Wikipedia QA

Mohamed Zakaria, Ali Amr, Sarah Hella

May 7, 2023

1 Introduction & Motivation

Wikipedia QA is a Wikipedia Question Answering model that serves to answers users question through the knowledge available on Wikipedia. Wikipedia contains tons and tons of information on almost all topics. However, due to the availability of this large amount of information, it can be hard to find the exact piece of information you are looking for. This is exactly the problem that Wikipedia QA aims to solve. It is designed to help searching for information easier by giving your the exact piece of information that you are looking for.

2 Challenges

With a system of this size, there are a lot of challenges to be faced. Here's a list of examples

1. Gathering information from wikipedia
2. Large number of documents to search through
3. Retrieved document may not contain the answer

Correct and sound architectural decisions will help overcome such obstacles. The plan to overcome these hurdles is as follows:

1. Use of Wikipedia API to gather information
2. Identifying if a dense method of retrieval can be used if the search space is too big with high similarity
3. Use of Top K document retrieval to extract answers from multiple docs and rank them. Highest ranking answer will be returned.

3 Dataset

For this project we will use the SQuAD2.0 dataset.

3.1 What is the SQuAD2.0 dataset?

Squad2.0 (Standford Question Answering Dataset) [RJL18] is a popular dataset used with question answering models. It is an extension of its predecessor, the SQuAD1.1 dataset [RZLL16]. Both the 1.1 and 2.0 versions are used in bench-marking of question answering models. The dataset extends on its predecessor by adding unanswerable questions to the existing answerable questions from SQuAD1.1. This was done to challenge the models performance and test its ability of not only identifying and extracting a correct answer, but also identifying when no correct answer is available and to refrain from answering. This extension serves the purpose of elevating the standards of question answering models and increase model reliability.

Dataset	Size	Articles	Contexts	Answerable,%	Unanswerable,%	Vocab Size	Stop Words
TRAIN	130319	442	19020	86821 , 66.6%	43498 , 33.37%	89982	623
DEV	11873	35	1204	5928 , 49.9%	5945 , 50.07%	18770	461

Table 1: Features of the SQuAD2.0 Train & Dev sets

3.2 Data Analysis

We analyzed the SQuAD dataset in order to identify its features and structure and how the data will be processed in order to be used for training.

3.2.1 Dataset Structure

The dataset is presented in the form of a JSON file. A file is available for each of the DEV and TRAIN datasets, both of which share the same structure with some minor differences.

The dataset heirarchy starts with a set of articles. Each article is split into paragraphs where each paragraph has a context and a questions & answers object. The context represents the paragraph text which may or may not contain the answer. The questions & answers objects contains a set of questions for that particular context. Each question has:

1. Question: the question text
2. Id: question id
3. Answers: list of answers to the question from the context
    - Text: the answer text form
    - Answer Start: the index representing the start of the answer in the context
4. Is Possible: boolean representing whether this is an impossible question or no
5. Plausible Answers: Only present if the question is impossible and contains answers which may be partially correct but are considered false. It takes the same form as the Answers feature.

Its important to note that the Answer feature is one area of difference between the DEV and TRAIN sets. The TRAIN set only contains a single answer for each question in the dataset while the DEV set contains multiple answers and their respective start indices. The multiple answers represent answers selected but different annotators. Most of these multiple answers appear to have very minor differences between them.

3.2.2 Data features

We analyzed each of the available set to identify the following features about each of them:

- Dataset Size
- Number of articles
- Number of unique contexts
- Number of answerable and unanswerable questions and their respective ratios
- Vocab size
- Number of stop words

The results of this analysis can be found in 1.

3.3 Data Processing

In order to perform the previous analysis step as well as prepare the data for training, several preprocessing steps needed to be taken.

3.3.1 Generating Data Examples

The data is originally provided in JSON format. Data must be extracted and repackaged into individual training examples in order to perform both analysis and data preprocessing for training. Each training example consists of the following:

- id : question id
- article title : name of article to which context belongs
- context : paragraph in question
- question
- answer which contains:
  - text : answer text form
  - answer start : answer start index in context

Two method variants exist for generating examples for analysis and training. The key difference is that the training examples variant adds the answer part as is. The analysis examples variant places a null value in the answers part if no answer exists. This is done to simplify parts of analysis process.

3.3.2 Data Preprocessing for Training

This is an additional step only applied to the training examples generated from the previous sub section. Getting SQuAD ready for training requires an input format goes as follows

- Input ids: appended question + context processed by the tokenizer (each word/token represented by its id)
- Attention mask: a mask the represents the words that the model should focus on using the attention mechanism
- Start position: the index of the first token in the answer in the
- End position: the index of the last token in the answer in the tokenized context

To achieve this, the question and context were both passed to the tokenizer. The output of the tokenizer includes the input ids as well as the sequence ids & offset mappings. The sequence ids are used to identify the context start and end. The offset mappings are used to identify the start and end positions of the answer in the tokenized input. Contexts which don't contain the answer partially or fully have a zero for both the start and end positions indicating that the answer is not indicated in the context.

4 System Architecture

In this section, we will discuss the model architecture used for this task. The architecture consists of 4 modules. A query processor, a Wiki Doc Fetcher & Splitter, a retriever and finally a model fine tuned for the extractive question answering task. Figure 1 shows the proposed system architecture.

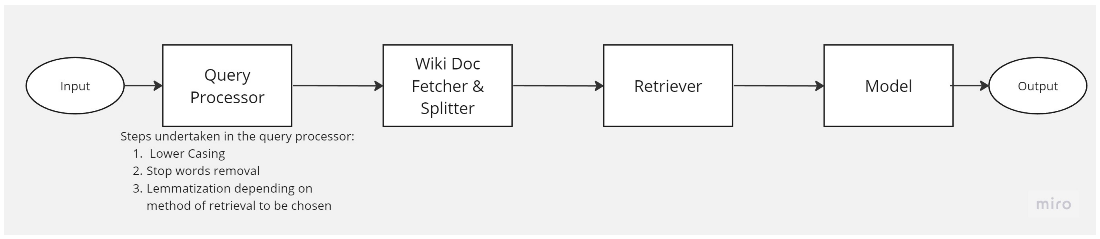
Input
Query Processor
Wiki Doc Fetcher & Splitter
Retriever
Model
Output
Steps undertaken in the query processor:
1. Lower Casing
2. Stop words removal
3. Lemmatization depending on method of retrieval to be chosen
miro

Figure 1: System Architecture

4.1 Query Processor

This is the module responsible for the processing the question or query received from the user. The query processing steps are the following, first the entire query is lower-cased. This will be followed by the removal of stop words such that only relevant words are included in the query. Finally, lemmatization of the query may be performed. This step is yet to decided as it depends on other architectural decisions yet be decided and discussed in the retriever sub section.

Removal of stop words will be done through part of speech (POS) tagging. This means that each word in the query will be tagged with its part of speech tag that indicates whether its a verb, noun, adjective or any other POS. Words that are not tagged as nouns, proper nouns, verbs, adjectives or numbers will be discarded. The result of this is filtered query only containing relevant terms.

4.2 Wiki Doc Fetcher & Splitter

This module performs 2 key important tasks for the system to function. The first task is using the processed query, the Wikipedia API will be used in order to fetch potential pages that may relate to the topic. Once the links are retrieved, each page's contents will be obtained from the API. For every page we retrieve, the module will perform a processing splitting step on the page and divide it into multiple short passages. This is done to optimize the answer extraction process as extracting from longer passages will take much longer. Once each of the pages are processed into passages, the system will move on to the retrieval step.

4.3 Retriever

The retriever is a critical module for this system's success. This is because the retriever is responsible for finding the candidate documents that may contain the answer to the user's question. There are 2 types of retrievers that may be used. The first is sparse retrievers which use metrics such word count and word document frequency to measure the similarity between the document and the query. The second type is a dense retriever or embedding retriever which uses dense vector representations of both the document and the query to measure their similarity in the word vector space.

Which type will be used in this project is yet to be decided based on the amount of data returned by the Wikipedia API and thus the passages created from the Wiki Doc Fetcher & Splitter module. A greater number of passages means a larger search space so using a dense retriever will help capture the relationship between the document and the query better leading to better results.

4.4 Question Answering Model

The question answering model is ultimately where the user's question will be answered. Each of the candidate documents returned from the retriever will be used in this step. The model will extract an answer to the user's question from each of the documents and the answers will be ranked. The highest ranking answer will be returned to user as the final answer.

The model used to accomplish this task will be the RoBERTa Large model. RoBERTa stands for Robustly Optimized BERT pretraining Approach [LOG+19]. RoBERTa extends on BERT [DCLT19] by modifying and optimizing its pretraining protocol to become a high performing model on the SQuAD2.0 dataset. Both BERT and RoBERTa are based on the transformers neural network [VSP+17] which was a ground breaking change in the world of NLP.

References

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[LOG+19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[RJL18] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad, 2018.

[RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text, 2016.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.