Agent conversationnel et chatbot

Utilisation de Large Language Models pour publier une base de connaissances
Partie 2

Nicolas Debeissat
nicolas.debeissat@mail-formateur.net

Plan du module

Pourquoi faire un agent conversationnel ?

Comment défendre votre projet en entreprise ?

Principes de base des LLMs

Ce qui va faire qu'il va donner des bons résultats, ou non

Entraînement - Fine Tuning

Comment adapter ses réponses

Retrieval Augmented Generation

Répondre à partir de vos données

Intégration dans une application

Créer une API à partir de votre modèle

Evaluation

Entraîner votre IA à faire votre évaluation

Principes de base des LLMs

Ce qui va faire qu’il va donner des bons résultats, ou non

Ce qu’il va avoir du mal à faire

Parler une langue rare

Les corpus d’entraînement sont généralement composés pour moitié d’anglais, le français, seulement 5%. Les LLMs ont des vocabulaires fixes de “sous-mots”, souvent 32 000. Parfois le modèle inclut des mots anglais dans le texte généré

Résumer un livre

L’API Llama2 plante au-dessus de 4000 caractères, il est censé pouvoir accepter 4096 tokens. LongT5 peut prendre jusqu’à 16 384 tokens. Les modèles qui annoncent pouvoir prendre beaucoup de tokens ont des “sliding window attention”

Gérer des doublons même entrée -> sortie différente

Au moment du fine-tuning, il faut éviter de l’entraîner avec la même entrée plusieurs fois, et une sortie différente

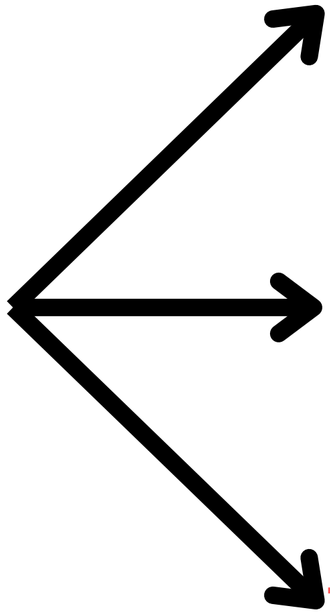
Le traitement automatique du langage (Natural Language Processing)

Entrée	Embedding	Algo	Résultat

“En clair, je suis vert, j'ai perdu mon ver, je vais vers mon verre pour me verser un liquide vert clair”

tensor([[[-0.0311, -0.1710, 0.1370, ..., -0.0768, -0.1337, -0.1754],
[-0.0021, -0.0086, -0.0081, ..., -0.0096, -0.0036, 0.1573],
[0.0563, -0.0936, 0.3475, ..., 0.0210, 0.0888, -0.1234],
...,
[-0.0025, -0.0064, -0.0060, ..., -0.0060, -0.0049, 0.1584],
[-0.1179, -0.0068, 0.3161, ..., 0.0253, -0.0010, -0.0546],
[-0.0004, 0.0009, -0.0029, ..., -0.0019, -0.0008, 0.1741]]],
grad_fn=<MulBackward0>)

Vecteur



Clustering

Classe (classification, choix multiples, recherche)

Distance (cosine, siamese, etc...)

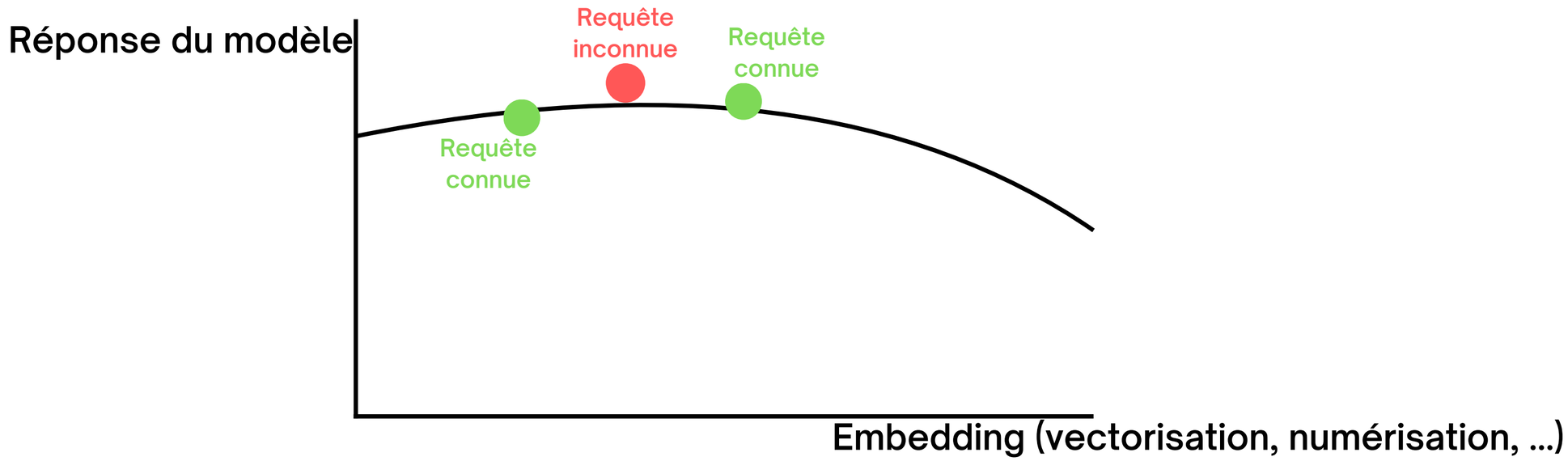
Similarité (recherche sémantique)

Transformers

Transformation (traduction, résumé, réponse)

Vector embedding

La transformation de l’entrée en un vecteur est nécessaire dans tous les domaines de l’IA, et sa qualité garantit que le modèle ne saura pas uniquement répondre aux requêtes qu’il a vu au cours de son entraînement.



Vector embedding

Transformer un texte en valeurs numériques

TF-IDF, BM25

Vectorisation d'une phrase en fonction de la rareté d'un mot dans le corpus

Word2Vec

Projection mot à mot dans un espace de plusieurs centaines de dimensions, des mots proches géométriquement ont le même sens

BERT

La phrase entière devient un vecteur, traduisant le sens de ses mots, avec son contexte, et l'importance de ces mots

Indexing

La phrase est transformée en une liste d'index, qui sont les index de chacun de ses mots ou sous-mots dans un vocabulaire fixe

elasticsearch

vespa

TD2-1 : Embedding TF-IDF

Trouver la réponse à une question parmi des exemples :

- Charger le fichier quizz.csv dans un DataFrame
- Entraîner un TfidfVectorizer sur les réponses
- Ecrire un fichier CSV associant un mot contenu dans les réponses et son score IDF
- Calculer mathématiquement le score IDF d'un mot présent une seule fois dans le corpus
- Faites calculer la vectorisation d'une question par TfidfVectorizer, puis recalculez cette vectorisation à partir d'un CountVectorizer
- A l'aide d'une distance cosinus, trouver la réponse à cette question
- Calculer un taux de réussite de cette méthode

TD2-2 : Embedding Word2Vec

Trouver la réponse à une question parmi des exemples :

- Télécharger un modèle fasttext pour le français avec gensim

```
import requests
resp = requests.get("https://api.pcloud.com/getpublinkdownload?
fileid=59871073335&code=XZJPgJ0ZfW7yy9sXaAHnTqHvzFAo45HBHf4k&linkpassword=&forcedownload=1&auth=seglRXZMwINZJ
iFnKwFkUzXCKDh863vb771YqceV")
json_resp = resp.json()
with open("cc.fr.25.zip", "wb") as f:
 f.write(requests.get(f"https://{json_resp['hosts'][0]}{json_resp['path']}").content)
```

- Calculer un taux de réussite sur l’association question-réponse
- Nettoyer la liste des mots avec la librairie nltk
- Vectoriser les réponses nettoyées avec fasttext
- Calculer un taux de réussite sur l’association question-réponse nettoyée

TD2-3 : Embedding BERT

Trouver la réponse à une question parmi des exemples :

- Utiliser un modèle d'embedding de phrases pour convertir les questions et réponses
- Projeter ces embeddings avec TSNE dans un graphique à 2 dimensions
- Utiliser https://projector.tensorflow.org/ pour visualiser ces embeddings en 3 dimensions. Que constatez-vous ?
- Pourquoi l'analyse PCA ne regroupe pas les questions-réponses ?
- Calculer un taux de réussite sur l'association question-réponse

TD2-4 : Indexing

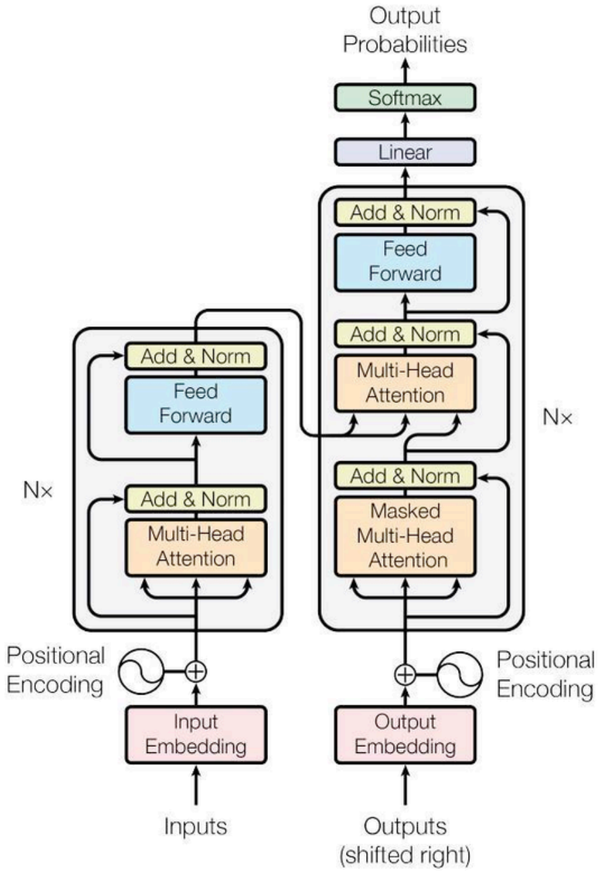
Explorer la tokenization d’un modèle Transformers :

- Utiliser le Tokenizer associé au modèle “google/flan-t5-small” et faites-lui tokenizer la phrase avec les tokens spéciaux

En clair, je suis vert, j'ai perdu mon ver, je vais vers mon verre pour me verser un liquide vert clair

- Afficher la correspondance entre les tokens et leurs index
- Tokenizez "Je, je, je je, jeje JE Je,JE JE,JE"
- La tokenization du mot dépend-elle de :
  - sa casse, si il est au début d’une phrase, si il est après un espace, si il est après une ponctuation
- Extraire last_hidden_state de l’encodage et l’utiliser comme embedding

Les Transformers



Tokenisation en sous-mots d'un vocabulaire généré statistiquement

Mécanisme d'attention, connaissance des entrées brutes à tous les niveaux

Réinjection de la sortie dans l'entrée du modèle

Structure de Mistral

```
model
```

```
MistralForCausalLM(
  (model): MistralModel(
    (embed_tokens): Embedding(32000, 4096)
    (layers): ModuleList(
      (0-31): 32 x MistralDecoderLayer(
        (self_attn): MistralSdpaAttention(
          (q_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear4bit(in_features=4096, out_features=1024, bias=False)
          (v_proj): Linear4bit(in_features=4096, out_features=1024, bias=False)
          (o_proj): Linear4bit(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): MistralRotaryEmbedding()
        )
        (mlp): MistralMLP(
          (gate_proj): Linear4bit(in_features=4096, out_features=14336, bias=False)
          (up_proj): Linear4bit(in_features=4096, out_features=14336, bias=False)
          (down_proj): Linear4bit(in_features=14336, out_features=4096, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): MistralRMSNorm()
        (post_attention_layernorm): MistralRMSNorm()
      )
    )
    (norm): MistralRMSNorm()
  )
  (lm_head): Linear(in_features=4096, out_features=32000, bias=False)
)
```

- Decoder seulement
- MistralSdpaAttention
- MistralRotaryEmbedding
- MistralMLP (Multiple Layer Perceptron)
- SiLU (Sigmoid Linear Unit ou Swish function)
- MistralRMSNorm (Root Mean Square Layer Normalization)

MistralMLP

```
def forward(self, x):
    return self.down_proj(self.act_fn(self.gate_proj(x)) * self.up_proj(x))
```

MistralSdpaAttention

```
attn_output = torch.nn.functional.scaled_dot_product_attention(
    query_states,
    key_states,
    value_states,
    attn_mask=attention_mask,
    dropout_p=self.attention_dropout if self.training else 0.0,
    # The q_len > 1 is necessary to match with AttentionMaskConverter.to_causal_4d that does not create a causal mask in case q_len =
    is_causal=self.is_causal and attention_mask is None and q_len > 1,
)
```

MistralRMSNorm

```
def forward(self, hidden_states):
    input_dtype = hidden_states.dtype
    hidden_states = hidden_states.to(torch.float32)
    variance = hidden_states.pow(2).mean(-1, keepdim=True)
    hidden_states = hidden_states * torch.rsqrt(variance + self.variance_epsilon)
    return self.weight * hidden_states.to(input_dtype)
```

Limitations naturelles de l’IA

https://blog.cubed.run/100-accurate-ai-claimed-by-acurai-openai-and-anthropic-confirm-acurais-discoveries-98fce1ddeb5b
https://youtu.be/K4Wg6QzPfyI

> “LLMs self-organize around Noun Phrases; and that the behavior of LLMs can be controlled through Noun Phrase manipulation.”

Limitations naturelles de l’IA

- Attention aux pronoms, ils peuvent mener à des ambigüités
- L’embedding des termes très précis comme “calcium” et “magnésium” est très proche, parfois plus proches que l’embedding de synonymes comme “voiture” et “automobile”
- L’embedding des nombres est random
- Des noms propres proches sont confondus, à cause de la tokenisation très proche
- L’activation des LLMs dépend fortement des groupes nominaux, plus que les verbes d’actions