Rapport final projet 2.02

Paul Lieutier, Lisa Mollo, Nicolas Péruchot

Théo Poncelet, Anaïs Potier, Alexandre Pradeilles

1 Introduction

1.1 Client

Notre client est Matthieu Sénéchal qui travaillait l'année dernière chez Mieuxplacer.tech en temps que co-fondateur. Mieuxplacer.tech était une entreprise spécialisée dans la préparation de solution BtoB et M. Sénéchal était en charge de la gestion de l'avenir de l'entreprise.

1.2 Problème

Le problème rencontré ici par notre client était la difficulté à analyser des profils Linkedin. En effet, il souhaiterait que les informations présentes sous forme de listes dans les profils soient présentées au recruteur sous la forme d'un texte clair et compréhensible lui permettant de se faire une idée globale de la personne. L'objectif de ce projet est donc d'implémenter un programme permettant de générer automatiquement des biographies à partir d'informations récupérées sur les pages Wikipédia (plus précisément, à partir des infobox des pages). Ce programme pourra ensuite être réutilisé par notre client et adapté pour générer toutes sortes de textes (notamment à partir de Linkedin).

2 Etat de l'art

Nous avons échangé avec le groupe ayant travaillé sur ce projet l'an dernier, ce qui nous a permis d'avoir une base de départ. En particulier, nous avons récupéré le dataset constitué l'an dernier, ce qui nous permettra d'entraîner et de tester nos modèles sans avoir à se soucier de récupérer des données. Ensuite, le groupe nous a expliqué que les réseaux de neurones récurrents type LSTM ne sont pas suffisamment complexes pour ce genre de

tâches. Nous nous sommes donc intéressés aux Transformers, un type de réseaux de neurones récent, utilisé en NLP et décrit par l'article *Attention is all you need* [5].
Chaque groupe de 2 s'est ensuite instruit sur un type de modèle différent :

2.1 Modèle de type CGE

Comme décrit précédemment, le modèle CGE (Cascaded Graph Encoder) permet avec des graphes de connaissances en entrée (Knowledge Graphs) de générer des textes de plusieurs phrases avec à la fois une cohérence locale, pour la formation de la phrase et l'agencement des mots, et globale, pour le sens général du texte.
C'est d'après nos recherches le premier article qui traite des deux aspects à la fois, mêlant ainsi les avantages des Transformers [5], et ceux des méthodes AMR-to-Graph (Abstract Meaning Representation) qui se concentrent trop sur le local. Comme son nom l'indique, le modèle CGE fonctionne "en cascade", c'est-à-dire qu'il encode d'abord l'aspect global puis réarrange localement la sortie.
Dans le plus récent article portant sur le modèle CGE [4], ce dernier est comparé à un modèle qui gère ces deux aspects en parallèle et les chercheurs penchent finalement pour le CGE qui obtient de biens meilleurs résultats (cela est déterminé de manière empirique).

Après quelques essais, nous avons choisi de ne pas travailler sur ce modèle.

2.2 Modèle de type T5

Le modèle T5 à été conçu par les équipes de Google. Il est inspiré de BERT, le précédent modèle de Google, pour sa structure mais des éléments ont été modifié afin de satisfaire ses tâches. Le modèle à été dévoilé au public en 2020 dans un article intitulé *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer* [2]

Comme nous l'avons vu, le modèle T5 est multi-tâche. Cela signifie que l'on peut l'utiliser pour tout besoin nécessitant une transformation d'un texte vers un autre texte.

Pour arriver à cela, les auteurs de l'article ont dû créer un ensemble colossale de textes pour l'entrainement appelé C4. Ce jeu de données fait deux fois la taille de wikipédia et contient des textes de natures très différentes contrairement à wikipédia dont les textes sont relativement lisses. Ce dataset

est mis à disposition tout comme le modèle T5.

L'objectif de cette équipe de Google, était d'explorer les limites du transfert learning qui est une technique très puissante en NLP et qui est très utile pour accélerer le développement de projets qui n'ont pas les moyens techniques d'entrainer leur modèle sur d'aussi grandes quantités de données. Grâce à leur immense dataset et aux modifications par rapport à BERT, ils ont ainsi pu mettre à disposition du public un modèle qui surperforme selon de nombreux indicateurs par rapport aux modèles plus anciens.

Ce modèle, permettant entre autres de completer des phrases, semble donc indiqué pour notre problématique. En réalisant du fine-tuning grâce à des données sous le format qui nous interesse, nous pouvons espérer atteindre des résultats satisfaisant.

2.3 Modèle de type GPT

Ce modèle de génération automatique du langage a été conçu par le laboratoire de recherche Open IA. Il a été rendu disponible au grand public en 2019. Le papier de publication du modèle un article intitulé *Language models are unsupervised multitask learners* [1]

Comme expliqué précédemment, ce modèle est capable de générer des séquences de texte ayant une cohérence syntaxique, grammaticale et informative. Il peut comprendre un texte, répondre à des questions dessus, ou encore écrire la suite d'un texte. La particularité de ce modèle est qu'il est capable de réaliser une grande diversité de tâches sans même avoir été spécifiquement entrainé à les faire.

En effet, la majorité des modèles de NLP sont aujourd'hui entrainés sur des données d'entrainement ayant un comportement relatif à une certaine tâche. Le modèle est ainsi entrainé à réaliser cette tâche puis testé et évalué selon sa réussite à réaliser la tâche demandée.

Le postulat des scientifiques ayant travaillé sur le développement de ce modèle était que la prévalence de l'entraînement du modèle à une seule tâche sur des ensembles de données à un seul domaine implique un fort manque de généralisation.

Le modèle utilise la modélisation du langage qui est une estimation non supervisée de la distribution des mots à partir d'un ensemble d'exemples. L'architecture d'auto-attention, comme par exemple les réseaux de neurones de type Transformer, a beaucoup simplifié la recherche des probabilités de la présence des mots dans la sortie de l'algorithme. En effet, l'apprentissage d'une tâche unique peut être exprimé dans un cadre probabiliste comme l'estimation d'une distribution conditionnelle p(sortie|entrée). Dans le cadre

de ce modèle, le but est de modéliser p(sortie|entrée, tâche).

Pour réaliser cela, on utilise la méthode d'apprentissage Zéro-shot. Le Zéro-shot vise à résoudre une tâche sans recevoir aucun exemple de cette tâche lors de la phase d'apprentissage.

Cette méthode nécessite le word embedding, qui consiste en associer un vecteur de nombres réels à chaque mot du dictionnaire, les mots apparaissant dans des contextes similaires ayant des vecteurs proches les uns des autres.

Ce modèle semble avoir de très bons résultats en ce qui concerne la compréhension de texte, le résumé de texte et la réponse à des questions.

Il n'a jamais été entrainé ou testé sur la tâche que nous souhaitons réaliser dans ce projet, à savoir, l'écriture d'un texte à partir de mots isolés.

Nous pouvons donc émettre quelques réserves quant à la réussite de cet algorithme à fournir des résultats satisfaisants sur ce sujet. Mais sa composante multitâche nous pousse à mener à bout l'expérience afin de comparer notre résultat aux autres modèles présentés ci-dessus.

3 Travail réalisé

3.1 Traitement du dataset

Comme nous l'avons précisé plus haut, nous avons récupéré les données du groupe de l'an dernier. Chaque élement de ce dataset est un ensemble d'informations sur une personnalité (nom, métier, nationalité...), associé à une phrase biographique. Cependant, nous avons remarqué que dans la plupart des cas, les phrases ne correspondaient pas du tout aux éléments en entrée : certaines informations n'y figurent pas et d'autres sont totalement inventées. On comprend donc que les informations que le modèle doit prendre en entrée n'ont pas été extraites des phrases mais ont été obtenues indépedemment.

Par exemple, avec comme entrée les informations : " Stevie Wonder | American | Pianist", on pouvait avoir comme phrase associée "Stevie Wonder was a musician from Michigan". Le dataset ne peut pas être utilisé tel quel pour entraîner notre modèle. Nous l'avons donc retravaillé.

Nous sommes partis des phrases complètes et nous avons extrait des informations de ces phrases, qui serviront d'entrée au modèle. De cette manière, nous sommes sûrs que les entrées et les phrases générées sont liées. Pour ce faire, nous avons utilisé la libraire spaCy, particulièrement performante pour la reconnaissance d'entités nommées, ou Name-Entity recognition (ner) en

anglais.

Pour reprendre notre exemple, nous avons extrait de la phrase "Stevie Wonder was a musician from Michigan" les informations "Stevie Wonder | musician | Michigan".

3.2 Le modèle GPT2 avec du fine-tuning

3.2.1 Description du modèle

Le modèle GPT2 est un modèle de NLP se basant sur des méthodes de Machine Learning d'apprentissage supervisé. Une de ses particularités est de renvoyer en sortie un token à la fois. Son architecture est construite en utilisant des blocs de décodeur de neurone transformer. L'architecture est présentée en figure 2.

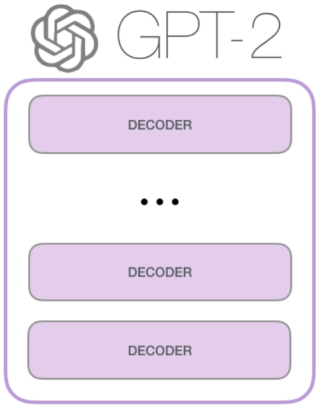


FIGURE 1 – Architecture du modèle GPT2

Le modèle fonctionne en plusieurs étapes qui sont les suivantes :

Étape 1 : Prétraitement de l'entrée Avant de passer par les blocs de transformer, chaque mot en entrée est traité de la manière suivante : Tout d'abord, on regarde l'embedding du token en entrée dans la embedding matrix (chacune de ses lignes correspond à l'embedding d'un mot dans le dictionnaire). Puis on y incorpore son positional encoding. Enfin, on associe un vecteur de nombres réels à chaque mot du dictionnaire, les mots apparaissant dans des contextes similaires ayant des vecteurs proches les uns des autres.

Étape 2 : Blocs de transformeur Le processus est identique dans chaque bloc décodeur, mais chacun des blocs a ses propres poids à la fois dans ses sous-couches de self-attention et de réseaux de neurones. Une particularité du bloc décodeur est que la couche d'auto-attention n'est autorisée qu'aux positions antérieures de la séquence de sortie. Cela se fait en masquant les positions futures (en les mettant à -inf) avant l'étape softmax du calcul d'auto-attention.

Étape 3 : Traitement à la sortie des blocs de transformer Lorsque le bloc supérieur produit son token de sortie (résultat de son propre mécanisme de self attention et son propre réseau de neurones), le modèle multiplie ce résultat par la embedding matrix. Le résultat de cette multiplication peut être interprété comme un score associé à chaque mot du dictionnaire du modèle. On n'a qu'un seul imput token car on ne prend que le premier score en compte. On le rajoute à l'imput token et on recommence jusqu'à l'obtention du token de fin.

Le fonctionnement du modèle GPT2 est illustré dans la figure 3.

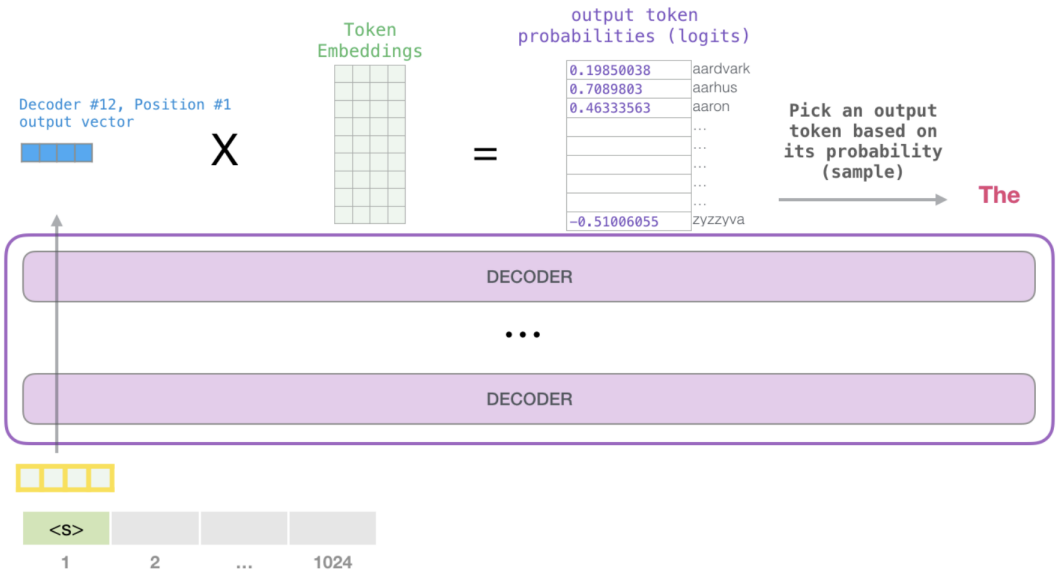


FIGURE 2 – Fonctionnement du modèle GPT2 (source : https ://jalammar.github.io/illustrated-gpt2/)

3.2.2 Comment fonctionne le fine-tuning ?

Le fine-tuning repose sur le fait de s'utiliser un modèle pré-entraîner puis de le réentraîner avec de nouvelles données. Cela permet d'éviter un temps d'entraînement trop long et de passer outre les bases de données un peu trop réduites.

Une des techniques utilisées pour le réentraînement est le frizing. Cette méthode consiste à "bloquer" certaines couches d'un modèle lors de l'entraînement empéchant ainsi leur modification. Les gradients à calculer sont donc moins nombreux.

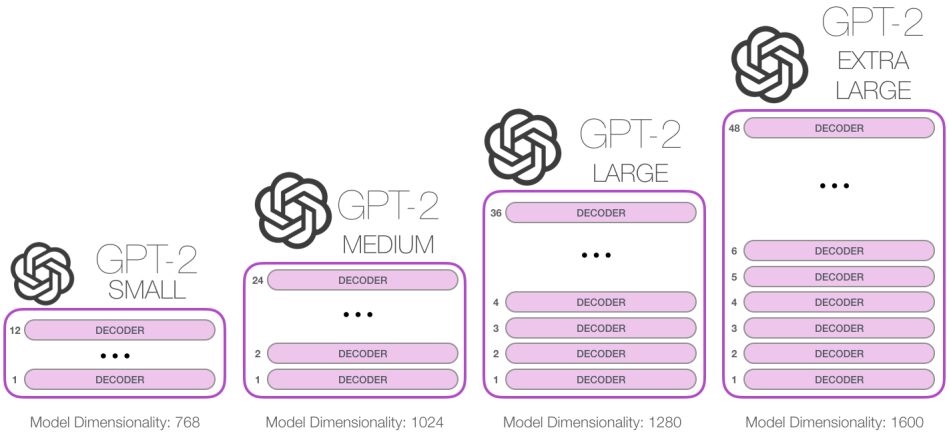


FIGURE 3 – Différents pré-trained modèles GPT-2

Pour notre modèle, nous allons utiliser une petite version du GPT-2 contenant 12 layers et nous allons seulement pré-entraîner les 6 premières layers en frisant les 6 suivantes. Cela permettra d'une part de conserver les qualités du GPT-2 intiallement entraîné sur plus de 8 millions de pages web tout en l'adaptant à la forme de nos données.

3.2.3 Les données et leur modification

La base de donnée fournie par les élèves ayant travaillé sur le projet création automatique de biographie l'an dernier était très complète et comportait beaucoup de données. Cependant, même après la modification évoquée en section 3.1, elle n'était pas adaptée à notre modèle. Nous avons donc du transformer les données une nouvelle fois pour que le modèle puisse les prendre en entrée.

En effet, la database que nous avions avait les données tous la forme d'un dictionnaire ayant pour clés : ['Unnamed : 0', 'txt', 'PERSON', 'JOB', 'JOB1', 'NORP', 'GPE', 'ORG', 'TENSE']. Cela représente un indice de numérotation des lignes, le texte extrait d'une biographie, ainsi que toutes les caractéristiques de la personne. Les valeurs associées à ces clés étaient une liste de toutes les données que nous avions. Par exemple : database['txt'] nous renvoyait tous les textes à notre disposition.

De notre côté, nous avions besoin de données ayant pour clé un indice et pour valeur un tuple du type : ('Jürgen Brecht', 'Jürgen Brecht is a German fencer', ['Jürgen Brecht', 'fencer', 'German', 'is']). La première valeur est le nom de la personne, la deuxième la phrase considérée et la troisième une liste des mots importants et caractéristiques de la personne.

3.2.4 L'implémentation

Après avoir ainsi mis en forme les données, il nous reste à implémenter le modèle. Pour cela, nous nous sommes inspirés d'un article détaillé de Ivan Lai. [3]

L'implémentation du modèle se fait en trois étapes principales :
1. La définition du modèle
2. Le fine-tuning avec la classe python Trainer
3. La génération de texte

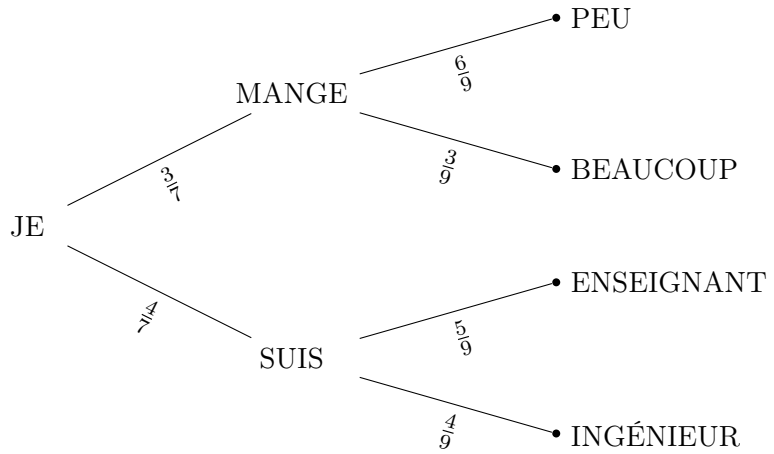
Nous allons détailler ces étapes ici. Notre code est disponible en annexe avec les commentaires.

Comme expliqué précédemment, nous utilisons ici le modèle GPT-2 dans sa version avec 12 couches dont nous avons geler les paramètres des 6 dernières couches. Cela nous permettra de conserver les caractéristiques apportées par le pré-entraînement.
Pour ce qui est de la pratique, nous avons simplement fait appel aux fonctions tokenizer et model du module transformers avec des paramètres nous permettant d'inclure des caractères spéciaux.

Pour le Fine-Tuning, c'est-à-dire l'entraînement partiel du modèle sur notre data (permis par le freeze des 6 dernières couches) va rendre le modèle plus adapté à notre problème. On utilise pour cela la classe Trainer de transformers qui fait le travail pour nous !

Enfin, pour la génération du texte, nous avons optez pour deux algorithmes différents : le "Beam-Search" ainsi que le "Top-p Nucleus". [6] Lorsqu'on a le modèle on est capable de prédire la suite de mots la plus probable à venir sachant les mots précédents. En prenant en exemple le mot "JE"

en début de phrase, on a donc affaire à un arbre de probabilité comme ci-dessous.



L'approche naïve dans cet exemple serait de ne regarder qu'une seule couche de mots à la fois et de prédire tout le temps le mot le plus probable soit ici : JE SUIS ENSEIGNANT. En réalité, en regardant deux couches, on voit que c'est la séquence JE MANGE PEU qui a une probabilité plus grande d'être correcte (selon le modèle). C'est la manière dont raisonne l'algorithme "Beam Search" qui prend entre autres, comme paramètres, un entier correspondant au nombre de couches à considérer pour prédire une séquence. Dans notre cas, nous avons pris cet entier valant 5.
La deuxième méthode utilisée, le "Top-p Nucleus", est un type de sampling. Cette catégorie de méthodes sélectionne aléatoirement le mot suivant selon la distribution de probabilités conditionnelles. Plus précisément, dans ce cas, il fait son choix parmi un ensemble minimal de mots tels que la somme des probabilités conditionnelles d'apparition de ses mots est supérieure à p. Dans notre modèle, nous avons pris $p = 0.7$.

3.2.5 Les résultats

Nous attendons en ce moment les résultats. Le script et les données ont été envoyés au mésocentre de Paris-Saclay. Néanmoins, en entraînant le modèle sur peu de données, il ne différait pas beaucoup de l'algorithme GPT-2 sans le Fine-Tuning et était très souvent hors sujet et composait des phrases très longues sans y inclure les mots-clés. Il ne correspond ainsi pas réellement à la demande du client.

3.3 Le modèle T5

3.3.1 Implémentation du modèle

Nicolas et Alexandre ont travaillé sur le modèle T5.

Nous avons dans un premier temps implémenté sur Tensorflow l'algorithme d'entrainement grâce à la bibliothèque Hugging Face. Celle ci nous à permis d'une part de récuperer le modèle T5 pré-entraîné, mais également le tokenizer adapté étant donné que l'encodage des mots est primordial en NLP. Cette partie de la tâche nous à pris un peu de temps à cause des problèmes de syntaxe et le temps de comprendre comment s'effectuait la mise à jour des poids du modèle avec cette bibliothèque.

Une fois l'algorithme d'entrainement fonctionnel, nous avons commencé par déterminer la valeur des hyperparamètres de l'apprentissage supervisé. Le premier que nous avons observé, est la taille des batchs. Dans un premier temps, nous avons observé le comportement de l'apprentissage avec des batchs de petite taille, pour des raisons de puissance de calcul, mais nous nous sommes rapidement aperçu que cela provoquait un apprentissage trop instable. En effet, l'entrainement sur une unique phrase pouvait provoquer un changement significatif des poids du modèles et ainsi modifier les phrases générées par rapport au dernier entrainement effectué ce qui n'était par satisfaisant. Afin de palier à cet effet stochastique, nous avons augmenté leur taille jusqu'a 32, ce qui permet d'obtenir un effet de moyenne sur plusieurs phrases et donc d'augmenter de façon plus générale la performance du modèle.

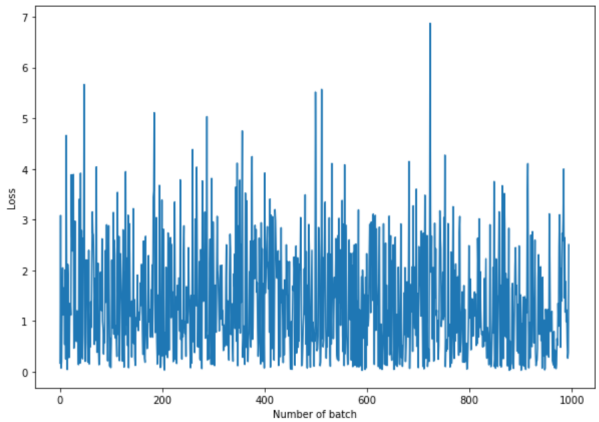


FIGURE 4 – Evolution de la fonction de perte pour des batchs de taille 1

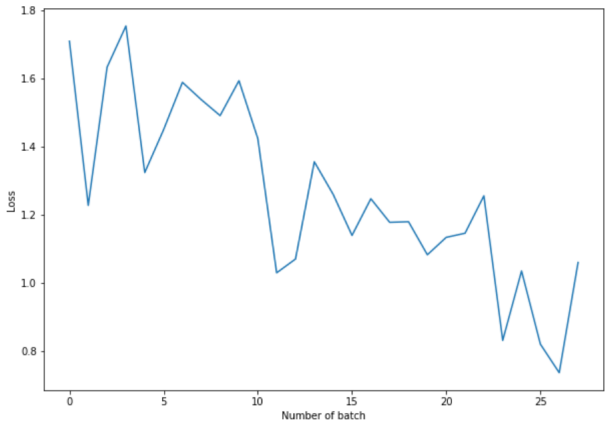


FIGURE 5 – Evolution de la fonction de perte pour des batchs de taille 32

Le second hyperparamètre, est le nombre d'époque que nous souhaitons effectuer. Pour celà, nous avons séparé notre dataset en deux ensembles, 80% pour l'entrainement et 20% pour la validation. Si l'entrainement cherche à minimiser la fonction loss sur l'ensemble d'entrainement, nous allons réaliser des époques tant que la fonction loss sur la validation continue de diminuer pour éviter l'overfitting. En utilisant nos ordinateur, il nous fallait 20 minutes pour entrainer sur 1000 exemples du dataset. Lorsque l'on sait que le dataset comprend plus de 300 000 exemples, on se rend bien compte que ce n'est pas envisageable de pouvoir entrainer notre modèle sur un ordinateur classique. Nous avons pu avoir accès à une machine de l'école, ce qui à permis de réaliser plusieurs époques.

3.3.2 Les résultats

Le modèle obtenu, permet de générer des phrases grammaticalement correctes qui utilisent toutes les informations données en entrée. Si l'on compare le modèle entrainé avec 20 000 exemples et le modèle final, on se rend compte que la construction des phrases est beaucoup plus rubuste car les informations sont mieux placées les unes par rapport aux autres avec des connecteurs plus cohérent, et il y a moins d'oubli d'informations. Cependant, dans les deux cas, on remarque l'ajout d'informations non présentes en entrée duent à la similarité entre un exemple vu lors de l'apprentissage et un élément voulant être génerer. En revanche, si on peut tirer quelque chose de positif, c'est que les ajouts fait par le modèle plus entrainé sont des éléments qui ont un rapport avec le contexte de la phrase lorque dans l'autre cas, les

ajouts sont beaucoup moins cohérents. Pour chiffrer cette progression, nous pouvons utiliser la fonction de perte sur les prédictions réalisées sur dataset de test. Pour le premier modèle, nous obtenons 0.699 alors pour que le second modèle nous avons 0.498 sois une progression de plus de 25%.

3.3.3 Livrable

Notre livrable prend la forme d'un container docker, ce qui permet de partager facilement notre modèle entraîné.

4 Conclusion

Par manque de temps, nous n'avons malheureusement pas pu comparer les deux modèles.
Néanmoins, l'algorithme T5 a donné de plutôt bons résultats trandis que le GPT-2 n'a pas été en mesure de fournir des séquences de mots qui contenaient les mots-clés et qui retranscrivaient ne serait-ce qu'un peu le sens qui s'y cachait. Ainsi, on peut pour l'instant raisonnablement opter pour le modèle T5.

Cependant, il reste des points d'amélioration sur lesquels pourraient travailler d'autres groupes l'an prochain. Notamment, l'implémentation de réseaux antagonistes génératifs, en anglais "generative adversarial networks", pourrait améliorer l'évaluation du score des modèles. De plus, le dataset sur lequel les modèles sont entraînés peuvent également être amélioré.

Ce projet nous a permis de tous progresser en deep learning et plus particulièrement en NLP. Certains membres du groupe vont approfondir leur connaisance en IA par des stages ou des spécialisations.

Références

[1] Rewon Child David Luan Dario Amodei Alec Radford, Jeffrey Wu and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

[2] Adam Roberts Katherine Lee Sharan Narang Michael Matena Yanqi Zhou Wei Li Peter J. Liu Colin Raffel, Noam Shazeer. Exploring the limits of transfer learning with a unified text-to-text transformer, 2020.

[3] Ivan Lai. Conditional text generation by fine tuning gpt-2. https ://www.ivanlai.project-ds.net/post/conditional-text-generation-by-fine-tuning-gpt-2. Accessed : 2021-06-01.

[4] Claire Gardentx Leonardo F. R. Ribeiroy, Yue Zhangz and Iryna Gurevychy. Modeling global and local node contexts for text generation from knowledge graphs, 2020.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[6] Patrick von Platen. How to generate text : using different decoding methods for language generation with transformers. https ://huggingface.co/blog/how-to-generate. Accessed : 2021-06-01.