

Mise en place d'un LLM en local

LP Rob&IA - 3

[R5.07 - Prog]

September 29, 2025

Introduction

Avant-Propos

- Les LLM (Large Language Models) sont clairement à l'origine de la popularité des algorithmes d'IA ces dernières années à l'image de ChatGPT, Mistral, Claude, Copilot, DeepSeek, etc...
- Leur utilisation pose un certain nombre de questions, notamment la sécurité des échanges et leur impact écologique.
- Pour autant, il existe une utilisation croissante qui peut être compatible avec le milieu industriel : installer un modèle plus léger en local.
- Les notions importantes attachées sont :
 - La compréhension des transformers à la base des LLM ;
 - Les techniques les plus répandues permettant de customiser le LLM
 - Le RAG (Retreaval Augmented Generation) : génération augmentée de récupération
 - Le Fine-Tuning : ajustement du modèle
 - Le LoRA (Low Rank Adaptation) : saturation couche de sortie

Point historique sur les Réseaux de Neurones

Le fonctionnement d'un LLM repose bien évidemment sur un réseau de neurones.

Faisons un bref historique :

① RNN (Recurrent Neural Network)

- 1988
- Séquentiel
- Sensible au Vanishing Gradient

② LSTM (Long Short Term Memory)

- 1997
- Séquentiel
- Répond aux problèmes de Vanishing Gradient

③ Transformers (GPT : Generative Pre-trained Transformers)

- 2017 ("Attention is all you need")
- Parallélisation des process (= traitement simultané des mots)
- Capturent l'attention (= pondération de l'importance des mots)

Fonctionnement de l'architecture

Regardons un LLM de loin !

Si on considère un algorithme de traduction d'une phrase comme :

the black cat is in the kitchen.

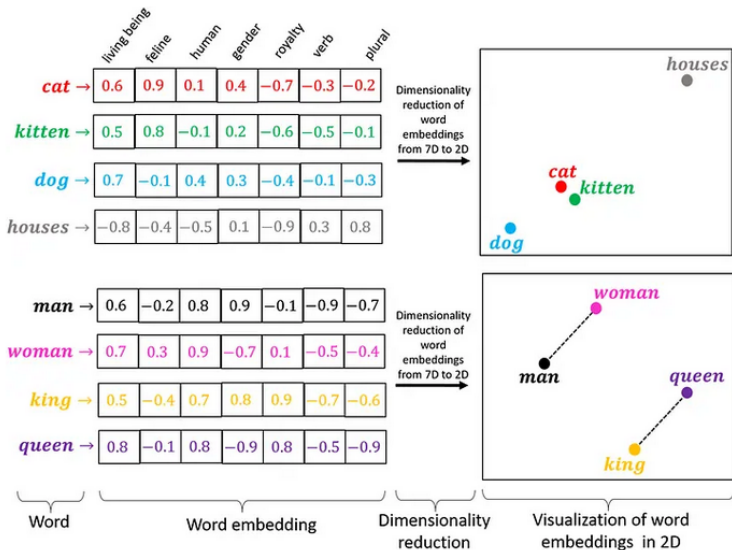
Ce sera l'entrée de l'algorithme bien entendu, puis :

- ➊ **Tokenisation** de l'entrée :

<i>the</i>	<i>black</i>	<i>cat</i>	<i>is</i>	<i>in</i>	<i>the</i>	<i>kitchen</i>
------------	--------------	------------	-----------	-----------	------------	----------------
- ➋ **Word Embedding** : vectorisation des tokens
- ➌ **Encoding** : traitement et compression de la donnée d'entrée
- ➍ **Context Vector** : utilisation de vecteurs de contexte
- ➎ **Decoding** : travail de ré-agrégation et génération de la sortie

Pour se faire une idée plus concrète, un RNN renverrai *le noir chat est dans la cuisine* , un GPT *le chat noir est dans la cuisine*

Schéma explicitant le word embedding



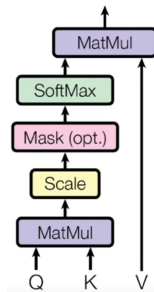
Mécanisme d'attention

Le mécanisme d'attention qui rend les GPT performants est une technique permettant au réseau de **se concentrer sur une partie spécifique** de la séquence d'entrée. Elle définit plusieurs vecteurs :

- *Query* : requêtes Q
- *Keys* : clés K
- *Values* : valeurs V
- *Output* : sortie O

L'attention est définie par la formule :

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

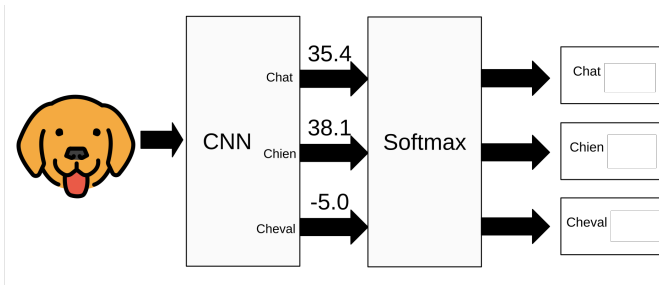


Exercice :

Pour un vecteur $z = (z_1, z_2, \dots, z_n)$, la fonction *softmax* est définie par

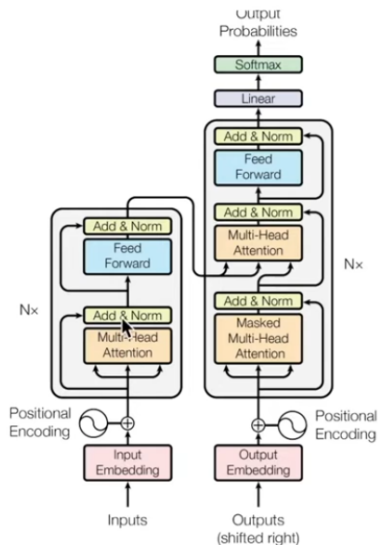
$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^n e^{z_k}}$$

- 1 Calculer $\sigma(z)$ pour $z = (35.4, 38.1, -5.0)$
- 2 Compléter alors le schéma ci-dessous :



- 3 Que peut-on en déduire pour cette fonction ?

Pipeline schématisé



→ Passer au note-book `NB_attention.ipynb` sur Moodle.

Mise en place du RAG

Le RAG sera la solution retenue ici car elle permet de relier le LLM à un fichier d'informations dans lequel le chatbot va passer avant de formuler sa réponse.

C'est la solution la moins gourmande en mémoire en comparaison au Fine-Tuning et au LoRA. Il faut veiller à bien comprendre certaines étapes :

- L'utilisation d'un environnement virtuel anaconda
- La compréhension du gestionnaire de LLM : Ollama
- L'exploitation du script `local_RAG.py`

Configurer Ollama

- Ollama est un outil open-source qui permet d'exécuter différents types de modèles de langages directement sur une machine locale.
- Une fois téléchargé, vous pouvez travailler hors connexion, grâce aux commandes suivantes depuis Anaconda Prompt (base)
- Sous **Ubuntu** (ou MacOS), lors de l'installation de ollama (via le script officiel), le binaire ollama est placé dans `/usr/local/bin/ollama`, qui est toujours dans le `$PATH`, peu importe le répertoire (conda ou ailleurs). Donc le client ollama marche partout.
- Sous **Windows**, Ollama n'est pas intégré dans conda, il est installé comme un programme Windows indépendant. L'environnement conda n'a donc aucune connaissance du binaire `ollama.exe` tant qu'il n'est pas ajouté au `PATH`. Ajouter ce dossier au `PATH` système ou utilisateur :
 - ① Taper `Win + R`, écrire `SystemPropertiesAdvanced`, puis aller dans *Variables d'environnement*.
 - ② Dans la section *Variables utilisateur* → sélectionner *Path* → clique sur *Modifier*.
 - ③ Ajouter la ligne : `C:\Users\<TON_USER>\AppData\Local\Programs\Ollama\ollama-app.exe`
 - ④ Valider, fermer et ouvrir un nouveau Anaconda Prompt (important).

Utiliser Ollama

Voici les commandes minimales pour utiliser Ollama :

- ❶ `Ollama --version` : pour vérifier la bonne installation
- ❷ `Ollama pull modele` : pour installer un des modèles de la liste proposée par le service

Attention ! Regardez la taille du modèle avant de télécharger (= nombre de paramètres 7b → 7 milliards...4,4Go)

Les modèles que j'utilise ici sont : llama3.2, mxbai-embed-large et mistral.

- ❸ `Ollama run modele` : lance un modèle dans votre console
- ❹ `CTRL + D` : Quitter le modèle
- ❺ Chargez un ou plusieurs modèles et testez ses connaissances et réponses !

”Peux-tu me faire une synthèse sur la régulation PID ?”

”Peux-tu m'expliquer la différence entre les protocoles ModBus et Profinet ?”

Anaconda

- Anaconda est l'OS de développement Python que nous avons couramment utilisé les années précédentes.
- Il n'est pas obligatoire mais assure une stabilité dans le développement de projets complexes.
- On peut créer un environnement Anaconda de deux façons :
 - Commande `conda env create -f environment.yml` (pratique sous Linux et Mac)
 - La création manuelle de l'environnement :

❶ `conda create --name envialocal python=3.11`

❷ `conda activate envialocal`

❸ `conda install -c conda-forge pytorch torchvision torchaudio cpuonly -c pytorch -c nvidia`

❹ `pip install openai PyPDF2 ollama pyyaml beautifulsoup4 lxml python-dotenv sentence-transformers`

Le RAG en plus !

- ➊ Récupérer le dossier *local_RAG* depuis Moodle
- ➋ Le placer dans votre `C:\Users\TON_USER`
- ➌ Ouvrir un Anaconda Prompt sous `envialocal`
- ➍ `python upload.py`
- ➎ Vous uploadez un fichier pdf de votre choix (commencez light !)
- ➏ `python localrag.py`
Il crée un fichier `vaul.txt` qui vous donne le contexte retenu par l'algorithme.
- ➐ Vous posez votre question plutôt personnelle au LLM !
- ➑ Le script `localrag_norewrite.py` ne recalcule pas les valeurs du RAG...

A vous de jouer !

- ① Mémo A4 bilan qui répond à la problématique : utilisation d'une IA locale dans mon entreprise (réelle ou imaginaire)
- ② Présentation de vos résultats en moins de 5 minutes
- ③ Pour aller plus loin :
 - <https://www.youtube.com/watch?v=3UQ7GY9hNwk> (finetuning)
 - <https://www.youtube.com/watch?v=XDOSVh9jJiA> (LoRA)