Personalized Context-aware Re-ranking for E-commerce Recommender Systems

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

ABSTRACT

Ranking is a core task in E-commerce 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 context-aware re-ranking model for E-commerce recommender systems. The proposed re-ranking model can be easily deployed as a follow-up modular after ranking by directly using the existing feature vectors of ranking. 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. Besides, we introduce the personalized embedding to model the differences between feature distributions 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-rank, 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[1], Wenwu Ou[1], Dan Pei[3]. 1997. Personalized Context-aware Re-ranking for E-commerce Recommender Systems . In *Proceedings of ACM Woodstock conference, France, 2019 (SIGIR19),* 11 pages.
DOI: 10.475/123_4

[*]Changhua Pei and Yi Zhang are equal contribution. Yongfeng Zhang is the corresponding author.

SIGIR19, France

DOI: 10.475/123_4

1 INTRODUCTION

Ranking is crucial in E-commerce recommender systems. The quality of the ranked list given by a ranking algorithm has a great impact on users' satisfactory as well as the revenue of E-commerce recommender systems. A large amount of ranking algorithms [5, 6, 8, 17, 22, 31, 36] have been proposed to optimize the ranking performance. Typically ranking in recommender system only considers the user-item pair features, without considering the local context information from other items in the list, especially by those items placed alongside[9, 39]. 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, 38, 41] 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 in feature space. The state-of-the-art methods for encoding the feature vectors are RNN-based, such as GlobalRerank[41] 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[23] 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, personalization is another important factor to be considered for re-ranking in E-commerce 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 price feature should be more important in the re-ranking model. Typical global ranking strategy may be not optimal as it ignores the differences between feature distributions for each user. In addition, the interaction between items is affect by user as well. For instance, when users are in the intention of price comparison, similar items with different prices

tend to be more aggregated in the list. When the user has no obvious shopping 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 items. The interaction between items in the list and user can be captured simultaneously in our personalized context-aware 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**. In our personalized context-aware re-ranking approach, 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 in one of the largest online shopping mobile App. Each record in the dataset contains a recommendation list for each user with users' basic information, 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 online shopping mobile App.

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*[13, 25], *pairwise*[7, 21, 22], and *listwise*[6, 8, 16, 22, 31, 36, 37]. To apply the learning to rank into the recommendation system, the *pointwise* method uses a single item as input to predict a score for ranking. The *pairwise* method taking two items as a pair to classify which item should be ranked first. The *listwise* method uses the whole list as input and judges the likelihood possibility of the full permutation of items. They 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 aware of the interactions not only between items but also between users and items.

Closed to our work are [1, 2, 4, 41], 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[11] to encode the information of the whole list into the representation of each item. [41] uses LSTM[20] and [4] uses pointer network[33] not only to encode the whole list information, but also to generate the ranked list using a decoder. For those methods which use either GRU or LSTM to encode the dependencies of items, the capacity of the encoder is limited. This is because the RNN network encodes the items in a sequentially way according to their positions in the initial list. Thus the item at the end of the initial list may have little influence to the item at the head of the initial list, which is not true in reality. 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 using 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 inferences, where n is the length of the output list. [2] propose 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.

Besides the above ranking and re-ranking methods, our work is inspired by the following works. [32] proposed Transformer which is widely used in machine translation as its effectiveness in encoding high-order and long distance dependencies in sequential list compared with RNN/CNN-based approaches[35, 40]. FPMC[18, 28] are personalized recommendation methods which learn user embedding from user's click history. YouTubeRec[14] and Bert[15] confirm that it is feasible to use the pre-trained embeddings to enhance certain task-specific model in practice.

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 FORMULATION OF RE-RANKING

In this section, we first give some preliminary knowledge about learning to rank and re-ranking methods in E-commerce recommendation system. Then we formulate the problem we aim to solve in this paper. The notation used in this paper is in Table 1.

Learning to rank (often labelled as **LTR**) method is widely used for ranking in real-work system to generate an ordered list for information retrieval[21, 26] and recommendation[16]. In E-commerce recommendation, 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 formulated as the following Equation:

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

where $\mathcal{R}$ is the set of all users' requests in 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 model with parameters θ. ℓ is the loss computed with the label y_i and the corresponding estimated possibility. The goal of LTR is to find the best parameters θ that minimize the loss function $\mathcal{L}$.

However, $\boldsymbol{x_i}$ is not enough to learn a good scoring function. We find that ranking in E-commerce recommender system should consider the following information: (a) dependencies between item pairs[9, 39]; (b) interactions between user and items. The dependencies between item pairs can be directly learned from the initial list $\mathcal{S} = [i_1, i_2, ..., i_n]$. We refer this kind of information as **local context information**, which is labeled as $\boldsymbol{X} = [\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, ..., \boldsymbol{x}_{i_n}]$ in this paper. Works[1][41][2][4] propose approaches to make better use of local context information. However, few works consider the interactions between the user and items. The specific information for user-item pairs represents the preferences of users on different aspects of items and can not be learned directly from initial list. In this paper, we introduce a personalized vector $\boldsymbol{pv}_i$ for each item i to represent the user's preference on latent characteristics of item i. For example, some people may be sensitive to price of items while others may be sensitive to brand of items. We refer this kind of information as **global context information**, which is labeled as $\boldsymbol{PV} = [\boldsymbol{pv}_{i_1}, \boldsymbol{pv}_{i_2}, \ldots, \boldsymbol{pv}_{i_n}]$ in this paper.

We propose a personalized context-aware model to re-rank the initial list by utilizing both *local* and *global* context information. Compared with the global scoring function learned by LTR, our re-ranking model seeks to learn a user-specific scoring function which is able to weight features distinctly for different users. 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 ranking model. $\hat{\theta}$ is the parameters of re-ranking model. Our personalized context-aware model is different from the typical ranking approaches in the following aspects: (a) We use a ordered list $\mathcal{S}_r$ instead of a candidate set $\mathcal{I}_r$. (b) When learning the scoring function for each item i, we use the total feature matrix $\boldsymbol{X}$ of all items in the list instead of one single feature vector $\boldsymbol{x_i}$ to model the mutual influences between items. (c) We use a personalized matrix $\boldsymbol{PV}$ to learn the interactions between user and items. Note that both $\boldsymbol{X}$ and $\boldsymbol{PV}$ can be used by the existing ranking approaches to achieve better performance.

4 PERSONALIZED CONTEXT-AWARE RE-RANKING MODEL

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

4.1 Model Architecture

The architecture of PCRM 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 LTR method as input and outputs a re-ranked list. In input layer, we utilize the interaction between user and each item i to generate a personalized vector $\boldsymbol{pv_i}$. $\boldsymbol{pv_i}, i = i_1, ..., i_n$ is the personalized vector of user u on item i which represents the user's preference on latent characteristics of item i. We concat $\boldsymbol{pv_i}$ with raw feature vector $\boldsymbol{x_i}$, then add it with a position embedding $\boldsymbol{pe}_i$ to get the input vector for encoding layer. The encoding layer is a Transformer-like encoder made up of multi-blocks, each of which contains a multi-head attention module and a feed forward network (abbreviated as FFN), the detailed structure is shown in Figure 1 (a). After encoding layer, the model generates a score for each item using a *softmax* function. The detailed structure will be introduced separately in the following sections.

4.2 Input Layer

The function of this layer is to make a comprehensive representation of all input features and feed it to the next layer. First we have a fixed length of initial sequential list $\mathcal{S} = [i_1, i_2, ..., i_n]$ given by the learning to rank method. Same as the learning to rank method, we have a 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 $i \in \mathcal{S}$.

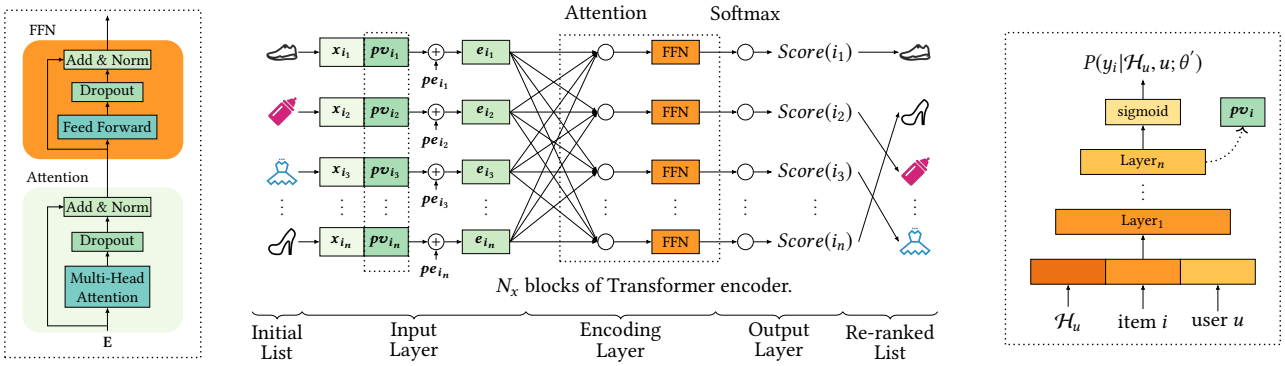
Personalized Vector. As shown in Figure 1 (b), we concat the raw feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d_{\text{feature}}}$ with a personalized matrix $\boldsymbol{PV} \in \mathbb{R}^{n \times d_{\text{pv}}}$ to get the intermediate embedding matrix $\boldsymbol{E}' \in \mathbb{R}^{n \times (d_{\text{feature}} + d_{pv})}$, which is shown in Equation 3. $\boldsymbol{PV}$ is produced by a pre-trained model which will be introduced in the following section.

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

Position Embedding. 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 led to slightly outperform the fixed position embedding used in [32].

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

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



(a) One block of Transformer encoder.

(b) Architecture of PCRM.

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

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

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 function of encoding layer in Figure 1(a) is to learn a high-level representation for item list $\mathcal{S}$ by combining both local and global context information. Transformer[32] has been proved to be effective in many NLP tasks, specially machine translation for its powerful encoding and decoding ability compared to RNN-based approaches[11, 12, 20]. Self-attention mechanism in Transformer is particularly suitable in our re-ranking task as it directly models the mutual influence for any two items regardless the distance between them. Without the decay with distance, Transformer can capture more interactions between items which are far away from each other in the initial list. As shown in Figure 1(b), our encoding module consists of N_x blocks. Each block (in Figure 1(a)) contains a multi-head attention layer and Point-wise Feed-Forward Network (FFN) layer.

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

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

where matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ represent queries, keys and values respectively. d_K 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. Each row in $\boldsymbol{E}$ is corresponding to each item in the ranking list. $\boldsymbol{E}$ contains not only the features of items but also the interaction between user and items.

In our model, any two items can attention with each other. The self-attention result of each item is the weight sum of the all the items in initial list. Compared with Language Model[27] and SASREC[23], which use the masked self-attention by preventing the afterward item i_{n+1} from being encoded into the representation of the item i_n. This is because that [23] focuses on next bucket recommendation: given a sequence of clicked items by the user from 0$^{\text{th}}$ time slot to t-1$^{\text{th}}$ time slot, the algorithm predicts which item the user will click at t^{th} time slot. Thus it is not allowed to leak the information of ground truth (item at at t^{th} time slot). However, in re-ranking settings, the information of all the items can be used.

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.

Moreover, to improve the generality of the model and make the model stable for training, we apply the dropout layer (abbreviated as *dropout*) and residual connections (abbreviated as *RC*). The dropout is proposed by [29] to address the problem of overfitting. The residual connections is used to alleviate the gradients vanishing problem as it directly propagates the raw input of lower layer to higher layer and adds with the self-attention[19] results. The RC is an essential part when we stacks multiple blocks of Transformer encoder. The ablation study in the next section will compare the performance when removing these two components separately. In total, the attention layer can be formulated as Equation 8.

$$\boldsymbol{S} = \text{LayerNorm}\big(\boldsymbol{E} + \text{Dropout}(\text{MH}(\boldsymbol{E}))\big), \tag{8}$$

where *LayerNorm* is the standard normalization layer proposed by[3].

Position-wise 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. Similar to *Attention* module, we also use *Dropout* to avoid overfitting and *residual connection* to enhance the power of the model by hierarchically learning. *LayerNorm* is

normalization which is effective on accelerating and stabilizing the training process. Finally, the output of the FFN layer can be calculated using Equation 9:

$$\boldsymbol{F} = \text{FFN}(\boldsymbol{S}) = \text{LayerNorm}\Big(\boldsymbol{S}+\text{Dropout}\big(\text{RELU}(\boldsymbol{S}\boldsymbol{W}^{(1)}+b^{(1)})\boldsymbol{W}^{(2)}+b^{(2)}\big)\Big), \tag{9}$$

where $\boldsymbol{W}^{(1)}$, $\boldsymbol{W}^{(2)}$, $\boldsymbol{b}^{(1)}$ and $\boldsymbol{b}^{(2)}$ are learnable parameters.

Stacking the Encoding Layer. Here we use attention module followed by the position-wise FFN as a block of Transformer[32] encoder. After encoding by one block, the output of encoding layer for item i (abbreviated as $\boldsymbol{F}_i^{(0)}$) has already contained the interaction between different items. We believe that taking $\boldsymbol{F}_i^{(0)}$ as the input of encoding layer to generate $\boldsymbol{F}_i^{(1)}$ is helpful. $\boldsymbol{F}_i^{(1)}$ contains more complex and high-order mutual information. In the next section, we study how many blocks is best for our task. The output of the encoding layer after stacking N_x blocks can be represented as Equation 10.

$$\boldsymbol{F}^{(\boldsymbol{N_x})} = \text{FFN}(\boldsymbol{S}^{\boldsymbol{N_x}}) = \text{FFN}\Big(\text{SA}(\boldsymbol{F}^{(N_x-1)})\Big) \tag{10}$$

where N_x is the stacking number of the encoding layer. Generally, N_x is larger than 0 since it is obvious that the embedding vector would directly connect to the output layer when $N_x = 0$. So the model reduces to a two-layer feed-forward network and the performance would greatly degrade.

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 one linear layer followed by a softmax layer. The output of softmax layer is the probabilities 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 $\boldsymbol{O}(1)$ time complexity. The formulation of $Score(i)$ is shown in Equation 11.

$$\begin{aligned} Score(i) &= P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta}) \\ &= \text{softmax}\Big(\boldsymbol{F}^{(\boldsymbol{N_x})}\boldsymbol{W}^F + \boldsymbol{b}^F\Big), i \in \mathcal{S}_r, \end{aligned} \tag{11}$$

where $\boldsymbol{W}^F$ is learnable projection matrix, and $\boldsymbol{b}^F$ is 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 12. We update the parameters $\hat{\theta}$ of the **PCRM** model by *Adam* optimizer[24].

$$\mathcal{L} = -\sum_{r\in\mathcal{R}}\sum_{i\in\mathcal{S}_r} y_i \log(P(y_i|\boldsymbol{X}, \boldsymbol{PV}; \hat{\theta}) \tag{12}$$

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 PCRM model in an end-to-end manner on the re-ranking loss. However, as explained in Section3, the re-ranking task is to refine of the output of learning to rank approaches. The task-specific representation learned on re-ranking task lacks users' global preference. Therefore, we utilize a pre-trained neural network to produce user's personalized embeddings $\boldsymbol{PV}$ which are then used as features for PCRM model. The pre-trained neural network is learned from the whole click-through logs in e-commerce platform. Figure 1(c) shows the structure of pre-trained model in our paper. This sigmoid layer outputs the click probability on item i for user u ($P(y_i|\mathcal{H}_u, u; \theta')$) given user's all behavior history ($\mathcal{H}_u$). $P(y_i|\mathcal{H}_u, u; \theta')$ is the predicted click probability. $\mathcal{H}_u$ is the sequence of items clicked by the user u. The side information of user is also used as input, *e.g., gender, age, purchasing level*, etc. The loss of the model is calculated by a point-wise cross entropy function which is shown in Equation 13.

$$\begin{aligned} \mathcal{L} = \sum_{i\in\mathcal{D}} &(y_i \log(P(y_i|\mathcal{H}_u, u; \theta')) \\ &+ (1-y_i)\log(1-P(y_i|\mathcal{H}_u, u; \theta')), \end{aligned} \tag{13}$$

where $\mathcal{D}$ is the set of items displayed to user u within the whole clickthrough logs in e-commerce 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[14], we employ the hidden vector before the *sigmoid* layer as the personalized vector $\boldsymbol{pv_i}$ (in Figure 1(c)) that feeds into our PCRM model.

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 PCRM 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 e-commerce 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 fit for the ranking model of recommendation using the same method in Seq2Slate[4]. 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 limit number of items, the bigger the ranked position of one item, the smaller 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 and η is the hyper parameter.

[1] http://webscope.sandbox.yahoo.com

[2] Our dataset is available at https://github.com/rank2rec/rerank. The code of this paper will be released when the paper is published.

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}

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 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[22]: This is a representative learning to rank method which use the pairwise loss to model the scoring function.
- LambdaMart[6]: 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 [34]'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[10] 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[4] and GlobalRerank[41]. DLCM[1] and GlobalRerank[41] focus on re-ranking in information retrieval. Seq2Slate[4] 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 14.

$$Precision@k = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \frac{\sum_{i=1}^{k} \mathbb{1}(\mathcal{S}_r(i))}{k} \tag{14}$$

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. $\mathbb{1}$ 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.

$$MAP@k = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \frac{\sum_{i=1}^{k} Precision@i * \mathbb{1}(\mathcal{S}_r(i))}{k} \tag{15}$$

For online A/B tests, 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

Training Dataset. For both Yahoo Letor dataset and E-commerce Re-ranking dataset, the length of the initial list given by LTR is 30. When preprocessing the Yahoo Letor dataset for recommendation, the decay parameter $\eta = 0.2$ and threshold T_b used to generate binary labels is 1.5. Same number of features are used by all the algorithms on each dataset to ensure the fairness of comparison.

Model. For both baselines and our PCRM 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 PCRM model is the same with [32]. Negative log likelihood loss function is used as shown in Equation 12. 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. We use 4 blocks of transformer with 3 multi-heads. We also tried different number of blocks and multi-heads to see the performance variances by ablation study. The rest 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 tests. We also conduct the ablation study to help finding which part of our PCRM 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:

- Q0: Does our PCRM model outperform the state-of-the-art methods and why?
- Q1: 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 PCRM-BASE model based on two different initial lists which are generated by LambdaMART and SVMRank respectively. PCRM-BASE is the variant of our PCRM model without the personalized module. Note that Yahoo Letor dataset does not contain user-related information, thus we only conduct PCRM-BASE for comparison. SVMRank and LambdaMart are also used for re-ranking. For SVMRank, we use the implementation in [22]. For LambdaMart, we use the implementation from RankLib[3].

Table 3 shows that our PCRM-BASE achieves stable and significant performance improvements comparing with all baselines. When based on the initial list generated by SVMRank, PCRM-BASE outperforms DLCM by 1.7% at MAP and 1.6% 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 n the initial list generated by LambdaMART, PCRM-BASE outperforms DLCM by 1.1% at MAP and 0.2% at Precision@5. PCRM-BASE also achieves 3.1% improvements on MAP and 1.3% improvements on Precision@5 comparing with LambdaMART .

PCRM-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 self-attention mechanism is more good at modeling mutual influence between two vectors, especially when the length of encoding list gets longer[23]. In our model, the attention mechanism can model the interactions of any item-pairs in $\mathbf{O}(1)$ encoding distance.

As PCRM-BASE uses the Transformer-like structure, there are many sub-modules which may contribute to the performance gain. 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 PCRM-BASE model. The third part (the rest rows) shows different ablation variants of our PCRM 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" PCRM model.

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. By 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 PCRM-BASE still achieve comparable performance with DLCM, which further confirms that our PCRM-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[30]. 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) [30] indicates that using more heads is helpful in encoding the information of long sequence. This is resonable 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 and the average length of the initial list is 18. There are minor improvements when mapping the original feature vector into different subspaces. Besides, our sequence is not long enough to use multiple heads. 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.

- Q2: What is performance of our PCRM model equipped with personalized module?

The evaluation results are shown in Table 5. We compare different variant PCRM model with DLCM baseline. For our PCRM models, we not only Waeluatethe performance of PCRM-BASE, but also evaluate the performance of the variant of model equipped with the pre-trained personalized vector $\boldsymbol{PV}$, which is labelled as PCRM-Personalized-Pretrain. do not compare wevith SVMRank and LambdaMART because the public released implementations can not handle our large dataset and out-of-memory error happens. As our previous evaluation on Yahoo Letor dataset already confirms that DLCM achieve better performance in all metrics and [1] has consistent results, we omit the comparison with SVMRank and LambdaMART on our E-commerce Re-ranking dataset. The initial

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

Table 3: Offline evaluation results on Yahoo Letor dataset. The p-value of significant test is below 0.005.

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
	PCRM-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
	PCRM-BASE	**52.63**	**43.41**	**78.62**	**72.67**	**65.72**

Table 4: Ablation study of PCRM-BASE on Yahoo Letor datasets with the initial list generated by SVMRank. All the numbers in the table are multiplied by 100. The p-value of significant test is below 0.005.

	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

list is generated by a DNN-based LTR method which is deployed in our real world recommender system. All the re-ranking methods listed in Table 5 use the same training dataset, which is also the same with DNN-based LTR method. The first row in Table 5 shows the performance of the DNN-based LTR without any re-ranking methods. The numbers in the brackets are the related improvement compared with DNN-based LTR without re-ranking.

Table 5 shows consistent results with Table 3 when comparing with PCRM-BASE and DLCM. Our PCRM-BASE outperforms DLCM by 2.34% at MAP and 4.1% at Precision@5. Recall that on Yahoo Letor dataset, PCRM-BASE achieves 1.7% improvements on MAP and 1.6% 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. As mentioned in the previous section, in order to simulate the training data in recommendation, we convert the ratings ({0,1,2,3,4}) into binary label (relevant and irrelevant) to simulate click-through data. Our statistics of the converted 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 clickthrough 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 model.

Table 5 shows that our PCRM-Personalized-Pretrain achieves significant performance improvements comparing with PCRM-BASE. PCRM-Personalized-Pretrain outperforms PCRM-BASE by 4.5% at MAP and 6.8% at Precision@5. This is mainly imported by the global context information ***PV***, which is learned by a pre-trained model whose architecture is illustrated in Figure 1 (c). PCRM-Personalized-Pretrain has three strengths: (1) The pre-trained model can use longer period of logs which is helpful in learning long-term user interest. (2) The pre-trained model can use logs from the whole e-commerce platform, which is not limited in the recommendation logs. (3) The personalized embeddings learned by pre-trained is more generic to represent user's preference on items.

5.6 Online Experiments

We also conduct online A/B tests 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. The online A/B tests last for one day. For each algorithm, there are hundred thousand users and millions of requests for online test.

Table 6 shows the relative improvements of three methods to an online base ranker. Firstly, the online A/B test shows that using re-ranking helps increasing 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 local context information. It is note that 0.7% increase on PV is significant in our large-scale online system because it means that about billions of extra items is viewed by the users after using the re-ranking method. Secondly, we can conclude that our PCRM-BASE model brings an extra 0.5% increases on viewed items and extra 2.44% increases on clicked items and extra 0.23% increases on the total GMV. Lastly, by using the personalized module, our PCRM-Personalized-Pretrain

Table 5: Offline evaluation results on E-commerce Re-ranking dataset. The p-value of significant test is below 0.005.

Init. List	Re-ranking	E-commerce Re-ranking dataset.				
		Precision@5	Precision@10	MAP@5(%)	MAP@10(%)	MAP(%)
DNN-based LTR	Without Re-ranking	12.09	9.53	29.06	29.99	27.79
	DLCM	12.21(0.99%)	9.73(2.10%)	29.32(0.89%)	30.28(0.97%)	28.19(1.44%)
	PCRM-BASE	12.71(5.13%)	9.99(4.83%)	29.80(2.55%)	30.83(2.80%)	28.85(3.81%)
	PCRM-Personalized-Pretrain	**13.58(12.32%)**	**10.52(10.39%)**	**31.18(7.3%)**	**32.12(7.1%)**	**30.15(8.49%)**

Table 6: Performance improvements in online A/B test compared with a DNN-based LTR without re-ranking method. The p-value of significant test is below 0.005.

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

can further improve the GMV by 6.31%. Recall that in offline experiments on E-commerce Re-ranking dataset, PCRM-Personalized-Pretrain has 4.5% extra increases compared with PCRM-BASE. The result shows that personalization with pre-trained user's representation can affect the interaction of item-item, and bring significant gain for re-ranking.

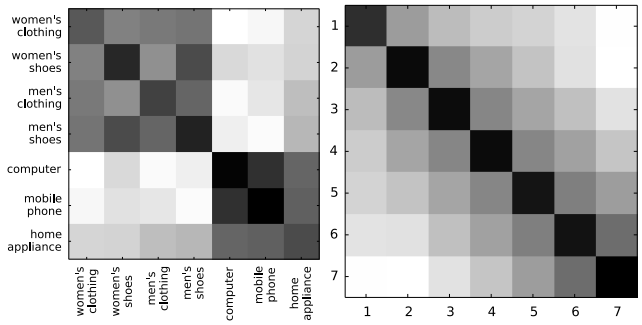
5.7 Visualizing Attention Weights

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

- Q3: Can self-attention learn meaningful information in respect to different aspects, for example, positions and characteristics of items?

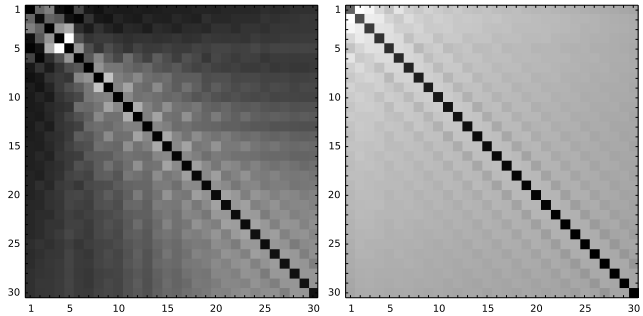
Attention on Characteristics. We first visualize the average attention weights between items on two characteristics: *category* and *price*. The results caculated on the test dataset are shown in Figure 2. Each block in the heatmap represents the average attention weights between items belonging to seven main cateogries. The darker the block, the larger the weight. From Figure 2(a) we can conclude that the attention mechanism can successfully capture mutual-influence 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 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 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



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

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



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

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

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 context-aware re-ranking model (PCRM) to refine the initial list given by state-of-the-art learning to rank methods. The introduction of personalized information into the re-ranking model can bring significant performance improvements. In the re-ranking model, we used Transformer network to encode both the dependencies among items and the interactions between the user and items. Both the online and offline experiments demonstrated that our PCRM model can greatly improve the ranking performance on either public benchmark dataset or our released dataset. In addition, the new released dataset can help researchers to optimize the ranking algorithms for e-commerce recommendation.

Our work explicitly models the complex item-item relationships in feature space. We believe that optimization in label space can also helps. The pairwise or listwise loss function aims to dig more information in label space. However, limited improvements are achieved in our experiments for re-ranking. The labels of click-through data in the paper is click or not. It is interesting to construct more partial ordering relations (e.g. order by stay time) in click-through data. Another future direction is learning to diversify by re-ranking. Even thougth our model does not hurt the ranking diversity in practice. It is worth 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] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *CoRR* abs/1607.06450 (2016).

[4] 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).

[5] 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.

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

[7] 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.

[8] 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.

[9] 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.

[10] 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

[11] 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).

[12] 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).

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

[14] 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.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[16] 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.

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

[18] Mihajlo Grbovic and Haibin Cheng. 2018. Real-time Personalization Using Embeddings for Search Ranking at Airbnb. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, New York, NY, USA, 311–320. https://doi.org/10.1145/3219819.3219885

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

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

[21] 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.

[22] 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.

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

[24] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[25] 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.

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

[27] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of ICML*. 1310–1318.

[28] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*. ACM, 811–820.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.

[30] 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).

[31] 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.

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

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

[34] 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

[35] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).

[36] 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.

[37] 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.

[38] 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

[39] 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.

[40] 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).

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