NEURAL INVERTED INDEX FOR FAST AND EFFECTIVE INFORMATION RETRIEVAL

Bonifacio Marco Francomano, Luigi Gallo
Sapienza, University of Rome
{francomano.1883955, gallo.1895146}@studenti.uniroma1.it

ABSTRACT

In the field of information retrieval (IR), the efficacy of systems in providing pertinent document rankings in response to user queries is paramount. Traditional IR systems commonly employ an index-then-retrieve pipeline. However, our work aims to introduce another approach, aligning with the recently proposed Differentiable Search Index [1] (DSI) concept, integrating the indexing and retrieval processes into a singular, cohesive Transformer language model, In this work we tried different strategies for training the DSI through an alternative setup for the auto-regressive mode; the main differences with the existent similar models in the literature are the fact that this model has a decreasing probability of using teacher forcing, allowing it to mimic better the behavior of the generation of the docids, and the fact that at inference time, it limits the generation only to existing docids by the use of a special data structure. Our project embarks on constructing a unified model that acts as a sequence-to-sequence architecture, designated as 'f', which adeptly handles a query 'q' and generates in an auto-regressive way, a relevant docid for that query. At inference time we retrieve a ranked list of document identifiers by using a constrained beam-search on the outputs of the model. This model is trained to mimic the functionality of an index constructed on a document corpus, subsequently utilized for retrieving relevant documents. The MS MARCO [2] dataset, supplemented by the Pyserini library, serves as the foundation for our model training and evaluation. To assess the performance of our model, we focus on Mean Average Precision (MAP), Precision@10, and Recall@1000[1].

1 Introduction

Since the task is very challenging and the computational resources are limited, in this work we try to examine several approaches, to understand which are the main ways to explore to get significant results in a more advanced environment of training. For this reason, we have tried different approaches on the pre-processing side, on the data representation side, and on the modeling of how the data is processed (architecture and inference).

2 Dataset Creation

To streamline the creation of datasets necessary for training and testing various models, the preliminary step involves the creation of two dictionaries (*queries* and *documents*). Iterating over all MS MARCO queries (and documents), the 'raw' text is stored in both dictionaries (using query and document IDs as keys), along with a list of relevant documents for each query. During the dictionaries construction, a corpus is also created, containing all the processed[2] 'raw' data. On this corpus, a Word2Vec is then trained to generate embeddings of length `EMBEDDING_SIZE` with a window size of `MAX_TOKENS`. The trained Word2Vec model is subsequently used to create 2 types of embeddings[3] for each query/document: `emb`, obtained as the average of the embeddings of all words, and `first_L_emb`, obtained by concatenating the embeddings of the first `MAX_TOKENS` words. With these two dictionaries, two datasets are then

[1] Recall@1000 is only computed for Siamese networks, as the computational cost of the beam search is quadratic in K.

[2] Converted to lowercase, tokenized, and with stopwords and punctuation removed.

[3] Embeddings are added as fields of the dictionaries.

created to train the Siamese networks, that return data in any desired format (i.e., `ID, raw, emb, first_L_emb`): `QueryDocumentDataset`, which returns (query, document, relevance) triplets, where relevance is 1 if the document is relevant[4] to the query, and 0 otherwise, and `TripletQueryDocumentDataset`, which returns (anchor, positive, negative) triplets, where the anchor is the query, positive is a document relevant to the query, and negative is a random non-relevant document. Two additional datasets are created, still using the queries and documents dictionaries, to train the Seq2Seq model: `DocumentDataset` that returns (encoded document, encoded docid) pairs, and `RetrievalDataset` that returns (encoded query, encoded docid) pairs. Documents and queries encodings are computed using the T5-small tokenizer, and we choose to pick the first `L=32` tokens for representing the documents, and the first `L=9` for representing the queries, carefully managing the case where there are fewer tokens[5]. For the docids we decided to build another vocabulary to reduce the search space: we picked each digit as a token (0-9) and we added 3 special tokens for starting (12), ending (10), and padding (11) the sequence.

3 Baseline Testing for Siamese Networks

To have baselines of comparison for this project, we first modeled the task with another paradigm, that is, trying to learn the relevance between documents and queries directly comparing the contents, without using the docids except in the final phase of retrieval. The kind of models that are presented in the following section, aim to output a score of relevance between a document and a query taken in input. To achieve this, we try to compute such a score in different ways, using the feature vector of each of the two samples (query and document), and passing them in the same network. For this reason, at inference time, we need to iterate over all the documents and pass each of them one by one in the model together with the query in exam. We have to pay for the reduced time of training concerning the DSI model, with a higher time of retrieving.

SiameseNetwork Baseline

The first tested baseline, `SiameseNetwork`, processes the embeddings (`emb`) of queries and documents, outputting the probability of their correlation by simply concatenating the two feature vectors, and passing the final vector into a fully connected layer followed by a sigmoid. Despite achieving high accuracy levels (up to 87%), the model's performance on precision at k metric reveals its inability to effectively discriminate between relevant and random documents, often assigning high similarity scores to irrelevant documents. This is evidenced by the Precision@10 and Recall@1000 scores reported in Table 1, which show that relevant documents do not actually have the highest scores and instead appear in lower-ranking positions.

SiameseTransformer Baseline

The second baseline, `SiameseTransformer`, utilizes the stack of the first `MAX_TOKENS` embeddings (`first_L_emb`) of queries and documents. This model tries to use directly the embeddings obtained by the token of the documents and the queries, to compute attention scores for building the two feature vectors. Similar to the SiameseNetwork, it generates a relevance score between them, this time by the use of Cosine Similarity. However, this model also struggles to distinguish relevant documents from random ones, frequently attributing high scores to both.

SiameseTriplet Baseline

The final baseline, `SiameseTriplet`, was trained using a `TripletMarginLoss`, by providing positive and negative examples corresponding to the queries. This method, which leverages the principles of contrastive learning, showed notable effectiveness over previous baselines. Unlike earlier methods, the contrastive approach in `SiameseTriplet` effectively discerns between similar and dissimilar query-document pairs. This is achieved by optimizing the embeddings in a way that similar pairs are pulled closer while dissimilar ones are pushed apart, creating a distinct margin as imposed by the triplet loss function. This significantly improved the model's ability to generalize from training data to unseen queries as Table 1 shows. Figure 1 depicts the ranked documents for a random test query.

[4] A document is relevant to a given query if it's ID appears in the query's `docids_list` (list of relevant documents for the query).

[5] Since we removed stop-words and other not representative tokens, when the sample has less tokens than L, we padded with 0-padding tokens.

Doc ID	Score	Relevant
5763747	0.9248231649398804	0
775666	0.9046399593353271	0
6702148	0.8990009427070618	0
5347376	0.8880547881126404	1
4019802	0.8878763318061829	0
354963	0.8771447539329529	1
1672345	0.8759815692901611	0
8291243	0.8745550513267517	0
7367184	0.8597170114517212	0
767321	0.8580762147903442	1

Figure 1: in the top 10 retrieved documents, 3 of them are relevant ones.

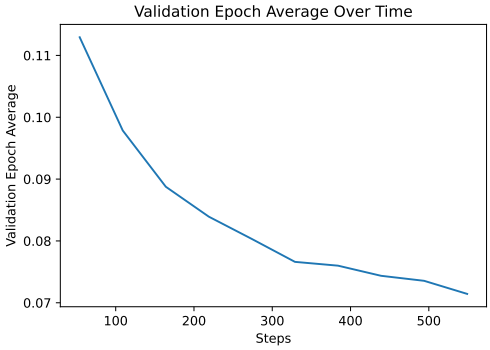

Figure 2: Validation loss during contrastive learning

4 DSI Transformer-Based Model

To reduce the risk of overfitting the documents, the architecture we chose for achieving the task is a very lightweight sequence-to-sequence model: we build two different embedding layers, one for the input sequence and one for the target sequence, since they're obtained by two different tokenizers (as discussed in section 2); then we have 3 encoder-decoder transformer layers and a final fully connected layer. The model outputs a probability vector of size `docid_vocab_size`[6] for each element in the sequence, and each element of the batch. In summary, the output shape is (`seq_len, batch_size, docid_vocab_size`).

Training Setup: Teacher Forcing and Auto-Regressive

Our training setup consists of two main sections: **indexing**, in which the model has to acquire the documents knowledge inside its parameters, so it is trained on the whole corpus without splitting the dataset, to map every document to its docid, and **retrieval**, in which the model must generate, starting only from the special start token, the entire target sequence given a query. To learn the retrieval, we fine-tuned on (query, docid) pairs and we split the dataset as usual in train, validation, and test set. The first attempt for training the model was to try to use simple teacher forcing, i.e. feed the decoder in the training step with the target sequence up to the last token and compute the loss with the target sequence shifted by one. Unfortunately, this approach failed since the model is not able to make inferences generating the target tokens one by one without having the ground truth. So we tried to modify the training step to **use teacher forcing with a certain probability** (initialized to 1) **that decreases every epoch**. When the teacher force is not used, we enter the auto-regressive setup: the model generates one token at a time by choosing the arg max of the prediction along the current token on the dimension "`seq_len`", and at each iteration, the decoder is fed with the last generated token. However, this approach generates a drop in the performances and the model wasn't able to learn. The reason is that every time the model generates a wrong token, the error is propagated along the next predictions, and the model can not look at the all context generated up to that point, for making the next decisions. So, the final idea was to **modify the auto-regressive mode to feed the decoder with all the generated sequences up to the current token.** We decided to not use this modality for the indexing part, to make the memorizing part of the documents simpler, and to use it instead for the retrieval only, by decreasing the probability of teacher forcing at each validation epoch by 0.003.

Hyper parameter optimization

After many attempts, we found out that the best combination for this training was: Embedding size: 120, Number of transformer layers = 3, dropout = 0.2, LR = 0.0005

5 Results

Due to the limits of usage imposed by Google Colab (RAM saturation and GPU usage), we couldn't train for more than 400 epochs for the indexing part and another 400 epochs for the retrieval part. For sure with more powerful computational resources we can achieve higher results, but the plots depicted in Figure 3 and 4 show that the setup we have adopted begins to show promising results. We trained until the RAM was not saturated and, when the model had memorized sufficient knowledge about the documents, we preceded with the fine-tuning. In the fine-tuning phase, we

[6]A restricted vocabulary composed by the 10 digits and 3 special tokens for `sos`, `eos` and `pad` tokens.

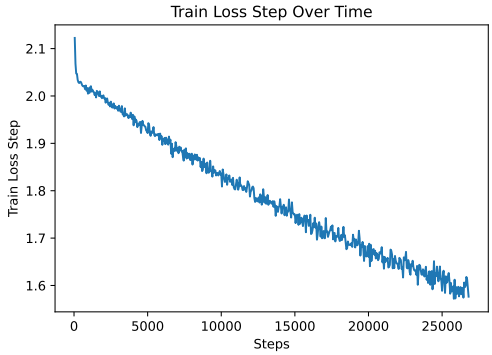

Figure 3: Train Loss during the memorizing train

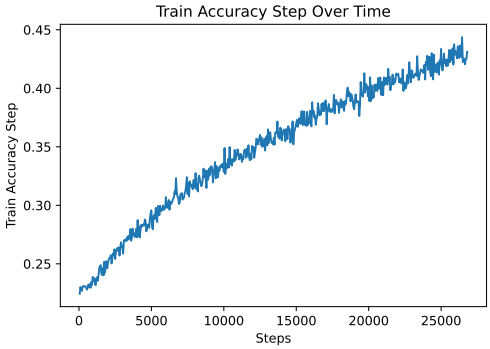

Figure 4: Accuracy during the memorizing train

have an initial high variability in the loss, as depicted in Figure 5, which stabilizes as the training progresses, indicating effective learning and adaptation to the retrieval task. Also, the accuracy, depicted in Figure 6, illustrates a positive trend in the model's ability to correctly associate queries with document identifiers. The validation loss, depicted in Figure 7, initially follows a downward trajectory with a subsequent rise, reflecting the slow fade out of teacher forcing. Contrary to the validation loss, the accuracy on the validation set, depicted in Figure 8, shows a consistent increase. This implies that despite the increase in validation loss, the model's predictions are more often correct, which suggests that the model is increasingly better at identifying the right document identifiers over time.

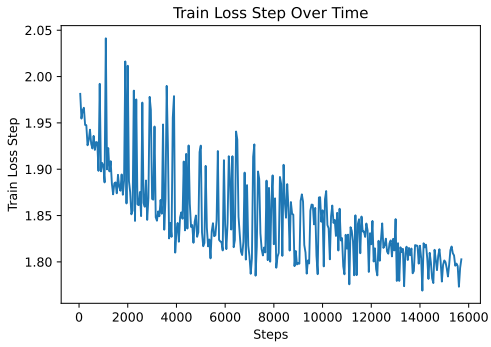

Figure 5: Train loss during retrieval training. It can be seen that the noise due to the reduction of teacher forcing decreases over time

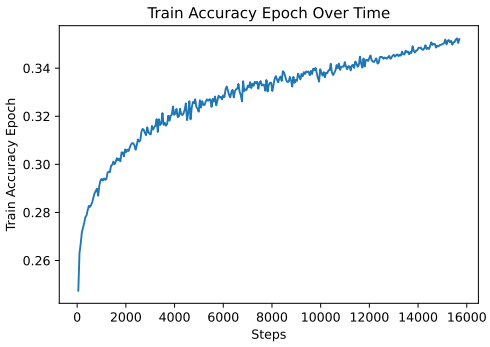

Figure 6: Train accuracy during retrieval training

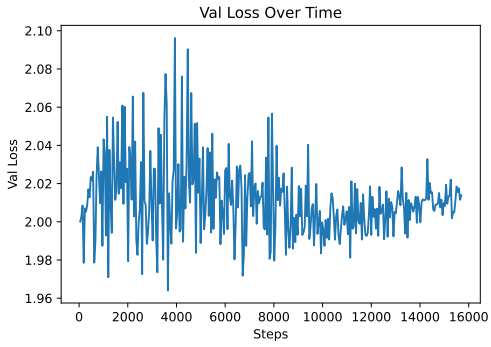

Figure 7: Validation loss during retrieval training starts to decrease and then rise up, because of the reduced capacity of the model and the reduction of teacher forcing

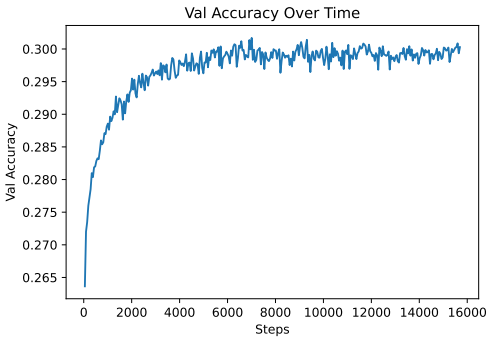

Figure 8: The validation accuracy surprisingly increases during the retrieval training

Metrics and Inference

At inference time, to compute the metrics, we constrained the model to generate only existent docids, by using a special data structure called "**Trie**". This object has the method `get_next_tokens` that takes in input a sequence of tokens (as a list) and returns another sequence of possible next tokens, to form a part of an existent docid. The `top_k_beam_search` function implements a **constrained beam search** algorithm, designed to generate sequences adhering to specific constraints encapsulated by this trie data structure. A list of hypotheses is initialized, each comprising a start-of-sequence token and an initial log probability of 0, and updated in each iteration (up to the maximum sequence length). For each hypothesis, if the sequence concludes with an end-of-sequence token, it is preserved as is, otherwise, the model predicts probabilities for the next possible tokens, constrained by the trie data. This step ensures that only valid extensions of the current sequence are considered. Each hypothesis is then extended with these tokens, updating their probabilities. After exploring all possible extensions, the algorithm selects the top k hypotheses based on their probabilities[7].

Table 1 reports the metrics obtained in the various models tested.

Table 1: Models Training and Validation/Test Scores

Model	**MAP**		**Precision@10**		**Recall@1000**	
	Train	Val	Train	Val	Train	Val
SiameseNetwork	0.060	0.000	0.020	0.000	0.240	0.420
SiameseTransformer	0.000	0.000	0.000	0.000	0.039	0.100
SiameseContrastive	0.302	0.213	0.158	0.158	0.783	0.778
Seq2Seq	0.192	0.190	0.172	0.168	/	/

6 Conclusion

This study aims to analyze the ambitious task of integrating indexing and retrieval processes into a singular cohesive Transformer language model, diverging from conventional bifurcated information retrieval (IR) system designs. Our approach, inspired by the Differentiable Search Index concept, aimed to unify these typically separate stages, enabling a more fluid, fast, and dynamic retrieval process. Through extensive experimentation with the MS MARCO dataset and the application of various training strategies, including the use of auto-regressive and teacher-forcing methods, we have demonstrated the feasibility and potential of this unified model. Our results show an improvement in retrieval effectiveness concerning the classic self-supervised models, as evidenced by the metrics of Mean Average Precision and Precision@10. Moreover, we have demonstrated that the gradual decrease in teacher forcing during training not only allows the network to learn to generate more correct sequences and to exceed the performance of Siamese networks but also to overcome the classic sequence-to-sequence models trained to predict a single missing token. However, challenges remain. The limitations in computational resources restricted the extent of training, suggesting that further improvements could be achieved with more powerful computational setups. Additionally, the gradual phase-out of teacher forcing during training revealed insights into the model's learning process, indicating avenues for future optimization. In conclusion, the potential applications of this unified approach in various domains of IR are vast and promising, opening the way for more efficient, accurate, and user-friendly information retrieval systems.

References

[1] Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. Transformer memory as a differentiable search index, 2022.

[2] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset, 2018.

[7] Probability of a sequence of tokens is computed as a sum of single tokens log probability