Information Retrieval: Neural retrieval model - Document ranking

Arno Deceuninck (s0181217)

Approach used

To make this assignment, I started taking a look at the tutorials given in the assignment (Winkler & Verma, 2021) (Verma, 2021). When following this, I came across the Python sentence-transformers library (Reimers, 2021). This library contained an easy interface to many pre-trained models that could be easily fine-tuned for your own tasks.
After testing some different pre-trained models, I decided to continue with the pre-trained MS MARCO model, which is made by Microsoft and trained on Bing questions and human generated answers (MSMarcoAI, 2020). This model was then fine-tuned using the provided dataset.

Conceptual Components

The main concept of the final model is based on the retrieve and re-rank model. To obtain the final ranked list, the query first gets processed by a bi-encoder. This bi-encoder is efficient, but doesn't always have a good performance. It works by mapping all documents to a vector space. When receiving a query, the query gets mapped to the same vector space and nearby documents get selected. Using this method, the top 50 results get selected. Next, those 50 results get reranked using a cross-encoder. A cross-encoder receives two input strings (in our case a query and a document string) and outputs a score stating how relevant the document is given the query. Based on this score, we can rerank the 50 results outputted by the bi-encoder and select e.g. the top 10 results of those with the highest score by a cross-encoder. Note that a cross-encoder achieves a better performance than a bi-encoder, but has a lower efficiency, which is why we can't just evaluate all results with the cross-encoder. (Reimers, 2021)
After experimenting a bit, it was clear the LuceneRetriever still gave the best results, so instead of using a bi-encoder, I used Lucene instead of a bi-encoder.

Implementation Details

Notebooks

I made the implementation inside Jupyter Notebooks, so my thoughts and process during the experiments could be better captured and you don't always have to wait several hours before getting results of a training.
The most important file is experiments.ipynb. The other two notebooks (sbert-doc.ipynb and tutorials.ipynb) contain some code which is mainly copied from the tutorials and adapted to our dataset.

Classes

DocumentRetriever

This is a super class of all document retrievers. The reason I used a class for this is to make it possible to rate all of them using the same raters. The most important method here is the retrieve_documents(self, query_number, n) method, which returns a list containing the ranked top-n documents for a given query_number.

LuceneRetriever

Retrieves the documents as Lucene would do it. Note that this doesn't actually execute Lucene code, but takes a look at raw_dev_Lucene_retrievals.csv.

GroundTruthRetriever

This retriever gives you the ground truth, always leading to a perfect recall and precision. This outputs the results from dev_data.csv.

Bm25Retriever

This is another lexical search retriever. It was used in the SBERT-Documentation to compare the results of their ReRank retriever, so I also used it to compare this next to the LuceneRetriever.

ReRankRetriever

The superclass for the retriever using the Retrieve & Re-Rank method, as described in the conceptual components. This first selects a set of candidate documents using an efficient retriever and then re-rank those documents (and select the most relevant ones) using a more performant, but less efficient retriever.

BiCrossRetriever

This is a ReRankRetriever that uses a bi-encoder for retrieving documents and a cross-encoder for re-ranking documents. By default, it uses a bi-encoder and a cross-encoder trained on MsMarco, but you can also supply another model.

LuceneCrossRetriever

This is a ReRankRetriever that uses the LuceneRetriever instead of a bi-encoder. This model was introduced since I noticed that the BiCrossRetriever gave worse results than I expected. By default, this also uses a cross-encoder based on MsMarco, but you can supply your own model if you want.

ModelRater

A rater class that checks how well retrievers perform against the GroundTruthRetriever. For this, the recommender recall@10 and precision@10 are used. We just generate the top 10 off the ground truth retriever and the top n of the given retriever and use those results to generate the recall and precision. Note that I used the top 10 of the GroundTruthRetriever instead of all results returned by it to achieve a better representing score, because otherwise a recall of 1 would be impossible (but now it’s possible with the GroundTruthRetriever). Note that this mainly focuses on retrieving and reranking and not so much on predicting the correct label (but of course those two are correlated to each other).

Training & Experiments

The code for training the models and experimenting with different parameters can also be found in the experiments.ipynb file.

Experiments

Some different experiments were done to find the best performing model and the best hyper parameters.

Retriever comparison

As mentioned in the previous chapter, there are different types of DocumentRetrievers. Here is an overview of how they compared against each other:

Name	Recall@10	precision@10
GroundTruthRetriever	1.00000	1.00000

LuceneRetriever	0.49958	0.49642
Bm25Retriever	0.43229	0.56536
BiCrossRetriever (MsMarco)	0.25137	0.30258
LuceneCrossRetriever	0.43721	0.43416

The GroundTruthRetriever was of course the one with a perfect score. The LuceneRetriever (which is the baseline) and Bm25Retriever were next on the list of best retrievers, but the assignment was to take a look whether we could improve the search results by using a neural model, which for both isn't the case. Since the BiCrossRetriever performed worse than expected and since the final task was to label and rerank the documents in the provided test dataset, I didn't spent time trying to train a bi encoder, but used Lucene to retrieve the initial documents first and rerank them using a cross encoder. This is done by the LuceneCrossRetriever, which (as expected) gave better results than the BiCrossRetriever (since it was more headed towards the LuceneRetrievers). Our main goal now is to improve the CrossRetriever used in the LuceneCrossRetriever to try to beat the scores of the LuceneRetriever.

CrossEncoder selection

Before trying to train the CrossEncoder, I first took a look at what the best performing cross encoders where. MsMarco was initially selected because the description in the sentence-transformers documentation matched most what I wanted to achieve with it.

Name	Recall@10	precision@10	Time
LuceneRetriever	0.49958	0.49642	431.44it/s
BiCrossRetriever (MsMarco)	0.24137	0.30258	1.62it/s
LuceneCrossRetriever (MsMarco-TinyBERT)	0.43721	0.43416	6.72it/s
LuceneCrossRetriever (MsMarco-MiniLM)	0.45076	0.44782	1.08s/it

LuceneCrossRetriever (QNLI)	0.36328	0.36055	3.20s/it
LuceneCrossRetriever (STSB-TinyBERT)	0.33324	0.33090	3.09it/s
LuceneCrossRetriever (STSB-Roberta)	ERROR	ERROR	ERROR

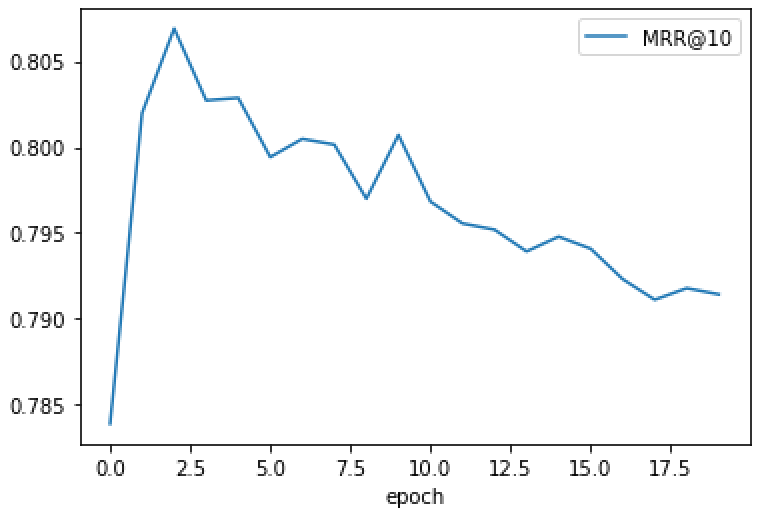
The description was accurate, since the MsMarco retriever indeed performed the best. The MsMarco-MiniLM performed slightly better, but the 6x speed difference was not worth the improvement. The STSB-Roberta model kept giving CUDA out-of-memory errors, but since the STSB-TinyBERT retriever also didn't provided that great results, I didn't spent a lot more time trying to fix it, because I was already struggling some time on it and it probably wouldn't give better results than MsMarco.

Training

Since we had chosen our pre-trained model, we could start fine-tuning those results based on the provided training dataset. I splitted this dataset into a training dataset (80%) and a validation dataset (20%), since the ground truth dataset was already used for testing the model at the end.

Epochs

For deciding the numbers of epoch, I tried 20 to see if it was still giving better results or already overtraining. To evaluate the model during the training, I used the CERerankingEvaluator, since this was the one that was the closest to what we wanted to achieve.

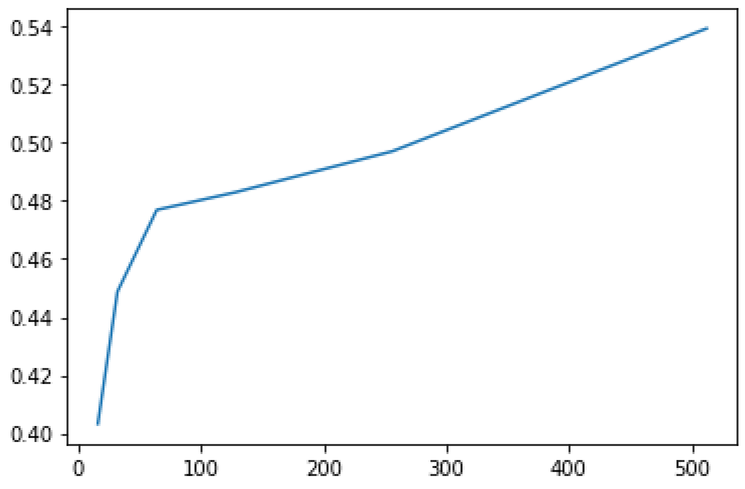


According to the CERerankingEvaluator the ideal number of epochs was at two epochs. The graph shows the score by the evaluator on the y axis. However, when I trained a model on 2 epochs, the model trained on four epochs was still slightly better with my own ModelRater, so I kept using that one.

Epochs	**CERerankingEvaluator score**	**ModelRater Recall@10**	**ModelRater Precision@10**
2	0.80691	0.53338	0.53012
4	0.80286	0.54272	0.53928
20	0.79141	0.53910	0.53571

Length

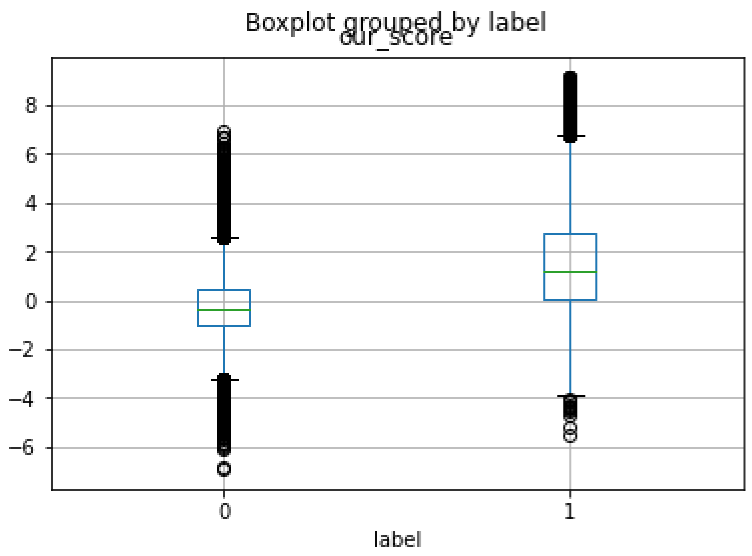
The max_length of our model was None, so it could probably compare strings of any length. I still tried to experiment with some maximum length. If a string (either the query or the document string) exceeds this maximum length, it gets truncated.



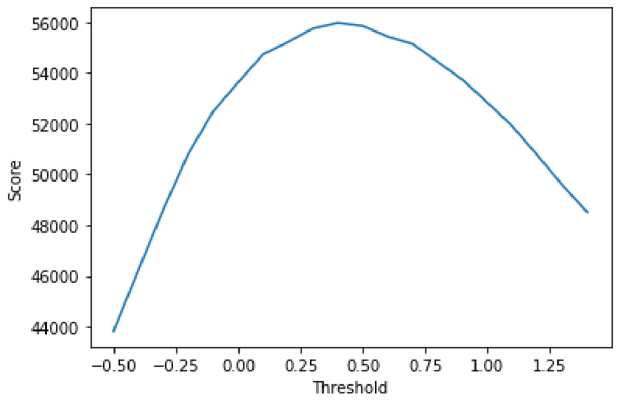
In the graph, the x-axis is the max_length and the y-axis is the precision returned by the ModelRater. As expected, the longer the maximum length, the better the results. However at length 512 it already gave the same results as without a maximum length set. This can be either because the first 512 tokens already describe the document accurately enough, or because somewhere in the model a maximum length is already applied, without having a maximum length said. The program crashes when I use 1024 as length. I just continued without setting a maximal length.

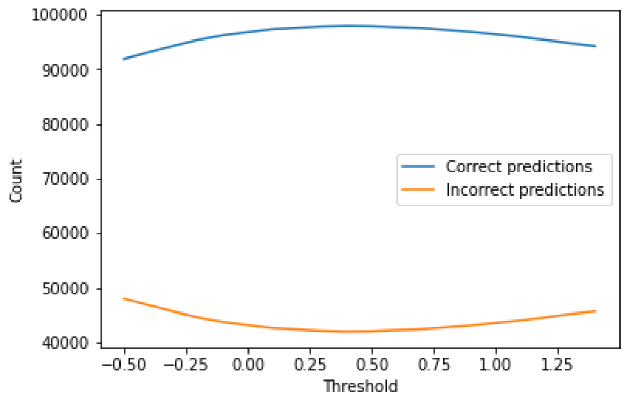
Relevance threshold

I currently mainly focussed on getting good recommendations given a query. To do this, our cross-encoder gives a score based on the relevance of that query. We also have to give a label whether a document is relevant for a given query, so we have to define a score threshold. If the score of the cross-encoder is above this threshold, the document gets considered as relevant. To determine our threshold, I first had a look at how our scores were distributed given the label.



The y-axis is the score of our cross-encoder. You can clearly see that on average the predicted scores are higher if the documents are relevant, but there won't be a threshold giving a perfect split. To find a good threshold, I tried iterating over some different values and calculated for each of them a score based on the number of labels the model got right (true positives and true negatives) and the number of labels the model got wrong (false positives and false negatives).





Based on this, I used a cross-encoder score of 0.4 as threshold for a document to be considered relevant.

Other parameters

There are still a lot of other hyperparameters you can play around with (e.g. train batch size, warmup steps, train/dev data distribution, …), but because of the long training times, I could only test a few.

We currently have been working with the LuceneCrossRetriever, but if you train a bi-encoder, you might be able to get a better initial retriever than Lucene. Feeding the results of this trained bi-encoder to the cross-encoder for re-ranking might give even better results, but we didn't test that.

Final Results

We calculated the recall and precision by comparing the top 10 of our model with the top 10 of the ground truth. This allowed us to use the Lucene retrievals as a baseline result.

Name	Recall@10	Precision@10
LuceneRetriever	0.49958	0.49642
Optimized LuceneCrossRetriever	0.54272	0.53928
Improvement	0.04314	0.04286

As we can see in the table, there is a slight increase (a bit more than 4%). This can probably still be increased if you experiment more with the other hyperparameters.
The training dataset also contained the data to determine a labeling recall and precision for our own model, but we didn't have enough data to determine a baseline recall for this with the LuceneRetriever (since training_data.csv only contained data already retrieved with Lucene, so no negative samples and raw_dev_Lucene_retrievals.csv didn't contain any labels).

Name	Recall	Precision
LuceneRetriever	UNKNOWN	0.49721
Optimized LuceneCrossRetriever	0.59636	0.68022
Improvement	UNKNOWN	0.18301

Here is the improvement already larger (18%), but we don't have an idea of the improvement in recall. But be careful with those results, since the recall and precision for our own model here are based on only the ones that were marked as relevant by Lucene, so they might not be accurate.

The most important message for ArchiveIR is that it's possible to improve the results using a neural retrieval model.

GitHub

All code can be found in the GitHub repository. In case the hyperlink doesn't work, you can also copy paste the link below to the address bar of your browser.
https://github.com/arnodeceuninck/neural-retrieval-bert

References

MSMarcoAI. (2020). *MS MARCO*. Microsoft Open Source. Retrieved January 3, 2022, from https://microsoft.github.io/msmarco/

Reimers, N. (2021). *Retrieve & Re-Rank — Sentence-Transformers documentation*. Sentence-Transformers. Retrieved January 3, 2022, from https://www.sbert.net/examples/applications/retrieve_rerank/README.html

Reimers, N. (2021). *SentenceTransformers Documentation*. SentenceTransformers Documentation — Sentence-Transformers documentation. Retrieved January 3, 2022, from https://www.sbert.net/

Verma, D. (2021, January 14). *Fine-tuning pre-trained transformer models for sentence entailment*. Towards Data Science. Retrieved January 3, 2022, from https://towardsdatascience.com/fine-tuning-pre-trained-transformer-models-for-sentence-entailment-d87caf9ec9db

Winkler, M., & Verma, S. (2021, June 4). *Semantic Search with S-BERT is all you need*. Medium. Retrieved January 3, 2022, from https://medium.com/mlearning-ai/semantic-search-with-s-bert-is-all-you-need-951bc710e160