Deep Learning Project Report: Research Assistant

Authors: Louis Baranzelli, Katarina Lechner, Julien Capitaine

Project: Project 1

The task: NLP domain

Category: Question answering / Conversational

Github link: https://github.com/catherinn/deep-learning-LLM-research-assistant.git

Goal: The goal of this project is to create a prototype of a solution that will inform the users on the state of the art of scientific research on a topic of the user's interest. We achieve this by enabling the user to conduct searches on any desired website through a chatbot-style interface.

User:

- PhD graduate who started to work in a private sphere, but still wants to be aware of the latest advances in the field of his studies or other fields.
- Business-related role/product leadership role who wants to explore a new business opportunity and wants to understand where the scientific research in that area is, what are the latest findings and where it is heading.
- Anyone who wants to understand the state-of-art of scientific research in a particular area of interest, e.g. health-related issues, environmental issues etc.

High-level description of our approach:

We will consider a dataset of scientific papers from Hugging Face and create a chatbot which will allow us to prompt and request detailed info on the state of the art of science. This dataset was collected in 2018 and contains information about scientific publications = from 2 sources: PubMed and Arxiv. We select a pre-trained model (see below) and we use a vector database to store the data and search for the relevant context based on similarity. We enhance it to a chat form, which takes into account both the context and chat_history to answer the questions of the user. To demonstrate our results, we will use the Gradio library as a prototyping tool for a chatbot.

As the next level of our app, we decided to build a tool to scrape a selected website (information source) and have the model answer only based on this. This enables us to further test the relevance of the context that we find with the similarity search and at the same time broaden the capabilities of our tool to wider use cases. To demonstrate our results, we will again use the user interface through the Gradio library.

As the last step, we dockerized our app with all the dependencies.

We collaborated on the project in github, where you can find the latest version of our project.

Model Selection:

We initially chose to work with the **Microsoft Phi 1.5** model. It is a Transformer model containing 1.5 billion parameters. We selected this model for the following reasons:

- Due to capacity reasons (available GPU), we were not able to experiment with larger models.
- At the time when we started to work on our project, Microsoft Phi 1.5 recently came out and showed remarkable results outperforming larger models like the Llama 2-7b on all major benchmarks. Thanks to a different training method Synthetic Data Generation, the model is considered as a starting point for significantly smaller and powerful models.

However, as we assumed, due to the size of the model, it was not suitable for the needs of our project. We decided to try if a stronger model would return better results and so we decided to switch to the **GPT-4 web API**, which provides access to a model with over 175 billion parameters. The results align with our expectations.

Model Selection for the embeddings:

We chose **all-mpnet-base-v2** for our project due to its demonstrated superior performance in benchmark tests against alternative models. Currently, this is a standard embedding model used in LLM projects.

Noteworthy tools used:

- HuggingFace Datasets for loading the initial dataset for vector search
- HuggingFaceEmbeddings, RecursiveCharacterSplitter and FAISS vector store for correctly embedding and tokenizing our dataset and FAISS similarity search for finding the right context to be used in chat
- Gradio for prototyping our work as a chat interface
- Langchain libraries - basics
- Beautiful Soup library for web scraping
- Docker for containerizing our work

We also experimented with the following (see DeepLearningProject - experiments.ipynb file):

- Langchain libraries: tools, chains, PromptTemplates
- BitsAndBytesConfig for optimizing our LLM's memory efficiency and speed
- using the APIs that PubMed and Arxiv provide and using them as retrievers

Action plan:

Step 1: Adapt an existing LLM model to our use case

We see these 3 initial approaches:

- **Alt 1.1 Use own data, vectorize it and search for context** -> using the latest advances in the field, using vector databases to use a standard pre-trained model and have the model search within one's own dataset return answers. This way is less computationally expensive than standard fine-tuning and so, it is often a preferred first step.

Our work: This is the main approach we used. We utilized a custom dataset sourced from HuggingFace, from which we use the 'abstract' column. We do not use the full 'article' texts, as we believe that as the first step, the information contained in the abstracts is enough for the purpose of our project.
We embedded and tokenized the data using HuggingFaceEmbeddings alongside a RecursiveCharacterSplitter. Our goal was to split the abstracts in a way, that separates the abstracts from different articles and enables us to chunk these abstracts into smaller chunks.
We experimented with different chunk sizes, as well as text splitters, and finally decided for chunk sizes of 500 characters. We did not experiment with different libraries for similarity search and clustering of vectors, because based on the experiments we performed, FAISS was able to find the relevant context (chunk_500_abstract.txt, chunk_fullabstract_abstract.txt), which was confirmed by the information available online based on which FAISS is currently state-of-art and in most projects the go-to option for similarity search.
The vector store created in this way we then use to query for the relevant context to the user's query which is fed into the model to answer the user's question.

- Alt 1.2 Fine-tuning -> take a standard model and fine-tune it with one's own dataset

  Our work: We have done some preliminary research and experiments, but we haven't done fully-fledged fine-tuning. We believe that this is the next step to make our initial model Microsoft Phi-1.5 return more relevant answers.

- Alt 1.3 Real-time search (e.g. by calling pubmed/arxiv APIs)

  Our work: To improve on the usability (see drawback 1 in the 'Drawbacks' section) of the solution and keep the solution up-to-date, we experiment with calling the pubmed/arxiv API (with the use of langchain wrappers) themselves. However, at the time of our experiment, the retrievers were not returning relevant information.

Step 2: Clean the codebase and structure our code into logical functions

1. **Grado_interface.py:** Contains the main code that initializes the project. It launches the console in which the user will interact.
2. **Models.py:** Contains two classes—one for managing the text vectorization part. This includes encoding a given text and other useful functions for querying this vector library to find a similar paragraph from the data library. This file also contains functions related to the Language Model (LLM) to create a natural interface with the user during the discussion.
3. **Collect_url.py:** Contains the entire code for recursively scraping a website to generate a library of text for analysis.
4. **logical_state.py:** Manages the variables that control the logical flow of the conversation. It checks if a correct web address is provided, if the vector database is created, etc.
5. **utils.py:** Contains peripheral functions used at different moments.

As an addition, we added a link to a Colab notebook **DeepLearningProject - experiments.ipynb**, which contains code for some of the further experiments we tried.

Step 3: Prompt the model to chat with the user and demonstrate our model with using Gradio

Gradio is a logical choice as it allows, in just a few lines of code, the generation of a conversation interface with the user. The conversation history is preserved, and we have added a reset button to clear all references recorded during the previous conversation (websites and state variables). This interface is accessible locally, with the link: http://127.0.0.1:7860/

Step 4: Upgrade the initial solution with the ability to scrape a selected website for up-to-date information

To get relevant context besides the original dataset from HuggingFace and as our initial experiments with the arxiv/pubmed apis failed (at the time of our experiment), we decided to upgrade our solution and enable it to scrape the web for relevant websites where we are most likely to get information for the question.

Scraping Methodology

We have developed an efficient web scraping methodology using the Beautiful Soup library. When the user provides a URL, we query the text of the URL. If a server responds (error code = 200), we use a function from the Beautiful Soup library to retrieve all the content within the <a> tags that contain an HTML link (href). If this link contains the base URL, we apply the same algorithm to it. We apply this recursion a desired number of times. However, on certain sites, one iteration was sufficient to generate a list of more than 150 websites.

Once this list is generated, we requery each of these sites to retrieve their main content (<p> block) and store it in as many text files as there are two explored sites. The vectorization will take this text folder. However, the vectorization algorithm we use is relatively slow. For 100 web pages of text, it may take up to 10 minutes to vectorize the entire set. To conduct tests, we suggest using only root-end addresses rather than the base root of a website, which potentially contains thousands of links.

Warning: Extracting information from websites without their explicit permission can be against the terms of use of many websites. Before undertaking any such action, make sure to adhere to the policies of the respective website.

Upgraded version with Gradio

To demonstrate our work, let's explore a concrete example:

The user must follow the conversation interface. The user is asked to provide a valid address. If the address is accepted, then the website at a depth of 2 levels is scraped (all web pages belonging to the website are present on the URL provided by the user). The user provides the

URL of a website he wants to explore, for example, the DWP (Department for Work and Pensions) website url for our test:

2023-11-26 12:33:10.102 | INFO | collect_url:get_urls:131 - get 19 results.

19 web pages have been found. All the web pages scraped are saved in a text file (depending on the name of the url filled)

```
https://www.gov.uk/government/organisations/department-for-work-pensions
https://www.gov.uk/government/organisations/department-for-work-pensions/about/publication-scheme
https://www.gov.uk/government/organisations/department-for-work-pensions/about/accessible-documents-policy
https://www.gov.uk/government/organisations/department-for-work-pensions/about/recruitment
https://www.gov.uk/government/organisations/department-for-work-pensions/about/research
https://www.gov.uk/government/organisations/department-for-work-pensions/about/personal-information-charter
https://www.gov.uk/government/organisations/department-for-work-pensions/about/equality-and-diversity
```

Now our users can ask any relevant questions that can be found on the website. This is just a demonstration, our user could try for example the website of National Geographic, or a scientific site where he believes relevant information could be found.

> What tool can I use to analyze statistics related to the DWP?

Context found: '*2023-11-26 12:39:27.624 | INFO | __main__:respond:55 - Paragraph: analysis tools Data visualisations You can explore and customise DWP statistics using the following interactive data visualisations: Benefit Cap Households on Universal Credit Personal Independence Payment (PIP) Stat-Xplore Stat-Xplore is an online tool that allows the creation and download of customised statistical tables, as well as data visualisation through interactive charts It currently holds statistics on: Alternative Claimant Count Attendance Allowance (AA) Benefit Cap Benefit*'

From that, the answer is done:

> The tool that can be used to analyze statistics related to the Department for Work and Pensions (DWP) is Stat-Xplore. It is an online tool that provides interactive data visualizations and allows the customization of statistical tables. Users can explore and download custom statistical tables as well as visualize data using interactive charts. Stat-Xplore currently holds statistics on various aspects such as Alternative Claimant Count, Attendance Allowance (AA), and Benefit Cap.

Under the hood, the question is vectorized, and the closest paragraph is determined based on similarity search as described in Step 1.1. We did some experimentation with the number of 'k' as a number of selected chunks in the similarity search to optimize the results and the chunk size and these are our findings:

- This question is a bit tricky and depends on the website. In this example, we are using the website: https://www.gov.uk/government/organisations/department-for-work-pension.
- With k = 2:

> The data analysis tools provided by DWP include interactive data visualizations and an online tool called Stat-Xplore. These tools allow users to explore and customize DWP statistics, create and download customized statistical tables, and visualize data through interactive charts. Additionally, DWP publishes management information and ad hoc analyses for transparency purposes, which are not classified as official statistics. The tools aim to provide users with in-depth exploration and analysis of DWP data.

- With k=5:

> what kind of data analysis tool do you have ?
>
> The data analysis tools mentioned in the context are Stat-Xplore and interactive data visualisations. Stat-Xplore is an online tool that allows users to create and download customized statistical tables and visualize data through interactive charts. The interactive data visualisations allow users to explore and customize DWP statistics on various topics such as Benefit Cap Households, Universal Credit, and Personal Independence Payment (PIP).

- What we see is that in this case with a higher value of k, perhaps 5, we provide more opportunities for the language model to utilize more details in its answer (at the risk of drowning it in information). So in our example, she selected paragraphs are very similar, but with k=5 we supply more example cases of the data visualisation, which we consider more nuanced and better.
- In our experiment we are using the chunk_size=500, and we do not see a reason to reduce it. However, the risk of drowning the model with lots of information is real and can be an issue for different web sites. This would be our further consideration when performing a larger amount of experiments with different websites,

Step 5: Containerize the application and its dependencies with Docker

To facilitate the occurrence process, we have dockerized the project. To initialize the environment, you will need to build the Docker image. The docker file is included in the repository.

The program only runs on Python 11. First, the requirements are installed (firstly, to avoid having to install all dependencies every time a new save of the Python file is made). Then weload all the Python files. Finally, we expose communication port 7860 to establish a link between the container's localhost and yours.

Warning: If you stop the Docker and run it again to enter a new URL, make sure to close all Gradios windows. Otherwise, it will reload the variables from the previous window (and thus continue searching on the old URL).

Drawbacks and further considerations for the real-life usability of our solution:

- One of the main drawbacks of our initial solution is the dataset we are using (loaded from HuggingFace). Not only is it outdated, but it also doesn't contain any information about the authors and title, academic institution, year written, and relevance (credentials) that would help us give a sense of the relevance and trustworthiness of the scientific advance made. This is also the case with the upgraded version with the scraper. We would need to improve our product in this area, depending on more or less scientific use cases.
- Speed: Both, vectorizing the dataset and scraping the web for a dataset take significant time. As the next step in our project, we would need to focus on making these processes more efficient.
- Directing the LLM to use the right website for the right information. Our tool should automatically know where to search/scrape the relevant information based on the question asked, e.g. if the user asks about health, go to pubmed; if the user asks about astrophysics, go to arxiv etc. This could be easily done by using langchain agents or other similar libraries.
- While experimenting with the more experimental langchain libraries: chains, agents, retrievers, we have discovered that these libraries are not yet mature enough for production use. Working with them is often buggy and our experiments have not yielded any practical breakthroughs that we could use in our project. However, we believe that similar libraries and tools will be further maturing and will make working with LLMs easier and more efficient.
- In-depth testing of the quality of the results - creating comprehensive test sets to make sure the results have a certain level of quality. Additionally, using a tool like Galileo to check for hallucinations could help.
- The ultimate question is whether to use smaller, self-hosted finetuned models or a robust API like ChatGPT. The decision often depends on the business case of the respective product. Currently a lot of use cases, like ours, our much more effectively and quickly facilitated by a powerful model like ChatGPT, but as this is associated with costs, we would opt for this option to prove that our product brings value, and later on we would consider a self-hosted smaller model. Of course, this remains to be seen as the LLM landscape evolves further.