BayDNN: Friend Recommendation with Bayesian Personalized Ranking Deep Neural Network

Daizong Ding[1], Mi Zhang[1], Shao-Yuan Li[2], Jie Tang[3], Xiaotie Chen[1], Zhi-Hua Zhou[2]
[1]Shanghai Key Laboratory of Intelligent Information Processing, School of Computer Science, Fudan University, China
{17110240010,mi_zhang,13300180119}@fudan.edu.cn
[2]National Key Lab for Novel Software Technology, Nanjing University, China
{lisy,zhouzh}@lamda.nju.edu.cn
[3]Department of Computer Science and Technology, Tsinghua University, China
jietang@tsinghua.edu.cn

ABSTRACT

Friend recommendation is a critical task in social networks. In this paper, we propose a Bayesian Personalized Ranking Deep Neural Network (BayDNN) model for friend recommendation in social networks. BayDNN first extracts latent structural patterns from the input network data and then use the Bayesian ranking to make friend recommendations. With BayDNN we achieve significant performance improvement on two public datasets: Epinions and Slashdot. For example, on Epinions dataset, BayDNN significantly outperforms the state-of-the-art algorithms, with a 5% improvement on NDCG over the best baseline.

The advantages of the proposed BayDNN mainly come from a novel Bayesian personalized ranking (BPR) idea, which precisely captures the users' personal bias based on the extracted deep features, and its underlying convolutional neural network (CNN), which offers a mechanism to extract latent deep structural feature representations of the complicated network data. To get good parameter estimation for the neural network, we present a fine-tuned pre-training strategy for the proposed BayDNN model based on Poisson and Bernoulli probabilistic models.

CCS CONCEPTS

•**Information systems** → **Social advertising;**

KEYWORDS

Bayesian Personalized Ranking Deep Neural Network; Probabilistic Model; Pre-training Strategy

1 INTRODUCTION

Recently, online social networks (OSN) such as Facebook, Twitter and Weibo have been growing exponentially. People from all over the world get connected to each other online, making friends with those sharing similar interest [1]. As the cornerstone of social networks, friendship and its formation has attracted tremendous attention from both academia and industry [20, 27]. For example, Granovetter categorized friendship into strong and weak, and found that novel information flows to individuals through weak ties rather than strong ties [9]. In online social networks, one phenomenon is that the leading reason for users to create new friendships is due to recommendation [28]. Therefore, friend recommendation becomes an essential task of social networks.

In this paper, our goal is to design a high-accuracy method for friend recommendation in online social networks. Different from previous works [11, 44] that rely on specific context information, we aim to design a method that is general enough to be applied to different social networks. We propose a deep neural network for friend recommendation using only network structure information.

Previous work on friend recommendation. From the algorithmic perspective, existing methodologies for the friend recommendation problem roughly fall into three categories: classification, fitting, and ranking [10, 22]. The classification method treats friendship between users as a binary classification problem, and trains a classifier to predict the likelihood of a friendship to be created between users based on pre-defined features. The fitting method transforms the friendship between users into a real-valued rating matrix, and utilizes collaborative filtering approaches such as matrix factorization to predict the probabilities of unknown friendships. Taking into account the common imbalance issue in most OSN [13], i.e., the amount of friendship is usually much less than that of non-friendships, which would lead the classification or fitting methods biased towards non-friendship, the ranking method has been proposed. Regarding friend recommendation as a learning to rank task — i.e., for each user, by ranking the probability of friendship for the users connected to it larger than that of users not connected to it — the ranking method has been proved to be effective combating the imbalance issue. Among them, the Bayesian personalized ranking (BPR) model [18, 30, 34] defined Bayesian pairwise ranking relation between users, i.e., the friendship probability of observed friends should be larger than the unobserved users pairs, was validated to achieve quite good performance.

Previous work on deep neural networks (DNN). Recently, deep neural networks have achieved pervasively excellent performance in various fields including image classification and phrase representations [4, 17]. Compared to traditional methods such as kernel machines [42] or matrix factorization [16] with shallow architecture or limited levels of feature extraction, deep neural networks like CNN with properly designed structure and careful parameter

CIKM'17 , Singapore, Singapore

DOI: 10.1145/3132847.3132941

optimization achieve superior advantages. However, till now few deep learning study has been conducted to address the task of friend recommendation.

To achieve high-accuracy recommendations, one straightforward idea is to apply deep neural network (DNN) to friend recommendation. However, the advantage of DNN is to learn a better feature representation, but it fails to consider the essential imbalance issue in friend recommendation. While a few efforts have been made using BPR (Bayesian Personalized Ranking) or DNN (Deep Neural Network) [41] for friend recommendation, no work has been done combining the strength of these two ideas.

Our solution and contributions. In this paper, we propose a deep Bayesian Personalized Ranking Deep Neural Network (BayDNN) to combine the advantage of BPR (Bayesian Personalized Ranking) and DNN (Deep Neural Network). The proposed BayDNN seamlessly combines DNN and Bayesian ranking, and integrates CNN at the stage of feature representation learning, which not only reduces the number of parameters compared to traditional full connect neural network, but also improves the effectiveness on feature mapping.

The proposed BayDNN model has several unique advantages compared with the previous methods. First, the model uses convolutional neural network (CNN) to extract latent deep structural feature representations of the complicated network data. Second, it uses a novel Bayesian personalized ranking learning algorithm to better capture users' personal bias based on the extracted deep features. Third, to avoid poor parameter estimation, we design a fine-tuned pre-training strategy for the proposed BayDNN model based on Poisson and Bernoulli probabilistic models. Moreover, to enhance the BPR, we group the friendship between users into three sets, *positive*, *negative*, and *unknown*. And then, during the learning process, we enforce the *positive* friendship to be ranked before the *unknow/negative* ones, and the *unknown* friendship to be ranked before the *negative* ones.

We conduct experiments on multiple real datasets. The results show that BayDNN not only extracts better feature presentation, which is different from classical or neighbor based methods, but also performs well on large-scale datasets in friend recommending in OSN. In terms of accuracy performance it outperforms the best baseline and achieves 5% improvement on NDCG. In terms of time cost for the model training, with the proposed pre-training strategy the number of convergence epoches is largely reduced by 92.5%.

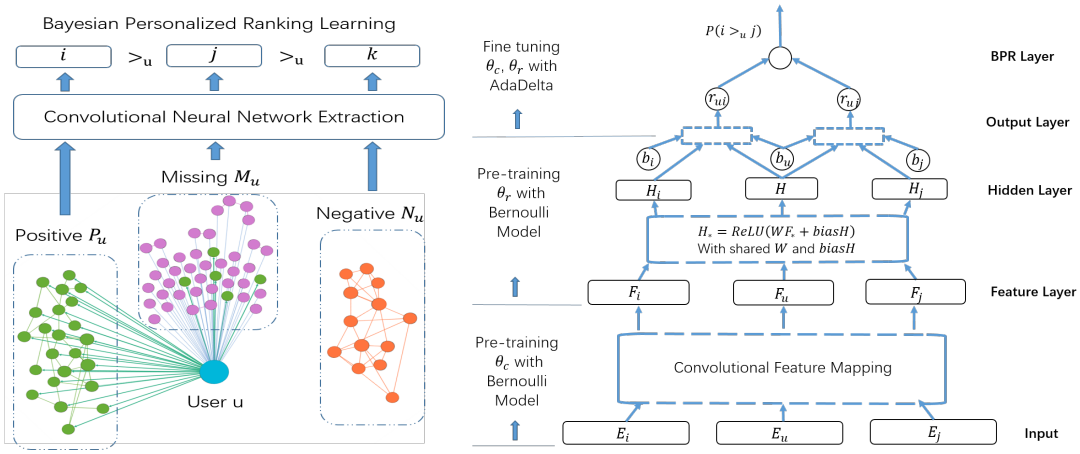
Organization. The rest of the paper is organized as following. We briefly review related work in Section 2. In Section 3 we propose our deep Bayesian Personalized Ranking Deep Neural Network (BayDNN) model and specify our pre-training strategies for BayDNN. In Section 4 we design several experiments to validate our model and conclude in Section 5.

2 RELATED WORK

We review existing methodologies for friend recommendation, highlighting the most relevant ones and explaining how our work distinguishes from them. Then, we briefly summarize recent development on deep neural networks.

Friend Recommendation. At the high level, methodologies dealing with friend recommendation can be grouped into three categories [22]: classification, fitting and ranking. The classification methods are based on extracted features, e.g., features between two nodes like path-based metric Katz [14] or neighbor-based metric Adamic/Adar [23]. By regarding the friendship status between two users as a binary classification task, they train classifiers such as SVM [8], logistic regression [29], and factor graph [40] to predict the friendship values between users. The fitting methods represent the friendship between users bu real values and try to approximate the values for observed friendship as close as they can, e.g., [12, 25] used matrix factorization to predict the value of unknown friendship. Considering that the value of friendship probabiliy is non-negative, Non-negative Matrix Factorization (NMF) was introduced by [19] constraining that two factorized matrix are non-negative, which showed better performance in non-negative data [3] compared to MF. However the classification and fitting methods fail to deal with the severe data imbalance issue due to the sparseness of OSN data, i.e., the amount of observed friendship is much smaller than the unobserved ones [26]. This leads to the models biased towards making low friendship probability[31]. Hence ranking methods that regard this problem as a learning to rank task are proposed. For each user, by predicting the likelihood of friendship between friends larger than non-friends, ranking methods are proved to be effective combating the imbalance issue [18, 30, 34]. Widely used in item recommendation tasks, one popular Bayesian personalized ranking (BPR) [30, 34] model was integrated with matrix factorization [18], called Bayesian Personalized Ranking Matrix Factorization (BPRMF). They defined a Bayesian pairwise ranking relation, i.e., observed data to be rated before unobserved data by user-item linkage probability, which is calculated by matrix factorization. We are inspired by this idea and apply the BPR idea to friendship network, by combining it with a DNN model rather than a shallow MF model.

Deep Neural Network. Recently, deep neural networks (DNN) have archived significant success in many machine learning and data mining tasks, e.g., image labeling and voice recognizing [38]. Bengio et al. [2] pointed out that the shallow architecture of kernel machines [42] and matrix factorization [16] restricts their learning performance; meanwhile DNN achieves superior performance with properly designed structure and carefully chosen parameters. Especially in non-linear feature mapping, convolutional neural network outperforms most models in image [5], text [36] and other fields. In existing literature, few studied have been conducted to apply DNN methods to the friend recommendation task. In fact, there is no straightforward way to do this. One reason is that there exists no corresponding pairwise ranking model for neural networks. Another reason is that DNN usually needs proper pre-training strategy to avoid poor parameter initialization, e.g., use pre-trained deep neural network like VGG-Net to initialize parameters of model [24, 32] However, it is unclear which pre-training strategy should be used for DNN in friend recommendation. Wang et al. [41] tried to use a four-layer neural network with a simple pre-training strategy called Pairwise Input Neural Network (PINN) to do link prediction, by training and predicting the probability of a link point-wisely. However, it did not achieve significant performance improvement.



(a) Overview (b) BayDNN

Figure 1: Bayesian Personalized Ranking Deep Neural Network (BayDNN)

3 THE PROPOSED FRAMEWORK

In this section, we first introduce the pair-wise ranking task of the friend recommendation problem. Then we propose the deep Bayesian Personalized Ranking Deep Neural Network (BayDNN) model and describe the pre-training strategies.

3.1 Preliminaries

A social network can be denoted as a graph $G =< V, E >$, where V is a set of $|V| = N$ users, and $E \in \{0,1\}^{N\times N}$ represents the undirected friendship structure between users. Notation $e_{ui} = E(u,i) = 1$ denotes that users $u \in V$ and $i \in V$ are friends, while $e_{ui} = E(u,i) = 0$ denotes that the friendship between u and i is unobserved. In practice, E is usually very sparse for two reasons: *Dunbars number* [6], which suggests that the number of one person's stable social relationships lies between 100 and 230, which is much smaller compared to the whole network; and *missing links*, which means that the observed relationships in an online social network usually only represent a limited part of one's real social circles [26].

We define a friendship probability matrix $R \in [0,1]^{N\times N}$, where $r_{ui} = R(u,i) = P(e_{ui} = 1)$ denotes the probability that a friendship between user u and i is formed. In this way, for each user u, all the other users can be grouped into two disjoint sets—i.e., a set with *positive friends* $\mathcal{P}_u = \{i|e_{ui} = 1\}$ and the rest with *unobserved friends* $\mathcal{U}_u = \{i|e_{ui} = 0\}$. For the task of friend recommendation, our idea is to construct a ranking model that is able to rank the positive friends before the unobserved friends. Inspired by [37], we further divide the unobserved friends $\mathcal{U}_u$ into two sets, the unknown friends $\mathcal{M}_u$ and the high probably negative friends $\mathcal{N}_u$, i.e., $\mathcal{U}_u = \mathcal{M}_u \cup \mathcal{N}_u$. In this work we construct the set of *negative friends* $\mathcal{N}_u$ by selecting the users that can not reach u through 6 people (i.e., within 6 hops in the social network). It has been concluded that people can on average connect to another through 6 people [21], which means the probability for those people being friends is quite low. Based on this division, e.g., for any other users i and j, if $i \in \mathcal{P}_u$ and $j \in \mathcal{U}_u$, or $i \in \mathcal{M}_u$ and $j \in \mathcal{N}_u$, the probability of friendship r_{ui} should be greater than r_{uj}. To describe this relation we define a *Partial Relation* $i >_u j$.

Definition 1. $i >_u j$: For a user u, and two other users i and j in the social network, i has a higher probability of friendship with u than j, i.e., $r_{ui} > r_{uj}$, we say that i and j have a *Partial Relation* on u.

Note that with this definition, partial relation on u has property of transitivity, i.e., if $i >_u j$ and $j >_u k$, we have $i >_u k$.

Based on $\mathcal{P}_u$, $\mathcal{U}_u$ and $\mathcal{N}_u$, in the social network there are two kinds of partial relation on u. First is the partial relation between $\mathcal{P}_u$ and $\mathcal{U}_u$, i.e., $\{i >_u j \mid i \in \mathcal{P}_u, j \in \mathcal{U}_u\}$. Then within $\mathcal{U}_u$ ($\mathcal{U}_u = \mathcal{M}_u \cup \mathcal{N}_u$), there is another partial relation between $\mathcal{M}_u$ and $\mathcal{N}_u$, i.e., $\{i >_u j \mid i \in \mathcal{M}_u, j \in \mathcal{N}_u\}$.The set of all the partial relations for u is depicted as $\mathcal{R}_u = \{(i,j) \mid i >_u j\}$.

In this way we formalize friend recommendation as a ranking problem. Compared to other options such as classification or rating problems that suffer from the imbalance problem, the ranking formulation can more or less avoid it [18, 30, 34]. Our model aims to maximize the ranking likelihood probability as follows:

$$\max \prod_{u\in V} \prod_{(i,j)\in\mathcal{R}_u} P(i >_u j) \tag{1}$$

3.2 Bayesian Personalized Ranking Deep Neural Network (BayDNN)

We propose a novel strategy that embed BPR learning framework in DNN to produce ranking result. Fig. 1(b) illustrates the architecture of the proposed Bayesian Personalized Ranking Deep Neural Network (BayDNN).

For user u and (i,j) BayDNN uses their relationship information $E_u, E_i, E_j = E(u,:), E(i,:), E(j,:) \in \{0,1\}^N$ as input. Specifically, for feature mapping from input to hidden layer H, we design a Convolutional Feature Mapping (CFM) structure. The details of Fig. 1(b)'s CFM part is given in Fig. 2. The output layer aims to produce

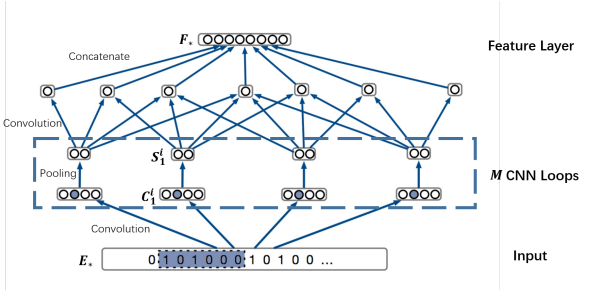


Figure 2: Convolutional Feature Mapping

the friendship probability $r_{ui} = P(e_{ui} = 1)$ and $r_{uj} = P(e_{uj} = 1)$ and Bayesian personalized ranking (BPR) layer predict the probability of this partial relation, i.e., $P(i >_u j)$.

Convolutional Feature Mapping (CFM). We use convolutional neural network (CNN) to extract latent structural patterns of the input data. Specifically, in our problem we want to extract the non-linear structural features from the relationship vector E_u for each user u. The convolution neural network (CNN) for feature mapping has been widely used in image classification, pattern recognition and so on, and has been proved to be very efficient and accurate for feature mapping [5]. We use one-dimensional CNN to be the convolution Feature Mapping layers in our model. Such a method has been also applied to synthesize lexical n-gram information in previous work such as [33] and has shown good performance. The design of the convolution feature mapping layer between the input E_u and feature H_u is given by Alg. 1 and depicted in Fig. 2.

We define a *loop* in CFM as a manipulation of one convolution layer and another pooling layer in order, where the convolution layer uses the output from last loop's pooling layer and pass its calculation result to the following pooling layer, as described in Alg. 1. Here we initialize the CFM by regarding input $E_u \in \{0,1\}^N$ for every user u as pooling result from loop 0, and pass it on to loop 1. After M loops we add another convolution layer upon CFM to reduce vectors to real value and concatenate them as extracted feature H_u. Specifically, for convolution layer in loop l, we have:

$$C_j^l = \tanh(K^l * (\mathbf{s}_j^{l-1})^T + biasF^l) \tag{2}$$

Here $(\mathbf{s}_j^{l-1})^T$ denotes the transpose of the vector $\mathbf{s}_j^{l-1}$, which is the embedding of the element j for all the output from loop $(l-1)$'s pooling layer S^{l-1}. In our model, we select an element j and its nearby elements to form $\vec{s_j}$, as shown in Fig. 2. K^l and $biasF^l$ are the convolution kernel and bias for loop i respectively. And the $tanh$ function outputs the result of convoluted feature C_j^l for element j in loop l. The most common choices of activation functions are the following: sigmoid function $\sigma(x)$, hyperbolic tangent function $tanh(x)$, and rectified linear function $ReLU(x)$. Considering that we want to learn non-linear and non-negative features of the relationship information, we choose $tanh$ function for the activation function.

The results from convolution layer are then passed to the next pooling layer, which aggregates the convoluted feature for further reduction. We define the following pooling operation in loop l:

$$S_l^l = pool^l(\mathbf{c}_j^l) \tag{3}$$

where $\mathbf{c}_j^l$ is the embedding of element j for all the output from convolution layer C^l in the same loop. Here $pool^l$ means the pooling function for loop l. In the first loop it is max function and in the following pooling layers they are $average$ function. We use this technique to avoid the sparsity of relationship information vector in first loop.

The pooling layer S^M outputs many short vectors after M loops, and the output differs with varied size of datasets. Then CFM will add a final convolution layer on it and convert these vectors to real values. We concatenate them to form a new vector as the extracted feature of CFM, defined as F_u for user u, where:

$$F_{uj} = \sigma(K^{M+1} * (\mathbf{s}_j^M)^T + biasF^{M+1}) \tag{4}$$

Here considering the input of CFM are non-negative, we use $\sigma(x)$ as activation function to output non-negative features.

Algorithm 1 CFM for Relationship Vector

Input: Link vector E_u for user u
Output: Extracted feature F_u for user u

function CFM_FORWARD(E_u)
  Let $S_0 = E_u, l = 1$
  for $l \le M, l = l + 1$
    Calculate convolution result C^l from S^{l-1} (Eq. 2)
    Calculate pooling result $S^i$from C^l (Eq. 3)
  Calculate convolution result C^{M+1} from S^M (Eq. 4)
  return $F_u = C_u^{M+1}$
end function

Bayesian Personalized Ranking (BPR) Learning. The rest part of BayDNN is based on Feature Layer H, and we adopt the Bayesian Personalized Ranking technique to produce friend recommendation via ranking learning. The model is trained to maximize the likelihood defined in Eq. 1, hence the loss function is:

$$\mathcal{L}(\theta_c, \theta_r) = -\sum_u \sum_{(i,j)\in\mathcal{R}_u} P(i >_u j) + \lambda_1 \|\theta_c\|^2 + \lambda_2 \|\theta_r\|^2 \tag{5}$$

where $\theta_c = \{K, biasF\}$, the parameters for kernels and biases in CNN layers, and $\theta_r = \{w, biasH, \mathbf{b}\}$ are the parameters in the rest part of BayDNN. The rest structure of BayDNN and inference method is defined as below.

Hidden layer: Regarding F_u, F_i, F_j as features of user u, i and j the hidden layer embedding their vectors into H_u, H_i and H_j for further calculation, e.g., for user u the hidden layer produces H_u by

$$H_u = ReLU(wF_u + biasH) \tag{6}$$

Here we use $ReLU$ function instead of non-linear function for the consideration of convergence speed, which is simply defined as $\max(0, x)$. Parameters w and $biasH$ are the weights and bias for units respectively.

Output Layer: Using H_u, H_i, H_j as input, the following layer produces the friendship probability of r_{ui} and r_{uj}, e.g.,

$$r_{ui} = \sigma(b_u + b_i + H_u^T H_i) \tag{7}$$

Here the activation function is the sigmoid function for the reason that the probability of becoming friends $r \in [0, 1]$. It's worth mentioning that, in the hidden layer all the users share the same latent representation learning parameters $\{w, biasH\}$ (Eq.6) for model simplicity. However in the output layer, we allow each user u to have its own bias parameter b_u (Eq.7) to capture the user's personal preference in making friends, which has been shown to be an important factor in recommendation problems. We validate the value of modeling users' personal bias in our experiments.

The probability of a partial relation between r_{ui} and r_{uj} is:

$$P(i >_u j) = \frac{r_{ui} - r_{uj}}{2} + 0.5 \tag{8}$$

Inference: To infer the parameters $\theta_c = \{K, biasF\}$ and $\theta_r = \{w, biasH, \mathbf{b}\}$ for our BayDNN model through Eq.5, we employ back propagation (BP) algorithm by using stochastic gradient descent (SGD). However, since the objective function of Eq.1 is non-convex, proper initialization of parameter is critical to the performance of SGD-based method. In the following, we propose our pre-training strategies to learn proper initial parameters θ_c and θ_r.

Prediction: Based on the output probability, the prediction for a friend is easy, e.g., for user u and i by forwarding on the network and fetch output layer r_{ui} as their probability being friends. Then for different users around u collect their probability and make a ranking list, based on the list the system could recommend potential friends to user u.

3.3 Pre-training Strategies

We design different pre-training strategies for θ_c in the CFM layers and θ_r in the rest layers, repectively, as depicted in Fig. 1(b). Firstly, to obtain θ_c in the convolution layers in Fig. 2, inspired by the idea of autoencoder, we expect the output of pre-training the same as input after specific mathematical operation, i.e., $E = F^T F$, and propose a Poisson probability model to derive the parameter value. Next, the pre-training is performed for parameters θ_r in ranking layers. We view each e_{ui} as one observation sampled from its corresponding Bernoulli probability distribution r_{ui}, and transform the problem into a Bernoulli probability model.

Pre-training for θ_c. Based on the two considerations that, 1)the output F_u of the convolution parts can be regarded as the latent representation for each user u, and 2)the elements of H are non-negative as the output of the sigmoid function, we can regard matrix F as the non-negative latent representation matrix for users with linkage structure E. We use an approach proposed by [19] to derive the latent non-negative matrix F based on E by minimizing the following:

$$\min KL\left(E \parallel F^T F\right) \tag{9}$$

Since the KL-divergence objective is hard to tackle, [3] proposed a probabilistic-based solution by assuming that matrix E obeys a Poisson distribution:

$$E \sim \mathcal{P}\vartheta(F^T F) \tag{10}$$

and [3] also proved that Eq. 9 is equivalent to maximize the likelihood of E, which is defined as:

$$\begin{aligned} L(F) &= \log p(E \mid F) = \log \prod_{u,i} \mathcal{P}\vartheta(e_{ui} \mid F_u^T F_i) \\ &= \sum_u \sum_i \left(e_{ui} \log(F_u^T F_i) - F_u^T F_i - \log \Gamma(e_{ui} + 1) \right) \end{aligned} \tag{11}$$

Substituting H defined by Eqs. 4 and 11 induces to

$$\begin{aligned} L(\theta_c) = \sum_{u,i} \Bigg(& e_{ui} \log \sum_k \left[f(E_u) \right]_k \cdot \left[f(E_i) \right]_k \\ & - \sum_k \left[f(E_u) \right]_k \cdot \left[f(E_i) \right]_k - \log \Gamma(e_{ui} + 1) \Bigg) - \lambda \|\theta_c\|^2 \end{aligned} \tag{12}$$

Here by regarding CFM operations as a non-linear function with input E_u and parameters θ_c, in Eq. 12, $f(E_u)$ represents the CFM's output vector given E_u. Since it's hard to infer θ_c from Eq.12, we propose to optimize Eq.12 by relaxing it to two sub-problems. The first sub-problem regards $f(E_u)$ simply as a variable rather than a function during the optimization, and aims to seek the optimal vectors F_u^* for every user u in Eq.12. Then given $\mathbf{F}^*$ estimated in the first sub-problem, the second sub-problem tries to estimate θ_c so that function $f(E_u)$ given E_u will best approximate F_u^*. These two sub-problems are iteratively solved in a relaxed EM-style. In the E step, the value of $f(E_u)$ is optimized given the parameters θ_c; in the M step, the parameters θ_c are optimized given $\mathbf{F}^*$. Specifically, the EM style algorithm is as following:

E Step: Regarding $F_u = f(E_u)$ as a variable, Eq.12 simplifies to:

$$\max_F \sum_{u,i} \left(e_{ui} \log \sum_k F_{uk} F_{ik} - \sum_k F_{uk} F_{ik} - \log \Gamma(e_{ui} + 1) \right) \tag{13}$$

For which [3] proposed an iterative optimization solution, in each iteration, the elements of $\mathbf{F}$ are updated using :

$$F_{uk}^* = F_{uk} \cdot \frac{\sum_x \frac{e_{ux} F_{xk}}{\sum_y F_{yk}}}{\sum_x F_{xk}}, F_{ik}^* = F_{ik} \cdot \frac{\sum_x \frac{e_{ix} F_{xk}}{\sum_y F_{yk}}}{\sum_x F_{xk}} \tag{14}$$

M Step: Given the estimated $\mathbf{F}^*$ from E step, we update the parameters $\theta_c = \{v, biasF\}$ by solving a least-square problem:

$$\min_{v, biasH} \sum_u \left\| F_u^* - f(E_u) \right\|^2 + \lambda \|\theta_c\|^2 \tag{15}$$

We use SGD to estimate θ_c. In each iteration of SGD, each θ_c is updated as following:

$$\Delta\theta_c = \eta \cdot \left(\left[F_u^* - f(E_u) \right] \cdot \frac{\partial f}{\partial \theta_c} - \lambda \cdot \theta_c \right) \tag{16}$$

where η is the learning rate and λ is the regularizing parameter.

Note that previous work [41] has a similar two-step pre-training design for a different application. However, for the first sub-problem, it directly obtains the value for $\mathbf{F}^*$ with a greedy algorithm. Then the second step approximate the fixed value $\mathbf{F}^*$ by adjusting other parameters. This may work well in neural network with simple

Algorithm 2 Pre-training Framework for BayDNN

Initialize: Random initial $\{\theta_c, \theta_r\}$ with Gaussian
Require: Hyper-parameters like learning rate λ, M
procedure PRE-TRAINING(E)
  repeat ▷ Pre-training CFM with Poisson Model
    Calculate F_u by CFM_FORWARD(E_u) for every u
    for all u, i
      Calculate optimized F^*_{ui} with F_{ui} (Eq. 14)
      Calculate square error $\varepsilon = \|\mathbf{F}^* - \mathbf{f}(\mathbf{E})\|^2$ (Eq. 15)
      Back propagation through CFM with error ε
  until Convergence
  Calculate F_u by CFM_FORWARD(E_u) for every u
  repeat ▷ Pre-training BayDNN with Bernoulli Model
    for all $u, i \in$ batch
      Calculate r_{ui} from F_u and F_i (Eq. 6,7)
      Back propagation through BayDNN
  until Convergence
  for Epoch in AdaDelta ▷ Fine-tuning for Pre-training
    Random select a partial relation u, i, j
    Calculate r_{ui},r_{uj} from E_u, E_i, E_j (Eq. 2,3,4,6,7)
    Update BayDNN with AdaDelta
end procedure

structure, however, in deep convolution neural network the problem may rise that the fixed value F^*_u is not within the range of $f(E_u)$.

Pre-training for θ_r. Using the embedded feature H_u by the CFM part as input, and r_{ui}, r_{uj} as the output, we pre-train the rest structure to estimate parameters $\theta_r = \{w, biasH, \mathbf{b}\}$ for user u. Regarding the relationship between users E_{ui} as the observation sampled from the the Bernoulli probability distribution with parameter r_{ui}:

$$p(e_{ui} \mid r_{ui}) = r_{ui}^{e_{ui}} \cdot (1 - r_{ui})^{1-e_{ui}}, \tag{17}$$

Its corresponding log-likelihood is defined as:

$$L = \sum_{u,i} \Big(e_{ui} \log r_{ui} + (1 - e_{ui}) \log(1 - r_{ui}) \Big) \tag{18}$$

To estimate θ_r, similarly we denote the non-linear function $g(F_u, F_i)$ given embedded feature $\mathbf{F}$ through parameters θ_r, which equals to output layer of our model as described in Eq. 7. To maximize the log-likelihood of Bernoulli distribution, for user u we sample positive users i where $e_{ui} = 1$ from set $\mathcal{P}_u$ and negative users j where $e_{uj} = 0$ from set $\mathcal{N}_u$, where both sets are previously defined in Section 3.1. In addition we just sample j from $\mathcal{N}_u$ instead of $\mathcal{U}_u$ because our purpose of pre-training is to accelerate the training of deep neural network. However if we sample j from $\mathcal{U}_u$ like others traditional methods the time efficiency will be low. Our log-likelihood function is defined as:

$$L(\theta_r) = \sum_u \left(\sum_{i \in \mathcal{P}_u} \log g(F_u, F_i) + \sum_{j \in \mathcal{N}_u} \log \Big[1 - g(F_u, F_j) \Big] \right) - \lambda \|\theta_r\|^2 \tag{19}$$

We use SGD to conduct the optimization. In each iteration of SGD, θ_r are updated as following:

$$\Delta\theta_r = \eta \cdot \Big([e_{ui} - g(F_u, F_i)] \cdot \frac{\partial g}{\partial \theta_r} - \lambda \cdot \theta_r \Big) \tag{20}$$

where η is the learning rate, λ is the regularizing parameter.

AdaDelta for Fine Tuning (Overall Tuning). After the pre-training, we adjust all parameters in order to get the uniformed output, we call it Fine Tuning for the pre-training. Fine tuning is necessary when the pre-training of parameters are separately processed. To give the whole parameters a better initial space we adopt AdaDelta algorithm that scale the learning rate of SGD based on the the history of both gradients and weights [43], which could speed up the convergence of deep neural network at first periods during the training process.

4 EXPERIMENTS

4.1 Experiment Settings

In this section, we compare the proposed BayDNN model with existing friend recommendation methods on two public social network datasets. For all the methods, 10-fold cross validation is performed on each dataset and the average results are reported.

Datasets. We use networks from two application scenarios with different user and relationship density to test the effectiveness and generality of our approach.

Epinions[1]**:** The dataset is extracted from the Epinions website, containing 3640 users and 40752 friend relationship. Each user is an online user of Epinions.

Slashdot[2]**:** The dataset is extracted from the Slashdot Zoo online website. It contains 3099 users and 33216 friend relationship.

Evaluation Metrics. We use AUC and NDCG to evaluate the recommendation performance of the different comparison methods.

AUC: Area Under the relative operating Characteristic (AUC) [15] is originally defined to evaluate the ranking performance for two class problem, which Relative Operating Characteristic (ROC) represents the comparison of two operating characteristics TPR and FPR as the classification threshold criterion changes. The AUC value ranges from 0.5 to 1 and high value indicates good performance. To evaluate the recommendation performance, we use the AUC between $\mathcal{P}$ and $\mathcal{U}$ to measure the ability the model rank $\mathcal{P}$ before the rest.

NDCG: Normalized Discounted Cumulative Gain (NDCG) is the ratio of DCG to the Ideal DCG, where the Ideal DCG for user u means, its positive friends $\mathcal{P}$ are always ranked before the rest. The higher NDCG value indicates better learning performance. Commonly $NDCG@n$ which calculates the NDCG result over the top ranked n items are used in recommendation tasks. In our experiments, we calculate $NDCG@5$ for each user and average them as a metric. Meanwhile we also try different n for detailed estimation.

Precision-Recall: We also plot the precision-recall curves for each comparison method. First we randomly sampled test negative labels from missing links three times the size of positive labels in test set. Then calculate different $Precision@n$ and $Recall@n$ by regarding first n users in the recommending list as positive labels and draw them on the figures to do further analysis.

[1] http://epinions.com/
[2] https://slashdot.org/

Comparison Methods. We compare our BayDNN model with several state-of-the-art methods for friend recommendation, including two popular graph feature based methods, three matrix factorization based methods and one artificial neural network based method.

Katz [14]: It is a shortest path distance based metric for evaluating the similarity between two nodes in graph. Although Katz is a variant of shortest path distance, it generally works better for link prediction because this metric extract the more detailed path based features for the network. Based on these features between two users a proper trained classifier will judge whether there exists a link between them, as described by [29] logistic regression performed better in this situation than other classifiers like SVM. Hence we adopted the logistic regression based on metric Katz between every two users.

Adamic/Adar [1]: This is a metric for quantifying the similarity between two nodes in an web pages network based on common neighbors. Regarding common neighbors as features, [23] used this metric to do link prediction in online social networks. Experiments showed that this algorithm not only performed well on link prediction, but also it could extracted precise topology information of the graph. Here we also adopted the logistic regression as Katz.

Matrix Factorization (MF) [25]: Matrix Factorization is a latent factor model, widely used for rating prediction [16, 35]. Menon and Elkan et al. [25] proposed to use MF for link prediction problem by regarding positive links as 1 and the others as 0, which outperformed most baselines.

Non-negative Matrix Factorization (NMF) [19]: It is a non-negative version of matrix factorization, introduced by [19] with constraining that two factorized matrix are non-negative.

Pairwise Input Neural Network (PINN) [41]: It is a four-layer artificial neural network with simple pre-training strategy for link prediction, which regarded two nodes' link vector as input and output the probability that there exists a link between them.

Bayesian Personalized Ranking Matrix Factorization (BPRMF) [37]: It is a matrix factorization based bayesian probabilistic model with typical ranking strategy that aims at maximizing the probability for all the partial relationship in dataset [34], which has been recognized as an effective ranking method for friend recommendation.

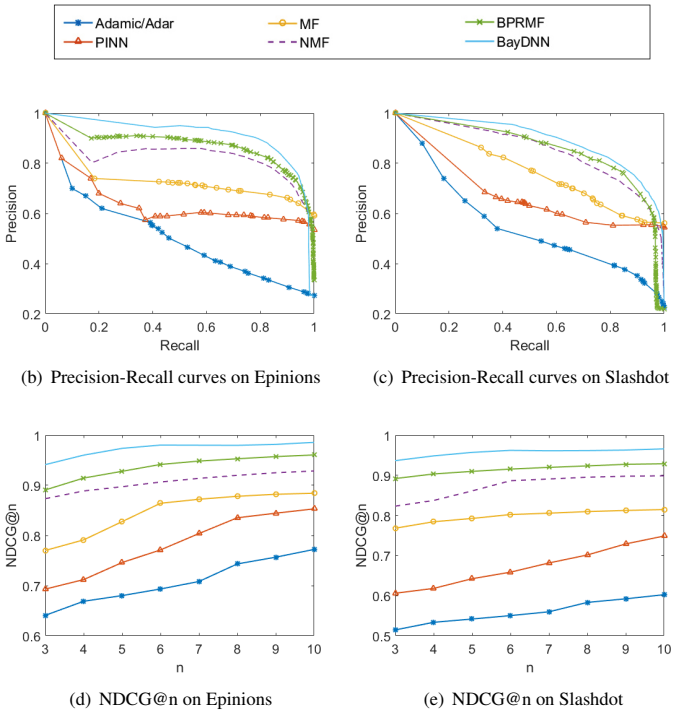
For our BayDNN method, we use Gaussian random variables to intialize the weight parameters in each layer and validate our model with dropout [39]. The batch size for BayDNN varies from 20 to 50 depending on the size of datasets, specifically, for Epinions we choose 40 as batch size and 30 for Slashdot. The experiments are conducted on a machine with one GPU (NVIDIA GTX-1080) and one CPU (INTEL Xeon E5-2620).

4.2 Recommendation Performances

In Table 1, the AUC and NDCG@5 results of the methods on the two datasets are demonstrated. In Fig. 3, the precision-recall curves and NDCG@n (with n rangeing from 3 to 10) results are ploted. From the results, we can see that our BayDNN outperforms greatly over the baselines on all the data sets and evaluation measures. On Epinions and Slashdot, it outperformes the best baseline BPRMFN respectively by 4.6% and 4.8% on NDCG@5, and much higher than the other baseline models.

Table 1: The AUC and NDCG@5 results of comparison methods. For both measures, the larger, the better.

Methods \ Results	Epinions		Slashdot	
	AUC	NDCG@5	AUC	NDCG@5
Katz	0.751	0.554	0.661	0.411
Adamic/Adar	0.858	0.680	0.891	0.542
MF	0.888	0.838	0.943	0.792
NMF	0.934	0.897	0.944	0.861
BPRMF	0.948	0.928	0.946	0.909
PINN	0.843	0.746	0.847	0.642
BayDNN	**0.974**	**0.974**	**0.965**	**0.957**



(b) Precision-Recall curves on Epinions

(c) Precision-Recall curves on Slashdot

(d) NDCG@n on Epinions

(e) NDCG@n on Slashdot

Figure 3: The Precision-Recall curves and NDCG@n results of different methods on the two datasets.

For the baselines, comparing the two graph based methods Katz and Adamic/Adar, Adamic/Adar performs better than Katz as it extracts more precise features by paying attention to features of the neighbors, compared to Katz which only utilizes the features of the path between two users. Whereas comparing the two graph based methods with the other methods which learn feature representations, We see that overall the feature learning methods—MF, NMF, BPRMF, PINN and BayDNN clearly outperform the graph methods. Matrix factorization based algorithms show a big advantage over the graph-based algorithms that only uses the topology features. Taking into account the non-negative property of friendship, NMF outperforms MF consistently over all data sets, which verifies the value

of of considering the specific property of tasks when designing the learning model.

What's worth noting is that the deep neural network model PINN underperforms most algorithms, which confirms our previous intuition that straightforwardly applying DNN to friend recommendation without carefully designed network structure and pre-training strategy is far from sufficient.

Our BayDNN model shows a remarkable advantage over these baseline algorithms by combining the ranking idea of BPR and a finely designed DNN with our pre-training strategy. On Epinions dataset, in terms of NDCG@5, it achieves a 29% improvement over Adamic/Adar, 23% over PINN, 8% over NMF and 5% over BPRMF. With n increasing the gaps become smaller. AUC also stated the same result.

4.3 Discussions

In this section, we discuss the effects of several different design parts in our BayDNN model from the following aspects:1) How does the CFM (Convolutional Feature Mapping) part affect the learning? 2) How does the user bias affect the learning? 3) How does the pre-training affect the learning? 4) We further investigate the effect of negative friendship on learning by changing the way negative friendship are generated. To conduct the above three types of study, we compare the proposed BayDNN with several different variants of BayDNN.

In order to further study the effects of our deep convolutional neural network, we firstly replace the CFM part of BayDNN with a full connection neural network, to construct a method that combines BPR and a common neural network, i.e., **BayNN**, and compare our **BayDNN** method with it. The feature extraction part of BayNN works similarly as PINN.

To study the effect of user bias modeling, we compare **BayDNN** with **BayDNN**$_b$, which shares the same network structure with BayDNN except that, a uniform bias b for all the users is used in **BayDNN**$_b$ rather than a unique b_u for each user for in **BayDNN**.

To study the effect of pre-training on learning, we compare **BayDNN** with two other models **BayDNN (raw)** and **BayDNN (SAE)** with different pre-training strategies. While **BayDNN (raw)** does not use any pre-training strategies, **BayDNN (SAE)** uses the stacked auto-encoder pre-training strategy proposed by [7]. It applies autoencoding on each layer of the neural network.

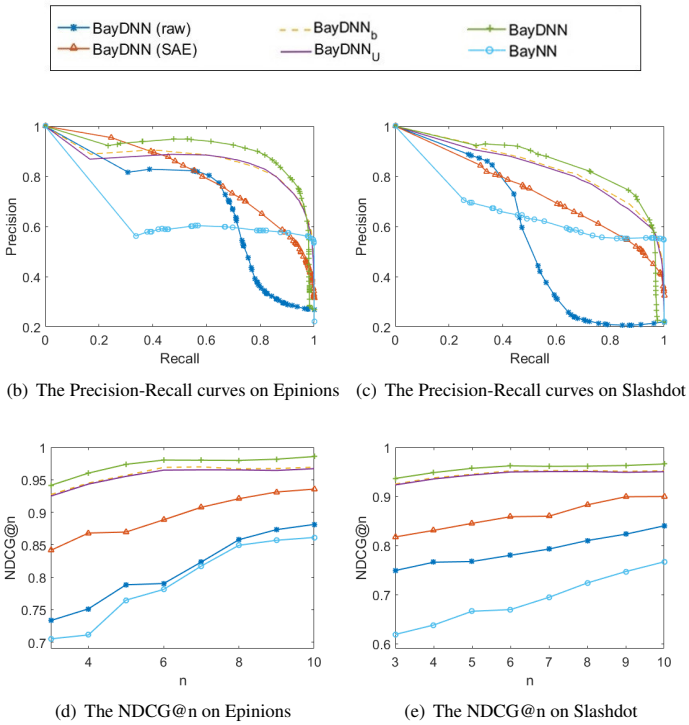
In BayDNN, the negative friendship are recognized as those user pairs who can't reach each other within 6 hops. To study the effect of negative friendship on learning, we first compare BayDNN with **BayDNN**$_{\mathcal{U}}$ which uses no negative friendship during learning. We also change the way the negative friendship defined by varying the number of hops in $\{0, 4, 6, 8\}$ to select negative friends.

The overall comparison results of the above 4 variants and BayDNN are shown in Table 2 and Fig. 4. From Table 2 and Fig. 4, our proposed BayDNN is optimal in all the three aspects. In the following, we discuss in more details from the respective aspects.

How does the deep structure affect the learning? We firstly study the effect of deep structure on BayDNN comparing to PINN with the same BPR learning strategy. Though with the help of BPR BayNN improves a bit than PINN (1.5% in AUC), it is still much worse than BayDNN (12% in AUC). We infer that the CFM part influence a lot because the our convolution neural network extract better latent feature on link information rather than shallow architecture under the same learning strategy.

Table 2: The AUC and NDCG@5 results of self comparison of BayDNN.

Methods \ Results	Epinions		Slashdot	
	AUC	NDCG@5	AUC	NDCG@5
BayNN	0.855	0.765	0.852	0.658
BayDNN$_b$	0.964	0.957	0.959	0.945
BayDNN (raw)	0.821	0.788	0.814	0.767
BayDNN (SAE)	0.912	0.869	0.897	0.845
BayDNN$_{\mathcal{U}}$	0.963	0.957	0.956	0.946
BayDNN	**0.974**	**0.974**	**0.965**	**0.957**



(b) The Precision-Recall curves on Epinions (c) The Precision-Recall curves on Slashdot

(d) The NDCG@n on Epinions (e) The NDCG@n on Slashdot

Figure 4: The Precision-Recall curves and NDCG@n curves for different self-comparison models on the two datasets.

How does the user bias affect the learning? We then look into the effect of b$_u$, i.e., the personalized bias in Eq.7, by comparing BayDNN with BayDNN$_b$. Results in Table 2 show that our BayDNN with individual bias b_u for each user outperforms BayDNN$_b$ with one single bias for all users 1% on average. Here we study the effects of the bias parameters by investigating the output value of the output layer of BayDNN and BayDNN$_b$. Fig. 5 depicts how the user bias influence the prediction value r_{ui} that will later be used for ranking

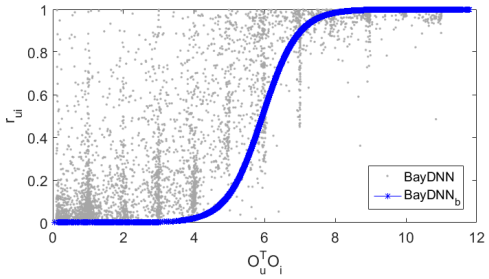


Figure 5: r_{ui} distribution with a uniform bias, i.e., $r_{ui} = \sigma(O_u^T O_i + b)$, against r_{ui} distribution with individual bias for each user, i.e., $r_{ui} = \sigma(O_u^T O_i + b_u + b_i)$, on Epinions dataset.

in the model on the Epinions dataset. From the figure we can see that the setting of personalized bias makes the values more distributed rather than within the range of a single sigmoid function, which brings in more flexibility and better performance.

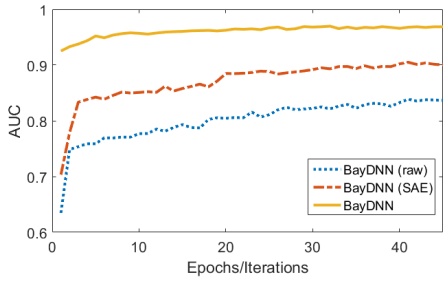
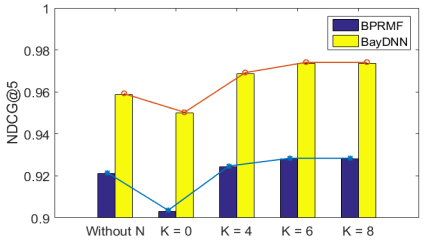


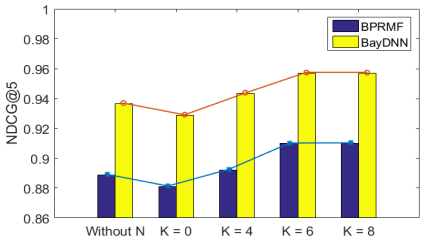
Figure 6: The iterative AUC results during SGD optimization for BayDNN with different pre-training strategies on the Epinions dataset.

How does pre-training affect learning? We further validate our pre-training strategy by comparing the results with BayDNN (raw) and BayDNN (SAE). Table 2 and Fig. 4 show that without any pre-training strategy, the performance of BayDNN (raw) drops significantly. With a stacked auto-encoding pre-training BayDNN (SAE) improves largely over BayDNN (raw). In terms of NDCG@5, our strategy achieves a 15% improvement over BayDNN (raw) and 6% over BayDNN (SAE).

Moreover, we study the effects of pre-training strategies by investigating their step-wise performance during the SGD optimization. Fig. 6 depicts the iterative AUC results of BayDNN, BayDNN (raw) and BayDNN (SAE) during the SGD optimization process on the Epinions dataset. We can see that the proper pre-training strategy of BayDNN gives the model a very advantaged initialized space and the model converges very quickly in about 15 epochs, whereas BayDNN (raw) and BayDNN (SAE) start from a very low point and take a long process to converge, i.e., about 200 epochs. To this respect we can say that with our pre-training design, the time cost of the model is largely reduced by 92.5%.



(a) Performances on Epinions by varying k



(b) Performances on Slashdot by varying k

Figure 7: Results changed by k in sampling for $\mathcal{N}$.

How does the negative friendship affect learning? The result comparing BayDNN with BayDNN$_{\mathcal{U}}$ (i.e., no $\mathcal{N}$) is given in Table 2 and Fig. 4 again, which depicts that it brings in 1% improvement in terms of NDCG. Here we further investigate how the introduction of negative friendship $\mathcal{N}$ works and how it changes with different k options, i.e., how many unreachable hops should it be in $\mathcal{N}$ selection.

Moreover, let k be the unreachable hop number we set, Fig. 7 depicts the precision results in the two datasets, respectively, with $k = 0$ (randomly select), $k = 4, 6, 8$ (unreachable by 4, 6, 8 hops). The results are consistent with the theory proposed by [21], i.e., as the number increases $k = 6$ reaches the best value of performance. The AUC and NDCG performance value with $k = 6$ are also given in Table 2. An interesting finding is that random selection ($k = 0$) is worse than no selection, which indicates that random selection brings in noise data (examples that do not actually belong to $\mathcal{N}$) that is big enough to harm the performance.

Moreover, we see how it impact on the best performed baseline method, BPRMF. The results show that by introducing $\mathcal{N}$ it works better than random selection and no $\mathcal{N}$ selection, too. The figures show that BayDNN outperforms BPRNN in both situation (with or without $\mathcal{N}$).

5 CONCLUSIONS

In this paper, we study the problem of friend recommendation and propose a novel BayDNN model by combining Bayesian Personalized Ranking and Deep Neural Networks. In BayDNN , we use an one-dimensional convolutional neural network (CNN) to extract latent deep structural feature representations from the input network data, and then use a Bayesian personalized ranking learning to captures users' preference based on the extracted deep features.

BayDNN shows clearly superior performance to several state-of-the-art methods for friend recommendations on two different public datasets. To avoid poor parameter estimation, we also present a fine-tuned pre-training strategy for BayDNN based on Poisson and Bernoulli probabilistic models, respectively for different layers.

As the general idea of using deep neural networks for friend recommendation represents an interesting research direction, there are many potential future directions of this work. First, it is interesting to study incrementally learning the deep neural networks so that we can involve online user feedback into the learning process. Second, another potential is to infer the fined category (e.g., family, friend, and colleague) of social relationships [40]. Last, it would be interesting to connect the study with social theories to further understand how the topological structure of online social networks is formed.

ACKNOWLEDGMENTS

The authors would also like to thank the anonymous referees for their valuable comments and helpful suggestions. The work is supported by the National Natural Science Foundation of China under Grant No.: 61333014.

REFERENCES

[1] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the Web. *Social Networks* 25, 3 (2003), 211–230.

[2] Yoshua Bengio, Nicolas Le Roux, Olivier Delalleau, Patrice Marcotte, and Pascal Vincent. 2005. Convex Neural Networks. *Advances in Neural Information Processing Systems* (2005), 123–130.

[3] Ali Taylan Cemgil. 2009. Bayesian Inference for Nonnegative Matrix Factorisation Models. *Intell. Neuroscience* 2009, Article 4 (Jan. 2009), 17 pages. DOI: http://dx.doi.org/10.1155/2009/785152

[4] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Computer Science* (2014).

[5] Ronan Collobert, Jason Weston, L Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural Language Processing (Almost) from Scratch. *Journal of Machine Learning Research* 12, 1 (2011), 2493–2537.

[6] Malcolm Gladwell. 2000. The Tipping Point - How Little Things Make a Big Difference. *Little, Brown and Company* (2000), 177–181,185–186.

[7] Stefan Glüge, Ronald Böck, and Andreas Wendemuth. 2013. Auto-encoder pre-training of segmented-memory recurrent neural networks.. In *ESANN*.

[8] Neil Zhenqiang Gong, Ameet Talwalkar, Lester Mackey, Ling Huang, Eui Chul Richard Shin, Emil Stefanov, Elaine, Shi, and Dawn Song. 2012. Jointly Predicting Links and Inferring Attributes using a Social-Attribute Network (SAN). *Computer Science* (2012).

[9] Mark Granovetter. 1973. The strength of weak ties. *Amer. J. Sociology* 78, 6 (1973), 1360–1380.

[10] Siyao Han and Yan Xu. 2014. Friend recommendation of microblog in classification framework: Using multiple social behavior features. In *International Conference on Behavior, Economic and Social Computing*. 1–6.

[11] John Hannon, Mike Bennett, and Barry Smyth. 2010. Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches. In *Proceedings of the Fourth ACM Conference on Recommender Systems*. 199–206.

[12] Chaobo He, Hanchao Li, Xiang Fei, and Yong Tang. 2015. A Topic Community-Based Method for Friend Recommendation in Online Social Networks via Joint Nonnegative Matrix Factorization. In *Third International Conference on Advanced Cloud and Big Data*. 28–35.

[13] L. A. Jeni, J. F. Cohn, and La Torre F De. 2012. Facing Imbalanced Data Recommendations for the Use of Performance Metrics.. In *Humaine Association Conference on Affective Computing and Intelligent Interaction*. 245–251.

[14] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

[15] C. David Page Kendrick Boyd, Kevin H. Eng. 2013. *Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals*. 451–466.

[16] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 426–434.

[17] Alex Krizhevsky and etc. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems* 25, 2 (2012), 2012.

[18] Artus Krohn-Grimberghe, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2012. Multi-relational matrix factorization using bayesian personalized ranking for social network data. In *International Conference on Web Search and Web Data Mining*. 173–182.

[19] Daniel D. Lee and H. Sebastian Seung. 2001. Algorithms for Non-negative Matrix Factorization. In *Advances in Neural Information Processing Systems*. 556–562.

[20] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2008. Statistical properties of community structure in large social and information networks. In *WWW'08*. 695–704.

[21] Wei Li and Sara Mcmains. 1963. Behavioral Study of obedience. *Journal of Abnormal Psychology* 67, 4 (1963), 371–378.

[22] Zhepeng Li, Xiao Fang, and Olivia R. Liu Sheng. 2015. A Survey of Link Recommendation for Social Networks: Methods, Theoretical Foundations, and Future Research Directions. *Computer Science* (2015).

[23] David Liben-Nowell and Jon Kleinberg. 2007. The link prediction problem for social networks. *Journal of the Association for Information Science and Technology* 58, 7 (2007), 1019C1031.

[24] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3431–3440.

[25] Aditya Krishna Menon and Charles Elkan. 2011. Link Prediction via Matrix Factorization. In *European Conference on Machine Learning and Knowledge Discovery in Databases*. 437–452.

[26] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. 2007. Measurement and analysis of online social networks. In *ACM SIGCOMM Conference on Internet Measurement 2007*. 1815–1816.

[27] Mark EJ Newman. 2006. Modularity and community structure in networks. *PNAS* 103, 23 (2006), 8577–8582.

[28] Lene Nielsen. 2012. *Personas-user focused design*. Vol. 15. Springer Science & Business Media.

[29] Joshua O'Madadhain, Jon Hutchins, and Padhraic Smyth. 2005. Prediction and ranking algorithms for event-based network data. *Acm Sigkdd Explorations Newsletter* 7, 2 (2005), 23–30.

[30] Shuang Qiu and etc. 2014. Item Group Based Pairwise Preference Learning for Personalized Ranking. In *Proceedings of the 37th International ACM SIGIR Conference on Research; Development in Information Retrieval*. 1219–1222.

[31] Matthew J. Rattigan and David Jensen. 2005. The case for anomalous link discovery. *Acm Sigkdd Explorations Newsletter* 7, 2 (2005), 41–47.

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[33] Yafeng Ren, Yue Zhang, Meishan Zhang, and Donghong Ji. 2016. Context-Sensitive Twitter Sentiment Classification Using Neural Network. AAAI.

[34] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Conference on Uncertainty in Artificial Intelligence*. 452–461.

[35] By Ruslan Salakhutdinov and Andriy Mnih. 2007. Probabilistic Matrix Factorization. *Advances in Neural Information Processing Systems* (2007), 1257–1264.

[36] Aliaksei Severyn and Alessandro Moschitti. 2015. Learning to Rank Short Text Pairs with Convolutional Deep Neural Networks. In *The International ACM SIGIR Conference*. 373–382.

[37] Dongjin Song, David A. Meyer, and Dacheng Tao. 2015. Efficient Latent Link Recommendation in Signed Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1105–1114.

[38] Suraj Srinivas and R. Venkatesh Babu. 2015. Deep Learning in Neural Networks: An Overview. *Computer Science* (2015).

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[40] Jie Tang, Tiancheng Lou, Jon Kleinberg, and Sen Wu. 2016. Transfer Learning to Infer Social Ties Across Heterogeneous Networks. *ACM Trans. Inf. Syst.* 34, 2 (2016), 7:1–7:43.

[41] Caihua Wang, Juan Liu, Fei Luo, Yafang Tan, Zixin Deng, and Qian Nan Hu. 2014. Pairwise input neural network for target-ligand interaction prediction. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*. 67–70.

[42] Nicolas Le Roux Yoshua Bengio, Olivier Delalleau. 2006. *The Curse of Highly Variable Functions for Local Kernel Machines*. 107–114.

[43] Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *Computer Science* (2012).

[44] Yu Zheng, Lizhu Zhang, Zhengxin Ma, Xing Xie, and Wei-Ying Ma. 2011. Recommending Friends and Locations Based on Individual Location History. *ACM Trans. Web* (2011), 5:1–5:44.