Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

# DAN: a deep association neural network approach for personalization recommendation*

Xu-na WANG, Qing-mei TAN‡

*College of Economic and Management, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China*

E-mail: Xuna@nuaa.edu.cn; tanchina@nuaa.edu.cn

Received May 10, 2019; Revision accepted Oct. 9, 2019; Crosschecked June 22, 2020

**Abstract:** The collaborative filtering technology used in traditional recommendation systems has a problem of data sparsity. The traditional matrix decomposition algorithm simply decomposes users and items into a linear model of potential factors. These limitations have led to the low accuracy in traditional recommendation algorithms, thus leading to the emergence of recommendation systems based on deep learning. At present, deep learning recommendations mostly use deep neural networks to model some of the auxiliary information, and in the process of modeling, multiple mapping paths are adopted to map the original input data to the potential vector space. However, these deep neural network recommendation algorithms ignore the combined effects of different categories of data, which can have a potential impact on the effectiveness of the recommendation. Aimed at this problem, in this paper we propose a feedforward deep neural network recommendation method, called the deep association neural network (DAN), which is based on the joint action of multiple categories of information, for implicit feedback recommendation. Specifically, the underlying input of the model includes not only users and items, but also more auxiliary information. In addition, the impact of the joint action of different types of information on the recommendation is considered. Experiments on an open data set show the significant improvements made by our proposed method over the other methods. Empirical evidence shows that deep, joint recommendations can provide better recommendation performance.

**Key words:** Neural network; Deep learning; Deep association neural network (DAN); Recommendation
https://doi.org/10.1631/FITEE.1900236 **CLC number:** TP391

## 1 Introduction

With the development of information technology, information overload has become an increasingly serious issue. The key to a personalized recommendation system is to explore users' preferences based on their interactive behavior (such as ratings, clicks, and comments) and provide users with personalized services in line with their wishes and needs (Lu et al., 2015).

For simplicity, we can regard the recommendation problem as a prediction of user preferences. According to the explicit or implicit feedbacks of users, a scientific method is adopted to predict users' preference for projects (Yeung, 2016). The widely used idea to achieve a personalized recommendation is to explore users' preferences based on their historical rating behavior or information (Jung, 2012; Buettner, 2016). There are two types of historical rating information, one of which has a rating of more than two scales, and the other just two scales (Aiolli, 2014; Fu et al., 2019). For example, in the former case, in the public data set "MovieLens," users' rating of a movie is an integer between 1 and 5, which indicates the users' preference for the movie using the rating data number. In the latter case, users express their preferences for the project through a "like" or "dislike," or suggest the users' preferences through implicit feedback that is expressed by the interaction

‡ Corresponding author

* Project supported by the National Social Science Foundation of China (No. 19AGL003)

ORCID: Xu-na WANG, https://orcid.org/0000-0002-5882-9562 between users and projects (He et al., 2017). In recent years, the high cost of collecting users' historical rating data and the fact that users' ratings on multiple scales tend to be concentrated around the central score have led to a challenge in reflecting user preferences. Thus, the interest in the former (that is, the multi-scale rating) has gradually decreased (Li Y et al., 2019). However, compared with the former case, it is relatively easy to obtain implicit feedback for various kinds of use, such as users' purchase history, watch history, and installed applications. Through the extension of implicit feedback information, users' preferences can be further excavated from one field to multiple fields, thereby improving the accuracy and diversity of the recommendation. In the latter case (that is, a binary rating), there is a growing tendency toward studying implicit feedback recommendations.

At present, traditional recommendation algorithms adopted by recommendation systems include collaborative filtering (CF) and matrix decomposition. Many researchers have launched some updated versions based on these classical algorithms (Barkan and Koenigstein, 2016; Hernando et al., 2016; Ha and Lee, 2017). The CF algorithm holds that similar users have the same or similar preferences for a certain project, which determines the nearest neighbor of the target user by analyzing the historical rating information, and generates recommendations for target users based on the neighbor ratings (Verstrepen et al., 2017). However, the major drawback of a CF algorithm lies in its data sparsity. In other words, due to the large number of users and projects, most users rate only a few projects (Zheng et al., 2016). In a historical rating matrix composed of users and projects, we find that the rating information for most positions in the matrix is absent, which leads to low recommendation accuracy in traditional CF (Liu Y et al., 2018). The matrix decomposition algorithm maps users and projects to the shared potential space, and reconstructs the scoring matrix using the inner product of the potential feature matrix of the users and projects. With the help of a gradient descent algorithm, the error loss between the reconstructed scoring matrix and the original scoring matrix is minimized (del Corso et al., 2019). At present, a large amount of work has been done to enhance the performance of matrix decomposition, such as comprehensive integration of the matrix decomposition algorithm and a neighborhood-based recommendation algorithm (Wu et al., 2012; Cao et al., 2018), integration of the matrix decomposition algorithm and time information (Luo et al., 2018; Xiao et al., 2018), and expanding it to factorization machines (Zhou et al., 2018; Knoll et al., 2019). However, the matrix decomposition algorithm assumes that the user and the project are independent for each dimension of the potential space, and linearly combines them with the same weight (Li ZC and Tang, 2017). The matrix decomposition algorithm can be regarded as a linear model of potential factors, but this simple way of linearly combining the product of potential features is not enough to capture the complex structure of the interaction between users and projects (He et al., 2017).

In recent years, owing to its strong learning ability, deep learning has made real progress in image processing, natural language processing, speech recognition, and other fields, and this progress has also introduced new opportunities for recommendation systems (Noda et al., 2015; Cheng et al., 2018; Hossain and Muhammad, 2018). Deep learning uses a deeper neural network to build the model and obtains the nonlinear structural features of multi-sourced heterogeneous information (Liu JT and Wu, 2017). Recommendation systems based on deep learning usually take data about users and projects as inputs, learn more abstract and dense representations of features, and generate recommendations for users based on these implicit representations (Pan J et al., 2017). Although some studies have applied deep neural networks (DNNs) to recommendations and have been shown to be effective, they mostly use DNNs to model some of the auxiliary information, such as text descriptions of recommended products, content descriptions of videos, and visual descriptions of images. For the key factors that affect the model's effectiveness, they still adopt traditional approaches, such as top-level connections or the inner product, to combine the potential characteristics of users and projects. In addition, the basic idea of most deep learning recommendations is to set up two types of mapping pathways, and to map the information of users and projects into the same hidden space. There is a lack of in-depth research on integrated recommendations from users, projects, and auxiliary information, and on recommendations derived from interactions of multiple sources of information.

Based on the above analysis, in this study we propose a recommendation method based on deep neural networks—a deep association neural network (DAN), and our work is concentrated on binary implicit feedback. Compared with explicit feedback, implicit feedback can be tracked automatically, making it easier to collect on the part of content providers (Pan WK et al., 2019). Wang et al. (2020) focused on exploring the effects of different types of joint forms on the final results of the model from the perspective of different joint forms. In contrast, this paper focuses on the alliance effect of different category features, and explores the influence of category feature alliance of the deep neural network on the model recommendation effect from a macro perspective. In this paper we not only discuss the basic performance of DAN, but also explore the performance from two aspects of "depth" and "width" of the model. To improve the recommendation accuracy, a neural network that is three layers deep is adopted in DAN to carry out the output vector. In addition, compared with the traditional deep neural network, the DAN model has the following contributions: First, DAN incorporates multiple types of input vectors at the bottom of the network, rather than just information about users and projects. Second, DAN considers the impact of joint characteristics derived from different projects on the recommendation. Finally, when considering the influence of joint characteristics on the model, DAN reconstructs the original feature projects from the perspective of feature vectors.

## 2 Problem statement

In this paper we consider a typical recommendation problem, the basic form of which is as follows. For a recommendation system, the basic information that can be collected includes users, items, users' occupations, genders, ages, and user's ratings of items. Although the information about occupations, genders, and ages may not be complete, it is still used in the recommendation system. Here, we use $\boldsymbol{U}$, $\boldsymbol{I}$, $\boldsymbol{O}$, $\boldsymbol{G}$, and $\boldsymbol{A}$ to represent the sets of users, items, occupations, genders, and ages, respectively. Since we concentrate our study on implicit feedback that is binary, we use "1" and "0" to indicate whether there is an interaction between the user and the item (He et al., 2017):

$$y=\begin{cases}1, & \text{if interaction(user } u\text{, item } i\text{) is observed,}\\ 0, & \text{otherwise.}\end{cases} \tag{1}$$

We obtain the user–item interaction $y$ from the users' implicit feedback. Here $y$=1 indicates that there is an interaction between the user and the item, and the interaction can be the action of clicking, commenting, browsing, etc.; $y$=0 indicates that there is no interaction between the user and the item. The value of 0 may indicate that the user does not like the item, or that the user is not aware of the item. The recommendation problem with implicit feedback can be expressed as the problem of estimating users' ratings on unobserved items (this score can be used to evaluate the ranking of items) (Hsu et al., 2018). The classical recommendation approaches suggest that the missing ratings can be generated by the underlying model, and that the interactions between users and items can be used to predict users' ratings on the unobserved items. In fact, if a user rates an item, then there is an interaction between the user and the item. In contrast to traditional recommendation ideas, by the reconstruction of category projects through the combination of the user and the item, we can predict the users' rating on the unobserved items (that is, predict users' rating on the non-interactive items). If the information that the recommendation system can use is not limited to the interactive records of users and items, but contains a mix of auxiliary information, such as occupation, gender, age, and other information about users, we need to consider whether the joint effect between different categories of information has a positive effect on the recommendation results. Therefore, the DAN constructed in this study focuses on the impact of the joint effect of input projects on the recommendation results. DAN considers the number of features at the element level to be huge in actual situations, which leads to the number of joint features increasing at the multiplier level, and makes it difficult to learn the model parameters. Thus, based on the project features at the vector level, we combine different categories of projects to construct new underlying input projects. Finally, to explore the influence of the combined effect of individual features and joint features on recommendation effectiveness, we combine the two models of multi-layer perceptron (MLP) and DAN, and compare the recommendation results of the different models.

## 3 Related works

The related works cover two systems: traditional recommendation systems and deep learning recommendation systems. The classical technique in traditional recommendation systems is CF, and CF based on matrix decomposition has been widely used. However, the classical recommendation model for deep learning is the MLP model.

### 3.1 Traditional recommendation

Reports in the early literature on recommendation systems focused mainly on explicit feedback, but due to the difficulty in data collection in explicit feedback, the focus in recommendation systems shifted to implicit feedback. CF can be used for implicit feedback, which transforms the recommendation task into a list of items matching the user's preferences. Unlike the widely used rating prediction for explicit feedback, solving the item recommendation problem is more in line with the actual needs of users.

The core idea of CF is to make a recommendation based on the preferences of users with the same interest. This kind of recommendation can be divided into two forms: user-based recommendation and item-based recommendation. User-based CF recommendation measures the similarity among users through their behaviors toward different items, finds "neighbor users," and makes recommendations based on such a similarity. The essence is to recommend to the target user what his/her neighbor likes. The principle of item-based CF recommendation is similar to that of user-based CF recommendation, except that similar neighbors are calculated from the perspective of the project, not from the perspective of the user. In other words, the system finds similar projects and recommends similar projects to target users. Overall, user- and item-based recommendations are applicable to different scenarios. For large e-commerce sites, the number of users is much larger than that of items, and the items are relatively stable, so it is appropriate to adopt item-based CF recommendation. Of course, user-based CF recommendation mechanisms are more effective in systems with social attributes.

When recommendation based on matrix decomposition is adopted, a rating matrix can be decomposed into two matrices about users and items, and each user and item can be represented by a potential eigenvector. The evaluation score $\hat{r}_{i,j}$ of user $i$ on item $j$ can be expressed by the following formula (Xiong et al., 2018):

$$\hat{r}_{i,j} = \boldsymbol{p}_i^{\mathrm{T}} \boldsymbol{q}_j = \sum_{k=1}^{K} p_{ik} q_{kj}, \tag{2}$$

where $K$ represents the dimensionality of potential eigenvectors, and $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ represent the potential eigenvectors of users and items, respectively. To learn the potential $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$, we usually use the square of the error between the original score and the prediction score as the loss function, and solve it using the gradient descent method to minimize the error loss. Here, we can use the product of $\boldsymbol{p}_i$ and $\boldsymbol{q}_j$ to predict users' ratings, and recommend several items with high ratings to the user.

### 3.2 Deep learning recommendation

As for deep learning recommendations, the early work used a two-layer restricted Boltzmann machine (RBM) to simulate users' ratings (Jia et al., 2016). Subsequently, autoencoders became the general choice for recommendation systems. Autoencoders take the users' historical ratings as input, and reconstruct the users' ratings by learning the hidden structure (Ma et al., 2018). To avoid the autoencoders learning constant functions, some researchers applied denoising autoencoders to accept corrupted data as input (Marchi et al., 2015). Zheng et al. (2016) proposed a neural autoregressive method for CF. He et al. (2017, 2018) designed the common framework of neural CF (NCF) based on the classical deep learning model, MLP. The above studies demonstrate the effectiveness of deep neural networks for recommendation systems. Specifically, the main architecture and idea behind the classical MLP deep neural network model is described as follows.

The generalized MLP is extended from the simple perceptron, and its main feature is that it has multiple layers of neurons. The network structure of a simple perceptron is depicted in Fig. 1.

For the perceptron model, the input signal and output signal simulate the biological input neuron and output neuron, respectively. The weight simulates the axon, which is the transmission mechanism between the input neuron and receiving neuron. Activation functions mimic the work of neural transmission. The biases simulate the structure of the synapse, indicating the extent to which neurons are easily activated (Liu WB et al., 2017).

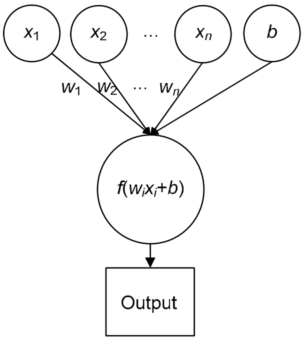


**Fig. 1 Simple perceptron model**

Assume that the input signal of the perceptron is $\boldsymbol{x}$ ($\boldsymbol{x}\in\mathbb{R}^n$), which is an $n$-dimensional vector. The input vector is received by the receiving neuron through the input neurons, and each input neuron has a weight that represents the weight of the input vector at the corresponding position. The weight is represented by $w_i$, and the bias is represented by $b$. The input of a simple perceptron can be expressed as

$$\begin{aligned} z &= w_1x_1 + w_2x_2 + \ldots + w_nx_n + b \\ &= \sum_{i=1}^{n} w_ix_i + b = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b, \end{aligned} \tag{3}$$

where $\boldsymbol{w}, \boldsymbol{x}\in\mathbb{R}^n$ are $n$-dimensional vectors. The output of the perceptron can be expressed as $y=f(z)$, where $f$ is the activation function of the neuron. However, a single-layer perceptron is limited to learning linear functions and cannot effectively learn nonlinear data. In theory, a multi-layer network can simulate any complex function. Therefore, the MLP model is generated by increasing the number of network layers based on the single-layer perceptron. The number of hidden layers is not specified in MLP, and there is no limit on the number of neurons of hidden layers and the output layer, which can be selected according to their own needs. The activation function of MLP is usually a nonlinear one, and adding the activation function can enable the neural network to deal with more complex nonlinear problems.

The input data set of MLP is $\boldsymbol{X}=[\boldsymbol{x}^1, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^m]$, where $m$ represents the sample size, and $\boldsymbol{x}^i$ ($i$=1, 2, …, $m$) represents a single training sample. In the previous expression, $\boldsymbol{x}^i\in\mathbb{R}^n$, and its components are $x_j^i$ ($j$=1, 2, …, $n$), where $n$ is the dimensionality of the input vector. For a sample $\boldsymbol{x}^i$ in the input sample data set, the process of obtaining the output result using MLP is as follows. Here, the inputs of the model are represented by $\boldsymbol{z}_i$ ($i$=1, 2, …, $L$), and the outputs of the middle layer by $\boldsymbol{\lambda}_i$.

The input of the first layer is

$$\boldsymbol{z}_1 = \boldsymbol{x}^i = (x_1^i, x_2^i, \ldots, x_n^i). \tag{4}$$

The output of the first layer is

$$\boldsymbol{\lambda}_1 = \phi_1(x_1^i, x_2^i, \ldots, x_n^i). \tag{5}$$

The input of the second layer is shown as

$$\begin{aligned} \boldsymbol{z}_2 &= (z_{21}, z_{22}, \ldots, z_{2k}) \\ &= (\boldsymbol{W}_{21}^{\mathrm{T}}\boldsymbol{\lambda}_1 + b_{21}, \boldsymbol{W}_{22}^{\mathrm{T}}\boldsymbol{\lambda}_1 + b_{22}, \ldots, \boldsymbol{W}_{2k}^{\mathrm{T}}\boldsymbol{\lambda}_1 + b_{2k}), \end{aligned} \tag{6}$$

where $k$ represents the number of neurons in the second layer, and $\boldsymbol{W}_{2i}^{\mathrm{T}}$ ($i$=1, 2, …, $k$) denotes the weight vector of layer 1 pointing to the $i^{\text{th}}$ neuron of layer 2.

The output of the second layer is

$$\boldsymbol{\lambda}_2 = \phi_2(\boldsymbol{z}_2) = \phi_2(\boldsymbol{W}_2^{\mathrm{T}}\boldsymbol{\lambda}_1 + \boldsymbol{b}_2). \tag{7}$$

The output of the third layer is

$$\boldsymbol{\lambda}_3 = \phi_3(\boldsymbol{W}_3^{\mathrm{T}}\boldsymbol{\lambda}_2 + \boldsymbol{b}_3). \tag{8}$$

The output of the $L^{\text{th}}$ layer is

$$\boldsymbol{\lambda}_L = \phi_L(\boldsymbol{W}_L^{\mathrm{T}}\boldsymbol{\lambda}_{L-1} + \boldsymbol{b}_L), \tag{9}$$

where $\phi_i$ ($i$=1, 2, …, $L$) represents the activation function of layer $i$, $\boldsymbol{W}_i^{\mathrm{T}}$ ($i$=2, 3, …, $L$) denotes the weight matrix transposition that layer $i-1$ points to layer $i$, and $\boldsymbol{b}_i$ ($i$=1, 2, …, $L$) represents the bias.

The final output result of the output layer is

$$\hat{y} = \sigma(\boldsymbol{h}^{\mathrm{T}}\boldsymbol{\lambda}_L) = \sigma(\boldsymbol{h}^{\mathrm{T}}\phi_L(\boldsymbol{W}_L^{\mathrm{T}}\boldsymbol{\lambda}_{L-1} + \boldsymbol{b}_L)), \tag{10}$$

where $\sigma$ represents the activation function of the output layer, and $\boldsymbol{h}^{\mathrm{T}}$ represents the weight matrix transposition of the output layer. The schematic of MLP can be depicted in Fig. 2.

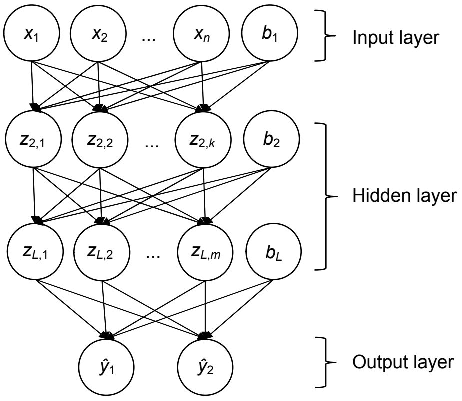


**Fig. 2 Classical multi-layer perceptron model**

## 4 The proposed model

In this section, we first introduce the extended MLP, which is the basis of our proposed model. Then, we explore the influence that the different types of joint actions may have on the recommendation results, and based on the above analysis, we put forward DAN for recommendation. Lastly, to learn the parameters of the proposed model, we provide our loss function designed for optimization.

### 4.1 Extended multi-layer perceptron

In this study, the input of the MLP model is not limited to the data of one category, but contains the raw data of multiple categories such as users, movies, occupation, gender, and age. Therefore, the input of the extended MLP adopted in this study is [$\boldsymbol{U}$, $\boldsymbol{I}$, $\boldsymbol{O}$, $\boldsymbol{G}$, $\boldsymbol{A}$], $\boldsymbol{U}$=($\boldsymbol{u}^1$, $\boldsymbol{u}^2$, …), $\boldsymbol{I}$=($\boldsymbol{i}^1$, $\boldsymbol{i}^2$, …), $\boldsymbol{O}$=($\boldsymbol{o}^1$, $\boldsymbol{o}^2$, …), $\boldsymbol{G}$=($\boldsymbol{g}^1$, $\boldsymbol{g}^2$, …), $\boldsymbol{A}$=($\boldsymbol{a}^1$, $\boldsymbol{a}^2$, …), where $\boldsymbol{U}$, $\boldsymbol{I}$, $\boldsymbol{O}$, $\boldsymbol{G}$, and $\boldsymbol{A}$ represent the data sets of users, items, occupations, genders, and ages, respectively. $\boldsymbol{u}^i\in\mathbb{R}^m$ ($i$=1, 2, …) represents a single training sample, and its components are $u_j^i$ ($j$=1, 2, …, $m$), i.e., $\boldsymbol{u}^i=(u_1^i,u_2^i,...,u_m^i)$, where $m$ denotes the dimensionality of the user vector. In the same way, we can find that $\boldsymbol{i}^i=(i_1^i,i_2^i,...,i_n^i)$, $\boldsymbol{o}^i$=$(o_1^i,o_2^i,...,o_s^i)$, $\boldsymbol{g}^i$=$(g_1^i,g_2^i,...,g_v^i)$, $\boldsymbol{a}^i$=$(a_1^i,a_2^i,...,a_r^i)$, where $n$, $s$, $v$, and $r$ represent the dimensionalities of the item, occupation, gender, and age, respectively. For a sample [$\boldsymbol{u}^i$, $\boldsymbol{i}^i$, $\boldsymbol{o}^i$, $\boldsymbol{g}^i$, $\boldsymbol{a}^i$] ($i$=1, 2, …) in the input sample data set, the process of obtaining the output result using the extended MLP is as follows. Similarly, the inputs of the model are represented by $z_i$ ($i$=1, 2, …, $L$), and the outputs of the middle layer by $\boldsymbol{\lambda}_i$.

The input of the first layer is

$$z_1=[\boldsymbol{u}^i,\boldsymbol{i}^i,\boldsymbol{o}^i,\boldsymbol{g}^i,\boldsymbol{a}^i]. \tag{11}$$

The output of the first layer is shown as

$$\begin{aligned}\lambda_1&=[\lambda_{11},\lambda_{12},\lambda_{13},\lambda_{14},\lambda_{15}]\\&=[\phi_1(u_1,u_2,...,u_m),...,\phi_1(a_1,a_2,...,a_r)].\end{aligned} \tag{12}$$

The output of the second layer is shown as

$$\begin{aligned}\lambda_2&=[\lambda_{21},\lambda_{22},\lambda_{23},\lambda_{24},\lambda_{25}]\\&=\phi_2(\boldsymbol{W}_2^{\mathrm{T}}\lambda_1+\boldsymbol{b}_2)\\&=[\phi_2(\boldsymbol{W}_{21}^{\mathrm{T}}\lambda_{11}+b_{21}),...,\phi_2(\boldsymbol{W}_{25}^{\mathrm{T}}\lambda_{15}+b_{25})].\end{aligned} \tag{13}$$

The output of layer $L$ is

$$\begin{aligned}\boldsymbol{\lambda}_L&=[\boldsymbol{\lambda}_{L1},\boldsymbol{\lambda}_{L2},\boldsymbol{\lambda}_{L3},\boldsymbol{\lambda}_{L4},\boldsymbol{\lambda}_{L5}]=\phi_L(\boldsymbol{W}_L^{\mathrm{T}}\boldsymbol{\lambda}_{L-1}+\boldsymbol{b}_L)\\&=[\phi_L(\boldsymbol{W}_{L1}^{\mathrm{T}}\boldsymbol{\lambda}_{(L-1)1}+b_{L1}),...,\phi_L(\boldsymbol{W}_{L5}^{\mathrm{T}}\boldsymbol{\lambda}_{(L-1)5}+b_{L5})].\end{aligned} \tag{14}$$

The result of the output layer is

$$\begin{aligned}\hat{y}=\sigma(\boldsymbol{h}^{\mathrm{T}}\lambda_L)=\sigma(\boldsymbol{h}^{\mathrm{T}}\phi_L((&\boldsymbol{W}_{L1}^{\mathrm{T}}\boldsymbol{\lambda}_{(L-1)1}+b_{L1})\\&+...+(\boldsymbol{W}_{L5}^{\mathrm{T}}\boldsymbol{\lambda}_{(L-1)5}+b_{L5}))).\end{aligned} \tag{15}$$

What needs to be explained is that recommendation systems usually use one-hot encoding to convert the input variables into a form that can be easily used by a computer. However, the vector transformed by one-hot encoding is high-dimensional and sparse. The embedding layer can convert a high-dimensional sparse vector into a dense one, which is convenient for computer processing. Based on the above analysis, the schematic of the extended MLP in this study can be expressed as Fig. 3.

### 4.2 Deep association neural network

In practice, the features of each field are not independent, and some features always interact with each other. Indeed, in recommendation systems, the interactions between different features also suggest a joint effect between them. At the same time, by observing a large amount of sample data, after some features are combined, the correlation with the target results will be improved. The polynomial model is the most intuitive one that contains joint features. For comparison, we focus only on the second-order

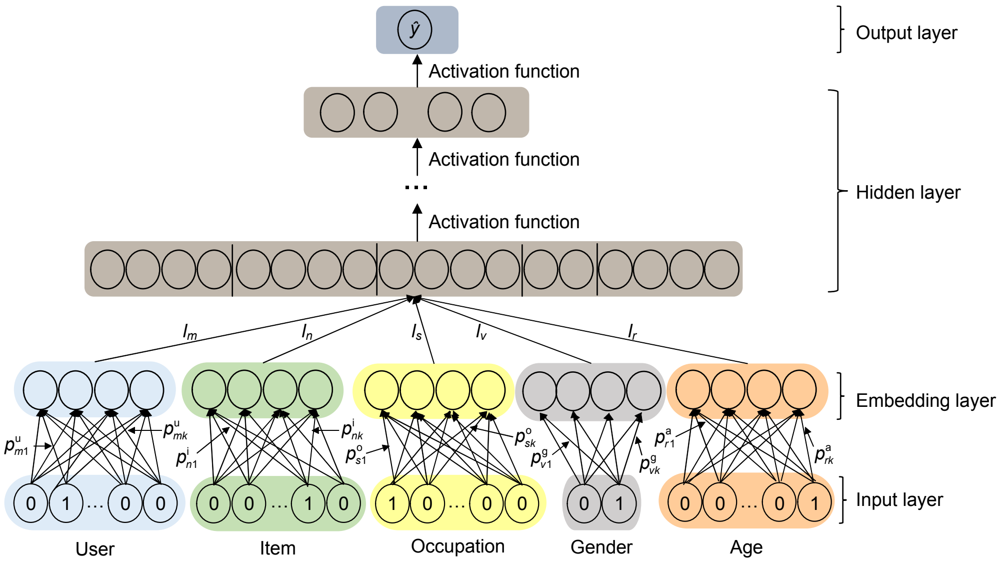


**Fig. 3 The extended multi-layer perceptron**

polynomial model. The expression for the polynomial model can be shown as $y = w_0 + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} w_{ij}x_i x_j$. Here, the bias is represented by $w_0$, with the combination of features $x_i$ and $x_j$ represented by $x_ix_j$. When both $x_i$ and $x_j$ are non-zero, the feature combination $x_ix_j$ is meaningful. In addition, the interaction weight between $x_i$ and $x_j$ is denoted by $w_{ij}$.

Note that, for the problem of second-order polynomial regression, if the number of input data features is $n$, then the feature interactions $w_{ij}$ that need to be learned have a total number of $n(n-1)/2$. When the number of features in the actual situation is large, it will cause difficulty in learning interactive parameters. In addition, after being encoded by one-hot encoding, most features of sample data are sparse. When the feature in any sample data is 0, it is impossible to learn the interaction weight $w_{ij}$ between other features and this feature.

We can consider matrix decomposition to solve this problem. Since joint features $x_ix_j$ and $x_jx_i$ are equivalent, it can be concluded that interaction weights $w_{ij}$ and $w_{ji}$ are equivalent (that is, $w_{ij}=w_{ji}$). In this case, all of the interaction weights form a symmetric matrix. Since weight $w_{ii}$ is meaningless, the diagonal elements of the symmetric matrix equal 0. The above matrix is decomposed into the product of two lower-order matrices, which can avoid the difficulty in learning $w_{ij}$. Thus, the second-order polynomial model can be expressed as $y = w_0 + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{f=1}^{k} v_{i,f}v_{j,f}x_i x_j$, where $v_{i,f}$ and $v_{j,f}$ represent the hidden factors of features $x_i$ and $x_j$, respectively. The product of $v_i$ and $v_j$ can be used to predict the interaction weight $w_{ij}$ between $x_i$ and $x_j$.

However, there are some limitations in using matrix decomposition to predict the weight of interactive features. Matrix decomposition uses the product of two hidden vectors to estimate the interaction between two features. It assumes that each hidden factor in the two hidden vectors is independent, and they are combined linearly with the same weight. This simple approach is insufficient for capturing the complex interactions between different features.

On the other hand, in the study of combined features using deep neural networks, the common mode with underlying data is to convert all features with one-hot encoding. In other words, all features are encoded uniformly, and different combined features are not distinguished at the bottom (Guo et al., 2017). In this way, the mixed data is put into the neural network after all the features are mixed. If the total number of features is $n$, this method nominally analyzes the influence of the $n^{\text{th}}$-order combined features on the output. However, it is not wise to classify all the features into one combination, and it is even more difficult to mine the effects of combination.

The limitations of the above methods are shown as follows: the above research predicts the interaction between all features by matrix decomposition. Yet, in actual situations, there is no interaction between some features, so it is meaningless to calculate the interaction weight $w_{ij}$ between these features, and it will also interfere with the output. We explain this with a simple example. Table 1 shows part of the user rating information on movies, and each row represents the rating information for a user.

**Table 1 Part of the user rating information**

| Rating | Movie | Genre | Occupation | Gender |
|---|---|---|---|---|
| 3 | 3 Idiots | Comedy, drama | Attorney | Male |
| 4 | Jumanji (1995) | Adventure, children's | Student | Female |

The "Rating" type in Table 1 is a numeric system, and it is labeled in the rating system of Movies. "Movie," "Genre," "Occupation," and "Gender" all belong to categorical variables which need to be converted into numerical features through encoding. The encoded data is shown in Table 2.

Excluding the labels, the categorical variables of these two records can be encoded into 10 features. In the second-order polynomial model, there are $n(n-1)/2=45$ kinds of combinations for these features. However, there are many combined features without practical significance. For example, in the case of "Gender," the combination of "male" and "female" does not make sense, because a single user's gender could only be "male" or "female." Furthermore, when it comes to "occupation," it does not make much sense for a user to be both a lawyer and a student. In the feature combination of "Genre," if the features "comedy" and "drama" are combined, on the surface, the learned weight $w$ is the interaction parameter of these two features. Yet, fundamentally speaking, the objects described by these two features are the same movie, and the interaction weight of these two features is also the weight for the same movie "3 Idiots." However, there still exist some problems if combinations of "Genre" are (comedy, adventure), (comedy, children's), (drama, adventure), and (drama, children's). For example, like the case in the above analysis, if these feature combinations potentially represent the same movie, then it does not make sense to obtain the interaction weight of these features. On the contrary, if these feature combinations represent different movie combinations, their interaction weights express the interactive effect of different genres on the user rating. However, a problem coming with this is that a single user rarely watches two or more movies at the same time and rates them. The reality is that users watch different movies at different times. In other words, two movies correspond to two labels, which show two records in the one-hot encoded data table. The interaction weight obtained by such a feature combination is not consistent with our intention, and what we need is the result when the user conforms to features $x_i$ and $x_j$ under the same condition. Similar to the classical example of "beer" and "diapers," it is emphasized that a user purchases "diapers" as well as "beer," and the combination of "beer" and "diapers" occurs during the same shopping trip.

Combined with the above analysis, we find that a feature combination within the same categorical variable tends to produce some meaningless interaction weights. These meaningless weights may even interfere with the output of the model. Compared with the above feature combinations, feature combinations between different categories are more likely to exist and more acceptable, so they are more valuable for research. An example is the feature combinations of (attorney, male) and (student, female), which suggest that there is a male attorney or a female student. Of course, although the above encoded data is not shown, the feature combination of (attorney, female) and (student, male) also makes sense.

In addition, it is found that in the feature sets composed by a variety of categorical variables, the local features covered by some categories are the ductility features of other categorical variables. For example, as shown in Table 1, the features covered by the categorical variable "Genre" are the expansion

**Table 2 Encoded user rating information**

| Rating | Movie= 3 Idiots | Movie= Jumanji (1995) | Genre= Comedy | Genre= Drama | Genre= Adventure | Genre= Children's | Occupation= Attorney | Occupation= Student | Gender= Male | Gender= Female |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |

features of another category "Movie." Specifically, the features "comedy" and "drama" for categorical variable "Genre" essentially describe "3 Idiots," which is included in the categorical variable "Movie." Therefore, it is not appropriate to combine features of different categories when they have a dependent relationship as described above.

Based on the above analysis, consider that the total number of categories in the original input data is limited compared with that of the features, and that a single categorical variable is a vector composed of multiple features. For a new data set, the features of different categories are combined according to the categories contained in the data set, and the combination of category features that have a membership relationship is eliminated to form a potential new category feature. Unlike traditional studies that feature interactions occurring at the element level, the new feature combinations adopt the method of category vector combinations. Each new feature is numerically encoded, and the encoded input data is mapped into a potential representation of the new feature through the embedding layer, and then sent to the neural network for learning.

In this study we select features of five categories in the public data set "MovieLens," including user, film, occupation, gender, and age. $\boldsymbol{U}$, $\boldsymbol{I}$, $\boldsymbol{O}$, $\boldsymbol{G}$, and $\boldsymbol{A}$ represent the feature data sets of the five categories. In this study, based on the above analysis about feature association, the features of the five categories are used to form four newly combined features, which are the combinations of user and film, occupation and gender, age and gender, and occupation and age. The new features are represented by $<\boldsymbol{U}\cdot\boldsymbol{I}>$, $<\boldsymbol{O}\cdot\boldsymbol{G}>$, $<\boldsymbol{A}\cdot\boldsymbol{G}>$, $<\boldsymbol{O}\cdot\boldsymbol{A}>$. The representation of the input for the DAN model is $z_i^{(l)}$, which denotes the input of the $i^{\text{th}}$ neuron in layer $l$. The output is represented by $a_i^{(l)}$, which represents the output of the $i^{\text{th}}$ neuron in layer $l$. The neuron connection weights between layers are represented by $W_{ij}^{(l)}$, which represents the weight from the $j^{\text{th}}$ neuron in the $l^{\text{th}}$ layer to the $i^{\text{th}}$ neuron in the $(l+1)^{\text{th}}$ layer. $\phi_l$ represents the activation function of the output layer, and $\phi_i$ ($i$=1, 2, …, $l$−1) represents the activation function in the $i^{\text{th}}$ layer. The process for a sample using DAN to obtain a prediction result is described as follows.

The first layer of DAN is the preparation layer, and its output can be obtained by combining the features of categorical variables. The training sample set is represented by ($\boldsymbol{X}$, $Y$), and the format of a single training sample is ($\boldsymbol{X}^{(s)}$, $Y^{(s)}$), which indicates the $s^{\text{th}}$ training sample.

$$\boldsymbol{X}^{(s)}=[\boldsymbol{U}^{(s)},\boldsymbol{I}^{(s)},\boldsymbol{O}^{(s)},\boldsymbol{G}^{(s)},\boldsymbol{A}^{(s)}], \tag{16}$$

$$Y^{(s)}=\begin{cases}1, & \text{if ratings are observed,}\\ 0, & \text{otherwise.}\end{cases} \tag{17}$$

The input for the DAN preparation layer is

$$\boldsymbol{z}^{(1)}=\boldsymbol{X}^{(s)}=[\boldsymbol{U}^{(s)},\boldsymbol{I}^{(s)},\boldsymbol{O}^{(s)},\boldsymbol{G}^{(s)},\boldsymbol{A}^{(s)}]. \tag{18}$$

The output of the $i^{\text{th}}$ neuron in the preparation layer is

$$\lambda_i^{(1)}=\varphi(\boldsymbol{X}^{(s)})=C(\boldsymbol{E},1),\ \ i=0,1,\ldots,l_1, \tag{19}$$

$$\begin{aligned}\boldsymbol{E}=(&<\boldsymbol{U}^{(s)}\cdot\boldsymbol{I}^{(s)}>,<\boldsymbol{O}^{(s)}\cdot\boldsymbol{G}^{(s)}>,\\ &<\boldsymbol{A}^{(s)}\cdot\boldsymbol{G}^{(s)}>,<\boldsymbol{O}^{(s)}\cdot\boldsymbol{A}^{(s)}>),\end{aligned} \tag{20}$$

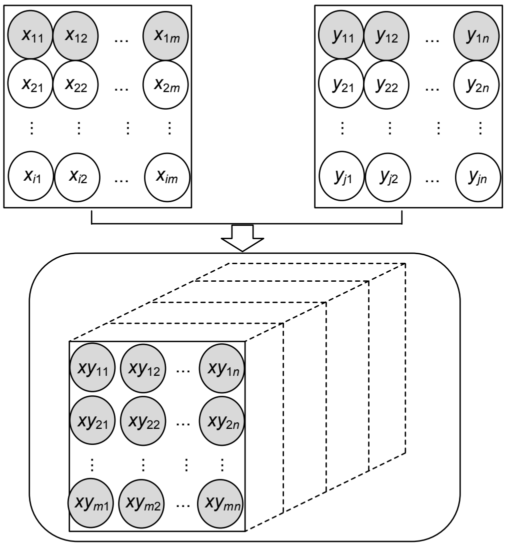
where $\varphi$ denotes the combination function that represents the combination mechanism of categorical vectors, $E$ represents different combinations of categorical vectors, and $l_1$ represents the number of neurons in the preparation layer.

The input data for DAN is essentially a matrix recombined by the categorical vectors contained in the original data, which is a three-dimensional tensor representing a new category project in the potential sense. The schematic of the input data can be expressed as follows.

As shown in Fig. 4, both $\boldsymbol{x}$=[$\boldsymbol{x}_1$, $\boldsymbol{x}_2$, …, $\boldsymbol{x}_i$] and $\boldsymbol{y}$=[$\boldsymbol{y}_1$, $\boldsymbol{y}_2$, …, $\boldsymbol{y}_j$] represent the input category matrices, where the components $\boldsymbol{x}_a$ ($a$=1, 2, …, $i$) and $\boldsymbol{y}_b$ ($b$=1, 2, …, $j$) have dimensionalities of $m$ and $n$, respectively. The category matrices $\boldsymbol{x}$ and $\boldsymbol{y}$ are combined to form a three-dimensional tensor, which is represented by symbol $\boldsymbol{xy}$, where the single matrix formed by the three-dimensional tensor has the form of $m$ rows and $n$ columns.

Therefore, the input for the DAN input layer is

$$\boldsymbol{z}^{(2)}=\boldsymbol{\lambda}^{(1)}=(\lambda_0^{(1)},\lambda_1^{(1)},\ldots,\lambda_{l_1}^{(1)}). \tag{21}$$



$x_{11}$ $x_{12}$ ... $x_{1m}$
$x_{21}$ $x_{22}$ ... $x_{2m}$
$x_{i1}$ $x_{i2}$ ... $x_{im}$
$y_{11}$ $y_{12}$ ... $y_{1n}$
$y_{21}$ $y_{22}$ ... $y_{2n}$
$y_{j1}$ $y_{j2}$ ... $y_{jn}$
$xy_{11}$ $xy_{12}$ ... $xy_{1n}$
$xy_{21}$ $xy_{22}$ ... $xy_{2n}$
$xy_{m1}$ $xy_{m2}$ ... $xy_{mn}$

**Fig. 4 Category composition of the input data for the DAN model**

The output of the input layer is

$$\boldsymbol{\lambda}^{(2)}=\phi_2(\boldsymbol{z}^{(2)})=\phi_2(\lambda_0^{(1)},\lambda_1^{(1)},\ldots,\lambda_{l_1}^{(1)}),\tag{22}$$

where the $i^{\text{th}}$ neuron in $\boldsymbol{\lambda}^{(2)}$ can be represented as $\lambda_i^{(2)}$. The input layer in DAN is followed by the embedding layer, which is used to map the new features to potential feature vectors. In the following formulae, $i$ represents the $i^{\text{th}}$ neuron, and the input of the $i^{\text{th}}$ neuron in the embedding layer is expressed as

$$z_i^{(3)}=\sum_{j=1}^{n^{(2)}}W_{ij}^{(2)}\lambda_j^{(2)}+b_2,\tag{23}$$

where $n^{(2)}$ stands for the number of neurons in the front layer. Usually, to improve the calculation efficiency, additional neuron nodes $a_0$=1 are introduced, and the original bias is represented by symbol $w_0$. In the current situation, it is expressed as $\lambda_0^{(2)}$=1, $b_2$=$w_0$.

After the form is changed, the input of the embedding layer can be expressed as follows:

$$\begin{aligned}z_i^{(3)}&=\sum_{j=1}^{n^{(2)}}W_{ij}^{(2)}\lambda_j^{(2)}+b_2=\sum_{j=1}^{n^{(2)}}W_{ij}^{(2)}\lambda_j^{(2)}+W_{i0}^{(2)}\lambda_0^{(2)}\\&=\sum_{j=0}^{n^{(2)}}W_{ij}^{(2)}\lambda_j^{(2)}.\end{aligned}\tag{24}$$

The output of the embedding layer can be expressed as

$$\lambda_i^{(3)}=\phi_3(z_i^{(3)})=\phi_3\left(\sum_{j=0}^{n^{(2)}}W_{ij}^{(2)}\lambda_j^{(2)}\right).\tag{25}$$

The fourth layer of DAN is the first hidden layer, and its input is

$$z_i^{(4)}=\sum_{j=1}^{n^{(3)}}W_{ij}^{(3)}\lambda_j^{(3)}+b_3=\sum_{j=0}^{n^{(3)}}W_{ij}^{(3)}\lambda_j^{(3)}.\tag{26}$$

The output of the first hidden layer is

$$\lambda_i^{(4)}=\phi_4(z_i^{(4)})=\phi_4\left(\sum_{j=0}^{n^{(3)}}W_{ij}^{(3)}\lambda_j^{(3)}\right).\tag{27}$$

Similarly, the output of the fifth layer, namely the second hidden layer, can be expressed as

$$\lambda_i^{(5)}=\phi_5(z_i^{(5)})=\phi_5\left(\sum_{j=0}^{n^{(4)}}W_{ij}^{(4)}\lambda_j^{(4)}\right).\tag{28}$$

The input of the last layer is

$$z_i^{(l)}=\sum_{j=0}^{n^{(l-1)}}W_{ij}^{(l-1)}\lambda_j^{(l-1)}.\tag{29}$$

The output of the last layer, namely the output layer, can be expressed as

$$\lambda_i^{(l)}=\phi_l(z_i^{(l)})=\phi_l\left(\sum_{j=0}^{n^{(l-1)}}W_{ij}^{(l-1)}\lambda_j^{(l-1)}\right).\tag{30}$$

Based on the above analysis, the overall structure of DAN is as shown in Fig. 5.

### 4.3 Parameter learning

A key issue for recommendation is to define a loss function based on the observed data and continuously optimize the model parameters with the goal of minimizing error losses. In terms of model parameter learning, the existing point-by-point method uses mainly the mean square error for regression:

$$L_{\text{sqr}}=\sum_{i,j\in y\cup y^-}w_{ij}(Y_{ij}-\hat{Y}_{ij})^2,\tag{31}$$

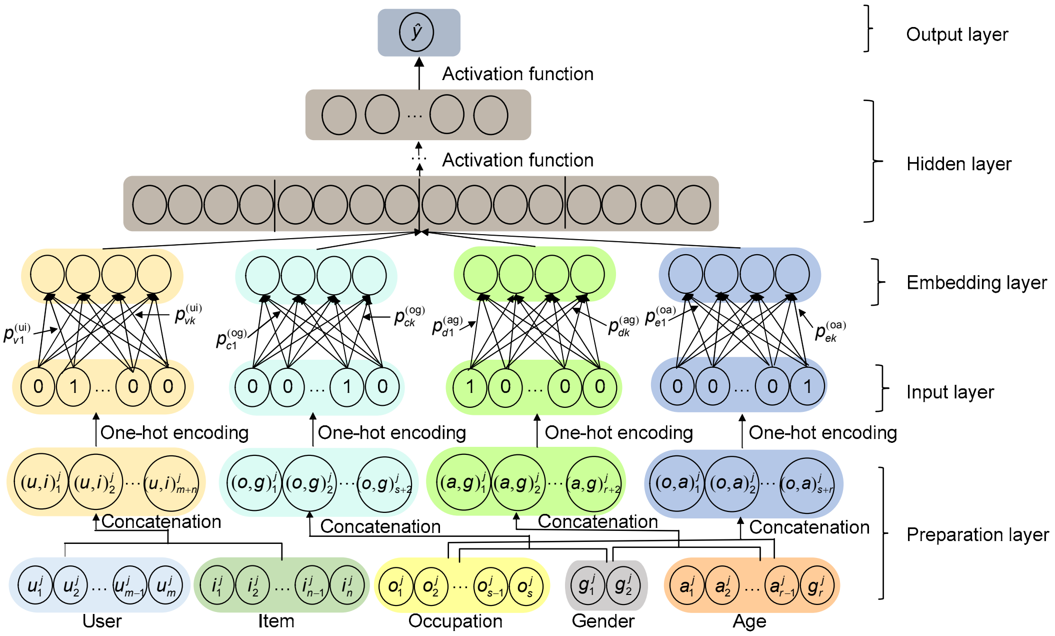


**Fig. 5 Deep association neural network model**

where $y$ represents the information that users have participated in the rating, and $y^-$ represents the negative sample information, that is, the information that users did not watch and did not participate in the rating. $w_{ij}$ represents a super parameter, which shows the weight of the training instance $(i, j)$. Considering that a recommendation with implicit feedback involves a problem of binary classification, the target value $Y_{ij}$ is represented by binary value "0" or "1"; "1" means there is interaction between $i$ and $j$, and "0" means there is no interaction. Since there are only two fixed target values for the binary classification problem, training the model with the mean square error method will cause the parameter adjustment gradient to be too large or too small, which will further lead to deviations in the prediction results. Thus, for the binary classification problem, it is not suitable to take the mean square error as the loss function.

Essentially, for the binary classification problem, the output of neurons in the output layer reflects the interaction probability between training instances. Since there are only two categories to choose, the probability distribution of the output layer obeys a Bernoulli distribution. $Y\in\{0, 1\}$ is used to represent the category of the target value. Then the probability that the actual output value belongs to "1" or "0" is expressed as follows:

$$p(Y=1\,|\,\boldsymbol{\lambda}^{L-1},\boldsymbol{W}^{L-1})=\lambda_1^{(L)}, \tag{32}$$

$$p(Y=0\,|\,\boldsymbol{\lambda}^{L-1},\boldsymbol{W}^{L-1})=1-\lambda_1^{(L)}, \tag{33}$$

where $\boldsymbol{\lambda}^{L-1}$ is the output vector of the last hidden layer, and $\boldsymbol{W}^{L-1}$ is the connection weight matrix from the last hidden layer to the output layer. After the above two formulas are combined, they can be expressed by the following formula:

$$p(Y\,|\,\boldsymbol{\lambda}^{L-1},\boldsymbol{W}^{L-1})=(\lambda_1^{(L)})^Y(1-\lambda_1^{(L)})^{1-Y}. \tag{34}$$

Therefore, we can define the negative logarithmic likelihood function as the loss function based on the above equation, and the final form of the loss function can be obtained as follows:

$$\begin{aligned}L&=-\sum\log(p(Y\,|\,\boldsymbol{\lambda}^{L-1},\boldsymbol{W}^{L-1}))\\&=-(1-Y)\log(1-\lambda_1^L)-Y\log\lambda_1^L.\end{aligned} \tag{35}$$

In this way, the parameter optimization problem of the model is transformed into the problem of minimizing the loss function value by adjusting the parameters of each layer. We can use the stochastic gradient descent algorithm to solve the problem. By applying such a probabilistic approach to the model, we transform the implicit feedback recommendation problem into a binary classification problem.

## 5 Experiments

We conducted a series of experiments based on an actual data set to answer the following questions:

RQ1: Is the DAN method proposed in this study better than traditional methods?

RQ2: Is there necessarily a linear relationship between the number of negative samples in training samples and the performance of the model?

RQ3: Of the additional feature nodes and deep network structure, which is useful?

### 5.1 Experimental settings

Data set: We used the public data set "MovieLens," which has been widely used in projects related to information filtering, CF, and recommendation systems. The version we used includes 100 000 ratings from 943 users, who rated 1682 movies on a scale of 1 to 5, and each user had at least 20 data ratings. The data set contains user id, item id, rating, timestamp, age, gender, occupation, zip code, and other information. Although it is an explicit feedback data set, we can still use it to learn some implicit feedback. We converted the data set into implicit data, where "1" means that there is an interaction between the user and the item, and "0" means that there is no interaction.

Evaluation schemes: To evaluate the performance of the recommendation, we used the leave-one-out method, which is used to train and test the learner, and has been widely used (Bayer et al., 2017; Guo et al., 2017). The advantage of the leave-one-out method is that it is not affected by the random sample division approach, and the largest number of available samples is used for training in each iteration. Specifically, the settings in our experiments are as follows. For each user, based on the time when the user rated the items, we took its last interaction as the test set and the rest of the data as the training set. In the evaluation process, we followed the main idea behind the common strategies (Elkahky et al., 2015; He et al., 2016); i.e., for each user, we randomly selected 20 items that did not interact with the user and ranked the test item among the 20 items. The model evaluation results were measured by the hitting ratio (HR) and normalized damage cumulative gain (NDCG) (He et al., 2017); if there was no special mention, we truncated the ranking list at 10 for both metrics. The higher the values of HR@10 and NDCG@10, the better the recommendation effect. HR intuitively measures whether the test item is present on the top-10 list, and NDCG accounts for the position of the hit by assigning higher scores to hits at top ranks. In this experiment, the two metrics HR and NDCG were calculated for each test user and the mean values obtained.

Baselines: The DAN approach proposed in this study was compared with the following methods:

1. userKNN: This is the standard user-based CF algorithm. Compared with deep neural network recommendation, it is the most traditional recommendation algorithm.

2. GMF (generalized matrix factorization algorithm): The algorithm is the interaction of the underlying factors of the user and the item. It decomposes the high-dimensional user-item scoring matrix into two low-dimensional factor matrices for the user and the item, and uses the inner product of these two matrices to predict the user's rating on unknown items.

3. MLP: This algorithm is a feedforward neural network model with hidden layers, mapping multiple input data sets to the output.

4. NCF: In terms of modeling the potential structure of the user/item, this algorithm unifies the linear modeling advantages of matrix decomposition and the nonlinear modeling advantages of MLP.

Parameter settings: To determine the parameters of the method proposed in this paper, we extracted interactive data for each user as test data. We set the loss function of the model and used the corresponding optimizer to optimize the loss function to learn and adjust the parameters. We sampled two negative samples for each positive sample. For the model trained from the beginning, the Gaussian distribution with mean of 0 and standard deviation of 0.01 was used to randomly initialize the model parameters, and the Adam optimizer was used to optimize the model. Without special mention, DAN and MLP in this paper adopted three hidden layers, and the common tower structure was accepted (that is, the number of neurons in each layer was gradually reduced from bottom to top). To facilitate a comparison of the model's basic performance, we unified the hidden layer structure of DAN and MLP in the experiment (that is, the number of neurons in the higher hidden layer was reduced by half compared with that in the previous hidden layer).

Because the last hidden layer of the model determines the model performance, we used it as the predictive factor. In this study, if the size of the predictive factor is 2, the structure of the hidden layer is 8-4-2, and the factor size of the embedding layer is 4. We used predictive factors of [2, 4, 8, 16, 32] and batch sizes of [60, 120, 240, 480, 960] for the model evaluation. To further explore the influence of the number of predictive factors on the model performance, we fixed the batch size to 480 and tested the model performance when the predictive factors were [2, 4, 8, 16, 32]. We tested the model performance in five different cases where the recommendation list was ranked [2, 4, 6, 8, 10], and evaluated the performance of the model when the numbers of negative samples of the training samples were [2, 4, 6, 8, 10].

## 5.2 Performance comparison (RQ1)

Fig. 6 shows the changes in HR@10 and NDCG@10 with the number of predictive factors under different training batches. For the userKNN model, we tested the performance under different numbers of neighbors. The GMF model needs to first encode the original data through one-hot encoding, and then sends the encoded data to the embedding layer. For the GMF model, the model performance was tested with different numbers of neurons in the embedding layer.

First, we can see that DAN performed well on both HR and NDCG. The performance of DAN with different numbers of predictive factors was better than that of NCF. This demonstrates the advantage of DAN, which uses the association feature to learn implicit feedback. Second, NCF, GMF, and MLP also had good performance, of which NCF was better than GMF and MLP, because NCF combines GMF's ability in linear expression with MLP's nonlinear expression ability. In general, GMF was slightly better than MLP, but note that MLP can be further improved by adding hidden layers; however, we discuss only the case of three hidden layers here. Under the same conditions, compared with many other models, the traditional KNN model had the worst performance, which further verifies that the performance of a deep neural network is better than that of the traditional method. Moreover, we found that there was no absolute positive or negative correlation between the number of predictive factors and the model performance, which was inconsistent with our expectation. Within a certain range, the number of predictive factors that leads to the best performance is neither the highest, nor the lowest.

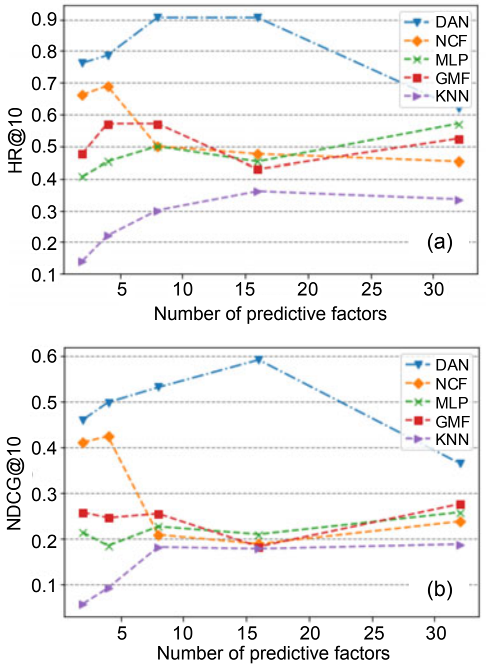


**Fig. 6 HR@10 (a) and NDCG@10 (b) w.r.t. the number of predictive factors under different training batches**

Fig. 7 shows that the same training batch, learning rate, and ranking list were sampled in the experiment, and the performance of various models was different when different numbers of predictive factors were adopted. The training batch was selected as 480, the learning rate was 0.01, 10 was selected from the ranking list, and the numbers of predictive factors were selected as 2, 4, 8, 16, 32, respectively. This experimental setup was to explore the influence of the number of predictive factors on the model performance. Overall, we found that the performance of DAN was still better than those of other models. Similar to the results in Fig. 6, NCF, GMF, and MLP had good performance, and the performance of the traditional KNN model was unfavorable. However, the overall performance of MLP_5 was better than that of MLP_2, and even better than those of NCF and GMF at some nodes. The reason may be that MLP_5 integrates auxiliary information based on the user and item, such as gender, age, and occupation. Compared with the traditional method, MLP_5 increased both the depth and width of the model, enhancing the model's ability to learn implicit feedback from large-scale data sets. Comparing Figs. 6 and 7, we found that the number of predictive factors (which also reflects the number of neurons in each hidden layer) is an important factor in determining the performance of the model. In contrast, other parameters had limited effect on the model performance.

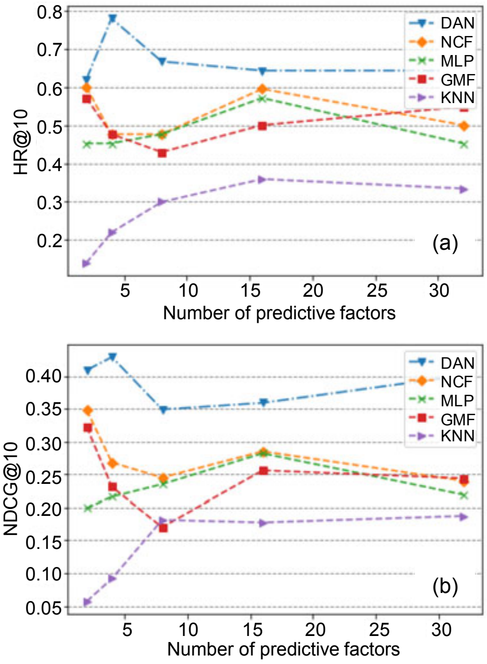


**Fig. 7 HR@10 (a) and NDCG@10 (b) w.r.t. the number of predictive factors**

Fig. 8 shows the performance of the top-*k* recommendation list. The numbers of interceptions in the ranking list were 2, 4, 6, 8, and 10. As can be seen, the performance for DAN was still in an optimal position. The performances of NCF, MLP, and GMF were also good, and the performance of userKNN was the worst. These results were consistent with the above analysis. When there was an increase in *k*, HR and NDCG also gradually increased, and the performance of the model was gradually improved.

### 5.3 Relationship between the number of negative samples and model performance (RQ2)

To illustrate the influence of negative sampling on the model performance, different numbers of negative samples were used to compare the performance. Under the premise that other parameters remain unchanged, we set the number of negative samples as 2, 4, 6, 8, and 10, respectively, to observe the relationship between the number of negative samples and the model performance.

As shown in Fig. 9, the relationship between the number of negative samples and the model performance is not very clear. What we can see is that

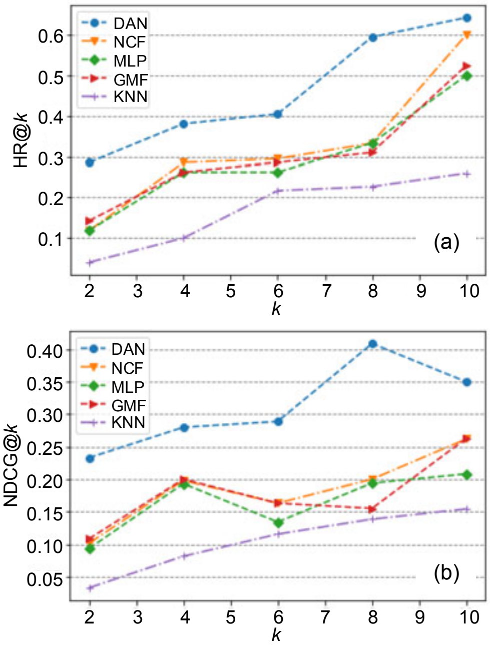


**Fig. 8 HR@*k* (a) and NDCG@*k* (b) w.r.t. *k***

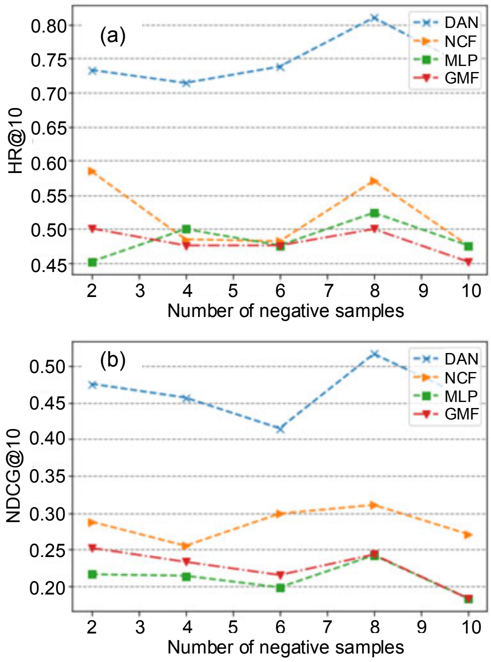


**Fig. 9 HR@10 (a) and NDCG@10 (b) w.r.t. the number of negative samples**

DAN still had a superior performance. Overall, in the local range, the more negative samples there were, the better the model performance was. However, the relationship is not absolute; when the number of negative samples continued to increase, the performance showed sign of inversion.

### 5.4 Deep learning vs. wide learning (RQ3)

At present, deep structure neural networks have been applied in many fields and achieved a breakthrough in large-scale data processing. Deep learning uses a deep network structure to learn the essential characteristics and implicit feedback of a data set. Another way is to try to obtain more effective recommendations by adding input data and feature nodes, that is, through increasing the width of the bottom layer of the model. It is necessary to further analyze the different performances of these two approaches. Based on the classical MLP model, we analyzed the performance of the deep neural network in depth using fewer feature nodes on the bottom layer (shown as "deep and narrow" in the network structure) and a shallow network with more feature nodes on the bottom layer (shown as "wide and shallow"). The experimental results are shown in Tables 3–6. With regard to the underlying data input in the experiment, the deep neural network with a "deep and narrow" structure selected the "user" and "item" indicators in the MovieLens data set, but the shallow network with a "wide and shallow" structure selected five indicators: user, item, occupation, gender, and age. For ease of expression, we expressed the network model with two input indicators and *n* hidden layers as MLP_2_*n*, and the deep network model with five input indicators and *n* hidden layers as MLP_5_*n*.

These tables showed that even for the model with the same input indicators, the increase in the number of layers was also conducive to an improvement of model performance. Under the same conditions, the average HR and average NDCG of MLP_5 were higher than the counterparts of MLP_2. This result has two implications: on one hand, it shows the effectiveness of using the deep structure model for recommendation; on the other hand, it shows that increasing the underlying width of the deep neural network (that is, enriching the feature categories of the underlying input) can improve the model performance based on the original deep neural network. Note that even if the input data has abundant feature categories, if the advantages of the deep neural network are not fully used, the model performance will be inferior to that of the deep neural network model with fewer feature categories.

**Table 3 HR@10 of MLP_2_*n* with different layers**

| Number of factors | HR@10 | | | | |
|---|---|---|---|---|---|
| | MLP_2_0 | MLP_2_1 | MLP_2_2 | MLP_2_3 | MLP_2_4 |
| 1 | 0.400 | 0.405 | 0.425 | 0.452 | 0.405 |
| 2 | 0.405 | 0.452 | 0.476 | 0.524 | 0.524 |
| 4 | 0.429 | 0.452 | 0.524 | 0.548 | 0.595 |
| 8 | 0.385 | 0.405 | 0.476 | 0.550 | 0.560 |
| 16 | 0.350 | 0.400 | 0.452 | 0.500 | 0.550 |
| 32 | 0.381 | 0.476 | 0.548 | 0.560 | 0.561 |
| 64 | 0.333 | 0.405 | 0.405 | 0.545 | 0.530 |
| 128 | 0.400 | 0.452 | 0.476 | 0.490 | 0.500 |
| Average | 0.385 | 0.431 | 0.473 | 0.521 | 0.528 |

**Table 4 NDCG@10 of MLP_2_*n* with different layers**

| Number of factors | NDCG@10 | | | | |
|---|---|---|---|---|---|
| | MLP_2_0 | MLP_2_1 | MLP_2_2 | MLP_2_3 | MLP_2_4 |
| 1 | 0.364 | 0.204 | 0.217 | 0.350 | 0.453 |
| 2 | 0.189 | 0.339 | 0.235 | 0.235 | 0.265 |
| 4 | 0.153 | 0.223 | 0.158 | 0.259 | 0.222 |
| 8 | 0.202 | 0.181 | 0.229 | 0.231 | 0.268 |
| 16 | 0.180 | 0.172 | 0.195 | 0.205 | 0.250 |
| 32 | 0.174 | 0.202 | 0.254 | 0.204 | 0.244 |
| 64 | 0.141 | 0.170 | 0.200 | 0.240 | 0.222 |
| 128 | 0.216 | 0.176 | 0.220 | 0.215 | 0.202 |
| Average | 0.202 | 0.208 | 0.214 | 0.242 | 0.266 |

**Table 5 HR@10 of MLP_5_*n* with different layers**

| Number of factors | HR@10 | | | | |
|---|---|---|---|---|---|
| | MLP_5_0 | MLP_5_1 | MLP_5_2 | MLP_5_3 | MLP_5_4 |
| 1 | 0.400 | 0.524 | 0.571 | 0.571 | 0.570 |
| 2 | 0.405 | 0.595 | 0.619 | 0.643 | 0.640 |
| 4 | 0.405 | 0.518 | 0.524 | 0.590 | 0.650 |
| 8 | 0.476 | 0.500 | 0.500 | 0.500 | 0.595 |
| 16 | 0.476 | 0.452 | 0.476 | 0.548 | 0.550 |
| 32 | 0.476 | 0.480 | 0.500 | 0.548 | 0.561 |
| 64 | 0.524 | 0.500 | 0.500 | 0.548 | 0.619 |
| 128 | 0.571 | 0.571 | 0.619 | 0.640 | 0.643 |
| Average | 0.467 | 0.518 | 0.539 | 0.574 | 0.604 |

**Table 6 NDCG@10 of MLP_5_*n* with different layers**

| Number of factors | NDCG@10 | | | | |
|---|---|---|---|---|---|
| | MLP_5_0 | MLP_5_1 | MLP_5_2 | MLP_5_3 | MLP_5_4 |
| 1 | 0.212 | 0.456 | 0.312 | 0.542 | 0.266 |
| 2 | 0.246 | 0.240 | 0.293 | 0.278 | 0.540 |
| 4 | 0.210 | 0.290 | 0.262 | 0.235 | 0.239 |
| 8 | 0.203 | 0.193 | 0.227 | 0.239 | 0.223 |
| 16 | 0.221 | 0.191 | 0.209 | 0.277 | 0.249 |
| 32 | 0.227 | 0.174 | 0.272 | 0.207 | 0.200 |
| 64 | 0.204 | 0.220 | 0.264 | 0.218 | 0.217 |
| 128 | 0.238 | 0.270 | 0.303 | 0.266 | 0.333 |
| Average | 0.220 | 0.254 | 0.268 | 0.283 | 0.283 |

## 6 Conclusions and future work

In this work, we explored deep association networks (DAN) for recommendations. We designed a generic framework for DAN to simulate user–item interactions. The proposed model framework is simple and universal. We put forward a new way to study deep learning recommendations and provide guidance for further study of deep learning and recommendation systems.

The proposed method can be used for personalized recommendations in large-scale e-commerce platforms, and in various Internet life service platforms based on various data derived from users and items, such as commodity recommendations, movie or music recommendations, and appropriate merchant recommendations.

In future work, we will explore the internal personality characteristics of users based on the joint features of users, and promote this research to the fields of e-commerce recommendations, advertising promotion, and personalized user customizations, etc. At the same time, we will extend the DAN model to the combination of three or more feature categories, and further study the interaction between multiple input categories, which would help optimize the combination of feature categories in further improving the model performance. In addition, we are particularly interested in tracking users' changes in preferences, and in exploring the factors that lead to such changes, such as the user's own factors, or some other external factors. In this way, this study puts forward a new idea for research into recommendation systems.

## Contributors

Xu-na WANG designed the research and drafted the manuscript. Qing-mei TAN helped organize the manuscript. Xu-na WANG revised and finalized the paper.

## Compliance with ethics guidelines

Xu-na WANG and Qing-mei TAN declare that they have no conflict of interest.

## References

Aiolli F, 2014. Convex AUC optimization for top-*n* recommendation with implicit feedback. Proc 8th ACM Conf on Recommender Systems, p.293-296. https://doi.org/10.1145/2645710.2645770

Barkan O, Koenigstein N, 2016. Item2Vec: neural item embedding for collaborative filtering. Proc IEEE 26th Int Workshop on Machine Learning for Signal Processing, p.1-6. https://doi.org/10.1109/MLSP.2016.7738886

Bayer I, He XN, Kanagal B, et al., 2017. A generic coordinate descent framework for learning from implicit feedback. Proc 26th Int Conf on World Wide Web, p.1341-1350. https://doi.org/10.1145/3038912.3052694

Buettner R, 2016. Predicting user behavior in electronic markets based on personality-mining in large online social networks. *Electron Mark*, 27(3):247-265. https://doi.org/10.1007/s12525-016-0228-z

Cao YL, Li WL, Zheng DX, 2018. An improved neighborhood-aware unified probabilistic matrix factorization recommendation. *Wirel Pers Commun*, 102(4):3121-3140. https://doi.org/10.1007/s11277-018-5332-2

Cheng G, Yang CY, Yao XW, et al., 2018. When deep learning meets metric learning: remote sensing image scene classification via learning discriminative CNNs. *IEEE Trans Geosci Remote Sens*, 56(5):2811-2821. https://doi.org/10.1109/TGRS.2017.2783902

del Corso GM, Gianna M, Romani F, 2019. Adaptive nonnegative matrix factorization and measure comparisons for recommender systems. *Appl Math Comput*, 354:164-179. https://doi.org/10.1016/j.amc.2019.01.047

Elkahky AM, Song Y, He XD, 2015. A multi-view deep learning approach for cross domain user modeling in recommendation systems. Proc 24th Int Conf on World Wide Web, p.278-288. https://doi.org/10.1145/2736277.2741667

Fu MS, Qu H, Yi Z, et al., 2019. A novel deep learning-based collaborative filtering model for recommendation system. *IEEE Trans Cybern*, 49(3):1084-1096. https://doi.org/10.1109/TCYB.2018.2795041

Guo HF, Tang RM, Ye YM, et al., 2017. DeepFM: a factorization-machine based neural network for CTR prediction. https://arxiv.org/abs/1703.04247

Ha T, Lee S, 2017. Item-network-based collaborative filtering: a personalized recommendation method based on a user's item network. *Inform Process Manag*, 53(5):1171-1184. https://doi.org/10.1016/j.ipm.2017.05.003

He XN, Zhang HW, Kan MY, et al., 2016. Fast matrix factorization for online recommendation with implicit feedback. Proc 39th Int ACM SIGIR Conf on Research and Development in Information Retrieval, p.549-558. https://doi.org/10.1145/2911451.2911489

He XN, Liao LZ, Zhang HW, et al., 2017. Neural collaborative filtering. Proc 26th Int Conf on World Wide Web, p.173-182. https://doi.org/10.1145/3038912.3052569

He XN, Du XY, Wang X, et al., 2018. Outer product-based neural collaborative filtering. https://arxiv.org/abs/1808.03912

Hernando A, Bobadilla J, Ortega F, 2016. A non negative matrix factorization for collaborative filtering recommender systems based on a Bayesian probabilistic model. *Knowl-Based Syst*, 97:188-202. https://doi.org/10.1016/j.knosys.2015.12.018

Hossain MS, Muhammad G, 2018. Emotion recognition using deep learning approach from audio-visual emotional big data. *Inform Fus*, 49:69-78. https://doi.org/10.1016/j.inffus.2018.09.008

Hsu CC, Yeh MY, Lin SD, 2018. A general framework for implicit and explicit social recommendation. *IEEE Trans Knowl Data Eng*, 30(12):2228-2241. https://doi.org/10.1109/TKDE.2018.2821174

Jia XW, Li XY, Kang L, et al., 2016. Collaborative restricted Boltzmann machine for social event recommendation. IEEE/ACM Int Conf on Advances in Social Networks Analysis and Mining, p.402-405. https://doi.org/10.1109/ASONAM.2016.7752265

Jung JJ, 2012. Attribute selection-based recommendation framework for short-head user group: an empirical study by MovieLens and IMDB. *Expert Syst Appl*, 39(4):4049-4054. https://doi.org/10.1016/j.eswa.2011.09.096

Knoll J, Stübinger J, Grottke M, 2019. Exploiting social media with higher-order factorization machines: statistical arbitrage on high-frequency data of the S&P 500. *Quant Finan*, 19(4):571-585. https://doi.org/10.1080/14697688.2018.1521002

Li Y, Wang SH, Pan Q, et al., 2019. Learning binary codes with neural collaborative filtering for efficient recommendation systems. *Knowl-Based Syst*, 172:64-75. https://doi.org/10.1016/j.knosys.2019.02.012

Li ZC, Tang JH, 2017. Weakly supervised deep matrix factorization for social image understanding. *IEEE Trans Image Process*, 26(1):276-288. https://doi.org/10.1109/TIP.2016.2624140

Liu JT, Wu CH, 2017. Deep learning based recommendation: a survey. Int Conf on Information Science and Applications, p.451-458. https://doi.org/10.1007/978-981-10-4154-9_52

Liu WB, Wang ZD, Liu XH, et al., 2017. A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11-26. https://doi.org/10.1016/j.neucom.2016.12.038

Liu Y, Li LF, Liu J, 2018. Bilateral neural embedding for collaborative filtering-based multimedia recommendation. *Multim Tools Appl*, 77(10):12533-12544. https://doi.org/10.1007/s11042-017-4902-8

Lu J, Wu DS, Mao MS, et al., 2015. Recommender system application developments: a survey. *Dec Supp Syst*, 74:12-32. https://doi.org/10.1016/j.dss.2015.03.008

Luo L, Xie HR, Rao YH, et al., 2018. Personalized recommendation by matrix co-factorization with tags and time information. *Expert Syst Appl*, 119:311-321. https://doi.org/10.1016/j.eswa.2018.11.003

Ma C, Zhang YX, Wang QL, et al., 2018. Point-of-interest recommendation: exploiting self-attentive autoencoders with neighbor-aware influence. Proc 27th ACM Int Conf on Information and Knowledge Management, p.697-706. https://doi.org/10.1145/3269206.3271733

Marchi E, Vesperini F, Eyben F, et al., 2015. A novel approach for automatic acoustic novelty detection using a denoising autoencoder with bidirectional LSTM neural networks. IEEE Int Conf on Acoustics, Speech and Signal Processing, p.1996-2000. https://doi.org/10.1109/ICASSP.2015.7178320

Noda K, Yamaguchi Y, Nakadai K, et al., 2015. Audio-visual speech recognition using deep learning. *Appl Intell*, 42:722-737. https://doi.org/10.1007/s10489-014-0629-7

Pan J, Zi YY, Chen JL, et al., 2017. LiftingNet: a novel deep learning network with layerwise feature learning from noisy mechanical data for fault classification. *IEEE Trans Ind Electron*, 65(6):4973-4982. https://doi.org/10.1109/TIE.2017.2767540

Pan WK, Chen L, Ming Z, 2019. Personalized recommendation with implicit feedback via learning pairwise preferences over item-sets. *Knowl Inform Syst*, 58(2):295-318. https://doi.org/10.1007/s10115-018-1154-5

Verstrepen K, Bhaduriy K, Cule B, et al., 2017. Collaborative filtering for binary, positiveonly data. *ACM SIGKDD Explor Newsl*, 19(1):1-21. https://doi.org/10.1145/3137597.3137599

Wang XN, Tan QM, Zhang LF, 2020. A deep neural network of multi-form alliances for personalized recommendations. *Inform Sci*, 531:68-86. https://doi.org/10.1016/j.ins.2020.03.062

Wu L, Chen EH, Liu Q, et al., 2012. Leveraging tagging for neighborhood-aware probabilistic matrix factorization. Proc 21st ACM Int Conf on Information and Knowledge Management, p.1854-1858. https://doi.org/10.1145/2396761.2398531

Xiao YY, Wang GW, Hsu CH, et al., 2018. A time-sensitive personalized recommendation method based on probabilistic matrix factorization technique. *Soft Comput*, 22(20):6785-6796.

https://doi.org/10.1007/s00500-018-3406-4

Xiong RB, Wang J, Zhang N, et al., 2018. Deep hybrid collaborative filtering for Web service recommendation. *Expert Syst Appl*, 110:191-205. https://doi.org/10.1016/j.eswa.2018.05.039

Yeung CH, 2016. Do recommender systems benefit users? A modeling approach. *J Stat Mech Theory Exp*, 4:2-13. https://doi.org/10.1088/1742-5468/2016/04/043401

Zheng Y, Tang BS, Ding WK, et al., 2016. A neural autoregressive approach to collaborative filtering. https://arxiv.org/abs/1605.09477

Zhou F, Zhou HM, Yang ZH, et al., 2018. EMD2FNN: a strategy combining empirical mode decomposition and factorization machine based neural network for stock market trend prediction. *Expert Syst Appl*, 115:136-151. https://doi.org/10.1016/j.eswa.2018.07.065