AQuA : Advanced Question Answering Model

Sai Sidhardha Grandhi CS19BTECH11050
Mylavarapu Sri Akhil ES19BTECH11014
Nishanth Kannan EE19BTECH11024
Charan Deep Pamuru ES19BTECH11012
Tarun Noonemunthala EE19BTECH11050

Abstract

In the wide area of solving Open-domain question answering mechanisms/methods, there are two major aspects one has to deal with. One being Information Retrieval (IR) and other being Reading Comprehension (RC). Some of the most popular retrieval and matching algorithms are Term Frequency — Inverse Document Frequency (TF-IDF) and Okapi BM25. For Machine Reading Comprehension, there are many prominent architectures such as BiDAF, S-Net, R-Net, match-LSTM, Document Reader and many more.

In this project, our aim is to create a dense embedding model for information retrieval which takes paragraphs from Wikipedia which improves the results without any pre-training on the given data set. Next step is to create a reinforcing document reader using RNN for the purpose of MRC and answer the questions asked. This is our basic idea for the project and we want to try out the scope of combining both the models into one if at all possible, probably as an extension to this.

1. Introduction

AQuA model (Advanced Question-Answering Model) is a model to solve open-domain questions using Wikipedia as an information resource. The motivation for this is to improve the search of information via available data resources and to implement models for IR and RC which can potentially provide better results when used together.

The pipeline to achieve such a goal is done using three important components: The data source, data retriever and a data reader.

The data source we use is Wikipedia which we use to provide answers to the questions. There are multiple ways from which we can download a Wikidump. One of those is by using the freebase data dump Data Dumps — Freebase API (Deprecated) — Google Developers. We then parse the data by splitting it into paragraphs and storing using appropriate data structures such as dictionaries.

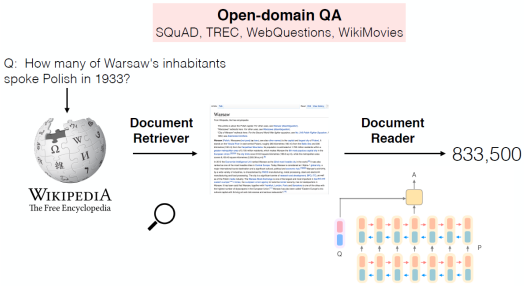


Figure 1. An Overview of the Question Answering System DrQA which our System is based on.

The data retriever is implemented using the DPR (Dense Passage Retrieval) model, which is a method that uses ML. We chose this over the conventional TF-IDF retriever method, which does not use ML, as this is a relatively new technology and it outperforms the former method. The job of the data retriever is to output relevant wiki articles based on the Question asked.

The data reader model is implemented using Bidirectional LSTMs and takes an input of the paragraphs and the question asked. It should output the relevant passage (a couple sentences) from the given paragraphs (which is the output of the data retriever). Hence we get our answer.

2. Literature Review

2.1. DPR for Open Domain Document Retrieval

https://arxiv.org/pdf/2004.04906.pdf

We use two encoders: E_P and E_Q for encoding the passages and questions respectively. And train the model such that the similarity value $sim(E_Q(q), E_P(p))$ is increased. Now we have embeddings that can relate questions and passages. Next, index all the passages in the WikiDump with these embeddings. Given a question q, the relevant passages are those which have highest similarity value $sim(E_Q(q), \text{Index}(P))$, and we have a model that can retrieve relevant

documents from the WikiDump.

2.2. Question Answering given Context

https://arxiv.org/pdf/1704.00051.pdf

- Firstly, we encode the paragraph into a sequence of feature vectors. Then the words are converted into word embeddings by feeding them through an RNN (LSTM). Some of the prioritized features for the feature vectors are : Exact word matches and Soft alignments. Manual features can also be added to the embeddings like: POS (part of speech) tags, NER (named entity recognition) tags and TF (Term frequency).
- Question encoding is done by converting the question words into word embeddings using another similar RNN (LSTM). Note that all the words of the questions are then encoded into a single vector. We generate question vector by using:

$$b_j = \frac{\exp(\mathbf{w} \cdot \mathbf{q_j})}{\Sigma_{j'} \exp(\mathbf{w} \cdot \mathbf{q_{j'}})} \tag{1}$$

$$\mathbf{q} = \mathbf{\Sigma_j b_j q_j} \tag{2}$$

- Prediction of the answer: For each word,we find the probability of it being the starting word and the probability of it being the ending word. Then maximize the probability $P_{start}(i) * P_{end}(i')$ where $i \leq i' \leq i + 15$.

  Here, the answers or the answer snippets are being limitted to 15 words

$$\begin{aligned} P_{start}(i) &\propto \exp(\mathbf{p_i W_s q}) \\ P_{end}(i') &\propto \exp(\mathbf{p_i W_e q}) \end{aligned} \tag{3}$$

3. Implementation

- We have tried to implement the paper Reading Wikipedia to answer open-domain questions. The system we made needs context and the question as inputs to the model and it gives out the starting index of the answer from the given context paragraph.
- We implemented this using the SQuAD Dataset.

3.1. Architecture

- The model takes 2 strings, i.e. context and question and input. The context is retrieved from the DPR.
- To simplify handling of strings, any punctuation marks are removed from the strings.
- Then the string is tokenized using Tensorflow's preprocessor.
- After this, the tokens are converted to embeddings. For this, we've tried using BERT embeddings and also using a default embeddings layer. Owing to the complexity of BERT model, the training time is increasing by a lot.
- Then the question embeddings are condensed into a single question vector, using weights, which are determined by the dot product of a learnable vector parameter and each word's embeddings.
- The output of the model is probability of each word being the start of the answer and the probability of each word being the end of the answer.
- To get the probability, we use softmax of bilinear similarity of each word in the context with the question vector.
- The word index with highest probability is considered as the model output.

3.2. Code - DPR

- We used quite a few libraries as per requirement. We have used Transformers and TensorFlow libraries to use some of the pre-existing layers, encoders and preprocessors among other things. We used the Faiss library for efficient similarity search and clustering of dense vectors for the appropriate answers.
- Firstly, we used TPU from TensorFlow to accelerate our TensorFlow operations. Then we took the input files in .json format from here and have done the formatting accordingly into a dictionary. For the model, we used one of the smaller BERT versions, google/bert_uncased_L4_H512_A8 considering the restricted computational resources.
- After encoding the data into the model with preset configuration, it is again split into query and passage to be encoded separately. Then the QueryEncoder and PassageEncoder are passed into our BiEncoderModel which maps both of the encoded data accordingly. Which is then put for training. You can check the progress of the epochs below:

```
Epoch 1/30
P pooled_output : (64, 512)
Q pooled_output : (16, 512)
Global Passage Shape : (512, 512)
Global Query Shape : (128, 512)
P pooled_output : (64, 512)
Q pooled_output : (16, 512)
Global Passage Shape : (512, 512)
Global Query Shape : (128, 512)
156/156 [==============================] - 80s 122ms/step - loss: 4.1588
Epoch 2/30
156/156 [==============================] - 19s 123ms/step - loss: 1.2614
Epoch 3/30
156/156 [==============================] - 19s 123ms/step - loss: 0.9183
Epoch 4/30
156/156 [==============================] - 19s 123ms/step - loss: 0.7557
Epoch 5/30
156/156 [==============================] - 19s 122ms/step - loss: 0.6178
Epoch 6/30
156/156 [==============================] - 19s 122ms/step - loss: 0.5247
Epoch 7/30
156/156 [==============================] - 19s 122ms/step - loss: 0.4558
Epoch 8/30
156/156 [==============================] - 19s 122ms/step - loss: 0.4053
Epoch 9/30
156/156 [==============================] - 19s 123ms/step - loss: 0.3698
Epoch 10/30
156/156 [==============================] - 19s 123ms/step - loss: 0.3454
Epoch 11/30
156/156 [==============================] - 19s 123ms/step - loss: 0.3336
Epoch 12/30
156/156 [==============================] - 19s 123ms/step - loss: 0.3161
Epoch 13/30
156/156 [==============================] - 19s 123ms/step - loss: 0.3062
Epoch 14/30
156/156 [==============================] - 19s 123ms/step - loss: 0.2973
Epoch 15/30
156/156 [==============================] - 19s 122ms/step - loss: 0.2907
Epoch 16/30
156/156 [==============================] - 19s 123ms/step - loss: 0.2817
Epoch 17/30
156/156 [==============================] - 19s 123ms/step - loss: 0.2773
Epoch 18/30
```

3.3. Code - Q&A

- We used libraries from TensorFlow like tensorflow_hub and tensorflow_datasets to utilize readily available and popular models/datasets. Tensorflow_text is a powerful tool used for handling raw text strings/documents which we did preprocessing with.
- The dataset that we used is SQuAD (Stanford Question Answering Dataset). It is a collection of question-answer pairs derived from Wikipedia articles (via crowd sourcing) in which the answers can be any sequence of tokens in the given text. The latest version has 100,000 questions and answers combined with another 50,000 unanswerable questions. To remove the excess number of tokens and to avoid indexing errors of the final output, we removed punctuation from the text.
- After this, we loaded in the pretrained BERT (Bidirectional Encoder Representations from Transformers) model. This is a very recent model for NLP in which Transformer is bidirectionally trained (both left-right and right-left; reads it all at once) for language modeling. We used a smaller BERT model here : bert_en_uncased_L2_H128_A2. We have tried using pretrained embeddings and simple word embeddings like : bert_en_uncased_L2_H128_A2/1 bert_en_uncased_preprocess
- We then defined the question encoder class which converts the question into a single vector of numbers corresponding to the words. The predictor class, using bilinear symmetry, gives us the probability of a word being the first word of the correct answer.

3.4. Evaluation and Observations

3.4.1 DPR

- The input data for training is loaded into eval_dicts and then passed through a similar preprocessing as train data before predicting the results. Then the extracted embeddings from the model are tested by comparing them with the actual embeddings.
- We checked the top k results of output context passages of the model with the actual relevant answer/passage and the results are here as follows:

```
who sings does he love me with reba
("You Don't Have to Say You Love Me (album)", 'You Don\'t Have to Say Yo
--------
where do the great lakes meet the ocean
('Great Lakes', 'Great Lakes The Great Lakes (), also called the Laurent
--------
when does the new my hero academia movie come out
('My Hero Academia', 'and cast from the anime series returning to repris
--------
who was the creator of victoria 's secret
("Victoria's Secret", 'Corporation with a mandate to establish a group o
--------
when did wesley leave last of the summer wine
('First of the Summer Wine', 'original series making an appearance in th
--------
who introduced the system of civil services in india
('Civil Services of India', "administer them. The civil service system i
--------
southern soul was considered the sound of what independent record label
('Soul music', 'many hit songs for Chicago artists and produced hits on
--------
who is the bad guy in lord of the rings
('Sean Astin', 'the 2016 presidential election, Astin campaigned for Hil
--------
where is the most distortion on a robinson projection
('Perspective projection distortion', 'Perspective projection distortion
--------
what is the name of wonder womans mother
('Who Is Wonder Woman?', 'and thinks of it as some sort of gift. Now tha
--------
```

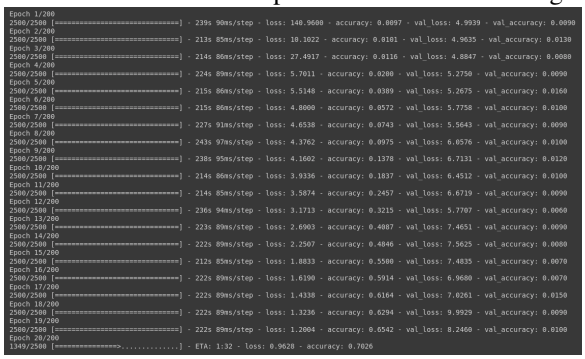
- From the above results we can say that the given output is matching with the appropriate till an extent and accuracy is increasing as the search results are extended as predicted.
- Even with a question which is not a part of train/test datasets like "what is the population of India ?", We get appropriate search results like:

  ('Demographics of India', "Demographics of India India is the second most populated country in the world with nearly a fifth of the world's population. According to , the population stood at . During 1975–2010 the population doubled to 1.2 billion. The Indian population reached the billion mark in 1998. India is projected to be the world's most populous country by 2024, surpassing the population of China. It is expected to become the first political entity in history to be home to more than 1.5 billion people by 2030, and its population is set to reach 1.7 billion by 2050. Its population growth rate")
- The code for the dense passage retriever part can be found here: https://colab.

research . google . com / drive / 1uyQRGUSwVFIjyGdCpw5KYtDiqldWYoK6 ? usp=sharing

3.4.2 Q&A

- Training is done with the help of Adam optimizer and the loss is calculated as cross-entropy loss. The training is capped at 95% to prevent over-fitting. However, no change in validation accuracy is observed. You can see the change of accuracies with each epoch in the below image.



- This is a sign of over-fitting which we currently haven't figured out how to overcome.

- The code for the question and answer from context part can be found here: https://colab.research.google . com / drive / 1hOuHMD0y2i4 _upMn0OotrR2tqARR9N1X?usp=sharing

3.4.3 Overall

Our aim is to check the combined result of both the models together once we try to fix the layers in the QnA model part and see the output which we are not able to do yet.

References

[1] Faheem Abbas, Muhammad Kamran Malik, Muhammad Umair Rashid, and Rizwan Zafar. Wikiqa — a question answering system on wikipedia using freebase, dbpedia and infobox. In *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pages 185–193, 2016.

[2] James Briggs. How to create an answer from a question with dpr, Sep 2021.

[3] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions, Apr 2017.

[4] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. Dense passage retrieval for open-domain question answering, 2020.

[5] Kyosuke Nishida, Itsumi Saito, Atsushi Otsuka, Hisako Asano, and Junji Tomita. Retrieve-and-read. *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, Oct 2018.