Embedding Adaptors

From DeepLearning.AI "Advanced Retrieval for AI with Chroma" course

The goal to train an adaptor matrix is that it can be used to customize query embeddings to a specific application.

1. Use LLM to generate queries related to the context.
2. Retrieve query embedding and retrieved text embedding.
3. For each query embedding, for each retrieved text embedding, use LLM to judge relevance with a `-1` or `1` label.
4. An adaptor_matrix of size [embedding_dim, embedding_dim] is trained with a goal of `cosine_similarity(matmul(adaptor_matrix, query_embedding), document_embedding)` equals to `adapter_labels`.
5. `adapted_query_embeddings = np.matmul(best_matrix, np.array(query_embeddings).T).T`

In [1]:
```
import chromadb
import numpy as np
import torch
import umap

from chromadb.utils.embedding_functions import SentenceTransformerEmbeddingFunction
from langchain.text_splitter import RecursiveCharacterTextSplitter, SentenceTransformersTokenTextSplitter
from pypdf import PdfReader
from tqdm import tqdm
```

```
def _read_pdf(filename):
    reader = PdfReader(filename)

    pdf_texts = [p.extract_text().strip() for p in reader.pages]

    # Filter the empty strings
    pdf_texts = [text for text in pdf_texts if text]
    return pdf_texts


def _chunk_texts(texts):
    character_splitter = RecursiveCharacterTextSplitter(
        separators=["\n\n", "\n", ". ", " ", ""],
        chunk_size=1000,
        chunk_overlap=0
    )
    character_split_texts = character_splitter.split_text('\n\n'.join(texts))

    token_splitter = SentenceTransformersTokenTextSplitter(chunk_overlap=0, tokens_per_chunk=256)

    token_split_texts = []
    for text in character_split_texts:
        token_split_texts += token_splitter.split_text(text)

    return token_split_texts


def load_chroma(filename, collection_name, embedding_function):
    texts = _read_pdf(filename)
    chunks = _chunk_texts(texts)

    chroma_cliet = chromadb.Client()
    chroma_collection = chroma_cliet.create_collection(name=collection_name, embedding_function=embedding_function)
```

```
    ids = [str(i) for i in range(len(chunks))]

    chroma_collection.add(ids=ids, documents=chunks)

    return chroma_collection


def word_wrap(string, n_chars=72):
    # Wrap a string at the next space after n_chars
    if len(string) < n_chars:
        return string
    else:
        return (
            string[:n_chars].rsplit(' ', 1)[0] + '\n'
            + word_wrap(string[len(string[:n_chars].rsplit(' ', 1)[0])+1:], n_chars)
        )


def project_embeddings(embeddings, umap_transform):
    umap_embeddings = np.empty((len(embeddings),2))
    for i, embedding in enumerate(tqdm(embeddings)):
        umap_embeddings[i] = umap_transform.transform([embedding])
    return umap_embeddings
```

In [2]:
```
embedding_function = SentenceTransformerEmbeddingFunction()

chroma_collection = load_chroma(
    filename='microsoft_annual_report_2022.pdf',
    collection_name='microsoft_annual_report_2022',
    embedding_function=embedding_function
)
chroma_collection.count()
```

349

In [3]:
```
embeddings = chroma_collection.get(include=['embeddings'])['embeddings']
umap_transform = umap.UMAP(random_state=0, transform_seed=0).fit(embeddings)
projected_dataset_embeddings = project_embeddings(embeddings, umap_transform)
```

```
/usr/local/lib/python3.9/site-packages/umap/umap_.py:1943: UserWarning: n_jobs value -1 overridden to 1 by setting random
_state. Use no seed for parallelism.
  warn(f"n_jobs value {self.n_jobs} overridden to 1 by setting random_state. Use no seed for parallelism.")
100%|██████████| 349/349 [06:22<00:00,  1.10s/it]
```

In [4]:
```
import os
import openai
from openai import OpenAI

from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
openai.api_key = os.environ['OPENAI_API_KEY']

openai_client = OpenAI()
```

Creating a dataset

In [5]:
```
def generate_queries(model="gpt-3.5-turbo"):
    messages = [
        {
            "role": "system",
            "content": "You are a helpful expert financial research assistant. You help users analyze financial statements
            "Suggest 10 to 15 short questions that are important to ask when analyzing an annual report. "
            "Do not output any compound questions (questions with multiple sentences or conjunctions)."
            "Output each question on a separate line divided by a newline."
        },
    ]

    response = openai_client.chat.completions.create(
        model=model,
        messages=messages,
    )
    content = response.choices[0].message.content
    content = content.split("\n")
    return content
```

In [6]:
```
generated_queries = generate_queries()
for query in generated_queries:
    print(query)
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling paralleli
sm to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
```

```
1. What is the company's revenue trend over the past few years?
2. How has the company's net income been performing?
3. What are the key drivers behind the changes in the company's profitability?
4. How does the company's current financial position compare to the previous year?
5. What is the company's debt level and how has it changed over time?
6. What is the company's cash flow from operating activities and how stable is it?
7. How are the company's key performance indicators like return on assets and return on equity trending?
8. What are the company's capital expenditures and investments in growth opportunities?
9. How does the company's financial performance compare to its industry peers?
10. How is the company managing its working capital, inventory, and accounts receivable?
11. What are the company's major sources of revenue and how diversified are they?
12. What risks and uncertainties does the company disclose in its annual report?
13. How does the company plan to address any identified risks and challenges?
14. What are the company's long-term strategic goals and how is it positioning itself for future growth?
15. Are there any significant changes in accounting policies or estimates that impact the financial statements?
```

In [7]:
```
results = chroma_collection.query(query_texts=generated_queries, n_results=10, include=['documents', 'embeddings'])
retrieved_documents = results['documents']
```

In [8]:
```
def evaluate_results(query, statement, model="gpt-3.5-turbo"):
    messages = [
    {
        "role": "system",
        "content": "You are a helpful expert financial research assistant. You help users analyze financial statements to
        "For the given query, evaluate whether the following satement is relevant."
        "Output only 'yes' or 'no'."
    },
    {
        "role": "user",
        "content": f"Query: {query}, Statement: {statement}"
    }
    ]

    response = openai_client.chat.completions.create(
```

```
        model=model,
        messages=messages,
        max_tokens=1
    )
    content = response.choices[0].message.content
    if content == "yes":
        return 1
    return -1
```

In [9]:
```
retrieved_embeddings = results['embeddings']
query_embeddings = embedding_function(generated_queries)
```

In [10]:
```
adapter_query_embeddings = []
adapter_doc_embeddings = []
adapter_labels = []
```

In [11]:
```
for q, query in enumerate(tqdm(generated_queries)):
    for d, document in enumerate(retrieved_documents[q]):
        adapter_query_embeddings.append(query_embeddings[q])
        adapter_doc_embeddings.append(retrieved_embeddings[q][d])
        adapter_labels.append(evaluate_results(query, document))
```

```
100%|██████████| 15/15 [00:59<00:00,  3.97s/it]
```

In [12]:
```
len(adapter_labels)
```

150

In [13]:
```
adapter_query_embeddings = torch.Tensor(np.array(adapter_query_embeddings))
adapter_doc_embeddings = torch.Tensor(np.array(adapter_doc_embeddings))
adapter_labels = torch.Tensor(np.expand_dims(np.array(adapter_labels),1))
```

In [14]:
```
dataset = torch.utils.data.TensorDataset(adapter_query_embeddings, adapter_doc_embeddings, adapter_labels)
```

Setting up the model

In [15]:
```
def model(query_embedding, document_embedding, adaptor_matrix):
    updated_query_embedding = torch.matmul(adaptor_matrix, query_embedding)
    return torch.cosine_similarity(updated_query_embedding, document_embedding, dim=0)
```

In [16]:
```
def mse_loss(query_embedding, document_embedding, adaptor_matrix, label):
    return torch.nn.MSELoss()(model(query_embedding, document_embedding, adaptor_matrix), label)
```

In [17]:
```
# Initialize the adaptor matrix
mat_size = len(adapter_query_embeddings[0])
adapter_matrix = torch.randn(mat_size, mat_size, requires_grad=True)
```

In [18]:
```
min_loss = float('inf')
best_matrix = None

for epoch in tqdm(range(100)):
    for query_embedding, document_embedding, label in dataset:
        loss = mse_loss(query_embedding, document_embedding, adapter_matrix, label)

        if loss < min_loss:
            min_loss = loss
            best_matrix = adapter_matrix.clone().detach().numpy()

        loss.backward()
        with torch.no_grad():
            adapter_matrix -= 0.01 * adapter_matrix.grad
            adapter_matrix.grad.zero_()
```

```
  0%|          | 0/100 [00:00<?, ?it/s]/usr/local/lib/python3.9/site-packages/torch/nn/modules/loss.py:535: UserWarning:
Using a target size (torch.Size([1])) that is different to the input size (torch.Size([])). This will likely lead to inco
rrect results due to broadcasting. Please ensure they have the same size.
  return F.mse_loss(input, target, reduction=self.reduction)
100%|██████████| 100/100 [51:41<00:00, 31.01s/it]
```

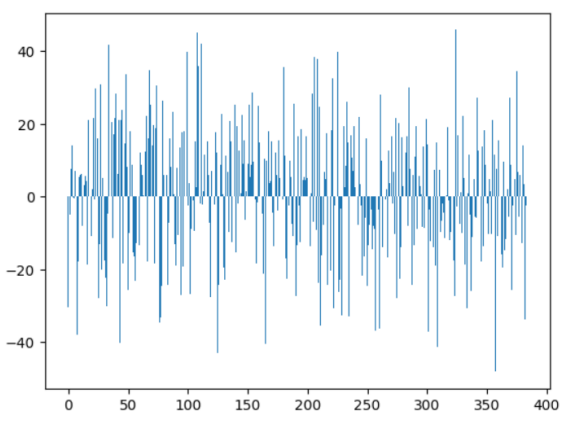
In [19]:
```
print(f"Best loss: {min_loss.detach().numpy()}")
```

```
Best loss: 0.5034573674201965
```

In [20]:
```
test_vector = torch.ones((mat_size,1))
scaled_vector = np.matmul(best_matrix, test_vector).numpy()
```

In [21]:
```
import matplotlib.pyplot as plt
plt.bar(range(len(scaled_vector)), scaled_vector.flatten())
plt.show()
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling paralleli
sm to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling paralleli
sm to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)
```



In [22]:
```
query_embeddings = embedding_function(generated_queries)
adapted_query_embeddings = np.matmul(best_matrix, np.array(query_embeddings).T).T

projected_query_embeddings = project_embeddings(query_embeddings, umap_transform)
projected_adapted_query_embeddings = project_embeddings(adapted_query_embeddings, umap_transform)
```

```
100%|██████████| 15/15 [00:20<00:00,  1.38s/it]
100%|██████████| 15/15 [00:13<00:00,  1.09it/s]
```

In [23]:
```
# Plot the projected query and retrieved documents in the embedding space
plt.figure()
plt.scatter(projected_dataset_embeddings[:, 0], projected_dataset_embeddings[:, 1], s=10, color='gray')
plt.scatter(
    projected_query_embeddings[:, 0],
    projected_query_embeddings[:, 1],
    s=150, marker='X', color='r', label="original"
)
plt.scatter(
    projected_adapted_query_embeddings[:, 0],
    projected_adapted_query_embeddings[:, 1],
    s=150, marker='X', color='green', label="adapted"
)

plt.gca().set_aspect('equal', 'datalim')
plt.title("Adapted Queries")
plt.axis('off')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb54b33d4c0>
```