Prévision de séries chronologiques par approche Deep Learning

Zo Lalaina Yannick RAHARIJAONA, Maxime Durand GASSELIN, Nabil BENJAA

I. RÉSUMÉ

Actuellement, le Deep Learning est de plus en plus utilisés dans les applications de série chronologique. Dans ce projet, nous allons résoudre le problème de prévision de série chronologique en utilisant des architectures de Deep Learning comme LSTM [2] et SerieNet [5]. Le but est de montrer l'efficacité de ces models de Deep Learning pour résoudre un problème de prédiction des variations d'une action coté en bourse mais aussi de justifier que ces approches surpassent en terme de performance les méthodes classiques comme ARIMA ("Autoregressive integrated moving average").

II. INTRODUCTION

La principale tâche à faire dans notre projet est de prédir le prix de la bourse dans le futur en se basant sur les données antérieures que nous connaissons a priori. Les approches traditionnelles comme ARIMA ont été utiliser pendant plusieurs années pour traiter les problèmes en séries chronologiques. Dans ce projet, ARIMA est considèrer comme notre "baseline model" ainsi, on s'attend à ce que les modeles de Deep Learning proposées soit beaucoup plus performant que ARIMA.

III. DATASET

Le dataset utilisé se nomme S&P500 et se retrouve à cette adresse [1]. Le dossier contient l'enregistrement journalier de 505 actions de février 2013 à février 2018. Pour chaque jour, on a la valeur à l'ouverture, à la fermeture, la valeur minimal, maximal et le volume d'action. Chaque fichier d'action enregistre 1259 points (jour) sauf quelque uns que ne serons jamais considérés.

Nous n'avons pas utilisé d'augmentation de données sur le dataset car il ne nous semble pas intuitif de rajouter des variations supplémentaire sur une série chronologique. Ça ajouterais du bruit plutôt. Dans les problèmes de classification (par exemple d'image), la cible d'une donnée ne change avec les techniques d'augmentation. Ce n'est pas le cas avec les séries chronologiques car les cibles sont aussi des entrée pour une prédiction ultérieur.

IV. ARCHITECTURES DES MODÈLES PROPOSÉS

SeriesNet

Ce modèle [5], publié en 2018, est spécialisé pour la prédiction de séries chronologique. Il s'est principalement inspiré de l'architecture du WaveNet [3] publié en 2016 par un groupe de Deepmind. Ce dernier modèle est utilisé pour la génération audio du langage. Dans ce projet, nous avons ré-implémenté une version de ce répertoire en ligne [4]. Une différence par rapport à [5] est qu'il n'utilise pas de modèle LSTM en parallèle. Nous avons fait ce choix pour isoler l'effet de chacun des morceaux. Notre contribution est d'avoir factorisé ce code disponible pour pouvoir le paramétrer à notre guise.

La particularité de WaveNet, SeriesNet et notre modèle est la convolution causal. C'est une adaptation de la convolution spatiale dilaté. Le noyau utilisé est de taille 2. La dilatation augmente par un facteur deux (2) entre deux blocs. Ainsi, le prochain point prédit dépend des N points antérieurs :

$$N = 1 + \sum_{i=0}^{B-1} 2^i \tag{1}$$

où B est le nombre de bloc causal utilisé dans le modèle. La figure 1 schématise les connexions et les calculs fait dans un bloc causal. La figure 2 montre l'interaction entre une chaîne de plusieurs blocs. À cause de la connexion résiduel, il faut padder l'entrée mais uniquement pour les temps antérieur à la première donnée.

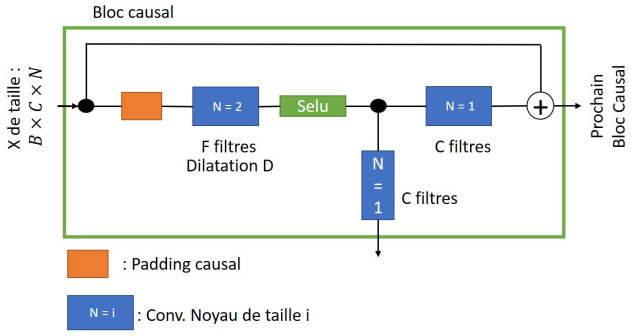

FIG. 1. Schéma représentant un bloc causal. Le bloc est constitué d'une entré et deux sorties. La première (en dessous) contribue à la prédiction et celle de droite sera l'entrée pour le bloc suivant.

Les hyper-paramètres de ce réseau sont le nombre de blocs causal, le nombre de filtres pour la convolution dilatée, le taux d'apprentissage, le type d'optimiseur et le nombre de blocs causal où on applique du dropout.

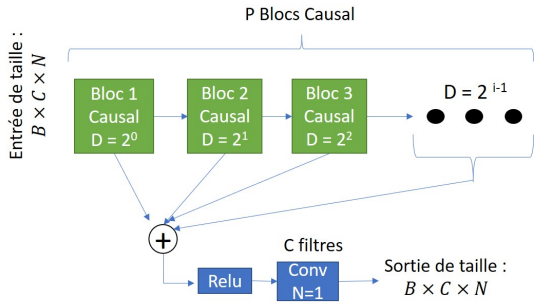

FIG. 2. Schéma représentant le réseau SeriesNet composé de P blocs. Le paramètre de dilatation de la convolution de chaque bloc est déterminé selon sa position dans la chaîne. La taille du vecteur de sortie est la même qu'en entrée.

Le modèle SeriesNet de la figure 2 prends 1:N points en entrée et produit 2:N+1 prédictions. Le dernier point produit est comparé au prochain point de donné pour calculer la perte. La figure 3 présente mieux le processus de prédiction du prochain point et du nombre de points nécessaire pour cette prédiction. Les points orange sont ceux nécessaire pour obtenir le point vert avec 3 blocs causals.

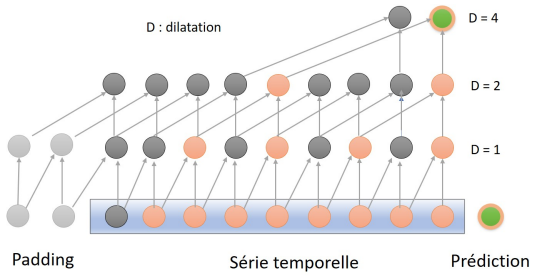

FIG. 3. Schéma expliquant la dépendance du vecteur de sortie avec le vecteur d'entrée. La zone bleuté constitue les données du réseau. Les pastilles grises claire correspondent au padding causal. Les pastilles orange sont nécessaire pour produire la verte. Elle a été dupliqué à la suite pour mettre en évidence que c'est la prédiction du prochain point.

LSTM

Dans les séries chronologiques l'observation à l'instant t_n dépend des observations à l'instant t_0 jusqu'à t_{n-1}. Justement LSTM permet de capturer cette dépendance chronologique et peut même apprendre à mémoriser et à oublier les dépendances nécessaires ainsi que ceux qui ne sont pas nécessaire à l'aide de ses "gate". La figure suivante illustre une séquence de LSTM [2].

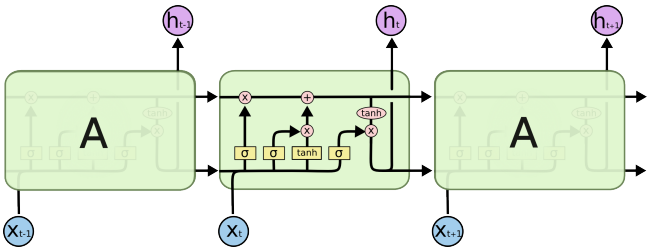

FIG. 4. Séquence de réseaux neuronaux récurrent où les couches cachées sont représenter par des LSTM

A l'entrée de LSTM les données se présentent comme $(B, T, 1)$ où B = Taille du batch, T = Longueur de la fenêtre. Lors de la propagation avant, on prend donc B fenêtres comme on le définis dans la section V puis on procède à la prédiction.

Les paramètres que l'on considère pour la recherche d'hyper-paramètres sont les suivants :

Hyper-paramètre	Explication
Fenêtre	Longueur de la séquence
Couche cachée	Nombre de couche cachée
Dropout	Taux de dropout

FIG. 5. Explication des hyper-paramètres de LSTM.

V. PRÉ-TRAITEMENT DES DONNÉES

Dans le projet nous n'avons pas considéré les corrélations existante entre les compagnies. Pour les entraînements on ne considère qu'une seule compagnie. De plus, on ne considère qu'un seul canal à la fois parmi les cinq présentés dans la section III. Les données sont centrés et normalisés pour SeriesNet et normalisés pour LSTM. Ceci facilite l'entraînement car sans ça, les gradients divergent très vite jusqu'à produire des "NaN" et stopper prématurément l'entraînement.

Pour entraîner les réseaux, on découpe la série chronologique d'intérêt en plusieurs sous-séries. Le procédés reviens à faire coulisser une fenêtre de taille k sur la suite et de prendre ces points comme l'entrée des réseaux et le point k+1 comme la cible. Le pas de déplacement est de 1 entre les fenêtre. La procédure est illustré dans la figure 6. La taille k des fenêtres dépend de la configuration des modèles. Avec LSTM, le nombre de point nécessaire est égal au nombre de cellules.

Pour SeriesNet, on a définit l'équation 1. Ceci limite le nombre de bloc utilisable. En effet, avec 11 blocs il faut 2048 points pour que la prédiction soit indépendante

du padding. Pour 10 blocs on a besoin de 1024 points et 512 pour 9 blocs. Nous disposons d'enregistrements de 1259 points. Les configurations à 11 blocs causal et plus sont d'office éliminés. On ne considère pas n'ont plus la configuration à 10 blocs car il n'y a plus assez de points pour l'entraînement. Au maximum, on a utilisé 9 blocs. De chaque enregistrement on retire un certain nombre de point pour valeur de test ($\approx$ 20 pour SeriesNet et une centaine pour LSTM). Lors du fenêtrage du dataset on garde un certain pourcentage du total de fenêtre et leurs cibles associées pour former l'ensemble d'évaluation. La position des fenêtres d'évaluation est choisis aléatoirement sur la série. On a put tester deux stratégies d'entraînement. La première est de garder entre les epochs l'ordre chronologique des données. La seconde est de mélanger après chaque epochs le dataset.

La procédure d'entraînement et de test diffère un peu. En effet, durant l'entraînement on prédit un seul point et on le compare avec la valeur cible pour calculer la loss. Lors du test on prédit un certain nombre de point d'affilié sans rétro-propagation. La première prédiction est accolé à la série chronologique et sert à prédire le prochain point. En adoptant cette stratégie on s'assure de s'entraîner sur des vrai valeur mais on a une certaine différence dans les procédure.

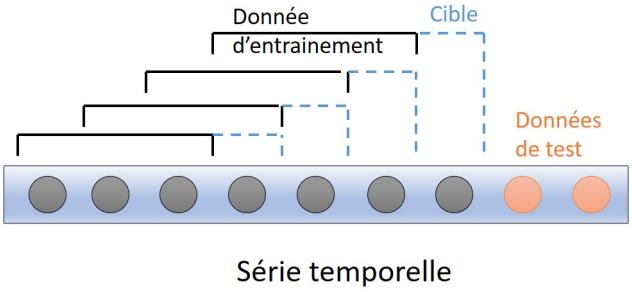

FIG. 6. Création du dataset avec le principe de fenêtre coulissante. Ici on représente des fenêtres contenant 3 points de la série temporelle (valeur en dessous de l'accolade). Le point tout de suite après sera conservé comme cible des trois points précèdent. Les données de test (orange) serve de cible dans le processus de test du modèle après entraînement.

VI. EXPÉRIMENTATION

SeriesNet

Dans cette section on s'est concentré sur la prédiction de la valeur de fermeture de l'action apple et Hologic. On a débuté par une recherche d'hyper-paramètres pour ces deux cas. Les différents paramètres sont présentés dans le tableau de la figure 7. Le choix du meilleur modèle est fait en prenant le minimum des moyenne des erreurs au carré sur les vingts derniers points de la courbe de l'action. Avec toutes les combinaisons c'est 270 modèles qui ont été exploré pour chacune des actions. La combinaison optimal pour l'action Apple est (8, *SGD*, 2, 32, 1 e^{-3}) en suivant le tableau 7 et pour Hologic (8, *SGD*, 3, 16, 1 e^{-3}).

Hyper-paramètre	Valeurs
Bloc causal	5, 6, 7, 8 9
optimiseur	'SGD' et 'Adam'
Bloc dropped	2, 3, 4
Filtre	16, 24, 32
Taux apprentissage	1 e^{-3}, 8 e^{-4}, 4 e^{-4}

FIG. 7. Tableau récapitulatif des hyper-paramètre utilisés lors de la recherche.

La figure 16 présente la prédiction faite après recherche d'hyper-paramêtre et entraînement avec ceux-ci. Les points orange dépendent des points bleus selon équation 1. On a prédit 20 points consécutivement comme expliqué dans la section V. Notre prédiction arrive à reproduire l'allure général. Cependant, le modèle n'est pas capable de suivre la double variation, l'augmentation puis la diminution de la courbe verte.

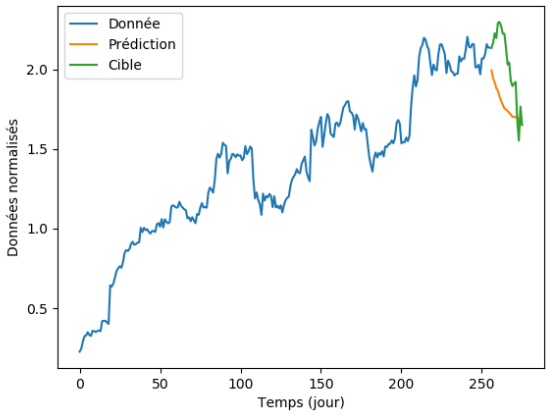

FIG. 8. Action Apple, paramètre du réseau : 8 blocs causal, 32 filtres, taux d'apprentissage 0.001, 2 bloc dropout et optimiseur SGD.

La courbe d'apprentissage 9 nous permet de dire qu'on a pas de sur-apprentissage. Il n'est pas nécessaire de s'entraîner sur plus d'époch dans cette configuration du modèle.

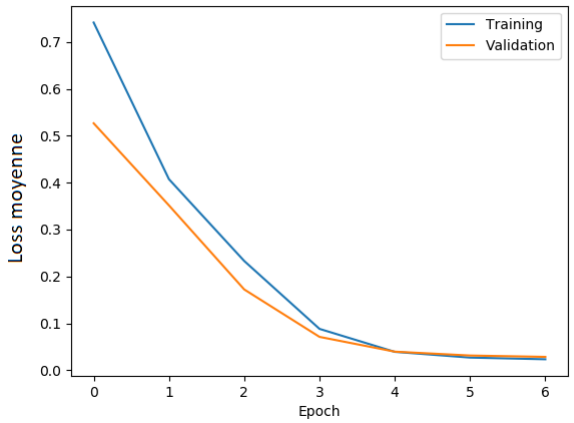

FIG. 9. Courbe d"apprentissage pour 7 epochs pour la figure 16.

On a mentionné dans la section V deux stratégies d'entraînement reposant sur le mélange des dataset d'entraînement. Ces deux méthodes sont comparés dans la figure 10. On voit une différence entre les courbes orange et verte. La courbe sans mélanger le dataset colle mieux aux données de test que celle avec mélange. En fait, il semble que la courbe verte reproduise mieux les variations que l'orange mais un décalage à la première valeur a corrompus le reste de la courbe.

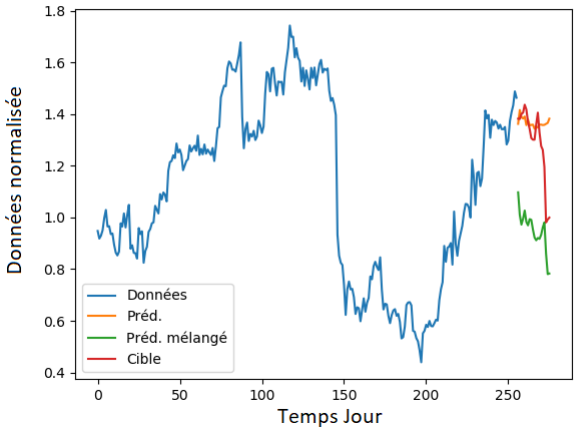

FIG. 10. Action Hologic, paramètre du réseau : 8 blocs causal, 16 filtres, taux d'apprentissage 0.001, 3 blocs dropout et optimiseur SGD. La différence entre prédiction et prédiction mélangé est que le dataset est mélangé entre chaque epoch.

Lors du développement du modèle SeriesNet, on a fait faces à un cas limites étrange. Lors de l'entraînement, le modèle tombe souvent dans un états où il prédits toujours la même valeur. C'est un problème tellement récurent, qu'on a dus ajouter un test dans la procédure de recherche d'hyper-paramètre pour détecter ces cas. Dans ces situations, on recommençais l'entraînement tant qu'on ne sortais pas cette situation. Ceci est problématique car augmente le temps d'évaluation. Il se peut que le bruit dans les courbes soit à l'origine de ce comportement. Une piste pour le mettre en évidence serait de faire un entraînement sur une fonction analytique comme une fonction trigonométrique (sin/cos) sans et avec un bruit blanc. L'environnement d'analyse serait plus contrôlé pour debogger le comportement du modèle.

LSTM

Arbitrairement, on prend un batch de 700, et une fenêtre de taille 60 jours. Après l'entraînement sur les données de Apple on aboutit à un résultat de prédiction comme montre la figure 11.

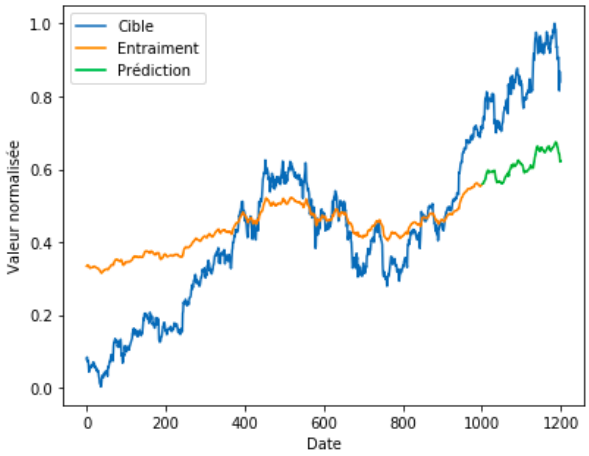

FIG. 11. Figure montrant l'entraînement et la prédiction sur les données de Apple pour un batch de taille 700, de fenêtre de 60 jours et 50 epochs

On voit bien qu'avec cette configuration LSTM arrive à capturer les petits variations mais a du mal reproduire le comportement qui s'écarte de la moyenne lorsqu'il ne s'est jamais rendu aussi haut. La figure ci-après montre la courbe d'entraînement ainsi que la courbe de validation de LSTM sur les données de Apple pour 48 epochs.

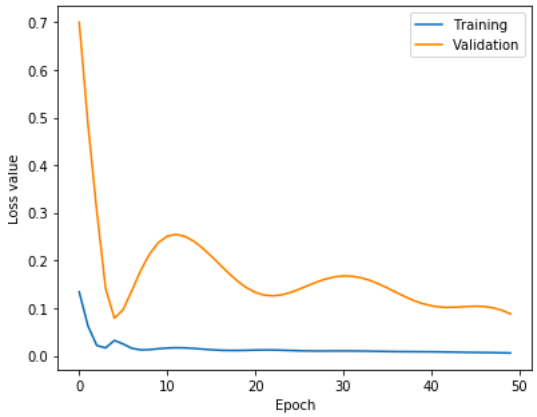

FIG. 12. Figure montrant la loss de LSTM sur les données de Apple pour un batch de taille 700, de fenetre de 60 jours et 50 epoch

Après 40 epoch le "training loss" ainsi que la "valiation loss" les deux tendent vers 0.

La remarque qu'on s'est fait est que plus on augmente la taille du batch et le nombre d'epoch plus l'entraînement est bon. La figure suivante illustre les résultats si la taille du batch est égal à 800 et le nombre d'epoch égal à 80.

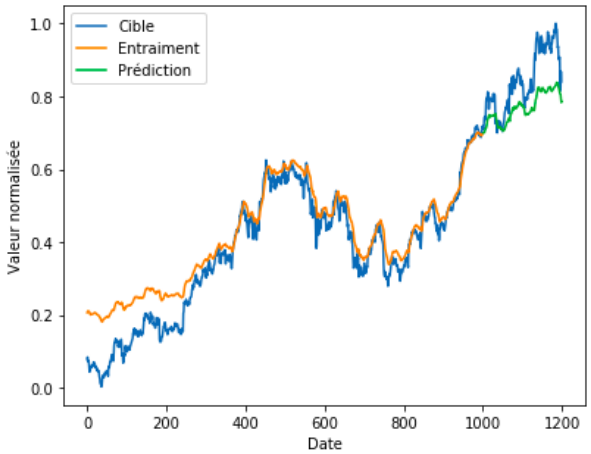

FIG. 13. Figure montrant la prédiction de LSTM sur les données de Apple pour un batch de taille 800, de fenêtre de 60 jours et 100 epochs

La figure suivante montre la loss lorsqu'on choisit cette configuration.

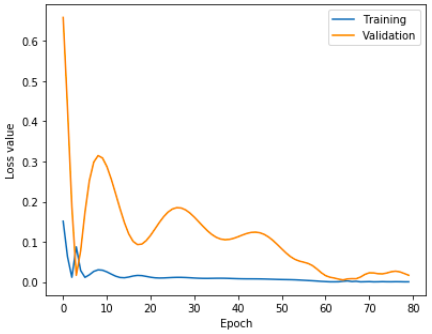

FIG. 14. Figure montrant la loss de LSTM sur les données de Apple pour un batch de taille 800, de fenêtre de 60 jours et 100 epoch

On voit bien que les deux loss convergent vers 0 après 60 epochs. Les oscillations sont curieuse et on ne s'y attends pas. On pourrait interpréter ça comme une sorte de sur-apprentissage périodique du modèle.

On a testé le mélange du dataset entre chaque epochs dans la figure suivante. Ceci semble améliorer l'entraînement surtout au début de la courbe comparé à la figure 13.

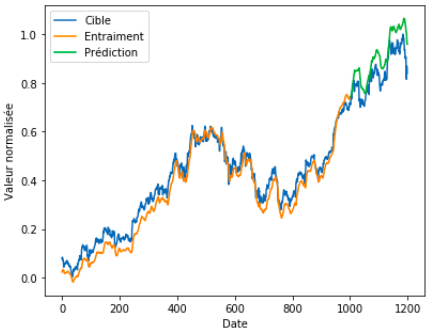

FIG. 15. Figure montrant la loss de LSTM sur les données de Apple lorsqu'on mélange les dataset

et une loss comme suit :

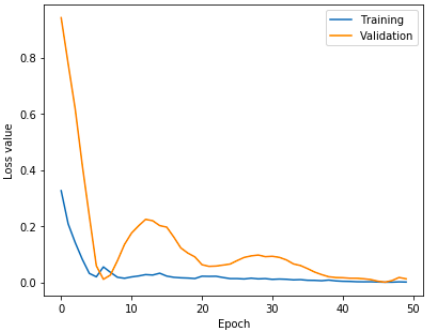

FIG. 16. Figure montrant la loss de LSTM sur les données de Apple lorsqu'on mélange les dataset

ARIMA

On a aussi utilisé ARIMA comme "baseline" model vu que ARIMA est très utilisé dans les séries chronologiques. Puisque les données ne sont pas stationnaires, on a transformé les données pour les rendre stationnaires avant de faire l'entraînement.

La figure suivante montre le résultat d'entraînement en utilisant ARIMA.

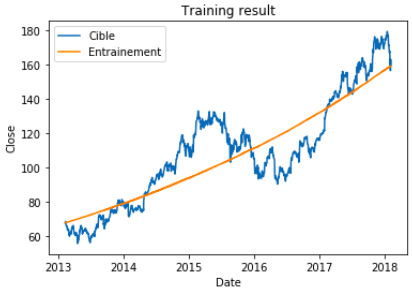

FIG. 17. Figure montrant l'entraînement à l'aide de ARIMA sur les données de Apple

Enfin, nous avons comparé LSTM avec ARIMA sur un même graphique pour voir l'efficacité l'un par rapport à l'autre.

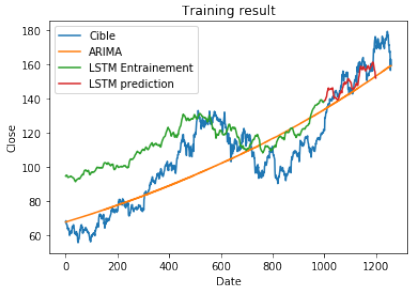

FIG. 18. Figure comparant la performance de ARIMA avec LSTM sur les données de Apple

Sur le graphique, on peut lire que LSTM surpasse les performance de ARIMA au niveau de la prédiction.

VII. CONCLUSION

Nous avons développé deux méthodes de Deep learning pour l'analyse de série chronologique. Les deux modèles semblent retrouver des patrons récurrents dans les données et semble prometteur pour ce type d'analyse comparé à la technique classique ARIMA qui produit une courbe de tendance général plutôt que de la prédiction précise. Nous avons exploré deux protocoles d'entraînement avec ou sans mélanges des datasets entre chaque epochs. On observe des différences.

VIII. CODE

Le code a été répertorié sur la plate-forme github tout au long du développement. Le lien suivant permet de trouver ce répertoire.

REFERENCES

[1] URL: `https://www.kaggle.com/camnugent/sandp500`.

[2] Colah. *Understanding LSTM Networks*. URL: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[3] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. 2016. arXiv: `1609.03499 [cs.SD]`.

[4] Krist Papadopoulos. *SeriesNet: a dilated causal convolutional neural network for forecasting*. URL: `https://github.com/kristpapadopoulos/seriesnet/blob/master/seriesnet-Krist-Papadopoulos-v1.pdf`.

[5] Z. Shen et al. "SeriesNet:A Generative Time Series Forecasting Model". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8.