MCS Portfolio: Modularized Transformer-based Ranking Framework CSE 575

Joshua O'Callaghan
jdocalla@asu.edu
CIDSE, Arizona State University
Tempe, Arizona

ABSTRACT

Transformer-based ranking models have evolved over the last few years as technology has evolved, improving ranking models in performance and efficiency. Despite these performance updates, the computational costs of these traditional transformer-based ranking models have increased. To further optimize the performance of these ranking models, several transformer-based models were proposed including the EARL framework. The Embed Ahead Rank Later (EARL) framework speeds up traditional rankers by pre-computing representations and keeping online computation shallow. To facilitate ranking, the ranker is separated into three independent tasks, thus enabling much of the computation work to be done offline. We have implemented the EARL framework and measured its performance on a computer with relatively limited computational power. In this report, the effectiveness of the EARL framework along with its accuracy and performance is presented. The advantages over traditional transformer-based models such as BERT as also discussed.

1 INTRODUCTION

1.1 Motivation

Information retrieval technologies such as search engines are ubiquitous in today's world. Modern search engines can retrieve and rank billions of results relevant to users' input in fractions of a second. These search engines are constantly looking at ways of refining their ranking algorithms and frameworks in order to get the most relevant data to users in the shortest amount of time. Thus, it is apparent that a robust yet efficient ranking algorithm is crucial in the operation of these search engines.

1.2 Previous Work

Traditionally, document ranking for a given corpus and query was done using a framework which was a combination of smaller algorithms and components [1] [2]. These components form a larger prediction model that would rank the documents. However, there are problems associated with this approach: each constituent model needs to be trained and tuned separately, and significant supervision is required for them. Therefore, it is desirable to adopt an end-to-end solution that can be trained and deployed efficiently. To tackle this problem, there has been a recent rise in the use of neural networks for ranking the relationship between two entities [3]. The current state-of-the-art models in this field of information retrieval are transformer-based ranking models. Transformers are deep learning models used primarily in the field of natural language processing (NLP). In basic terms, transformers take in some sequence as input and then transform that input into a different sequence representation [4]. Devlin et al. proposed a model that pre-trains the representations from unlabeled documents called Bidirectional Encoder Representations from Transformers (BERT) [5]. However, BERT still experiences trade-off between processing time and ranking performance. Therefore, it is necessary to further optimize the pre-training and ranking strategy to minimize this trade-off and achieve better performance.

1.3 Methods and Results

One possible way to improve the performance is to pre-compute wherever possible so that online relevance judging can be as light as possible. Towards this goal, Luyu Gao, Zhuyun Dai, and Jamie Callan proposed the Embed Ahead Rank Later (EARL) framework [6] that breaks down the attention [7] in typical transformer architecture into three asynchronous tasks. By unraveling the blackbox that is normally seen in transformer-based models into three distinct modules, we can run the modules to pre-compute the query and document representations. This decreases the processing time taken as only one module has to be computed online. With EARL, it is possible to achieve 50 to 120 times speedup on a dedicated graphics processing unit (GPU) compared to BERT counterparts.

1.4 Contributions

Through transformers and attention models, we were able to implement EARL and create the three modules: Document Understanding Module, Query Understanding Module, and Relevance Judging Module. In this report we present our implementation of the EARL model as presented in the paper. We go on to compare our findings of the efficiency and effectiveness of our EARL model implementation with the efficiency and effectiveness of the EARL model produced in the original paper. Furthermore, we evaluated the effectiveness of the EARL model compared to the BERT model.

2 SOLUTION

In this section, we provide an in-depth explanation of the methods and techniques used throughout the project. The models and how they were implemented are discussed in detail. EARL as a framework is defined and explored in this section.

2.1 Notations

Many of the notations used in the original EARL paper are described here for clarity. Attention is written among three matrices: X, Y and Z as $Attention(X, Y, Z)$ [7]. This function is defined as:

$$Attention(X, Y, Z) = softmax(\frac{XY^T}{\sqrt{d_{model}}})Z$$

Similarly, multihead attention among the X, Y, and Z matrices is denoted as $MH(X, Y, Z)$ [7]. This function is defined as:

$$MH(X, Y, Z) = MH(head_1, ..., head_h)W^O$$

$$head_i = Attention(XW_X^i, YW_Y^i, ZW_Z^i)$$

The feed-forward neural network is denoted as $FFN(x)$ and the function is defined as follows [7]:

$$FFN(x) = ReLU(xW1 + b1)W2 + b2$$

In each layer we see a normalization function with input x and this is written as $Norm(x)$.

Using these notations, the EARL framework is explained.

2.2 The EARL Framework

Typical transformers use a blackbox process which takes a concatenated version of the query and document and then applies a string of attention operations on itself [11]. The last layer of self-attention will output the relevance prediction. Unfortunately, because of the blackbox nature of this process, the hidden states have no explicit semantic meanings. This results in slow rankers and the overall design remains difficult to understand in terms of what exactly is happening in our strings of attention.

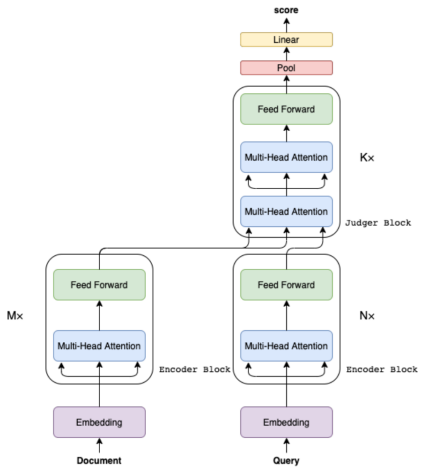

Figure 1

The researchers who proposed the EARL model assume that the blackbox process could be made simpler by decompiling the blackbox process. It is assumed that typical rankers perform document representation and query representation together as it interprets the pair together. The beauty of EARL is that we separate the document and query understanding into two separate pre-processed modules. Not only does this allow us to save time during online processing, but it also gives us some much needed transparency and semantic meanings to the output of our layers.

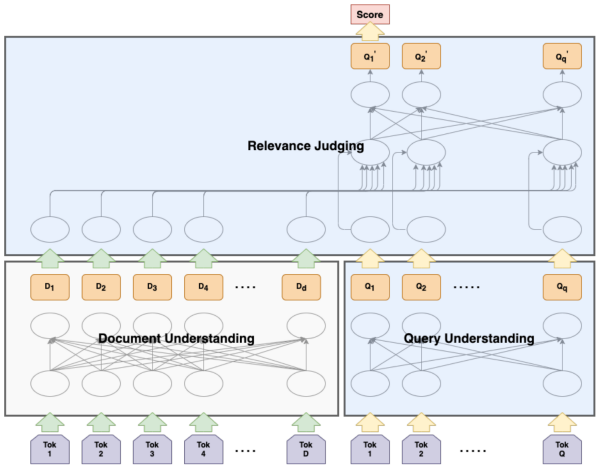

Figure 2

Figure 1 illustrates the architecture of EARL excluding layer normalization [6]. We see the ranking task has been divided into three parts: document understanding, query understanding, and relevance judging. For all documents in a corpus, we can run the document understanding once offline. Similarly, query understanding only needs to be run the first time a given query is seen. Figure 2 is a detailed representation of how attention in EARL works [7]. The document and query modules are the same as a transformer encoder, while the relevance judging module applies a query-to-document cross-attention followed by a query-side self attention.

We can describe our document and query token representations as the following [6]:

$$D = T_{doc}(doc)$$

$$Q = T_{qry}(qry)$$

We can describe the judger block that looks up the representations and model the attention between the two tokens to output a score for the document as the following [6]:

$$score(qry, doc) = T_{judge}(Q, D)$$

2.3 Document Understanding Module

The document understanding module is one of the three modules in this ranker. This module builds document token representations using a Transformer model. These representation tokens are independent of the query. As a result, we can calculate all the tokens for all documents ahead of time due to their query-agnostic nature. These tokens can then be cached, and these cached tokens can be used for the entire ranking process. This means we only need to run the module once for the entire corpus. Furthermore, this module can be run before accepting any queries and performing any ranking, meaning the computation of tokens from this module is done "offline" as they are pre-computed.

In Figure 2, we can see that the document understanding module is the same as an encoder. We embed an input of length d with an embedding function to get an embedding matrix E_{doc} [6].

$$E_{doc} = embed_{doc}(document)$$

We then process each of the embeddings with a series of M transformer blocks, leaving us with H_1, ..., H_M hidden layers. Each layer is further encoded with the previous layer where [6]:

$$H_l^{doc} = Encoder_l^{doc}(H_{l-1}^{doc})$$

$$H_1^{doc} = Encoder_1^{doc}(E_{doc})$$

Each encoder is distinct and takes advantage of the multihead function, feed forward neural network, and normalization function defined earlier [7].

$$A^l = Norm(MH(H_{l-1}^{doc}, H_{l-1}^{doc}, H_{l-1}^{doc}) + H_{l-1}^{doc})$$

$$Encoder_l^{doc}(H_{l-1}^{doc}) = Norm(FFN(A^l) + A^l)$$

The output of the last layer is what we will use for the final document representation denoted as $D = H_M$. D is a series of d token embeddings of size n. These tokens get used in the later relevance judging module.

2.4 Query Understanding Module

The query understanding module is used to process the input query with a transformer encoder. In this module, each query is transformed into a token-level representation. The process for calculating tokens is similar to that of the document understanding module: the query is embedded as input and a token-level representation is output to be used for the relevance judging module. Unlike in the Document Understanding Module, not all queries are given ahead of time. As a result, we will have to run this module more than once. However, similar to the Document Understanding Module, the token representation for a given query will always remain the same. Because of this, once a representation is generated for a given query, this representation can be cached. This way, if this same query is used at a later time, we do not need to rerun the Query Understanding Module and can instead use the cached query token representation.

We obtain these tokens in a $d \cdot n$ matrix E_{qry} [6].

$$E_{qry} = embed_{qry}(query)$$

A series of transformer encoders are used to process the token embeddings into a set of hidden query representations, H_1^{qry}, ...H_N^{qry} [6]:

$$H_l^{qry} = Encoder_l^{qry}(H_{l-1}^{q})$$

$$H_1^{qry} = Encoder_1^{qry}(E_{qry})$$

The output from the last layer of the transformer is used as the final query representation of size $q \cdot n$, containing a sequence of q contextualized token embeddings for a query of length q. The encoders are the same as the functions defined in the previous module but with respect to the query rather than the document.

2.5 Relevance Judging Module

The relevance judging module is the final module. This module takes in the token-level representations from both the query understanding module and document understanding module. It uses a judger block to improve the standard Transformer technique and compute the relevance between query and document.

The judger block, proposed in the original EARL paper [6], only performs cross attention from query to document and self attention on the query tokens. This completely avoids document self attention and document to query cross attention [6].

$$Q_{cross} = Norm(MH(Q, D, D) + Q)$$

$$Q_{self} = Norm(MH(Q_{cross}, Q_{cross}, Q_{cross}) + Q_{cross})$$

As a result, we end up with qd attention pairs from cross attention and q^2 attention pairs from our self attention while avoiding d^2 document attention pairs from the standard Transformer technique. This helps to improve our computation time significantly.

Multiple judger block layers are computed to repeat the process and refine the hidden token representations giving us hidden states H_1, ..., H_K such that [6]:

$$H_l^j = Judger_l^j(H_{l-1}^j, D)$$

$$H_1^j = Judger_1^j(Q, D)$$

Finally, the score is created through a linear projection by pooling the last layer's query token representation [6].

$$score(qry, doc) = w^T Pool(H_K^j)$$

3 RESULTS

3.1 Setup

We use the MS MARCO Dataset to test the effectiveness and efficiency of our algorithm. MS MARCO is a collection of datasets focused on deep learning in search. We are specifically using the Dev Queries subset of this larger MS MARCO dataset. We chose a single specific subset of this larger dataset so that we could get results for our model in a reasonable amount of time. We chose the Dev Queries subset as this subset has a single relevant document with a binary relevance label, allowing for us to easily determine accuracy of our predictions. Following P. Bajaj et al [13], we calculate the mean reciprocal rank of the top 10 ranked items (MRR@10) to determine the accuracy of our model.

We compare the results of the EARL model to the results of the BERT model on this same dataset. EARL is trained using stochastic gradient descent with a batch size of 128. We also use the AdamW optimizer with a learning rate of $3e^{-5}$ and a warmup of 1,000 steps. These parameters were chosen as they are the parameters used in the original EARL paper. Our BERT model is trained using similar parameters. We did not develop our BERT model, but instead used an existing BERT framework that we found already implemented in PyTorch.

Given our two models, we aim to prove that EARL achieves the same effectiveness as BERT, while also showing EARL to be more

efficient than BERT. Furthermore, we attempt to duplicate the effectiveness and efficiency of EARL as seen in the paper while running our implementation of the EARL model with significantly less computational resources.

3.2 Ranking Effectiveness

Model Accuracy MRR @ 10	MS MARCO Passage Ranking	
	Our Model	Paper Model
BERT ranker	0.3180	0.3527
EARL ClsPool	0.3086	0.3442
EARL AvePool	0.3116	0.3415
EARL ClsPool Tied	0.3126	0.3456
EARL AvePool Tied	0.3217	0.3468

Figure 3

Figure 3 shows the ranking accuracy of our implementation as compared to the results shown in the original paper. In all the models, our implementation is around 10% less accurate than the implementation results from the paper. Given our lower computational resource capacity, this makes sense. Because of our limited resources as compared to the original paper, our implementation most likely had less hidden layers than the implementation used in the paper. As a result, it would take more iterations for our model to reach the accuracies seen from the paper.

When comparing the accuracy of our EARL model implementation to our BERT model implementation, we see that our EARL model not only kept up with the same accuracy as the BERT model, but in some cases surpassed the BERT ranker's accuracy. This shows that EARL has maintained the same ranking effectiveness as BERT, showing that EARL can keep up in effectiveness with the leading state-of-the-art information retrieval models.

3.3 Ranking Efficiency

Evaluation Time (s) 1 query / 1k docs	CPU					
	Our Model			Paper Model		
Document Length	EARL	BERT	Speedup	EARL	BERT	Speedup
128	392.34	7171.85	18.28	8.11	161.51	19.91
256	569.43	14685.65	25.79	12.39	349.7	28.22
512	898.65	27309.64	30.39	19.99	698.01	34.92

Figure 4

Figure 4 displays the time taken to run our implementation and the original paper's implementation of BERT and EARL on a CPU. We vary the number of words per document in the corpus in order to see how our model scales in comparison to the BERT model.

When we compare the efficiency of our implementation of the BERT and EARL model to the efficiency of the original paper's implementation of the BERT and EARL model, we see that our BERT model ran 39 to 44 times slower than the paper's implementation of the BERT model. Similarly, our EARL model took 44 to 48 times as much time to run than the paper's implementation of the EARL model. We attribute this to the computational power of our CPU in comparison to the CPU used by the researchers in the original paper. Our model was run on an Intel i5 processor, while the paper's model was run on an Intel Xeon processor. Given the increased amount of cores a Xeon processor has compared to an i5 processor, it is easy to see how the EARL and BERT models implemented in the paper would outperform the EARL and BERT models we implemented.

When we compare the efficiency of our implementation of the EARL framework in relation to the efficiency of our implementation of the BERT model, we see speedups in the range of 18 to 30 times. Despite our lower computational resources, we see that our EARL model still outperforms the BERT model in terms of efficiency. Figure 5, similarly to figure 4, displays the time taken to

Evaluation Time (s) 1 query / 1k docs	GPU					
	Our Model			Paper Model		
Document Length	EARL	BERT	Speedup	EARL	BERT	Speedup
128	32.57	1739.894	53.42013	0.05	2.7	54
256	47.13	3132.731	66.47	0.07	5.78	82.57143
512	58.77	7035.581	119.7138	0.1	13.05	130.5

Figure 5

run our implementation and the original paper's implementation of BERT and EARL. However, this table shows the time taken to execute the models on a GPU. When we compare the efficiency of our implementation of the BERT and EARL models to the efficiency of the original paper's models, we see that our models take hundreds of times longer to run. Similar to our explanation of the CPU results, we believe this discrepancy is due to differences in our GPU hardware's capabilities. The original paper states that their models were run on an Nvidia RTX 2080. Our models were run on an Nvidia GTX 1070. Therefore, our results are inline with what is expected given our more limited computing resources.

When we compare the efficiency of our implementation of the EARL framework in relation to the efficiency of our implementation of the BERT model on GPU, we see speedups in the range of 50-120 times. This once again shows that the EARL framework is more efficient than the BERT framework.

3.4 Results Summary

As shown in the two previous sections, EARL is able to maintain the same ranking effectiveness as BERT while actually increasing the ranking efficiency. Furthermore, for executions on both CPU and GPU, we see that our speedup factor increases as the length of documents increases. This further illustrates that the EARL framework is a more scalable solution than the BERT framework. In conclusion, we see that we were able to successfully implement the EARL model as shown in the original paper, and we were able to show that EARL provides a more scalable and efficient ranking algorithm than BERT, while not sacrificing on the effectiveness of state-of-the-art information retrieval models.

4 PERSONAL CONTRIBUTIONS

Personally, I helped throughout the entirety of the course. Like all other members, I annotated and read through the EARL paper initially to obtain a clear understanding of the strategies used and the goal we were aiming to complete. After splitting up the programming portions, I decided to take lead on the Document Understanding Module and completed it with the help of Pratik's implementation of the Query Understanding Module. I also assisted with programming the driver code and integration of the modules. The majority of the project work was done in the testing phase where we bench-marked our implementation using data from the MS Marco dataset. Along with my other group members i partook in testing and running our code on multiple devices. Finally, I took lead on writing the report which I completed with the rest of my group mates.

As part of some extra work I decided to use servers that I have access to at my job to run this project on to see whether or not our results would improve. My work proved to show an improvement on the efficiency and effectiveness of our code but still fell short of what was written in the paper. This proved two things to me, that our system was a bottleneck to our implementation and our code was not as effective as the paper. The system that I had run it on had much better specs than what was used in the paper. I hypothesize that because of some of the design aspects we skipped in the original implementation, our version suffered as a result.

5 CONCLUSION

Neural ranking proves to be very promising and we are excited to see the path it takes to improve current search engines with higher performance and less computational complexity. Throughout this project, we were able to recreate the implementation of EARL and approximate the results produced in the original paper. We implemented the three separate modules: document understanding, query understanding, and relevance judging. EARL takes advantage of the fact that document understanding and query understanding can be computed once and then reused. This allows us to build our document representations beforehand, thus improving our ranking time. We also implement a new judger transformer block which helps us keep our online computations light and shallow.

5.1 Reflections

Many times while implementing the EARL model, we became stumped because either the implementation described in the original paper was unclear, or the PyTorch modules were not working together properly. These issues were exacerbated by simply not having enough background knowledge in this field and having to find adjacent papers to supplement our knowledge. In the end, we gained a much greater understanding of transformer-based models and produced results that outperformed what we thought we would be able to achieve. This project has taught us about feed forward neural networks, as well as the concept of attention and transformers. We were also able to explore many of the robust features that the python library PyTorch offers in the realm of machine learning. Furthermore, we learned about BERT, EARL, and other implementations of document rankers that have been implemented over the years. We also explored the intricacies and vastness of the MS MARCO dataset. Finally, we learned the importance of document rankers, search engines, and how all the work being done in this field now will greatly impact how data is organized and filtered in the future.

5.2 Skills Gained

Over the course of the semester I learned a great deal about Machine Learning. This course taught me many aspects of Machine Learning such as classifiers and the statistical methods behind them. This project introduced concepts of transformers, neural networks and rankers. I gained a lot of expertise with the PyTorch library which I am excited to use in the future.

5.3 Group Members

My team members were: Xuanli Lin, Pratik Panda, Nishant Ravinuthala, and Cody Schierbeck. I am thankful for such a committed group and without them this project would not have been possible.

REFERENCES

[1] "RANKING CLASSES OF SEARCH ENGINE RESULTS," Proceedings of the International Conference on Knowledge Discovery and Information Retrieval. 2010, doi: 10.5220/0003100902940301.

[2] M. Libes and B. Lent, "Method and system for active ranking of browser search engine results," 8583632, Nov. 12, 2013.

[3] C. dos Santos, C. dos Santos, B. Xiang, and B. Zhou, "Classifying Relations by Ranking with Convolutional Neural Networks," Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2015, doi: 10.3115/v1/p15-1061.

[4] T. Wolf et al., "Transformers: State-of-the-Art Natural Language Processing," Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2020, doi: 10.18653/v1/2020.emnlp-demos.6.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," Oct. 11, 2018.

[6] L. Gao, Z. Dai, and J. Callan, "EARL: Speedup Transformer-based Rankers with Pre-computed Representation," Apr. 28, 2020.

[7] A. Vaswani et al., "Attention Is All You Need," Jun. 12, 2017.

[8] W. Antoun, F. Baly, and H. Hajj, "AraBERT: Transformer-based Model for Arabic Language Understanding," Feb. 28, 2020.

[9] Y. Wang et al., "Transformer-Based Acoustic Modeling for Hybrid Speech Recognition," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2020, doi: 10.1109/icassp40776.2020.9054345.

[10] J. Harer, C. Reale, and P. Chin, "Tree-Transformer: A Transformer-Based Method for Correction of Tree-Structured Data," Aug. 01, 2019.

[11] L. Gao, Z. Dai, and J. Callan, "Modularized Transfomer-based Ranking Framework," Apr. 28, 2020.

[12] Z. Wu, J. Mao, Y. Liu, M. Zhang, and S. Ma, "Investigating Passage-level Relevance and Its Role in Document-level Relevance Judgment," Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. 2019, doi: 10.1145/3331184.3331233.

[13] P. Bajaj et al., "MS MARCO: A Human Generated MAchine Reading COmprehension Dataset," Nov. 28, 2016.