Learning Item Embedding with Heterogeneous Information for Collaborative Filtering

Xiusi Chen #1, Xiaoyu Li #2, Chang Zhou *3, Xiaofei Liu *4, Jun Gao #5

School of EECS, Peking University
Science Building #1, 5 Yiheyuan Rd, Beijing, China
[1] xiusichen@pku.edu.cn
[2] lxy1995@pku.edu.cn
[5] gaojun@pku.edu.cn

* Alibaba Group
8 West Wenyi Rd, Hangzhou, Zhejiang, China
[3] ericzhou.zc@alibaba-inc.com
[4] hsiaofei.hfl@alibaba-inc.com

***Abstract*—Item-based collaborative filtering captures the item-item similarities via user-item interactions. However, in online platforms with tens of millions items, it is hard to precisely model the similarities since user behavior is very sparse w.r.t large number of items. Items associated with little user behavior are usually not sufficiently modeled. In fact, the similarities between items could be affected by many facts such as their properties like title and description. In this paper, we propose a representation learning based approach that utilizes the heterogeneous information of users, items, and user-item interactions, to bridge the user behavior and item content. The produced representation models the item similarity from multi-aspects, alleviating the problems above. We evaluate on several tasks, deploy our framework onto a real-world recommender system, and run A/B tests on one of the online recommendation services in Alibaba Group, proving our method to be effective and scalable.**

I. Introduction

In recent years, recommendation systems have obtained crucial importance under the rapid growth of massive information on the internet. Collaborative Filtering[1] (CF) which is based on interactions between users and items, is widely adopted in online services like e-commerce and online music service, etc. The insight is that users sharing similar historic behavior should be alike in their preference. Conventional CF often works by computing the similarities between users or between items, and then conducts the recommendation based on similarities between their representations. Item embeddings, also known as distributed item representations, typically represent items with dense, low-dimensional and real-valued vectors. The embedding captures the similarity between items well via user behavior when built from small datasets where user behavior is usually dense. However, when it comes to industrial level datasets, user behavior would be very sparse w.r.t the large number of items. The conventional CF approaches which only consider user behavior would not be robust enough. Say, the embedding of two items could be totally different even if they indicate the same product, because one of them is lately on sale by a side seller and therefore it is associated with sparse user behavior. As a result, we would like to integrate richer information to capture relationships other than user behavior.

Content-based recommendation is also widely adopted, since it is reasonable that items with similar content should be similar to each other. Content-based approaches[16] solve the cold-start issue, whereas if several items are similar in content, this kind of methods are no longer able to capture the subtle differences. Also, merely recommending through content suffers from low recall, because it cannot well generalize the recommendation results through user-item interactions. It is straightforward to model the unique role of each time more precisely by combining user behavior with the content.

For the real-world industrial applications, massive information generated and uploaded every day can be exploited and incorporated into the item embedding to enrich the semantics of the embedding and further improve the item-based collaborative filtering. Unfortunately, unlike the common cases of open source data, the information in the online recommender systems is usually collected from multiple sources, so these information is probable to be heterogeneous. In order to fully and smoothly incorporate these heterogeneous information, we have to put effort on preprocessing, and try to map these multi source data into a unified vector space.

Towards these goals, we utilize the multi-source data in addition to user's behavior on items to help improve the quality of the embedding for similarity measurement, to enhance the item-based collaborative filtering. We accomplish this goal by learning a distributed representation via deep neural networks. Our produced embedding maps items into a low-dimensional vector space. In recent years, representation learning[2] is growing into a prospective issue, since the low-dimensional embedding can be further used as the input of a variety of successor machine learning applications such as text classification, speech recognition, and transfer learning, etc. Here, we produce the item embedding with heterogeneous information to overcome the common problem of data sparsity, improving the similarity modeling.

In this work, we leverage deep neural network techniques to construct semantically rich representations of items. We have conducted extensive experiments to show that the proposed method improves the performance of inner-shop personalized

item recommendation at *Alibaba Group*[1].

In summary:

- We use deep neural network to incorporate multi source information to produce item embedding.
- We embed the heterogeneous features into latent space and train our model simultaneously, turning our solution into an end-to-end one to improve the performance.
- We evaluate our method both offline and online to verify the superiority of our method.

II. Related Work

Matrix factorization has been widely studied in both academia[3] and industry[3] to address the recommendation tasks where explicit scores on items are available. However, the matrix itself costs large amount of memory. The computation is costly as well. Another problem of the MF-based methods is interpretability—The latent factors obtained by factorizing a user-item matrix is hard to interpret[4].

Several algorithm frameworks based on generative model have been proposed in recent years for dealing with recommendation tasks. Heckel et al. presents a generative model based on the assumption of overlapping co-clustering between users and items for giving more interpretable recommendations than traditional CF[5]. SPORE recommends personalized location-based services in a social network by modeling both user interest and sequential influence of locations in a probabilistic topic model[6]. Li et al. proposes to map both static user interest and dynamic social news into one latent topic space for getting context-aware user representations, which are utilized to get better performance when recommending in a high-speed social news feed[12].

Deep learning techniques have been dominating a variety of domains, e.g. computer vision[7], speech recognition[8] and natural language processing[9] due to its natural ability to combine the features and take in non-linear factors. However, there is still space of development for recommending with deep learning. Neural networks are used for recommending news[10] and citations[11]. News recommendation, characterized by dynamic and real-time requirements, raises higher demand for the efficiency and flexibility of the recommending algorithm. A novel embedding-based recommendation method has been put forward recently in which a recurrent neural network is adopted to generate user representations and the news are ranked by simple inner-product operations[13]. Deep neural network formulates collaborative filtering in [14]. Deep learning is used for cross domain user modeling in [15]. Burges et al. used deep neural networks to recommend music in a content-based setting[16]. The recurrent neural network is used to model the sequential information and user behavior in [17] and provide customized recommendation based on real-time user patterns in [18]. Apart from RNN which achieves reputations in natural language processing, the generative models such as Markov chains has also been proved effective in recommendation tasks[19]. Nevertheless, neither the RNN model or Markov chain is likely to be deployed in the production environment due to the high latency brought by model complexity. The user and the Point of Interests (POI) are encoded to preserve the similarities between both user-POI and POI-POI in [20].

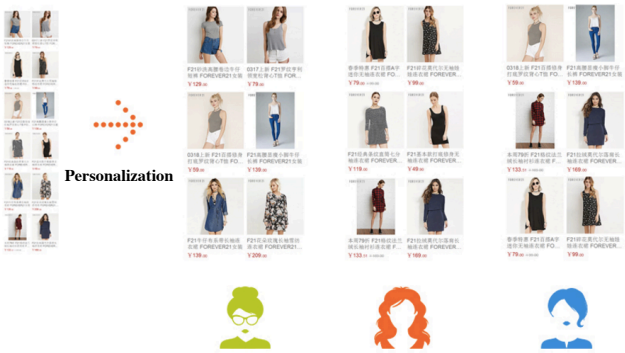

Fig. 1. Inner-shop personalized recommendation

The most related paper to our work is [21], which designs a deep learning approach to recommending videos on YouTube[2] website. Nevertheless, it aims to generalize matrix factorization, and the inference phase goes through multiple table reading for constructing model input and a complete calculation of the neural network, leading to a dragging online latency. In our case, we expect all computation for conducting the personal recommendation to finish in a strictly low latency. Despite the similarity in the model architecture, their idea of generalized matrix factorization differs from representation learning and item collaborative filtering, on which our approach mainly focuses. Moreover, item recommendation at Alibaba may make use of some additional information, including the explicit hierarchical categories of items and prices, etc. These factors may be exploited to produce better recommendation in our cases.

III. Application Scenario

In this section, we introduce in detail the specific scenario, i.e., the inner-shop personalized item recommendation service. We will conduct our recommendation algorithm in this specific scene. Our solution is initially developed to improve the online click through rate (CTR) in this scene, but it can be easily generalized to other recommendation tasks.

There are millions of shops resided on our inner-shop personalized recommendation service. As shown in Figure 1, shop owners will offer several item sets (S) on their homepage, each of which contains tens of candidate items according to their own marketing strategies. Our task is to expose the personalized top-6 items within S to the customers when they visit a specific shop. The common case in our service is that, many items in a same shop tend to share similar property features such as title and description, so it would be hard to sufficiently model the subtle differences between these items if we merely consider their properties. On the other hand, most of the user behavior concentrates on a small number of shops. Items in these shops ("hot" items) are usually affiliated with rich user behavior, and these

[1]www.taobao.com

[2]www.youtube.com

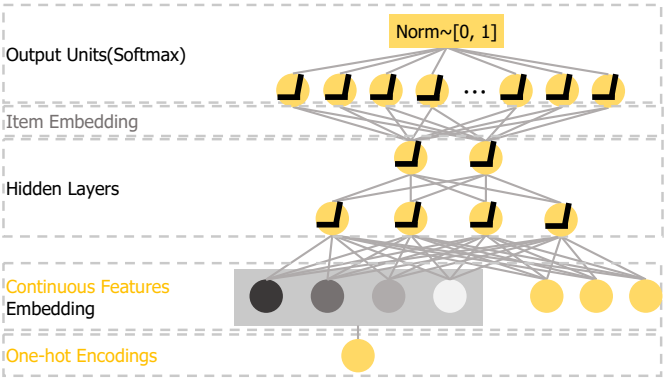

Fig. 2. Model Architecture

items can be easily associated to other hot items via user behavior. However, a large percent of the shops is very sparse w.r.t. user behavior, so it would be problematic if we only leverage the user behavior. A better solution is to combine the user related information such as profile, historic behavior, and item's attributes like textual title and descriptions, and prices to enhance the conventional collaborative filtering.

IV. Produce Item Embedding

This section introduces in detail our neural network architecture to produce the item embedding and how to incorporate heterogeneous information. Our produced item embedding is then streamed to item-based collaborative filtering by item similarity measurement.

A. Model Architecture

Our approach incorporates the heterogeneous information by feeding all useful features into the neural network during the training phase. Combining and feeding those information into the neural network as the input seems to be a straightforward solution and turns out to be an effective one.

As shown in Figure 2, our model architecture follows a feedforward network pattern in general. We specially treat the category features with embedding layers. The model outputs the probability for each item to be the next watched item, which is normalized by a softmax function.

The bottom layers are where we feed the features. Throughout our exploration, we found that the most significant features are the information related to users and items, and the ones that indicate user-item correlations. All of the features can be divided into continuous features and discrete ones (which is usually called categorical features) by their values. Neural networks are good at dealing with normalized continuous features. One way to address the categorical features is to embed them in advance to build a look-up table before training the deep model, then read the table to build each training example for the model's input. In contrast, we add embedding layers to map the one-hot encodings of these features to the dense representations, and train the model at the same time. That is, the parameters of embedding layers are learned jointly with all successor parameters in the fully connected layers. All the embedded categorical features are then concatenated with normalized continuous features at a merge layer. We will describe in detail how we address specific data in the next subsection.

Fully connected layers come subsequently, which combine these features and incorporate non-linear factors. Suppose the i-th hidden layer has output $\vec{x_i}$. The output signal of the $(i+1)$-th hidden layer forms:

$$\vec{x_{i+1}} = f(W_i\vec{x_i} + \vec{b_i})$$

where W_i is the weight matrix between the two layers, and $\vec{b_i}$ denotes the bias. The activation function f is here to bring in the non-linear factors, for which ReLU or tanh is typically used.

At the output layer, each neuron corresponds to a candidate item. Neuron i outputs a value that measures the probability of item i to be watched next. The softmax layer intends to normalize the measurement. The probability for user U to watch item i given the context C forms as:

$$P(watch = i|U, C) = \frac{e^{\vec{v_i}\vec{x}}}{\sum_{j\in I} e^{\vec{v_j}\vec{x}}} \tag{1}$$

where $\vec{x}$ denotes the output of the last hidden layer (which can be viewed as user embedding) and I denotes item ID set. The vector $\vec{v_j}$ above denotes the weights on the connections between neurons at the last hidden layer and the j-th neuron of the output layer, and that is the embedding of item j.

We maximize the probability of the actually next watched item i_a given the recently watched items. Nevertheless, it is hard to directly optimize the probability formation of the function, since it is costly to put effort on computing the summation of inner products between all the candidate items and the current user's embedding. In practice, we take the next watched item as a positive sample and sample several items as negative samples. Thus, the conditional probability in equation 1 can be written down as:

$$log\sigma(\vec{v_i} \cdot \vec{x}) + k \cdot E_{v_j \sim P_D}[log\sigma(-\vec{v_j} \cdot \vec{x})] \tag{2}$$

meaning that we sample k negative samples subject to the item occurrence distribution P_D in the user click log. Specifically, we sample a negative item according to the uniform distribution, $P_D(i) \sim \frac{1}{|I|}$, where $|I|$ is the total number of items. Then the comprehensive loss function can be then written as:

$$\begin{aligned}\ell = \sum_i \sum_j & \#sampled_(i, j) \cdot (log\sigma(\vec{v_i} \cdot \vec{x}) \\ & + k \cdot E_{v_j \sim P_D}[log\sigma(-\vec{v_j} \cdot \vec{x})])\end{aligned} \tag{3}$$

where $\sigma(x) = 1/(1 + exp(-x))$ is the sigmoid function. $\#sampled_(i, j)$ denotes the count of samples that item i and item j are sampled as positive pairs.

B. Training Samples

Here we illustrate how we derive our training samples from the original user click log. As illustrated in Fig. 3, an arbitrary user's click sequence forms a single directed linked list. When

we generate the positive samples, our policy is similar to the case of *word2vec*[9], but we make restrictions on our policy so that it is more adaptive to the recommendation case. Specifically, our training samples (x, y) means that our model should predict item y given x. Intuitively, we know the preference of a user only when we gather his/her past behavior. Therefore, it is reasonable that item x is an upstream node of item y. We assume that more recently viewed items have more impact on the prediction of the next watched items, so we sample item pairs with the probability weighted with decaying factors:

$$p(x|y) = \frac{1}{2^h}$$

where h denotes the hops between x and y.

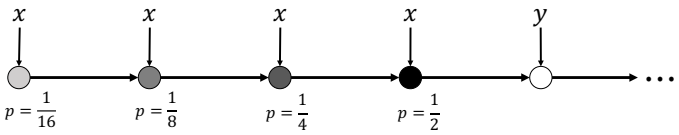

Fig. 3. Generating samples from user click sequence

The extreme multi-class classification severely hurts the training efficiency in that the probabilities of every items are computed before the most likely next watched item is produced. In other words, fully calculating the softmax may dominate the entire feedforward procedure. Negative sampling[22] is then adopted to accelerate the overall speed of training. Generally speaking, the next watched item relative to the current one being watched is labeled positive example, and we sample 5 to 10 items which are not the following watched items, labeling them as negative samples. The negative sampling brings around 6 times of acceleration, reducing the training time from 24 hours per epoch to 4 hours.

C. Feature Processing

As discussed above, our model takes item related information as input, and the heterogeneous features available on the online item recommendation service include but not limit to: user behavior, demographic features, geographic information, device models, item's text information, category information, and price, etc.

The text information is fused by the word embedding of the item's title and descriptions. For controlling the parameter size in the entire model, we filter and select the top-K most frequent words, then all items are represented by these words by one-hot encodings before embedded and concatenated into the feature layer.

Just like Figure 4, the category information at Alibaba forms a hierarchical category tree with the max depth 5. Every single item is classified into one leaf node in the tree. The path from root to item i can be denoted as $(c_1, c_2, c_3, c_4, c_5)$. Here we input the category information in five-hot encoding, where positions of c_1, c_2, c_3, c_4 and c_5 are set to 1. The five-hot encoding vectors are then fed into an embedding layer.

Other continuous features such as demographic features and price are normalized to [0, 1] before directly concatenated into the input layer, while categorical ones are encoded in one-hot vectors. The embeddings of these features are trained jointly with other parameters affiliated with those full connected layers.

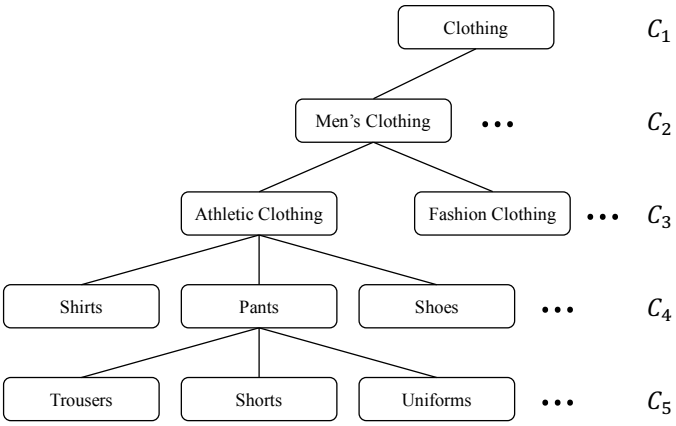

Fig. 4. Hierarchical Category

In summary, we take heterogeneous features into account including continuous and category ones via normalization and embedding, respectively. The training of our item embedding takes in many factors including information related to users and items, and the pair-wise training example itself is a reflection of user's behavior on items, namely user-item interactions. The incorporated information aims to assist the item embedding in modeling the more comprehensive relations between items, improving the result of item-based collaborative filtering.

V. Experiments

We conduct both offline and online experiments to evaluate the performance of our methods. The offline tasks include click prediction and ranking, while the online experiment is conducted on the industrial recommendation service.

A. Datasets

We collect original logs of user click records to make our training examples. The details of the logs are as follows. For consecutive 7 days in May, 2017, there are 282,403,881 click records. There are 14,333,506 unique users in the log, who viewed 22,067,643 different items. The log is split into sessions and fed to the deep neural network model.

B. Competitors

DeepWalk [23]. DeepWalk samples multiple paths from the graph, each of which is regarded as a word sequence. For each vertex in the sequence, it predicts the nearby vertices in both directions, and updates the vector according to the Skip-Gram model. It cannot capture asymmetric relationships in a graph, which restricts its applications.

LINE [24]. LINE introduces the 2-nd order proximity between a pair of vertices, which encodes the similarity measured by their local neighborhood. It samples individual edges in the graph, and updates the corresponding vector according to SGD in the Skip-Gram model. The node pairs from two more

hop away will be regarded as negative labels, which means it cannot capture the higher-order similarities.

APP [25]. It is a graph-based embedding approach, which preserves asymmetric similarities between vertex pairs. Under the recommendation setting, it constructs a graph from user's click sequence, then samples paths and builds positive samples by selecting head and tail nodes of these paths. The negative samples are randomly sampled. We notice that its performance is reported to be the state-of-the-art among several well-known graph embedding methods including DeepWalk[23], LINE[24] and Node2Vec[26]. User behavior in this method is well exploited, yet information other than that is not used.

Textual word2vec [22]. It is reasonable that people would be attracted by items sharing similar title and description, so we also extract each item's title and description. We use the textual lexicon as the vocabulary and run *word2vec* to produce their word embedding. This textual embedding is to capture the text characteristics of the items.

C. Training

Our model is trained on a server equipped with two Xeon 2.5GHz 16-core CPUs, 128GB memory and one Nvidia Tesla M40 GPU (12GB GDDR5). The model is built with TensorFlow, which is configured to GPU mode at runtime. Under our experiment setup, there are 800 neurons at the input layer, with each layer halving the number of neurons of its former layer. The last hidden layer has 200 neurons, indicating our derived embedding has a dimension of 200. All activation functions in our model are ReLU. Our optimization objective is the standard cross-entropy, and Stochastic Gradient Descent (SGD) is adopted to train our model. We set the vocabulary size K in the textual embedding 100,000. During the training process, 70% sessions are used for training, 20% are used for validation, and the rest are used for testing. Normally, it takes 18 to 24 hours before a model completely converges, corresponding to 4 to 6 iterations. We train APP with the collaborative structural training data, which is an item graph constructed by the item click sequences organized by user sessions, within the last 7 days. Note that, the graph contains both inter and inner shop item connections.

D. Click Prediction

In recommendation scenarios, accurately predicting the user's behavior is a standard for evaluating the recommender's performance. The click prediction task is here to show the superiority of our method over the competitors. We adopt $precision@K$ as the evaluation metrics for node recommendation methods, where

$$precision@K = \frac{|PredSet \bigcap TestSet|}{|PredSet|}$$

Table I shows the result of click prediction. The basic features include user's previous watched item's embedding and textual embedding, demographic features (age, gender), geographic information such as location and device models.

We can see that our item embedding shows better performance than both APP and word2vec. Meanwhile, we try different combinations of category and price to see their influences on the results. It turns out that the embedding with category information show better results on the precision in click prediction task.

TABLE I
PRECISION@N FOR CLICK PREDICTION

Embedding Type	P@1	P@4
word2vec	0.000057	0.000114
DeepWalk	0.019387	0.050062
LINE	0.016326	0.045056
APP	0.024897	0.054234
basic features+cate+price	0.046110	**0.076506**
basic features+cate	**0.046576**	0.076210
basic features+price	0.045171	0.076147
basic features+device	0.046462	0.075935

E. Ranking

As is alike to search engines, the ranking of recommending results has impact on user experiences. We expect the produced embedding by our methods to produce a decent ranking impression. We adopt the AUC (Area Under the Curve) measurement to further evaluate the overall scoring quality.

TABLE II
AREA UNDER CURVE (AUC) SCORES FOR RANKING

Embedding Type	AUC
word2vec	0.495024
DeepWalk	0.876122
LINE	0.776706
APP	0.877373
basic features+cate+price	0.937838
basic features+cate	0.938206
basic features+price	**0.939401**
basic features+device	0.938096

As illustrated in Table II, we notice our conducted embedding generally performs better than APP on precision, both of which outperform the textual word embedding. We hardly observe any improvement brought by pure textual embedding in terms of AUC compared to the naive method which selects recommended items randomly. We attribute this to the exceeding overlaps of titles and descriptions between massive items in the same shop, so that simply offering items with the most similar textual information won't receive a decent ranking performance on AUC. In contrast, we should rely more on user-item interactions to measure the similarities between items.

F. Online Deployment

We deploy our approach on the personalized online recommendation services in *Alibaba Group*.

The online recommender follows the conventional item-based collaborative filtering, taking our produced embedding and APP as different similarity measurements. We score each item in the candidate set according to the average proximity between items in footprint item set and the candidate set. Based on the offline training data, we use APP to produce embedded vectors for each vertex. We set the dimensions of

each vertex to be 200. The last hidden layer has 200 neurons so that each item has embedding of exactly the same dimension as in APP. We compare our method with APP since the latter is reported to have better performance than competitors for node recommendation and link prediction tasks[25]. The A/B

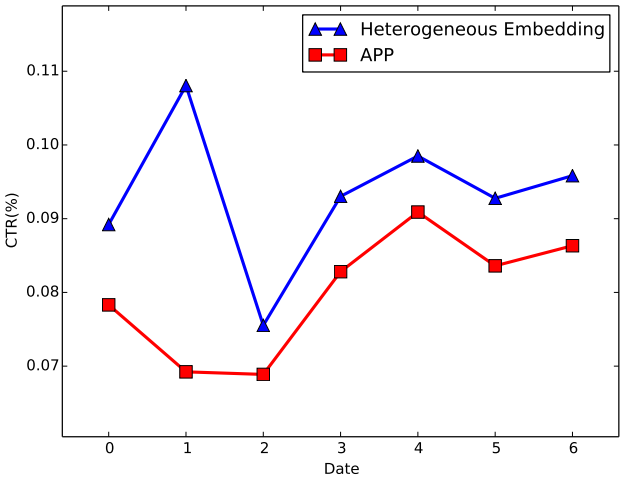

Fig. 5. CTR for Item Recommendation

test is conducted for both methods under the same traffic flow, and the Click Through Rate(CTR) for a continuous 7 days is reported in Figure 5. The relative gap between the two methods is quite stable, and our method is significantly better than the graph-based item embedding approach for this service, due to the benefit of the fusion of heterogeneous information.

VI. Conclusion

In this paper, we propose an approach to learn item embedding, which uses deep neural network to incorporate heterogeneous information, to improve the overall click through rate. Experimental results have shown the superiority of the proposed approach, which outperforms the existing graph embedding methods and textual word embedding in a number of evaluation tasks. What's more, the method is deployed on the inner-shop personalized item recommendation service in Alibaba, and exhibits a great performance boost.

References

[1] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[2] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[4] M. Rossetti, F. Stella, and M. Zanker, "Towards explaining latent factors with topic models in collaborative recommender systems," in *Database and Expert Systems Applications (DEXA), 2013 24th International Workshop on*. IEEE, 2013, pp. 162–167.

[5] R. Heckel, M. Vlachos, T. Parnell, and C. Duenner, "Scalable and interpretable product recommendations via overlapping co-clustering," in *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 2017, pp. 1033–1044.

[6] W. Wang, H. Yin, S. Sadiq, L. Chen, M. Xie, and X. Zhou, "Spore: A sequential personalized spatial item recommender system," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 954–965.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[8] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[9] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[10] K.-J. Oh, W.-J. Lee, C.-G. Lim, and H.-J. Choi, "Personalized news recommendation using classified keywords to capture user preference," in *Advanced Communication Technology (ICACT), 2014 16th International Conference on*. IEEE, 2014, pp. 1283–1287.

[11] W. Huang, Z. Wu, L. Chen, P. Mitra, and C. L. Giles, "A neural probabilistic model for context based citation recommendation." in *AAAI*, 2015, pp. 2404–2410.

[12] Y. Li, D. Zhang, Z. Lan, and K.-L. Tan, "Context-aware advertisement recommendation for high-speed social news feeding," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 505–516.

[13] S. Okura, Y. Tagami, S. Ono, and A. Tajima, "Embedding-based news recommendation for millions of users," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1933–1942.

[14] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1235–1244.

[15] A. M. Elkahky, Y. Song, and X. He, "A multi-view deep learning approach for cross domain user modeling in recommendation systems," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 278–288.

[16] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *Advances in neural information processing systems*, 2013, pp. 2643–2651.

[17] Y. Zhang, H. Dai, C. Xu, J. Feng, T. Wang, J. Bian, B. Wang, and T.-Y. Liu, "Sequential click prediction for sponsored search with recurrent neural networks." in *AAAI*, 2014, pp. 1369–1375.

[18] S. Wu, W. Ren, C. Yu, G. Chen, D. Zhang, and J. Zhu, "Personal recommendation using deep recurrent neural networks in netease," in *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*. IEEE, 2016, pp. 1218–1229.

[19] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme, "Factorizing personalized markov chains for next-basket recommendation," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 811–820.

[20] S. Feng, X. Li, Y. Zeng, G. Cong, Y. M. Chee, and Q. Yuan, "Personalized ranking metric embedding for next new poi recommendation." in *IJCAI*, 2015, pp. 2069–2075.

[21] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016, pp. 191–198.

[22] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[23] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[24] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 1067–1077.

[25] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity." in *AAAI*, 2017, pp. 2942–2948.

[26] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 855–864.