Replication and Extension of BERTserini: End-to-end Open-Domain Question Answering

Riccardo Bosio
Data Science and Engineering
Polytechnic of Turin
Turin, Italy
s291299@studenti.polito.it

Beatrice Macchia
Data Science and Engineering
Polytechnic of Turin
Turin, Italy
s292178@studenti.polito.it

***Abstract*—Question answering (QA) is a branch of artificial intelligence within the natural language processing and information retrieval fields; it aims at building systems that answer questions posed in a natural language by humans. As a matter of fact question answering is commonly used to build conversational client applications, which include social media applications, chat bots, and speech-enabled desktop applications. Current open-domain question answering systems often follow a Retriever-Reader architecture, where the retriever first retrieves relevant passages and the reader then reads the retrieved passages to form an answer. In this project we investigate and reproduce one of these end-to-end systems: BERTserini. It integrates a BERT reader with the open-source Anserini information retrieval toolkit built on top of the popular open-source Lucene search engine. The BERT model is fine-tuned on SQuAD, a reading comprehension dataset, consisting of questions+answers pairs posed on a set of Wikipedia articles. Moreover we propose and test three other BERT-based reader models: RoBERTa, ALBERT and SpanBERT. Finally we introduce a new re-ranking strategy for retrieved contexts based on RIDER. We observe an improvement in terms of Exact-Match and F1-Score by using RoBERTa as a reader, thanks to its enhanced training strategy. Instead, RIDER-based contexts re-ranking does not improve the performances of our model, suggesting that the retriever is already performing well enough. The code is available at https://github.com/riccardobosio/QA.**

***Index Terms*—NLP, QuestionAnswering, BERTserini**

I. Problem Statement

Question Answering systems allow users to ask a question in natural language and get a specific and immediate answer: depending on the model used, the answer can be directly extracted from text or generated from scratch. QA models are now a critical NLP topic, as they can be easily found in search engines and phone conversational interfaces. They are often used to automate the response to frequently asked questions by using a knowledge base (e.g. documents) as context. As such, they can be useful as customer support for smart virtual assistants, or to set up FAQ bots. While in closed-domain QA answers are retrieved within a specific area, open-domain QA is the task of question answering on open-domain datasets such as Wikipedia: questions can cover any topic, and consequently it makes it even more difficult to the QA system to find the right answer.

Traditionally, OpenQA systems comprised three stages:

- **Question Analysis:** it aims at understanding and classifying the question, so that it facilitates the retrieval of question-relevant documents;
- **Document Retrieval:** it obtains a small number of relevant documents from a collection of unstructured documents;
- **Answer Extraction:** it extracts the final answer to user question from the relevant documents gotten in the previous step.

Recently deep learning techniques have been successfully applied to OpenQA in almost every stage. For example, CNN-based models and LSTM-models have been used to classify the given question, leading to better performances. As a matter of fact, the success of answering a question is highly dependent on the performance of Question Analysis: the classification errors would easily result in the failure of answer extraction, thus severely hurting the overall performance of the system.

Neural models are able to automatically transform questions from natural language to representations that are more recognisable to machines and to provide powerful solutions to Answer Extraction.

DrQA [1] has been the first one to incorporate neural models in OpenQA, evolving to a "Retriever-Reader" [2] architecture:

- **Retriever:** it is responsible for collecting relevant documents from a given corpus w.r.t. the question;
- **Reader:** it aims at identifying the final answer from the received documents.

It is now considered the most efficient structure. A representation of its architecture can be seen in Figure 1.

II. Methodology

The BERTserini [3] architecture can be seen in Figure 2: it is basically composed of an Anserini [4] retriever and a BERT reader. The retriever identifies the paragraphs of text containing the answer. The selected paragraphs are then passed to the reader to identify an answer span.

As corpus we use the "enwiki-paragraphs" pyserini pre-built Lucene index. Pyserini [5] is a Python toolkit for reproducible information retrieval research.

We describe each module in the following subsections. Concerning the code, we have a `BERTserini` class containing

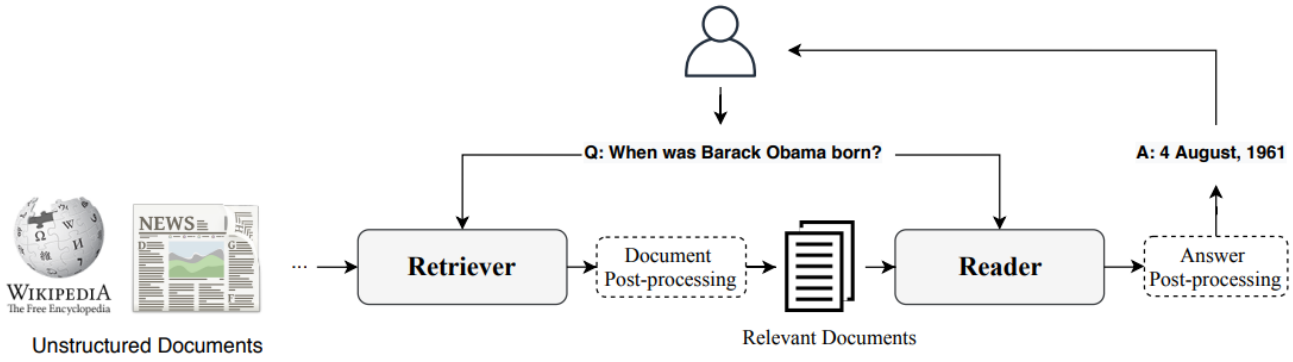

Fig. 1. The "Retriever-Reader" architecture.

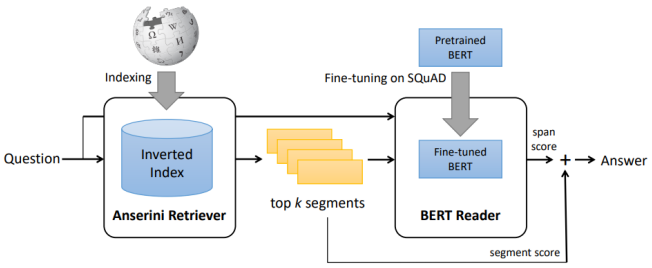

Fig. 2. BERT-Serini architecture.

both a `Retriever` object (the retriever) and a `BERT` object (the reader).

A. Anserini retriever

Instead of a multi-stage retriever that first identifies relevant documents and then ranks passages within, we use a single-stage retriever that directly identifies segments of text from Wikipedia. We retrieve top-k paragraphs using BM25 as the ranking function and the question as a "bag of words" query. BM25 (where BM stands for Best Matching) is a ranking function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document

In order to build our retriever we exploit Pyserini, which is a Python toolkit for reproducible information retrieval research with sparse and dense representations. It is designed to provide an easy-to-use first-stage retrieval in a ranking architecture.

More specifically, the class `Retriever` initializes the LuceneSearcher, that requires a pre-built index as a single point of entry. Then the text of the question is passed to the `retrieve` method, which searches the top-k hits. Each hit is finally transformed into a `Context`, by assigning its score, language and text.

B. BERT reader

As reader we use the pretrained BERT [6] ("bert-base-uncased") from Transformers [7].

BERT is an open source machine learning framework for natural language processing (NLP). It is designed to help computers to understand the meaning of ambiguous language in text by using surrounding text to establish context. The BERT framework was pre-trained using text from Wikipedia and can be fine-tuned with question and answer datasets. Historically, language models could only read text input sequentially – either left-to-right or right-to-left – but couldn't do both at the same time. BERT is designed to read text input both from left to right and from right to left at the same time. This capability, enabled by the introduction of Transformers [8], is known as bidirectionality. Using this capability, BERT is pre-trained on two different NLP tasks: Masked Language Modeling and Next Sentence Prediction:

- **Masked Language Modeling (MLM):** before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.
- **Next Sentence Prediction (NSP):** the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence of the first one in the original document; while in the other 50% a random sentence from the corpus is chosen as the second sentence.

Moreover the pre-trained BERT model that we use is "uncased", which means that it does not make any difference between "english" and "English".

One great BERT's potential is that, after pre-training in an unsupervised way on a massive amount of text data, it can be rapidly fine-tuned on a specific downstream task with relatively few labels, because the general linguistic patterns have already been learnt during pre-training. The intuition behind BERT is that the early layers learn generic linguistic patterns that have little relevance to the downstream task, while the later layers learn task-specific patterns.

We finetune it on SQuAD (The Stanford Question Answer-

ing Dataset) [9]: a reading comprehension dataset, consisting of questions posed on a set of Wikipedia articles, where the answer to every question is a segment of text from the corresponding reading passage. It contains 100,000+ question-answer pairs on 500+ articles.

This fine-tuning has been made using 80% of both SQuAD training and validation sets, learning rate equals to $3e-5$, both train and batch size equal to 8 and $num\_epochs = 2$.

In order to build our reader we use Transformers [7] by Hugging Face: it is a state-of-the-art Machine Learning library that provides APIs to easily download and train models.

The reader is managed by the `BERT` class. At inference time, the `predict` method is applied: a `SquadExample` object is created for each retrieved question+context pair. The resulting list of examples is then transformed into a list of features that can be directly given as input to the model. In order to allow comparison and aggregation of results from different segments, we remove the final softmax layer over different answer spans.

Now we combine the retriever score with the reader score: we use linear interpolation to compute the final score:

$$SCORE = (1 - \mu) \cdot \text{score}_{Retriever} + \mu \cdot \text{score}_{Reader}$$

with $\mu \in [0, 1]$. μ is a parameter that can be finetuned; in our experiments we set $\mu = 0.5$.

III. Extensions

We propose two extensions: some alternatives to the BERT reader and a re-ranking passage for retrieved paragraphs.

A. Reader alternatives

- **RoBERTa:** Just like BERT, RoBERTa [10] (which stands for "Robustly Optimized BERT pre-training Approach") is used to improve performances in NLP tasks as it makes use of embedding vector space that is rich in context. It has the same structure as BERT and surprisingly it does nothing but simply optimize some hyper-parameters for BERT. These simple changes sharply enhance the model performance in all tasks as compared to BERT.
  The authors of RoBERTa suggest that BERT is largely undertrained and hence they brought some improvements to it:
  - More training data (16G vs 160G) and a larger batch size;
  - Use of dynamic masking pattern instead of static masking pattern;
  - Removal of the next sentence prediction task;
  - Training on longer sequences.
- **ALBERT:** It was proposed by researchers at Google Research in 2019 [11]. It aims at improving the training and results of BERT architecture by using different techniques like parameter sharing, factorization of embedding matrix, Inter sentence Coherence loss.
  The backbone of ALBERT architecture is similar to BERT with GELU (Gaussian Error Linear Unit) activation function. However, below there are the three main changes that are present in ALBERT but not in BERT:
  - **Factorization of the Embedding matrix:** The authors separated the input layer embeddings and hidden layer embeddings. This is because input-level embedding needs to refine only context-independent learning but hidden level embedding requires context-dependent learning. This step leads to a reduction in parameters by 80%.
  - **Cross-layer parameter sharing:** Parameter sharing between different layers of the model aims at improving efficiency and decreasing redundancy. This step leads to a 70% reduction in the overall number of parameters.
  - **Inter Sentence Coherence Prediction:** As BERT, ALBERT uses Masked Language model in training. However, instead of using NSP (Next Sentence Prediction) loss, ALBERT uses a new loss called SOP (Sentence Order Prediction).

  As a result ALBERT model has a smaller parameter size as compared to BERT due to the above changes authors made in the architecture. For Example, BERT base has 9x more parameters than the ALBERT base, and BERT Large has 18x more parameters than ALBERT Large.
- **SpanBERT:** It [12] deviates from BERT in three specific ways:
  - The model uses a different random process to mask spans of tokens rather than individual ones.
  - Introduction of a novel auxiliary objective - Span Boundary Objective (SBO), which tries to predict the entire masked span using only the representations of the tokens at the span's boundary.
  - SpanBERT samples a single contiguous segment of text for each training example (instead of two) and thus, does not use BERT's next sentence prediction objective. (It is inspired by RoBERTa)

B. RIDER re-ranking

Retrieving passages from a very large corpus, the retriever could fail at ranking the most relevant passages at the very top, then better performances could be achieved by working on the retrieval.

This is the reason behind our desire to extend the project with a re-ranking passage for retrieved paragraphs, as proposed by [13]. It is a simple idea, which does not involve training and merely re-ranks the retrieved passages based on their lexical overlap with the top predicted answers of the reader before re-ranking. The intuition is that, regardless of their correctness, the top predictions of the reader are closely related to the true answer.

More specifically, we denote the initially retrieved contexts of the retriever as $\mathbf{C}$ and we denote the top-N predictions of the reader on the top-k contexts $\mathbf{C^{:k}}$ as $\mathbf{A^{:N}}$. RIDER creates a re-ranked list $\mathbf{C'}$ following these steps:

1) It scans $\mathbf{C}$ from the beginning;
2) If a context $c \in \mathbf{C}$ contains any $a \in \mathbf{A^{:N}}$, then c is appended to $\mathbf{C'}$;

3) The remaining retrieved contexts are finally added to $\mathbf{C}'$ according to their original order.

Therefore if the reader initial prediction is correct, RIDER moves all the contexts containing the correct answer to the top (if not already). While if the reader prediction is wrong, RIDER could be helpful if the the predicted answer co-occurs with the correct one. It means that, if the reader's performances are reasonably good, then RIDER is likely to re-rank the retrieved passages well.

IV. Experiments

In this section we introduce the metrics used in our experiments and then we present the following results: BERT finetuning on SQuAD, different readers testing, RIDER re-ranking.

All experiments are made on Google Colab, using mainly P100 GPU. To replicate the experiments, you can follow the GitHub instructions, which include the installation of needed libraries (through requirements.txt file).

A. Metrics

There are two main metrics used by many question answering datasets, including SQuAD: exact match (EM) and F1 score. Both the scores are computed on individual question&answer pairs. However in some cases multiple correct answers are possible for a given question, then the maximum score over all possible correct answers is computed. Finally, overall EM and F1 scores are computed for a model by averaging over the individual example scores.

- **Precision:** it is computed over the individual words in the prediction against those in the True Answer. In particular:

$$p = \frac{\text{No. of shared words between prediction and GT}}{\text{No. of total words in the prediction}}$$

- **Recall:** following the same principle:

$$r = \frac{\text{No. of shared words between prediction and GT}}{\text{No. of total words in the GT}}$$

- **F1-score:** it is a common metric for classification problems. It exploits the potentials of recall and precision by calculating their harmonic mean, i.e.:

$$F1 = 2 \cdot \frac{\mathrm{p} \cdot \mathrm{r}}{\mathrm{p} + \mathrm{r}}$$

- **Exact-Match:** it is a true/false metric that measures each question&answer pair. If the predictions match the correct answers exactly, then the EM = 1 or else the EM = 0.

$$EM = \begin{cases} 1 & \text{if predicted answer=correct answer,} \\ 0 & \text{otherwise} \end{cases}$$

B. Fine-Tuning results

In Table IV-B you can see the results obtained fine-tuning BERT reader on SQuAD. As a basic model we use "bert-base-uncased" from Transformers.

This fine-tuning has been made using 80% of both SQuAD training and validation sets, learning rate equals to $3e-5$, both train and batch size equal to 8 and $num\_epochs = 2$.

The fine-tuning took 1 hour and 53 minutes.

Reader	EM	F1-score
bert-base	80.18	87.69

C. Different Readers results

These are the results obtained testing four different reader models on 100 examples taken from SQuAD and passing $k = 10$ contexts. As explained in the previous section, we decided to evaluate three BERT variants: RoBERTa, spanBERT and ALBERT.

Reader	EM	F1-score
bert-base	34	44.17
roberta-base	40	52.50
spanbert	35	42.5
albert-base	37	46.36

As reported in Table IV-C, we can see that RoBERTa performs better than BERT, SpanBERT and ALBERT both in terms of Exact-Match and F1-score. The reason behind this success could be that, as described in the previous section, RoBERTa's authors have changed the training structure of BERT, making its performances higher.

In order to evaluate the entire end-to-end system using a specific kind of reader, we take a question&answer pair from SQuAD at a time, we feed the question to our retriever and we obtain a list of contexts. Then these contexts are passed to our reader model together with the original question. The reader predicts an answer, so that we can compute metrics comparing the predicted answer with the true original answer.

The time needed by this process for each model, using the parameters described above, depended a lot on Google Colab's performance. It usually took a lot of time, ranging from 15 to 20 hours. Also there was high risk of being kicked out because of inactivity or disconnected runtime.

D. RIDER Results

We evaluate our system using BERT as reader and applying RIDER re-ranking, assigning different values to both k (the top-k contexts) and N (the top-N predictions). The more k and N increase, the more the re-ranking becomes useless, so we use very small values for both of them. We use the same evaluation method described in the previous subsection and we compute EM and F1-score as evaluation metrics. The results are reported in Table IV-D. The time needed for the evaluation process with RIDER re-ranking is slightly higher than the standard evaluation because of the additional retrieving-reader step.

Unfortunately this re-ranking strategy does not bring any additional value to our model. We could observe that the contexts used to extract the final answer have been successfully re-ranked, but it was not sufficient to bring about a change in terms of final answer. The majority of contexts present in the top-k retrieved ones are still present after re-ranking and consequently, answers actually change in a very small number of cases.

	EM	F1-score
NO RIDER k=5 (baseline)	33.3	40.9
RIDER k=5, N=1	33.3	40.9
NO RIDER k=10 (baseline)	30.0	39.6
RIDER k=10, N=1	30.0	39.6
RIDER k=10, N=5	30.0	39.6

V. Conclusions

We implemented our own BERTserini system to tackle end-to-end open domain question answering. Despite a simple architecture, it gives satisfactory results, in particular using RoBERTa as reader.

BERT is undoubtedly a breakthrough in the use of Machine Learning for Natural Language Processing. The fact that it's approachable and allows fast fine-tuning has allowed a wide range of practical applications. However, improvements in the training phase of BERT has shown to lead to significant increase in performances, as we observed with RoBERTa's behaviour as a reader: it improves by 6 points the EM score and by 8 points the F1-score w.r.t. BERT.

Concerning RIDER instead, we can state that the retriever is already performing well and probably the contexts containing the right answer are already retrieved at the first step, so the re-ranking does not lead to improvements. We suppose that it could lead to some kind of improvement if we tested it on the entire SQuAD dataset, or at least the system would achieve the same results as if there was no re-ranking. Anyway due to our limited computing resources, we have not been able to do this kind of test yet.

References

[1] D. C. et al., "Reading wikipedia to answer open-domain questions," 2017.

[2] F. Z. et al., "Retrieving and reading : A comprehensive survey on open-domain question answering," 2021.

[3] W. Y. et al., "End-to-end open-domain question answering with bertserini," *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

[4] P. Y. et al., "Anserini: Enabling the use of lucene for information retrieval research," 2017.

[5] Pyserini, "https://github.com/castorini/pyserini."

[6] J. D. et al., "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[7] H. Face, "https://huggingface.co/models."

[8] A. V. et al., "Attention is all you need," 2017.

[9] P. R. et al., "Squad: 100,000+ questions for machine comprehension of text," 2016.

[10] Y. L. et al., "Roberta: A robustly optimized bert pretraining approach," 2019.

[11] Z. L. et al., "Albert: A lite bert for self-supervised learning of language representations," 2019.

[12] M. J. et al., "Spanbert: Improving pre-training by representing and predicting spans," 2019.

[13] Y. M. et al., "Rider: Reader-guided passage reranking for open-domain question answering," 2021.