Web Data Processing Systems Project

Aydan Gadirzada (2842755)

Introduction

This report presents how a system processes the raw outputs of large language models, to return the correctness of the raw outputs of Large Language models (LLMs). For this I have trained two models: one for Named Entity Recognition (NER) using a BiLSTM-CRF architecture, and another for extractive question answering (QA) using DistilBERT. First, the system identifies key entities in the text and retrieves relevant Wikipedia urls and scrapes its content using web scraping techniques, and then it splits the Wikipedia content into smaller chunks and retrieves the particular chunk that potentially has the factual answer to the question. Then, the QA model extracts the answer from a raw response of the LLM and then the program checks whether the raw answer (yes/no or an entity) is correct by comparing it with the ground truth.

Methods

Named Entity Recognition

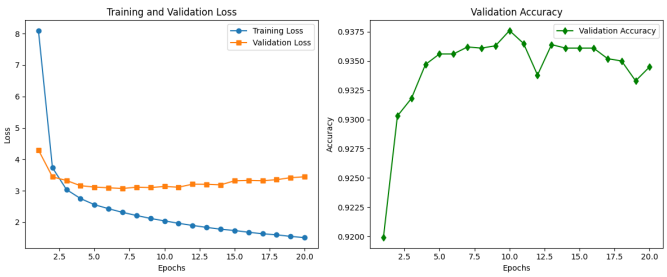
This part of the project trains a deep learning model by using the Few-NERD dataset in order to classify named entities in text into predefined categories such as Person, Location, Organization, etc.

The model is based on a Bidirectional LSTM with a Conditional Random Field (BiLSTM-CRF) architecture. The components of the model include: Embedding Layer: converting words into dense vector representations, BiLSTM Layer: getting contextual dependencies in both directions and CRF Layer: taking the sequence context into account for improved entity recognition [2].

As for the datasets, I initially used the CoNLL-2003 NER dataset for training the BiLSTM-CRF model. However, it contains smaller corpus of sentences (14,987 sentences for training), which can result it to not perform very well on unseen data. For that reason, I proceeded with a dataset that consists of much larger corpus which is DFKI-SLT/few-nerd dataset (131767 sentences for training).

One of the key differences between the two datasets is that CoNLL-2003 includes the Inside–outside–beginning (IOB) annotations, whereas few-nerd does not. Because of this, I adjusted the inference function so that the consecutive entities of the same type in a sentence would be merged.

The results of the trained model on the test set is 93.4%.



NER Training Loss

Wikipedia Page

To retrieve the corresponding Wikipedia page for an entity, web scraping tools were used. First, the system checks if a disambiguation page exists using the URL format: `wikipedia.org/wiki/entity_(disambiguation)`. If this page is found, relevant Wikipedia links containing the entity name are extracted. If no disambiguation page exists, the main page of the entity is checked for disambiguations as well, because in some cases the main page itself is the disambiguation page (e.g., *James Craig*).

After collecting all possible relevant pages, similarity is calculated between the query and Wikipedia summaries (the introductory section of the page). Initially, Mahalanobis distance was tested to compare the sentence embedding of the question to the set of Wikipedia summary embeddings. The motivation behind the Mahalanobis distance was that it can measure how far a point is from a distribution, considering the question as a single point and the summary embeddings as a distribution. However, this approach did not work well because the sentence embeddings do not follow a normal distribution, making Mahalanobis distance unreliable, and because of the high dimensionality issues of the embeddings as well.

Due to these limitations, Jaccard similarity was used instead. This method measures the word overlap between the question and Wikipedia summaries. The similarity score was further improved by prioritizing main Wikipedia pages if the title is the same as the entity name and penalizing the very short summaries as they could be lacking the useful information.

Sentence Embedding

In order to get sentence embeddings, I used pre-trained GloVe vectors to obtain word embeddings of each word in a given sentence. Then, the sentence embedding is computed using the Smooth Inverse Frequency (SIF) method. This method reduced the impact of high-frequency words, leading to a better sentence representations than just averaging the embedding of the words. [1] The algorithm is as follows:

Tokenization - The input sentence is split into individual words, while the punctuation is preserved.

Word Frequency - The word frequency is computed across the dataset.

Weighted Averaging - Each word embedding is weighted using the formula:

$$w_i = \frac{a}{a + p(w)}$$

a is smoothing parameter, and $p(w)$ is the word frequency.

The smoothing parameter controls the influence of frequent and rare words. Frequent words are down-weighted to reduce their dominance in the sentence representation. Rare words receive higher weights, which contributes significantly to the final sentence embedding. A small a value (e.g., 10^{-3}) strongly penalizes frequent words, while a larger value makes the weighting more uniform across all words.

Once the weighted average embedding is computed, Singular Value Decomposition (SVD) is applied to remove the first principal component. As a result, the embeddings focus on content-specific differences rather than generic structures present across all sentences.

Mini RAG System

This part of the project is inspired by Retrieval-Augmented Generation (RAG) [3]. The system utilizes sentence chunking, getting relative text embeddings using the above method, and similarity-based retrieval using cosine similarity to enhance the exact answer extraction. Instead of retrieving just a page summary, the function scrapes the whole wikipedia content ,and then it segments

the content into smaller chunks of 3 sentences with 1 sentence overlapping each time and ranks them by similarity to the query question.

As a result of chunking the text, retrieved content is more specific and relevant to the user's question. It reduces the likelihood of retrieving an entire passage that may contain irrelevant details. The overlapping method helps to maintain contextual coherence between the chunks themselves, preserving information that might potentially be lost.

Additionally, the ranking step based on cosine similarity allows the retrieval of the most contextually similar chunk, unlike the jaccard similarity which relies on keyword matches.

Answer Extraction

This part is divided into two sections. Firstly the program detects whether the given answer is a yes or no. This is done through the cosine similarity between the sentence embedding of a given answer and the embeddings of "yes" or "no" words. If the similarity is higher than a certain threshold of 0.7, that answer is considered as a yes or no. Otherwise, it uses the trained extractive question-answering model for further processing.

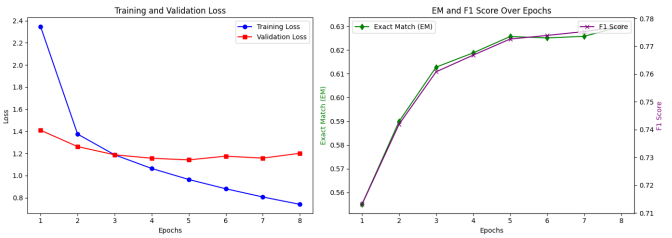
Extractive QA

This part of the project trains a deep learning model using the SQuAD1 dataset to extract answer spans from the given LLM answer to the corresponding user query question. [4]

The model is based on DistilBERT, a lighter and faster version of BERT. It's used for efficiency without any significant loss in accuracy. The architecture consists of: Embedding Layer: The input text is tokenized and converted into dense vector representations using DistilBERT, Answer Span Prediction: Two linear layers predicting the start and end positions of the most relevant answer span within the passage, and Context Classification: A separate classification head assigns probabilities to each token, helping the model determine which parts of the passage are most relevant to the question. The dataset used for training is SQuAD1 (87599 question-context-answer span for training).

For training, the model optimizes a combined loss function: Answer Span Loss: Cross-entropy loss for predicting start and end positions. Context Loss: Binary cross-entropy loss that helps classify which words in the passage are most relevant to the answer.

The final trained model achieves an exact match of 62.9% on the validation set.



Extractive QA Training Metrics

Fact Checking

This part of the project verifies the correctness of the extracted answers by comparing it to the ground truth answer that is retrieved from Wikipedia. Both the given LLM answer and the ground truth chunk from Wikipedia is sent to the extractive QA model. Then, the extracted LLM answer and the ground truth answer is compared using semantic similarity. For standard answers, cosine similarity is computed between the embeddings of the predicted and ground truth answers. If the similarity score exceeds a threshold of 0.7, the answer is considered correct.

For Yes/No questions, a negation-aware similarity function is applied. This function tokenizes the answers and detects the presence of negation words (e.g. as no, not, never, none). If one answer contains a negation while the other does not, the similarity score is flipped in order to show the opposite meaning. Again, the similarity score should be above 0.7 for it to be considered correct.

Scalability

In the mini RAG system, multithreading is used to speed up chunk-level embedding extraction. As questions or raw LLM answers grow longer, the number of entities increases, requiring more extensive Wikipedia content chunking and analysis of them. Parallelism is necessary here in this particular case.

I chose multithreading instead of multiprocessing because embedding extraction relies on a large shared variable (GloVe embedding variable in this case). With multithreading, all threads share the same memory space, reducing overhead and improving performance when accessing these embeddings.

Conclusion

Overall, the proposed pipeline demonstrates how combining entity recognition, chunk-based retrieval, and an extractive QA component can effectively improve the outputs generated by LLMs. The BiLSTM-CRF model accurately identifies key entities, while the RAG-inspired retrieval provides relevant context. Finally, a DistilBERT-based extractive QA module checks each LLM-generated answer against Wikipedia sources to confirm its correctness.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International Conference on Learning Representations (ICLR)*, 2017.

[2] Domino Data Lab - Nikolay Manchev. Building a named entity recognition model using a bilstm-crf network, 2023.

[3] MyScale Team. The ultimate guide to evaluate rag system components: What you need to know, 2024.

[4] Yeon Seonwoo, Ji-Hoon Kim, Jung-Woo Ha, and Alice Oh. Context-aware answer extraction in question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2418–2428. Association for Computational Linguistics, November 2020.