entropy

MDPI

Article

A List-Ranking Framework Based on Linear and Non-Linear Fusion for Recommendation from Implicit Feedback

Buchen Wu [1,2] and Jiwei Qin [1,2,*]

[1] School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China; buchen_wu@stu.xju.edu.cn
[2] Key Laboratory of Signal Detection and Processing, Xinjiang Uygur Autonomous Region, Xinjiang University, Urumqi 830046, China
* Correspondence: jwqin@xju.edu.cn

Abstract: Although most list-ranking frameworks are based on multilayer perceptrons (MLP), they still face limitations within the method itself in the field of recommender systems in two respects: (1) MLP suffer from overfitting when dealing with sparse vectors. At the same time, the model itself tends to learn in-depth features of user–item interaction behavior but ignores some low-rank and shallow information present in the matrix. (2) Existing ranking methods cannot effectively deal with the problem of ranking between items with the same rating value and the problem of inconsistent independence in reality. We propose a list ranking framework based on linear and non-linear fusion for recommendation from implicit feedback, named RBLF. First, the model uses dense vectors to represent users and items through one-hot encoding and embedding. Second, to jointly learn shallow and deep user–item interaction, we use the interaction grabbing layer to capture the user–item interaction behavior through dense vectors of users and items. Finally, RBLF uses the Bayesian collaborative ranking to better fit the characteristics of implicit feedback. Eventually, the experiments show that the performance of RBLF obtains a significant improvement.

Keywords: multilayer perceptrons; collaborative filtering; list ranking

Citation: Wu, B.; Qin, J. A List-Ranking Framework Based on Linear and Non-Linear Fusion for Recommendation from Implicit Feedback. *Entropy* **2022**, *24*, 778. https://doi.org/10.3390/e24060778

Academic Editors: Andrea Prati, Luis Javier García Villalba and Vincent A. Cicirello

Received: 30 April 2022
Accepted: 28 May 2022
Published: 31 May 2022

1. Introduction

Many experiments have shown that deep neural networks (DNNs) are used in several fields because of their ability to capture complex and deeper information, including image segmentation [1], natural language processing [2,3], speech recognition [4], and recommendation systems [5–7]. Dailing Zhang et al. [8] designed deep learning-based frameworks that consist of both convolutional and recurrent neural networks to precisely identify human intentions in brain–computer interfaces. Kaixuan Chen et al. [9] proposed a semisupervised deep model for imbalanced activity recognition and pattern-balanced co-training for extracting and preserving the latent activity patterns to improve the robustness of co-training on imbalanced data. Minnan Luo et al. [4] exploited a novel semisupervised feature selection method through incorporating the exploration of the local structure to simultaneously learn the optimal graph. The primary assumption underlying the model is that the instances with similar labels should have a larger probability of being neighbors. With the large-scale application of neural networks in recommendation systems, it was found that neural networks that can fit most functions [10] are better than matrix factorization in extracting implicit user information. For instance, generative adversarial networks have gained increasing attention in the recommendation systems. CFGAN [11] and LARA [12] are pioneering methods to prove the potential of generative adversarial networks in recommendation systems. DeepCF [13] combines representation learning into a framework to overcome their disadvantages. A novel neural cooperative filtering (NCF) [14] framework based on deep learning directly learned user ratings of items and used MF and MLP to fit linear and nonlinear interactions. Ding, [15] et al. proposed

the LMDB framework, which used modular functions to model the relevant attributes of each item and used discrete functions to describe the diversity attributes of the item set. Qiu, [16] et al. used content-rich domains to complement user representation and introduced user encoders and comment encoders to model the user's behavior. Liu, [17] et al. found that directly fusing various types of side information into item embeddings brought less negative impact and better performance on the model.

Although all of these methods have achieved good results, they have different problems in user preference modeling and in terms of rating prediction. In terms of user preference modeling models, they either focus only on shallow information or only on in-depth information. Meanwhile, models based on MLP alone can easily fall into the overfitting problem and lose plenty of low-rank features. In terms of rating, there are many 0/1 ratings in implicit feedback, and the model has difficulty specifying the ranking order of these items with the same rating, thus limiting the model performance.

To solve these two problems, we propose a column label ordering architecture called RBLF that combines linear-nonlinear features with Bayesian coordination ranking for the first time. We set the potential features of user and item to different size dimensions to make the model more realistic. We explicitly fused user-embedding vectors and item-embedding vectors to learn the shallow linear interaction, feeding the obtained results into MLP to enhance the non-linearity of the model. The model fits the user–item feature relationship more comprehensively to guide the prediction method to better learn the user's true preference attributes, thus giving more accurate rating prediction results. In the list ranking, we used Bayesian collaborative ranking, called deep-setrank [18], to better fit the features of implicit feedback in reality.

RBLF focuses on predicting the exact ranking of each item rather than the accuracy of specific scores. In practice, users mostly try to click the music ranked first in the list. Therefore, the ultimate goal of RBLF is list ranking. Furthermore, since there are far more implicit data (indirectly reflecting user preferences) in the real world than explicit feedback, this results in low cost of data collection. Our main contributions are as follows:

We propose a novel architecture to accommodate linear and non-linear interaction feature vector processes and design a neural network-based list-learning RBLF framework.

We use the deep-setrank list ranking and two other traditional list ranking algorithms. We compare these three ranking algorithms and conclude that our ranking algorithm is the best.

We explore the impact of the shallow and deep interaction behavior on feature grabbing layers.

We conduct many experiments on three datasets and show that RBLF greatly outperforms other recent algorithms.

2. Preparations

2.1. List-Ranking Methods

Many collaborative filtering technologies have been used in recommendation systems [19,20], and matrix factorization (MF) [21] has gained industrial recognition since it came out on top in the Netflix-Prize competition, creating many derivative models. Their prominent examples include a large number of enhancing MF, such as using them with biases [6] and extending them with the implicit parameters [22] to achieve universal feature modeling. Wang H, et al. [23] used feature matrices for capturing implicit user–item interactions [24]. Koren Y, et al. [25] used time-series matrix factorization to capture user preferences over time.

However, high-quality prediction accuracy and high-quality sorting recommendations are not strongly correlated [26,27]; thus, MF needs to be modified when applying it to the list sorting model. For example, Wu et al. [28] combined a learned ranking algorithm for lists with matrix factorization (MF) by modifying the loss function to linearly relate the observed scores of a given user–item matrix. Shi Y et al. [29] developed MF in the scenarios with binary relevance data.

There is little user–item interaction information because of the sparse rating matrix, and the performance of MF, which can be hindered by simply choosing the interaction function (inner product) [30], is poor in most cases. As the user rating matrix expands, simply linear combinations of the product of potential features cannot learn the users' non-linear preferences. At the same time, they cannot effectively learn the implicit feature between users and items, which results in their poor ability to simulate implicit user feedback.

In recent years, DNNs, which can theoretically model any function, have been used in recommendation systems. Bai, et al. [31] created the NNCF model to capture localization information that traditional latent factor models cannot capture effectively, which integrates neighbor information as input into DNNs.

2.2. Implicit Feedback

Explicit feedback refers to numerical feedback with explicit rating criteria, such as the five-star rating system for MovieLens movies. Implicit feedback contains only positive and unobserved samples. Common implicit feedback includes click history, purchase history, like history, etc.

Given a recommendation problem, we suppose there is a series of users i and a series of items j. The recommendation task focused on in this paper requires both explicit feedback and implicit feedback from users on the items. Implicit feedback y_{ij} is expressed as predicting whether there is an interaction and is defined as follows:

$$y_{ij} = \begin{cases} 1 & user\ i\ has\ interaction\ on\ item\ j \\ 0 & otherwise \end{cases} \tag{1}$$

Here, y_{ij} is 1, indicating that there is a click or browsing behavior between user i and item j. However, that does not mean that i truly likes j. Similarly, y_{ij} is 0, which does not indicate that i is extremely averse to j, or that the user has not browsed and clicked on these items. Although the observed items at least reflect the user's specific interests, the unchecked items may simply lose data.

In daily life, most users feedback is implicit, and the implicit feedback data will have a large number of identical items; this is a challenge for the list sorting model based on implicit feedback.

2.3. MLP

MLP is essentially a neural network. It aims to solve the nonlinear question that the single-layer perceptron cannot solve. The MLP model is defined as follows:

$$\phi^{MLP} = \sigma\left(h^T \phi_L(wx + b)\right) \tag{2}$$

The structure of MLP is improved in two aspects from the single-layer perceptron structure:

(1) The hidden layer is added, which can have multiple layers to enhance the model's expressive ability. However, it also increases the complexity of the model.

(2) To extend the activation function, although the activation function of the perceptron is simple, the processing capacity is limited. Thus, other activation functions are generally used in the neural network. By using different activation functions, the expression ability of the neural network is further enhanced.

For the activation function, sigmoid has limited the performance and caused a overfitting problem. Although tahn is applied more, it does not solve the above issues. The relu function is simple, has a fast-fitting speed, and will not cause oversaturation problems. We used the relu function.

$$Relu(x) = \max(0, x) \tag{3}$$

3. The Experimental Model

Figure 1 illustrates the overall framework of RBLF, Figure 2 shows the shallow-deep interaction grabbing layer structure of RBLF, and Figure 3 shows the idea of working with three different sorting algorithms. The second subsection details the interaction capture layer and describes how the layer works and the formula expression. In the third subsection, we describe in detail the formulas used by the four different sorting algorithms and how the results are sorted.

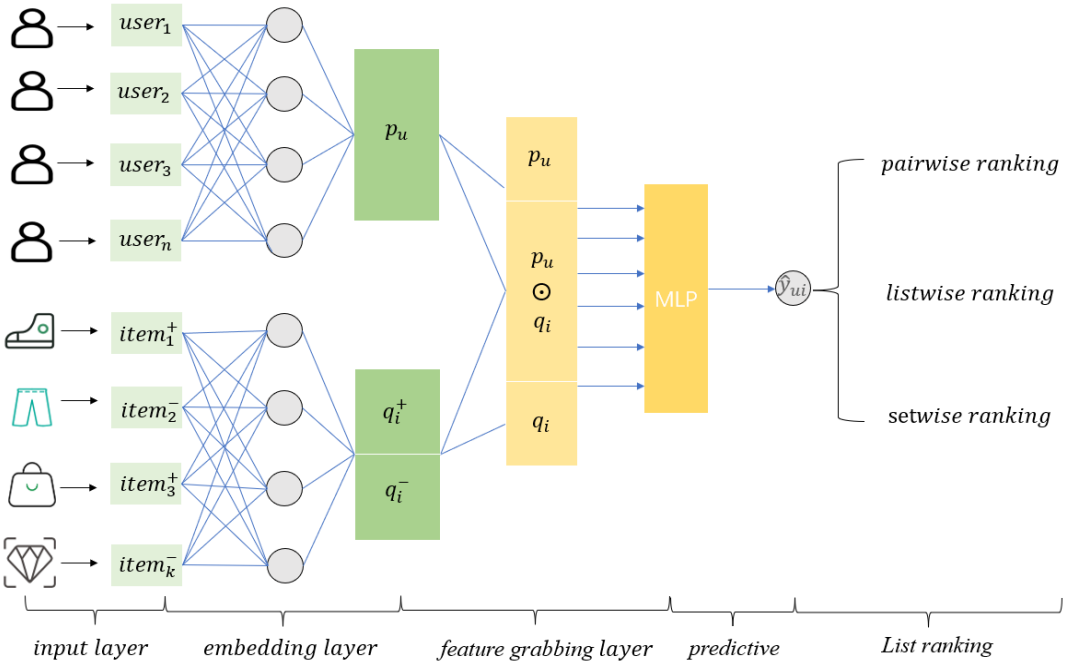


Figure 1. The architecture of RBLF.

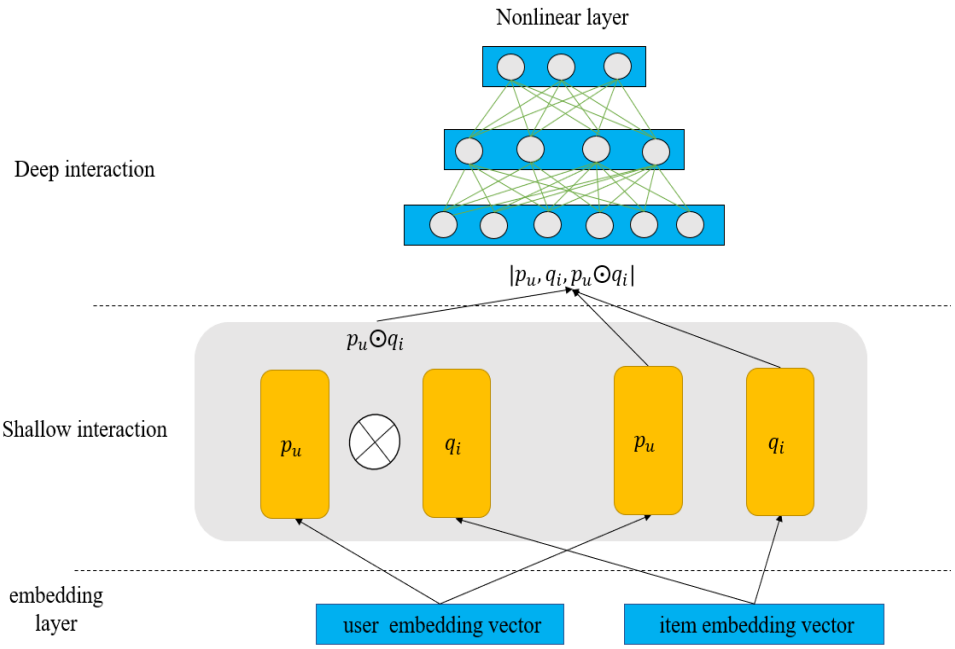


Figure 2. The architecture of interaction grabbing layer.

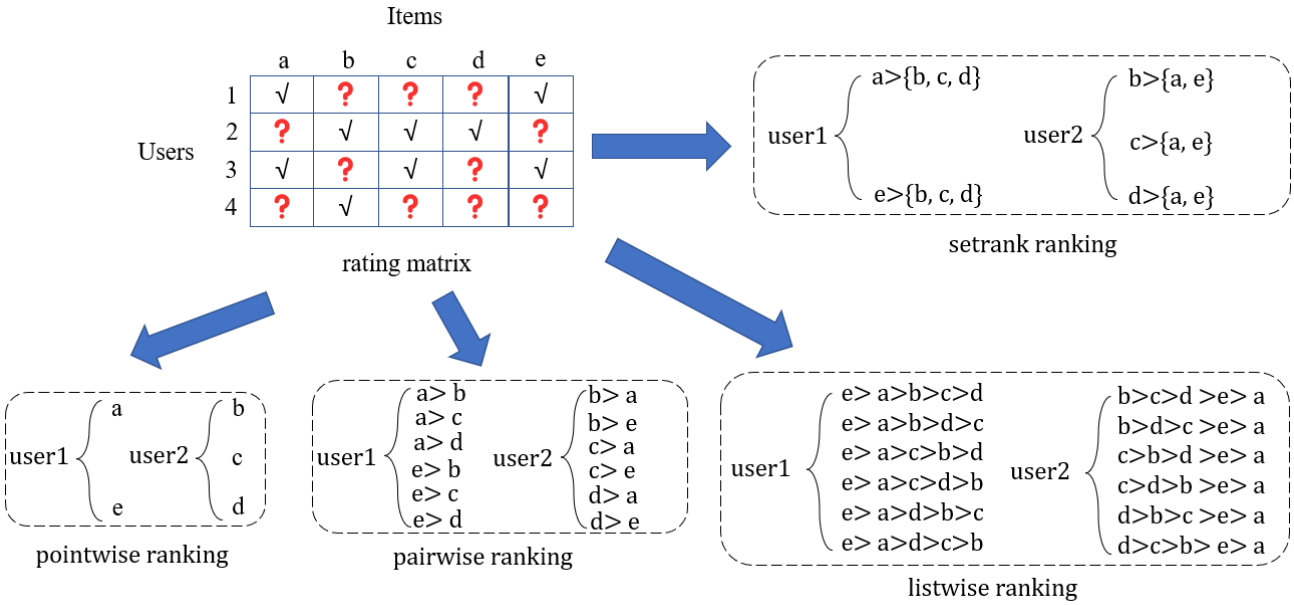


Figure 3. Comparison of different list ranking methods.

3.1. Input Layer and Embedding Layer

The input layer transforms the sparse user–item vector into one-hot encoding and feeds the result into the embedding layer.

The function of the embedding layer in RBLF is to transform the one-hot encoding of users and items into a low-dimensional space and represent them using a dense vector.

The embedding layer is defined as follows:

$$p_u = f_{lookup}(u) \tag{4}$$

$$q_i = f_{lookup}(i) \tag{5}$$

where p_u and q_i represent the embedding vectors of the user and item.

In reality, users and items are independent of each other, and they own a different number of latent features. Conversely, user preferences change over time, which leads to changing latent features. Still, items have relatively fixed attributes from the beginning of their creation; thus, the latent features of items do not change drastically [32]. We set different latent features. In the embedding layer, we use the list of K^+ positive items and K^- negative items. q_i^+ and q_i^- denote the positive and negative samples.

3.2. Shallow-Deep Interaction Grabbing Layer

We want to capture shallow linear user–item interactions and deep non-linear interactions in this layer.

First, the shallow interaction grabbing layer model carries the shallow user–item interaction behavior through explicitly fusing p_u and q_i:

$$X_{uk} = \left| p_u, q_i, p_u \odot q_i \right| \tag{6}$$

$p_u \odot q_i$ is the element-wise of two vectors. $\left| p_u, q_i, p_u \odot q_i \right|$ is the concatenation of these three vectors. By concatenating the $\left| p_u \odot q_i \right|$, the interaction can be better accommodated and prevent loss of feature [33]. We place X_{uk} into the deep interaction grabbing layer.

The goal of the deep interaction grabbing layer is to learn deep and non-linear interaction behavior, and a standard MLP (multilayer perceptron) is used to learn the interaction latent features. Therefore, we can give the model nonlinear modeling capability rather than simply using the inner product multiplied element-by-element as MF (generalized matrix factorization) to describe the potential interaction characteristics. Since multilayer

perceptron can simulate any function, we hope that this layer perceptron can better simulate the implicit preferences between user items. The model connects element-wise in series with multilayer perceptrons, which can be defined as follows:

$$\begin{array}{c} X_{uk}^1 = \sigma\left(W_n^1 X_{uk} + b_n^1\right) \\ X_{uk}^2 = \sigma\left(W_n^2 X_{uk}^1 + b_n^2\right) \\ \cdots \\ X_{uk}^l = \sigma\left(W_n^l X_{uk}^{l-1} + b_n^l\right) \end{array} \tag{7}$$

X_{ij}^l is the result of the lth-layer, W_n^l b_n^l is the weight matrix and bias.

3.3. Predictive Layer

3.3.1. Pairwise Ranking

After we obtain the result of the interaction grabbing layer, we need the interaction grabbing layer to map the result to the probability $\hat{y}_{uk}$. Probability has two properties: the predicted probability is a nonnegative number. Softmax converts the prediction results from negative infinity to positive infinity through the promotion of the two classification functions sigmoid in multiclassification according to these two steps. The probability $\hat{y}_{uk}$ is formulated as follows:

$$\hat{y}_{uk} = softmax\left(X_{uk}^l\right) = \frac{e^{X_{uk}^l}}{\sum_{k=1}^{K} e^{X_{uk}^l}} \tag{8}$$

The pairwise RBLF is a special case of the listwise RBLF method. When the list is 2 (K = 2), the list-wise becomes the pairwise RBLF. The pairwise method models the relative ranking of double items to make predictions. Therefore, the pairwise method constructs the relationship between positive and negative examples. Then, the loss function we set through cross entropy is as follows:

$$p(y, \hat{y}) = -\sum_{u=1}^{U}\left(\sum_{i \in l_u^+} \log \hat{y}_{ui} + \sum_{j \in l_u^-} \log(1 - \hat{y}_{uj})\right) \tag{9}$$

y_{ui} indicates the true probability, $\hat{y}_{ui}$ indicates the predicted probability, l_u^+ and l_u^- represent interactive and noninteractive items by user u, respectively.

Compared with listwise RBLF, the pairwise RBLF is used to determine the relative order of two products and to integrate the results to obtain the final recommendation list, which emphasizes the short running time of the code.

Finally, we use regularization methods to avoid overfitting, and the regularization is defined as:

$$P = p(y, \hat{y}) + \alpha\left(\sum_{l=1}^{L} \|W_l\|_F^2 + \sum_{u=1}^{U} \|p_u\|_F^2 + \sum_{i=1}^{I} \|q_i\|_F^2\right) \tag{10}$$

where $\|\cdot\|_F^2$ represents the Fibonacci-norm and α is the regularization coefficient set in the experiment.

3.3.2. Listwise Ranking

As in method pairwise RBLF, the listwise RBLF also needs the interaction-grabbing layer to map the result to the probability $\hat{y}_{uk}$. Unlike method pairwise RBLF, the listwise RBLF learns the sample features of an ordered list instead of learning an ordered pair. The probability of items is defined as:

$$p(S(i_1, i_2, \ldots, i_K)) = \prod_{i \in l_u^+} \hat{y}_{ui} \prod_{j \in l_u^-} 1 - \hat{y}_{uj} \tag{11}$$

where $S(i_1, i_2, \ldots, i_K)$ denotes the set of all items in list l_u and K denotes the number of items in the list l_u. Then, the model simulates the distribution between the true list and the predicted list by cross entropy. The listwise ranking with regularization is defined as follows:

$$p(y, \hat{y}) = -\sum_{u=1}^{U}\left(\sum_{i\in l_u^+}\log \hat{y}_{ui} + \sum_{j\in l_u^-}\log(1-\hat{y}_{uj})\right) + \alpha\left(\sum_{l=1}^{L}\|W_l\|_F^2 + \sum_{u=1}^{U}\|p_u\|_F^2 + \sum_{i=1}^{I}\|q_i\|_F^2\right) \tag{12}$$

3.3.3. Deep-Setrank

Considering that each user's rating process can be approximated as independent of each other and not influenced by other users, we first assume that each user's rating results are independent. Then, for each user, its preference for the positive sample can be considered higher than that of the unobserved sample. Thus, we can compare each positive sample of users with the set of unobserved samples and assume that the probability of a user liking a positive sample is greater than the probability of liking the set of unobserved samples. We can then maximize the likelihood probability values of these comparisons to solve the problem. Compared to the pairwise assumption, the setrank assumption avoids the problem of inconsistent independence by relaxing the independence requirement. The Bayesian posterior probability of the setrank preference structure can be given as:

$$p(>_{total}|\Theta) = \prod_{u=1}^{U} p(>_u|\Theta) = \prod_{u=1}^{U}\prod_{k\in l_u} p\left(l_u^+ > l_u^-|\Theta\right) \tag{13}$$

where $>_{total}$ is the preference structure of all users, $>_u$ is a random variable representing the preference structure of the representation user u, and Θ is the parameter to be learned in the scoring modeling section. $p(l_u^+ > l_u^-|\Theta)$ represents the probability that a positive sample l_u^+ is better than the preference structure of the set consisting of some unobserved samples l_u^-. This probability can be equivalently considered as the probability that this positive sample ranks first in the list consisting of this positive sample and all unobserved samples, while the order between unobserved samples or within positive samples is not to be considered; thus, there is no problem in sorting items with the same rating by the listwise method.

As shown in Figure 3, the preferred items for user 1 are $\{a, e\}$ and the unobserved items are $\{b, c, d\}$. We only need to express that $a > \{b, c, d\}$ and $e > \{b, c, d\}$ without sorting between items $\{a, e\}$ and $\{b, c, d\}$. In the embedding layer, we use the list of one positive items, and $K-1$ negative items.

Our method only cares about the sorting probability when a positive sample is ranked first; thus, introducing the listwise method's list order-based probability modeling formula to calculate the probability can be simplified as:

$$p_{s,1}\left(l_u^+ > l_u^-|\Theta\right) = \frac{\varnothing\left(s_{l_u^+}\right)}{\sum_{a=1}^{K}\varnothing(s_a)} \tag{14}$$

where $p_{s,1}(l_u^+ > l_u^-|\Theta)$ denotes the probability that item l_u^+ is ranked first, and s_a is the rating of item a.

The complete probabilistic modeling form of the setrank method as:

$$p(>_{total}|\Theta) = \sum_{u=1}^{U}\sum_{k\in l_u} -log\frac{\varnothing(\hat{y}_{ui})}{\varnothing(\hat{y}_{ui}) + \sum_{j\in l_u^-}^{K-1}\varnothing(\hat{y}_{uj})} \tag{15}$$

Since we do not sort the set of unobserved items or the set of positive sample items internally, we use the sigmoid function to map the results of the interaction-grabbing layer into probabilities:

$$log\varnothing(\hat{y}_{uk}) = sigmoid\left(X_{uk}^{l}\right) = \frac{1}{1+e^{X_{uk}^{l}}} \tag{16}$$

By means of maximizing the posterior probabilities, the final optimization objective function can be given as:

$$\mathrm{P} = \prod_{u=1}^{U} \prod_{\mathrm{k} \in l_u} p(>_{total}|\Theta) + \alpha(\sum_{l=1}^{L} \|W_l\|_F^2 + \sum_{u=1}^{U} \|p_u\|_F^2 + \sum_{i=1}^{I} \|q_i\|_F^2) \tag{17}$$

4. The Experiment Evaluation

In this section, we introduce the dataset, the experimental evaluation index and the algorithm to compare with our experiment. Our experiment aims to answer four key questions

1. RQ1: How does RBLF perform compared to the currently popular list-sorting algorithms?
2. RQ2: What are the effects of shallow and deep interaction methods in the feature capture module on RBLF?
3. RQ3: What are the effects of different list ranking methods in the predictive layer on RBLF?
4. RQ4: How does it impact the effect of different hyperparameters of the model?

4.1. Datasets, Evaluation Metrics and Compared Models

As shown in Table 1, we evaluated our model on several public datasets, including MovieLens—100 k, MovieLens—1 M, and Yahoo. We randomly selected 80% of the scores for training the data for each dataset.

Table 1. Statistics of the datasets.

Datasets	Users	Items	Rating	Sparsity
ML-100 K	943	1682	100,000	93.63%
ML-1 M	6040	3952	1,000,000	95.81%
Yahoo	7642	11,915	211,231	99.77%

In addition, we used two popular accuracy metrics, the HR@N and the NDCG@N (N denotes that the RBLF generate the number of the top-n items).

The larger the value of HR and NDCG, the better the model's performance. The HR@N score is defined as:

$$HR@N = \frac{\sum_{u=1}^{\#users} \frac{hits}{N}}{\#users} \tag{18}$$

where #users are the total users whose items in the test set appear.

NDCG@N is defined as follows:

$$DCG(b,N) = \sum_{i=1}^{b} r_i + \sum_{i=b+1}^{N} \frac{r_i}{log_b i} \tag{19}$$

$$NDCG = \frac{DCG}{iDCG} \tag{20}$$

r_i indicates whether the item ranked i is preferred by the user. $r_i = 1$ indicates that the user likes the product; $r_i = 0$ indicates that the user does not like the product; b is the free parameter, which is generally set to 2; N is the number of the top-n items from RBLF. DCG was normalized to obtain NDCG.

We used the compared methods as shown below:

The ItemKNN model considers the evaluation bias after the calculation is completed and obtains the k most similar items.

BPR [34] is a widely used Bayesian-sorting algorithm.

ListRank-MF [26] uses a learning-sorting algorithm and matrix factorization to improve performance while maintaining low complexity.

Neural cooperative filtering (NCF) [14] directly uses a combination of the DNN and matrix factorization, thereby alleviating the problem of DNN overfitting and ignoring low-rank information.

The DeepCF [13] model is the deep matrix decomposition model (DMF) and gives its own solution to NCF's problem. It uses matching learning, which combines the advantages of the two methods, and effectively avoids the shortcomings of the two methods.

The DeepRank [32] model is built on natural language processing capabilities and is currently one of the best sorting algorithms.

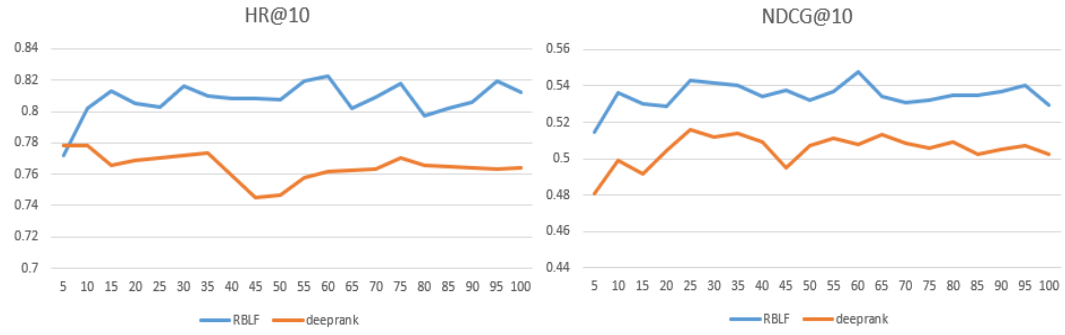
4.2. Performance Evaluation (RQ1)

The model is compared with the benchmark in Table 2. The best marks are highlighted in bold.

Table 2. Comparison results in different datasets.

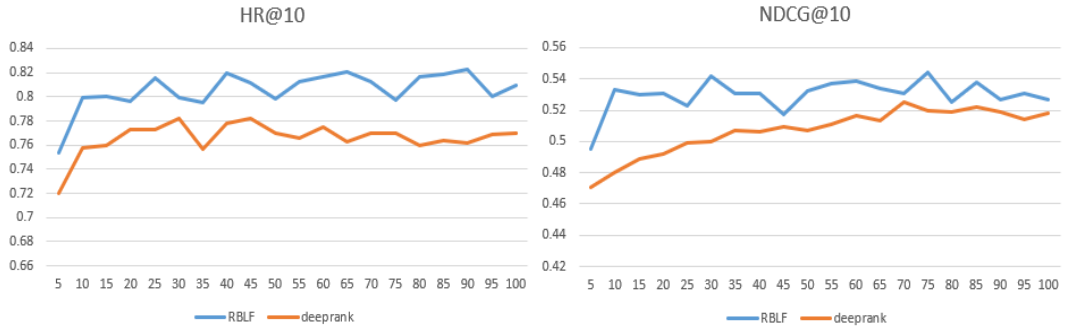
Dataset	Metrics	HR@5	HR@10	NDCG@5	NDCG@10
MovieLens—100 K	ItemKNN	0.482	0.498	0.339	0.362
	BPR	0.536	0.668	0.371	0.411
	List-rank MF	0.511	0.653	0.355	0.399
	NCF	0.627	0.721	0.432	0.472
	DMF	0.653	0.750	0.431	0.487
	DeepRank	0.778	0.768	0.521	0.511
	RBLF	**0.812**	**0.826**	**0.533**	**0.541**
MovieLens—1 M	ItemKNN	0.452	0.478	0.226	0.279
	BPR	0.516	0.696	0.351	0.413
	List-rank MF	0.488	0.637	0.344	0.389
	NCF	0.607	0.711	0.381	0.447
	DMF	0.693	0.725	0.437	0.445
	DeepRank	0.749	0.761	0.493	0.499
	RBLF	**0.789**	**0.819**	**0.531**	**0.529**
Yahoo	ItemKNN	0.482	0.498	0.339	0.362
	BPR	0.641	0.796	0.571	0.641
	List-rank MF	0.618	0.737	0.556	0.608
	NCF	0.785	0.821	0.648	0.694
	DMF	0.787	0.825	0.657	0.703
	DeepRank	0.853	0.868	0.695	0.713
	RBLF	**0.928**	**0.917**	**0.722**	**0.741**

We compared the performance of RBLF and the adaptability of the model when facing different datasets in Table 2 and the best performing numbers are in bold. At the same time, we intuitively compared the DeepRank model, which is the closest to our model and records results within each epoch at 100. As seen in Figures 4–6, the x-axis represents the epoch and the y-axis represents the results.



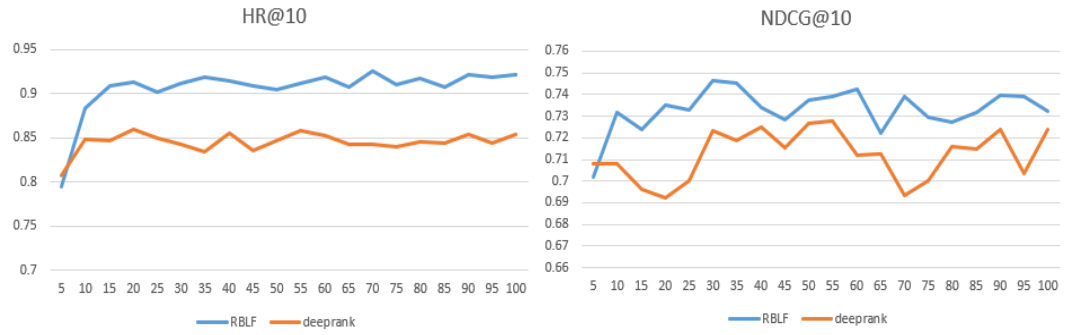
(**a**) (**b**)

Figure 4. (**a**) HR@10 comparison of RBLF and DeepRank in MovieLens—100 K; (**b**) NDCG@10 comparison of RBLF and DeepRank in MovieLens—100 K.



(**a**) (**b**)

Figure 5. (**a**) HR@10 comparison of RBLF and DeepRank in MovieLens—1 M; (**b**) NDCG@10 comparison of RBLF and DeepRank in MovieLens—1 M.



(**a**) (**b**)

Figure 6. (**a**) HR@10 comparison of RBLF and DeepRank in YaHoo Movie; (**b**) NDCG@10 comparison of RBLF and DeepRank in YaHoo Movie.

As seen in Table 2, our proposed method achieves excellent ranking performance and has considerable advantages on each dataset. We believe that it is precisely because our model better simulates the user's preferences that the performance is ahead of the performance of the comparison algorithm. In addition, RBLF consistently outperforms the DeepRank model on the three datasets and increases by 6.3%, 7.1%, and 4.5%, respectively. (According to the paper and our experimental data, the specific values of the hyperparameters when DeepRank achieves the best performance are: the length of the list K = 15, the user and item dimension sizes of embedding $d_u, d_u = \{16, 8\}$, and the depth of MLP L = 4.) Figures 4–6 show that the RBLF and DeepRank epoch are between 0–100,

including the values of HR@10 and NDCG@10. In the figure, we can see more clearly that the performance of RBLF is better at each epoch, and at the same time, it avoids DeepRank due to the problem of mid-term performance degradation.

On the sparsest dataset Yahoo! Movie, RBLF is also superior to other methods, indicating that the idea of combining RBLF in our model simulates the invisible preferences of users, ensuring high performance and high flexibility of the model. Since BPR and ListRank-MF are simply linear interactions, they perform relatively poorly on all datasets, although they also model invisible preferences. The DeepRank model uses the MLP model, which omits some low-rank information and some simple user characteristics in the user–item matrix. As a result, although their performance is high, they are still inferior to RBLF. Although the NCF model uses MLP and MF, our model is a cascade fusion, which allows the model to better integrate these two algorithms, and thus, our model performance is even better. The DeepCF method is concerned with the point-by-point method and ignores the paired ranking information. Our model captures the user's characteristics from the user's paired item. This result in our model is more powerful than theirs in predicting the performance of personalized rankings.

4.3. Ablation Experiments of Shallow and Deep Interaction Methods (RQ2)

As shown in Table 3, √ represents whether the feature capture layer includes this module. We did three types of experiments in total and recorded the performance changes when the model only has a shallow interaction grabbing layer, only a deep interaction grabbing layer, and a complete shallow deep interaction grabbing module. The performance of the RBLF model decreases by 1.5% when the shallow interaction-grabbing layer is lost. In comparison, the overall performance of the model reduces by 3.4% when the multilayer perceptron for grabbing the deep interaction is lost. This proves that the model for capturing shallow interaction has less impact on the overall performance than MLP, and the nonlinear module can better model the user's preference.

Table 3. Comparative results of ablation experiments.

Dataset	Module		Evaluation Indicators	
	Shallow	Deep	HR@10	NDCG@10
100 K		√	0.803	0.518
	√		0.787	0.509
	√	√	0.815	0.521
1 M		√	0.795	0.517
	√		0.774	0.515
	√	√	0.813	0.519
Yahoo		√	0.881	0.736
	√		0.870	0.711
	√	√	0.895	0.740

4.4. Different List Ranking Methods (RQ3)

The results for different list ranking methods are shown in Table 4. Deep-setrank outperforms the pairwise ranking and listwise ranking on the two datasets and increases by 10% and 1.8%, respectively.

Table 4. Comparison results for different list ranking methods.

List—Methods	MovieLens—100 K		MovieLens—1 M	
	hr@10	ndcg@10	hr@10	ndcg@10
pairwise ranking	0.7407	0.4756	0.7429	0.4940
listwise ranking	0.8080	0.5271	0.7937	0.5390
Deep-set rank	0.8234	0.5408	0.8191	0.5291

The reason Deep-setrank performance is substantially better than pairwise ranking is that pairwise methods typically model the preference structure in implicit feedback based on an item pair consisting of a positive feedback item and an unobserved item. This approach is prone to the problem of inconsistent independence in assumptions and implementation. Pairwise ranking attempts to maximize the probability of pairwise comparisons between positive and unobserved simples. This work requires the strict assumption that two items have independent pairwise preferences as the basis for constructing the loss function. Therefore, the independence between preference pairs cannot be guaranteed, which affects the optimization results of the pairwise loss function. Only the order of the two documents is considered, and the position of the documents in the search list is not, resulting in a less-than-optimal final ranking.

The reason Deep-setrank outperforms listwise ranking is that the list method is implemented by defining a probabilistic relationship between the preference sizes on the list of items. For the list method, items with the same rating value cannot be handled efficiently, especially because there is no explicit graded rating in the implicit feedback but 0/1 rating, which can lead to a large number of items with the same rating. In contrast, Deep-setrank does not sort the set of unobserved items or the set of positive sample items internally but only ensures that each positive sample is larger than the set of unobserved samples. Therefore, Deep-setrank models the implicit feedback data more realistically than listwise ranking.

4.5. Different Hyperparameters of the Model (RQ4)

The effect of the length of the list is shown in Table 5. First, as the list length K increases, the model performance increases. After K = 5, the performance is not significantly improved. After K = 10, the performance of the model begins to decrease as K increases. This is because our model is more complex, and the meaningless increase in the length of the list affects the final performance. At the same time, the longer list length inevitably leads to a substantial increase in the running time. Therefore, we finally take K as 5, the optimal parameter after combining time and performance.

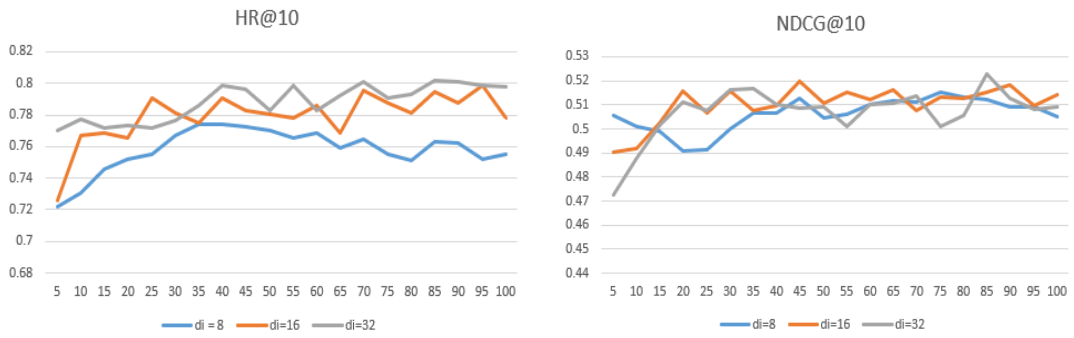
Table 5. Comparison results for the different lengths of the list.

K	MovieLens—100 K		MovieLens—1 M		Yahoo! Movie	
	hr@10	**ndcg@10**	**hr@10**	**ndcg@10**	**hr@10**	**ndcg@10**
2	0.7440	0.4849	0.7494	0.4867	0.8515	0.6706
5	0.8235	0.5480	0.8185	0.5486	0.8991	0.7453
10	0.8189	0.5282	0.7993	0.5319	0.8941	0.7354
15	0.8218	0.5329	0.8033	0.5366	0.8990	0.7312

The embedding size is important for the representation of the project. We conducted experiments to determine the impact of embeddings of different dimensional sizes on the performance of the model. We set the user and item embedding dimension size $d_u = \{8, 16, 32\}$, e $d_i = \{8, 16, 32\}$, and the results are shown in Table 6. Figures 7–9 show that the user embedding remains unchanged in MovieLens—100 K and the impact of changing the size of the item embedding, where the x-axis represents the epoch and the y-axis represents the results.

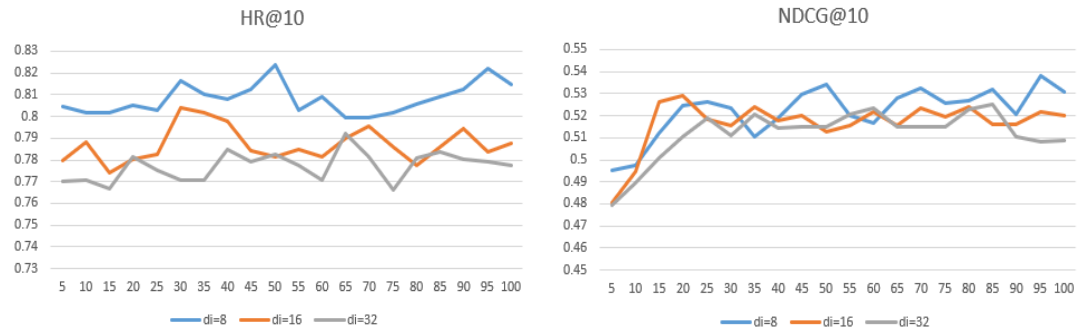
Table 6. Comparison results in the different dimension sizes of embedding.

d_u	d_i	MovieLens—100 K		MovieLens—1 M	
		hr@10	**ndcg@10**	**hr@10**	**ndcg@10**
8	8	0.7467	0.5003	0.7676	0.5113
8	16	0.8167	0.5348	0.8071	0.5274
8	32	0.8006	0.5329	0.7991	0.5148
16	8	0.8145	0.5231	0.8058	0.5228
16	16	0.8021	0.5217	0.7964	0.5199
16	32	0.7826	0.5232	0.7637	0.5190
32	8	0.7607	0.5156	0.7629	0.5140
32	16	0.7544	0.5074	0.7553	0.4999
32	32	0.7588	0.0.5095	0.7487	0.5029



(**a**) (**b**)

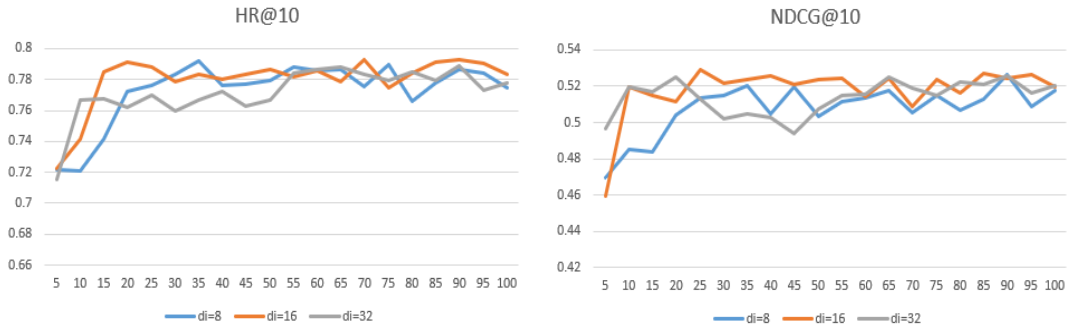
Figure 7. (**a**) d_u = 8, comparisons of HR@10 under different d_i in MovieLens—100 K; (**b**) d_u = 8, comparisons of NDCG@10 under different d_i in MovieLens—1 M.



(**a**) (**b**)

Figure 8. (**a**) d_u = 16, comparison of HR@10 under different d_i in MovieLens—100 K; (**b**) d_u = 16, comparison of NDCG@10 under different d_i in MovieLens—1 M.

After conducting many experiments, we can conclude the following: First, when the user–item embedding dimensions are different, the performance is better than that of the user and the item with the same embedding dimension. Second, if you want to achieve better results, neither user embedding nor item embedding can take 32 because excessive dimensions.



(**a**) (**b**)

Figure 9. (**a**) d_u = 32, comparison of HR@10 under different d_i in MovieLens—100 K; (**b**) d_u = 32, comparison of NDCG@10 under different d_i in MovieLens—1 M.

Third, the dimension of user embedding is preferably smaller than the dimension of item embedding. Finally, we conclude that when d_u = 8 and d_i = 16, the model's performance can reach its highest and performs much better on different datasets, proving that the generalization ability is well.

In the new experiment, we set the size of MLP to [8], [16,8], [32,16,8], [64,32,16,8] and [128,64,32,16,8], and the results are shown in Table 7.

Table 7. Comparison results for different numbers of layers for MLP.

L	MovieLens—100 K		MovieLens—1 M	
	hr@10	**ndcg@10**	**hr@10**	**ndcg@10**
1	0.7504	0.5149	0.7691	0.4731
2	0.7916	0.5289	0.7667	0.4768
3	0.8181	0.5300	0.7996	0.4986
4	0.8234	0.5408	0.8191	0.5291
5	0.8049	0.5326	0.7885	0.4861

In the beginning, as the number of layers of MLP increases, the performance of the model also improves because the deep neural network is similar to the structure of the shallow neural network when the number of layers is too small, and the model does not have sufficient fitting capabilities. Therefore, as L increases, the fitting ability of the deep neural network also increases, which drives the improvement of the model's performance. After L = 4, the performance decreases instead because the model adds a shallow interaction-grabbing layer, limiting the MLP model's depth.

In this section, we compare the performance and time cost of three list ranking methods. Table 8 shows the training time of the three models on the MovieLens—100 K and Movielens—1 M datasets.

Table 8. Time cost for different list ranking methods.

	MovieLens—100 K	**Movielens—1 M**
Pairwise-ranking	2 m 12 s	14 m 49 s
Listwise-ranking	5 m 5 s	38 m 20 s
Deep-setrank	5 m 1 s	36 m 22 s

The time cost of the pairwise-ranking model is far superior to that of those models, although the performance of pairwise-ranking is the worst. As for listwise-ranking and deep-setrank, they take about the same amount of time, but the performance of deep-setrank is better than the listwise-ranking. Thus, we suggest that if you care about the

time cost, you should choose pairwise-ranking, and if you prefer higher performance, you should choose deep-setrank.

5. Conclusions and Future Work

This paper proposes a list-ranking framework based on linear and non-linear fusion for recommendation (RBLF). The model addresses the problem in that current list-sorting recommender systems always fail to capture the full range of user–item interaction information by adding a shallow–deep interaction-grabbing layer, thus improving the model performance.

In the future, we want to optimize the way to obtain the potential vectors of users and items. As the theory of graph neural networks matures, the potential vectors obtained by graph neural networks are much richer than one-hot coding and embedding. In addition, multimodal auxiliary information can make the features more adequate, such as using the item's image and the user's comment to optimize the feature vector.

Author Contributions: Conceptualization, B.W.; methodology, B.W.; software, B.W.; validation, B.W.; formal analysis, J.Q.; writing—original draft preparation, B.W.; writing—review and editing, B.W.; supervision, B.W.; funding acquisition, J.Q. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Science Foundation of China under grant no. 61867006. This work was supported by the Science Fund for Outstanding Youth of Xinjiang Uygur Autonomous Region under grant no. 2021D01E14; the Key Laboratory Open Project of Science and Technology Department of Xinjiang Uygur Autonomous Region named Research on video information intelligent processing technology for Xinjiang regional security; the Major science and technology project of Xinjiang Uygur Autonomous Region under grant no 2020A03001; the project of China Mobile Communications Group Xinjiang Ltd. under grant no. CMXJ-20201279.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: We evaluated our algorithm on three datasets: MovieLens—100 k, MovieLens—1 M, and Yahoo. https://grouplens.org/datasets/MovieLens/100k/ (accessed on 15 October 2020); https://grouplens.org/datasets/MovieLens/1m/http://webscope.sandbox.yahoo.com/catalog.php?datatype=r&did=75 (accessed on 16 October 2020).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
2. Serban, I.; Sordoni, A.; Bengio, Y.; Courville, A.; Pineau, J. Building end-to-end dialogue systems using generative hierarchical neural network models. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AR, USA, 12–17 February 2016; Volume 30.
3. Yu, L.; Zhang, W.; Wang, J.; Yu, Y. Seqgan: Sequence generative adversarial nets with policy gradient. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
4. Luo, M.; Chang, X.; Nie, L.; Yang, Y.; Hauptmann, A.G.; Zheng, Q. An adaptive semisupervised feature analysis for video semantic recognition. *IEEE Trans. Cybern.* **2017**, *48*, 648–660. [CrossRef] [PubMed]
5. Sedhain, S.; Menon, A.K.; Sanner, S.; Xie, L. Autorec: Autoencoders meet collaborative filtering. In Proceedings of the 24th International Conference on World Wide Web, Florence, Italy, 18–22 May 2015; pp. 111–112.
6. Gong, J.; Du, W.; Li, H.; Li, Q.; Zhao, Y.; Yang, K.; Wang, Y. Score prediction algorithm combining deep learning and matrix factorization in sensor cloud systems. *IEEE Access* **2020**, *9*, 47753–47766. [CrossRef]
7. Zhang, Q.; Cao, L.; Zhu, C.; Li, Z.; Sun, J. Coupledcf: Learning explicit and implicit user-item couplings in recommendation for deep collaborative filtering. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, Stockholm, Sweden, 13–19 July 2018.
8. Zhang, D.; Yao, L.; Chen, K.; Wang, S.; Chang, X.; Liu, Y. Making sense of spatio-temporal preserving representations for EEG-based human intention recognition. *IEEE Trans. Cybern.* **2019**, *50*, 3033–3044. [CrossRef] [PubMed]
9. Chen, K.; Yao, L.; Zhang, D.; Wang, X.; Chang, X.; Nie, F. A semisupervised recurrent convolutional attention model for human activity recognition. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *31*, 1747–1756. [CrossRef] [PubMed]

10. Hornik, K.; Stinchcombe, M.; White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **1989**, *2*, 359–366. [CrossRef]
11. Chae, D.K.; Kang, J.S.; Kim, S.W.; Lee, J.T. Cfgan: A generic collaborative filtering framework based on generative adversarial networks. In Proceedings of the 27th ACM International Conference on Information and Knowledge Management, Turin, Italy, 22–26 October 2018; pp. 137–146.
12. Sun, C.; Liu, H.; Liu, M.; Ren, Z.; Gan, T.; Nie, L. LARA: Attribute-to-feature adversarial learning for new-item recommendation. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February 2020; pp. 582–590.
13. Deng, Z.H.; Huang, L.; Wang, C.D.; Lai, J.H.; Philip, S.Y. Deepcf: A unified framework of representation learning and matching function learning in recommender system. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 61–68.
14. He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; Chua, T.S. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, Perth, Australia, 3–7 April 2017; pp. 173–182.
15. Ding, Q.; Liu, Y.; Miao, C.; Cheng, F.; Tang, H. A hybrid bandit framework for diversified recommendation. *arXiv* **2020**, arXiv:2012.13245.
16. Qiu, Z.; Wu, X.; Gao, J.; Fan, W. U-BERT: Pre-training user representations for improved recommendation. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-21), Menlo Park, CA, USA, 2–9 February 2021; pp. 1–8.
17. Liu, C.; Li, X.; Cai, G.; Dong, Z.; Zhu, H.; Shang, L. Non-invasive Self-attention for Side Information Fusion in Sequential Recommendation. *arXiv* **2021**, arXiv:2103.03578.
18. Wang, C.; Zhu, H.; Zhu, C.; Qin, C.; Xiong, H. Setrank: A setwise bayesian approach for collaborative ranking from implicit feedback. In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI-20), New York, NY, USA, 7–12 February 2020; Volume 34, pp. 6127–6136.
19. Goldberg, D.; Nichols, D.; Oki, B.M.; Terry, D. Using collaborative filtering to weave an information tapestry. *Commun. ACM* **1992**, *35*, 61–70. [CrossRef]
20. Adomavicius, G.; Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 734–749. [CrossRef]
21. Koren, Y.; Bell, R.; Volinsky, C. Matrix factorization techniques for recommender systems. *Computer* **2009**, *42*, 30–37. [CrossRef]
22. Xue, H.J.; Dai, X.; Zhang, J.; Huang, S.; Chen, J. Deep matrix factorization models for recommender systems. In Proceedings of the 2017 International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017; Volume 17, pp. 3203–3209.
23. Wang, H.; Zhang, F.; Zhao, M.; Li, W.; Xie, X.; Guo, M. Multi-task feature learning for knowledge graph enhanced recommendation. In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019; pp. 2000–2010.
24. Kim, D.; Park, C.; Oh, J.; Lee, S.; Yu, H. Convolutional matrix factorization for document context-aware recommendation. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016; pp. 233–240.
25. Koren, Y. Collaborative filtering with temporal dynamics. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, 28 June–1 July 2009; pp. 447–4566.
26. Shi, Y.; Larson, M.; Hanjalic, A. List-wise learning to rank with matrix factorization for collaborative filtering. In Proceedings of the Fourth ACM Conference on Recommender Systems, Barcelona, Spain, 26–30 September 2010; pp. 269–272.
27. Zhou, X.; Wu, S. Rating LDA model for collaborative filtering. *Knowl.-Based Syst.* **2016**, *110*, 135–143. [CrossRef]
28. Wu, L.; Hsieh, C.J.; Sharpnack, J. Sql-rank: A listwise approach to collaborative ranking. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 5315–5324.
29. Shi, Y.; Karatzoglou, A.; Baltrunas, L.; Larson, M.; Oliver, N.; Hanjalic, A. Climf: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In Proceedings of the Sixth ACM Conference on Recommender Systems, Dublin, Ireland, 9–13 September 2012; pp. 139–146.
30. He, X.; Zhang, H.; Kan, M.Y.; Chua, T.S. Fast matrix factorization for online recommendation with implicit feedback. In Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, Pisa, Italy, 17–21 July 2016; pp. 549–558.
31. Bai, T.; Wen, J.R.; Zhang, J.; Zhao, W.X. A neural collaborative filtering model with interaction-based neighborhood. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, Singapore, 6–10 November 2017; pp. 1979–1982.
32. Chen, M.; Zhou, X. DeepRank: Learning to rank with neural networks for recommendation. *Knowl.-Based Syst.* **2020**, *209*, 106478. [CrossRef]
33. Mu, N.; Zha, D.; He, Y.; Tang, Z. Graph attention networks for neural social recommendation. In Proceedings of the 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), Portland, OR, USA, 4–6 November 2019; pp. 1320–1327.
34. Rendle, S.; Freudenthaler, C.; Gantner, Z.; Schmidt-Thieme, L. BPR: Bayesian personalized ranking from implicit feedback. *arXiv* **2012**, arXiv:1205.2618.