Rapport TER : Génération de tweets de Donald Trump

Gabin Marc Mbery Kongo, Florent Jakubowski

Rapport TER : Génération de tweets de Donald Trump **1**
1) Introduction 2
2) Environnement et présentation des données 3
3) Réseaux de neurones récurrents 3
3.a) LSTM simple 5
i. Aspects théoriques 5
ii. Implémentations et résultats expérimentaux 5
3.b) GRU avec attention 7
i. Aspects théoriques 7
ii. Implémentations et résultats expérimentaux 8
4) Modèle de chaîne de Markov 11
i. Aspects théoriques 11
ii. Implémentations et résultats expérimentaux 12
5) Test auprès des étudiants 12
6) GPT-2 13
i. Architecture transformeur 13
ii. Parallélisation 16
iii. Généralisation et transférabilité 16
Conclusion et perspectives 18
Bibliographie 19

1) Introduction

Nous nous sommes penchés dans ce travail d'étude et de recherche sur la génération automatique de texte à l'aide de réseaux de neurones profonds. Le but fut d'utiliser plusieurs architectures différentes de réseaux de neurones profonds pour la génération de tweets de l'ancien président américain Donald Trump.

Nous nous sommes concentrés dans un premier temps sur 2 types d'architectures de réseaux de neurones récurrents, une architecture de réseaux de neurones récurrents de type LSTM et une architecture de réseaux de neurones récurrents de type GRU avec attention. Nous avons également testé un modèle de type chaîne de Markov qui utilise non pas des représentations vectorielles des mots mais encode le code ASCII de chaque caractère.

Nous expliquerons plus en détails par la suite pourquoi nous avons voulu utiliser ce type de modèle.

Nous avons aussi étudié les modèles de langages pré-entraînés et leur spécialisation pour la génération de texte. Notamment le modèle de langage pré-entrainé GPT2. Nous n'avons malheureusement pas eu le temps de mettre en place une implémentation de ce type d'architecture.

2) Environnement et présentation des données

Nous avons développé notre code en python avec l'outil Tensor Flow [1] et l'API Keras [2]. Nous avons utilisé des notebooks sur le site Google Colab [3] ce qui nous a permis d'avoir accès à un environnement d'exécution relié à un GPU. Nous avons pu entraîner nos modèles plus rapidement grâce à cela.

Le jeu de données provient du site thetrumparchive.com [4] et regroupe plus de 50 000 tweets de Donald Trump. Ce jeu de données comporte aussi bien les tweets de Donald Trump ainsi que les tweets qu'il a retweetés, nous les avons par la suite retirés du jeu de données pour l'entraînement de nos modèles. Après nettoyage des données, le nombre de tweets s'élevait à 52183.

Pour l'entraînement nous avons séparé les tweets en deux parties, une première partie de 5 mots et une deuxième partie avec le reste du tweet.

3) Réseaux de neurones récurrents

Nous avons utilisé deux types de réseaux de neurones récurrents, un premier de type LSTM (Hochreiter,S. et al. (1997) [5]) et un deuxième de type GRU (Cho,K., et al. (2014) [6]). Un mécanisme d'attention a été ajouté à l'architecture de type GRU (Vaswani,A., et al. (2017) [7]).

Les réseaux de neurones récurrents (Rumelhart,D., et al. (1985) [8] ; Michael,J., (May 1986) [9]) ont pour but de pallier au phénomène d'évanescence du gradient (ou "vanishing gradient" en anglais) des réseaux neuronaux. Les cas typiques d'application sont les cas où la donnée contient de l'information supplémentaire dans son ordonnancement, typiquement des séries temporelles ou bien le langage naturel.

Le RNN est constitué de plusieurs cellules contenant chacune un système de portes servant à garder en mémoire de l'information. Chaque cellule produit également un ou des états en fonction du type d'architecture (un état pour les GRU et deux états pour les LSTM). Ce ou ces états sont passés à la cellule suivante qui la combine avec la nouvelle entrée.

Ces mécanismes de portes permettent à l'architecture RNN d'apprendre quelles informations gardées en mémoire et lesquelles elle peut oublier. Les états permettent quant à eux de fournir du contexte à chaque nouvelle entrée. Ainsi ces deux mécanismes permettent au réseau de neurones d'apprendre quelles sont les informations intéressantes à garder en mémoire et qu'il devra fournir à la cellule suivante. Grâce à l'entraînement, le réseau comprend les relations intéressantes entre les différents mots et les encodes.

Le RNN est constitué de plusieurs cellules mémoires contenant chacune un système de porte. Une porte d'entrée, de sortie et d'oublis (input gate, output gate et forget gate en anglais) pour le LSTM. Une porte de réinitialisation ("reset gate") et une porte de mises à jour ("update gate"). Chaque cellule génère également un état ou des états, un état caché pour le GRU et deux états pour le LSTM, un état caché et un état de cellule. Ce ou ces états sont passés à la cellule suivante qui la combine avec la nouvelle entrée.

Ces mécanismes de portes et d'états permettent à l'architecture RNN d'apprendre quelles informations gardées en mémoire et lesquelles elle peut oublier et permet aussi de fournir du contexte à chaque nouvelle entrée. Permettant ainsi de préserver l'information intéressante provenant des précédentes entrées. Cela permettra aux réseaux de neurones de comprendre les relations entre les différents mots dans le cas qui nous intéresse ici.

Pour la génération de texte, une architecture particulière de réseaux de neurones a fait ses preuves, l'architecture SeqtoSeq (Sutskever,I. et al. (2014) [10]). Cette architecture combine deux réseaux de neurones, un premier appelé encodeur et un deuxième appelé décodeur. Nous avons dans un premier temps utilisé deux réseaux de neurones LSTM puis deux GRU avec un mécanisme d'attention. Nous verrons par la suite le principe de l'attention.

Dans le cas d'une architecture SeqtoSeq avec deux RNN :

Le réseau de neurones encodeur se charge d'encoder l'information en entrée. Dans notre cas ici, ce fut quelque mot de début de tweet. Puis génère un ou deux états (selon que nous sommes en présence d'un GRU ou d'un LSTM) qui est ou sont passés en entrée au réseau de neurones décodeurs. Ce ou ces états vont servir de contexte à la première cellule du réseau de neurones décodeur.

Le réseau de neurones décodeur génère ensuite un premier mot à partir de ce ou ces états et du token de début que nous lui avons passé en entrée. Ce mot est ensuite passé avec le ou les nouveaux état générés par la cellule à la cellule suivante et ainsi de suite pour générer la fin du tweet. Le décodeur s'arrête une fois qu'il a généré le token de fin.

3.a) LSTM simple

i. Aspects théoriques

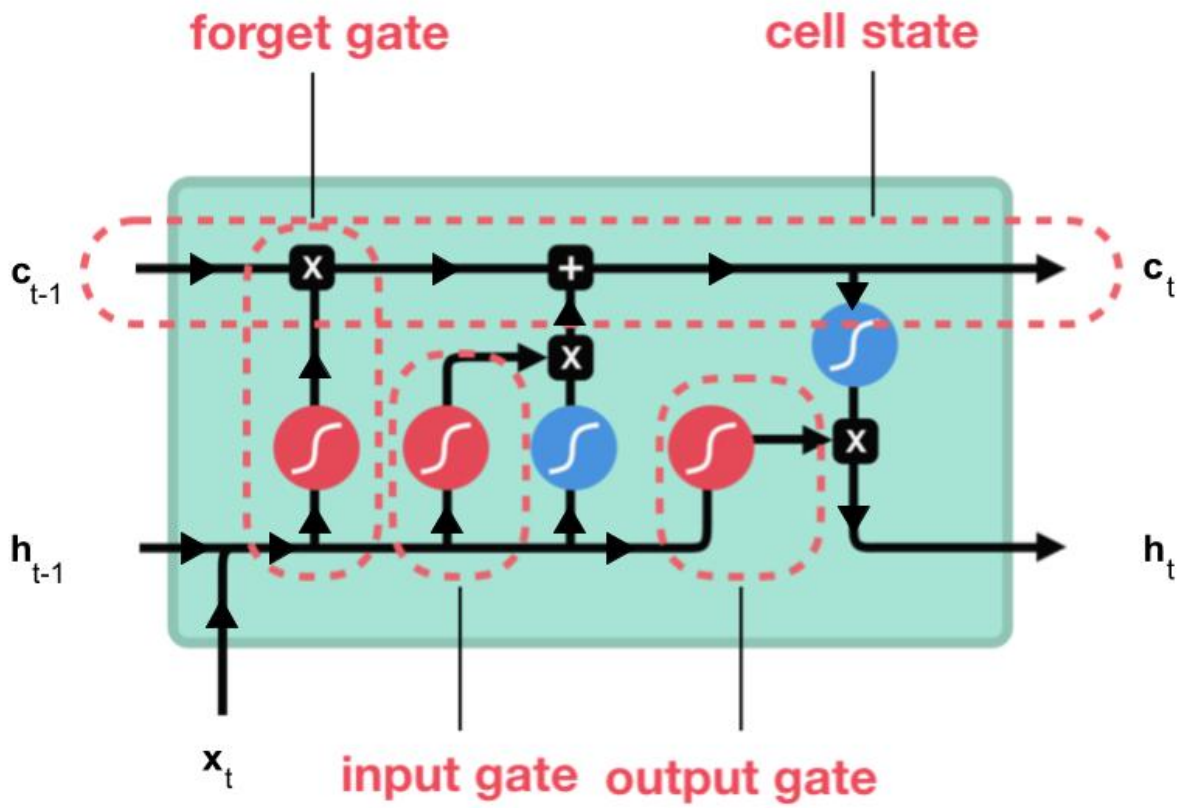

Figure 1 : Schéma d'une cellule LSTM.

Le LSTM est constitué de plusieurs cellules mémoires contenant chacune une porte d'entrée, de sortie et d'oublis (input gate, output gate et forget gate en anglais). Chaque cellule génère également un état caché en plus d'un état de cellule. Ces deux états (cachés et de la cellule) sont passés à la cellule suivante qui la combine avec la nouvelle entrée.

ii. Implémentations et résultats expérimentaux

Nous avons utilisé au début 8 epochs et avons pu constater déjà des tweets avec un début de cohérence syntaxiques et sémantiques.

- Exemple de tweets générés :

Entrée	Sortie
“barack obama is the”	“great job for the world . i will be a great job !”

Nous sommes ensuite passé à 30 epochs et nous avons réglé un label smoothing à 0.2.

Le label smoothing sert à mieux généraliser en faisant en sorte que les “labels” ne soient plus assignés avec des 1 ou des zéros mais avec des nombres compris en 1 et 0. Par exemple 0.2 ou 0.8. Ce qui permet de limiter le surapprentissage.

- Exemple de tweets générés :

Entrée	Sortie
“barack obama is the”	“worst president in the world. all of the world is laughing at the same time ?”
“71 000 000 votes incredible”	“- we are going to win !”
“thanks to all of you”	“to the great state of texas. i will be a great governor of montana !”

On peut remarquer des comportements intéressants comme avec le début du tweet “71 000 000 votes incredible”, nous pouvons voir que le réseau de neurones complète le tweet en commençant par un tiret pour rendre la phrase syntaxiquement exacte.

On remarque également dans d’autres tweets l’écueil classique des LSTM à savoir la répétition de séquences. Quelques exemples ici :

Entrée	Sortie
“hillary clinton is”	“to be a total loser- but he is a total loser !”
steeve banon what a great	guy who is doing a great job. he is a great guy and a great guy !

Nous avons essayé avec quelques personnalités provenant soit du camp de Donald Trump soit du camp opposé, pour voir si le réseau de neurones avait pu encoder cette information. Les résultats sont assez mitigés, cela semble être assez probant avec Hillary clinton mais avec d’autres personnes comme Barack Obama ou Steeve Banon l’avis exprimé est moins tranché :

Entrée	Sortie
“steeve banon is”	“a total loser- he is a total loser !”
“steeve banon what a great “	“guy who is doing a great job. he is a great guy and a great guy !”
“hilary clinton what a”	“racist stereotypes are not getting worse . --albert einstein”
“barack obama what a”	“people who are doing a great job .”

Il semble que la tournure de phrase suivant la mention des noms à son importance. Le réseau de neurones a réussi à encoder les sentiments négatifs de Donald Trump seulement envers Hillary Clinton.

3.b) GRU avec attention

i. Aspects théoriques

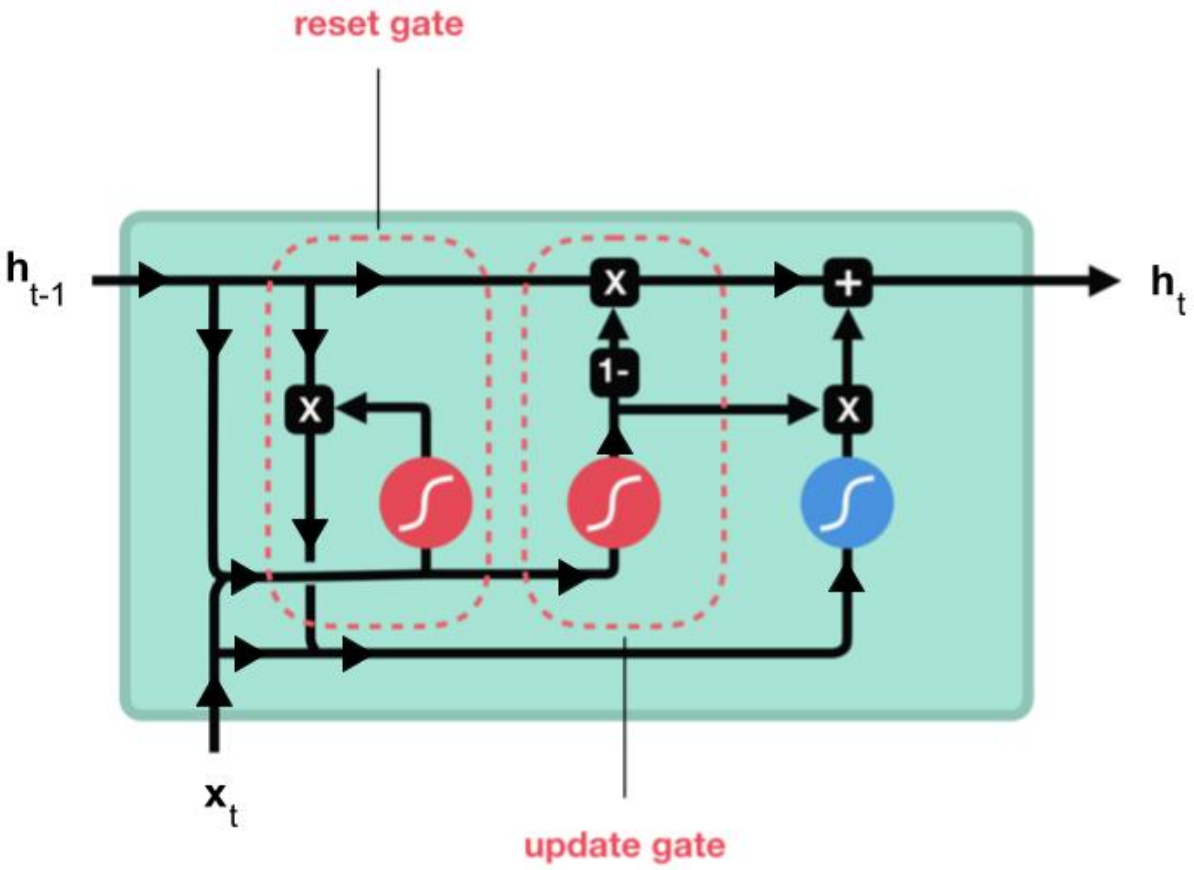

Figure 2 : Schéma représentatif d'une cellule GRU.

Dans la suite de notre projet nous avons utilisé une architecture SeqtoSeq avec deux GRU couplés à un mécanisme d'attention. Plus précisément une attention de type "dot product attention". Le but est de fournir encore plus de contexte en combinant les différents états cachés de chaque cellule de l'encodeur à l'aide d'une couche neuronale. Cette couche va apprendre à ajuster ses poids en fonction des cellules plus ou moins intéressantes à considérer, et pouvoir ainsi modéliser plus en profondeur les relations entre les mots dans une phrase.

Un autre bénéfice est de pouvoir afficher des cartes d'attention représentant les poids de cette couche d'attention et leurs activation en fonction des mots. On peut ainsi voir en les mots qui ont été plus considérés en entrée dans la génération d'un mot en sortie.

Ces cartes d'attention sont intéressantes car l'on peut comprendre sur quels mots le réseau de neurones s'est basé pour produire des résultats cohérents ou au contraire générés des résultats incorrects.

ii. Implémentations et résultats expérimentaux

Nous pouvons analyser les cartes d'attention en plus des tweets générés et observer comment le réseau se comporte avec certaines entrées. Des cas sont plus probants que d'autres et on peut remarquer sur les cartes d'attention que certaines entrées permettent d'activer de plus grandes valeurs de poids et des poids plus nombreux. Comme par exemple avec l'exemple suivant :

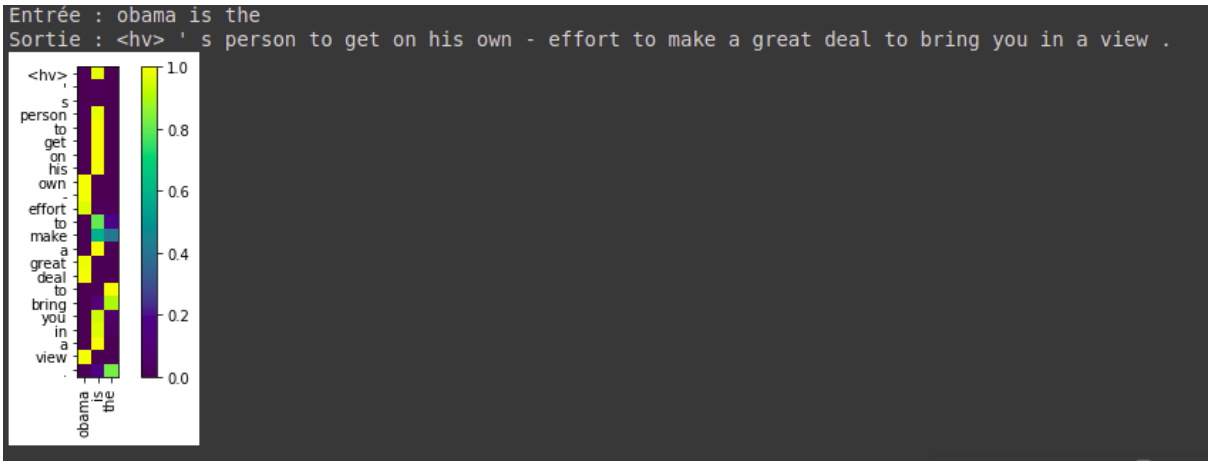

Figure 3 : Tweet 1 généré par le GRU avec attention et sa carte d'attention.

Nous pouvons voir que le mot "obama" a fait générer au réseau de neurone les mots "view", "deal", "great", "effort", "-" et "own". Il est intéressant de constater que le mot "is" a entraîné la génération de "person", car il existe une relation entre les deux dans ce cas ci avec la

personne de Barack Obama. Quant bien même le plus souvent le modèle génère des <hv> “hors vocabulaire” en se basant sur “is”.

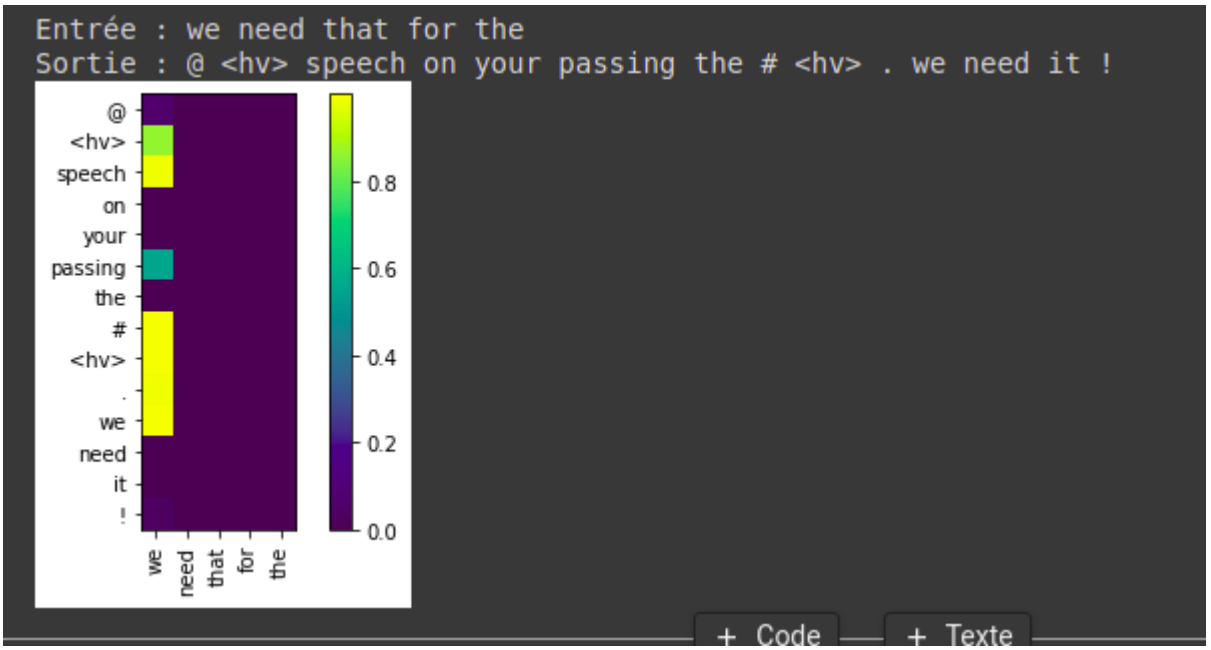

Figure 4 : Tweet 1 généré par le GRU avec attention et sa carte d’attention.

Pour cette carte d’attention au contraire on peut clairement voir que seulement le mot “we” a permis d’activer des poids et de générer certains mots et non le reste des entrées. Nous pouvons voir clairement que les entrées ont activé peu de poids du réseau, ce qui nous permet de comprendre pourquoi le réseau a eu du mal à générer un tweet cohérent.

Nous pouvons voir ensuite d'autres cas où les poids du réseau ont été faiblement activés et qui ont produit des phrases syntaxiquement peu cohérentes.

Figure 5 : Tweet 2 généré par le GRU avec attention et sa carte d’attention.

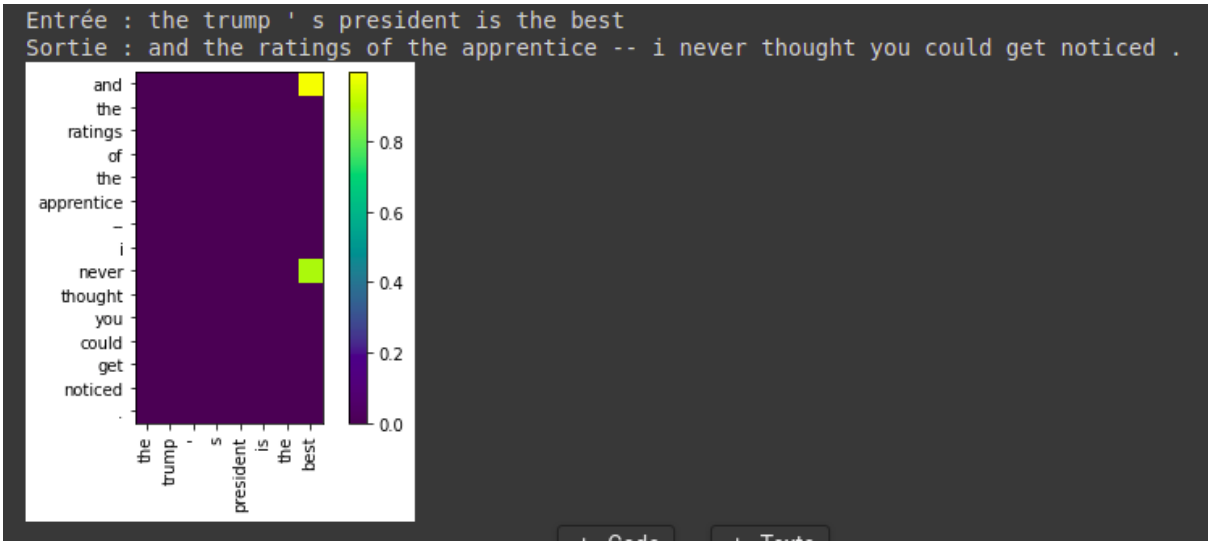

Figure 6 : Tweet 3 généré par le GRU avec attention et sa carte d'attention.

La figure 6 montre comme dans le cas des cartes d'attentions aux dessus (Figure 4 et 5) des poids peu activés. On voit sur la carte d'attention que "best" a permis de générer la conjonction "and" pour faire la jonction entre les deux parties du tweet mais aussi l'adverbe de temps "never" qui souligne le plus souvent les superlatifs utilisés dans les tweets de l'ancien président américain.

Figure 7 : Tweet 4 généré par le GRU avec attention et sa carte d'attention.

Les cartes d'attention nous permettent clairement de voir ainsi comment la partie attention du réseau de neurones a fonctionné pour générer des nouveaux mots en sortie. Nous pouvons comprendre les relations qu'a pu encoder la partie attention du réseau de neurones.

4) Modèle de chaîne de Markov

Le président américain utilise très largement la ponctuation et les majuscules pour transmettre ses messages. En constatant ça, nous nous sommes demandés si un modèle basé sur la génération de caractères et non sur la représentation vectorielle de mot pourrait fournir des résultats intéressants. Notamment fournir des hashtags et des mentions de personnes là où les modèles récurrents oblitèrent cette information.

En effet le vocabulaire utilisé pour entraîner les réseaux de neurones oblitèrent les hashtags, les mentions de personnes, les retweets, ainsi que les émoticônes. Aucune représentation vectorielle n'a été créée pour encoder ces informations.

i. Aspects théoriques

Nous nous sommes penchées sur les modèles de chaîne de Markov pour cela. Claude Shannon est le premier à avoir utilisé les chaînes de Markov pour créer un modèle statistique d'enchaînements des lettres dans un bout de texte (Shannon, C. (1948) [11]).

Un modèle de type chaîne de Markov pour le texte est un modèle permettant d'approximer la structure statistique d'un texte. Il existe plusieurs ordre pour ce type de modèle.
Un modèle d'ordre 0 fixe une probabilité fixe à chaque lettre d'apparaître dans le texte. Ce type de modèle se résume à faire un décompte de chaque lettre dans un texte et dresser ensuite une fréquence d'apparition. Ces fréquences sont ensuite utilisées comme probabilités d'apparitions pour générer un nouveau texte.

Le problème avec les modèles d'ordre 0 est qu'ils ne modélisent pas les relations statistiques que peuvent entretenir les lettres entre elles dans une langue telle que l'anglais. Les lettres sont tirées au sort de manière indépendante lors de la génération.

Nous avons donc eu recours à un modèle de Markov d'ordre n. Un modèle de Markov d'ordre n prédit que chaque lettre se produit avec une probabilité fixe, mais cette probabilité peut dépendre des n caractères précédents, que nous appelons un n-gramme.

Par exemple, supposons que le texte d'entrée ait 100 occurrences de «th», avec 60 occurrences de «the», 25 occurrences de «thi», 10 occurrences de «tha» et 5 occurrences de «tho». Ensuite, le modèle de Markov d'ordre 2 prédit que la lettre qui suit immédiatement toute occurrence de «th» est «e» avec probabilité 3/5, «i» avec probabilité 1/4, «a» avec probabilité 1/10 et « o 'avec probabilité 1/20.

Plus le nombre n est grand et plus la génération sera réaliste.

ii. Implémentations et résultats expérimentaux

Pour l'implémentation nous nous sommes basés sur un article de towardsdatascience.com ([12] Clark, T., (2020)). Cette implémentation permet de fournir les caractères en entrée, la longueur des n-grammes et la longueur du tweet.

n-gramme initial	longueur du n-gramme	longueur du tweet	Sortie
"Hill"	4	70	"Hillars they way use othere of in the phone who do sold at @Trump. Nice.If"
"Hillary"	7	65	"Hillary Clinton like him! I will never given? RT @newtgingrich on an int"
"Today I"	7	75	"Today I will be with living away from a person--there and falling me LIVE in Moon"
"Barack Obama"	12	50	"Barack Obama was president! RT @tashlutsa: @CLewandowski_: Wat"

Voici un extrait ci-dessus des tweets générés par notre modèle de Markov. La syntaxe est assez crédible tandis que la sémantique est plus ou moins plausible sur l'ensemble des tweets. Au-delà d'une séquence de 4, 5 mots, le tweet perd vite son sens sémantiquement. Ce qui est logique vu que le modèle de Markov n'encode pas le sens sémantique des séquences de mots.

Néanmoins nous pouvons constater que le modèle a bien réussi à générer des retweets et des mentions. Les mentions sont assez satisfaisantes, sur les tweets générés nous avons réussi à obtenir 2 mentions sur 3 qui sont de vrais comptes de personnalités politiques américaines comme @CLewandowski et @newtgingrich. Les initiales du retweet, RT, sont de plus placées au bon endroit avant les mentions et font sens dans le tweet.

Parmi les tweets que nous avons générés, les plus réalistes syntaxiquement et sémantiquement sont ceux avec en entrée un n-gramme assez long et une longueur de tweet relativement courte.

5) Test auprès des étudiants

Le but de ce test est de voir si les sorties générées par nos réseaux de neurones peuvent tromper des humains.

Nous avons récolté 17 avis sur une quinzaine de tweets. Nous avons mis 6 tweets de Donald Trump, 3 tweets provenant du LSTM, 3 tweets provenant du GRU avec le mécanisme d'attention, et 3 tweets avec le modèle de Markov. Nous les avons placés au hasard dans le formulaire pour ne pas biaiser les résultats.

En moyenne le modèle de Markov a réussi à tromper 25,3% des participants, le LSTM, 39,3% des participants et le GRU avec attention 59 %. A noter que les participants ont confondus en moyenne 41,3% des tweets de Donald Trump placé dans le formulaire avec des tweets générés par des réseaux de neurones.

Nous pouvons constater, malgré le petit échantillon de participants, que les réseaux de neurones récurrents avec le système d'attention réussissent mieux à générer des tweets vraisemblables capables de tromper des humains.

6) GPT-2

GPT-2 (Generative Pre-Trained) est un modèle pré-entraîné développé par OpenAI basé sur l'architecture transformeur [17].

Ce modèle successeur de GPT, a été entraîné sur un corpus de 40GB de texte extrait sur internet, dans le seul but de prédire le "prochain mot" sachant la séquence de mots précédente. Par exemple, le modèle cherche à prédire le mot le plus probable en complétant la séquence suivante [citation de l'article web] : "Le train INOUI 6798 rentre en gare veuillez vous... [1- Éloigner][2- Ecarter][3- Pousser][4- Jetez]..." Le modèle cherche donc à répondre parmi les mots suivants: lequel sera le plus probable.

La base de données de GPT-2 est composée de 8 millions de pages web et le modèle GPT-2 est composé de 1.5 milliard de paramètres.

i. Architecture transformeur

Comme les modèles récurrents de type séquence à séquence, les transformeurs se composent d'un encodeur et d'un décodeur. A la différence des modèles récurrents, les transformeurs font abstraction de la récurrence et n'ont gardé que le mécanisme d'attention "Attention Is All You Need" (Vaswani,A., et al. (2017) [5]).

- Encodeur :

Dans une architecture transformeur, l'encodeur est composée de :

- une couche d'entrée,
- suivie d'une couche de représentation vectorielle des mots,
- puis un ensemble de N couches empilées. Les couches empilées sont composées de deux sous-couches chacune. La première sous-couche est une sous-couche

d'auto-attention à multi-tête et la deuxième sous-couche est une sous-couche "Feed Forward".

A la sortie de chaque sous-couche, il y' a une opération par une couche de normalisation de la somme d'entrée avec la somme la sortie de la couche: $layerNorm(x + sublayer(x))$

Le rôle et l'objectif de la sous-couche "Feed Forward" sont de traiter la sortie d'une couche d'attention de manière à mieux s'adapter à l'entrée de la couche d'attention suivante.

- Décodeur :
  Le décodeur est composé de :

- Une couche de représentations vectorielle des mots,
- puis de N couches empilées. Chaque couche est composée de trois sous-couches : une première sous-couche dite de "Attention à multi-tête cachée" qui applique le mécanisme d'attention aux sorties de la couche d'embedding. Ce masquage, combiné au fait que les sorties de la couche d'embedding sont décalées d'une position, garantit que les prédictions pour la position i ne peuvent dépendre que des sorties connues aux positions inférieures à i en d'autre termes, elle ne conserve que les mots produits en sortie jusqu'à cette étape.

  Les dernières sous-couches sont des sous-couches d'attention multi-tête et "feed forward" comme dans le cas de l'encodeur.
  Comme pour l'encodeur à la sortie de chaque sous-couche on applique l'opération suivante :
  $layerNorm(x + sublayer(x))$

- Le mécanisme d'attention dans un transformateur
  Comme pour des modèles à RNN, les transformeurs utilisent les mêmes mécanisme d'attention tel que "Dot Product Attention" à la seule différence, qu' une normalisation est présente ensuite. D'ou l'appellation de "Scaled Dot-Product Attention"

$$softmax(\frac{QK^T}{\sqrt{d_v}})V$$

Les transformeurs utilisent plusieurs couches d'attention et notamment plusieurs couches de "Scaled Dot-Product Attention" pour former les sous-couches d'attention à Multi-tête.

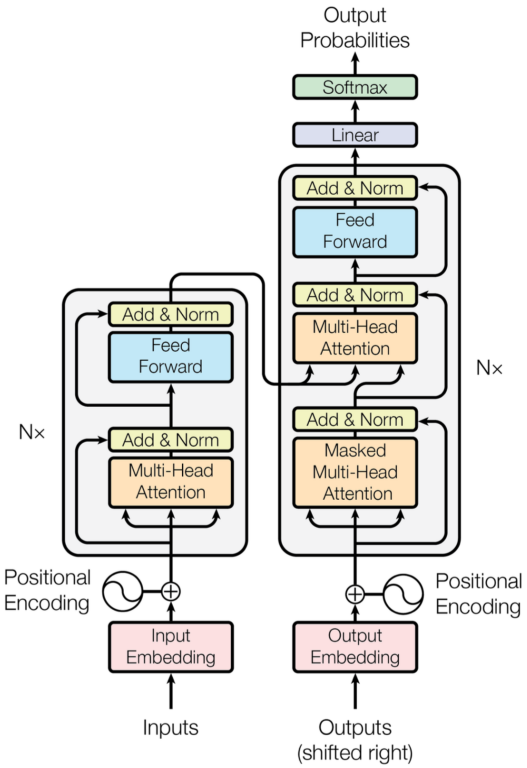

Figure 8 : Architecture d'un Transformeur [7]

- Que se passe-t-il dans une tête d'attention ?

Pendant l'attention, chaque vecteur de représentation du mot dont on cherche à encoder le contexte est multiplié avec des représentations vectorielles des autres mots de la séquence. Les vecteurs obtenus sont normalisés pour obtenir des poids. Chaque poids est ensuite multiplié avec des représentations vectorielles des mots par rapport leurs positions les vecteurs obtenues sont sommés pour obtenir le vecteur contexte.

A chaque étape d'auto-attention la représentation vectorielle des mots est utilisée 3 fois. Pour rendre la sous-couche flexible et donc entrainable, les entrées que représente la couche d'embedding de la séquence est multiplié par des matrices de poids *Wq, Wk, Wv* pour obtenir respectivement la matrice des requêtes (Query), la matrice des clés (keys) et la matrice des valeurs (Values)[7].

Dans un transformeur, il existe 3 type de mécanisme d'attention:

- L'attention input-input celle-ci se fait dans la sous-couche d'attention multi-tête des couches d'encodeur.
- L'attention input-output celle-ci se fait dans la sous-couche d'attention multi-tête du décodeur.
- L'attention output-output celle-ci se fait dans la sous-couche d'attention multi-tête avec masque.

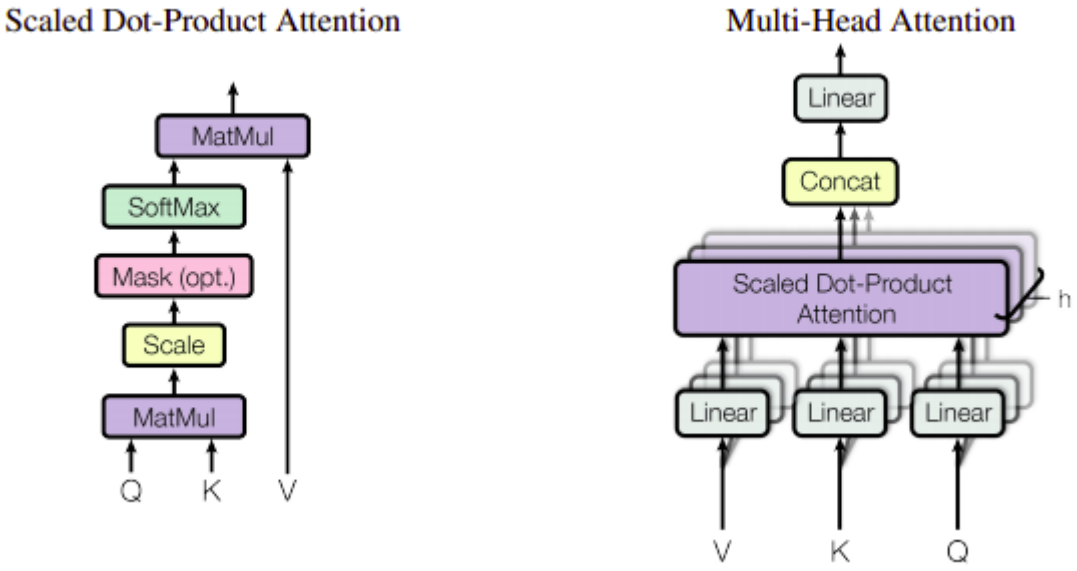

Figure 9 : Mécanisme d'attention dans un transformeur et attention à multi-tête[7]

ii. Parallélisation

L'un des avantages des architectures transformeurs est la parallélisation, en effet les opérations n'étant pas pour la plupart pas séquentielles, notamment le système à Multi-Tête et les sous-couche "Feed Forward"[11][12], l'exécution de leur opérations n'ont aucune dépendance entre elle et permettent de faciliter la parallélisation du modèle. Cela a permis un énorme gain de temps dans l'entraînement par rapport aux modèles de type RNN.

iii. Généralisation et transférabilité

Un énorme avantage des modèles de langage pré entraîné est leur capacité de génération et de transférabilité. La spécialisation des modèles pré-entraînés pour différentes tâches de NLP se fait de plusieurs manières. Jacob Delvin, et al.(2019)[15] décrit deux stratégie de spécialisation des modèles pré-entraînés: le *feature-based* et le *fine-tuning*.

Généralement, c'est le peaufinage ou "*fine-tuning*" qui est le plus utilisé pour entraîner une les modèles pré-entraînés.

Brown et al(2020)[16]. décrivent l'approche du "*fine-tuning"* comme étant une approche qui consiste à mettre à jour les poids d'un modèle pré-entraîné en effectuant un entraînement sur un ensemble de données supervisées spécifiques à la tâche souhaitée. L'un des principaux inconvénient de cette approche est qu'il faudrait disposer d'une base de données conséquente. Cette stratégie est décrite comme étant du *transfer learning* [13][14].

Une autre stratégie de spécialisation est le "*p-tuning"*. Cette technique permet elle de spécialiser un modèle de type unidirectionnel dans les tâches de compréhension de textes [17].

Conclusion et perspectives

Ce travail d'études et de recherches nous a permis d'approfondir les notions que nous avons en cours et d'avoir une expérience pratique dans la génération de texte. Nous sommes assez satisfaits des tweets obtenus au vu des réponses au formulaire.
Le mécanisme d'attention apporte une réelle plus value au niveau de la génération et au niveau de l'analyse grâce aux cartes d'attention. C'est l'architecture qui nous a permis de méprendre le plus de personnes *in fine*.

Nous avons quelques regrets de ne pas avoir eu le temps d'aller plus loin que la partie théorique concernant les modèles de langages pré-entraînés. Nous nous attendions à avoir des résultats assez probants et surpassants les autres architectures utilisées précédemment.

Une autre perspective intéressante également aurait été de couper le jeu de données en fonction des périodes de la présidence de Donald Trump et d'entraîner nos modèles avec. Nous aurions pu avoir des tweets plus marqués par certains champs lexicaux ou mentionnant des événements plus précisément. Il est possible que de cette façon les tweets obtenus auraient été d'autant plus crédible d'un point de vue humain.

Bibliographie

[1] Tensorflow, https://www.tensorflow.org/
[2] Keras, https://keras.io/
[3] Google Colab, https://colab.research.google.com/notebooks/intro.ipynb
[4] thetrumparchive.com, http://www.trumptwitterarchive.com/
[5] Hochreiter,S. et Schmidhuber, J. (1997) "LONG SHORT-TERM MEMORY".
[6] Cho,K., Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, Yoshua Bengio (2014) Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.
[7] Vaswani,A., Shazeer,N., Parmar,N., Uszkoreit,J., Jones,L., Gomez,A., Kaiser,L. et Polosukhina,I. (2017). "Attention Is All You Need".
[8] Rumelhart,D., Geoffrey, H., Williams,R. (1985) "Learning internal representations by error propagation".
[9] Michael,J., (May 1986) "Serial order: a parallel distributed processing approach.".
[10] Sutskever,I., Vinyals,O. et Le, Q. (2014) "Sequence to Sequence Learning with Neural Networks".
[11] Shannon, C. (1948) "A Mathematical Theory of Communication", Bell System Technical Journal.
[12] Radford, A., Wu, J., Child, R., Amodei, D., Sutskever, I. (2019) "Language Models are Unsupervised Multitask Learners", Arxiv
[13] Thomas Wolf, Victor Sanh, Julien Chaumond, Clement Delangue, (2019) "TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents".
[14] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., (2018) Improving Language Understanding by Generative Pre-Training.
[15] Devlin, J., Chang, M., Lee, K., Toutanova, K., (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
[16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, Dario Amodei (2020) Language Models are Few-Shot Learners
[17] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, Jie Tang (2020) GPT Understands, Too.
[18] blog https://lbourdois.github.io/blog/nlp/GPT2/
[19] Clark, T., (2020) "Generate Fake Donald Trump Tweets using Python", https://towardsdatascience.com/generate-fake-donald-trump-tweets-using-python-8d83885fd5c6