Personalized Re-ranking for Recommendation

Changhua Pei[1*], Yi Zhang[1*], Yongfeng Zhang[2*]
Fei Sun[1], Xiao Lin[1], Hanxiao Sun[1], Jian Wu[1], Peng Jiang[3], Junfeng Ge[1], Wenwu Ou[1], Dan Pei[4]
[1] Alibaba Group [2] Rutgers University [3] Kwai Inc. [4] Tsinghua University
[1] {changhua.pch, zhanyuan.zy, ofey.sf, hc.lx, hansel.shx, joshuawu.wujian, beili.gjf, santong.oww}@alibaba-inc.com
[2] yongfeng.zhang@rutgers.edu [3] jiangpeng@kuaishou.com [4]peidan@tsinghua.edu.cn

ABSTRACT

Ranking is a core task in recommender systems, which aims at providing an ordered list of items to users. Typically, a ranking function is learned from the labeled dataset to optimize the global performance, which produces a ranking score for each individual item. However, it may be sub-optimal because the scoring function applies to each item individually and does not explicitly consider the mutual influence between items, as well as the differences of users' preferences or intents. Therefore, we propose a personalized re-ranking model for recommender systems. The proposed re-ranking model can be easily deployed as a follow-up modular after any ranking algorithm, by directly using the existing ranking feature vectors. It directly optimizes the whole recommendation list by employing a transformer structure to efficiently encode the information of all items in the list. Specifically, the Transformer applies a self-attention mechanism that directly models the global relationships between any pair of items in the whole list. We confirm that the performance can be further improved by introducing pre-trained embedding to learn personalized encoding functions for different users. Experimental results on both offline benchmarks and real-world online e-commerce systems demonstrate the significant improvements of the proposed re-ranking model.

CCS CONCEPTS

• **Information systems → Recommender systems**.

KEYWORDS

Learning to rank; Re-ranking; Recommendation

ACM Reference Format:
Changhua Pei[1*], Yi Zhang[1*], Yongfeng Zhang[2] and Fei Sun[1], Xiao Lin[1], Hanxiao Sun[1], Jian Wu[1], Peng Jiang[3], Junfeng Ge[1], Wenwu Ou[1], Dan Pei[4]. 2019. Personalized Re-ranking for Recommendation . In *Thirteenth ACM Conference on Recommender Systems (RecSys '19), September 16–20, 2019, Copenhagen, Denmark.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3298689.3347000

[*]Changhua Pei and Yi Zhang contribute equally. Yongfeng Zhang is the corresponding author.

RecSys '19, September 16–20, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-6243-6/19/09…$15.00
https://doi.org/10.1145/3298689.3347000

1 INTRODUCTION

Ranking is crucial in recommender systems. The quality of the ranked list given by a ranking algorithm has a great impact on users' satisfaction as well as the revenue of the recommender systems. A large amount of ranking algorithms [4, 5, 7, 15, 19, 27, 32] have been proposed to optimize the ranking performance. Typically ranking in recommender system only considers the user-item pair features, without considering the influences from other items in the list, especially by those items placed alongside[8, 35]. Though *pairwise* and *listwise* learning to rank methods try to solve the problem by taking the item-pair or item-list as input, they only focus on optimizing the loss function to make better use of the labels, *e.g.,* click-through data. They didn't explicitly model the mutual influences between items in the feature space.

Some works[1, 34, 37] tend to model the mutual influences between items explicitly to refine the initial list given by the previous ranking algorithm, which is known as re-ranking. The main idea is to build the scoring function by encoding intra-item patterns into feature space. The state-of-the-art methods for encoding the feature vectors are RNN-based, such as GlobalRerank[37] and DLCM[1]. They feed the initial list into RNN-based structure sequentially and output the encoded vector at each time step. However, RNN-based approaches have limited ability to model the interactions between items in the list. The feature information of the previous encoded item degrades along with the encoding distance. Inspired by the Transformer architecture[20] used in machine translation, we propose to use the Transformer to model the mutual influences between items. The Transformer structure uses self-attention mechanism where any two items can interact with each other directly without degradation over the encoding distance. Meanwhile, the encoding procedure of Transformer is more efficient than RNN-based approach because of parallelization.

Besides the interactions between items, personalized encoding function of the interactions should also be considered for re-ranking in recommender system. Re-ranking for recommender system is user-specific, depending on the user's preferences and intents. For a user who is sensitive to price, the interaction between "price" feature should be more important in the re-ranking model. Typical global encoding function may be not optimal as it ignores the differences between feature distributions for each user. For instance, when users are focusing on price comparison, similar items with different prices tend to be more aggregated in the list. When the user has no obvious purchasing intention, items in the recommendation list tend to be more diverse. Therefore, we introduce a personalization module into the Transformer structure to represent user's preference and intent on item interactions. The interaction between

items in the list and user can be captured simultaneously in our personalized re-ranking model.

The main contributions of this paper are as follows:

- **Problem**. We propose a personalized re-ranking problem in recommender systems, which, to the best of our knowledge, is the first time to explicitly introduce the personalized information into re-ranking task in large-scale online system. The experimental results demonstrate the effectiveness of introducing users' representation into list representation for re-ranking.
- **Model**. We employ the Transformer equipped with personalized embedding to compute representations of initial input ranking list and output the re-ranking score. The self-attention mechanism enable us to model user-specific mutual influences between any two items in a more effective and efficient way compared with RNN-based approaches.
- **Data**. We release a large scale dataset (E-commerce Re-ranking dataset) used in this paper. This dataset is built from a real-world E-commerce recommender system. Records in the dataset contain a recommendation list for user with click-through labels and features for ranking.
- **Evaluation**. We conducted both offline and online experiments which show that our methods significantly outperform the state-of-the-art approaches. The online A/B tests show that our approach achieves higher click-through rate and more revenue for real-world system.

2 RELATED WORK

Our work aims to refine the initial ranking list given by the base ranker. Among these base rankers, learning to rank is one of the widely used methods. The learning to rank methods can be classified into three categories according to the loss function they used: *pointwise*[12, 21], *pairwise*[6, 18, 19], and *listwise*[5, 7, 14, 19, 27, 32, 33]. All these methods learn a global scoring function within which the weight of a certain feature is globally learned. However, the weights of the features should be able to be aware of the interactions not only between items but also between the user and items.

Closest to our work are [1–3, 37], which are all re-ranking methods. They use the whole initial list as input and model the complex dependencies between items in different ways. [1] uses unidirectional GRU[10] to encode the information of the whole list into the representation of each item. [37] uses LSTM[17] and [3] uses pointer network[29] not only to encode the whole list information, but also to generate the ranked list by a decoder. For those methods which use either GRU or LSTM to encode the dependencies of items, the capacity of the encoder is limited by the encoding distance. In our paper, we use transformer-like encoder, based on self-attention mechanism to model the interactions for any of two items in $\mathbf{O}(\mathbf{1})$ distance. Besides, for those methods which use decoder to sequentially generate the ordered list, they are not suitable for online ranking system which requires strict latency criterion. As the sequential decoder uses the item selected at time t-1 as input to select the item at time t, it can not be parallelized and needs n times of inferences, where n is the length of the output list. [2] proposes a *groupwise* scoring function which can be parallelized when scoring the items, but its computation cost is high because it enumerates every possible combinations of items in the list.

Table 1: Notation used in this paper.

Notation.	Description.
$\boldsymbol{X}$	The matrix of features.
$\boldsymbol{PV}$	The matrix of personalized vectors.
$\boldsymbol{PE}$	The matrix of position embeddings.
$\boldsymbol{E}$	The output matrix of the input layer.
$\mathcal{R}$	The set of total users' requests.
$\mathcal{I}_r$	The set of candidate items for each user's request $r \in \mathcal{R}$.
$\mathcal{S}_r$	The initial list of items generated by the ranking approaches for each user's request r.
$\mathcal{H}_u$	The sequence of items clicked by user u.
$\theta, \hat{\theta}, \theta'$	The parameter matrices of ranking, re-ranking and pre-trained model respectively.
y_i	The label of click on item i.
$P(y_i\|\cdot)$	The click probability of item i predicted by the model.

3 RE-RANKING MODEL FORMULATION

In this section, we first give some preliminary knowledge about learning to rank and re-ranking methods for recommendation systems. Then we formulate the problem we aim to solve in this paper. The notations used in this paper are in Table 1.

Learning to rank (often labelled as **LTR**) method is widely used for ranking in real-work systems to generate an ordered list for information retrieval[18, 22] and recommendation[14]. The LTR method learns a global scoring function based on the feature vector of items. Having this global function, the LTR method outputs an ordered list by scoring each item in the candidate set. This global scoring function is usually learned by minimizing the following loss function $\mathcal{L}$:

$$\mathcal{L} = \sum_{r \in \mathcal{R}} \ell\Big(\{y_i, P(y_i|\boldsymbol{x_i}; \theta)|i \in \mathcal{I}_r\}\Big) \tag{1}$$

where $\mathcal{R}$ is the set of all users' requests for recommendation. $\mathcal{I}_r$ is the candidate set of items for request $r \in \mathcal{R}$. $\boldsymbol{x_i}$ represents the feature space of item i. y_i is the label on item i, *i.e.,* click or not. $P(y_i|\boldsymbol{x_i}; \theta)$ is the predicted click probability of item i given by the ranking model with parameters θ. ℓ is the loss computed with y_i and $P(y_i|\boldsymbol{x_i}; \theta)$.

However, $\boldsymbol{x_i}$ is not enough to learn a good scoring function. We find that ranking for recommender system should consider the following extra information: (a) mutual influences between item-pairs[8, 35]; (b) interactions between the users and items. The mutual influences between item-pairs can be directly learned from the initial list $\mathcal{S}_r = [i_1, i_2, ..., i_n]$ given by the existing LTR model for the request r. Works[1][37][2][3] propose approaches to make better use of mutual information of item-pairs. However, few works consider the interactions between the users and items. The extent of mutual influences of item-pairs varies from user to user. In this paper, we introduce a personalized matrix $\boldsymbol{PV}$ to learn user-specific encoding function which is able to model personalized mutual influences between item-pairs. The loss function of the model can

be formulated as Equation 2.

$$\mathcal{L} = \sum_{r \in \mathcal{R}} \ell\Big(\{y_i, P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta}) | i \in \mathcal{S}_r\}\Big) \tag{2}$$

where $\mathcal{S}_r$ is the initial list given by the previous ranking model. $\hat{\theta}$ is the parameters of our re-ranking model. $\boldsymbol{X}$ is the feature matrix of all items in the list.

4 PERSONALIZED RE-RANKING MODEL

In this section, we first give an overview of our proposed **P**ersonalized **R**e-ranking **M**odel (PRM). Then we introduce each component of our model in detail.

4.1 Model Architecture

The architecture of PRM model is shown in Figure 1. The model consists of three parts: the *input* layer, the *encoding* layer and the *output* layer. It takes the initial list of items generated by previous ranking method as input and outputs a re-ranked list. The detailed structure will be introduced separately in the following sections.

4.2 Input Layer

The goal of the input layer is to prepare comprehensive representations of all items in the initial list and feed it to the encoding layer. First we have a fixed length of initial sequential list $\mathcal{S} = [i_1, i_2, ..., i_n]$ given by the previous ranking method. Same as the previous ranking method, we have a raw feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{feature}}}$. Each row in $\boldsymbol{X}$ represents the raw feature vector $\boldsymbol{x_i}$ for each item $i \in \mathcal{S}$.

Personalized Vector (PV). Encoding the feature vectors of two items can model the mutual influences between them, but to which extent these influences may affect the user is unknown. A user-specific encoding function need to be learned. Though the representation of the whole initial list can partly reflects the user's preferences, it is not enough for a powerful personalized encoding function. As shown in Figure 1 (b), we concat the raw feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{feature}}}$ with a personalized matrix $\boldsymbol{PV} \in \mathbb{R}^{n \times d_{\text{pv}}}$ to get the intermediate embedding matrix $\boldsymbol{E}' \in \mathbb{R}^{n \times (d_{\text{feature}} + d_{pv})}$, which is shown in Equation 3. $\boldsymbol{PV}$ is produced by a pre-trained model which will be introduced in the following section. The performance gain of $\boldsymbol{PV}$ will be introduced in the evaluation section.

$$\boldsymbol{E}' = \begin{bmatrix} \boldsymbol{x}_{i_1} ; \boldsymbol{pv}_{i_1} \\ \boldsymbol{x}_{i_2} ; \boldsymbol{pv}_{i_2} \\ \cdots \\ \boldsymbol{x}_{i_n} ; \boldsymbol{pv}_{i_n} \end{bmatrix} \tag{3}$$

Position Embedding (PE). In order to utilize the sequential information in the initial list, we inject a position embedding $\boldsymbol{PE} \in \mathbb{R}^{n \times (d_{\text{feature}} + d_{pv})}$ into the input embedding. Then the embedding matrix for encoding layer can be calculated using Equation 4. In this paper, a learnable $\boldsymbol{PE}$ is used which we found that it slightly outperforms the fixed position embedding used in [28].

$$\boldsymbol{E}'' = \begin{bmatrix} \boldsymbol{x}_{i_1} ; \boldsymbol{pv}_{i_1} \\ \boldsymbol{x}_{i_2} ; \boldsymbol{pv}_{i_2} \\ \cdots \\ \boldsymbol{x}_{i_n} ; \boldsymbol{pv}_{i_n} \end{bmatrix} + \begin{bmatrix} \boldsymbol{pe}_{i_1} \\ \boldsymbol{pe}_{i_2} \\ \cdots \\ \boldsymbol{pe}_{i_n} \end{bmatrix} \tag{4}$$

At last we use one simple feed-forward network to convert the feature matrix $\boldsymbol{E}'' \in \mathbb{R}^{n \times (d_{\text{feature}} + d_{pv})}$ to $\boldsymbol{E} \in \mathbb{R}^{n \times d}$, where d is latent dimensionality of each input vector of encoding layer. $\boldsymbol{E}$ can be formulated as Equation 5.

$$\boldsymbol{E} = \boldsymbol{E}\boldsymbol{W}^E + b^E \tag{5}$$

where $\boldsymbol{W}^E \in \mathbb{R}^{(d_{\text{feature}} + d_{pv}) \times d}$ is the projection matrix and b^E is d-dimensional vector.

4.3 Encoding Layer

The goal of the encoding layer in Figure 1(a) is to integrate the mutual influences of item-pairs and other extra information, includes the user preferences and the ranking order of the initial list $\mathcal{S}$. To achieve this goal, we adopt Transformer-like encoder because Transformer[28] has been proven to be effective in many NLP tasks, specially in machine translation for its powerful encoding and decoding ability compared to RNN-based approaches[10, 11, 17]. The self-attention mechanism in Transformer is particularly suitable in our re-ranking task as it directly models the mutual influences for any two items regardless the distances between them. Without distance decay, Transformer can capture more interactions between items that are far away from each other in the initial list. As shown in Figure 1(b), our encoding module consists of N_x blocks of Transformer encoder. Each block (Figure 1(a)) contains an attention layer and a Feed-Forward Network (FFN) layer.

Attention Layer. The attention function we used in this paper is defined as Equation 6:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V}, \tag{6}$$

where matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ represent queries, keys and values respectively. d is the dimensionality of matrix $\boldsymbol{K}$ to avoid large value of the inner product. *softmax* is used to convert the value of inner-product into the adding weight of the value vector $\boldsymbol{V}$. In our paper, we use self-attention where $\boldsymbol{Q}, \boldsymbol{K}$ and $\boldsymbol{V}$ are projected from the same matrices.

To model more complex mutual influences, we use the multi-head attention as shown in Equation 7:

$$\begin{aligned} \boldsymbol{S}' &= \text{MH}(\boldsymbol{E}) = \text{Concat}(\text{head}_1, ..., \text{head}_h)\boldsymbol{W}^O \\ \text{head}_i &= \text{Attention}(\boldsymbol{E}\boldsymbol{W}^Q, \boldsymbol{E}\boldsymbol{W}^K, \boldsymbol{E}\boldsymbol{W}^V), \end{aligned} \tag{7}$$

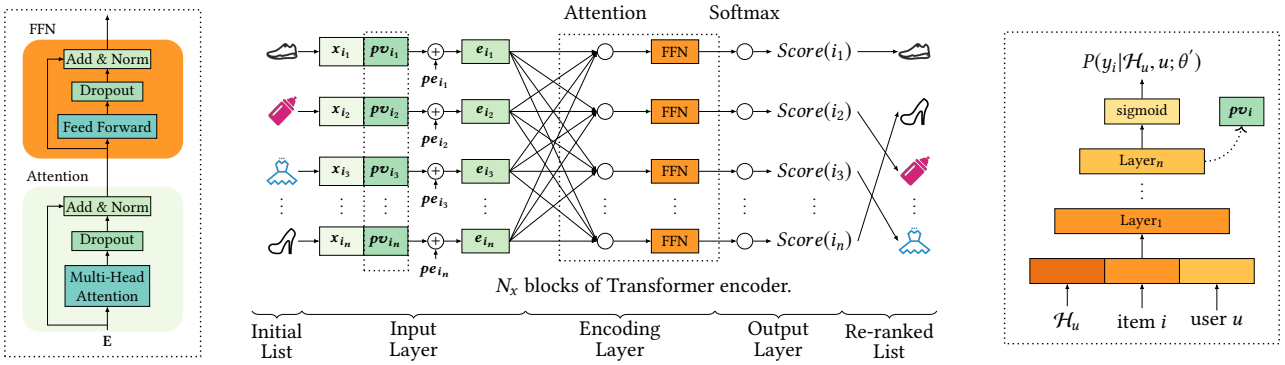
where $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V \in \mathbb{R}^{d \times d}$. $\boldsymbol{W}^O \in \mathbb{R}^{hd \times d_{\text{model}}}$ is the projection matrix. h is the number of headers. The influence of different value of h will be studied in the ablation study in the next section.

Feed-Forward Network. The function of this position-wise Feed-Forward Network (FFN) is mainly to enhance the model with non-linearity and interactions between different dimensions of the input vectors.

Stacking the Encoding Layer. Here we use attention module followed by the position-wise FFN as a block of Transformer[28] encoder. By stacking multiple blocks, we can get more complex and high-order mutual information.

4.4 Output Layer

The function of the output layer is mainly to generate a score for each item $i = i_1, \ldots, i_n$ (labeled as $Score(i)$ in Figure 1 (b)). We use



(a) One block of Transformer encoder.

(b) Architecture of PRM.

(c) The pre-trained model to generate $\boldsymbol{pv_i}$, $i = i_1, ..., i_n$.

Figure 1: The detailed network structure of our PRM (Personalized Re-ranking Model) and its sub-modules.

one linear layer followed by a softmax layer. The output of softmax layer is the probability of click for each item, which is labeled as $P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta})$. We use $P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta})$ as $Score(i)$ to re-rank the items in one-step. The formulation of $Score(i)$ is:

$$Score(i) = P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta}) = \text{softmax}\left(\boldsymbol{F}^{(\boldsymbol{N_x})}\boldsymbol{W}^F + \boldsymbol{b}^F\right), i \in \mathcal{S}_r \tag{8}$$

where $\boldsymbol{F}^{(\boldsymbol{N_x})}$ is the output of N_x blocks of Transformer encoder. $\boldsymbol{W}^F$ is learnable projection matrix, and $\boldsymbol{b}^F$ is the bias term. n is the number of items in the initial list.

In the training process, we use the click-through data as label and minimize the loss function shown in Equation 9.

$$\mathcal{L} = -\sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{S}_r} y_i \log(P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta}) \tag{9}$$

4.5 Personalized Module

In this section, we introduce the approach to calculate the personalized matrix $\boldsymbol{PV}$, which represents interactions between user and items. The straightforward approach is to learn $\boldsymbol{PV}$ with PRM model in an end-to-end manner via the re-ranking loss. However, as explained in Section3, the re-ranking task is to refine the output of previous ranking approaches. The task-specific representation learned on re-ranking task lacks users' generic preferences. Therefore, we utilize a pre-trained neural network to produce user's personalized embeddings $\boldsymbol{PV}$ which are then used as extra features for PRM model. The pre-trained neural network is learned from the whole click-through logs of the platform. Figure 1(c) shows the structure of pre-trained model used in our paper. This sigmoid layer outputs the click probability ($P(y_i|\mathcal{H}_u, u; \theta')$) on item i for user u given user's all behavior history ($\mathcal{H}_u$) and the side information of the user. The side information of user includes *gender, age* and *purchasing level*, et.al. The loss of the model is calculated by a point-wise cross entropy function which is shown in Equation 10.

$$\begin{aligned}\mathcal{L} = \sum_{i \in \mathcal{D}} & (y_i \log(P(y_i|\mathcal{H}_u, u; \theta')) \\ & + (1 - y_i) \log(1 - P(y_i|\mathcal{H}_u, u; \theta')),\end{aligned} \tag{10}$$

where $\mathcal{D}$ is the set of items displayed to user u on the platform. θ' is the parameter matrix of pre-trained model. y_i is the label (click or not) on item i. Inspired by the work[13], we employ the hidden vector before the *sigmoid* layer as the personalized vector $\boldsymbol{pv_i}$ (in Figure 1(c)) that feeds into our PRM model.

Figure 1(c) shows one possible architecture of the pre-trained model, other general models such as FM[25], FFM[23], DeepFM[16], DCN[30], FNN[36] and PNN[24] can also be used as alternatives to generate $\boldsymbol{PV}$.

5 EXPERIMENTAL RESUTLS

In this section, we first introduce the datasets and baselines used for evaluation. Then we compare our methods with baselines on these datasets to evaluate the effectiveness of our PRM model. At the same time, the ablation study is conducted to help understand which part of our model contributes most to the overall performance.

5.1 Datasets

We evaluate our approach based on two datasets: Yahoo! Webscope v2.0 set 1[1] (abbreviated as Yahoo Letor dataset) and E-commerce Re-ranking dataset[2] To the best of our knowledge, there is no publically available re-ranking dataset with context information for recommendation. Therefore, we construct E-commerce Re-ranking dataset from a popular e-commerce platform. The overview of two datasets are shown in Table 2.

Yahoo Letor dataset. We process the Yahoo Letor dataset to be fit for the ranking model of recommendation using the same method in Seq2Slate[3]. Firstly, we convert the ratings (0 to 4) to binary labels using a threshold T_b. Secondly, we use a decay factor η to simulate the impression probabilities of items. All the documents in Yahoo Letor dataset are rated by the experts under the assumption that all documents for each query can be viewed by the users compeletely. However, in the real world recommendation scenario, items are viewed by the users in a top-down manner. As the screen of the mobile App can only show limited number of items, the higher the ranked position of one item, the smaller

[1] http://webscope.sandbox.yahoo.com
[2] Our dataset is available at https://github.com/rank2rec/rerank.

Table 2: Overview of the datasets.

	Yahoo Letor Dataset	E-commerce Re-ranking Dataset
#Users	-	743,720
#Docs/Items	709,877	7,246,323
#Records	29,921	14,350,968
Relavance/Feedback	{0,1,2,3,4}	{0,1}

probability of that this item can be viewed by the user. In this paper, we use $1/pos(i)^\eta$ as the decay probability, where $pos(i)$ is the ranking position of item i in the initial list.

E-commerce Re-ranking dataset. The dataset contains a large-scale records in form of click-through data from a real world recommendation system. Each record in the dataset contains a recommendation list for each user with users' basic information, click-through labels and raw features for ranking.

5.2 Baselines

Both learning to rank (LTR) and re-ranking methods can act as our baselines.

LTR. The LTR methods are used in two tasks. Firstly, the LTR methods can generate an initial list $\mathcal{S}_r$ for the re-ranking model from a candidate set $\mathcal{I}_r$ for each user request r. Secondly, the LTR methods which use pairwise or listwise loss function can act as re-ranking methods by taking the initial list $\mathcal{S}_r$ as input and conducting the ranking algorithm for another time. The representative LTR methods used in this paper include:

- SVMRank[19]: This is a representative learning to rank method which use the pairwise loss to model the scoring function.
- LambdaMart[5]: This is a representative learning to rank method which use the listwise loss to model the scoring function. LambdaMart is the state-of-the-art LTR among those LTR methods equipped with the listwise loss function according to [31]'s evaluation.
- DNN-based LTR: This is the learning to rank method which is deployed in our online recommender system. It use the standard Wide&Deep network structure[9] to model the scoring function via the pointwise loss function.

Re-ranking. As mentioned in the related work section, the existing re-ranking methods include DLCM[1], Seq2Slate[3] and GlobalRerank[37]. DLCM[1] and GlobalRerank[37] focus on re-ranking in information retrieval. Seq2Slate[3] focuses on re-ranking in both recommendation and information retrieval. In this paper, we only choose DLCM as baseline method. Seq2Slate and GlobalRerank are not chosen as baselines because they all use the decoder structure to generate the re-ranked list. Seq2Slate uses pointer network to generate re-ranked list sequentially. GlobalRerank uses RNN equipped with attention mechanism as the decoder. The decoder structure outputs the item one by one. Whether an item is selected depends on the items which are chosen before it. As a consequence, both Seq2Slate and GlobalRerank can not be parallelized in online inference. The time complexity for Seq2Slate and GlobalRerank at interference phase is $\mathbf{O}(n) \times RT$, where n is the length of the initial list and RT is the time for a single ranking or re-ranking request. The latency for re-ranking by Seq2Slate and GlobalRerank is unacceptable because of the strict latency criterion for online recommender service.

- DLCM[1]: It is a re-ranking model used in information retrieval based on the initial list generated by LTR methods. The GRU is used to encode the local context information into a global vector. Combing the global vector and each feature vector, it learns a more powerful scoring function than the global ranking function of LTR.

5.3 Evaluation Metrics

For offline evaluation, we use *Precision* and *MAP* to compare different methods. More specifically, we use Precision@5, Precision@10 for precision and MAP@5, MAP@10 and MAP@30 for MAP. As the maximum length of initial list in our experiments is 30, MAP@30 represents total MAP and is denoted by MAP in this paper. The definitions of the metrics are as follows.

Precision@k is defined as the the fraction of clicked items in the top-k recommended items for all test samples, as shown in Equation 11.

$$Precision@k = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \frac{\sum_{i=1}^{k} \mathbf{I}(\mathcal{S}_r(i))}{k} \tag{11}$$

where $\mathcal{R}$ is the set of all user requests in the test dataset. $\mathcal{S}_r$ is the ordered list of items given by the re-ranking model for each request $r \in \mathcal{R}$ and $\mathcal{S}_r(i)$ is the i-th item. $\mathbf{I}$ is the indicator function whether item i is clicked or not.

MAP@k is short for the mean average precision of all ranked lists cut off by k in the test dataset. It is defined as follows.

$$MAP@k = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \frac{\sum_{i=1}^{k} Precision@i * \mathbf{I}(\mathcal{S}_r(i))}{k} \tag{12}$$

For online A/B test, we use PV, IPV, CTR and GMV as metrics. PV and IPV are defined as the total number of items viewed and clicked by the users. CTR is the clickthrough rate and can be calculated by IPV/PV. GMV is the total amount of money (revenue) user spent on the recommended items.

5.4 Experimental Settings

For both baselines and our PRM model, we use the same value for those critical hyper parameters. The hidden dimensionality d_{model} is set to 1024 for Yahoo Letor dataset and 64 for E-commerce Re-ranking dataset. The learning rate of Adam optimizer in our PRM model is the same with [28]. Negative log likelihood loss function is used as shown in Equation 9. p_{dropout} is set to 0.1. The batch size is set to 256 for Yahoo Letor dataset and 512 for E-commerce Re-ranking dataset. These settings are got by fine-tuning the baselines to achieve better performance. We also try different experimental settings, the results are consistent with the current settings and are ommited. The rest of the settings belonging to the customized parts of our model will be listed at the corresponding parts in the evaluation section.

5.5 Offline Experiments

In this section, we first conduct offline evaluations on Yahoo Letor dataset and E-commerce Re-ranking dataset. Then we show the

results of online A/B test. We also conduct the ablation study to help finding which part of our PRM model contributes most to the performance.

5.5.1 Offline Evaluation on Yahoo Letor dataset. In this section, we conduct evaluation on Yahoo Letor dataset to discuss the following questions:

- RQ0: Does our PRM model outperform the state-of-the-art methods and why?
- RQ1: Does the performance vary according to initial lists generated by different LTR approaches?

The evaluation results are shown in Table 3. We compare the baselines and our PRM-BASE model based on two different initial lists which are generated by LambdaMART and SVMRank respectively. PRM-BASE is the variant of our PRM model without the personalized module. Note that Yahoo Letor dataset does not contain user-related information, thus we only conduct PRM-BASE for comparison. SVMRank and LambdaMart are also used for re-ranking. For SVMRank, we use the implementation in [19]. For LambdaMart, we use the implementation from RankLib[3].

Table 3 shows that our PRM-BASE achieves stable and significant performance improvements comparing with all baselines. When based on the initial list generated by SVMRank, PRM-BASE outperforms DLCM by 1.7% at MAP and 1.4% at Precision@5. The gap gets larger when comparing with SVMRank which has 5.6% increase at MAP and 5.7% increase at Precision@5. When based on the initial list generated by LambdaMART, PRM-BASE outperforms DLCM by 0.7% at MAP and 2.1% at Precision@5. PRM-BASE also achieves 2.6% improvements on MAP and 3.1% improvements on Precision@5 comparing with LambdaMART .

PRM-BASE uses the same training data as DLCM and does not contain the personalized module. The performance gain over DLCM mainly comes from the powerful encoding ability of Transformer. Multi-head attention mechanism performs better at modeling mutual influence between two items, especially when the length of encoding list gets longer[20]. In our model, the attention mechanism can model the interactions of any item-pairs in $\mathbf{O}(1)$ encoding distance.

As PRM-BASE uses the Transformer-like structure, there are many sub-modules which may contribute to the performance. We conduct the ablation study to help us understand which sub-design helps the most to beat the baselines. The ablation study is conducted on the initial list generated by SVMRank. Similar results were found when using the initial list generated by LambdaMART and we omits the results in this paper as space is limited. Table 4 show the results of ablation in three parts: The first part (first row) shows the performance of the baseline DLCM. The second part (second row) "Default" is the best performance of our PRM-BASE model. The third part (the remaining rows) shows different ablation variants of our PRM model which include: remove position embedding (PE), remove residual connection (RC), remove dropout layer, use different number of blocks and use different number of heads in multi-head attention. Note that we set b=4 and h=3 in our "Default" PRM model.

[3]https://sourceforge.net/p/lemur/wiki/RankLib/

As shown in Table 4, the performance of our model degrades greatly after removing position embedding. This confirms the importance of sequential information given by the initial list. After removing the position embedding, our model learns the scoring function from the candidate set instead of an ordered list. Note that even without position embedding, our PRM-BASE still achieve comparable performance with DLCM, which further confirms that our PRM-BASE model can encode the initial list more effectively than DLCM.

The MAP of our model slightly decreases by 0.1% and 0.7% respectively when removing residual connections and dropout layer, which indicates that our model is less severe to the problems such as gradients vanishing and overfitting. The performance of our model first increases with the number of blocks (1->2->4) and decreases afterwards (4->6->8), as overfitting happens when we stack 8 encoding blocks together.

We also tried different settings($h = 1, 2, 3, 4$) in the multi-head attention layer. No significant improvements are observed in Table 4, which is different from the conclusions derived from NLP tasks[26]. The experiments in NLP show that when using more heads in multi-head attention mechanism, it is usually helpful since more information can be captured for the following reasons. (1) From Equation 7 we find that the function of each *head* is playing a role of mapping the original feature vector into a different subspace. Thus using more heads, we can model more interactions of items in different sub-spaces. (2) [26] indicates that using more heads is helpful in encoding the information of long sequence. This is reasonable because the output vector for a certain item is the weighted sum of all item vectors in the list. When the sequence becomes longer, each item in the list contributes less to the output vector. However, in our re-ranking settings, all the items in the initial list are highly homogenous. There are minor improvements when mapping the original feature vector into more different subspaces. As a consequence, we suggest to use only one head to save computation costs because the performance improvements are not obvious.

5.5.2 Offline Evaluation on E-commerce Re-ranking dataset. We conduct the offline evaluation on E-commerce Re-ranking dataset to answer the following question.

- RQ2: What is the performance of our PRM model equipped with personalized module?

The evaluation results are shown in Table 5. For our PRM models, we not only evaluated the performance of PRM-BASE, but also evaluated the performance of the variant of model equipped with the pre-trained personalized vector $\boldsymbol{PV}$, which is labelled as PRM-Personalized-Pretrain. As our previous evaluation on Yahoo Letor dataset already confirms that our model and DLCM achieve better performance in all metrics and DLCM[1] also has consistent results, we omit the comparison with SVMRank and LambdaMART on our E-commerce Re-ranking dataset. The initial list is generated by a DNN-based LTR method which is deployed in our real world recommender system.

Table 5 shows consistent results with Table 3 when comparing PRM-BASE with DLCM. Our PRM-BASE outperforms DLCM by 2.3% at MAP and 4.1% at Precision@5. Recall that on Yahoo Letor dataset, PRM-BASE achieves 1.7% improvements on MAP and 1.4%

Table 3: Offline evaluation results on Yahoo Letor dataset.

Init. List	Reranking	Yahoo Letor dataset.				
		Precision@5(%)	Precision@10(%)	MAP@5(%)	MAP@10(%)	MAP(%)
SVMRank	SVMRank	50.42	42.25	73.71	68.28	62.14
	LambdaMART	51.35	43.08	74.94	69.54	63.38
	DLCM	52.54	43.26	76.52	70.86	64.50
	PRM-BASE	**53.29**	**43.66**	**77.62**	**72.02**	**65.60**
LambdaMART	SVMRank	50.41	42.34	73.82	68.27	62.13
	LambdaMART	52.04	43.00	75.77	70.49	64.04
	DLCM	52.54	43.16	77.81	71.88	65.24
	PRM-BASE	**53.63**	**43.41**	**78.62**	**72.67**	**65.72**

Table 4: Ablation study of PRM-BASE on Yahoo Letor datasets with the initial list generated by SVMRank. All the numbers in the table are multiplied by 100.

	Yahoo Letor dataset				
	P@5	P@10	MAP@5	MAP@10	MAP
DLCM	52.54	43.26	76.52	70.86	64.50
Default(b=4,h=3)	53.29	43.66	77.62	72.02	65.60
Remove PE	**52.55**	**43.56**	76.11	70.74	**64.73**
Remove RC	53.24	43.63	77.52	71.92	65.52
Remove Dropout	53.17	43.42	77.41	71.80	65.17
Block(b=1)	53.12	43.59	77.58	71.91	65.49
Block(b=2)	53.19	43.58	77.51	71.86	65.49
Block(b=6)	53.22	43.63	77.64	72.02	65.61
Block(b=8)	**52.85**	**43.32**	**77.43**	**71.65**	**65.14**
Multiheads(h=1)	53.17	43.67	77.65	71.96	65.55
Multiheads(h=2)	53.29	43.60	77.68	72.00	65.57
Multiheads(h=4)	53.20	43.61	77.72	72.00	65.58

improvements on Precision@5. The performance gain on our E-commerce Re-ranking dataset is much larger than on Yahoo Letor dataset. This is highly related with the properties of Yahoo Letor dataset. Our statistics of the Yahoo Letor dataset show that the average click through rate is 30%, which mean that for each query with 30 recommended documents, about 9 documents are clicked by the users. However, the average click-through rate in our real world E-commerce Re-ranking dataset is no more than 5%. It means that ranking on Yahoo Letor dataset is much easier than on E-commerce Re-ranking dataset. This is also confirmed by the value of MAP for the same ranking methods on two datasets: DLCM can achieve 0.64 MAP on Yahoo Letor dataset but can only achieve 0.28 MAP on E-commerce Re-ranking dataset. Combining Table 5 and Table 3, we find that the harder the ranking task, the larger improvements of our PRM model.

Table 5 shows that our PRM-Personalized-Pretrain achieves significant performance improvements comparing with PRM-BASE. PRM-Personalized-Pretrain outperforms PRM-BASE by 4.5% at MAP and 6.8% at Precision@5. This is mainly imported by the personalized vector $\boldsymbol{PV}$, which is learned by a pre-trained model whose architecture is illustrated in Figure 1 (c). PRM-Personalized-Pretrain has two advantages: (1) The pre-trained model can fully utilize longer period of users' logs to provide more generic and representative embeddings of users' preferences. (2) Equipped with long term and generic user embedding, our PRM model is able to learn better user-specific encoding function which can more precisely capture mutual influences of item-pairs for each user. Note that the architecture of the pre-trained model is not highly coupled with our PRM model, other general models[16, 23, 24, 30, 36] can also be used as alternatives to generate $\boldsymbol{PV}$.

5.6 Online Experiments

We also conduct online A/B test at a real world e-commerce recommender system on online metrics which includes PV, IPV, CTR and GMV. The meaning of these metrics is explained in the previous "Evaluation Metrics" section. These metrics evaluate how much willingness for users to view (PV), click (IPV, CTR) and purchase (GMV) in a recommender system. For each algorithm, there are hundreds of thousands of users and millions of requests for online test.

Table 6 shows the relative improvements of three methods to an online base ranker (DNN-based LTR). Firstly, the online A/B test shows that re-ranking helps increase the online metrics no matter what kinds of re-ranking methods are. Again, we can conclude that re-ranking helps improving the performance by considering the mutual influences of items in the initial list. It is noteworthy that 0.77% increase (DLCM v.s. Without re-ranking) on PV is significant in our online system because it means that about billions of extra items are viewed by the users after using the re-ranking method. Secondly, we can conclude that our PRM-BASE model brings an extra 0.50% absolute increase on viewed items and extra 0.69% absolute increase on clicked items compared with DLCM. Lastly, by using the personalized module, our PRM-Personalized-Pretrain model can further improve the GMV by 6.29% absolute increase compared with PRM-BASE. Recall that in offline experiments on E-commerce Re-ranking dataset, PRM-Personalized-Pretrain has 4.5% increase at MAP compared with PRM-BASE. The result shows that personalized encoding function with pre-trained users' representations can help capture more precise interactions of item-pairs and bring significant performance gain for re-ranking method.

5.7 Visualizing Attention Weights

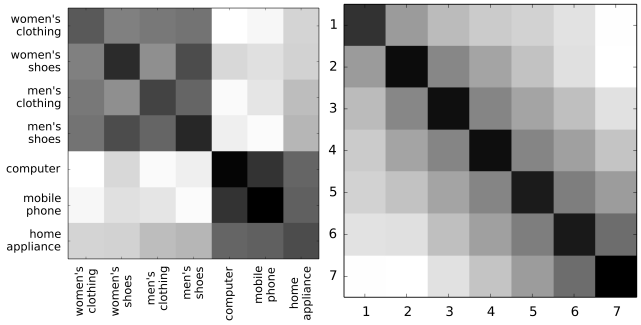
We visualize the attention weights learned by our model to answer the following question.

Table 5: Offline evaluation results on E-commerce Re-ranking dataset.

Init. List	Re-ranking	E-commerce Re-ranking dataset.				
		Precision@5	Precision@10	MAP@5(%)	MAP@10(%)	MAP(%)
DNN-based LTR	DLCM	12.21	9.73	29.32	30.28	28.19
	PRM-BASE	12.71	9.99	29.80	30.83	28.85
	PRM-Personalized-Pretrain	**13.58**	**10.52**	**31.18**	**32.12**	**30.15**

Table 6: Performance improvements in online A/B test compared with a DNN-based LTR without re-ranking method.

Reranking	PV	IPV	CTR	GMV
DLCM	0.77%	1.75%	0.97%	0.13%
PRM-BASE	1.27%	2.44%	1.16%	0.36%
PRM-Personalized-Pretrain	**3.01**%	**5.69**%	**2.6**%	**6.65**%



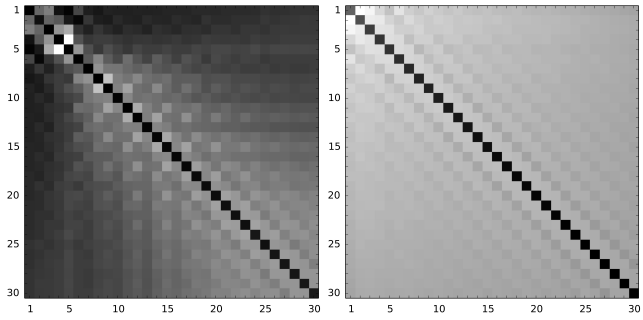
(a) Category. **(b) Price.**

Figure 2: Average attention weights related to items' attributes.

- RQ3: Can self-attention mechanism learn meaningful information with respect to different aspects, for example, positions and characteristics of items?

Attention on Characteristics. We first visualize the average attention weights between items on two characteristics: *category* and *price*. The results calculated on the test dataset are shown in Figure 2. Each block in the heatmap represents the average attention weights between items belonging to seven main categories. The darker the block, the larger the weight. From Figure 2(a) we can conclude that the attention mechanism can successfully capture mutual-influences in different categories. The items with similar categories tend to have larger attention weights, indicating larger mutual influences. For example, "men's shoes" has more influences on "women's shoes" than on "computer". It is also easy to understand that "computer", "mobile phone" and "home appliance" have large attention weights with each other because they are all electronics. Similar cases can be observed in Figure 2(b). In Figure 2(b), we classify the items into 7 levels according to their prices. The closer price between items, the larger the mutual influences.

Attention on Positions. The visualization of average attention weights on different positions in the initial list is shown in Figure 3. Firstly, Figure 3(a) showed the self-attention mechanism in our model can capture the mutual influences regardless of the encoding



(a) With position embedding. **(b) Without position embedding.**

Figure 3: Average attention weights on positions in the initial list of two PRM models: w/o position embedding.

distances as well as the position bias in recommendation list. Items ranked ahead of the list usually is more likely to be clicked and thus have more influences on those items at the tail of the list. For example, we observe that items at the first position have larger impacts on items at 30th position than those items at 26th position even though the latter is more closer to it. The effect of position embeddings is also obvious compared with the Figure 3(b), whose attention weights between each position are more uniformally distributed.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a personalized re-ranking model (PRM) to refine the initial list given by state-of-the-art learning to rank methods. In the re-ranking model, we used Transformer network to encode both the dependencies among items and the interactions between the user and items. The personalized vector can bring further performance improvements to the re-ranking model. Both the online and offline experiments demonstrated that our PRM model can greatly improve the ranking performance on both public benchmark dataset and our released real-world dataset. Our released real-world dataset can enable researchers to study the ranking/re-ranking algorithms for recommendation systems.

Our work explicitly models the complex item-item relationships in the feature space. We believe that optimization in the label space can also helps. The pair-wise or list-wise loss function aims to dig more information in label space. It is interesting to construct more pair-wise or list-wise ordering relations (e.g. order by stay time) in click-through data. Another future direction is learning to diversify by re-ranking. Even though our model does not hurt the ranking diversities in practice. It is worthy to try to introduce the goal of diversification into our re-ranking model. We will further explore this direction in the future work.

REFERENCES

[1] Qingyao Ai, Keping Bi, Jiafeng Guo, and W Bruce Croft. 2018. Learning a Deep Listwise Context Model for Ranking Refinement. *arXiv preprint arXiv:1804.05936* (2018).

[2] Qingyao Ai, Xuanhui Wang, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. Learning groupwise scoring functions using deep neural networks. *arXiv preprint arXiv:1811.04415* (2018).

[3] Irwan Bello, Sayali Kulkarni, Sagar Jain, Craig Boutilier, Ed Chi, Elad Eban, Xiyang Luo, Alan Mackey, and Ofer Meshi. 2018. Seq2Slate: Re-ranking and Slate Optimization with RNNs. *arXiv preprint arXiv:1810.02019* (2018).

[4] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 89–96.

[5] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[6] Christopher J Burges, Robert Ragno, and Quoc V Le. 2007. Learning to rank with nonsmooth cost functions. In *Advances in neural information processing systems*. 193–200.

[7] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. ACM, 129–136.

[8] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 335–336.

[9] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS 2016)*. ACM, New York, NY, USA, 7–10. https://doi.org/10.1145/2988450.2988454

[10] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[12] David Cossock and Tong Zhang. 2008. Statistical analysis of Bayes optimal subset ranking. *IEEE Transactions on Information Theory* 54, 11 (2008), 5140–5154.

[13] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 191–198.

[14] Yajuan Duan, Long Jiang, Tao Qin, Ming Zhou, and Heung-Yeung Shum. 2010. An empirical study on learning to rank of tweets. In *Proceedings of the 23rd International Conference on Computational Linguistics*. Association for Computational Linguistics, 295–303.

[15] Jerome H Friedman. 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics* (2001), 1189–1232.

[16] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-machine Based Neural Network for CTR Prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, 1725–1731. http://dl.acm.org/citation.cfm?id=3172077.3172127

[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.

[18] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 133–142.

[19] Thorsten Joachims. 2006. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 217–226.

[20] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. *arXiv preprint arXiv:1808.09781* (2018).

[21] Ping Li, Qiang Wu, and Christopher J Burges. 2008. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in neural information processing systems*. 897–904.

[22] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[23] Weiwen Liu, Ruiming Tang, Jiajin Li, Jinkai Yu, Huifeng Guo, Xiuqiang He, and Shengyu Zhang. 2018. Field-aware Probabilistic Embedding Neural Network for CTR Prediction. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA, 412–416. https://doi.org/10.1145/3240323.3240396

[24] Yanru Qu, Bohui Fang, Weinan Zhang, Ruiming Tang, Minzhe Niu, Huifeng Guo, Yong Yu, and Xiuqiang He. 2018. Product-Based Neural Networks for User Response Prediction over Multi-Field Categorical Data. *ACM Trans. Inf. Syst.* 37, 1, Article 5 (Oct. 2018), 35 pages. https://doi.org/10.1145/3233770

[25] Steffen Rendle. 2010. Factorization Machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*. IEEE Computer Society, Washington, DC, USA, 995–1000. https://doi.org/10.1109/ICDM.2010.127

[26] Gongbo Tang, Mathias Müller, Annette Rios, and Rico Sennrich. 2018. Why self-attention? a targeted evaluation of neural machine translation architectures. *arXiv preprint arXiv:1808.08946* (2018).

[27] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 77–86.

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*. 5998–6008.

[29] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. 2692–2700.

[30] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17 (ADKDD'17)*. ACM, New York, NY, USA, Article 12, 7 pages. https://doi.org/10.1145/3124749.3124754

[31] Liang Wu, Diane Hu, Liangjie Hong, and Huan Liu. 2018. Turning Clicks into Purchases: Revenue Optimization for Product Search in E-Commerce. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '18)*. ACM, New York, NY, USA, 365–374. https://doi.org/10.1145/3209978.3209993

[32] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: theory and algorithm. In *Proceedings of the 25th international conference on Machine learning*. ACM, 1192–1199.

[33] Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 391–398.

[34] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, Jean-Marc Langlois, and Yi Chang. 2016. Ranking Relevance in Yahoo Search. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 323–332. https://doi.org/10.1145/2939672.2939677

[35] ChengXiang Zhai, William W Cohen, and John Lafferty. 2015. Beyond independent relevance: methods and evaluation metrics for subtopic retrieval. In *ACM SIGIR Forum*, Vol. 49. ACM, 2–9.

[36] Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. 2016. Deep recurrent models with fast-forward connections for neural machine translation. *arXiv preprint arXiv:1606.04199* (2016).

[37] Tao Zhuang, Wenwu Ou, and Zhirong Wang. 2018. Globally Optimized Mutual Influence Aware Ranking in E-Commerce Search. *arXiv preprint arXiv:1805.08524* (2018).