

GPTeirb : une méthode de génération de texte sans paramètres utilisant des compresseurs

Léo Paillé - Gatien Menaud - Alix Renoncourt

EI5IS102 - Traitement de l'Information
Département informatique
S5 - 2023/2024



Table des matières

1	Introduction	2
2	Présentation des données	2
3	Présentation de la méthode	3
3.1	Création des exemples	3
3.2	Création du modèle	3
3.3	Optimisations du calcul de la matrice de NCDs	4
3.4	Utilisation du modèle	4
4	Analyse du modèle et des résultats	5
5	Conclusion	5
6	Références	5

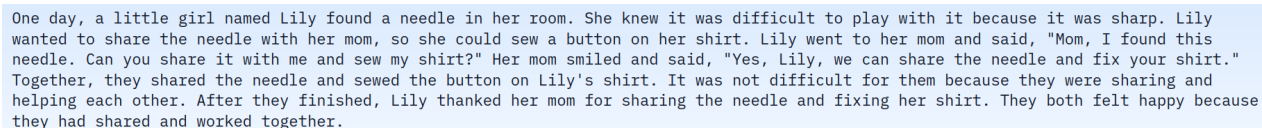
1 Introduction

Dans ce rapport, nous allons décrire le projet que nous avons décidé de réaliser. Il s'agit du traitement d'un jeu de données publiques issues de Kaggle, ainsi que de la création d'un modèle de génération de texte. Nous avons d'abord formaté les données pour créer des exemples d'entraînement, puis nous avons utilisé une approche nouvelle basée sur KNN et gzip dans le but de générer du texte. Cette approche se base sur l'article "Low-Resource" Text Classification : A Parameter-Free Classification Method with Compressors[?] et utilise le moyen de classification à la place d'un transformeur génératif pré-entraîné pour sélectionner le prochain token.

Cette méthode de traitement de données est semblable aux modèles de génération de texte par intelligence artificielle, tels que ChatGPT développé par la société OpenAI. Notre code est disponible à <https://github.com/Leikoe/gpt-eirb>.

2 Présentation des données

Pour créer un modèle capable de générer du texte à partir d'un mot ou d'une phrase (appelé *prompt*), il est tout d'abord nécessaire de l'entraîner à partir de texte préexistant. Nous avons décidé pour cela de récupérer un jeu de données *TinyStories.txt*[?] (une fois décompressé), contenant plusieurs histoires de quelques lignes, en anglais, totalisant plus de 1.6G de caractères. Notez qu'une partie non négligeable de ces textes a été elle-même générée par les modèles GPT-3.5 et GPT-4. Voici un échantillon du jeu de données en figure 1 :



```
One day, a little girl named Lily found a needle in her room. She knew it was difficult to play with it because it was sharp. Lily wanted to share the needle with her mom, so she could sew a button on her shirt. Lily went to her mom and said, "Mom, I found this needle. Can you share it with me and sew my shirt?" Her mom smiled and said, "Yes, Lily, we can share the needle and fix your shirt." Together, they shared the needle and sewed the button on Lily's shirt. It was not difficult for them because they were sharing and helping each other. After they finished, Lily thanked her mom for sharing the needle and fixing her shirt. They both felt happy because they had shared and worked together.
```

FIGURE 1 – Échantillon de *TinyStories.txt*

3 Présentation de la méthode

L'objectif étant de générer du texte, notre modèle doit être capable de déterminer le token (ici un caractère) suivant d'une suite de tokens. Comme dans le papier original, le but est de classer du texte, c'est-à-dire des échantillons de phrases (suites de tokens). Les classes sont tous les tokens possibles (comme pour un transformeur génératif). Nous avons choisi ici les caractères du texte d'entraînement.

3.1 Création des exemples

Un exemple est un couple (*contexte*, *cible*). Pour créer un de ces couples, nous générons un entier aléatoire que nous utilisons comme offset dans le texte de base, puis nous récupérons les *context_size* tokens suivants.

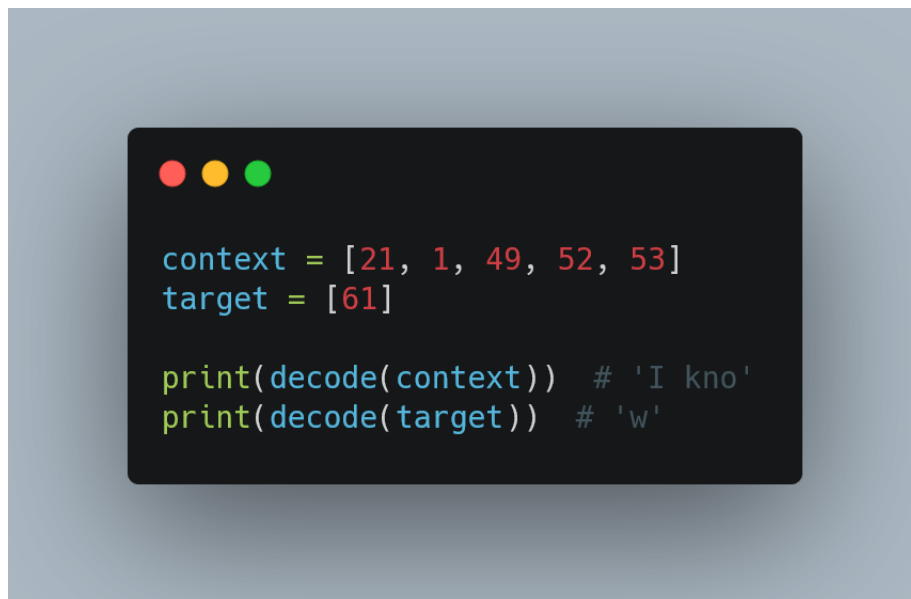


FIGURE 2 – *Un exemple d'entraînement*

3.2 Création du modèle

Pour créer le modèle, nous devons calculer une matrice de NCD (distances de compression normalisées). Chaque ligne de cette matrice est un vecteur qui correspond à un exemple d'entraînement. Le vecteur des NCD de cette ligne est calculé à partir des NCD entre l'exemple et tous les autres exemples.

On obtient donc une matrice de taille $N \times N$ où N est le nombre d'exemples.

		[Hello] World	[The] man	[A] woman	[NNs] are	[I eat] apples
I am a	22	87	92	14	53	

Exemple d'un vecteur de NCD

On peut observer que les suites de tokens avec un sens proche (et donc un token cible suivant logique) obtiennent un score plus élevé lors du calcul de la NCD. Ici, nous pourrions sélectionner "man" ou "woman" comme prochain token pour le prompt "I am a".

Pour cela, nous allons récupérer n_{train} échantillons, puis à partir de chacun, générer $context_size$ exemples. Pour chaque exemple, nous allons calculer un vecteur de NCD vers tous les autres exemples. Ce calcul est très lent et nous avons donc dû effectuer les compressions par batch sur GPU. Pour cela, nous avons utilisé nvCOMP[?], la librairie de compression propriétaire de NVIDIA. Cependant, cette librairie est en C++, et les seuls bindings à jour en python sont disponibles dans la librairie kvikio [?] de la suite RAPIDS[?] de NVIDIA.

3.3 Optimisations du calcul de la matrice de NCDs

Nous avons seulement besoin des tailles compressées des exemples et avons donc modifié la fonction de compression par batch de kivikio[?]. Après avoir mesuré le code, nous nous sommes aperçus que la majorité du temps était passé à copier chacun des strings vers la mémoire du GPU. Nous avons donc copié à la suite dans un buffer tous les strings de chaque groupe (*batch*), pour copier le buffer en une seule opération sur le gpu. Cela a grandement accéléré la génération du modèle.

3.4 Utilisation du modèle

Une fois que nous avons obtenu notre matrice correspondant à notre modèle, il est simple et rapide de faire générer du texte par un programme prenant en entrée un prompt écrit par l'utilisateur, ainsi que notre modèle, et générant en sortie du texte. Ce texte se développe alors, token par token de la manière suivante :

- On récupère les $max(taille_prompt, context_size)$ derniers caractères du contexte (le prompt suivi du texte généré jusque là)
- On calcule le vecteur de NCDs pour les $context_size$ derniers tokens du contexte. Cela donne une liste de flottants de longueur n_{train}
- On utilise un KNN pour trouver les lignes les plus proches du contexte actuel.
- On sélectionne le token suivant comme la classe prédite par le KNN
- On rajoute le nouveau token à la suite du contexte
- On réitère l'opération jusqu'à qu'une condition d'arrêt soit satisfaite (fin de phrase, nombre

de caractères requis par l'utilisateur atteint, arrêt forcé...)

4 Analyse du modèle et des résultats

Nous observons une génération médiocre, sans réel sens. Cela peut être dû à plusieurs raisons,

Premièrement, le compresseur utilisé est l'implémentation GPU de LZ4. Celui-ci ne possède pas un très bon facteur de compression contrairement à l'implémentation CPU de GZIP (celui par défaut en python). Cela influe très fortement sur les tailles compressées des exemples. Ici on observe que tous les exemples pour un *context_size* entre 4 et 16 ont une taille compressée tournant autour de 260 octets, ce qui rend très difficile pour le KNN de faire la différence entre les divers exemples.

Deuxièmement, nous avons utilisé un *tokenizer* de niveau caractère. Nous obtiendrons probablement de meilleurs résultats avec un *tokenizer* basé sur SentencePiece [?]. Cela aiderait énormément la qualité de génération pour des petites valeurs de *context_size*.

Troisièmement, l'approche utilisée ne s'échelonne pas bien. La taille du modèle est quadratique par rapport à la taille des données sur lesquelles il est entraîné. Cela devient vite bien trop grand, peut-être qu'un gigantesque modèle de cette architecture parviendrait à formuler des phrases.

5 Conclusion

Ce travail nous a permis de mettre en pratique des connaissances en programmation Python, mais aussi en traitement d'informations. Nous avons pu manipuler un grand corpus de données textuelles non labellisées afin de créer un modèle, permettant par la suite de générer du texte. Nous n'avons cependant pas pu atteindre cet objectif dans son entièreté, notamment par manque de ressources numériques, telles que de la puissance de calcul et de l'espace mémoire.

Nous avons tout de même pu approfondir nos connaissances, à la fois dans l'écriture du code, que dans les recherches qui nous ont permis de le réaliser. Nous avons pu utiliser divers outils mis à notre disposition depuis le web, en se basant plus particulièrement sur des projets communautaires ayant déjà traité ce sujet et des vidéos abordant le thème [?]. Pour conclure, le développement de l'intelligence artificielle est un enjeu majeur dans le monde du numérique d'aujourd'hui, surtout en ce qui concerne les discussions textuelles, et nous avons pu profiter de ce projet pour tenter de créer notre propre modèle de langage, nous permettant d'explorer en profondeur et d'en apprendre davantage sur ce sujet.

6 Références