2023.11.09 - homework 4

December 1, 2023

0.0.1 2023.11.30 - Introduction to Transformers | Homework 4

In this exercise, you will implement key components of Retrieval-Augmented Generation (RAG): Data Ingestion, Retrieval and Augmentation. RAG significantly enhances the capabilities of language models by allowing them to incorporate external knowledge.

In case you are interested in diving deeper into RAG, checkout the following resources: - Original Paper on RAG: Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks - LamaIndex Tutorial Series: Building RAG from Scratch (Lower-Level)

Base your code on the following skeleton code that we provide:

[4]:
```
import requests
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np
```

0.0.2 Embedding Model

The embedding model transforms textual data into a numerical format (embeddings) that can be easily stored and processed.

In our exercise we will leverage the free inference API from huggingface as well as an open source model. In order to use this API you need to create an account and obtain an access token under https://huggingface.co/settings/tokens.

[5]:
```
token = "{YOURE ACCESS TOKEN}"
```

[6]:
```
API_URL = "https://api-inference.huggingface.co/models/BAAI/bge-small-en-v1.5"
headers = {"Authorization": f"Bearer {token}"}

def query(payload):
        response = requests.post(API_URL, headers=headers, json=payload)
        return response.json()
```

To keep our example simple we will use a small set of predefined, small sentences as our knowledge base. Keep in mind that in real life scenario pre-processing is an important step.

[7]:
```
knowledge_base = [
    "on the 23th december i ate a lovely cheesecake for dinner and a carrot as␣
 ↪a breakfast",
    "the second name of my aunt's second chicken is miranda",
```

```
    "the eiffel tower is located in south tirol."
]
```

[8]:
```
embeddings = query({"inputs": knowledge_base})
embeddings # NOTE: Sometimes the API returns an error, if this is the case,␣
 ↪just run this cell again
```

[8]:
```
{'error': 'Authorization header is correct, but the token seems invalid'}
```

After encoding our knowledge base into embeddings we need to store them together with the original text, since most embedding models don't provide a decoder element.

Task: Create an array of nodes, where each node has the form {"embd": THE EMBEDDING, "text": THE HUMAN READABLE TEXT}. Each element of the knowledge base should have one node. So your db should look something like [{"embd": [0,321, ...], "text": "on the 23th ..."}, ...]

[9]:
```
db = [] # TODO: DB Ingestion
```

To be able to query our db we need to transform a given prompt into the same vector space

[10]:
```
prompt = "What is the second name of my aunt's second chicken?"
```

[11]:
```
prompt_embd = query({"inputs": prompt})
prompt_embd # NOTE: Sometimes the API returns an error, if this is the case,␣
 ↪just run this cell again
```

[11]:
```
{'error': 'Authorization header is correct, but the token seems invalid'}
```

Task: Implement a function named calculate_similarity which takes two arguments, vec1 and vec2. These arguments represent text embeddings that should be semantically compared. The function should return a single similarity value between 0 and 1, where 1 indicates an identical vector and 0 orthogonal vectors.

[12]:
```
def calculate_similarity(vec1, vec2):
    """
    Calculate the cosine similarity between two vectors.

    Args:
    vec1 (list or array): The first vector.
    vec2 (list or array): The second vector.

    Returns:
    float: A similarity score between 0 and 1, where 1 means identical and 0␣
 ↪means orthogonal.
    """
    # TODO: Implement this function
    pass
```

Task: Calculate the cosine similarity between a given prompt embedding and each embedding in your database (db). Identify the database entry (node) that has the highest similarity to the

prompt and retrieve the text associated with this most similar node as your augmentation data. (*hint:* you might want to use np.argmax on an array of similarities)

[13]:
```
# TODO: Implement similarity search
augmentation_data = "" # text of the most similar node
```

[14]:
```
def get_augmented_prompt(prompt, augmentation):
    return f"""
Context information: "{augmentation}".
Given the context information and no prior knowledge, answer the query.
Query: {prompt}
Answer: \
"""
```

[15]:
```
"""
Expected Output:
'\nContext information: "the second name of my aunt's second chicken is␣
 ↪miranda".\nGiven the context information and no prior knowledge, answer the␣
 ↪query.\nQuery: What is the second name of my aunt's second chicken?\nAnswer:␣
 ↪'
"""

augmented_prompt = get_augmented_prompt(prompt, augmentation_data)
augmented_prompt
```

[15]:
```
'\nContext information: "".\nGiven the context information and no prior
knowledge, answer the query.\nQuery: What is the second name of my aunt\'s
second chicken?\nAnswer: '
```