[12] L. Felizardo, R. Oliveira, E. Del-Moral-Hernandez et F. Cozman, "Comparative study of Bitcoin price prediction using WaveNets, Recurrent Neural Networks and other Machine Learning Methods," in *2019 6th International Conference on Behavioral, Economic and Socio-Cultural Computing (BESC)*, Beijing, China : IEEE, oct. 2019, p. 1-6, isbn : 978-1-72814-762-8. doi : 10.1109/BESC48373.2019.8963009. adresse : https://ieeexplore.ieee.org/document/8963009/ (visité le 18/08/2023).

[13] K. Murray, A. Rossi, D. Carraro et A. Visentin, "On Forecasting Cryptocurrency Prices : A Comparison of Machine Learning, Deep Learning, and Ensembles," en, *Forecasting*, t. 5, n° 1, p. 196-209, jan. 2023, issn : 2571-9394. doi : 10.3390/forecast5010010. adresse : https://www.mdpi.com/2571-9394/5/1/10 (visité le 18/08/2023).

[14] P. T. Yamak, L. Yujian et P. K. Gadosey, "A Comparison between ARIMA, LSTM, and GRU for Time Series Forecasting," in *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, sér. ACAI '19, event-place : Sanya, China, New York, NY, USA : Association for Computing Machinery, 2020, p. 49-55, isbn : 978-1-4503-7261-9. doi : 10.1145/3377713.3377722. adresse : https://doi.org/10.1145/3377713.3377722.

[15] P. Mohanty, D. Patel, P. Patel et S. Roy, "Predicting Fluctuations in Cryptocurrencies' Price using users' Comments and Real-time Prices," in *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, Noida, India : IEEE, août 2018, p. 477-482, isbn : 978-1-5386-4692-2. doi : 10.1109/ICRITO.2018.8748792. adresse : https://ieeexplore.ieee.org/document/8748792/ (visité le 18/08/2023).

[16] M. Saad, J. Choi, D. Nyang, J. Kim et A. Mohaisen, "Toward Characterizing Blockchain-Based Cryptocurrencies for Highly Accurate Predictions," *IEEE Systems Journal*, t. 14, nº 1, p. 321-332, mars 2020, ISSN : 1932-8184, 1937-9234, 2373-7816. DOI : 10.1109/JSYST.2019.2927707. adresse : https://ieeexplore.ieee.org/document/8840919/ (visité le 18/08/2023).

[17] A. M. Sharifi, K. Khalili Damghani, F. Abdi et S. Sardar, "A hybrid model for predicting bitcoin price using machine learning and metaheuristic algorithms," eng, *Journal of Applied Research on Industrial Engineering*, t. 9, nº 1, mars 2022. DOI : 10.22105/jarie.2021.291175.1343. adresse : https://doi.org/10.22105/jarie.2021.291175.1343 (visité le 18/08/2023).

[18] K. Kumar et M. T. U. Haider, "Enhanced Prediction of Intra-day Stock Market Using Metaheuristic Optimization on RNN–LSTM Network," en, *New Generation Computing*, t. 39, nº 1, p. 231-272, avr. 2021, ISSN : 0288-3635, 1882-7055. DOI : 10.1007/s00354-020-00104-0. adresse : https://link.springer.com/10.1007/s00354-020-00104-0 (visité le 18/08/2023).

[19] Z. Yang et Z. Yang, "Artificial Neural Networks," en, in *Comprehensive Biomedical Physics*, Elsevier, 2014, p. 1-17, ISBN : 978-0-444-53633-4. DOI : 10.1016/B978-0-444-53632-7.01101-1. adresse : https://linkinghub.elsevier.com/retrieve/pii/B9780444536327011011 (visité le 18/08/2023).

[20] D. Gupta, *Fundamentals of Deep Learning - Activation Functions and When to Use Them ?* en, jan. 2020. adresse : https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/ (visité le 18/08/2023).

[21] A. Quintiliano Bezerra Silva, "Predicting Cervical Cancer with Metaheuristic Optimizers for Training LSTM," en, in *Computational Science – ICCS 2019*, J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro et al., éd., t. 11540, Series Title : Lecture Notes in Computer Science, Cham : Springer International Publishing, 2019, p. 642-655, ISBN : 978-3-030-22749-4 978-3-030-22750-0. DOI : 10.1007/978-3-030-22750-0_62. adresse : https://link.springer.com/10.1007/978-3-030-22750-0_62 (visité le 18/08/2023).

[22] *Vanishing gradient problem*, fr, Page Version ID : 1006243644, fév. 2021. adresse : https://en.wikipedia.org/w/index.php?title=Vanishing_gradient_problem&oldid=1006243644 (visité le 18/08/2023).

[23] W. Jia, Y. Zhang, Z. Wei, Z. Zheng et P. Xie, "Daily reference evapotranspiration prediction for irrigation scheduling decisions based on the hybrid PSO-LSTM model," en, *PLOS ONE*, t. 18, nº 4, A. Lewis, éd.,

e0281478, avr. 2023, ISSN : 1932-6203. DOI : 10.1371/journal.pone.0281478. adresse : https://dx.plos.org/10.1371/journal.pone.0281478 (visité le 18/08/2023).

[24] Y. QIN, D. SONG, H. CHEN, W. CHENG, G. JIANG et G. COTTRELL, *A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction*, arXiv :1704.02971 [cs, stat], août 2017. adresse : http://arxiv.org/abs/1704.02971 (visité le 18/08/2023).

[25] T. GUI, Q. ZHANG, L. ZHAO et al., "Long Short-Term Memory with Dynamic Skip Connections," 2018, Publisher : arXiv Version Number : 1. DOI : 10.48550/ARXIV.1811.03873. adresse : https://arxiv.org/abs/1811.03873 (visité le 18/08/2023).

[26] *Litecoin price today, LTC to USD live price, marketcap and chart*, en. adresse : https://coinmarketcap.com/currencies/litecoin/ (visité le 18/08/2023).

[27] L. CHMIELEWSKI, M. JANOWICZ, J. KALETA et A. ORŁOWSKI, "Pattern Recognition in the Japanese Candlesticks," en, in *Soft Computing in Computer and Information Science*, A. WILIŃSKI, I. E. FRAY et J. PEJAŚ, éd., t. 342, Series Title : Advances in Intelligent Systems and Computing, Cham : Springer International Publishing, 2015, p. 227-234, ISBN : 978-3-319-15146-5 978-3-319-15147-2. DOI : 10.1007/978-3-319-15147-2_19. adresse : https://link.springer.com/10.1007/978-3-319-15147-2_19 (visité le 18/08/2023).

[28] A. USORO, "On the Stationarity of Multivariate Time Series," t. 3, p. 160-169, août 2020.

[29] C.-H. WU, C.-C. LU, Y.-F. MA et R.-S. LU, "A New Forecasting Framework for Bitcoin Price with LSTM," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, Singapore, Singapore : IEEE, nov. 2018, p. 168-175, ISBN : 978-1-5386-9288-2. DOI : 10.1109/ICDMW.2018.00032. adresse : https://ieeexplore.ieee.org/document/8637486/ (visité le 18/08/2023).

[30] L. PAN, "Cryptocurrency Price Prediction Based on ARIMA, Random Forest and LSTM Algorithm," *BCP Business & Management*, t. 38, p. 3396-3404, mars 2023, ISSN : 2692-6156. DOI : 10.54691/bcpbm.v38i.4313. adresse : https://bcpublication.org/index.php/BM/article/view/4313 (visité le 18/08/2023).

[31] A. Crespo Márquez, "The Curse of Dimensionality," en, in *Digital Maintenance Management*, Series Title : Springer Series in Reliability Engineering, Cham : Springer International Publishing, 2022, p. 67-86, ISBN : 978-3-030-97659-0 978-3-030-97660-6. DOI : 10.1007/978-3-030-97660-6_7. adresse : https://link.springer.com/10.1007/978-3-030-97660-6_7 (visité le 18/08/2023).

[32] S. Chen et L. Ge, "Exploring the attention mechanism in LSTM-based Hong Kong stock price movement prediction," en, *Quantitative Finance*, t. 19, n° 9, p. 1507-1515, sept. 2019, ISSN : 1469-7688, 1469-7696. DOI : 10.1080/14697688.2019.1622287. adresse : https://www.tandfonline.com/doi/full/10.1080/14697688.2019.1622287 (visité le 18/08/2023).

[33] S. Ioffe et C. Szegedy, "Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015, Publisher : arXiv Version Number : 3. DOI : 10.48550/ARXIV.1502.03167. adresse : https://arxiv.org/abs/1502.03167 (visité le 18/08/2023).

[34] S. Santurkar, D. Tsipras, A. Ilyas et A. Madry, "How Does Batch Normalization Help Optimization ?," 2018, Publisher : arXiv Version Number : 5. DOI : 10.48550/ARXIV.1805.11604. adresse : https://arxiv.org/abs/1805.11604 (visité le 18/08/2023).

[35] J. Kennedy et R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, t. 4, Perth, WA, Australia : IEEE, 1995, p. 1942-1948, ISBN : 978-0-7803-2768-9. DOI : 10.1109/ICNN.1995.488968. adresse : http://ieeexplore.ieee.org/document/488968/ (visité le 18/08/2023).

[36] D. P. Dickstein, "Editorial : It's Difficult To Make Predictions, Especially About the Future : Risk Calculators Come of Age in Child Psychiatry," en, *Journal of the American Academy of Child & Adolescent Psychiatry*, t. 60, n° 8, p. 950-951, août 2021, ISSN : 08908567. DOI : 10.1016/j.jaac.2020.12.029. adresse : https://linkinghub.elsevier.com/retrieve/pii/S0890856720322188 (visité le 18/08/2023).

A Résultats des modèles entraînés

Modèle	Bitcoin			Ethereum			Litecoin		
	rmse	acc	pear.	rmse	acc	pear.	rmse	acc	pear.
Base model	1.87×10^{-3}	0.4998	0.0094	1.49×10^{-3}	0.50	0.0087	1.12×10^{-3}	0.50	0.0098
GRU	1.84×10^{-3}	0.4996	0.0105	1.42×10^{-3}	0.50	0.0093	1.15×10^{-3}	0.5003	0.0086
LSTM	1.77×10^{-3}	0.5011	0.0109	1.44×10^{-3}	0.5002	0.0112	1.09×10^{-3}	0.4988	0.0091
Stacked-LSTM	1.76×10^{-3}	0.4992	0.0095	1.43×10^{-3}	0.4999	0.0103	1.11×10^{-3}	0.5001	0.0101
Stacked-Att-LSTM	1.76×10^{-3}	0.5009	0.0109	1.41×10^{-3}	0.4999	0.0093	1.08×10^{-3}	0.5011	0.0099
Stacked-Att-Skip-LSTM	1.74×10^{-3}	0.5014	0.0104	1.4×10^{-3}	0.50	0.0107	1.08×10^{-3}	0.4997	0.0087
Stacked-Att-Skip-LSTM-PSO	1.73×10^{-3}	0.5017	0.0121	1.39×10^{-3}	0.5009	0.0114	1.07×10^{-3}	0.5005	0.0098

TABLE A.1 – Résultats non-probants sur l'entièreté des données

Modèle	Bitcoin			Ethereum			Litecoin		
	rmse	acc	pear.	rmse	acc	pear.	rmse	acc	pear.
Base model	1.82×10^{-3}	0.4999	0.0091	1.51×10^{-3}	0.5001	0.0092	1.12×10^{-3}	0.4999	0.0076
GRU	1.65×10^{-3}	0.5054	0.0126	1.41×10^{-3}	0.5028	0.0074	1.04×10^{-3}	0.5034	0.0109
LSTM	1.61×10^{-3}	0.5063	0.0161	1.37×10^{-3}	0.5085	0.0105	1.00×10^{-3}	0.5069	0.0122
Stacked-LSTM	1.60×10^{-3}	0.5059	0.0154	1.38×10^{-3}	0.5074	0.0163	1.02×10^{-3}	0.5169	0.0230
Stacked-Att-LSTM	1.43×10^{-3}	0.5128	0.0259	1.23×10^{-3}	0.5141	0.0204	9.53×10^{-4}	0.5199	0.0309
Stacked-Att-Skip-LSTM	1.34×10^{-3}	0.5334	0.0523	1.11×10^{-3}	0.5499	0.0389	9.12×10^{-4}	0.5549	0.0302
Stacked-Att-Skip-LSTM-PSO	1.29×10^{-3}	0.5548	0.0616	1.08×10^{-3}	0.5502	0.0533	9.09×10^{-4}	0.5788	0.0464

TABLE A.2 – Résultats intéressants sur un échantillon plus ancien des données

B Valeurs des hyperparamètres

Avec le nombre élevé d'hyperparamètres à optimiser, il a fallu en fixer certains de manière arbitraire, en se basant sur la littérature scientifique. Voici, à la Table B.1, la liste de ces hyperparamètres :

Hyperparamètre	**Valeur**
Optimizer	Adam
Training size	80%
Fonction d'activation	*LeakyReLU*
Scaler	*MinMaxScaler*
Fonction d'erreur	MSE

Table B.1 – Caption

Enfin, les hyperparamètres personnalisés pour chaque modèle sont cités à la Table B.2

Modèle	Données	*Batch*	*Dropout*	*Lookback*	*Hidden dim*	*Learning rate*
GRU	BTC	64	0.2	5	$(80,)$	10^{-4}
	ETH	64	0.15	5	$(80,)$	10^{-4}
	LTC	64	0.15	5	$(80,)$	10^{-4}
LSTM	BTC	64	0.2	10	$(80,)$	10^{-4}
	ETH	128	0.1	10	$(80,)$	10^{-4}
	LTC	64	0.2	15	$(80,)$	10^{-4}
Stacked-LSTM	BTC	128	0.2	15	$(100, 60, 30)$	10^{-4}
	ETH	128	0.1	15	$(80, 40, 30)$	5×10^{-4}
	LTC	64	0.15	10	$(100, 60, 30)$	5×10^{-4}
Stacked-Att-LSTM	BTC	64	0.15	10	$(100, 60, 30)$	10^{-4}
	ETH	128	0.15	15	$(100, 60, 30)$	10^{-4}
	LTC	64	0.2	15	$(100, 60, 30)$	10^{-4}
Stacked-Att-Skip-LSTM	BTC	128	0.2	10	$(80, 60, 30)$	10^{-4}
	ETH	128	0.2	15	$(80, 40, 30)$	10^{-4}
	LTC	128	0.2	15	$(100, 60, 30)$	5×10^{-4}
Stacked-Att-Skip-LSTM-PSO	BTC	95	0.19	12	$(86, 72, 40)$	1.35×10^{-4}
	ETH	78	0.19	9	$(94, 54, 28)$	1.05×10^{-4}
	LTC	74	0.2	15	$(76, 60, 32)$	1.22×10^{-4}

TABLE B.2 – Liste des hyperparamètres des modèles entraînés

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**