LIM Annie
MARCOCCIA Félix
VICHAIKIT Jean-Philippe
ZHOU Sébastien

Seq2seq : Depuis les réseaux de neurones vers deux applications phares

Jean-François Bercher, Julien Krywyk

Sommaire

Abstract

Notre équipe se compose de Annie LIM, étudiante en filière Datascience et Intelligence Artificielle, Félix MARCOCCIA, étudiant en filière Informatique, Jean-Philippe VICHAIKIT, étudiant en Datascience et Intelligence Artificielle, ainsi que Sébastien ZHOU, étudiant en filière Informatique qui nous a rejoint un peu plus tard.

Nous sommes tous en quatrième année à l'ESIEE Paris et réalisons ce projet dans le cadre de notre projet de fin d'année. Du 11 mai au 6 juillet 2020 nous étions sous la supervision de Jean-François Bercher, responsable de la filière Datascience et Intelligence Artificielle, et de Julien Krywyk, membre de la société Well&Wiz.

Ce projet est un projet exploratoire dont le but est d'apprendre le fonctionnement de différents modèles notamment le Seq2Seq afin de traiter des données temporelles, mais aussi de découvrir de nouvelles méthodes.

C'est un projet connecté à l'entreprise Well&Wiz. Nous rassemblerons ce que nous avons appris lors de ce projet dans un article qui sera ajouté au blog de l'entreprise, permettant de partager nos connaissances à toute une communauté.

Suite à une crise sanitaire, notre projet s'est exceptionnellement déroulé à distance avec des appels vidéos et des compte-rendus écrits réguliers.

Introduction

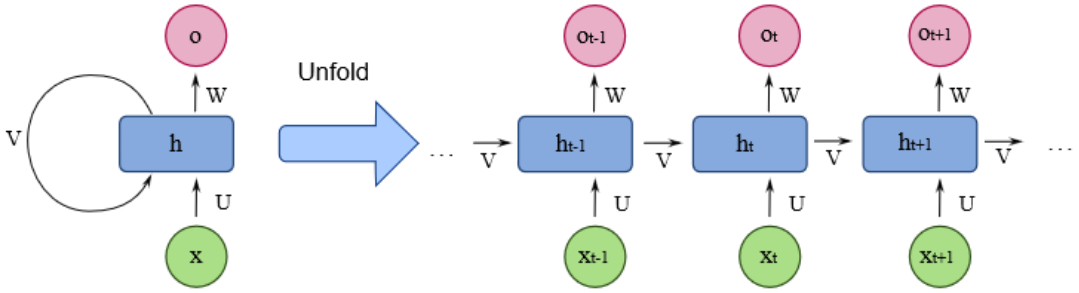
Si le modèle Seq2Seq est en vogue, c'est en grande partie grâce à son utilisation par Google notamment, pour les algorithmes de traduction automatique, et plus largement de Language Processing.
Sa capacité à prendre en compte le contexte dans lequel un mot est utilisé et des mots qu'il suit a été jugée primordiale dans ce domaine.
En effet, comment traduire une phrase, déduire le sens d'un mot, si on ne sait même pas distinguer la structure de la phrase, comment placer l'adverbe si on n'a aucune intuition sur la position du verbe ? Un CNN classique ne saurait même pas rendre compte du sempiternel "Sujet + Verbe + Complément".
Nous souhaitons utiliser ces mêmes capacités pour interpréter des données relevées continuellement dans le temps, quand l'évolution temporelle de celles-ci semble suivre certaines logiques.
Le principe est donc ici d'associer le Seq2Seq à des séries temporelles, la prédiction qu'il sera en mesure de produire évoluera alors avec le temps, et prendra en compte l'évolution de la situation au fur et à mesure de la prédiction.
Sans rentrer dans les écueils sur la nécessité de "s'adapter à un monde qui change", il est de plus en plus fréquent de travailler sur des données temporelles qui ne peuvent pas être analysées sorties de leur contexte, et sans considérer les évolutions qu'elles connaissent

avec le temps. Les prédictions données sont alors intrinsèquement liées à leur “époque” et aux évolutions qu’elles viennent de connaître.
Nous aborderons les principes fondamentaux de l'architecture d’un Seq2seq et de ce qui le rend particulier, nous détaillerons son implémentation et son implémentation sur un cas type (prédiction du nombre de visites sur des pages Wikipedia) et sur un papier scientifique pour optimiser (révolutionner ?) le procédé industriel de calcination.

0 - Les réseaux de neurones à mémoires

Réseau de neurones récurrents (RNN)

Le réseau de neurones récurrents (RNN)[1][2] 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.



RNN avec sa version dépliée par Ixnay, Wikipedia

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[3][4] 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. 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 cell 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.

1 - Le modèle Seq2Seq

Le modèle Seq2Seq[5][6] 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.

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 :

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

avec h_t l'état caché, W le poids et x_t 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 :

$$h_t = f(W^{(hh)} h_{t-1})$$

avec h_{t-1} l'état caché précédent et W le poids.

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

$$y_t = softmax(W^S h_t)$$

avec W^s 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.

Modèle ARIMA

Le modèle ARIMA[7] est un processus non stationnaire, formalisé par Box et Jenkins en 1970 [8], cherche à déterminer chaque valeur de la série en fonction des valeurs qui la précède, c'est-à-dire de prévoir l'évolution future d'un phénomène.

Le modèle est composé de trois parties paramétrées par les indices p, d, q tel que ARIMA (p, d, q) :

- Autoregression (AR) / p : Les données passées sont utilisées pour calculer un modèle de régression pour les données futures. Le paramètre p indique le nombre de termes auto-régressifs c'est-à-dire le décalage (lags).
- Integrated (I) / d : Il correspond au paramètre de différenciation pour permettre à la série temporelle de devenir stationnaire, c'est-à-dire que les valeurs de données sont remplacées par la différence entre les valeurs actuelles et les précédentes. Il est rare de rencontrer d>2, en effet il est dangereux de sur-différencié un processus.
- Moving Average (MA) / q : C'est le nombre de moyennes mobiles. On considère que la meilleure estimation est représentée par la moyenne pondérée d'un certain nombre de valeurs antérieures, permettant de lisser les données.

2 - Prévision de séries temporelles : Le trafic Web

Maintenant que nous avons notre modèle au complet, nous allons l'utiliser sur les données d'une compétition kaggle : "Web Traffic Time Series Forecasting"[9]. Ce Kaggle est un moyen d'illustrer la problématique de prédiction du trafic et non pas une finalité.

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 :

	Dates
Nom des pages sous la forme : “nom” ”langue.wikipedia.org” “type d'accès” “type d’agents”	Nombre de visites

- key_1.csv et key_2.csv :

Nom des pages	Id des pages

- sample_submission_1.csv et sample_submission_2.csv :

	Prediction
Id des pages	Nombre de visites

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 :

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0
2	3C_zh.wikipedia.org_all-access_spider	1.0	0.0	1.0	1.0	0.0	4.0	0.0
3	4minute_zh.wikipedia.org_all-access_spider	35.0	13.0	10.0	94.0	4.0	26.0	14.0
4	52_Hz_I_Love_You_zh.wikipedia.org_all-access_s...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...
145058	Underworld_(serie_de_películas)_es.wikipedia.o...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
145059	Resident_Evil:_Capítulo_Final_es.wikipedia.org...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
145060	Enamorándome_de_Ramón_es.wikipedia.org_all-acc...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
145061	Hasta_el_último_hombre_es.wikipedia.org_all-ac...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
145062	Francisco_el_matemático_(serie_de_televisión_d...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

145063 rows × 551 columns

Nous avons donc le nombre de vues sur 145063 articles Wikipédia sur 550 jours, du 1er Juillet 2015 au 31 Décembre 2016.
Nous devons ainsi prévoir le nombre de vues sur les 60 prochains jours, jusqu'au 1er mars 2016.

Feature Engineering

Nous avons pu extraire de nombreuses features à partir des données à notre disposition, qui apporteront des informations supplémentaires pour affiner nos prédictions.

BeautifulSoup

Nous voulions récupérer des informations relatives à chaque page d'article tels que le thème de l'article, le nombre de mots ou encore le nombre d'images présentes.

Pour cela, nous avons créé une fonction pour reconstituer l'url de la page wikipédia, à partir du nom de la page et de sa langue que nous avons dans la base de données.

Nous utiliserons la bibliothèque BeautifulSoup qui permet d'extraire des données à partir de fichiers HTML. Or à partir de l'url de chaque page que nous avons créé, nous accédons au code HTML relatif à chacune. En parcourant ainsi le code HTML grâce à BeautifulSoup nous pouvons récupérer le nombre de mots sur chaque page mais également la fréquence d'apparition de ceux-ci. Nous avons utilisé le module *Counter* de Python afin de

compter la fréquence d'apparition de chaque mot dans un dictionnaire. Ainsi nous avons les 10 premiers mots les plus utilisés par exemple, ce qui nous permet d'avoir une bonne idée sur le thème de la page.

```
https://zh.wikipedia.org/wiki/Apple_II
[('apple', 350), ('ii', 174), ('macintosh', 171), ('the', 110), ('please', 89), ('do', 89), ('not', 89), ('use', 89), ('it', 89), ('is', 89)]
nombre de mots :  7366
https://zh.wikipedia.org/wiki/As_One
[('', 1775), ('·', 1060), ('as', 159), ('one', 104), ('music', 93), ('please', 82), ('do', 82), ('not', 82), ('use', 82), ('it', 82)]
nombre de mots :  15087
https://zh.wikipedia.org/wiki/B-PROJECT
[('vocal', 222), ('(off', 180), ('the', 123), ('it', 104), ('king', 93), ('is', 86), ('please', 82), ('do', 82), ('not', 82), ('use', 82)]
nombre de mots :  8253
https://zh.wikipedia.org/wiki/B1A4
[('', 976), ('ver', 773), ('·', 678), ('-japanese', 523), ('曲目', 408), ('you', 405), ('b1a4', 371), ('語言: 韓語', 358), ('the', 340), ('love', 330)]
nombre de mots :  33279
https://zh.wikipedia.org/wiki/BDSM
[('it', 99), ('do', 94), ('please', 89), ('not', 89), ('use', 89), ('is', 89), ('deprecated', 89), ('the', 88), ('class', 63), ('and', 58)]
nombre de mots :  5277
```

On observe que de la ponctuation et des mots courants tels que des déterminants qui n'apportent aucune information sur la page sont parmi les mots les plus utilisés. Afin d'améliorer notre recherche et ne garder que les termes pertinents nous pourrions supprimer les mots très usuels (stopwords) de chaque langue avec la classe CountVectoriser de scikit learn. Nous pourrions également extraire la racine des mots (stemming) pour ne pas avoir toutes les variations d'un mot et ses différentes conjugaisons.

De la même façon, en parcourant chaque page avec BeautifulSoup, nous pouvons compter le nombre de fois où "img" apparaît dans le code HTML, correspondant au nombre d'images dans l'article.

```
https://zh.wikipedia.org/wiki/FAIRY_TAIL
nombre d'images :  102
https://zh.wikipedia.org/wiki/FIESTAR
nombre d'images :  16
https://zh.wikipedia.org/wiki/FIRST_CLASS
nombre d'images :  11
https://zh.wikipedia.org/wiki/Facebook
nombre d'images :  27
https://zh.wikipedia.org/wiki/Fantastic_Duo
nombre d'images :  7
https://zh.wikipedia.org/wiki/Fate/Grand_Order
nombre d'images :  21
```

Cependant nous n'avions pas pu extraire ces features sur toutes les pages du dataset car cela nous prenait trop de temps. Nous n'avons pu parcourir que 3248 pages en 1h, 5980 pages en 2h et il nous a fallu 3h12 pour scraper 10000 pages. Il aurait alors été trop long de scraper les 145063 pages.

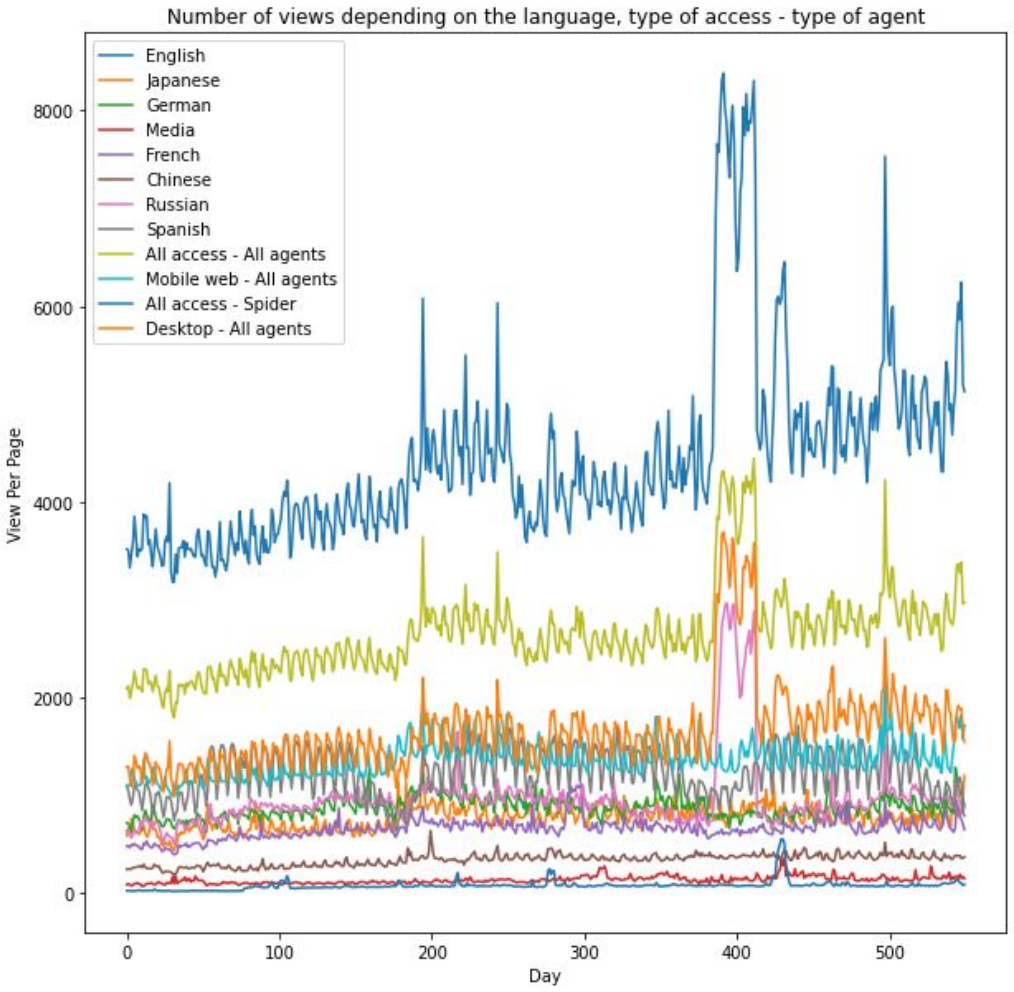
Ajout de features

Puisque chaque page de notre base de donnée est sous la forme “nom”_”langue.wikipedia.org”_”type d’accès”_”type d’agent”, il est facile d’y extraire la langue, ainsi que le type d’accès et le type d’agent.

En faisant cette extraction, nous pouvons distinguer 8 langues : anglais, japonais, allemand, media (correspondant aux pages de configuration), français, chinois, russe et espagnol, tel que : 'en'(Anglais): 24108, 'ja'(Japonais): 20431, 'de'(Allemand): 18547, 'na'(Media): 17855, 'fr'(Français): 17802, 'zh'(Chinois): 17229, 'ru'(Russe): 15022, 'es'(Espagnol): 14069.

De même, nous pouvons observer qu’il n’existe que 5 combinaisons différentes entre le type d’accès et le type d’agent : 'all-access_all-agents': 35099, 'mobile-web_all-agents': 30923, 'all-access_spider': 30614, 'desktop_all-agents': 30572, 'na': 17855.

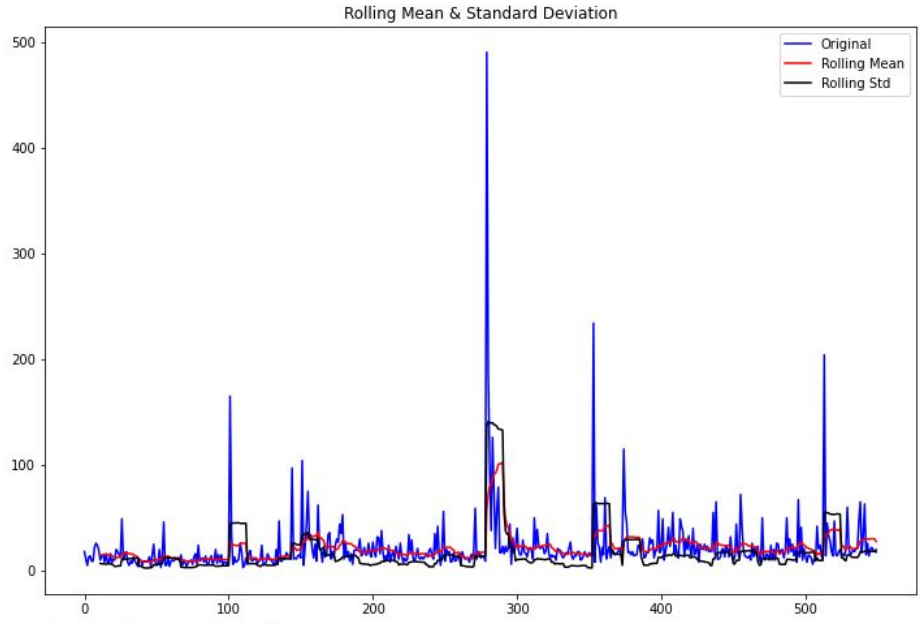
Plutôt que de séparer le type d’accès et type d’agent en deux features différentes il est alors plus judicieux de les regrouper en un feature “type d’accès_type d’agent”.



Nous pouvons alors clairement observer que selon la feature (langue, type d'accès_type d'agent), le nombre de visites ne sera pas le même. Il y a par exemple beaucoup plus de visites sur des pages en anglais plutôt qu'en chinois, sachant qu'il y a plus d'articles en anglais qu'en chinois.

Modèle ARIMA

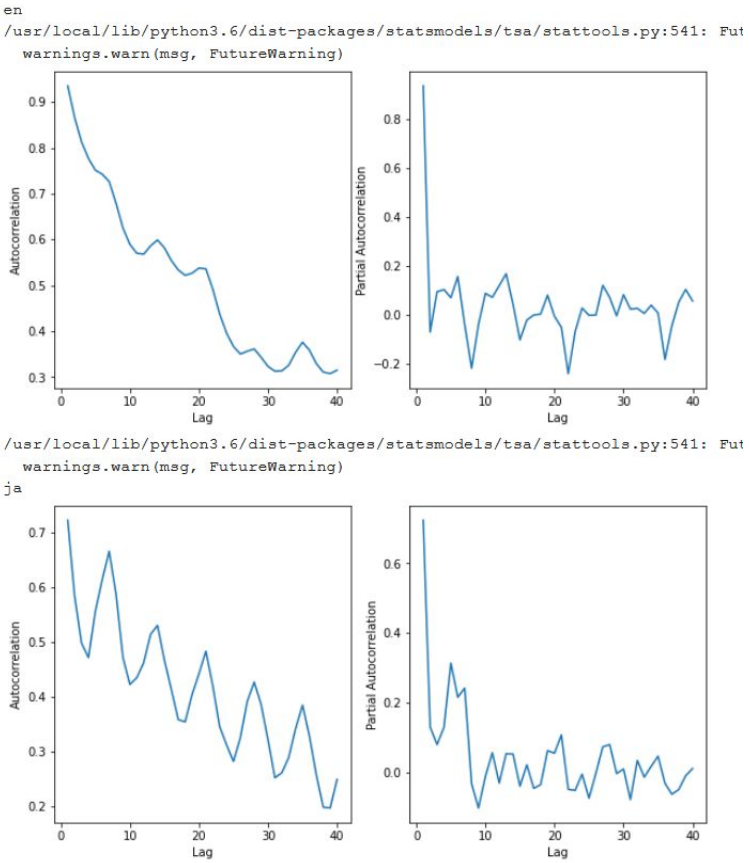
Nous allons ensuite soumettre nos données à un modèle ARIMA. Pour cela, nous devons tout d'abord vérifier sa stationnarité avec le test Augmented Dickey Fuller. [10][11]




```
Results of Dickey-Fuller Test:
p-value = 0.0000. The series is likely stationary.
Test Statistic                -8.884038e+00
p-value                        1.294670e-14
#Lags Used                     3.000000e+00
Number of Observations Used    5.460000e+02
Critical Value (1%)           -3.442384e+00
Critical Value (5%)           -2.866848e+00
Critical Value (10%)          -2.569597e+00
dtype: float64
```

Notre p-value est extrêmement faible donc notre série est stationnaire. De plus notre test statistique a une valeur bien inférieure à la valeur critique de 1% ce qui est très précis.

Nous pourrons ensuite adapter nos hyperparamètres du modèle ARIMA en observant leur graphe d'autocorrélation et d'autocorrélation partielle.



Nous pouvons regrouper les features en deux groupes selon l'allure de leur autocorrélation et autocorrélation partielle :

- 'en', 'na', 'fr', 'ru', 'all-access_all-agents', 'all-access_spider', 'desktop_all-agents'
- 'ja', 'de', 'zh', 'es', 'mobile-web_all-agents'

Nous devons déterminer les hyperparamètres Autoregression (p), Integrated (d) et Moving Average (q).

Nous aurons d=1 pour tous car la première différence d'ordre arrive à 1.

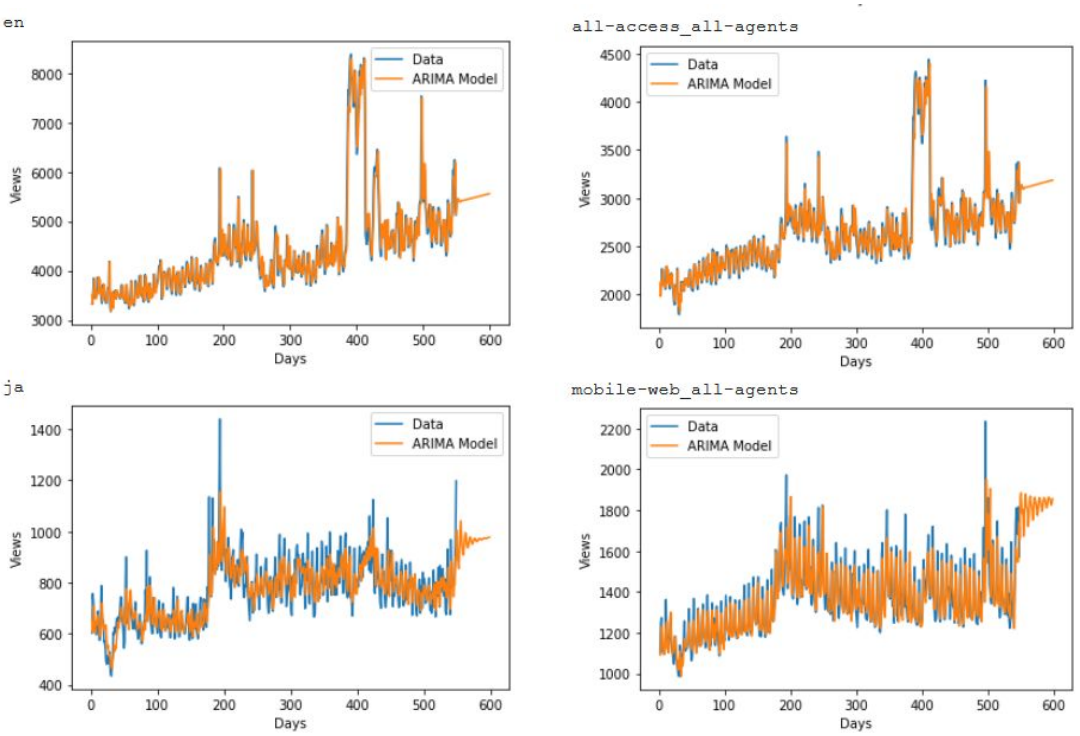
On observe dans l'autocorrélation partielle des cas comme 'ja', l'autoregression devient plus importante à partir de 7 lags, donc p=7.

Et dans les cas comme 'en', nous obtenons une bonne autoregression dès 4 lags, donc p=4.

On mettra q=1 pour les cas comme 'ja' afin de lisser les données. Pour les cas comme 'en' on laissera q à 0 comme les données sont déjà assez lisses.

```
'en': [4,1,0], 'ja': [7,1,1], 'de': [7,1,1], 'na': [4,1,0], 'fr': [4,1,0], 'zh':[7,1,1], 'ru': [4,1,0], 'es': [7,1,1],
'all-access_all-agents':[4,1,0],'mobile-web_all-agents' : [7,1,1], 'all-access_spider' : [4,1,0],
'desktop_all-agents':[4,1,0]
```

En appliquant les hyperparamètres tels que ci-dessus à notre modèle ARIMA, nous obtenons les prévisions ci-dessous pour les 60 prochains jours demandés sur le Kaggle :



Les valeurs du modèle ARIMA suivent bien les valeurs de nos données mais les valeurs prédites ne semblent pas correctes puisque ne suit pas l'allure précédente, mais l'ordre des valeurs reste cohérent

Nous pensons améliorer la précision des valeurs prédites en combinant ce modèle ARIMA qui prenait en compte nos différentes features, avec un modèle Seq2Seq qui s'entraînera sur nos séries temporelles.

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 éléments 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 associés à un label qui représente les Y nombres suivants de visites.

```
Données : tensor([18., 11.,  5., 13., 14.,  9.,  9., 22., 26., 24., 19., 10., 14., 15.,
         8., 16.,  8.,  8., 16.,  7., 11., 10., 20., 18., 15., 14., 49., 10.,
        16., 18.,  8.,  5.,  9.,  7., 13.,  9.,  7.,  4., 11., 10.,  5.,  9.,
         9.,  9.,  9., 13.,  4., 15., 25.,  9.])
Label : tensor([5., 6.])
```

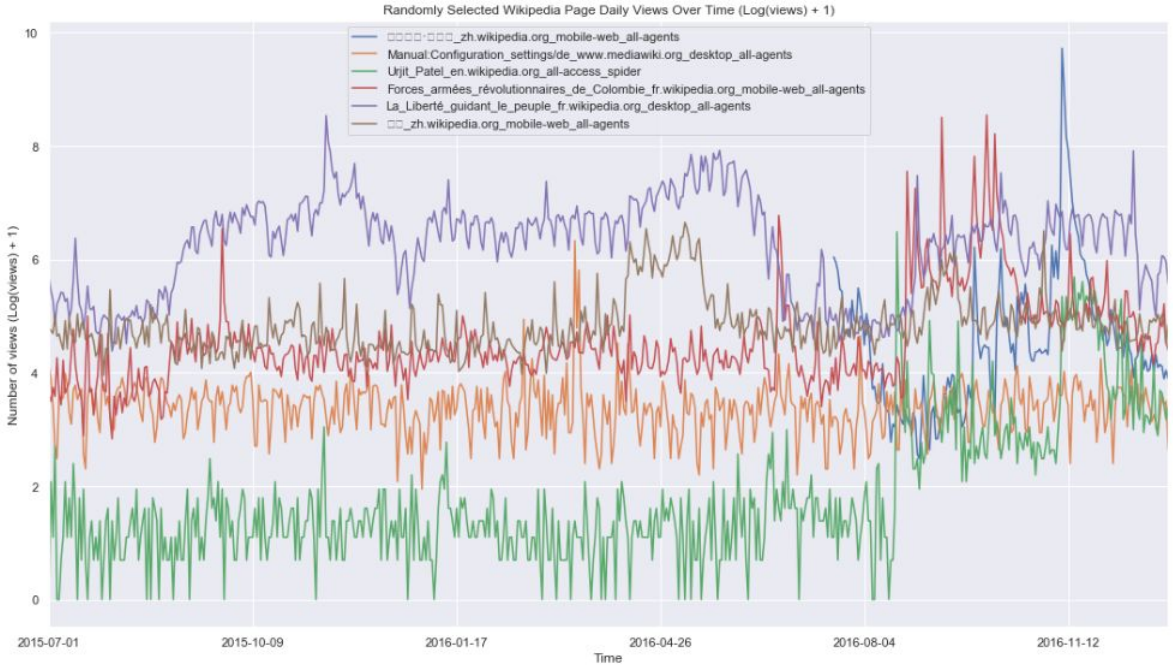
avec X = 50 et Y = 2.

La loss de notre modèle ne convergeait pas. Nous avons donc décidé d'ajouter d'autres 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. Ne trouvant pas assez d'information sur les time series avec seq2seq et Torch, nous avons décidé de basculer vers Keras, pouvant plus facilement trouver des informations dessus.

Modèle - Keras

Pour le second modèle avec Keras, nous nous sommes inspiré d'un modèle déjà existant reprenant les données de ce Kaggle, fait par Joseph Eddy. [12]

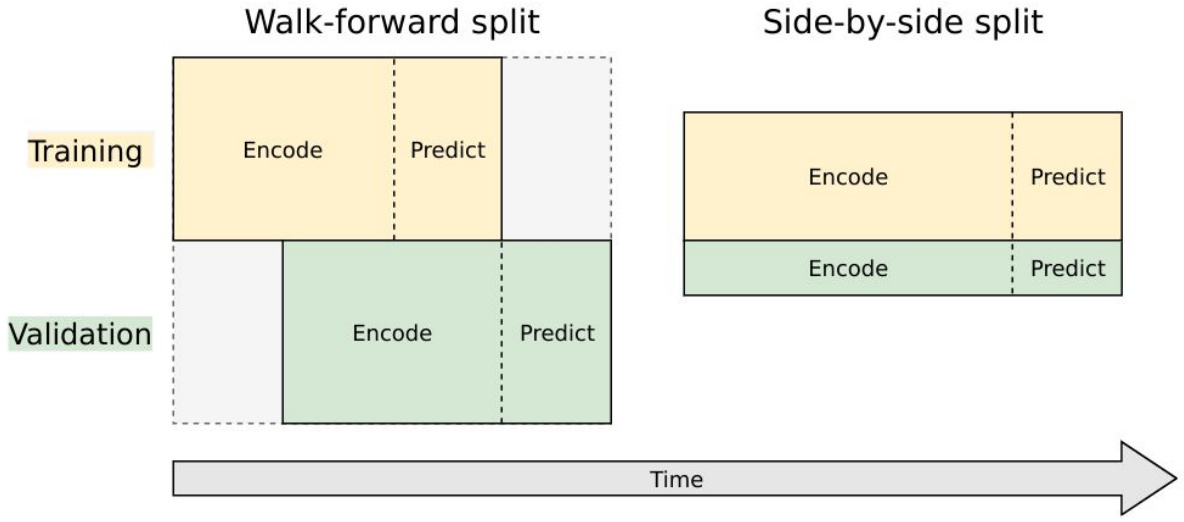
Dans un premier temps, nous visualisons la fréquence de visites quotidiennes de quelques pages choisies aléatoirement pour avoir un aperçu de nos données.



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.

Monitoring de séries temporelles

Nous allons donc utiliser une Walk-Forward Split, qui consiste à prendre une même durée pour notre ensemble de train et de validation, mais décalée dans le temps (de 14 jours ici). L'encoding de la validation est donc dans la même période temporelle qu'une partie de l'encoding et de la prédiction de l'ensemble de train, mais avec des données différentes.
Une autre méthode est le Side-by-Side Split. Il s'agit d'une méthode plus traditionnelle, qui sépare les données en deux, tout en gardant la même plage de temps.
On préfèrera utiliser une Walk-Forward Split, car elle correspond plus à ce que nous souhaitons prédire, c'est-à-dire des données futures. Cependant, cette méthode consomme des données qui se situe à la fin de la série temporelle, rendant ainsi difficile d'entraîner le modèle à prédire le futur.



Différence entre une Walk-Forward Split et une Side-by-Side Split, Arthur Suilin

Pour mieux expliquer le problème, prenons un exemple. Disons que nous avons 300 jours de données, et qu'on souhaite prédire les 100 prochains jours. Sur ces données, les 100 premiers jours seront utilisés pour le training, puis les 100 prochains jours pour la prédiction du training, ensuite les 100 derniers jours de nos données pour la validation, puis les 100 jours d'après seront des valeurs prédites. Donc au final, nous n'utilisons qu'un tier des données pour le training, et on aura un écart de 200 jours entre la dernière donnée de training et la première donnée de prédiction. Cet écart est trop élevé car la qualité des prédictions diminue de manière exponentielle au fur et à mesure que l'on s'éloigne des données de training. Un entraînement sur un modèle avec 100 jours d'écart serait considérablement meilleur.

D'un autre côté, une Side-by-side split aurait très bien pu marcher aussi, mais les performance du modèle de l'ensemble de données de validation sont très corrélées aux

performances sur les données de training, et ceux presque sans corrélations avec les performances réelles des futures données. Entre autre, cette méthode est difficilement exploitable.

Donc tout d'abord, nous devons avoir 4 sous-segments des données:

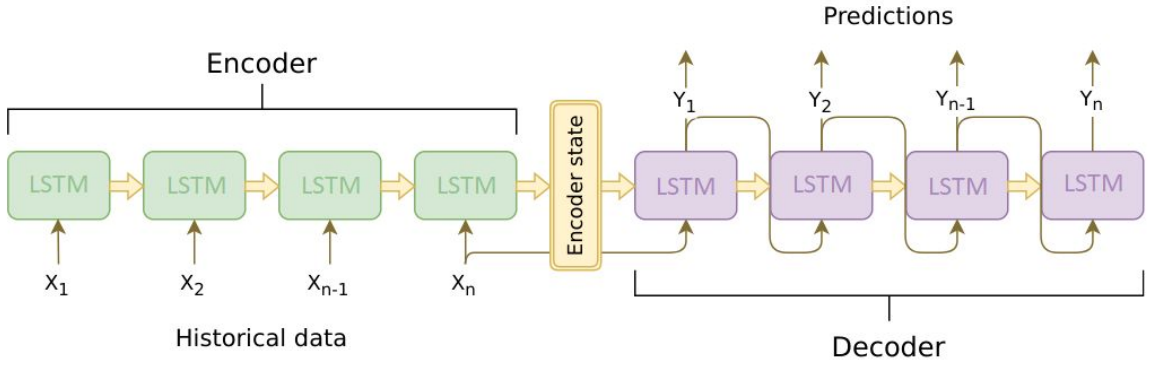
- la période d'encoding pour le train
- la période de decoding pour le train
- la période d'encoding pour la validation
- la période de decoding pour la validation

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 intervalles de temps spécifiques de toutes les séries
- créer des fonctions qui transforment toutes les séries: on réajuste l'échelle en prenant le log1p et en redéfinissant chaque séries en prenant la moyenne des séries, puis en le mettant dans un format que Keras acceptera

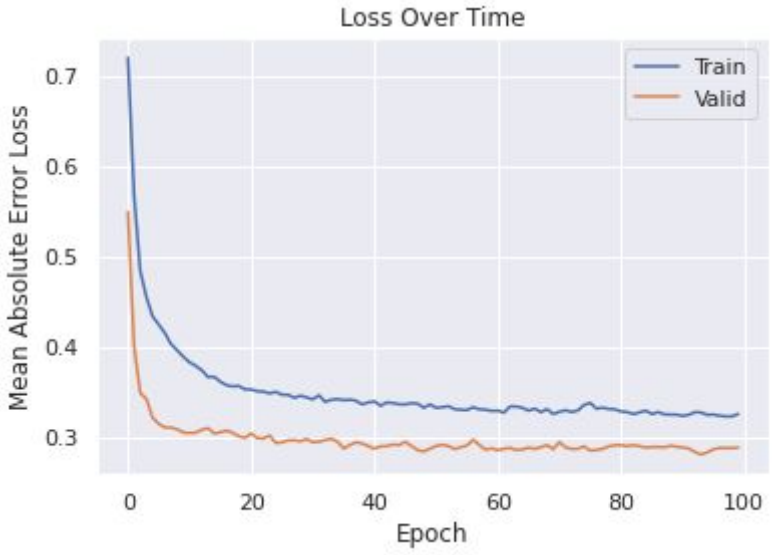
Pour construire le modèle, nous utilisons une architecture utilisant le LSTM.



Architecture avec LSTM, Artur Suilin

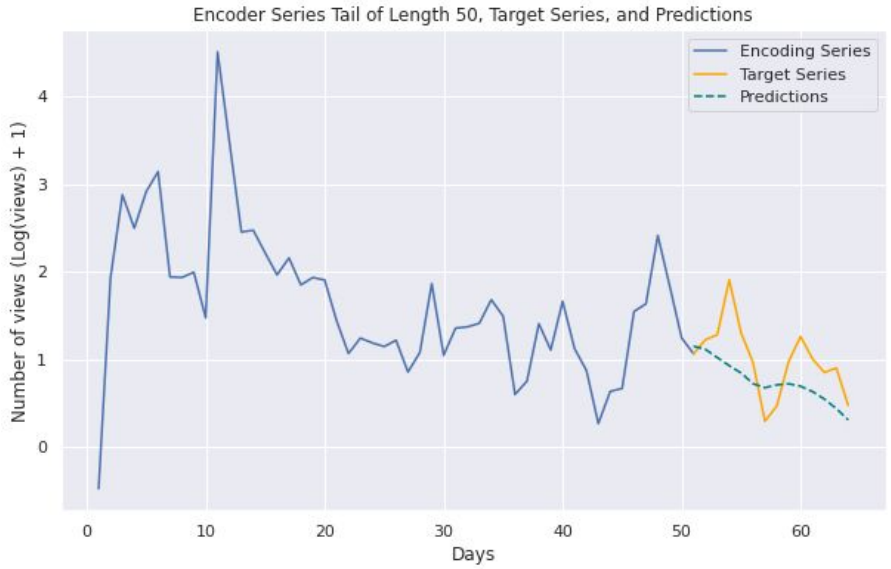
```
[11] model.summary()

Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, None, 1)      0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, None, 1)      0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 50), (None,  10400       input_1[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, None, 50), ( 10400       input_2[0][0]
                                                                 lstm_1[0][1]
                                                                 lstm_1[0][2]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, None, 1)      51          lstm_2[0][0]
==================================================================================================
Total params: 20,851
Trainable params: 20,851
Non-trainable params: 0
__________________________________________________________________________________________________
```

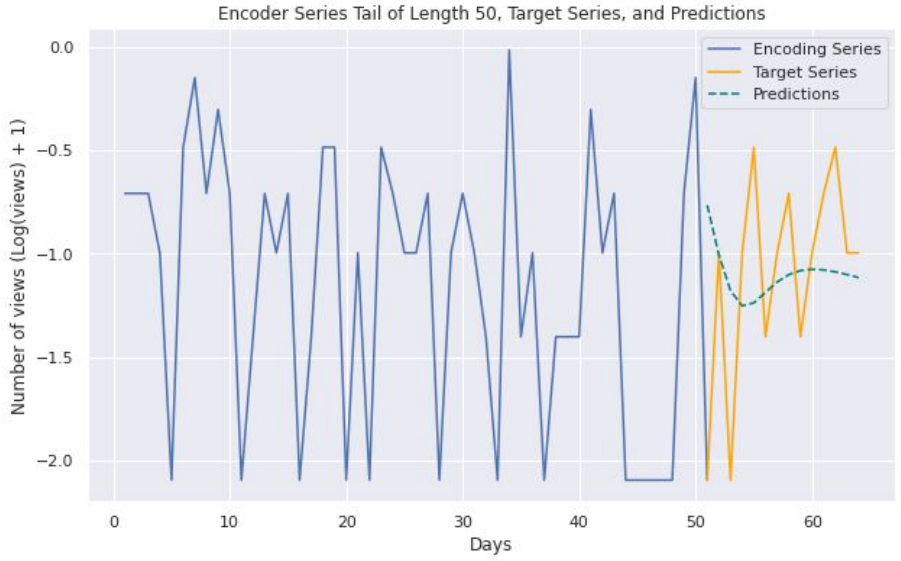
En utilisant la moyenne absolue de la loss, cette dernière converge.



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.

Voici quelques exemples de prédictions sur 50 jours d'encoding, et de prédiction sur 14 jours.





On observe que le modèle a globalement appris à approximer les futures données, mais en négligeant les pics de variations. Pour améliorer le modèle, nous pourrions mettre plus de données dans le train, optimiser les hyperparamètres, ou encore faire plus epoch.
Nous n'avions pas eu le temps de faire de soumission sur le Kaggle, mais nous pouvons comparer les meilleurs scores.

#	△pub	Team Name	Score
1	—	Arthur Suilin	35.48065
2	—	CPMP	36.78499
3	—	thousandvoices	36.85302

La mesure de performance utilisée pour calculer l'erreur dans cette compétition Kaggle est le SMAPE (Symmetric Mean Asolute Percentage Error). Cette métrique a pour particularité d'ignorer les cas particuliers et est invariant si l'on redimensionne linéairement les données.
Ce score est compris entre 0 et 200, avec 0 la meilleure valeur.

Nous pouvons voir que le meilleur score de la compétition est de 35.48 qui est plutôt bon mais pas parfait, il est donc possible de prévoir les nombres de visites sur les pages Wikipédia sur une durée assez courte (60 jours). Il sera encore plus compliqué d'avoir des prévisions précises sur une plus longue durée.

3 - Une autre application, la calcination

Principe

La calcination[13] est le processus visant à brûler un corps dans une enceinte fermée 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.

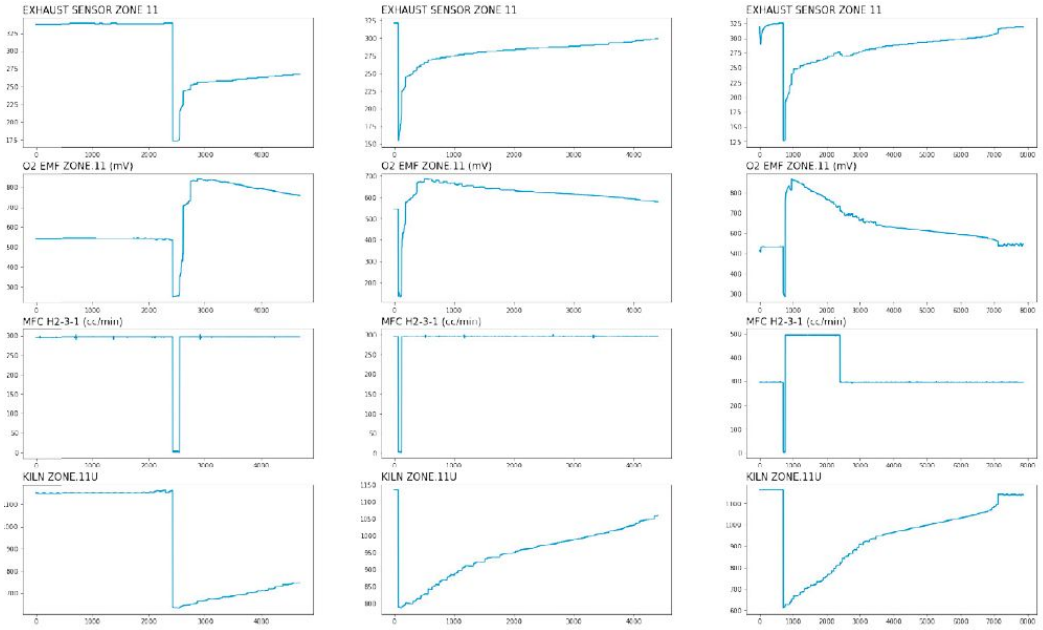
Application

Plusieurs paramètres sont continuellement mesurés.

Variable	Explain
$Volt.SCR.11U$	Voltage
$Ampare.SCR.11U$	Ampare
$Res.SCR.11U$	Resistance
$EXHAUSTSENSORZONE11$	Exhaust gas
$O2EMFZONE.11(mV)$	Oxygen saturation
$MFCH2-3-1(cc/min)$	Amount of hydrogen injection
$KILNZONE.11U$	Temperature in the firing furnace

Ces données sont sous la forme de séries temporelles de températures, pression etc…
L'enjeu sera alors de prédire les prochaines températures à partir de ces données.

Le modèle utilise alors les valeurs observées lors des moments de ruptures, qui présentent des évolutions anormales, et analyse l'évolution de ces valeurs avant un "'incident".
Voici à quoi ressemblent les évolutions de certains paramètres (ici les émissions de CO2, les gaz exhalés), c'est dans ce genre de zone de rupture que nous prélevons nos données utiles. [13]

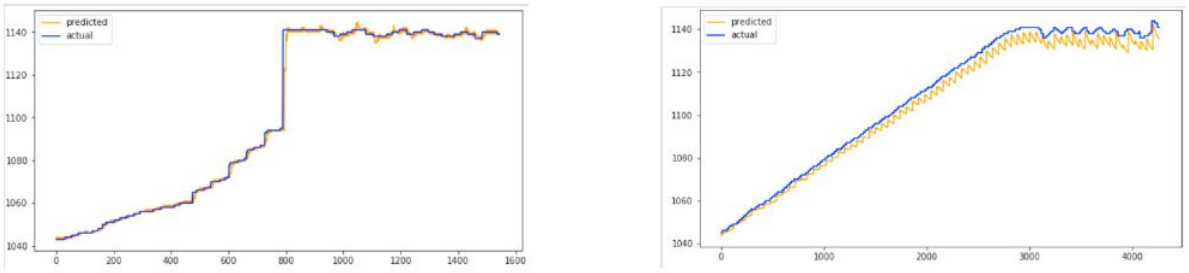


Le but est alors de déterminer des motifs (patterns) de variation de paramètres qui engendrent typiquement ce genre de réaction brutale.

Pertinence du Seq2Seq

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.

Les résultats obtenus par l'équipe confirment la pertinence de la démarche.[13]



(a) InputSeq:30 OutputSeq:5 (b) InputSeq:300 OutputSeq:60

Résultats du modèle Seq2Seq

Conclusion

Ce projet nous a donc permis d'en apprendre énormément sur le modèle Seq2Seq afin de traiter les séries temporelles. A travers deux cas concrets nous avons vu que le Seq2Seq était très utile pour prédire des valeurs continues dans le temps, autant la température pour l'article de la calcination, que le nombre de visites sur les articles Wikipédia.

Cet algorithme est donc très adapté pour prédire des séries temporelles qui ont un lien direct entre elles tout en conservant un contexte pour une meilleure prédiction. La périodicité et les tendances de la série sont donc bien prises en compte par exemple. Il peut traiter une grande quantité de données sans perdre le contexte entre chaque. Cependant il devient plus compliqué d'y ajouter d'autres informations de type catégoriel comme nos autres features (langue, type d'agent, type d'accès).

Utiliser Torch est plus adapté pour des travaux de recherche ce qui est notre cas, alors que Keras est plus adapté pour des travaux en entreprise. Il était plus simple de trouver des documentations et solutions avec Keras, notamment pour le Seq2Seq.

Le Seq2Seq est très utilisé dans le Language processing notamment par Google pour la traduction de texte par exemple, car traiter des séquences en prenant en compte les données précédentes se prête très bien au mécanisme du modèle.

Pouvant prédire la température, le nombre de visites d'une page d'article, prévoir des tendances statistiques ou encore de la traduction, ou faire des résumés de texte, le Seq2Seq a un large domaine d'applications et est de plus en plus utilisé pour son efficacité, c'est donc un algorithme très prometteur.

Bibliographie

1. Grant Sanderson - But what is a Neural Network? | Deep learning, chapter 1 https://www.youtube.com/watch?v=aircAruvnKk
2. Andrew NG - What is a Neural Network? (C1W1L02) https://www.youtube.com/watch?v=n1l-9lIMW7E
3. Andrej Karpathy - CS231n Winter 2016: Lecture 10: Recurrent Neural Networks, Image Captioning, LSTM https://www.youtube.com/watch?v=yCC09vCHzF8&t=1529s
4. Lambert R., Comprendre le fonctionnement d'un LSTM et d'un GRU http://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/
5. 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
6. 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
7. Florin Avram. *Series temporelles : regression, et modelisation ARIMA(p,d,q).* 2 décembre 2012. http://avram.perso.univ-pau.fr/sertemp/ser.pdf
8. G. Box and G. M. Jenkins, Time Series Analysis, Forecasting, and Control, Holden-Day, San Francisco, 1970. http://www.ru.ac.bd/stat/wp-content/uploads/sites/25/2019/03/504_05_Box_Time-Series-Analysis-Forecasting-and-Control-2015.pdf
9. Kaggle. Web Traffic Time Series Forecasting. https://www.kaggle.com/c/web-traffic-time-series-forecasting
10. Dickey, D.A. et W.A Fuller (1981), "Likelihood Ratio Statistics for Autoregressive Time Series with a Unit Root", Econometrica, 49, p. 1057-1072. http://www.u.arizona.edu/~rlo/readings/278800.pdf
11. Sean Abu. *Seasonal ARIMA with Python*. 22 mars 2016. http://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/
12. Notebook Github, Joseph Eddy. https://github.com/JEddy92/TimeSeries_Seq2Seq/blob/master/notebooks/TS_Seq2Seq_Intro.ipynb
13. S. Hwanga, G. Jeona, J. Jeonga, J. Lee. (2019). A Novel Time Series based Seq2Seq Model for Temperature Prediction in Firing Furnace Process. https://www.sciencedirect.com/science/article/pii/S1877050919309196