État de l’art

Prédiction à l’aide de séries chronologiques et réseaux de neurones réccurents

Thomas Chaigneau

May 24, 2022, Brest

Contents

1 Etat de l'art

Dans cette première section, nous allons décrire les avancées de la modélisation IA et des algorithmes de prédiction sur des données temporelles. Ce ne sera malheureusement pas une liste exhaustive de toutes les options disponibles, ni un historique complet des différentes évolution des algorithmes de prédiction sur des séries temporelles, car cela est trop riche pour figurer dans ce rapport.

Nous allons donc nous concentrer sur les avancées les plus récentes et celles qui ont un rapport direct avec la solution envisagée dans ce projet. Commençons par un bref aperçu et historique de la tâche de prédiction à l'aide de données temporelles.

1.1 Aperçu et historique rapide

Les séries temporelles, ainsi que leur analyse, sont de plus en plus importantes en raison de la production massive de données dans le monde. Il y a donc un besoin, en constante augmentation, dans l'analyse de ces séries chronologiques avec des techniques statistiques et plus récemment avec apprentissage automatique.

L'analyse des séries chronologiques consiste à extraire des informations récapitulatives et statistiques significatives à partir de points classés par ordre chronologique. L'intérêt étant de diagnostiquer le comportement passé pour prédire le comportement futur d'une série donnée.

Aucune des techniques ne s'est développée dans le vide ou par intérêt purement théorique. Les innovations dans l'analyse des séries chronologiques résultent de nouvelles méthodes de collecte, d'enregistrement et de visualisation des données (Nielsen 2019).

Il existe énormément de domaine d'application tels que la médecine, la météorologie, l'astronomie ou encore ce qui va nous intéresser ici, les marchés financiers et notamment celui des crypto-monnaies.

Pour revenir à l'aspect historique, les organisations privées, et notamment bancaires, ont commencé à collecter des données par imitation du gouvernement américain qui collectait des données économiques publiques. Les pionniers de l'analyse des données chronologiques des cours de la bourse ont fait ce travail mathématique à la main, alors que de nos jours ce travail est réalisé avec l'assistance de méthodes analytiques et des algorithmes de *Machine Learning* (Nielsen 2019).

Richard Dennis, dans les années 80, a été le premier à développer un algorithme de prédiction des cours de la bourse qui ne comprenait que quelques règles de base, permettant à quiconque les connaissant de prévoir le prix d'une action et d'en retirer des bénéfices par spéculation.

Progressivement, avec l'accumulation de personnes utilisant ces règles, elles sont devenues de plus en plus inefficaces. Il aura donc fallu développer de nouvelles méthodes, notamment statistiques, plus complexes pour toujours mieux prévoir l'évolution des cours des marchés financiers.

C'est ainsi que des méthodes historiques telles que *SARIMA* ou *ARIMA* se sont démocratisées. Elles présentent néanmoins un inconvénient : elles nécessitent des données stationnaires pour fonctionner. De plus, ces techniques statistiques dites historiques ont des résultats médiocres sur le long terme. C'est pourquoi ce sont développés d'autres méthodes d'apprentissage automatique utilisant la puissance des réseaux de neurones, comme *RNN* (Recurrent Neural Network) (Rumelhart and Williams 1985).

1.2 Fondamentaux des séries temporelles

Comme évoqué précédemment, la stationnarité d'une série est une propriété essentielle pour l'analyse statistique. Une série chronologique est dite stationnaire si ces propriétés telles que la moyenne, la variance ou la covariance sont constantes au cours du temps. Or, cela n'est pas vrai pour toutes les séries temporelles et notamment les données issues des marchés financiers, qui ne sont stationnaires que sur une période de temps fixée (souvent courte).

Il existe trois composantes qui constituent une série temporelle :

- **Tendance (T = Trend)** : correspond à une augmentation ou à une diminution sur le long terme des données et qui peut assumer une grande variété de modèles. Nous utilisons la tendance pour estimer le niveau, c'est-à-dire la valeur ou la plage typique de valeurs, que la variable doit avoir au cours du temps. On parle de tendance à la hausse ou à la baisse.
- **Saisonnalité (S = Seasonal)** : est l'apparition de schémas de variations cycliques qui se répètent à des taux de fréquence relativement constants.
- **Résidu (R = Remainder)** : corresponde aux fluctuations à court terme qui ne sont ni systématiques ni prévisibles. Au quotidien, des événements imprévus provoquent de telles instabilités. Concrètement, la composante résiduelle est ce qui reste après l'estimation de la tendance et de la saisonnalité, et leur suppression d'une série chronologique.

Voici une représentation de la décomposition des composantes d'une série temporelle :

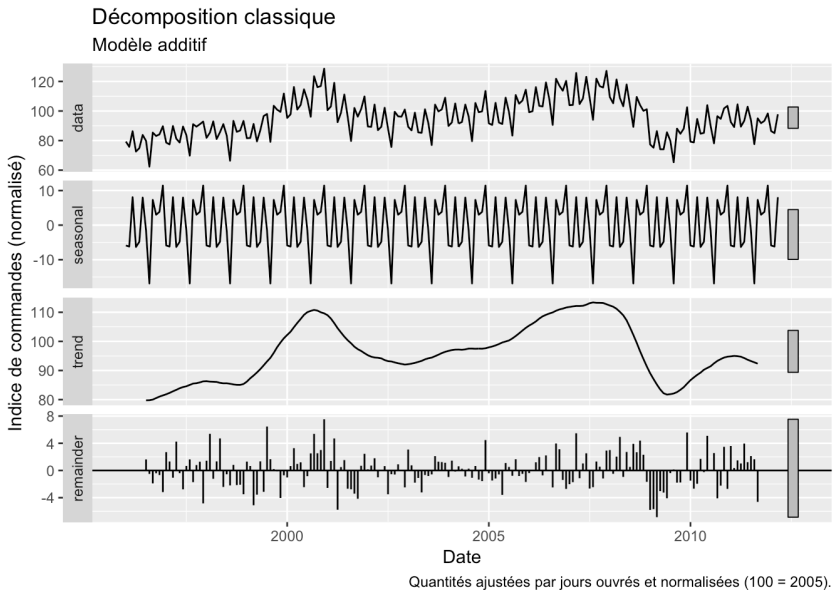


Figure 1: Graphique présentant la décomposition classique d'un modèle additif d'une série temporelle (Chailan and Palacios-Rodríguez 2018)

1.3 Différents modèles de prédiction

Nous allons voir ici les différentes techniques et modèles qui existent pour la prédiction à l'aide de données temporelles.

Nous pouvons d'ores et déjà distinguer deux catégories de modèles de prédiction :

- **Modèles statistiques** : dits traditionnels, ils regroupent les modèles univariés et multivariés comprenant respectivement *ARIMA*, *SARIMA* et *VAR*.

Un processus stationnaire X_t admet une représentation $ARIMA(p, d, q)$ dite minimale s'il existe une relation (Goude 2020) :

$$\Phi(L)(1-L)^d X_t = \Theta(L)\epsilon_t, \forall_t \in Z$$

avec pour conditions :

- $\phi_p \neq 0$ et $\theta_q \neq 0$
- Φ et Θ doivent être des polynômes de degrés respectifs p et q, n'ont pas de racines communes et leurs racines sont de module > 1
- ϵ_t est un BB de variance σ^2

De même, un processus stationnaire X_t admet une représentation $SARIMA(p, d, q)$ dite minimale si la relation suivante est vraie (Goude 2020) :

$$(1 - L)^d \Phi_p(L)(1 - L^s)^D \Phi_P(L^s) X_t = \theta_q(L)\theta_Q(Ls)\epsilon_t, \forall_t \in Z$$

avec les mêmes conditions que pour les modèles ARIMA.

- **Modèles d'apprentissage automatique** : ils regroupent les modèles de régression par amplification de gradient et les modèles par apprentissage profond comprenant les réseaux de neurones récurrents et convolutionnels.

Voici une représentation des différents modèles de prédiction :

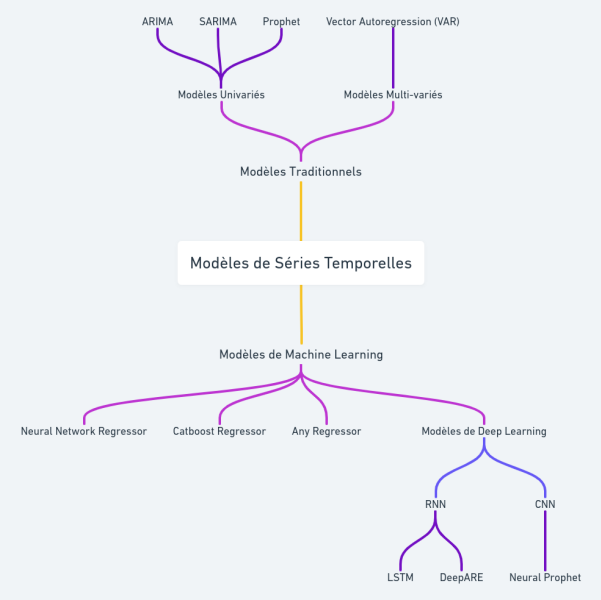


Figure 2: Liste non-exhaustive des modèles utilisés pour la prédiction de séries chronologiques.

Nous pourrions également ajouter à cette liste les très récents modèles basés sur l'architecture *Transformers*, comme **Temporal Fusion Transformers** (TFT) qui est un modèle de Google (Lim et al. 2019) permettant de combiner des données temporelles avec des données non temporelles, des données statiques comme des informations de localisation dans le cas de prédictions météorologiques (Kafritsas 2021).

Dans notre projet, nous allons nous concentrer sur les modèles de *Machine Learning* les plus récents, qui sont les modèles de *Deep Learning* tels que les architectures réseaux de neurones convolutionnels et réseaux de neurones récurrents.

1.4 Réseaux de neurones récurrents (*RNN*)

Les réseaux de neurones récurrents, ou *Recurrent Neural Network*, sont des architectures de neurones qui sont utilisés dans beaucoup de cas d'usage. Ils sont appelés réseaux de neurones récurrents car ils sont capables de se réguler en fonction de la sortie des neurones précédents (Sherstinsky 2020). Ils sont notamment utilisés pour la prédiction de séries temporelles, car ils permettent de prédire la valeur d'une variable à partir de ses valeurs précédentes. Ils sont également à l'origine des architectures *Transformers* notamment pour les tâches de traduction en *NLP*.

1.4.1 Architecture du réseau de neurones récurrents

C'est par le biais d'états cachés (en anglais *hidden states*) qu'un modèle *RNN* est capable de réaliser la prédiciton d'une variable. Ainsi, un modèle *RNN* prend en entrée des séquences de vecteurs de données, et non pas des vecteurs de données individuels.

Une architecture traditionnelle d'un *RNN* se présente comme suit (Amidi and Amidi 2020) :

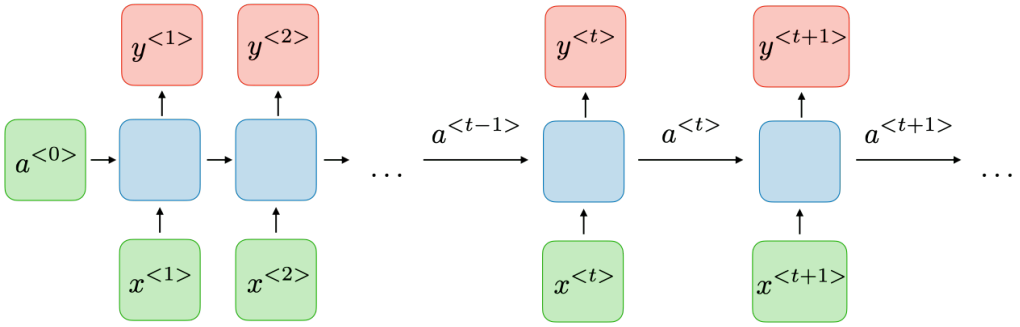


Figure 3: Architecture d'un réseau de neurones récurrents (*RNN*)

À l'instant *t*, l'activation $a^{<t>}$ d'un neurone est définie par la fonction d'activation suivante :

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

et la sortie $y^{<t>}$ est de la forme :

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

où $W_{ax}, W_{aa}, W_{ya}, b_a$ et b_y sont des coefficients de poids partagés temporellement entre les fonctions d'activation g_1 et g_2.

Il est également intéressant de s'intéresser à l'architecture d'une cellule (en anglais *block*) qui compose un réseau de neurones récurrents. Cela permet de comprendre les mécanismes qui ont lieu à chaque étape de la propagation de données dans un réseau *RNN*. Ce sera également utile dans un second temps pour pouvoir comparer les différences majeures avec des architectures plus évoluées que les versions basiques, que nous verrons juste après.

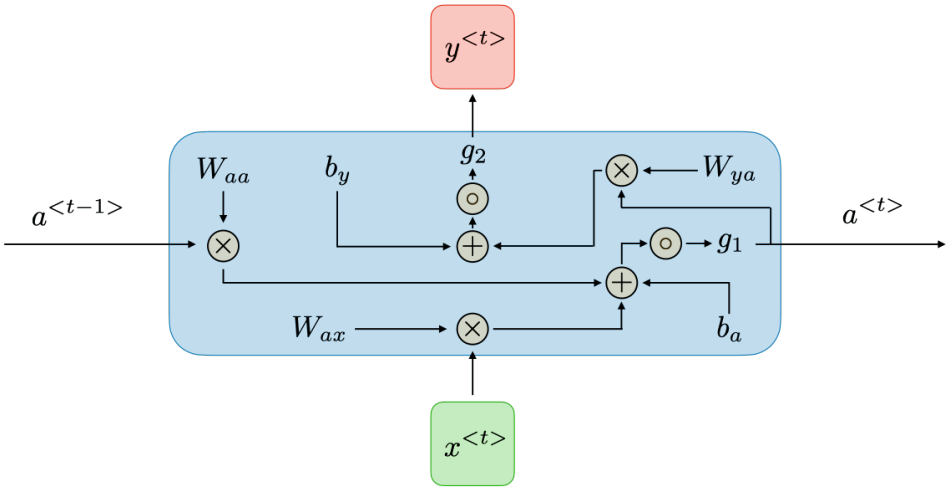


Figure 4: Architecture d'une cellule d'un réseau de neurones récurrents (*RNN*) (Amidi and Amidi 2020)

Nous pouvons voir que chaque cellule va prendre un état de la donnée précédente, et qu'elle va produire une sortie en fonction de son état et de la fonction d'activation associée.

1.4.2 Avantages et inconvénients

Voici un tableau récapitulatif des avantages et inconvénients d'utiliser ce genre de modélisation (Amidi and Amidi 2020) :

Table 1: Avantages et inconvénients des modèles *RNN*

Avantages	Inconvénients
- Possibilité de traiter une entrée de n'importe quelle longueur	- Le calcul est plus consommateur en ressources (par rapport à d'autres modèles)
- La taille du modèle n'augmente pas avec la taille de l'entrée	- Difficulté pour accéder aux informations trop lointaines

Avantages	Inconvénients
- Le calcul prend en compte les informations antérieures	- Impossibilité d'envisager une entrée future pour un état donné
- Les coefficients sont indépendants du temps	

Nous pouvons constater que comme attendu lors de l'utilisation de modèles de *Deep Learning*, les modèles *RNN* sont plus consommateurs en ressources que d'autres modèles de *Machine Learning*. Ils présentent néanmoins des avantages non négligeables, en dehors d'un gain de performances, pour notre cas d'usage dans le cadre de la prédiction de séries temporelles financières.

1.4.3 Gestion des gradients

Il existe également un autre inconvénient des modèles *RNN* qui sont les phénomènes de gradients qui disparaissent et qui explosent lors de l'apprentissage. En anglais, on parle de *vanishing gradient* et *exploding gradient*.

Cela est expliqué par le fait que sur le long terme il est très difficile de capturer les dépendances à cause du gradient multiplicatif qui peut soit décroître, soit augmenter de manière exponentielle en fonction du nombre de couches du modèle.

1.4.3.1 Cas de gradient qui explose

Pour contrer les phénomènes de gradient qui explose, il est possible d'utiliser une technique de *gradient clipping* (Sherstinsky 2020) (en français *coupure de gradient*) qui permet de limiter le gradient à une valeur fixée. Puisque la valeur du gradient est plafonnée les phénomènes néfastes de gradient sont donc maîtrisés en pratique.

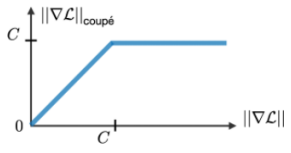


Figure 5: Technique de gradient clipping

Grâce à cette technique, nous pouvons donc éviter que le gradient devienne trop important en le remettant à une échelle plus petite.

1.4.3.2 Cas de gradient qui disparaît

Concernant les phénomènes de gradient qui disparaissent, il est possible d'utiliser des *portes* de différents types, souvent notées Γ et qui sont définies par :

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

où W, U et b sont des coefficients spécifiques à la porte et σ est une fonction sigmoïde.

Les portes sont utilisées dans les architectures plus spécifiques comme *GRU* et *LSTM* que nous verrons juste après.

Table 2: Comparaison des différents types de portes et leurs rôles

Type de porte	Rôle	Utilité
Porte d'actualisation Γ_u	Décide si l'état de la cellule doit être mis à jour avec la valeur d'activation en cours	GRU, LSTM
Porte de pertinence Γ_r	Décide si l'état de la cellule antérieure est important ou non	GRU, LSTM
Porte d'oubli Γ_f	Contrôle la quantité d'information qui est conservée ou oubliée de la cellule antérieure	LSTM
Porte de sortie Γ_o	Détermine le prochain état caché en contrôlant quelle quantité d'information est libérée par la cellule	LSTM

Ces différents types de portes permettent de corriger les erreurs de calcul du gradient en fonction de la mesure de l'importance du passé, et ainsi de s'affranchir en partie des phénomènes de gradient qui disparaissent. Il est important de noter que les portes d'oubli et de sortie sont utilisées uniquement dans les architectures *LSTM*. *GRU* dispose donc de deux portes, alors que *LSTM* dispose de quatre portes.

1.4.4 Fonctions d'activation

Il existe trois fonctions d'activation qui sont utilisées dans les modèles *RNN* :

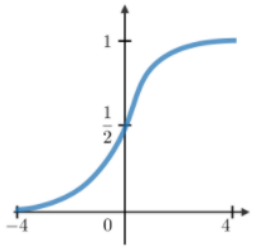
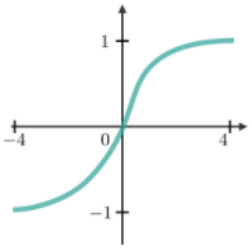
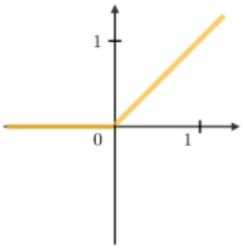
Sigmoïde	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

Figure 6: Fonctions d'activation communément utilisées et leurs représentations (Amidi and Amidi 2020)

- La *fonction d'activation sigmoïde* représente la fonction de répartition de la loi logistique, souvent utilisée dans les réseaux de neurones, car elle est dérivable.
- La *fonction d'activation tanh*, ou *tangente hyperbolique*, représente la fonction de répartition de la loi hyperbolique.
- La *fonction d'activation RELU*, ou *rectified linear unit,* représente la fonction de répartition de la loi linéaire.

Le rôle d'une fonction d'activation est de modifier de manière non-linéaire les valeurs de sortie des neurones, ce qui permet de modifier spatialement leur représentation. Une fonction d'activation est donc définie et spécifique pour chaque couche du réseau de neurones. Il ne faut pas confondre avec les fonctions de *loss* qui sont utilisées pour déterminer la qualité de l'apprentissage et sont quant à elles uniques, c'est-à-dire que l'on doit définir une unique fonction de loss pour chaque modèle.

1.4.5 *GRU* et *LSTM*

Nous pouvons distinguer les unités de porte récurrente (en anglais *Gated Recurrent Unit*) (*GRU*) et les unités de mémoire à long/court terme (en anglais *Long Short-Term Memory*) (*LSTM*). Ces deux architectures sont très similaires et visent à atténuer le problème de gradient qui disparaît, rencontré avec les *RNNs* traditionnels lors de l'apprentissage. *LSTM* peut être vu comme étant une généralisation de *GRU* en utilisant des cellules de mémoire à long ou court terme.

Pour comprendre les différences fondamentales entre les deux architectures, il est nécessaire de regarder en détails les différentes équations utilisées par chacune d'elles.

1.4.5.1 Gated Recurrent Unit (*GRU*)

Comme nous l'avons vu précédemment, l'architecture *GRU* comporte deux portes : une porte d'actualisation Γ_u (en anglais *update gate*) et une porte de pertinence Γ_r (en anglais *reset gate*).

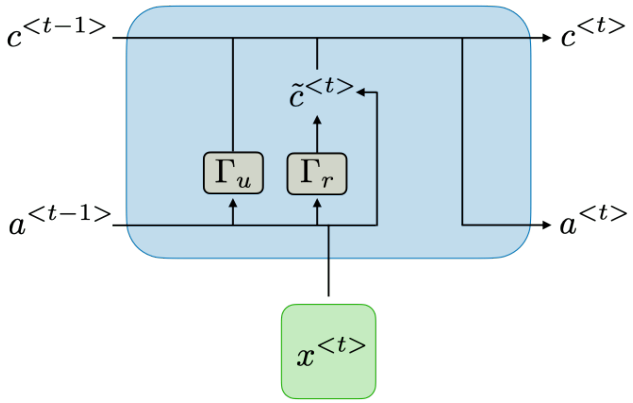


Figure 7: Architecture d'une unité de *GRU* (Amidi and Amidi 2020)

Il faut discerner trois composantes importantes pour la structure de l'unité de *GRU* :

- La cellule candidate $c^{<t>}$, où $c^{<t>} = tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$
- L'état final de la cellule $c^{<t>}$, où $c^{<t>} = \Gamma_u \star c^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$

  L'état final de la cellule est calculé par la somme des produits de la porte d'actualisation Γ_u et de la valeur de la cellule candidate $c^{<t>}$ et de l'inverse de la porte d'actualisation $1 - \Gamma_u$ multiplié par la valeur de l'état final de la cellule antérieure c^{t-1}.

  Cet état final de la cellule est donc dépendant de la porte d'actualisation Γ_u et peut soit être mis à jour avec la valeur de la cellule candidate $c^{<t>}$ ou soit conservé la valeur de l'état final de la cellule antérieure.
- La fonction d'activation $a^{<t>}$, où $a^{<t>} = c^{<t>}$

1.4.5.2 Long Short-Term Memory (*LSTM*)

Maintenant que nous avons vu plus en détails l'architecture générale des *RNNs*, ainsi que les particularités de *GRU*, il est temps d'aborder en détails les particularités de l'architecture de *LSTM*, qui sera l'architecture choisie pour le projet final.

En plus des deux portes d'actualisation et de pertinence, *LSTM* intègre une porte d'oubli Γ_f et une porte de sortie Γ_o.

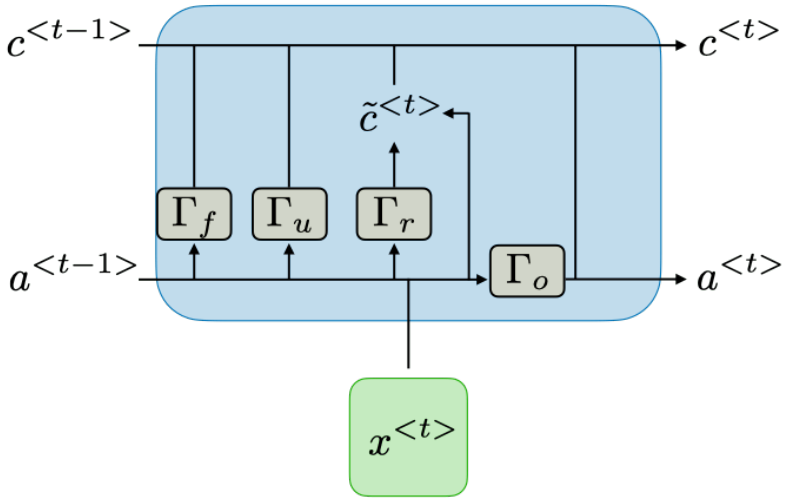


Figure 8: Architecture d'une unité de *LSTM* (Amidi and Amidi 2020)

L'architecture *LSTM* est similaire à l'architecture *GRU*, mais permet de gérer le problème de gradient qui disparaît. Ainsi, aux trois composantes de l'unité de *GRU* que nous avons vu précédemment, il y a deux différences :

- La fonction d'activation $a^{<t>}$ est désormais multipliée par la porte de sortie Γ_o, ce qui permet de contrôler la quantité d'information qui est libérée par la cellule. On a donc : $a^{<t>} = \Gamma_o \star c^{<t>}$
- L'état final de la cellule $c^{<t>}$ est désormais influencé par la porte d'oubli Γ_f qui devient un facteur de la valeur de l'état final de la cellule antérieure c^{t-1}. En agissant ainsi, la porte d'oubli permet de réguler la quantité d'information retenue de la cellule antérieure. Il y a donc un choix sur ce qui est conservé et oublié. On a ainsi : $c^{<t>} = \Gamma_f \star c^{t-1} + \Gamma_f \star c^{<t-1>}$

1.5 L'avènement des modèles Transformers

Le premier papier de recherche qui présente les modèles *Transformers* est *Attention Is All You Need* (Vaswani et al. 2017), présenté en juin 2017. L'objectif initial de ce genre d'architecture, qui vient ajouter un mécanisme d'attention aux *RNNs*, était d'améliorer les performances des modèles existants sur les tâches de traduction en *NLP*.

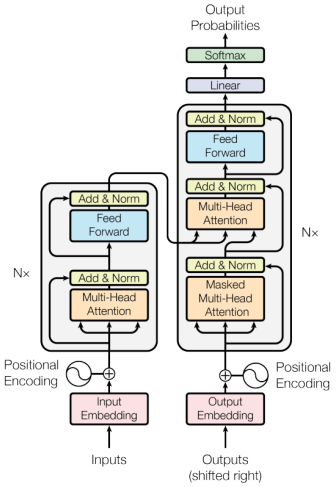


Figure 9: Architecture d'un modèle Transformer (Vaswani et al. 2017)

Nous ne détaillerons pas ici les différentes composantes de cette architecture, mais ce qu'il faut retenir c'est que par l'utilisation d'un encodeur et d'un décodeur, couplés au mécanisme d'attention, il est possible d'entraîner des modèles de manière non-supervisée et surtout d'obtenir des performances encore jamais atteintes par les modèles existants.

L'entraînement de ces modèles de manière non-supervisée est très simple et permet dans un second temps le *fine-tuning* des modèles avec beaucoup moins de données sur des tâches précises. Ainsi, un même modèle de base peut être entraîné sur différentes tâches et produire des modèles finaux à la pointe des performances des modèles existants.

L'essor de cette architecture a principalement eu lieu dans le domaine du traitement du langage naturel avec des modèles très célèbres tels que *BERT* (Devlin et al. 2018), *GPT-2* (Radford et al. 2019) ou encore *T5* (Raffel et al. 2019). Mais aujourd'hui, il existe également des applications dans les domaines de la vision par ordinateur (*Computer Vision*), des séries temporelles (*Time Series*) ou encore dans le traitement du signal audio (*Audio Processing*).

Cette architecture passionnante est en train de devenir un modèle de référence pour énormément de projet de recherche et cela dans quasiment tous les domaines du *Machine Learning*. Nous ne détaillerons pas plus ici puisque nous n'avons pas fait usage de cette architecture dans **Make Us Rich**, et cela nécessiterait un dossier à part entière tellement il y a de choses à préciser.

Maintenant que nous avons un meilleur aperçu des modèles de référence, et des détails de l'architecture des modèles *RNN*, et plus particulièrement des modèles *LSTM*, nous allons pouvoir présenter le projet **Make Us Rich** qui vise à automatiser l'entraînement et le service de modèles *LSTM* pour de la prédiction sur des données financières.

Amidi, Afshine, and Shervine Amidi. 2020. "Recurrent Neural Networks Cheatsheet Star." *CS 230 - Recurrent Neural Networks Cheatsheet*. Stanford.edu. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

Chailan, Romain, and F. Palacios-Rodríguez. 2018. "TP-Timeseries Novembre 2018." *Rchailan.github.io*. https://rchailan.github.io/assets/lectures/timeseries/tp_timeseries.html.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." arXiv. https://doi.org/10.48550/ARXIV.1810.04805.

Goude, Yannig. 2020. "Les Processus Arima." Paris-Saclay. https://www.imo.universite-paris-saclay.fr/~goude/Materials/time_series/cours6_ARIMA.pdf.

Kafritsas, Nikos. 2021. *Temporal Fusion Transformer: Time Series Forecasting with Interpretability*. https://towardsdatascience.com/temporal-fusion-transformer-googles-model-for-interpretable-time-series-forecasting-5aa17beb621.

Lim, Bryan, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. 2019. "Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting." arXiv. https://doi.org/10.48550/ARXIV.1912.09363.

Nielsen, Aileen. 2019. *Practical Time Series Analysis*. O'Reilly Media.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. *Language Models Are Unsupervised Multitask Learners*. OpenAI. https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." arXiv. https://doi.org/10.48550/ARXIV.1910.10683.

Rumelhart, Geoffrey E, David E; Hinton, and Ronald J Williams. 1985. "Learning Internal Representations by Error Propagation," September. https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf.

Sherstinsky, Alex. 2020. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network." *Physica D: Nonlinear Phenomena* 404 (March): 132306. https://doi.org/10.1016/j.physd.2019.132306.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." arXiv. https://doi.org/10.48550/ARXIV.1706.03762.