Projet INF8460 - Team 02-15

Randy Sab-Roy
Polytechnique Montréal
randy.sab-roy@polymtl.ca

Clément Bernard
Polytechnique Montréal
clement-1.bernard@polymtl.ca

Thierry Dubois
Polytechnique Montréal
thierry.dubois@polymtl.ca

Abstract

This project consists in implementing a question answering pipeline and comparing different approaches for question answering in natural language processing. This pipeline is separated in three different parts: Document Representation, Document Ranking, and Question Answering. Using different techniques, we were able to create a pipeline that is reasonably performing considering the limited resources at hand.

1 Introduction

The natural language processing has gradually evolved with the growth of artificial intelligence. What we can do nowadays is quite large and is not limited to binary classification of the feeling of a text. We can now ask a machine to output an answer to a question, given a context. Such tasks have become popular with the Stanford Question Answering Dataset from Stanford (SQuAD) (Rajpurkar et al., 2016b) in 2016. This dataset contains more than 100,000 questions with answers coming articles of Wikipedia. It has been upgraded in 2018 (Rajpurkar et al., 2016a) to provide unanswerable questions. Therefore, the machine needs to not only find the best answer from a context, but to figure out if the question is answerable or not given the current context.

The chatbot (system where a user can have a question-answer conversation with a machine) task is quite challenging for machines : it requires both the comprehension of the natural language and general knowledge about the world. This article presents our method to answer this problem. This large problem is separated into three subtasks.

The first task is the representation of documents in order for them to be understood by the machine. The goal is to embed and summarize the most relevant features from the document. There are many ways to do this, including statistical approaches and semantic approaches.

The previous sub-task simplifies the last problem to solve: extracting the answer (sequence of tokens) to a question, given a context to retrieve it from. In our case, the machine needs to be able to infer that the question is unanswerable (in the current context), and give no answer to these type of questions.

In this work, we explain our different experimentations toward the sub-problems (3.1,3.2). We implemented different algorithms for the Document Ranking task and Question Answering, but we used pre-trained contextual embedding for the answer extraction task. Then, we describe the state-of-the art approaches for each step of the problem (4). Finally, we discuss the results of our work (3.1.1, 3.2.1), which show how our methods improve on the baseline set at the beginning of the project.

Significant work as already been done on all the respective sub-tasks. Classical QA systems use a non-machine learning probabilistic approach to document retrieval, using variations of the TF-IDF bag-of-words (Chen et al., 2017). Then, most QA systems use a machine-learning approach for the answer extraction. For example, some systems like DrQA (Chen et al., 2017) use a multilayer bidirectional LSTM network to encode the paragraphs. Then, the question itself is encoded using a RNN on top of word embeddings in this specific case. In this work, we tried to implement the BERT + model3 architecture of the paper (Hu, 2019), and we used the BERT pre-trained on SQuAD2.0 to achieve our best results.

2 Methodology

Our general approach concerning the methodology was to keep our pipeline as simple as possible.

Our hypothesis was that using basic methods to give early results would prove to be a good investment to save time during the course of the project and competition. These early results would give us plenty of time to adjust our course and optimize the sub-tasks as needed. This means that we wanted to have as much control over our data, using as little machine-learning methods as possible. We hypothesized that diving too deep in machine learning would be a pitfall hard to climb out of, considering our limited resources, the limited time span, and the complexity of the subject.

2.1 Document Representation & Ranking

Our document representation is tightly related to our document ranking approach, hence why these two subsections are merged into the following paragraphs. Two different approaches were tested in the context of this project. Our baseline approach was to use TF-IDF on our corpus, and then compare the similarity of each question to every paragraph to choose the closest match (using cosine distance). Multiple methods of document ranking were tested during the development of the pipeline.

The simplest method we found that we could test early in the project was the Okapi-BM25 algorithm (Willet, 1988). Since we are using this algorithm to rank the documents, it is important to represent the corpus as usable data for this algorithm (TF and IDF). The first step is to create a set containing every individual type that exists in the list of questions we are going to use. That is, of course, after proper pre-processing of said questions (removing stop words and words of length 1, removing HTML tags, lemming, then stemming).

Following the creation of this set, we scan each paragraph and create a dictionary that has query types as keys, and the count of these types in a given paragraph as the key. A paragraph is now represented by its ID and by the frequencies of the different types tokens that it contains. This representation facilitates the computation for the Okapi-BM25 algorithm. We hypothesized that most of the time, a specific sentence fits the question better than the whole paragraph. In the final implementation, each sentence in a paragraph is represented separately after the original paragraph representation.

$$D_1 : "aabbaacc" \tag{1}$$
$$D_2 : "abbbacccc" \tag{2}$$
$$TermFrequencies^{1} : \tag{3}$$
$$D_1 : \{a : 4, b : 2, c : 2\} \tag{4}$$
$$D_2 : \{a : 2, b : 3, c : 4\} \tag{5}$$
$$\tag{6}$$

The Okapi-BM25 algorithm is based on the frequency of the query tokens in a document, while taking into account the length of the said document compared to the average and the inverse document frequency of the query tokens. In other words, it is a probabilistic bag-of-word retrieval framework. The Okapi-BM25 score is defined as :

$$score(D, Q) = \sum_{i}^{n} IDF(q_i) \cdot \frac{f(q_i, D)}{f(q_i, D) + c_1} \tag{7}$$
$$c_1 = k \cdot (1 - b + b \cdot \frac{|D|}{avgdl}) \tag{8}$$
$$IDF = ln(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}) \tag{9}$$

where N is the total number of documents, $avgdl$ is the average document length and $n(q_i)$ is the number of documents where q_i appears. c_1 is the computed weight of the document length (normalized). b and k are hyper-parameters for the Okapi-BM25 algorithms, and are set to:

$$k = 1.0 \tag{10}$$
$$b = 0.675 \tag{11}$$

The hyper-parameter values were chosen by trial and error to optimize the results on the validation set. Our methodology included testing different variations of this algorithm, as the original BM-25 algorithm has some limitations. For example, it tends to discriminate unfairly against longer documents (Kamphuis, 2020), and can sometimes yield negative scores for words that appear in most documents. Usually, the latter is solved when processing the data (particularly when removing stop-words.) We tested BM25+(Lv, 2011) and TF $l \cdot \delta \cdot p \times IDF$(Rousseau, 2013), without improving our results.

2.2 Answer extraction

First of all, our baseline approach was to use the cosine distance. We had a question and a context, and we embedded each word in the context with TF-IDF. We then took the two words that

minimize the cosine distance between the question and the words of the context. We took the sub-sequence from these two words and return the candidate answer if the cosine distance was above a threshold of 0.01.

As expected, this approach was much too naive and didn't lead to optimal results. To do better with the Question Answering task, we base our implementation on BERT.

2.2.1 BERT

BERT takes as inputs a pair of question and context embedded with WordPiece. It adds special tokens such as [CLS] at the beginning of the sequences, and [SEP] to separate the query from the context. Then, each word is assigned to a segment("A" or "B"). "A" means this is either the query or the padding, and "B" means this is the context where the answer is. The input of BERT shouldn't have more than 512 tokens (for the question, context and the special tokens). Finally, the inputs of BERT can be summarized by the concatenation of three components : the token embedding, segment embedding and position embedding. Then, BERT uses 12 encoders of hidden size 768 before using a final layer to output the score of starting or ending of each word in the context.

For this work, we tried to use a pre-trained BERT model (SQuAD dataset with 24 layers, 1024 hidden states size with 336M parameters). We cropped the input to be 250 tokens at maximum, pad the input if there are fewer than 250 tokens, and delete the other tokens if there are more[2]. We applied two dense layers after the output of last hidden state of BERT. Let denotes the last hidden state by **G**. The equations of our model are the following :

$$\boldsymbol{U} = gelu(\boldsymbol{W_1 G}) \tag{12}$$

$$\boldsymbol{U'} = gelu(\boldsymbol{W_2 U}) + \boldsymbol{G} \tag{13}$$

where gelu is the Gaussian Error Linear Unit [3]. Then the probability for starting or ending is computed as follows :

$$\boldsymbol{p_{start}} = softmax(\boldsymbol{W_{start} U}) \tag{14}$$

$$\boldsymbol{p_{end}} = softmax(\boldsymbol{W_{end} U'}) \tag{15}$$

where $\boldsymbol{W_1}, \boldsymbol{W_2}, \boldsymbol{W_{start}}, \boldsymbol{W_{end}}$ are learnable parameters. We also normalize the output after each activation, and add a dropout to the second dense layer.

To make the inputs consistent for our data, we decided to crop the embedding of the WordPiece tokenizer to 250.

As the output of our fine-tuned BERT does not explicitly specify if the answer is relevant or if it is better to consider the question as unanswerable, we add a criterion. We consider the 20 most probable positions as start or end positions, then consider all the combinations of pairs that are valid. We denote i_{start} and i_{end} the indexes of starting and ending in the tokenized inputs, we consider that a pair is valid if :

$$i_{end} - i_{start} <= maxAnswerLength \tag{16}$$

Then, we choose the pair that maximises the product :

$$i_{start}, i_{end} = \underset{i,j}{\operatorname{argmax}}(p_{start}(i) \times p_{end}(j)) \tag{17}$$

Once we have our candidate answer, we set $p_{start}(0) \times p_{end}(0)$ as the score of an unanswerable question and use a threshold (h) as follows :

$$p_{start}(0)p_{end}(0) - p_{start}(i_{start})p_{end}(j_{end}) > h \tag{18}$$

where h is a hyperparameter to optimize. If the inequality above is true, then we output the candidate, otherwise we output '$< NoAnswer >$'.

2.2.2 BERTv2 model

After spending few weeks on the pre-trained model, we didn't perform well with this model. We did the training on 50 000 queries for 5 epochs, but the loss stayed far away from 0. Therefore, to get better results, we found a BERT model that was pre-trained on SQuAD2.0 dataset(Rajpurkar et al., 2016a). It is fitted to detect unanswerable queries, which is the problematic we tried to fix with the fine-tuned model. This improved our performance, as seen in section 3.2.1.

3 Experiments

The corpus from the dataset that we used is divided into 83,327 paragraphs. The training set is comprised of 106,176 tuples (question, context,

[2] In the training set, there are only 0.3 % of the answers that are after the first 250 tokens.

[3] gelu(x) = $0.5x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$

answer) where the contexts are document indices from the corpus. To complete this dataset, we have a validation set and testing set with 10,000 queries each. All the following experimentations (except the fine-tuning of BERT) was done using the validation set of 10,000 queries.

3.1 Paragraph Ranking & Representation

Using the BM-25 algorithm, we are able to give a score to every single paragraph in the dataset, and keep the paragraph indices corresponding to the top-n scores, also known as $Precision\ at\ n$ metric (the right paragraph id is one of the first n entries $x\%$ of the time). We are using two metrics for this task. The first one, exact-match (EM) measures if we obtain the exact paragraph that was presented as the right one in the dataset. The second measure, which we call segment-presence (SP) is true when the exact wording of an answer is given in the specified paragraph. With this metric, we can evaluate if it was possible to find the answer without the exact paragraph needed. We expect the results to be growing with higher n values, asymptotically at $y = 1$ (as it is impossible to have $y > 1$ and that all methods will have $y = 1$ at $x = 83,327$)

3.1.1 Results

The results for the 10,000 questions of the validation set can be seen in Figure 1. These results show that about 43% of the time, we retrieve the good paragraph at rank #1 (when the question is answerable), and 55% of the time, the answer is contained in the first paragraph we retrieve (BM25-SP). This is an important value to keep in mind moving forward, as our final performance can only be as good as this number (on top of $< NoAnswer >$ queries) if the answer ranking from different paragraphs proves not to be working in the answer extracting portion of this project.

3.1.2 Discussion

Optimizing this value is important for the answer extraction part, as distinguishing the right answers between two candidates with a confidence score is a hard task. We need to trust that our system will retrieve a sufficiently good paragraph at rank #1 most of the time, as we are not sure how to solve the task of ranking potential answers in our current pipeline and we need to reduce the bottle-necking of the pipeline as much as possible.

The BM25 algorithm outperforms the Word2Vec approach we had as a baseline[4], but does not quite reach the performance or DrQA (Chen et al., 2017) at 77.8% for the first 5 paragraphs. The SP precision at 10 is 79.1% for BM25. Interestingly, we observe that all methods follow a similar semi-logarithmic function (growing similarly in lower x values but asymptotic at y=1, as expected) with an $R^2 > 0.98$. As expected, the SP metric is significantly higher than EM for both our baseline and the BM25 algorithm. For example, the precision at 1 for BM25-SP has a value of 55.7% compared to 43.3% for EM. This indicates that we have a potential gain of approximately +1200 questions that we could still find the right answer for, even without the right paragraph.

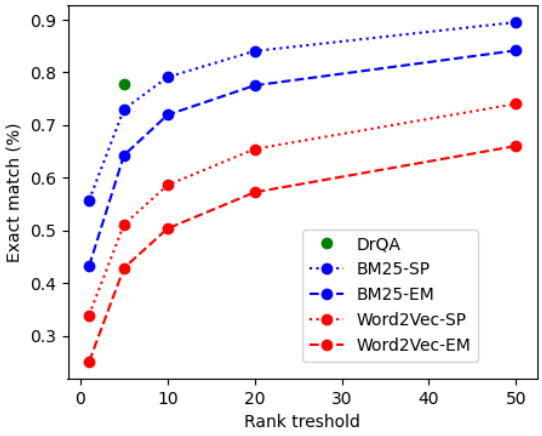

Figure 1: Comparison of a@n performance

3.2 Answer extraction

We first tested a BERT model pre-trained on the SQuAD dataset, then our fine-tuned model and finally another BERT model pre-trained on the SQuADv2 dataset (which includes the prediction of unanswerable questions). When the score is cited with 'prediction', it means we consider the top-1 paragraph from the document ranking part. The metrics used in this section are exact-match (exact same sub-string as answer) and F1 (adjusted score for partial answers). F1 score is calculated as such:

$$F1 = 2 \times \frac{accuracy \times recall}{(accuracy + recall)} \quad (19)$$

[4]The required TD-IDF baseline approach gave us memory error, and we did not spend the time necessary to fix this error, as we had already done another baseline

Model	EM	F1
TF-IDF	1.6%	2.8%
Word2Vec	17.9%	18.0%
BERTv1	30.1%	39.6%
BERTv2	**44.6%**	**51.1%**
Fine-tuned	16%	20.8%

Table 1: EM and F1 of our models

3.2.1 Results

As expected with the pre-trained BERT model, it performs well without fine-tuning in the validation set when given the ground truth context (f1 score of 72.7% and exact match of 63.2%), and achieve reasonably good results with our prediction on the top-1 paragraph (f1 score of 51.1% and 44.6% for EM). Unfortunately, the fine-tuned model performs quite badly in the training set when given ground truth paragraph id(f1 score of 32.9% and EM of 30.3%). On the validation set, the fine-tuned model gets points only for the $< NoAnswer >$ queries, which is an interesting behaviour. It could not answer answerable questions anymore, and performed a lot better on unanswerable questions. This can be due to the model learning that everything is unanswerable. It is important to note that answering $< NoAnswer >$ yields an EM of 18.25%. This indicates that our baseline models perform worse than a naïve model that would answer the most common answer everytime (Table 1).The optimal result was found using the model pre-trained on the SQuAD2.0 dataset (we named this model BERTv2 to distinguish it from the original BERT model used). The differences between these two models can be seen in Fig. 2

We also generated more results using the first 5 paragraphs for each question, and only kept the answer with the best cumulative confidence score $(P(start) + P(end))$, but no value between 2 and 5 paragraphs improved our scores. We did not have the time to investigate this behaviour, as generating answers for multiple paragraphs greatly increase the computation time and we were investigating methods that strictly improved our results. We think that this behaviour is due to the model finding answers in unanswerable questions, which would reduce our performance overall.

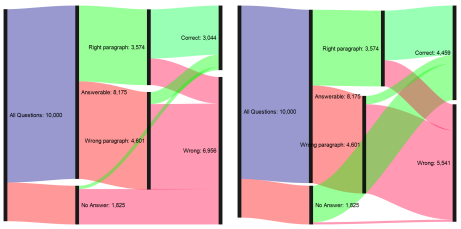

Figure 2: Result distribution for BERTv1 - BERTv2

3.2.2 Discussion

The most obvious difference between the two flowcharts is the "No Answer" section. The first model was only able to detect 154 cases, versus 1725 for the model that was trained for this task (8.4% vs 94.5%). This variation represents, by itself, an upgrade of +15.71% on the final results, which is very significant. With that said, this change corresponds with a small loss of performance for the answerable questions where we did not have the right paragraph to begin with (12.4% vs 8.5%). However, this results in a net gain of +1392 correct answers, as the share of wrong paragraphs lost was quite small. We think that this is due to the model classifying these questions as unanswerable questions. We can confirm this hypothesis by showing that the second model classifies 52.32% of the queries as unanswerable, compared to 8.77% for the first model. Overall, the model trained on SQuAD2.0 offers a relative gain of +46.48% for EM in relation to the one trained on the SQuAD dataset.Our prediction of a maximum score of 55% accuracy for the answerable questions is confirmed, as we get a score of 33.44% for these questions.

These metrics are aligned with the Kaggle competition results. We won this competition for both the pre-trained embeddings category and the other category (in which we were eligible by only using pre-trained models for the answer extraction section), with a final EM score of about 45%. This result confirms our hypothesis that using simpler methods would be a better approach for the scope of this project, as no other teams (with potentially more complicated approaches) broke the 40% mark.

4 Related work

To embed a word to be understood by machines, it is required to take into account the context around the word and not to consider it as if it were in a vacuum. The order of the words is crucial, and

it should be highlighted in the embedding of it. A classic way to do it is with the Word2Vec approach (Mikolov et al., 2013), where a neural network is trained with either the word or the context as input, and either the context or the word as output (Skip-gram or CBOW methods). Once each word is embedded, there different ways to consider the average of the embedding of each word of a sentence. With a vector representation for each sentence, the issue is to find the paragraphs that are more likely to contain the answer. To do so, (Lee et al., 2018) uses two different embeddings for the questions and the paragraphs, and then compute the probability that contains the paragraph contains the answer, with either the sigmoid function of a dot product or a MLP. All these approaches are based on the use of a similarity metric between the question and the paragraphs. These approaches are done without pre-trained contextual embeddings, which are the state-of-the art methods for this task.

Once we know where the answer should be, finding the exact sub-sentence that answers the question remains to be done. To do so, BERT (Devli et al., 2018) (Bidirectional Encoder Representations from Transformers) is an innovative approach that has achieved most of the best scores in the past years (F1 score of 93.2% on the SQuAD testing set and F1 score of 83.1% on SQuAD2.0 testing set). The original paper (Devli et al., 2018) mentions to fine-tune BERT with a last output layer to reach high performances in a lot of tasks. Recent methods try to be more specific in the fine-tuning by normalizing the probability of a word to be the start or the end of the answer over multi-passages (Wang et al., 2019). Another method uses siamese-networks and either cosine distance or softmax classifier to bypass the bottleneck of BERT, which is the time spent to extract answers. Finally, (Hu, 2019) uses three different models to fine-tune BERT and implements the detection of unanswerable questions. Training on BERT on the SQuAD2.0 dataset also solves this problem.

4.1 Conclusion

In conclusion, we managed to have satisfying results using relatively simple methods for the different parts of this project. The Okapi-BM25, while long to execute, had good performance compared to other methods and was getting close to the DrQA score. Since the score on the questions without the right paragraph were significantly lower than the one with the right paragraph, we expect the final accuracy to grow proportionally to the improvement on this measure for paragraph retrieval. There are other advanced implementations of this algorithm which we did not have the time to try, but we are confident that there is still space for optimization in this sub-task.

For the answer extracting, using a model pre-trained for finding out unanswerable questions is what gave us the largest gain in final accuracy. We think that our performance disparity compared to the other teams originates from the usage of this model. Unfortunately, our fine-tuning on BERT did not prove to work well. We think that by spending a bit more time figuring out the parameters for this training, we could figure out a way to make it work better.

While experimenting, we realized that the validation dataset is separated into 5,000 "easy" questions and 5,000 "hard" questions. The hard questions are missing punctuation or capitalization, have typos or are not even questions at all. On the first 5,000 questions, we had an exact match of 63% versus 17% on the last 5,000. With this observation, we can conclude that our model performs better on questions that are well formulated and have no typos, which makes sense. This steep separation between the two is interesting, as finding ways to better format the poorly worded questions could potentially increase performance by a significant margin. This could be done as a supervised pre-processing task with the "easy" questions used as style and format reference in future work.

References

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

Jacob Devli, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers forlanguage understanding. Version 1.

Zhangning Hu. 2019. Question answering on squad with bert.

de Vries A. P. Boytsov L. Lin J. Kamphuis, C. 2020. Which bm25 do you mean? a large-scale reproducibility study of scoring variants.

Jinhyuk Lee, Seongjun Yun, Hyunjae Kim, and Miyoung KoandJaewoo Kang. 2018. Ranking paragraphs for improving answer recall inopen-domain question answering.

Zhai C. Lv, Y. 2011. Lower-bounding term frequency normalization.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. Version 3.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2016a. Know what you don't know: Unanswerable questions for squad. Version 2.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016b. Squad: 100,000+ questions for machine comprehension of text. Version 1.

Vazirgiannis M. Rousseau, F. 2013. Composition of tf normalizations: new insights on scoring functions for ad hoc ir.

Zhiguo Wang, Patrick Ng, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2019. Multi-passage bert: A globally normalized bert model foropen-domain question answering.

Peter Willet. 1988. *Document retrieval systems*. Taylor Graham Publishing500 Chesham House 150 Regent St. London W1R 5FAUnited Kingdom.