Seq2seq :Depuis les réseaux de neurones vers deux

applications phares



**Sommaire**

[**Introduction**](#_146zvr2l7l1u) **3**

[**1 - Le modèle Seq2Seq**](#_6sihqdnibbjq) **4**

[Réseau de neurones récurrents (RNN)](#_p92h14dp1aad) 4

[Long short-term memory (LSTM)](#_vsffaflg0ceu) 4

[Encoder](#_ln5kah9m10qp) 5

[Decoder](#_kpy5k18ejsfh) 5

[**2 - Kaggle, Web Traffic Time Series Forecasting**](#_4vla9hx4mslp) **6**

[Contexte de la compétition](#_86qugrbzqqqg) 6

[Modèle - Torch](#_bc8i4umupdrm) 7

[Modèle - Keras](#_8k6jruv6mhr4) 8

[**3 - Une autre application, la calcination**](#_jln5tlu0aikm) **8**

[**Conclusion**](#_ryca9fc7urpe) **8**

[**Bibliographie**](#_1q17giifhef4) **9**

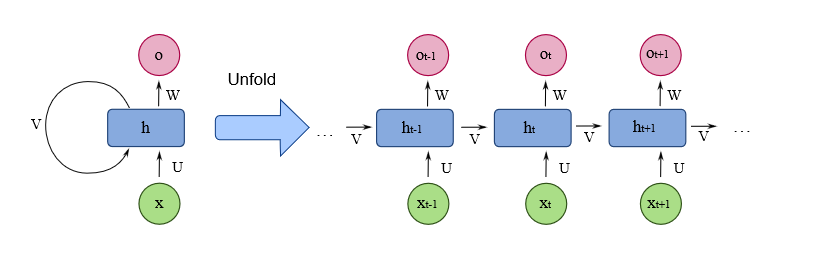
# Introduction

# 1 - Le modèle Seq2Seq

Le modèle Seq2seq est un réseau neuronal qui va convertir une séquence de données en entrée vers une nouvelle séquence de données en sortie. Il est utilisé dans de nombreux domaines d’application tels que celui du traitement automatique du langage naturel (NLP) ou encore pour l’analyse de séries temporelles. Ce modèle est composé de deux couches de réseaux de neurones récurrents (RNN) : l’encoder et le decoder, et la séquence d’entrée et celle de sortie n’ont pas forcément la même longueur.

## **Réseau de neurones récurrents (RNN)**

Le réseau de neurones récurrents (RNN) est un réseau d’unités reliées par des arcs possédant des poids. Ce réseau possède des connexions récurrentes puisque l’information est conservée en mémoire en prenant en compte les états précédents. C’est pour cette raison que les RNN sont particulièrement adaptés à des données faisant intervenir un contexte comme des suites de données.



Ce réseau ne peut cependant mémoriser qu’une cinquantaine d’itérations et donc rester sur un passé proche, c’est le problème de disparition de gradient. Les réseaux Long short-term memory (LSTM) et les réseaux Gated Recurrent Unit (GRU) vont alors pouvoir répondre à ce problème. Pour notre modèle nous allons utiliser des LSTM.

## **Long short-term memory (LSTM)**

Un LSTM est composé de trois portes (forget gate, input gate et output gate) et de deux sorties (hidden state, cell state). Les portes peuvent être vus comme des zones de calculs et ceux-ci s’occupent du [traitement des données d’entrée](https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/). Les LSTM vont à la fois passer entre chaque étape la prédiction mais aussi un autre terme, l’état de la cellule (cell state), qui contient des information plus globales et moins liées à l’entrée courante, auquel des informations peuvent être ajoutées au fur et à mesure. Le celle state fournit donc des informations sur notre séquence dans sa globalité et donc le problème de rester uniquement sur un passé proche n’est plus.

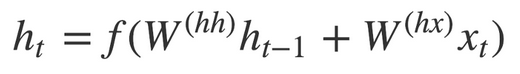
Notre modèle nécessite deux réseaux de neurones pour l’encoder et le decoder. Chaque réseau utilisera donc un LSTM.

## **Encoder**

L’encoder va récupérer le *contexte* de la séquence en entrée sous la forme d’un *vecteur d’état caché*, ainsi que la valeur d’entrée pour générer un nouveau vecteur d’état caché et l’envoyer au decoder pour produire la séquence en sortie.

En général il sera composé de réseaux de type RNN comme LSTM (Long Short-Term Memory) ou GRU (Gated Recurrent Units).

Chaque état caché sera calculé avec la formule :

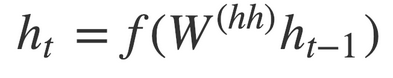


avec ht l’état caché, W le poids et xt le vecteur d’entrée

L’état caché final en sortie de l’encoder est donc un vecteur qui sera en même temps état caché initial du decoder. Ce vecteur va encapsuler les informations de l’ensemble des éléments en entrée pour permettre au decoder de faire ses prédictions.

## **Decoder**

La couche decoder prend en entrée le vecteur généré à la sortie de l’encoder. Elle sera également créée avec les architectures RNN (LSTM, GRU). Chaque unité récurrente prendra un état caché précédent en entrée et produira un autre état caché en sortie, selon la formule :



avec ht-1 l’état caché précédent et W le poids.

Le vecteur de sortie sera alors calculé avec la formule suivante :



avec Ws leur poids respectif

La séquence en sortie va alors énormément dépendre du contexte défini par l’état caché à la sortie de l’encoder. Il est donc possible de perdre le contexte initial si la séquence devient trop longue.

# 2 - Kaggle, Web Traffic Time Series Forecasting

Maintenant que nous avons notre modèle au complet, nous allons l'utiliser sur les données d’une compétition kaggle : [“Web Traffic Time Series Forecasting”](https://www.kaggle.com/c/web-traffic-time-series-forecasting).

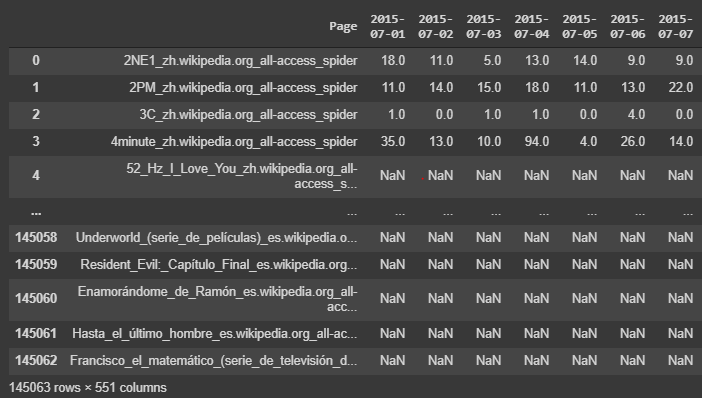
## **Contexte de la compétition**

Le but de cette compétition est de prédire le nombre de visites sur des articles de pages Wikipédia. Ce problème fait partie de la prédiction des valeurs des séries temporelles.

Nous avons à notre disposition différents fichiers :

* train\_1.csv et train\_2.csv : contiennent les données de fréquentation avec un article par ligne et une date par colonne. Le nom de la page Wikipédia est sous la forme “nom”\_“pays.wikipedia.org”\_”type d’accès”\_”type d’agent”.
* key\_1.csv et key\_2.csv : donnent la correspondance entre le nom des pages et leur id, qui nous sera utile pour générer nos soumissions
* sample\_submission\_1.csv et sample\_submission\_2.csv : montrent le format des soumissions pour la compétition Kaggle

Les données d’entraînement ont environ 145k séries temporelles. Chacune correspond à un nombre de visites par jour sur un article Wikipédia. Dans notre cas nous ne traitons que train\_1.csv :

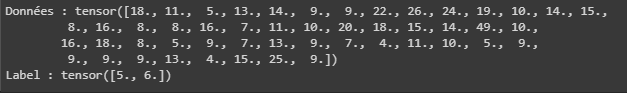


Nous avons donc le nombre de vues sur 145063 articles Wikipédia sur 550 jours, du 1er Juillet 2015 au 31 Décembre 2016.

## **Modèle - Torch**

Dans un premier temps nous avons décidé d’implémenter notre modèle avec Torch sur google colab pour pouvoir utiliser leur GPU.

Nous avons formaté nos données grâce aux aux élément Dataset et Dataloader de Torch. Cela permet de mettre en forme et de customiser nos données plus facilement. Nous avons au final un élément qui représente X nombres de visites associé à un label qui représente les Y nombres suivants de visites.



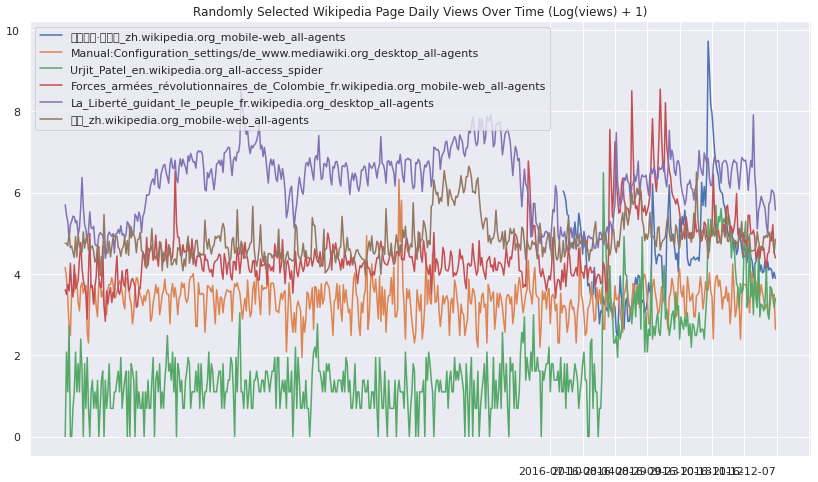
avec X = 50 et Y = 2.

La loss de notre modèle ne convergeait pas. Nous avons donc décidé d’autre données à nos éléments : la langue du site, le type d’accès, le type d’agent ou encore les mots avec les plus hautes fréquences d’apparition dans l’article. Au final nous n’avons pas su mélanger série temporelle et les autres données et nous avons décidé de recréer un seq2seq avec lstm mais cette fois-ci avec Keras au lieu de Torch (pouvant trouver plus d’information sur Keras avec les séries temporelles).

## **Modèle - Keras**

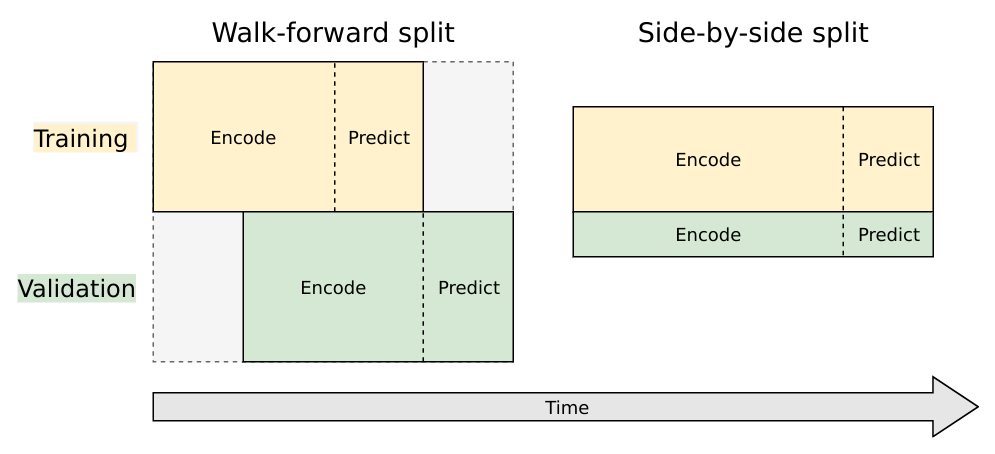
Pour le second modèle avec Keras, nous nous sommes inspiré d’un modèle déjà existant reprenant les données de ce Kaggle. Vous pouvez retrouver son notebook ici: <https://github.com/JEddy92/TimeSeries_Seq2Seq/blob/master/notebooks/TS_Seq2Seq_Intro.ipynb>

Dans un premier temps, nous visualisons la fréquence de vu quotidienne de quelques pages choisis aléatoirement pour avoir un aperçu de nos données.



Pour pouvoir exploiter les données, nous devons transformer les données en numpy array de sorte à ce qu’on puisse les exploiter avec Keras. Pour cela, nous devons partitionner correctement les données de times series en encoding et decoding pour le training et la validation.

Nous allons donc utiliser une walk-forward validation, qui consiste à prendre une même durée pour notre ensemble de train et de validation, mais décalée dans le temps (de 14 jours ici). Cette méthode nous permet de simuler les performances de notre modèle sur des données pas encore vues. Voici une image de [Artur Suilin](https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md) qui illustre ceci.



Tout d’abord, nous devons avoir 4 sous-segments des données:

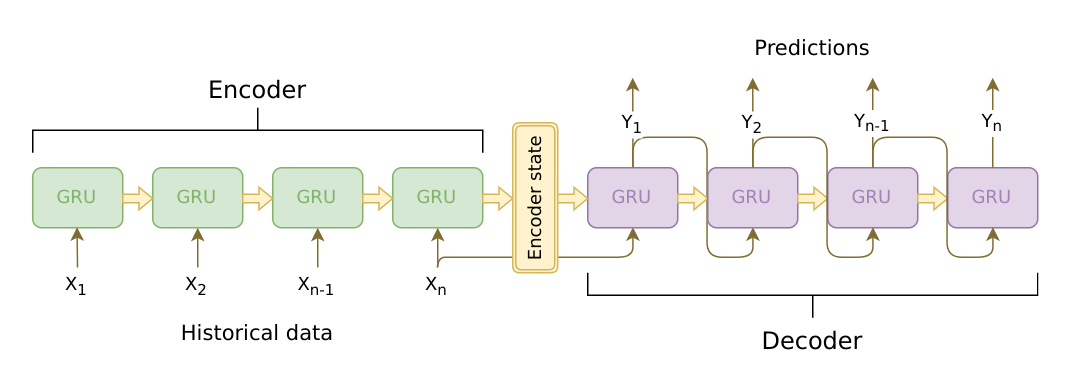
* la période d’encoding pour le train
* la période de decoding pour le train
* la période d’encoding pour la validation
* la période de decoding pour la validation

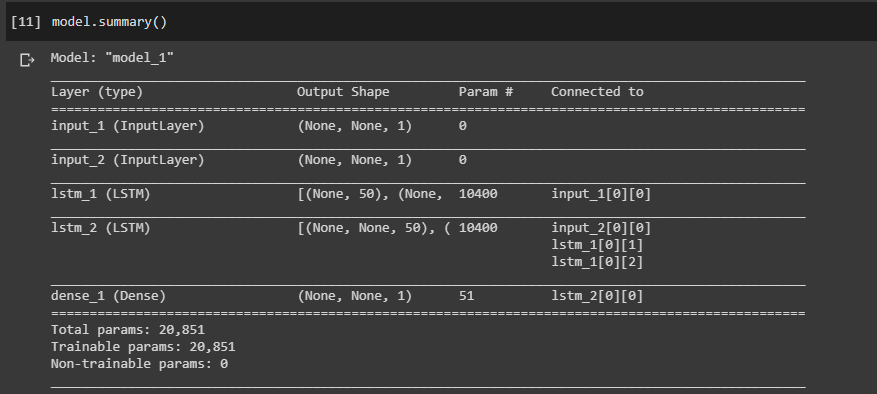
Ces données sont récupérées en prenant pour le training et la validation de la prédiction les données en partant de la fin, et pour le training et la validation de l’encoding en partant du début.

Ensuite, il faut formater les données pour que Keras puisse les comprendre. Des fonctions sont donc créer pour ça, reprenant ces étapes:

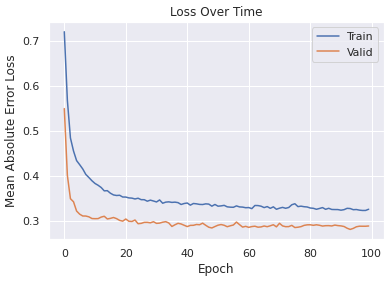
* récupérer les times series dans des arrays
* créer une fonction qui extrait des intervals de temps spécifiques de toutes les series
* créer des fonctions qui transforment toutes les series: on réajuste l’échelle en prenant le log1p et en redéfinissant chaque series en prenant la moyenne des series, puis en le mettant dans un format que Keras acceptera

Pour construire le modèle, nous utilisons une architecture utilisant le LSTM. Voici une image de [Artur Suilin](https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md) représentant l’architecture (nous utilisons des LSTM plutôt que des GRU ici).





En utilisant la moyenne absolue de la loss, cette dernière converge.



A partir de notre réseau de neurones, nous pouvons définir un autre modèle pour générer les prédictions. Cette architecture va encoder les données d’entrées, puis générer les prédictions une par une.

# 3 - Une autre application, la calcination

La calcination est le processus visant à brûler un corps pour le décomposer ou pour le soumettre à une réaction chimique (oxydation, réduction…).

Elle est utilisée industriellement dans la cimenterie par exemple, ou dans la recherche par exemple pour isoler une composante du matériau calciné (grâce à la décomposition).

Au cours de la calcination, plusieurs réactions se produisent, et ces dernières modifient drastiquement la forme du combustible utilisée, et changent ainsi ses propriétés physiques.

Il est intéressant de détecter ces réactions chimiques pour adapter la température de la combustion à l’état physique dans lequel le corps calciné se trouve à un instant t.

Certaines de ces réactions, de ces changements d’état peuvent être brusques et imprévus, pouvoir les anticiper devient donc un réel enjeu.

Plusieurs paramètres sont continuellement mesurés.

Notre modèle relève alors les valeurs de ces paramètres lors des moments de de brusque changements d’état, et essaie alors de prédire grâce l’évolution de ces paramètres quand se situeront les prochaines réactions, les prochaines ruptures et la température qui résultera de celles-ci.

Le Seq2Seq est intéressant pour pouvoir utiliser ces séries temporelles de mesures pour sa capacité à, au lieu de simplement d’essayer de déterminer quelle combinaison de valeurs a des chances de créer une réaction, réellement saisir la logique dans l’évolution des valeurs paramètres qui conduit à une rupture et à l’appliquer pour en déduire les prochaines ruptures. Cela passe par le mécanisme d’attention et de contexte qui permet de prendre en compte les valeurs récentes, les évolutions récentes pour par exemple repérer des patterns temporels qui provoquent une rupture : imaginons qu’une soudaine augmentation du CO2 signifie une grande flamme, nous pourrions imaginer que deux pics de CO2 suivi d’une légère baisse globale de la température du corps corresponde à une bûche qui se fend en deux pendant sa calcination, une fois ce stade atteint, une nouvelle baisse de la température pourrait correspondre à sa transformation totale en cendres.

Cet exemple montre la nécessité d’apprécier les états plus ou moins récents des paramètres pour établir : telle évolution de tels paramètres à l’intérieur d’un état A provoque sa transition à un état B.

# Conclusion

# Bibliographie

RNN encoder-decoder :

* Sutskever, I., Vinyals, O., and Le, Q. (2014). Sequence to sequence learning with neural networks. InAdvances in Neural Information Processing Systems (NIPS 2014)<https://arxiv.org/pdf/1409.3215.pdf>
* Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using RNN encoder-decoder for statistical machine translation.InProceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014). to appear.<https://arxiv.org/pdf/1406.1078.pdf>
* [https://fr.wikipedia.org/wiki/Réseau\_de\_neurones\_récurrents](https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_r%C3%A9currents)

Fonctionnement du LSTM :

Lambert R., Comprendre le fonctionnement d’un LSTM et d’un GRU

<http://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/>

Kaggle :

<https://www.kaggle.com/c/web-traffic-time-series-forecasting>