International Journal of Hybrid Innovation Technologies
Vol.1, No.1 (2021), pp.15-32
http://dx.doi.org/10.21742/ijhit.2021.1.1.02

Learning-to-Rank: A New Web Ranking Algorithm using Artificial Neural Network

Falah Al-akashi

Faculty of Engineering, University of Kufa, Najaf, Iraq
falahh.alakaishi@uokufa.edu.iq

Abstract

The objective of this manuscript is to present a neural network algorithm for boosting the relevancy rank of 50000 documents (1000 results for each query) retrieved by our traditional algorithm that ranks a top on the list of algorithms participated in the National Institute of Standards and Technology (NIST). The assessors in NIST explore and evaluate web retrieval technological approaches over a large collection of Web data. Considering the deficiency of current learning to rank approaches lacks the continual learning ability, we introduce a new lifelong learning model that combines web search items with artificial neural networks. Working with the metaphor of our neural network algorithm, each node represents a search item while items potentially learned through observing other items towards optimized communications in the learning environment. Once all items built the relevancy ranks at the end of the iteration, top items would make their decision to declare the relevancy paths for moving to the next iteration. At that end, items would discover some ranking decisions in the available paths as relevant or not. The algorithm integrates unbiased relevance ranks while provides an explicit controller that balances the selection of documents to maximize the marginal relevance in top k results. Theoretical and empirical analysis showed that, with small compromises on long list ranking, our proposed method achieves superior relevancy and efficiency compared with the state-of-the-art algorithms meanwhile the rate of ranking is coupled to the size of the learning environment.

Keywords: *Deep learning, Convergence analysis, Ranking aggregation, Supervised learning, Collaborative filtering*

1. Introduction

Learning-to-Rank is a class of algorithmic techniques that use supervised machine learning to solve evaluation problems in search relevancy ranks. Until a few years ago, the publications in learning algorithms were available in private-scientific-sectors and technical areas. The scale of relevancy and diversity of ranking on the final ranking list astound users, that is, losing any item on the list would cause little or no impact on the list as a whole due to the majority robustness of the related clauses. Information collected from items might be able to make a complex task needed for high levels of collaboration and coordination between items before being viewed to the users. However, the learning environment plays a vital and essential role in the mechanisms of solving some types of learning complex problems [1]. Most of the learning solutions tasks splits-up the process apart while small sections of input data are passed

Article history:
Received (January 31, 2021), Review Result (March 4, 2021), Accepted (May 19, 2021)

for learning in several stages. That means, portions of content examine its local information and make neighborhood actions to update the status of learning in the next stages. This technology becomes rather complex in the distributed learning while researchers did not assume formal solutions for solving such problems [2]. Algorithms that are conducted in the training of ranking techniques might use local information to generate insights and enterprise results. Transferring this technology to web information retrieval models is not an easy task because in traditional ranking models a list of top 10 ranked items reveals the necessary information for Adhoc tasks expected by the user's queries. Enormous and more classes of data become more knowledgeable, and some are available implicitly for ranking mechanisms in the forms of abstracts, in which, each new data added to the list instincts the emulation behaviors of observed items. In neural network algorithms, the hypothesis of observing knowledge is sufficient for unlearned examples to be learned by experienced examples [3]. While the new item monitors the experienced items, it might compute the affection presence of irrelevant items. Yet, the ranking mechanisms can solve complex problems as organized tasks [4]. However, when comparing web learning environments with neural learning environments, there are common interesting features recommended for common attributes. Although tasks of successful deep learning neural networks in real-world problems need to be done fairly, moving ranking models towards the optimal paths for enhancing long-term learning becomes extremely strong. The vast majority of our distributed environment relies on the idea that the network is professionally adapted and the collection of items retrieved from the distributed resources represents the feed-forward networks by accepting vectors of inputs and producing singular output for each network. While the feed-forward networks are adjusted frequently using back-propagation algorithms, the most common network errors are adjusted by the difference between actual outputs and observed outputs when items learn in each network. As a result, Learning-to-Rank has become one of the most crucial fields in machine learning algorithms [5][6][7]. While ranking algorithms play a central portion of the information retrieval problems, web ranking has become one of the most ambiguous algorithms to deal with presenting relevant information tasks on the web [5][8]. While small numbers of potentially relevant items are identified for rendering fast query evaluation, top items with the heuristic models in the literature of accelerating process are utilized in a static quality score and tiered indexes [5]. However, despite machine learning models are computationally expensive, they have been used automatically to improve documents relevancy ranking upon building neural algorithms for significantly increasing the ranking performance. While this process is run extremely slowly on a single machine, fine-grained or coarse-grained parallelisms might be hardly utilized by machine clusters. Thus, efficient implementation of Artificial Neural Network (ANN) models helps to build an acceptable model on a frequently updated dataset while the task could be broken down into smaller ones [9]. Anyways, apart from traditional ranking algorithms that present statistical ranked results, the dynamic ranking paradigms that change ranking results on the fly have been received more and more attention academically. The notation of dynamic Learning-to-Rank is to learn and adapt ranking models based on real-time user feedback in which past user interactions and result distributions might impact the exposure of future results. Despite the impact of user examination biases exposed implicitly, dynamic Learning-to-Rank allows ranking systems to present multiple ranked lists in a single request which makes it possible to explicitly control or balance the exposure of results in different groups for ranking web (i.e., race, gender, etc.).

The rest of this paper is organized as follows: Section 2 reveals some related work. Section 3 defines the metaphor of the learning network. Section 4 elaborates on the data labeling mechanism. Section 5 outlines the features construction technique, while Section 6 describes the learning algorithm. Section 7 shows the experimental evaluation measures, and conclusion remarks are presented in the last Section 8.

2. Related works

It is known that several internet search engines have been built their approaches based on Learning-to-Rank algorithms [9][10]. The functions are different from the underlying machine learning algorithms and training dataset volumes. During the learning process, the algorithm must rank k documents for computing the relevancy cost function. When k is '1', the model is called point-wise and the cost function is computed by assigning a value to the single document for comparing with the relevance value. When k is '2', the model is called pairwise and two documents are specified to determine which one is more relevant. Otherwise, the model is called list-wise whilst the relevancy cost is applied on the whole list of documents. Some examples of point-wise models were proposed by [11][12][13], some examples of pair-wise models were proposed by [12][14], and examples of list-wise models were proposed by [15][16][17]. The RankNet and LambdaMART algorithms [18][19] were other examples of ANN that utilized Learning-to-Rank models to define single output for pairs of documents. A new continual learning idea that combines a multi-agent autonomy learning mechanism with a molecular immune mechanism for ranking is proposed [19]. Balancing the relevance and fairness of information exposure was considered as one of the key problems for modern IR systems [20]. Theoretical and empirical analysis showed that the proposed algorithm was efficiently evaluated by movie, news, and personal data. Even with small compromises on long list fairness, the method could achieve superior efficiency and effectiveness comparing to the state-of-the-art algorithms in both relevance and fairness of top rankings. The models that serve web-scale traffic with billions of training examples were limited expressiveness in the cross-network at learning more predictive feature interactions. Despite significant research progress made, many deep learning models in production still rely on traditional feed-forward neural networks to learn feature crosses inefficiently [21]. The analysis and evaluation of Learning-to-Rank models based on ensembles of regression trees were proposed by [22]. Even the most traditional Information Retrieval (IR) evaluation metrics differ from library to library, objective evaluation and comparison between trained models are difficult tasks. Apart from tree-based models which require extensive features to engineer and handle textual features, neural ranking models could effectively handle sparse features through embedding mechanisms. Recently [23] used a supervised dataset, MSLRWEB30K, where all the ground-truth labels of each training query were used during the optimization process to provide a convenient open-source platform for evaluating and developing Learning-to-Rank models based on deep neural networks. There are numerous algorithms available in today's world regarding Learning-to-Rank problems in web search paradigms that are related to our research, e.g., context-aware search [24] that integrated different context information into a ranking model, the proposed approach improved the ranks of commercial search results which ignored contextual information. A machine learning framework [25] for compositing the presence of multiple relevant verticals assumed freshness into account and sensitive queries in federated searches [26], a gradient boosted decision tree was utilized for learning the training data. Personalized search [10] that mined various data sources in LinkedIn was inferred searchers' intents, e.g., hiring, job seeking, etc. as well as extended the concept of homophile to capture the search results' similarities on many aspects. Thereafter, learning to rank was applied to combine these signals with standard search features. Practical challenges in applying learning to rank methods in the E-Commerce search, including feature representation, obtaining reliable relevance judgments, and optimally exploiting multiple user feedback signals; such as click rates, add-to-cart ratios, order rates, and revenue [27]. Various techniques were proposed in the area of learning to rank, for instance, Support Vector Machines (SVM) [28][29], Decision Trees [12], Artificial Neural Networks [13], Boosting [18], Reinforcement Learning [30], etc. Despite the breakthrough in the neural networks, many approaches [21][22][23][31] built upon neural networks which were referred to as neural ranking models.

However, our approach is highly complementary to the aforementioned algorithms. Yet, to the best of our experience, we are committed to building an in-depth comparable Learning-to-Rank algorithm based on Web track volume across several benchmark datasets.

3. Metaphor of learning networks

Practical algorithms often learn more about data that is refined over time. Data in collaborative learning is given in a matrix, in which, rows correspond to users instead of domains whereas columns correspond to objects. Learning-to-Rank algorithms have been used to enhance the ranks of items and produce ranks similar to that available in the training data. In our perspective, features and local information play the largest source of learning algorithms and consequently incorporate for improving the previous algorithms [32]. Local information and features collection help the network to converge their objects to the optimal solution under the condition $0 < q < 1$. The collection of networks could converge to the isomorphic edges in weight space using locally stored information. In other words, given a similar input vector, each network could produce similar output for each vector if converged by local information. Assume a set of m items (1, 2..., k..., m), where m ∀ k ∈ learned items includes an optimal object 'Ʀ' while other items under learning denoted by '∂', m is the total number of items in the list, $\pi_{ij}\ (t)$ is the amount of information at rank e_{ij} (i, j ∈ m) due to some items at time instant t. If Ʀ anticipated in the list, the entire list would converge to reproduce Ʀ behavior. The player Ʀ produced constant examples data that create stabilizers focused on the learning algorithms to follow. We began by formalized the concept list of items that responds to its local stimulus. At that end, the proposed list represents a collection of items randomly weighted the represented items under learning. Objects tend to persist in learning when they faced a manageable challenge in a continuous learning process while others tend to fix the output patterns. Deep learning models are often performed by forwarding and backward propagation in ANN. Systematically thought, the backward propagation algorithm trains weights in a multilayer feed-forward neural network. As such, forward propagation (fprop) corresponds to recalculate the final score based on the new parameter values, whereas backward propagation (backprop) corresponds to recalculate the parameters. We committed to making the gain function into a sigmoid logistic function while the back-propagation trains each item. For back-propagation to work properly, we need to compute the desired output of errors and to back-propagate through the gained network. While this would normally come from a predefined set of exemplars, we defined the desired output based on the outputs of other items on the unranked list. A set of items for a given network used to compute the desired output is called network neighborhood. At that end, we could define the desired output of a given network to represent the average output vector of all items in the network neighborhood. Averaging output over several networks might reduce the convergence time and this would resist the desired outputs to random variations. Finally, we define two measures of convergence in the collection of networks using the mean squared error. If there are p networks in the collection and $y_j^v(n)$ was the output vector v in the network j at time instant t, then

$$A_j^v(t) = \frac{1}{p-1} \sum_{\substack{j=1 \\ j\ \ i}}^{p} y_j^v(t) \tag{1}$$

$$Convergence_j(t) = \sqrt{\sum_{i \in Network} (y_j(t) - A_i(t))^2} \tag{2}$$

The Convergence represents the difference of squares output in the root-mean-square deviation error from the average output; it is useful to measure how far the network is from others. This measure differs from the standard deviation and does not divide based on the rank-size regularity, and it is useful to see the total errors rather than the average error. When Ɽ is not available in the collection, we measure how far the collection is from learning behavior. If the desired output of Ɽ is d^v, then:

$$Error(t) = \sqrt{\sum_i (d_j(t) - A_i(t))^2} \tag{3}$$

This measure shows how far the items consensus from the desired output; and essentially, the convergence measuring has seen by Ɽ. This measure will be used as the most useful and effective for computing the convergence of items in the list. Eventually, the speed of converges increases exponentially regarding the number of items on the list that shares the same operations or functions.

4. Data labeling

Currently, there are two platforms to build training data: human judgments and exploring searching-log data. In the first one, that we used, a set of queries is randomly selected from the query logs of the search algorithm. Assuming that multiple search systems were available, submit queries to the search algorithms would select all top-ranked documents. At that end, each query would be associated with multiple documents. Human judges would then be requested to make relevant judgments on all query-document pairs. Relevance judgments are often measured at five scores: High Relevant (HR), Relevant (R), Good (G), Fair (F), and irrelevant (I). The human judgment makes relevancy ranks from the viewpoints of average users; for instance, if a query is 'Apple' and the webpage is "apple.com", then the score is 'HR'; whereas, the Wikipedia topic about 'Apple' is 'R', and so on. Ranks represent a relevancy assignment to the query-document pairs. Relevancy judgment on a query-document pair could be performed by a judicial process and then conducted by a majority selecting process. Learning-to-Rank benchmark datasets have also been presented by [33]. In the second platform, data stored in query logs of web search engines could be used to add valuable insight into understanding information-searching pairs, but some limitations had been faced in this method. First, certain types of data might not be available in the searching log as users' identities became the most common example. The IP address typically represents the "user" in a transaction log. Since more than one user might use a computer, the IP address is an imprecise representation of the user. Search engines addressed this somewhat limitation by using cookies. Second, there is no way to collect demographic data when using query logs in a naturalistic setting. Third, the query log does not record the searchers' goals, the searchers' motivations, or other qualitative aspects. Query logs also do not record the underlying situational, cognitive, or affective elements in the searching process. However, while crowdsourcing is often useful for obtaining relevance judgments for Web search, it does not work as well for E-Commerce search due to difficulty in eliciting sufficiently fine-grained relevance judgments [21].

5. Features construction

In machine learning, features play a vital element in the algorithms. Learning effective feature crosses is the key behind building recommender systems. However, the sparse and large feature space requires exhaustive search to identify effective crosses. The Deep & Cross Network (DCN) was proposed to automatically and efficiently learn bounded-degree predictive feature interactions. Models that served web-scale traffic with billions of training examples, DCN showed limited expressiveness in its cross-network at learning more predictive feature interactions. Although significant research progress made, many deep learning models in production still rely on traditional feed-forward neural networks to learn feature crosses inefficiently. In light of the pros/cons of DCN and existing feature interaction learning approaches, an improved framework DCN-V2 was proposed by [34] to make DCN more practical in large-scale industrial settings.

By featuring, the ranking model f (q, d) is defined as f (x) where x is a feature vector for q and d. Features in objective learning are used for combining multiple ranking metrics formulated by pairs and often represented by numerical vectors or feature vectors [35][36]. Such representation is usually named bag-of-features and analogous to the Bag of Words (BoW) and vector space models that are used in information processing for documents representation. Elements in vectors are named features and divided into three types:

(a) Static features: Such features depend on the documents only; for instance, PageRank, document's length, etc. Such features could be pre-computed during the offline ranking mode. They used to compute document static rank to speed up the evaluation processor.

(b) Dynamic features: Such features depend on the contents of documents and the query string, such as Term Frequency (TF) and Term Frequency Inverse Document Frequency (TF-IDF) scores or non-machine learning functions.

(c) Query features: They depend on the query keywords exclusively, e.g., the number of terms in a query string.

Language modeling scores in document zones, e.g. title, body, anchors text, and Uniform Resource Locator (URL) are examples of other features. A feature vector $X_{i,j}= \acute{\emptyset}\ (q_i,\ d_{i,j})$ is built from each query-document pair $(q_i,\ d_{i,j})$, i=1, 2, 3, …, m; j=1, 2, 3, …, n_i, where $\acute{\emptyset}$ denotes the feature functions that match the functions of query and document. Assuming $X_i=\{X_{i,1}, X_{i,2}, X_{i,3}, \ldots, X_{i,ni}\}$, we represent the training dataset as S=$\{X_{i,}\ Y_{i,}\}$ i=1,..,m. The goal is to train the local ranking function f(q, d) = f(x) that assigns ranks to textual and non-textual document-query pair features (q, d), or equivalent to feature vector x.

The technique of selecting relevant features, usually so-called feature engineering, is very important in web ranking. We extracted the relevant features from the distributed resources to represent the components in the learning algorithm. In web search, PageRank has been used widely for evaluating features because it connects differing viewpoints and thoughts in a single place. It works by counting the number and quality of links to a page to determine the rough estimation of how important the website is. The underlying assumption is that more important websites are likely received more links from other websites. However, other features are also utilized in web search, e.g., IDF, local rank, global rank, or query occurrences, alongside title, anchor, URL, and body. Such features impacted highly in item relevancy weights. We computed the cosine similarity measure between the query terms and Metadata terms. We assumed that the terms on snippets are more scattered over the document body than on the titles and URLs, in which, we assigned 0.3, 0.2, and 0.1 as a normalized value to URLs, titles, and snippets, respectively. While global rank and page rank play the major features for moving the relevant pages to the top on the resulting list [15][28], they extracted from an authoritative website e.g., Alexa.com[2]. The inverse document frequency was computed by the number of items in a lookup field on the document involves similar query terms, as follows:

$$IDF_k = 1 + log_2 \frac{n}{d_k} \tag{4}$$

where d_k denotes the document frequency at term k, n denotes the number of documents in the collection.

In terms of local rank, we assume that at most 10 items were retrieved from each resource, and thus, the sequences of items were ranked from "0" to "1". During our training session, the cost function $r(d_i)(1 - o1(d_i, d_j))^2$ is applied as $r(d_i) - r(d_j) = 1$, where r(d) is a relevancy label or document d.

6. Learning principle

In this section, we will demonstrate two important aspects: (1) what would happen if the algorithms were not pre-programmed but rather learned? (2) Could the imitation of items in the ranking list be sufficient to each other of a new item? Our experiments were centralized under these issues, thereafter, untrained items were tested to learn a complex behavior using imitation of other items in the neighborhood. Two learning algorithms are discussed in this section as shown in [Figure 1] and [Figure 2]. The proposed network involves three hidden layers with a single output, represented as:

$$S = f(x;\theta) = f(\textstyle\sum_j w_j . f_j\ (\sum_k w_{jk} x_k + b_{jk}) + b) \tag{5}$$

where x_k is the k-th of input x; w_{jk}, b_{jk}, and x_k are the weight, offset, and activation function of the first layer, respectively; w_j, b, and f are the weight, offset, and activation parameters of the second layer, respectively; S is the final output; and θ is the feature vector. The activation function is a sigmoid function (nonlinear function). A sigmoid function is a squashing or S-shaped function, which limits the output to a range between 0 and 1. It is a bounded, differentiable, and real function defined for all real-input values and has a non-negative derivative at each point which is exactly one inflection point.

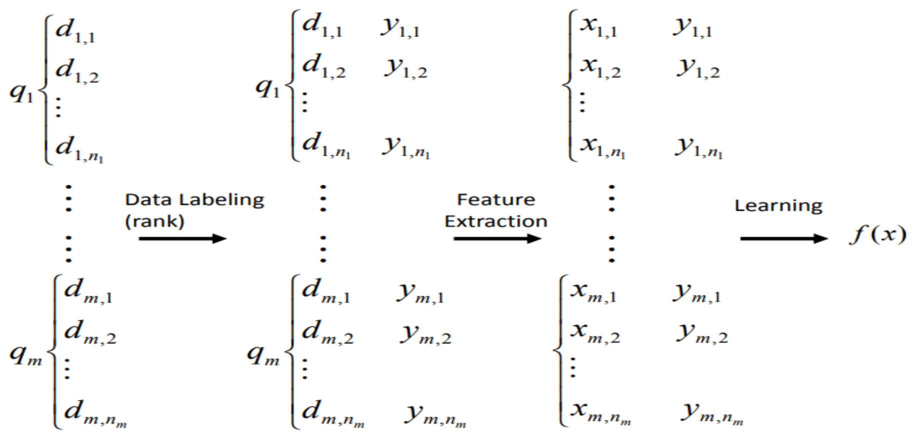

Figure 1. Query to the documents training process

[2] http://www.alexa.com

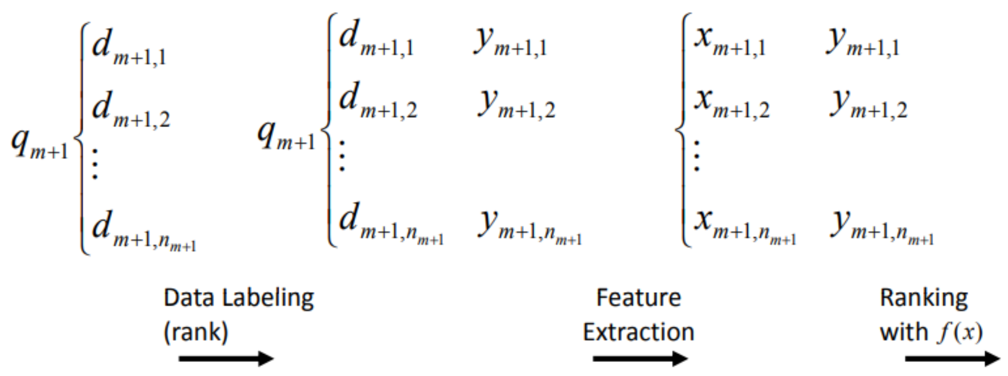

Figure 2. Query to documents testing process

6.1. Objects learning

At every time instant t, assuming the input is a random vector of length n, each item is evenly distributed in a weighted variable, and the threshold value is 0.5, the output is defined as a final vector over the space of features, as input = $(x_1, x_2, \dots, x_n)$ where $x_i \in \{0, 1\}$

$$f(q, d) = f(x, \ell) = f\left(\textstyle\sum_j W_j \cdot f_j \left(\sum_k W_{jk} X_k + b_j\right) + b_k\right) \qquad (6)$$

where X_k is the k-th element of input x, W_{jk}, and b_{jk}; f_j is the weight, offset, and activation function of the first layer; W_j, b, and f are the weight, offset, and activation function of the second layer; and L is the final output and ℓ is the parameter vector.

The activation function is sigmoid or non-linear. The neighborhood of the network was tested statically as a bi-connected graph (for example, people sitting around the table where the two people on either side of a person are that person's neighborhood) and dynamically (for example, any randomly selected pair of people at that table). However, dynamic testing creates a neighborhood with some interesting effects on the rate of convergence in the resulting list [37]. As with any updating scheme, one could implement it synchronously or asynchronously. While asynchronous implementation is much easier to code through iteration every network, in turn, synchronous implementation is much more representative in real-world scenarios, and this might be seen counterintuitive paradigm. Synchronicity means each item did not have access to the updates made by other items. Regardless of whether the actions took place at identical times or not, if each independent only has its available information, it is essentially updated synchronously. To achieve this, it first computes a dynamic nearest neighbor solution for each network; then, generates the average neighbor outputs overall data in epochs. Every epoch is randomized suitably at each input-output vector pair. The back-propagation algorithm requires two parameters to function that aspect, the learning rate and the momentum constant. Due to numerical instabilities and implementation issues, the momentum constant was left at zero and compensated by taking a small learning rate of η=0.3. To prevent saturation problems, the network would cut back the desired outputs from (0 and 1) to (0.005 and 0.995) by a well-known heuristic.

Using interaction data to improve relevance for the user's next query is an important task to transform the framework of learning to rank to re-rank items or objects for matching a user model. Such models are often accumulated over time based on the user's browsing behavior. Interaction between data is important to map each session in the learning models, then novel features were estimated. Extensive experiments on test collections from the TREC session track showed a statistically significant improvement over learning algorithms [38].

6.2. Objects ranking

In this section, we will use the input and output as a distinctive meaningful collection to learn a particular behavior. Using the same network template, we taught one to three networks using XOR functions and then placed them as trained items in the collection of random networks. The ratio of Ʀ was changed the object ∂ from 1:9 to 1:3, and the size of an object ∂ in the neighborhood from 9 to 3. This makes each object Ʀ to see other objects in the neighborhood. In most cases, the collection converged to correct solutions except when very few Ʀ is available meanwhile the neighborhood is too large. This is due to the reason that the output vector of object Ʀ did not change the average neighborhood significantly enough for back-propagation to move fast in the convergence direction. When the ratio of Ʀ in ∂ is high, a large neighborhood size would likely have a large fraction of item Ʀ in item ∂, hence, it is likely very well converged. Similarly, if the ratio of item Ʀ in ∂ is small and the neighborhood is small too, the network would converge well. This is because item Ʀ was part of an observed neighborhood and has a greater effect on the output vector and consequently produced a back-propagation of large error gradient sufficient to move down [Figure 3].

The construction of our network and the successful convergence of ranking items in an XOR function led to generalizing the attempt to model any arbitrary Boolean function, assume values from a set of two elements, usually {0,1} of the Boolean domain, $f\colon \{0,1\}^k \rightarrow \{0,1\}$ where k is a non-negative integer. Though this may not seem a large improvement. The ability to model any such function implies this list could learn to compute any function towards sufficient accuracy as Boolean logic is isomorphic to digital computers. We choose to simulate the four input Boolean function $(a \Leftrightarrow b) \wedge (c \vee d)$, but there is no particular significance to this function. Sixteen input vectors were selected randomly for each output while other functions could work as well. To compromise a larger input size, we expanded the network template to include two hidden layers with four neurons in each. The second layer was added in this assumption because the conjunction of two terms might require one layer to compute the terms and another layer to compute the conjunction. This assumption might not available instantly but they converged and were discovered later. The inputs and output were chosen as binary variables with the same displacement from 0 to 1, as before. The learning rate and the momentum constant involve the same value as others. The experiments run two times using Ʀ - ∂ ratio 3:6 and the dynamic neighborhood of size 3. The experiments took approximately 16000 epochs to converge, we expect these larger networks would converge but likely took much longer, nearly ten times as long. One epoch means that each sample in the training dataset had an opportunity to update the internal model parameters. Assuming we have a dataset with 200 samples (rows of data) and an update size of 5 and 1,000 epochs. This means that one epoch will involve 40 iterations (updates) to the model and a total of 40,000 updates cycled in 40,000 iterations during the entire training process. Also, we observed the XOR networks did not converge and the network would correctly classify most of the inputs (i.e., present outputs approximately 0.005 to 0.995) but was not classified outputs in 0.25 or 0.75. If the network output is used in a discrete simulation, the limitation of values presents the correct answers. Results were not limited strictly due to uncertain answers which means the possible uncertainty in actual group learning could not be reflected and therefore could be useful as opposed to a source of error.

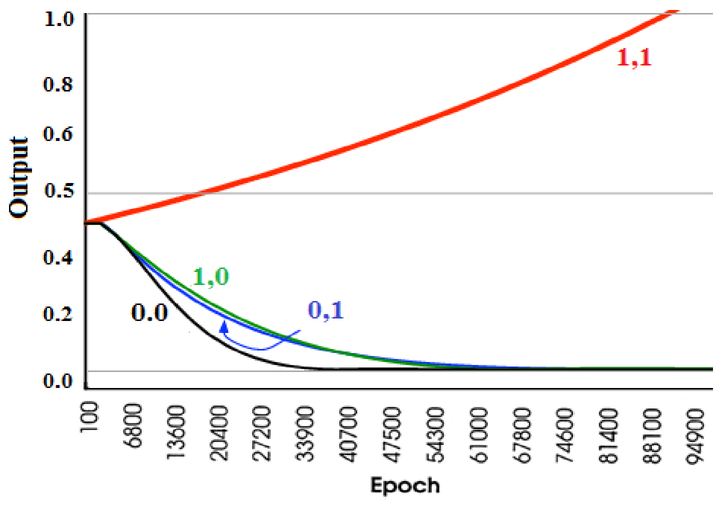

Figure 3. ∂ -learning convergence using an XOR function

6.3. Risk minimization

In a typical scenario, measuring the loss function of ranking results for a given query was proposed f with a loss function F(f(x), y^*)), which was a common sensitive rank. Thereafter, the aim of the Learning-to-Rank algorithm is to learn the optimal ranking function over a hypothesis space S that might minimize the expected risks, as defined below:

$$Min\ F(f)_{f\in S} = Min_{f\in S} \int_{x.y} F(f(x), y^*) dP(x, y^*) \tag{7}$$

Because F(f) is obvious to directly optimize the function and the joint distribution is undefined, the risk minimization approximates the expected risk, which is defined as follows:

$$Min\ F(f; L)_{f\in S} = Min_{f\in S} \frac{1}{n} \sum_{j=1}^{n} F\left(f(x_j), y_j^*\right) \tag{8}$$

Mostly, the Learning-to-Rank algorithms of such kind differ in how they define the surrogate loss function, which consequently classified the algorithms in three categories: Point-wise, Pair-wise, and List-wise, as mentioned.

7. Experimental evaluation

The information retrieval evaluation campaigns provide sets of queries and relevance judgments of expected solutions for proposed queries. Learning approaches in web information retrieval are typically evaluated for two tasks, diversity and Adhoc. Although diversity retrieval is similar to Adhoc retrieval it applies different judging processes and evaluation metrics. The goal of the diversity task is to return a ranked list of pages that represent a complete coverage for a query string while avoiding any duplication in the final resulting list. Performance evaluation in the ranking algorithm is disbursed by comparison between the ranking list outputs proposed by a model and also a ranking list is given as ground truth. The Adhoc task documents are judged concerning the topic as a whole. The goal of an Adhoc task is to return a ranking of the documents in the collection in order of decreasing the probability of relevancy ranking. The probability of relevance of a document is considered independent of other documents that appear before it in the results list. For the Adhoc task, documents are judged based on the description fields. For the diversity task, relevance is judged separately for each subtopic.

7.1. Measuring metrics

Analysis measures are widely utilized in information retrieval and other alternative fields, e.g., Normalized Discounted Cumulative Gain (NDCG), Discounted Cumulative Gain (DCG) [30], Mean Average Precision (MAP) [31], and Mean Reciprocal Rank (MRR). DCG measures documents ranking quality which often uses to measure the effectiveness of the algorithm. DCG measures the usefulness, or gain, of a document based on its position in the result list. nDCG produces the maximum possible DCG through position p. MAP is the average of AP. In some context, we compute AP for each class and average them. In the retrieval system, when a system has returned a ranked list of top 20 items, it is considered most relevant to query. RR is 1 if a relevant document was retrieved at rank 1, otherwise, it is 0.5 if it was retrieved at rank 2, and so on. Given a query Q_i and a set of documents D_i, suppose that πi is a ranking list on D_i, and Y_i is a set of scores on D_i. DCG measures how the relevance of the ranking list at that score. More contrast, DCG at position k is defined as:

$$DCG(k) = \sum_{j:\pi i(j) \leq k} G(j).\, D(\pi i(j)) \tag{9}$$

where G denotes a gain function, D denotes a position of discount function, and πi(j) denotes the position of D_{ij} in πi.

Thus, the result is seized the highest k positions on the ranking list πi. DCG represents the accumulative gain of accessing data from position '1' to position k with discounts on positions. NDCG at position k for q_i is defined as follows:

$$NDCG(k) = DCG^{-1}_{max(k)} \sum_{j:\pi i(j) \leq k} G(j) D(\pi i(j)) \tag{10}$$

where $DCG^{-1}_{\mathrm{max(k)}}$ represents the normalizing value chosen an ideal ranking value of NDCG score that positioned k at '1'.

In an excellent ranking, the documents with higher ranks are hierarchically higher and their multiple excellent rankings for a query and documents. The gain function is often outlined as an associated mathematical function of grade. Satisfaction of accessing data exponentially increases once the rank of relevancy increases too. For example,

$$G(j) = 2^{yi,j} - 1 \tag{11}$$

where yi,j denotes the rank of $d_{\mathrm{i,j}}$ in list π_{i} .

The discount function is often referred to as a logarithmic function of position. Satisfaction of accessing data logarithmically decreases once the position of data accessing increases.

$$D(\pi_i(j)) = \frac{1}{log_2(1+\pi_i(j))} \tag{12}$$

where $\pi_i(j)$ denotes the position of $d_{\mathrm{i,j}}$ in ranking list π_i.
Hence, the DCG and NDCG metrics at position k for q_i are defined as:

$$DCG(k) = \sum_{j:\pi i(j) \leq k} \frac{2^{yi,j} - 1}{log_2(1+\pi_i(j))} \tag{13}$$

$$NDCG(k) = DCG^{-1}_{max(k)}\ DCG\ (K) \tag{14}$$

The DCG and NDCG are additionally averaged over queries (i = 1, ⋯, m), NDCG has an impact on giving high scores to the ranking lists of relevant documents. In the optimal ranking,

NDCG score at every position is almost '1'; otherwise, it is imperfect rankings. MAP is another metric that is also widely utilized in information retrieval. In MAP, the score of relevancies is computed at two levels, ‘1’ and ‘0’. Given query q_i, associated documents D_i, ranking list πi on D_i, and labels y_i on D_i, the Average Precision (AP) of q_i is computed as:

$$AP = \frac{\sum_{j=1}^{ni} P(j) \,.\, y_{i,j}}{\sum_{j=1}^{ni} y_{i,j}} \tag{15}$$

where πi(j) denotes the position of $d_{i,j}$ in πi. P (j) represents the precision until reaching the position $d_{i,j}$ for q_i.

Labels are either '1' or '0', and Average Precision represents average precision overall positions of documents with label '1' for query q_i. Average precision scores are additionally averaged over queries to become the Mean Average Precision (MAP). However, the relevance and primary effectiveness measures for two query tasks are defined by calculating the graded precision of the top ten items [41], or the rank at k ‘P@k’. Documents are ranked either: Nav. (navigational), Key (top relevance), HRel (highly relevant), Rel (minimally relevant) or Non (non-relevant). The relevancy of each web resource is determined by calculating the graded precision of the documents list that belongs to similar resources. It takes the graded relevance levels of the top ten documents into account. Similarly, topic relevancy for a given query is defined by the best-performing search in the index. The final evaluation of binary relevance rank is determined by a threshold, by which, the results with a minimum graded precision, e.g., 0.5, were considered relevant. The threshold assumption was chosen based on data analysis, and for most queries, there were only a small set of relevant items. If no item exceeds the threshold for a given query, the top items with the maximum relevancy features are considered relevant.

7.2. Loss functions

In information retrieval, the true loss function is often ranged between ‘0’ and ‘1’. In ranking, there are different methods to outline the true loss function. The true loss functions might be those used by NDCG and MAP. Experimentally, we have

$$L(F(x), y) = 1 - NDCG \tag{16}$$

and

$$L(F(x), y) = 1 - MAP \tag{17}$$

Assuming a permutation value of π is defined by F (x), hence NDCG for *n* items is showed as follows:

$$NDCG = \frac{1}{G_{max}} \sum_{j:\pi(j) \leq n} G(j) D(\pi(j)) \tag{18}$$

$$G(j) = 2^{yj} - 1 \ , D(\pi(j)) = \frac{1}{log_2(1+\pi(j))} \tag{19}$$

where y_i denotes the score of items *i*, π(i) denotes the score of objects *i* in π, *G* denotes the gain function, *D* denotes the position of discount function, and *Gmax* denotes the normalizing parameter.

Assuming a permutation value of π is defined by F(x), hence MAP for *n* items is defined as follows:

$$MAP = \frac{\sum_{j=1}^{n} P(j) \,.\, y_j}{\sum_{j=1}^{n} y_j} \tag{20}$$

where y_j denotes the score of item j at 1 or 0, π(j) denotes the score of object j at π, and p(i) is the precision until reaching the score of item j, defined by:

$$P(i) = \frac{\sum_{j:\pi(j) \leq \pi(i)} y_j}{\pi(i)} \tag{21}$$

The true loss functions in NDCG and MAP are not continuous, they are based on sorting F(x).

7.3. Experimental results

Effective precisions for some queries were registered but for few queries, the precisions were low because the relevancy results were far away from the informational queries and near to the navigational queries which means they selected only from specific resources. In few cases, the retrieval documents were deemed relevant based on our experiments but they were not if compared with the user's relevancy judgments. Because users have different viewpoints at different times, it is difficult to recall all relevant documents for involving all users' needs in one relevancy judgment. Also, we had a few irrelevant precisions, in some cases, since the queries were categorized as ambiguous. Evaluation experts at the Text Retrieval Conference (TREC) and NIST had assessed our system using 50 test queries, and the overall graded precision was '0.405' as the best accuracy compared with other algorithms. Figure 4 and Figure 5 show the detailed comparison between approaches using accuracy at P(20) and nDCG(20), respectively. The best approach proposed by the University of Glasgow "uogTr" was reformulated the user's initial query with a Terrier data-driven learning model which was a framework for the fast computation of document features. It is used with state-of-the-art learning to rank logistic regression algorithm based on gradient-boosted regression trees. In a realistic setting, the predictive model did not capture the complex interactions among the variables in data. In logistic regression, the interaction could be included through various degrees of interaction terms but the uogTr approach was normally led to computational challenges in a model estimation poor fit. Contrary to our proposed method, logistic regression did not provide a technique focusing the computations on the smallest subset of variables like decision variables to the target variable. Our previous approach "DFalah" was ranked third, it used relations and connections between terms' impacts in documents' titles and its contents. The "ICTNET" ranking algorithm used the Learning-to-Rank platform to combine multiple features but the performance was poor due to the low quality of training data that did not fit the proposed algorithm. The QUT_Para algorithm proposed word associations in natural language processing known as syntagmatic and paradigmatic associations (words in first and second-order). Our proposed approach 'SAMA' brings an empirical improvement in overall performance compared with other approaches that used a corpus of 50 million documents including all resources and testing queries. Despite the related models are a bit old but comparisons between different approaches in information retrieval must have a similar dataset and testing queries. However, experimental results have shown that the ranking methods used neural networks are statistically outperformed than the ranking methods used SVM and Naive Bayes, for two reasons: (1) it is more often to assume the probability of a phrase being a key-phrase in a relative sense than in an absolute sense, (2) features for specifying whether a phrase is a key-phrase are also relative.

However, to show how the setting of activation function affects the performance of different Learning-to-Rank methods by the deep neural networks, we apply the same training framework for all the methods. Specifically, we used a simple 3-layer feed-forward neural network, where the size of the hidden layer is set as 100. Each method is evaluated with different activation functions, we report its best performance and the corresponding activation function in Table 1. We note that Lambda Rank achieved the best performance if compared with similar work, but our proposal achieves better performance in all nDCG ranks concerning the number of layers of the scoring function from 2 to 20. Furthermore, it is noted that the performance values of both SAMA and Lambda Rank fluctuate when changing the number of hidden layers rather than a proportional improvement. One possible explanation is that: as the number of hidden layers increases, the ability to approximate more complex ranking functions (i.e., the model capacity) also increases. However, too many hidden layers might result in overfitting. To summarize, the factors, such as different activation functions and the number of layers, greatly affect the performance of a neural Learning-to-Rank method. Careful examinations of these factors are highly recommended in experimental comparisons of different Learning-to-Rank methods.

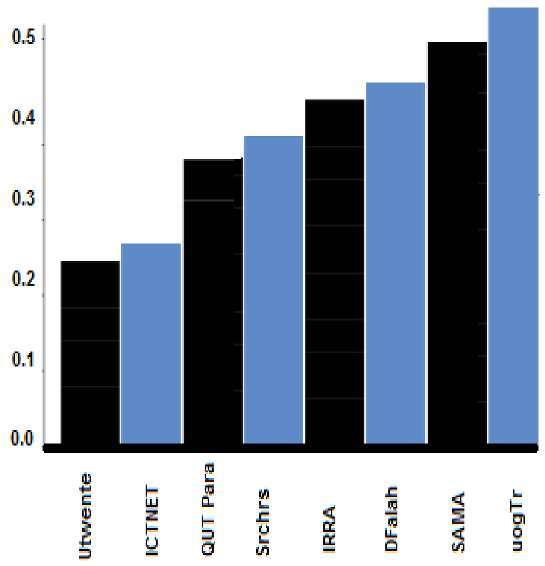

Figure 4. Accuracy at P(10)

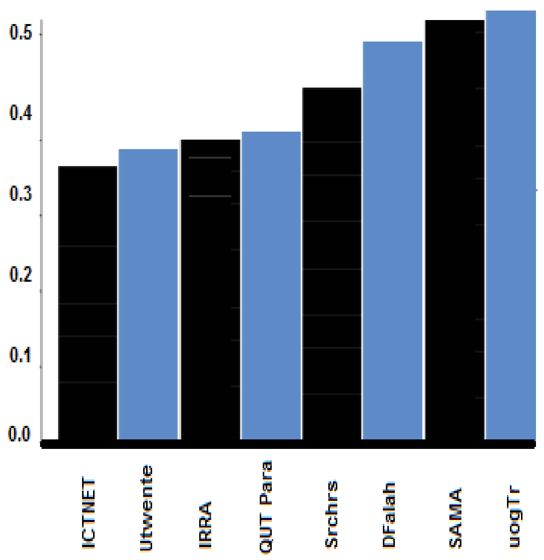

Figure 5. Accuracy at nDCG (20)

Table 1. The performance of different learning-to-rank methods

Method	Function	nDCG@1	nDCG@3	nDCG@5	nDCG@`0	nDCG@20
RankMSE	ReLU	0.446	0.430	0.432	0.447	0.469
RankNet	ELU	0.444	0.434	0.439	0.455	0.479
LambdaRank	RReLU	0.467	0.449	0.452	0.468	0.491
ListNet	ReLU	0.454	0.432	0.434	0.450	0.473
ListMLE	ELU	0.452	0.434	0.439	0.455	0.476
RankCosine	LeakyReLU	0.446	0.430	0.434	0.448	0.471
ApproxNDCG	Sigmoid	0.447	0.426	0.428	0.442	0.465
WassRank	ELU	0.449	0.430	0.434	0.449	0.470
ST-ListNet	ReLU	0.450	0.434	0.438	0.453	0.475

8. Conclusion

Web search engines researchers are widely used Learning-to-Rank algorithms to improve searching performance, meanwhile, ANN algorithms have been significantly used for enhancing learning quality. The major contribution of this manuscript is a supervised learning task for a powerful learning schema. Although enormous algorithms in the neural network have been used in many fields, the thematic analysis of convergence throughout learning is much less. Based on our experiments with the XOR problem, which took approximately 16000 epochs to converge or when the XOR networks proceeded within the error rate of 0.04, larger networks convergence took much longer and perhaps nearly ten times as long. Others ultimately converges proceeded within $4X10^{-3}$ of the desired output after one million epochs, these networks never converged much closer than 0.02 within the same period. However, it could be difficult to rank the items from disparate sources in real-time, as this requires multiple predictive algorithms and deep learning concepts. The professional implementation is essential to timely generate acceptable neural network models on frequently updated training datasets. Efficiently mapping the query levels on the neural network computation and data structure and fully utilizing inherent fine-grained concurrency are the most important assumptions which must be considered. The speedup took up to 17.9X over the software implementation on datasets from a commercial search engine. We evaluate and compare our algorithm with existing state-of-the-art methods on both synthetic and real-world crawled datasets. Theoretical and empirical analysis shows that our method can achieve significant efficiency and effectiveness improvements in top k relevance. We faced many issues, including coordination, synchronization, conjunctions, and ranking multiple autonomous web resources, which consequently led to the context of a robust approach. In the future, we will further explore the possibility of extending ANN for more general ranking scenarios or construct a new Learning-to-Rank model that integrates the optimization of relevance and web from the bottom of model design.

References

[1] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: Optimization by a colony of cooperating agents,” IEEE Transactions on Systems, Man, and Cybernetics, Part B, vol.26, no.1, vol.29-41, **(1996)**

[2] H. Bottee and E. Bonabeau, “Evolving ant colony optimization,” Adv. Complex Syst., vol.1, no.2/3, pp.149-159, **(1998)**

[3] J. Ast, R. Babuˇska, and B. Schutter, “Ant colony learning algorithm for optimal control,” in Interactive Collaborative Information Systems (R. Babuˇska and F.C.A. Groen, eds.), vol.281 of Studies in Computational Intelligence, Germany: Springer, ISBN 978-3-642-11687-2, pp.155-182, **(2010)**

[4] J. Anderson and L. Miller, “Deterministic parallel list ranking,” Algorithmica 6, pp.859-868, **(1991)**, DOI:10.1007/BF01759076

[5] C. Manning, P. Raghavan, H., and Schütze, “Introduction to information retrieval,” Cambridge University Press, Section 7, **(2008)**

[6] A. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, “Efficient query evaluation using a two-level retrieval process,” in Proceedings of the Twelfth International Conference on Information and Knowledge Management, pp.426-434, ISBN 978-1-58113-723-1, **(2003)**

[7] M. Mohri, A. Rostamizadeh, and A. Talwalkar, “Foundations of machine learning,” The MIT Press ISBN 9780262018258, **(2012)**

[8] J. Allan, J. Aslam, N. Belkin, C. Buckley, J. Callan, B. Croft, S. Dumais, N. Fuhr, D. Harman, D. Harper, D. Hiemstra, E. Hovy, J. Lafferty, V. Lavrenko, D. Lewis, L. Liddy, R. Manmatha, A. McCallum, J. Ponte, J. Prager, D. Radev, P. Resnik, S. Robertson, R. Rosenfeld, S. Roukos, M. Sanderson, R. Schwartz, A. Singhal,

A. Smeaton, H. Turtle, E. Voorhees, R. Weischedel, J. Xu, and C. Zhai, "Challenges in information retrieval and language modeling," report of a workshop held at the center for intelligent information retrieval, University of Massachusetts Amherst, ACM SIGIR Forum, **(2003)**, DOI:10.1145/945546.945549

[9] J. Yan, N. Xu, X. Cai, R. Gao, Y. Wang, R. Luo, and F. Hsu, "FPGA-based acceleration of neural network for ranking in web search engine with a streaming architecture," International Conference on Field Programmable Logic and Applications **(2009)**

[10] V. Ha-Thuc and S. H. Sinha, "Learning to rank personalized search results in professional networks," Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, **(2016)**

[11] S. Cooper, C. Gey, and P. Dabney, "Probabilistic retrieval based on staged logistic regression," In Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval, pp.198-210, **(1992)**, DOI: 10:1145/133160:133199

[12] H. Friedman, "Greedy function approximation: A gradient boosting machine," Annals of Statistics 29, pp.1189-1232, **(2000)**

[13] P. Li, Q. Wu, and J. Burges, "Mcrank: Learning to rank using multiple classifications and gradient boosting," In Advances in Neural Information Processing Systems, pp.897-904, **(2008)**

[14] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," In Proceedings of the 22nd International Conference on Machine Learning. pp.89-96, ICML '05, ACM, New York, NY, USA, **(2005)**, DOI: 10:1145/1102351:1102363

[15] Z. Cao, T. Qin, Y. Liu, F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," **(2007)**, https://www:microsoft:com/en-us/research/publication/learning-to-rank-from-pairwise-approach-to-listwise-approach/

[16] S. Ibrahim and D. Landa-Silva, "Es-rank: Evolution strategy learning to rank approach," in Proceedings of the Symposium on Applied Computing, pp.944-950, ACM, **(2017)**, DOI: 10.1145/3019612.3019696

[17] J. Xu and H. Li, "Ada rank: A boosting algorithm for information retrieval," in Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.391-398, SIGIR '07, New York, NY, USA, **(2007)**, DOI: 10.1145/1277741.1277809

[18] Q. Wu, J. Burges, M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," Information Retrieval 13, pp.254-270, **(2010)**, https://www:microsoft:com/en-us/research/publication/adapting-boosting-for-information-retrieval-measures/

[19] X. Dong, X. Chen, Y. Guan, and S. Li, "An overview of learning to rank for information retrieval," in Proceedings of the CSIE, 2009 WRI World Congress on Computer Science and Information Engineering, March 31-April 2, 2009, Los Angeles, California, USA, vol.7, **(2009)**

[20] T. Yang and Q. Ai, "Maximizing marginal fairness for dynamic learning to tank," International World Wide Web Conference Committee, ACM ISBN 978-1-4503-8312-7/21/04, **(2021)**, DOI: 10.1145/3442381.3449901

[21] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "DCN V2: Improved deep and cross-network and practical lessons for web-scale learning to rank systems," arXiv:2008.13535, **(2020)**

[22] C. Lucchese, C. Muntean, F. Nardini, R. Perego, and S. Trani, "RankEval: An evaluation and analysis framework for learning-to-rank solutions," Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.1281-284, **(2017)**, DOI: 10.1145/3077136.3084140

[23] H. Yu, "PT-ranking: A benchmarking platform for neural learning-to-rank," ArXiv Journal, vol. abs/2008.13368, **(2008)**

[24] B. Xiang, D. Jiang, J. Pei, X. Sun, E. Chen, and H. Li, "Context-aware ranking in web search," in Fabio Crestani, Stéphane Marchand-Maillet, Hsin-Hsi Chen, Efthimis N. Efthimiadis, and Jacques Savoy, editors, SIGIR, ACM, 75, pp.451-458, **(2010)**

[25] A. Dong, Y. Chang, Z. Zheng, G. Mishne, J. Bai, R. Zhang, K. Buchner, C. Liao, and F. Diaz, "Towards recency ranking in web search," in WSDM, pp.11-20, **(2010)**, DOI: 10.1145/1718487.1718490 75

[26] A. Ponnuswami, K. Pattabiraman, Q. Wu, R. Gilad-Bachrach, and T. Kanungo, "On the composition of a federated web search result page: using online users to provide pairwise preference for heterogeneous verticals," in WSDM, pp.715-724, **(2011),** DOI: 10.1145/1935826.1935922 75

[27] S. Kanti, K. Santu, P. Sondhi, C. Zhai, "On application of learning to rank for e-commerce search," arXiv:1903.04263, DOI: 10.1145/3077136.3080838, **(2019)**

[28] Y. Cao, J. Xu, Y. Liu, H. Li, Y. Huang, and W. Hon, "Adapting ranking SVM to document retrieval," in Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. pp.186, 193, **(2006)** DOI: 10.1145/1148170.1148205

[29] R. Arens, "Learning to rank documents with support vector machines via active learning," Ph.D. Thesis, University of Iowa, **(2009)**

[30] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," arXiv:2009.07888v1 [cs.LG] **(2020)**

[31] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in Proceedings of 27th NIPS, pp.2042-2050, **(2014)**

[32] F. Al-akashi and D. Inkpen, "Term impact-based web page ranking," in Proceeding of the International Conference on Web Intelligence, Mining and Semantics (WIMS14), Thessaloniki, Greece, **(2014)**

[33] T. Qin, T. Y. Liu, J. Xu, "LETOR: A benchmark collection for research on learning to rank for information retrieval," Information Retrieval, vol.13, pp.346-374, **(2010)**, DOI: 10.1007/s10791-009-9123-y

[34] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "DCN V2: Improved deep and cross-network and practical lessons for web-scale learning-to-rank systems," arXiv:2008.13535v2 [cs.IR], **(2020)**

[35] X. Jifeng and M. Martin, "Learning to combine multiple ranking metrics for fault localization," IEEE International Conference on Software Maintenance and Evolution. pp.191-200, CiteSeerX 10.1.1.496.6829, **(2014)**, DOI: 10.1109/ICSME.2014.41, ISBN 978-1-4799-6146-7

[36] L. Tie-Yan, "Learning to rank for information retrieval," Foundations and Trends in Information Retrieval. pp.225-331, **(2009)**, DOI:10.1561/1500000016, ISBN 978-1-60198-244-5

[37] W. Kim, P. Sharma, J. Lee, S. Baneriee, J. Tourilhes, S. Lee, and P. Yalagandula, "Automated and scalable QoS control for network convergence," in Proceedings of the Internet Network Management Conference on Research on Enterprise Networking, pp.1, **(2010)**

[38] S. Aloteibi and S. Clark, "Learning to personalize for web search sessions," in Proceedings of the CIKM, **(2020)**

[39] J. Kalervom and K. Jaana, "In evaluation methods for retrieving highly relevant documents," in Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '00, pp.41-48, New York, NY, USA, **(2000)**, ACM. DOI: 10.1145/345508.345545 17

[40] E. Voorhees and D. Harman, "TREC: Experiment and evaluation in information retrieval," MIT, 17, **(2005)**

[41] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan, "Expected reciprocal rank for graded relevance," in Proceedings of the CIKM, **(2009)**

This page is empty by intention.