Passage Re-Ranking Using Various Neural Network Architectures

Michal Rein, Bc.
xreinm00@fit.vut.cz

Karel Ondřej, Ing.
ondrej@fit.vut.cz

Abstract

In this paper, we present a study on constructing and evaluating three different neural network models for the passage re-ranking task. The models include a CNN-based model, an LSTM-based model, and a BERT-based model. Our experiments show that the BERT-based model outperforms the other two models, and is only slightly behind the BM25 algorithm in terms of performance. We conducted a detailed analysis of the ranking results and discuss the implications of our findings. We also suggest directions for future work.

1 Introduction

Passage re-ranking is a crucial task in information retrieval, where the goal is to rank a set of passages according to their relevance to a given query. The quality of the ranking can significantly impact the effectiveness of search engines and other information retrieval systems. Traditional ranking algorithms, such as the BM25 algorithm, have been widely used for this task, but there is ongoing research on the use of neural network models to further improve the performance of passage re-ranking. In this paper, we aim to contribute to this research by constructing and evaluating three different neural network models: a CNN-based model, an LSTM-based model, and a BERT-based model.

2 Task Definition

In this project, our focus is on the passage re-ranking task, which involves ranking a set of passages retrieved by an efficient document retrieval algorithm, according to their relevance to a given query. We specifically targeted the top 5 and top 10 rankings, as these are the positions that are most likely to be examined by users. The proposed method aims to enhance the ranking of retrieved passages compared to the scores generated solely by the retrieval algorithm. (Trabelsi et al., 2021)

3 Method

We use the BM25 algorithm as the retrieval method to retrieve a set of the top 100 documents for a given query. These documents serve as the input for our passage re-ranking models.

Our models are built using neural networks, with a focus on encoding the information contained in a passage in order to measure the similarity between the passage and a given query. The general architecture of our models consists of a preprocessing step, where we transform the input (query and passage pairs) into a suitable representation for the model. For the CNN-based and LSTM-based models, we use GloVe embeddings as the input representation, while for the BERT-based model, we use the preprocessing method recommended by the BERT authors (Devlin et al., 2019).

After the preprocessing step, the input representations are passed through an encoding layer, which transforms them into a global embedding representation. For encoding purposes, the siamese network architecture is used for each type of model. The global embeddings for the query and passage are then subtracted and the cosine distance between them is calculated. The concatenation of absolute and cosine distance is passed through a dense layer and then into a sigmoid dense node, which outputs a relevance score. The general architecture can be represented visually in Figure 1.

4 Experimental Setup

The experimental setup involved the use of TensorFlow and Keras frameworks to build and train the models using the Keras functional API. The MS MARCO[1] Passage Retrieval dataset was converted into a binary labeled dataset with query, document, and relevance label samples. A total of 1 million samples were used for training, with 20% of the data being reserved for validation purposes.

[1] https://microsoft.github.io/msmarco/

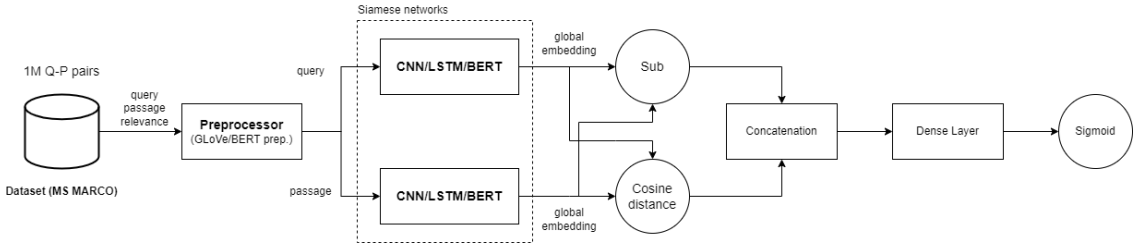

Figure 1: General architecture of proposed models

To create a usable dataset, the Pandas package was used to manipulate data from multiple TSV[2] files. The resulting dataset was then transformed into the TensorFlow Record Dataset format, which allowed for efficient training and data preprocessing.

The evaluation of the models was carried out on the dev split of the MS MARCO passage ranking dataset, using a total of 1000 unique queries. The main metrics used for evaluation were accuracy, recall, and mean reciprocal rank.

The models were trained on a computer with a 2x NVIDIA GeForce RTX 2080 Ti graphics card using the TensorFlow framework's Mirrored strategy for distributive training. The hyperparameters for each model included a learning rate of 0.001 optimized with the Adam optimizer, a batch size of 2048 (split equally between GPUs), and 30 epochs of training. The input sequences were encoded using siamese networks that shared trainable parameters. Sequences of text data were at the maximum of 128 long and padded when needed. The binary cross entropy function was used as the loss function, and other loss functions such as contrastive loss and hinge loss were also tested as introduced in (Mitra and Craswell, 2018) but did not outperform the binary cross entropy function.

In the following subsections, we will further discuss the specific implementation and decisions made for each architecture.

4.1 LSTM-based encoder

The LSTM-based encoder used in this project has a unit size of 512, resulting in a final hidden state output that is a 512-dimensional vector. We found that using two consecutive LSTM layers performed better than using only a single layer. The specific implementation of this encoder can be found in the file *siamese_LSTM_v6.py* in the project repository. The rest of the architecture is depicted in Figure 1, which includes a Dense layer with 256 fully-connected units. The total number of trainable parameters for this model is 3,486,465.

4.2 CNN-based encoder

The CNN-based encoder used in this project has the following configuration:

1. 1D Convolution with 256 filters and a kernel size of 3
2. 1D Convolution with 128 filters and a kernel size of 3
3. 1D Convolution with 64 filters and a kernel size of 5
4. 1D Convolution with 256 filters and a kernel size of 3
5. Global max pooling

The total number of parameters for this model is 332,225. The final version of this encoder is called CNN v4 and the corresponding implementation can be found in the file *siamese_CNN_v4.py* in the project repository.

4.3 BERT-based encoder

For the BERT-based encoder, we used the smallest pre-trained model available, the L2-128H configuration from Tensorflow Hub[3], which has 2 layers and 128 heads. The total number of trainable parameters for this model is 4,419,457. There are two final versions of this model, called BERT v2 and BERT v2.1, which can be found in the *siamese_BERT_v2.py* and *siamese_BERT_v2_1.py* source files, respectively. The modifications made in BERT v2.1 include expanding the last fully-connected dense layer to have 1024 neurons and applying L1 regularization to these weights, based on suggestions made during the poster session.

[2]Tab-separated values

[3]https://tfhub.dev/google/collections/bert/1

5 Results and Analysis

To evaluate the performance of each model and baseline on a test set of 1000 queries, we measured the following metrics:

- accuracy@k: This metric measures how often some of the relevant documents were placed in the top k ranks.
- recall@k: Similar to accuracy, but this metric takes into account all of the relevant documents, even if there are more than one (which was not common in this case).
- MRR@k: The mean reciprocal rank, which is a measure of the quality of a ranked list that takes into account the position of the relevant documents.
- MR@k: The mean rank, which is the average position of the relevant documents in the ranked list.

To evaluate the performance of the top k ranking produced by the concrete model, we calculated the following metrics by comparing the ranks produced by the BM25 algorithm with those produced by the concrete model:

- equal@k: The proportion of instances in the top k ranking where the prediction of the concrete model was equal to that of the BM25 algorithm.
- better@k: The proportion of instances in the top k ranking where the prediction of the concrete model was better than that of the BM25 algorithm.
- worse@k: The proportion of instances in the top k ranking where the prediction of the concrete model was worse than that of the BM25 algorithm.
- MRDB@k: The mean rank difference for instances in the top k ranking where the prediction of the concrete model was better than that of the BM25 algorithm.
- MRDW@k: The mean rank difference for instances in the top k ranking where the prediction of the concrete model was worse than that of the BM25 algorithm.

As shown in Tables 1 and 4, none of the models were able to outperform the baseline algorithm. This means that using this approach could result in the loss of some relevant documents that would not make it into the top 5 or top 10 lists.

However, it is worth noting that, according to metrics in Table 2 and Table 5, for the samples that were included in the top 5 and top 10 lists in the results produced by the BERT v2.1 model, the majority (84.17% and 77.59%, respectively) of passages had an equal or better rank compared to the ranking list produced by the BM25 algorithm alone. The same is also true for the LSTM v6 model, which performed slightly worse than the BERT v2.1 model overall. This suggests that re-ranking only the top k results might still be able to improve the MRR of the BM25 output. We conducted experiments to test this hypothesis, which can be seen in Tables 3 and 6 for the top 5 and top 10 lists, respectively. However, even the best version of the BERT v2.1 model was unable to improve the ranking list. The reason for this is the ratio between the MRDB and MRDW metrics. We observed that when any model makes a mistake, it tends to degrade the rank much more on average, and even a higher rate of positive results is unable to balance this disparity.

There are several possible reasons why the models were unable to improve the ranking list produced by the BM25 algorithm. One possibility is that some of the documents that were indeed relevant were not annotated in the dataset, leading to inaccurate measurements. Another factor to consider is that the models were trained on a classification task, where they had to determine whether a given query and passage were relevant or not. However, when re-ranking the given list of results, the models do not have access to the context of the other documents in the list, which may lead to inaccurate results.

In the future, it may be worth to approach this task as a multi-label classification problem, where each sample consists of a query, 100 passages, and 100 labels indicating the relevance of each passage, as described in (Mitra and Craswell, 2018). A draft of such a model and the dataset have already been implemented, but the exponential increase in computational and spatial complexity has made it impractical to pursue this approach at this time. Instead, we decided to focus on the binary classification approach, which is more computationally feasible.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Bhaskar Mitra and Nick Craswell. 2018. An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126.

Mohamed Trabelsi, Zhiyu Chen, Brian D. Davison, and Jeff Heflin. 2021. Neural ranking models for document retrieval. *Information Retrieval Journal*, 24(6):400–444.

6 Conclusion

In conclusion, our project demonstrated that the BERT-based model is an effective choice for text classification tasks. It outperformed the other models we tested, and came close to the performance of the BM25 algorithm. However, there is still room for improvement and further experimentation. By exploring different techniques, architectures, and hyperparameters, it is possible that we could achieve better results. Some possible avenues for further exploration include using different pretrained embeddings or training them from scratch, using larger versions of the BERT encoder, and using different datasets.

Model	accuracy@5	recall@5	MRR@5	MR@5
BM25	0.7694	0.7597	0.8337	1.5034
CNN v4	0.4354	0.4246	0.6825	2.0955
LSTM v6	0.4749	0.4650	0.7810	1.7617
BERT v2	0.5645	0.5502	0.6629	2.1020
BERT v2.1	0.6403	0.6291	0.7948	1.6716

Table 1: Overall performance of each model on top-5 ranking task given a set of 100 documents

Model	equal@5	better@5	worse@5	MRDB@5	MRDW@5
CNN v4	45.34%	25.00%	29.66%	1.90	2.20
LSTM v6	55.51%	24.04%	20.45%	1.88	2.35
BERT v2	36.11%	26.65%	37.24%	1.71	2.06
BERT v2.1	55.17%	29.00%	15.83%	1.58	2.12

Table 2: Ranking comparison between models and BM25 for top-5

Model	MRR@5	MR@5	MRDB@5	MRDW@5	equal@5	better@5	worse@5
BM25	0.8337	1.5034	-	-	-	-	-
BERT v2.1	0.8211	1.5214	1.65	1.74	62.00%	19.00%	19.00%
LSTM v6	0.7537	1.7350	1.80	1.82	54.51%	16.50%	28.99%

Table 3: Ranking quality measured only on top-5 outputs of the BM25

Model	accuracy@10	recall@10	MRR@10	MR@10
BM25	0.8217	0.8139	0.7894	1.8844
CNN v4	0.5635	0.5512	0.5579	3.3663
LSTM v6	0.5987	0.5879	0.6465	3.0427
BERT v2	0.6787	0.6658	0.5745	3.0157
BERT v2.1	0.7235	0.7122	0.7190	2.3480

Table 4: Overall performance of each model on top-10 ranking task given a set of 100 documents

Model	equal@10	better@10	worse@10	MRDB@10	MRDW@10
CNN v4	35.80%	21.78%	42.42%	2.65	4.02
LSTM v6	44.21%	23.71%	32.09%	2.54	4.32
BERT v2	30.35%	25.00%	44.65%	2.45	3.21
BERT v2.1	49.12 %	28.47%	22.42%	2.27	3.54

Table 5: Ranking comparison between models and BM25 for top-10

Model	MRR@10	MR@10	MRDB@10	MRDW@10	equal@10	better@10	worse@10
BM25	0.7894	1.8844	-	-	-	-	-
BERT v2.1	0.7419	2.100	2.33	2.73	53.12%	21.04%	25.84%
LSTM v6	0.6540	2.5532	2.50	2.94	44.03%	17.92%	38.05%

Table 6: Ranking quality measured only on top-10 outputs of the BM25