Open Domain Question Answering with BERT

Yuqian Lei
Knowledge Processing in
Intelligent Systems: Practical Seminar
Knowledge Technology, WTM, Department
of Informatics, University of Hamburg
yuqian.lei@informatik.uni-hamburg.de

Yunlong Wang
Knowledge Processing in
Intelligent Systems: Practical Seminar
Knowledge Technology, WTM, Department
of Informatics, University of Hamburg
yunlong.wang@informatik.uni-hamburg.de

***Abstract*—Open domain Question Answering (QA) is a research field within information retrieval and natural language processing, which concentrates on how to automatically answer factoid questions posed by a human in a natural language. In this paper, we focus on building an open domain QA system that answers questions based on the given texts in German. We choose a pretrained German BERT model as our base model and fine-tune it on the dataset GermanQuAD, a German Question Answering Dataset.**

I. Introduction

There are two main types of Question Answering systems, one is open domain Question Answering and the other one is closed domain Question Answering. Open domain QA systems can be used across different domains and answer questions posed in a natural language. [1] The goal of an open domain QA system is to provide an answer to a question posed by a user in the form of short texts. The user does not need domain-specific knowledge to prepare the question. A good example can be a system that answers a question based on the information found on the World Wide Web, such as Wikipedia [2]. A closed domain Question Answering system is restricted to a specific domain, and such a system usually makes use of terminology and vocabulary that are specific within a certain domain, such as medicine, biology and physics [1].

A typical Question Answering system is defined by two parts, according to [2]. Firstly, it uses information retrieval to find where the related information is. Then, it uses machine reading to extract the exact position(s) of an answer, which are usually located in several sentences, i.e. the model is trained to give the span of an answer with the information provided or retrieved.

In this paper, we will focus on building a machine reading system that answers questions addressed in German, based on German paragraphs that are provided together with the questions. More specifically, we will train a model with QA tasks from a German dataset and evaluate its performance.

A. Research Question

Transfer Learning aims to apply the knowledge obtained by solving a task in a domain to a different but related task or domain. In this work, similar to [3][4], we will focus on investigating the benefits of transfer learning for Question Answering. More specifically, we will use transfer learning by applying the knowledge that the German BERT model obtained from the pre-training stage to our fine-tuning stage. One important thing to note is that the tasks in the two stages are different, but are also related because they both deal with tasks in the same language.

II. Related work

A. Early QA-System

According to [5], the early QA system was used to deal with statements(and/or questions), which only works in closed domain QA. The earliest QA machine can be traced back to [6] in 1961, which answers questions about baseball. A representative QA system was introduced in Text Retrieval Conferences (TRECs) in the 1990s [7]. It defines a typical pipeline of the QA system as shown in Figure 1, which is mainly consisted of three parts. Firstly, an input question is

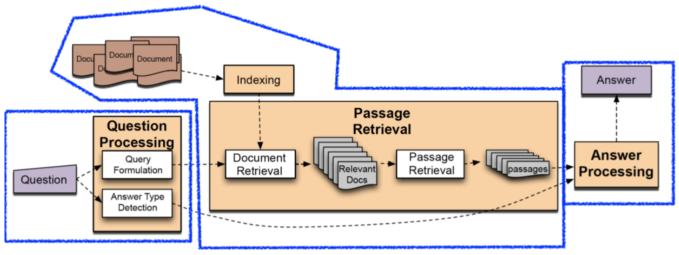

Figure 1. **Typical pipeline of TREC-QA systems.** This figure is taken from [7].

preprocessed by forming a query and detecting the answer type. Then, the formed query and the indexed documents are used to retrieve all the relevant documents. Finally, the detected answer type and the retrieved documents are used to find the corresponding answer.

B. Recent QA-System

After deep learning became popular, the first representative open domain QA system (DrQA) that applies neuron networks was introduced in 2017 [2]. Its general workflow is

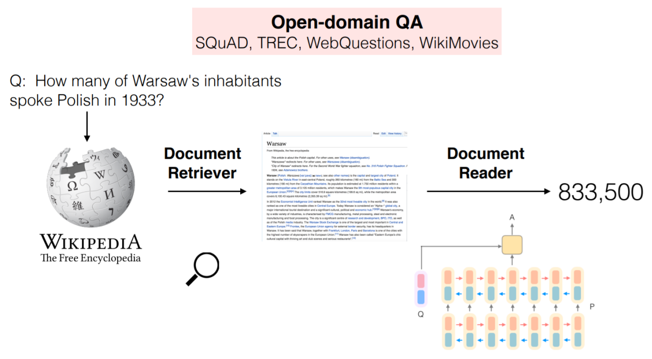
Open-domain QA
SQuAD, TREC, WebQuestions, WikiMovies
Q: How many of Warsaw's inhabitants spoke Polish in 1933?
Wikipedia
The Free Encyclopedia
Document Retriever
Document Reader
833,500

Figure 2. **The pipeline of DrQA system.** This figure is taken from [2]

shown in Figure 2. It comprises two stages, document retrieving using an untrainable TD-IDF algorithm, and document reading using a multi-layer recurrent neural network. After that, the main works focus on improving the performance of these two stages, from which BERTserini was created [8]. BERTserini uses BERT [4] as machine reader and open source toolkit Anserini [9] as information retriever to build an end-to-end QA system.

III. Model: BERT

In recent years, a new language representation model called BERT [4] is introduced, which stands for Bidirectional Encoder Representations from Transformers. BERT achieved state-of-the-art performance on various natural language processing tasks such as Question Answering. Because of its outstanding performance, we decided to choose BERT as the base model for our Question Answering system. In this paper, we will use the pretrained German BERT model [10] from Huggingface for building our QA system. Two extra dense layers are added on the top of the network to train the model, so that it adapts to Question Answering. The two layers help predict the start and end indices of the answer separately.

A. *Model description*

German BERT has the same model architecture as BERT, except that it is pretrained with a different corpus compared to BERT. Therefore, we will explain the general structure and the main components of BERT, and all the mentions of BERT can be referred as German BERT in this subsection. As shown in Figure 3, The base version of BERT is consisted of twelve transformer encoders, which take embeddings as input and yield hidden vectors as output. The source of the embeddings contains a question sentence and a paragraph or a passage related to that question. The final output of hidden vectors are used for various downstream tasks such as classification. Since the task in our work is Question Answering, only the hidden vectors corresponding to the embeddings of the paragraph are needed. [4]

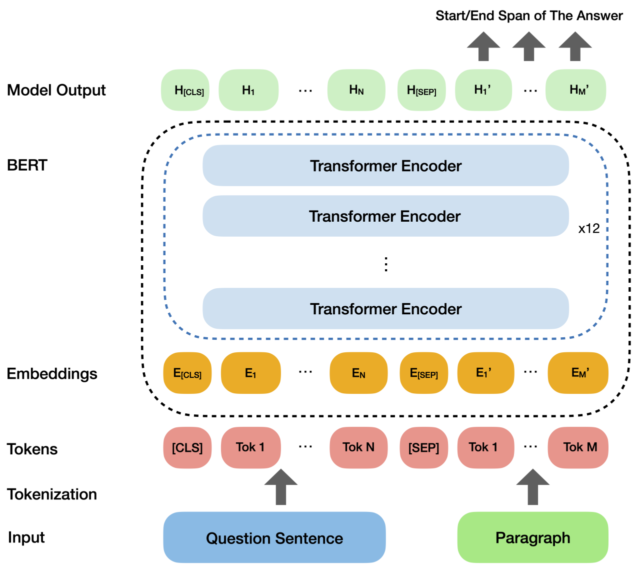

Figure 3. General Structure for Question Answering using BERT

1) Tokenization: Tokenization is an important step before constructing the input embeddings for BERT. As shown in Figure 4, BERT uses a tokenizer to convert a sentence into a sequence of tokens and break down certain words into subwords. For example, the word *'embeddings'* is split into four tokens, namely *'em'*, *'##bed'*, *'##ding'*, and *'s'*. This is a useful method that can help the model understand a new word by breaking it down into the words that it has already learned. [11] Two special tokens are added to the beginning and the end of the converted tokens. The front token is [CLS], which can be used for classification task. The other token is [SEP], which is used to separate different sentences in one paragraph. [4]

2) Embeddings Layer: After the tokenization process, three different embeddings are created for each token, and each embedding is a vector of dimension 768 for the base BERT model. Token embeddings are created using the WordPiece model [12] with a token embedding table that has embeddings corresponding to 30,000 unique tokens. Segment embeddings are used to identify which sentence a token belongs to. Position embeddings are used to locate a token in a sequence. The input embeddings are the sum of all token embeddings, segment embeddings and position embeddings combined. [4]

3) Transformer Encoder: The architecture of a transformer encoder is consisted of two main components. The most important component is the self-attention layer, which applies an attention mechanism that is used for natural language understanding. The other main component is the feed forward network, which applies the relu activation function and linear transformation to the output of the self-attention layer. In simple terms, self-attention is used to quantify how

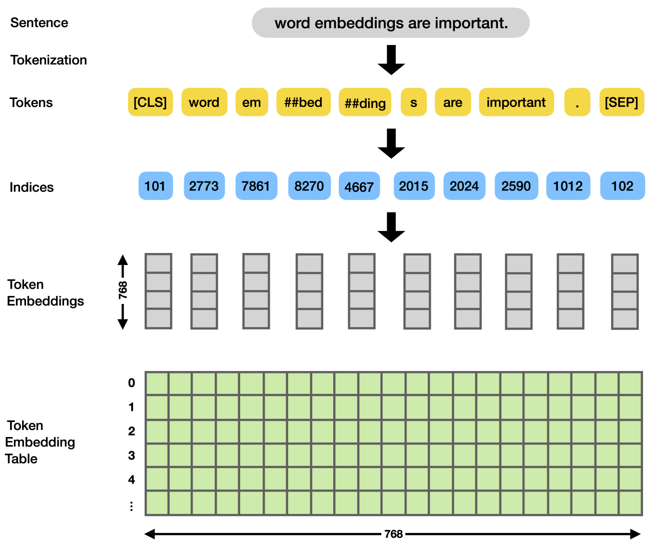

Figure 4. BERT Tokenization and Token Embeddings

much a token is related to all the tokens in a sequence, including itself. [13] Figure 5 shows the general architecture for a transformer encoder.

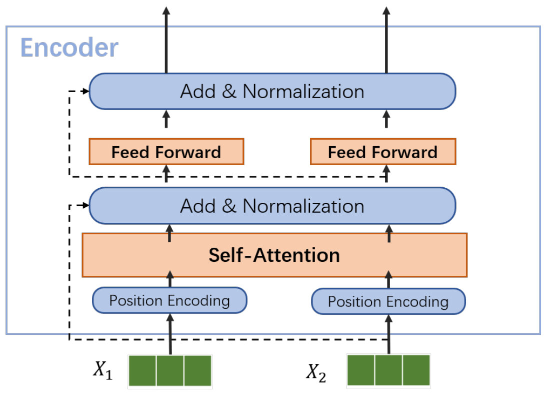

Figure 5. **General architecture for a transformer encoder in base DistilBERT.** The primary parts of an encoder are the self-attention layer and the feed forward network.

4) Pre-training German BERT: Same as BERT, the base model of German BERT is pretrained using the strategies Masked LM and Next Sentence Prediction (NSP). These two strategies are applied in order to overcome the challenge faced by many models that use a directional approach to predict a word in a sequence, and to train a bidirectional model. [4] In a directional approach, the model predicts a certain word based either on the context before the word or after the word. In the first task Masked LM, 15% of the tokens in a sequence are randomly selected and replaced with a [MASK] token 80% of the time, with a random token 10% of the time, and are not changed 10% of the time. In the second task NSP, two sentences A and B are grouped to create an example. The sentence B is actually the next sentence of the sentence A 50% of the time, and is labelled as *'IsNext'*. The other 50% of the time, sentence B is a random sentence from the corpus and is labelled as *'NotNext'*.

IV. Approach

A. Dataset: GermanQuAD

We will use the human-labeled German QA dataset GermanQuAD, [3] which is inspired by the English QA dataset SQuAD [14]. GermanQuAD is consisted of 13,722 questions, 18,054 answers and 3,014 passages. There are 11,518 examples in the training set and 2,204 examples in the test set. We first randomly split the training set into two datasets with the ratio 8:2, namely a train dataset and a validation dataset. The test dataset is used for determining the final performance of the trained model.

B. Experimental setup

1) Detailed workflow: We fine-tune the pretrained German BERT model on the task Question Answering with the dataset GermanQuAD. The detailed workflow is as following:

i) We preprocess our dataset by splitting it into three datasets, one for training, one for validation and one for testing. All the examples in the datasets are then tokenized and fed into the model.
ii) To fine-tune our model with the GermanQuAD dataset, we first put the examples in forms of batches into the model, then calculate the loss values(cross-entropy) and the related metrics(f1-score and exact match).
iii) Next, we do back-propagation based on the loss values in order to optimize the parameters in the neuron network BERT.
iv) Go to step ii, until the performance of the model could not be further improved.

2) Hyperparameters: We choose a small learning rate 1e-5 to prevent the model from forgetting too much of the previous knowledge. We choose a batch size of 6 to satisfy our computing ability. We pick the Adam optimizer [15] because it has an outstanding performance and is most commonly used in deep learning. With the Adam optimizer, we do not have to worry much about the hyperparameter-design. In the other words, we only need to pick a learning rate at the beginning, and then let the Adam optimizer calculate the adaptive learning rate by itself. We choose multi-class cross-entropy as our loss function.

C. Evaluation

It is essential to choose some metrics to evaluate the performance of a model after training. Here we use two metrics, f1-score and exact match(EM). The final value of a metric is calculated by averaging the metric values of all examples.

1) F1 score: The calculation of F1 score is based on the predicted text Pr and ground-truth text Gt, which are found using the output of the model that contains a text span with a start index and an end index. F1 score is calculated using precision and recall. Precision is measured by dividing the length of the text that Pr and Gt overlap by the length of Pr. Recall is measured similarly, the details are shown as following:

$$F1 = \frac{2 \times precision \times Recall}{precision + Recall} \tag{1}$$

$$Precision = \frac{Pr \cap Gt}{Pr} \tag{2}$$

$$Recall = \frac{Pr \cap Gt}{Gt} \tag{3}$$

2) Exact match: Exact match(EM) for each example is calculated using the following method:

$$EM = \begin{cases} 1 \text{ if } Pr = Gt \\ 0 \text{ if } Pr! = Gt \end{cases} \tag{4}$$

V. Result

The F1 score obtained using the validation dataset was used to pick the best model, which was found after the training in epoch 42. The F1 score of the best model was 0.4776 after evaluating with the validation dataset, 0.4270 with the test dataset and 0.7968 with the training dataset. Moreover, the EM score of the best model was 0.2747 for validation, 0.2243 for testing and 0.7968 for training. More details of these metrics and loss value of each epoch can be referred to Figure 6.

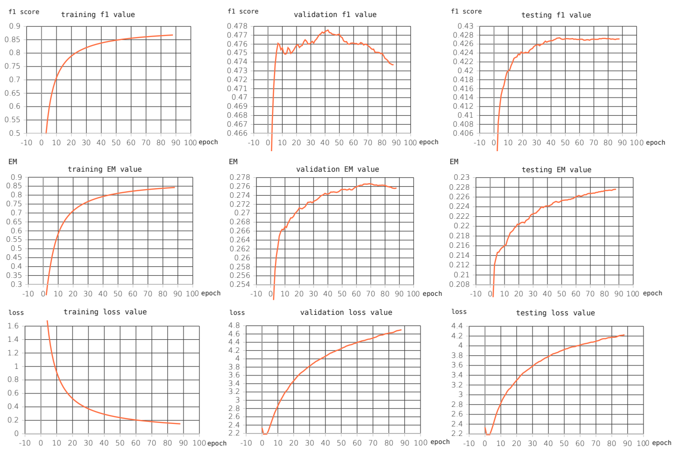

Figure 6. **The results after training.** F1-score(top), exact match(middle) and loss value(bottom) for each epoch after training the model.

VI. Discussion

We picked the model after the training in epoch 42 as our best model because there was a sign of overfitting after epoch 42, as shown in the graph for validation F1 value in Figure 6. Overfitting occurs when a model is trained too much that it only fits its training data. We can see that the F1 score for the training dataset keeps increasing after each epoch, while the F1 score for the validation dataset starts to decrease and the F1 score for the testing dataset stops increasing after epoch 42. This shows very clearly that the model starts to overfit after epoch 42.

By averaging the F1 score and the EM score of our best model, we got a score around 0.45, which was not a very good performance. We consider two reasons why we could not get a better performance for our model. Firstly, we didn't preprocess the QA dataset other than splitting them into the training, validation and testing datasets. The performance of the model could be improved if we preprocessed the datasets more. Secondly, we didn't do much hyperparameter tuning that could also help improve the model performance.

Although the performance of our best model was not great, it was not too bad given the difficulty of the QA task. For some other tasks like binary classification, a score of 0.45 will be considered poor, because even in the worst case, a model could randomly predict a label, and it can still have a 50% chance of getting the right label. QA task is a lot more complex, there are a lot more possible answers, so getting a provided answer correct cannot be done by simply guessing.

Given the above reasoning, we consider that transfer learning from a pretraining stage to a fine-tuning stage with the QA task is possible for our model. However, other models can also be considered, and there is a possibility that the performance would be improved with the same or similar experimental setup. On one hand, we could consider some models such as RoBERTa [16] and ELECTRA [17] that have a better performance than BERT; on the other hand, we could keep using BERT but with a larger structure consisting of more encoders. Ensemble methods using multiple models could also be considered.

VII. Conclusion

In this paper, we introduced the concept of a Question Answering system, the general structure of a state-of-the-art language model named BERT. Moreover, we showed how to use the pretrained model German BERT to build a QA system, which achieved a decent performance considering the difficulty of the task. Furthermore, we showed that transfer learning from the pre-training stage to the fine-tuning stage with the QA task for the German BERT model should be possible, since our experiment was not very extensive and the model can still be improved.

References

[1] Manvi Breja and Sanjay Kumar Jain. A survey on why-type question answering systems. *CoRR*, abs/1911.04879, 2019.

[2] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

[3] Timo Möller, Julian Risch, and Malte Pietsch. GermanQuAD and GermanDPR: Improving Non-English Question Answering and Passage Retrieval, 2021.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[5] Robert F Simmons. Answering english questions by computer: a survey. *Communications of the ACM*, 8(1):53–70, 1965.

[6] Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224, 1961.

[7] Ellen M Voorhees, Dawn M Tice, et al. The trec-8 question answering track evaluation. In *TREC*, volume 1999, page 82. Citeseer, 1999.

[8] Wei Yang, Yuqing Xie, Aileen Lin, Xingyu Li, Luchen Tan, Kun Xiong, Ming Li, and Jimmy Lin. End-to-end open-domain question answering with BERTserini. *arXiv preprint arXiv:1902.01718*, 2019.

[9] Peilin Yang, Hui Fang, and Jimmy Lin. Anserini: Reproducible ranking baselines using lucene. *Journal of Data and Information Quality (JDIQ)*, 10(4):1–20, 2018.

[10] German BERT. https://huggingface.co/bert-base-german-cased.

[11] Pritesh Prakash. An explanatory guide to BERT tokenizer, Sep 2021. https://www.analyticsvidhya.com/blog/2021/09/an-explanatory-guide-to-bert-tokenizer/.

[12] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

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

[14] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[16] 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. *arXiv preprint arXiv:1907.11692*, 2019.

[17] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020.