

DL Language Model

Maxime Lefranc

December 2023

1 Introduction

L'objectif de ce mini-projet est d'implémenter un réseau de neurones appelé "Transformers" en ce concentrant seulement sur l'aspect décodeur. Ce type de réseau de neurones se caractérise par une architecture particulière, pouvant s'exprimer par une succession d'appels de classes et de fonctions et donc facilement implémentable. Nous parlerons plus en détails de cette architecture dans la section "Explication de code".

2 Détails techniques

Ce projet est composé de plusieurs fichiers :

- **model.py** : contenant les classes essentielles a l'implémentation de notre transformer.
- **gptlike.py** : permettant d'effectuer l'entraînement de notre transformer.
- **generate.py** : permettant de générer du texte Shakespearian a partir d'un modèle déjà entraîné.
- **utils.py** ; contenant toutes les fonctions relatives aux fichiers, plots etc.

Comment entraîner un modèle ?

Il suffit d'exécuter "gptlike.py". Le temps d'exécution peut être dratisquement différent en fonction de nombre d'itérations souhaité, des paramètres de notre configuration ou encore de quelques autres éléments que nous évoqueront plus en détails dans la partie "Résultats".

Comment sauvegarder un modèle ?

Il est possible d'utiliser la fonction "save_model" directement après avoir entraîné notre modèle pour le sauvegarder. Cela permet notamment de ne pas devoir entraîner notre modèle avant chaque génération. Cette fonction possède deux paramètres, le modèle a sauvegarder ainsi que le nom que vous souhaitez donner au fichier sauvegarder.

Comment charger et utiliser un modèle ?

Il suffit simplement d'exécuter le fichier "generate.py". Cela permet, dans un premier temps de charger un modèle en spécifiant le chemin relatif vers ce modèle, mais aussi d'utiliser ce modèle afin de prédire du texte Shakespearean d'une longueur souhaitée.

Comment passer de GPU à CPU et vice-versa ?

Suivant l'utilisation de ce code, il peut être nécessaire de savoir comment passer d'une execution GPU à une execution CPU avec torch. Pour faire cela, il suffit de modifier le paramètre "self.device" de notre configuration avec la valeur suivante :

- **torch.device('cuda')** pour du GPU.

- `torch.device('cpu')` pour du CPU.

Quels sont les modèles disponibles ?

Il existe plusieurs modèles disponibles, les éléments qui vont changer d'un modèle à un autre sont :

- Les paramètres de configuration.
- Le nombre d'itérations.
- La fonction MLP utilisée.

3 Torch et compagnie

Afin d'implémenter notre Transformers, nous avons utilisé plusieurs modules de torch, notamment :

nn.Linear

Cette fonction permet d'effectuer une transformation linéaire de la forme $y = Ax + b$. Cela permet de transformer les données d'entrées dans une dimension supérieure, ce qui peut s'avérer utile dans des situations plus complexes.

nn.LayerNorm

Cette fonction permet la normalisation des couches en considérant un batch d'entrées. Cette normalisation prend la forme suivante : $\frac{x - E[x]}{\sqrt{Var[x] + eps}} * \gamma + \beta$

nn.Dropout

Considérant une probabilité p , met à zéro certains éléments d'un tenseur. C'est une méthode efficace afin d'éviter le sur-apprentissage des données d'entrées.

nn.Embedding

Création de vecteurs d'embeddings d'un dictionnaire spécifié. Ces vecteurs sont stockés dans une table, ce qui facilite l'usage.

nn.MultiheadAttention

Permet le mécanisme d'attention. Ce procédé est un élément majeur de notre Transformers. Une version faite à la main est disponible dans le code, mais pour des raisons de performances, nous avons choisi d'utiliser la fonction de torch.

4 Explication de code

Ce code n'est pas considéré comme du Python parfait, notamment car il existe des soucis d'abstraction, certaines classes connaissent des éléments qu'elles ne devraient pas, ou encore, il y a clairement un manque d'optimisation de certaines tâches. Cependant, c'est du code fonctionnel, assez simple à utiliser et à comprendre, cette partie n'aura donc pas pour but de décrire chacune des fonctions, mais de décrire le fonctionnement global (sans trop de détails) de notre algorithme d'entraînement et de génération.

Configuration & Dataset

La configuration va nous permettre de stocker les paramètres importants de notre modèle. Si une modification est souhaitée, il faut directement modifier les éléments présents dans la classe associée.

De plus, nous sommes dans un modèle de langage qui s'intéresse aux caractères en particulier, il est donc nécessaire de faire un pré traitement des données d'entrées. Ce traitement est assez simple, on forme un set avec les caractères de nos phrases, ce qui nous donne un vocabulaire précis, on classe cet ensemble dans l'ordre alphabétique et ensuite on associe chaque caractère a son index dans ce tableau.

Entraînement

Les différentes étapes de l'entraînement sont :

- Définition de l'optimisateur d'Adam.
- n itérations :
 - Mise a jour des batchs.
 - Forward
 - Reset des gradients du modèle.
 - Calcul des nouveaux gradients
 - Mise a jour des paramètres (optimisateur d'Adam)

Forward (modèle)

La fonction forward qui intervient dans l'entraînement peut-être définie à l'aide des étapes :

- token embedding
- position embedding
- Dropout
- Forward (bloc)
- layerNorm
- Linear

Forward (bloc)

La fonction forward qui intervient dans précédemment peut-être définie à l'aide des étapes :

- LayerNorm
- SelfCausalAttention
- LayerNorm
- MLP

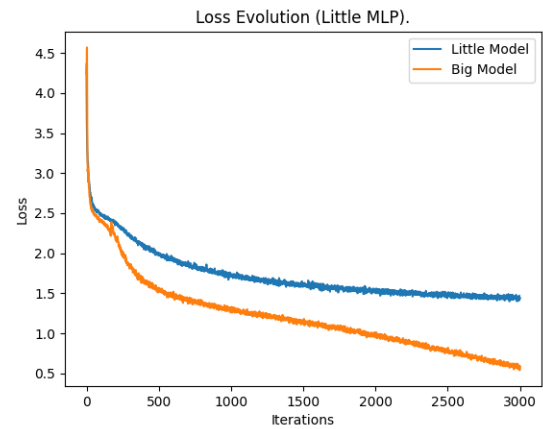
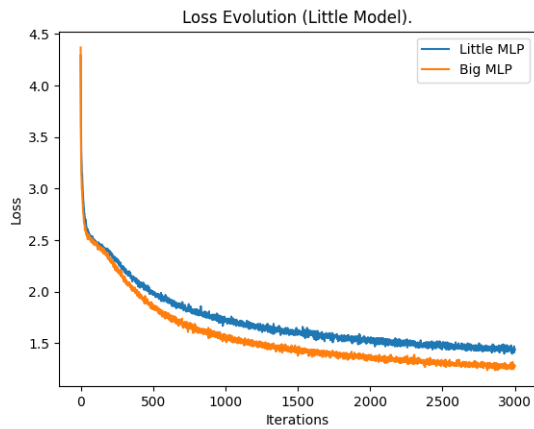
Génération

Plusieurs choix sont possibles, soit nous pouvons sélectionner le caractère qui a la plus grande probabilité, ou alors on peut suivre une distribution de probabilité, ce qui donnera des chances a des caractères avec moins de probabilité.

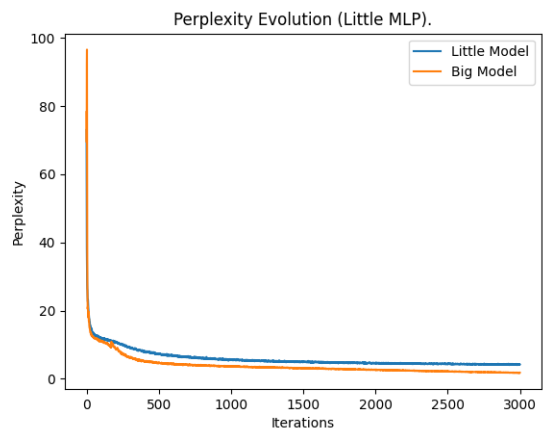
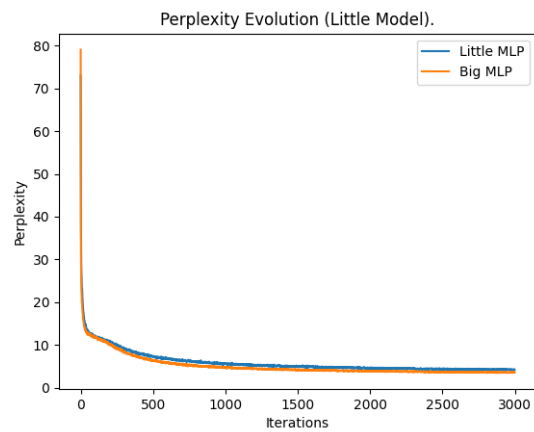
5 Résultats

Pour comparer nos résultats, nous allons utiliser deux modèles avec des paramètres différents, $n_{layer} = 6$, $n_{embd} = 192$, $n_{head} = 6$ et $n_{layer} = 12$, $n_{embd} = 768$, $n_{head} = 8$, ainsi que deux fonctions de MLP.

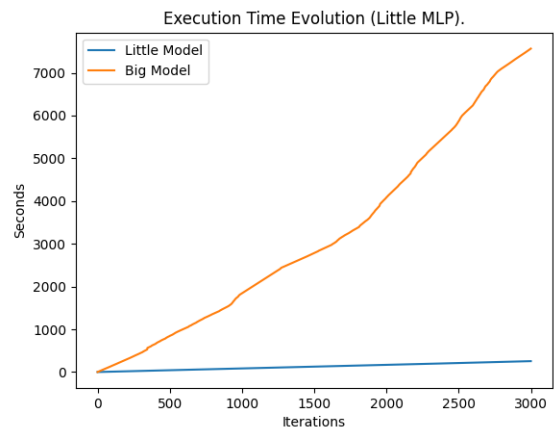
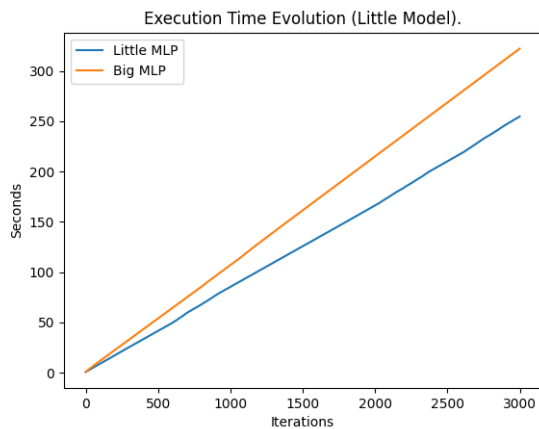
Loss.



Perplexity.



Execution Time.



6 Contenu additionnel possibles

il peut être intéressant d'implémenter un système de learning rate decay, pour ce faire il suffit de modifier le learning rate tous les k itérations.