Dynamic Item Block and Prediction Enhancing Block for Sequential Recommendation

Guibing Guo[1], **Shichang Ouyang**[1,*], **Xiaodong He**[2], **Fajie Yuan**[3] and **Xiaohua Liu**[2]

[1]Northeastern University, China
[2]JD AI Research, Beijing, China
[3]Tencent, Shenzhen, China

guogb@swc.neu.edu.cn, 1701282@stu.neu.edu.cn, {xiaodong.he, roy.liu}@jd.com, fajieyuan@tencent.com

Abstract

Sequential recommendation systems have become a research hotpot recently to suggest users with the next item of interest (to interact with). However, existing approaches suffer from two limitations: (1) The representation of an item is relatively static and fixed for all users. We argue that even a same item should be represented distinctively with respect to different users and time steps. (2) The generation of a prediction for a user over an item is computed in a single scale (e.g., by their inner product), ignoring the nature of multi-scale user preferences. To resolve these issues, in this paper we propose two enhancing building blocks for sequential recommendation. Specifically, we devise a Dynamic Item Block (DIB) to learn dynamic item representation by aggregating the embeddings of those who rated the same item before that time step. Then, we come up with a Prediction Enhancing Block (PEB) to project user representation into multiple scales, based on which many predictions can be made and attentively aggregated for enhanced learning. Each prediction is generated by a softmax over a sampled itemset rather than the whole item space for efficiency. We conduct a series of experiments on four real datasets, and show that even a basic model can be greatly enhanced with the involvement of DIB and PEB in terms of ranking accuracy. The code and datasets can be obtained from https://github.com/ouououououou/DIB-PEB-Sequential-RS

1 Introduction

Sequential systems have been a core technique in many real-world applications. They are used to infer the next item or action that the users may be interested in from their sequential behaviors. For example, one may purchase a GPU fan after buying a GPU; and people may continue to buy books by their favorite authors or purchase clothes from the online store where they had a good experience. To make accurate next-item prediction, it is important to learn multi-scale user preferences, such as long-term and short-term preferences.

Many approaches have been proposed to resolve the task of next-item prediction, and they can be broadly classified into three types. The first type aims to model user preference drift by temporal matrix factorization based on heuristic assumptions on user behavior patterns [Koren, 2009]. The second type focuses on the short-term dependency with the previous items a user just interacted with, and typical approaches are based on Markov chain [He and McAuley, 2016a; Rendle *et al.*, 2010]. Recently, the models based on neural networks quickly emerge as the third type to model user preference. They can capture both short- and long-term dependency with the interacted items through recurrent neural network (RNN) [Hidasi *et al.*, 2015], convolutional neural network (CNN) [Tang and Wang, 2018; Yuan *et al.*, 2019] and their memory variants [Chen *et al.*, 2018]. Our work follows the third direction to design an end-to-end neural model for sequential recommendation.

However, the existing neural sequential models suffer from two limitations, which may severely degrade the performance of sequential recommendation. Firstly, the representation of an item is relatively static and fixed for all users. Researchers tend to presume that an item will not change over time and users. We argue that even the same item may be viewed distinctively for different users at specific time steps. Secondly, the generation of a prediction for a user towards an item is computed in a single scale, e.g., by an inner product of their embedding. Existing works often take a uniform representation for a user, and thus limit the prediction in a single scale. We contend that generation should be accomplished at multiple scales to be consistent with the nature of user preference.

To resolve these issues, in this paper we propose two building blocks to enhance the performance of neural sequential models. Firstly, we devise a Dynamic Item Block (DIB) to learn dynamic item representation by aggregating the embeddings of those (similar users) who rated or interacted with the same item before that time step. The importance of different similar users is determined by an attention mechanism. Secondly, we come up with a Prediction Enhancing Block (PEB) to project user representation into multiple scales, based on which many predictions can be made accordingly and then attentively aggregated together for enhanced learning. Each

*Corresponding author

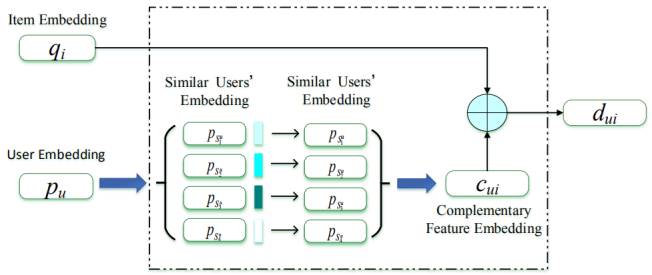

Figure 1: Structure of Dynamic Item Block (DIB)

prediction is calculated by a softmax function over a sampled itemset rather than the whole item space for efficiency's purpose. We construct two end-to-end neural sequential models by involving both DIB and PEB blocks. To evaluate the effectiveness of our approaches, we conduct extensive experiments on four real-world datasets, and the results show that our approach beats other counterparts in terms of ranking accuracy and even a baseline can be greatly improved with the help of both DIB and PEB blocks.

2 Our Sequential Building Blocks

In this section, we will elaborate our proposed two building blocks for sequential recommendations, namely dynamic item block and prediction enhancing block. To facilitate discussion, we introduce a number of notations. Suppose I is the set of all items and let p_u and q_i be the embedding of user u and item i. The items that user has rated before can be arranged into a list, defined as I_u^+. The target (positive) item denoted as i_p^u and the set of negative items i_n^u sampled from $I_u^- = I/I_u^+$ for training the i_p^u is denoted as $N_{i_p^u}$.

2.1 Dynamic Item Block (DIB)

The first sequential building block we propose is Dynamic Item Block (DIB), which aims to learn dynamic item representation associated with those users who also rated or interacted with the item in question before a specific time step. The general structure of DIB is illustrated in Figure 1. To be specific, DIB can be viewed as a transformer that takes the embeddings p_u, q_i of both user u and item i as input and then transforms the item embedding q_i to a new user-related embedding d_{ui}. The whole process is finished in two steps.

Step 1 :

The first step is to search for users who rated the same item, i.e., like-minded users sharing similar interest. This step is specifically useful for the cold-start users who have only rated few items, and thus depends more on similar users to learn accurate user representation. The process to select similar users is given in Figure 2. Suppose the current input of DIB is user u_3 and item i. We can search for a number of similar users having interaction with item i and ordered by interaction time steps. We find the exact position (i.e., time step) of user u_3 in the user list, and select the users interacting before this time step, i.e., users u_4, u_5, u_6, u_7 in this case. To speed up processing, we set a window size $K = 3$ to only preserve the

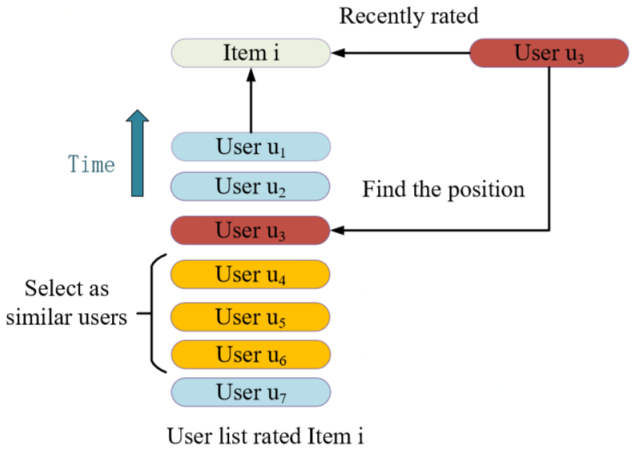

Figure 2: The process of selecting similar users within DIB structure

latest K users (relative to the u_3's position), that is, user u_7 will be removed from the selection.

Step 2 :

The second step is to aggregate the embeddings of similar users as the complementary feature embedding c_{ui} to item embedding q_i, as shown in Figure 1. Suppose we have chosen K similar users for user u and the embedding of similar user s_i^u denoted as $p_{s_i^u}$. We use an attention-like structure to capture the dynamic association between users. Formally, the attention weight is calculated as follows:

$$z_{s_i^u} = \frac{\exp(p_u^\top p_{s_i^u})}{\sum_{k=1}^{K} \exp(p_u^\top p_{s_k^u})} \tag{1}$$

Then, we merge these users' feature embedding to get the complementary feature embedding c_{ui} as follows:

$$c_{ui} = \sum_{k=1}^{K} z_{s_k^u} \cdot p_{s_k^u} \tag{2}$$

Finally, the item embedding and complementary feature embedding are merged together to get the final user-related embedding for item i, defined by:

$$d_{ui} = \text{merge}(q_i, c_{ui}) \tag{3}$$

where merge$(\cdot)$ is a function that combines two vectors into one. The particular choice of merge$(\cdot)$ in our model is a simple weighted vector addition, that is:

$$\text{merge}(x, y) = x + \alpha y \tag{4}$$

where α is a weighting parameter to indicate the importance of variable y, i.e., the complementary feature embedding c_{ui} in our case.

2.2 Prediction Enhancing Block (PEB)

The second sequential building block we design is the Prediction Enhancing Block (PEB), which aims to project user representation into multiple scales such that a number of (item score) predictions can be done in the meanwhile. The structure of PEB can be found in Figure 3 (right-side block). The whole process of PEB can be also completed in two steps.

Step 1 :
In the first step, a user representation p_u will be projected into K embeddings through a fully connected layer, where the k-th projected embedding is denoted as $\hat{p}_u^k$. For each projected user embedding, we can make a prediction $\hat{y}_{ui}^k$ over items to determine the probability of each item i to be selected as the next recommendation. The probability is often calculated by a softmax function over all the item space. However, due to the large volume of items, the calculation will be very expensive and time-consuming.

To resolve this issue, we opt to adopt a negative sampling strategy to randomly select a subset of items (yet not interacted) for fast softmax computation. The idea here has similar motivations with [Yuan *et al.*, 2016] and [Guo *et al.*, 2018]. We denote the interacted item as i_u^p and sampled (negative) itemset as $N_{i_p^u}$, and thus the prediction is given by the following softmax:

$$\hat{y}_{ui}^k = \frac{\exp(\hat{p}_u^{k\top} q_i)}{\sum_{j \in \{i\} \cup N_{i_p^u}} \exp(\hat{p}_u^{k\top} q_j)} \tag{5}$$

Note that the item space we operate on is reduced from all the items I to a much smaller space, i.e., the union combination of current interacted item i_p^u and negative itemset $N_{i_p^u}$.

Step 2 :
The second step is to attentively aggregate all the projected predictions into a final value. Although K predictions can be obtained in the last step, their importance (or reliability) to final item prediction is not the same. Thus, we project p_u into K reliability coefficient through another layer, where the k-th coefficient is denoted as β_k (the importance of the k-th prediction).

Specifically, the straightforward goal of a recommendation loss function is to maximize the predicted probability of positive items $\hat{y}_{ui_p^u}$, and minimize that of negative items $\hat{y}_{ui_n^u}$. That is, if $\hat{y}_{ui_p^u}$ is high and $\hat{y}_{ui_n^u}$ is low, the prediction is reliable (i.e., greater importance β); otherwise the prediction is unreliable. In other words, $\hat{y}_{ui_p^u}$ and β are positively correlated, while $\hat{y}_{ui_n^u}$ and β are negatively associated. Let $a_{ui} = |\hat{y}_{ui} - \beta|$ be the difference between the predicted probability and reliability coefficient. In this paper, we impose another optimization goal for importance learning: to maximize $a_{ui_n^u}$ and minimize $a_{ui_p^u}$. We use a constant weight denoted as η (usually set to 0.5) to balance these two goals. The final loss function is defined as follows:

$$\begin{aligned} L_u \;=\; & \frac{1}{K} \sum_{k=1}^{K} -\log\left(\eta \cdot y_{ui_p^u}^k + (1-\eta)\cdot(1 - a_{ui_p^u}^k)\right) \\ & + \sum_{i_n \in N_{i_p^u}} -\log\left(\eta \cdot (1 - y_{ui_n^u}^k) + (1-\eta)\cdot a_{ui_n^u}^k\right) \end{aligned} \tag{6}$$

After the training phase, when test the performance of our proposed models (will be elaborated in next section), as shown in Figure 3, we will fuse all the predictions $\hat{y}_{ui}^k$ to obtain the final predicted probability $\hat{y}_{ui}^*$. The weight of each $\hat{y}_{ui}^k$ is calculated by normalizing their prediction reliability:

$$w_k = \frac{\exp(\beta_k)}{\sum_{j=1}^{K} \exp(\beta_j)} \tag{7}$$

Thus, the predicted probability $\hat{y}_{ui}^*$ that user u is likely to interact with item i in the next time step is given by:

$$\hat{y}_{ui}^* = \sum_{k=1}^{K} w_k \cdot \hat{y}_{ui}^k \tag{8}$$

2.3 Models with Sequential Building Blocks

We proceed to present two sequential recommendations by applying our proposed sequential building blocks, i.e., DIB and PEB. Two sequence models (GRU) and memory network (MN) are used as examples in this paper. For ease of discussion, we denote these two models as GRU-DIB-PEB and MN-DIB-PEB respectively.

GRU-DIB-PEB
As shown in Figure 3, the GRU-DIB-PEB model will first transform the embedding of items in the training sequence through DIB structure. After that, GRU will compress all the d_{ui} into a fixed vector, denoted as h_u. Lastly, we get the final user embedding by concatenating the h_u and user feature embedding p_u:

$$\hat{p}_u = \text{concat}(h_u, p_u) \tag{9}$$

Previous models then will directly use $\hat{p}_u$ to compute $\hat{y}_{ui}$ (represent the probability of item i to be the next item that user u will interact with) by:

$$\hat{y}_{ui} = \text{predict}(\hat{p}_u, q_i) \tag{10}$$

where predict$(\cdot)$ could be an arbitrary function. The sigmoid inner product $\hat{y}_{ui} = \sigma(\hat{p}_u^\top q_i)$ is a popular implementation. However, in GRU-DIB-PEB model, we will input $\hat{p}_u$ to the two fully connected layers in PEB and obtain K new user feature embedding $\hat{p}_u^k$ and K reliability parameter β^k. After that, during the training phase, Eq. 6 will be adopted as the loss function. When making prediction, we will use Eq. 8 as the implementation of predict$(\cdot)$.

MN-DIB-PEB
In the case of memory network, each item will be embedded to vector q_i^c for memory output other than the feature embedding q_i. Suppose we are given an item set $\{i_1, i_2 \cdots i_k\}$ to be stored in the memory of user u. Each item in the set is converted into a memory vector, denoted as m_i (corresponding to d_{ui} in GRU-DIB-PEB). Besides, each item also has a corresponding output vector $m_i^c = \text{DIB}(q_i^c, p_u)$. Then, we compute the matching score between p_u and each memory m_i by taking the inner product, followed by a softmax function:

$$z_{ui} = \text{softmax}(p_u^\top m_i) \tag{11}$$

where z_{ui} is a weight vector over the memory. The user memory vector p_u^m is then a summation over the memory m_i^c, weighted by the weight vector z_{ui}, defined by:

$$p_u^m = \sum_{k=1}^{K} z_{uk} \cdot m_i^c \tag{12}$$

After that, we get the final user embedding by concatenating the p_u^m and user feature embedding p_u:

$$\hat{p}_u = \text{concat}(p_u^m, p_u) \tag{13}$$

Note that the final prediction part is the same with the GRU-DIB-PEB model, and thus we omit it for space saving.

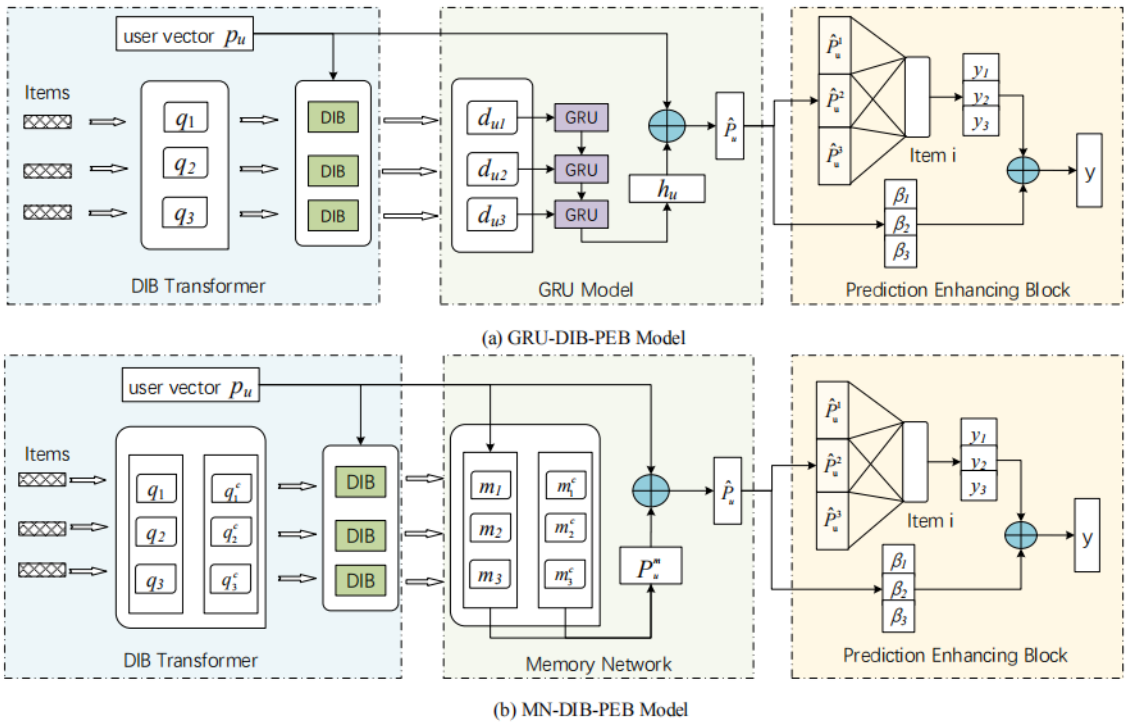

Figure 3: Our proposed GRU-DIB-PEB and MN-DIB-PEB models

3 Experiments

3.1 Experiment Setup

Data sets

We conduct our experiments on four real-word datasets, including three Amazon datasets[1] [He and McAuley, 2016b; McAuley *et al.*, 2015] and MovieLens-100K[2]. The Amazon datasets contains user-product purchasing behaviors from Amazon spanning from May 1996 to July 2014. We evaluate our models on three product categories, including Movies and TV, CDs and Vinyl and Kindle Store. To provide sequential recommendations, we select those users with at least 20 purchasing records for experiments.

Evaluation metrics

To evaluate the performance of recommendation, each of our data sets is split into training/validation/testing sets. For each user, we preserve the last two interactions to validation and testing sets, while the rest interactions are used for training. To reduce the expensive cost of ranking all items for each user during evaluation, we use a strategy similar to [He *et al.*, 2017; Zhang *et al.*, 2018], that is, we randomly sample 999 items that have not been interacted by the user, and then to rank the target item among the 1000 items. We adopt Recall@N and NDCG@N [Yining Wang, 2013] to evaluate the models. Generally, higher metric values indicate better ranking accuracy.

[1]http://jmcauley.ucsd.edu/data/amazon/

[2]https://grouplens.org/datasets/movielens/100k/

Baselines

We compare with the following sequential recommendation models to justify the value of our approaches.

- **FPMC** [Rendle *et al.*, 2010]: Factorized personalized Markov chains, a model combined matrix factorization and Markov chain.
- **GRU4rec** [Hidasi *et al.*, 2015]: a session-based recommendation model uses GRU to capture sequential dependencies and make predictions.
- **RUMI** [Chen *et al.*, 2018]: Item-level RUM adopts the memory mechanism to manipulate the users' historical records in a more explicit and effective manner.
- **Caser** [Tang and Wang, 2018]: Caser utilizes CNN to model the union-level sequential patterns by capturing local sequential patterns of adjacent items.

Note that all the baselines use BPR as the loss function, which provides the optimal performance. Besides, in order to analyze the effect of PEB, we applied four other loss functions (below listed) to both MN-DIB and GRU-DIB models. They are also used as the complementary baselines.

- **NCE**: Noise-Contrastive Estimation (NCE) is a pointwise loss which was first introduced by Gutmann and Hyvarinen [Gutmann and Hyvärinen, 2012].
- **BPR**: Bayesian Personalized Ranking [Rendle *et al.*, 2009] is a pairwise ranking loss, which compares the score of a positive and a negative items.
- **TOP1**: A ranking loss first devised by [Hidasi *et al.*, 2015]. It will approximately estimate the relative rank of the relevant item in a list.

Dataset	Metrics	FPMC	RUMI	GRU4Rec	Caser	GRU-DIB-PEB	MN-DIB-PEB	Improve
Movielens-100K	Recall@10	0.2045	0.3397	**0.3546**	0.3504	**0.4388**	0.3887	+23.75%
	NDCG@10	0.1007	0.1723	**0.1868**	0.1833	**0.2476**	0.2054	+32.55%
CDs and Viny1	Recall@10	**0.4244**	0.3968	0.3853	0.3389	0.5040	**0.5098**	+20.12%
	NDCG@10	0.2553	**0.2571**	0.2535	0.2253	**0.3501**	0.3463	+36.17%
Kindle Store	Recall@10	0.4238	0.4556	**0.4656**	0.4136	**0.6066**	0.6064	+30.28%
	NDCG@10	0.2306	0.2901	**0.3046**	0.2896	**0.4285**	0.4184	+40.48%
Movies and TV	Recall@10	0.3236	0.3635	**0.3869**	0.3623	**0.4895**	0.4631	+26.52%
	NDCG@10	0.1686	0.2167	**0.2448**	0.2297	**0.3244**	0.2962	+32.52%

Table 1: The performance comparison of baselines and our models, where the best results of baselines and our models are in bold-faced.

Dataset	Metrics	MN-DIB-NCE	MN-DIB-BPR	MN-DIB-TOP1	MN-DIB-CE	MN-DIB-PEB	Improve 1	Improve 2
Movielens-100K	Recall@10	0.3738	**0.3834**	0.2609	**0.3908**	**0.3887**	+1.38%	-0.54%%
	NDCG@10	0.1954	**0.2010**	0.1326	**0.2121**	**0.2054**	+2.19%	-3.16%
CDs and Viny1	Recall@10	0.4490	**0.4881**	0.4204	**0.4856**	**0.5098**	+4.45%	+4.98%
	NDCG@10	0.2807	**0.3187**	0.2688	**0.3341**	**0.3463**	+8.66%	+3.65%
Kindle Store	Recall@10	0.5357	**0.5760**	0.3931	**0.5849**	**0.6064**	+5.28%	+3.68%
	NDCG@10	0.3396	**0.3763**	0.2296	**0.4147**	**0.4184**	+11.19%	+0.89%
Movies and TV	Recall@10	0.4257	**0.4429**	0.3962	**0.4400**	**0.4631**	+4.56%	+5.25%
	NDCG@10	0.2612	**0.2765**	0.2385	**0.2874**	**0.2962**	+7.12%	+3.06%

Table 2: The performance comparison of MN-DIB model with different loss functions. Improve 1: compare PEB with the best sample-based loss. Improve 2: compare PEB with the cross entropy

- **CE**: Cross entropy represents the difference between two probability distribution and it often used as the loss function in machine learning and optimization.

Parameter settings

We implement our proposed methods and all baselines using TensorFlow and Adam optimizer [Kingma and Ba, 2014]. For each method, the grid search is applied to find the optimal settings of hyperparameters using the validation set. These include embedding dimensions d from $\{16, 32, 50, 100, 150\}$ and the learning rate from $\{0.001, 0.002, 0.005, 0.1, 0.2, 1\}$. For RUMI, Caser, MN-DIB, GRU-DIB and GRU4Rec, the sequence length L is from $\{3, 5, 10, 15, 20\}$. For MN-DIB and GRU-DIB, the window size of latest similar users is chosen from $\{3, 5, 10, 15\}$. To compare each loss function fairly, the sampling number of BPR, TOP1, NCE and PEB is set to 25. We report the results of each method under its optimal hyperparameter settings.

3.2 Results and Analysis

Overall comparison

Table 1 presents the performance comparison among a number of baselines and our models, where the percentage of improvements our models achieve relative to the best of others is also given. The experimental results show that significant improvements (performance improved in 20% to 40%) can be reached by our models in comparison with others, by integrating the two building blocks in sequential models. Specifically, DIB helps construct better item representation with similar users as contextual information, while PEB enhances the prediction in a multi-scale manner for higher recommendation reliability.

Impact of PEB

Table 2 and Figure 4 show the performance comparison when replacing PEB with traditional predictions. It can be observed that taking more items into account rather than calculating the probability independently can help the model obtain better performance. The pair-wise loss BPR consistently outperforms the point-wise loss NCE. The performance of cross entropy is always better than the sample-based loss, especially in the ranking accuracy metrics such as NDCG, because it takes the whole items set into account when making probability estimation. Although TOP1 is also a pair-wise function, but it improperly uses $sigmoid(\cdot)$ to calculate the final loss instead of $log(sigmoid(\cdot))$, which limits the updating range and validity of the parameters in the model and dramatically degrades the model performance. PEB enables the model to judge the reliability of its own prediction. Then, by effectively fusing multiple predictions as the final result, the performance of PEB is significantly better than the sample-based loss functions even the cross entropy. More importantly, compared with cross entropy, the computational cost of PEB decreases a lot(from $O(N)$(N=290,298) to $O(K * n)$(n=25) in CDs and Viny1 dataset).

Impact of DIB

We can compare the performance of RUMI with MN-DIB-BPR and GRU4Rec with GRU-DIB-BPR(in Figure 4) to analyze the impact of DIB, because the network structure used in each pair are similar and we also use BPR to train the GRU4Rec and RUMI model. The experimental results show that huge performance improvements can be obtained by using DIB and the gap will increase as the sparsity of datasets. The reason is that DIB could convert the item embedding q_i to

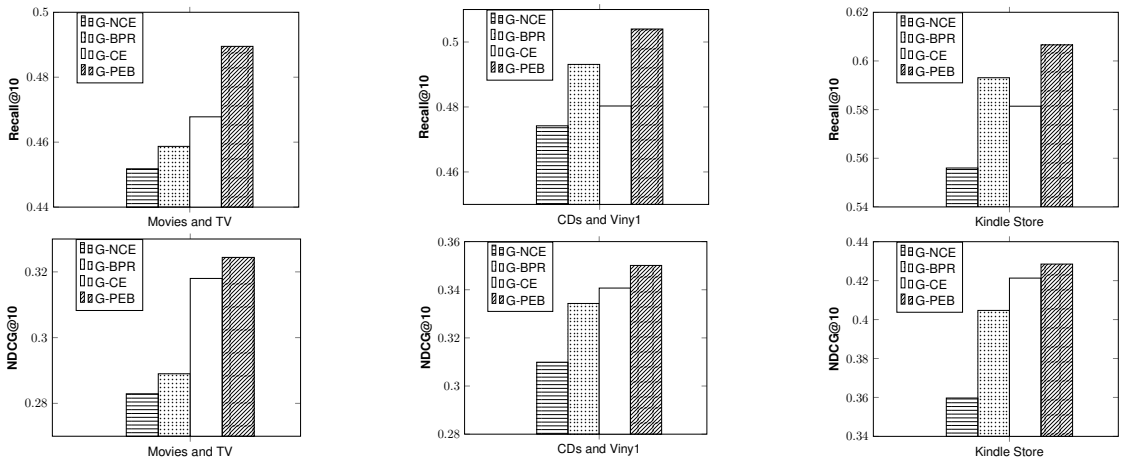

Figure 4: The best performance of GRU-DIB model with different loss functions in three real-world data sets. G-NCE represents the GRU-DIB model with NCE loss function and other abbreviations follow the same rules

user-related item embedding d_{ui} which is well-suited to the personalized recommendation. Otherwise, the features extracted from the dynamic similar users help complement the user profiles for the "cold-start" users.

Optimal value of K used in PEB

Choosing an appropriate value of K in PEB is important. Through experiments, we found that the optimal value of K increases as the items in the data set. The reason is that the loss functions based on sampling only consider a small subset rather than the whole items when calculate the probability of each item. Therefore, the more items in the data set, the probability estimated by them is more inaccurate. To solve this, PEB teaches the model how to judge the reliability of its predictions. A larger K means the model can make predictions more times, which improves the opportunity of obtaining a more reliable probability estimation. For the small data sets, such as ml-100k, the K is generally set to 3$\sim$5 to get the best performance. When the number of items increases to nearly 100,000 (Movies and TV), setting K to 8$\sim$10 is much better. If the items further increase, K can be set to 15 or higher.

4 Related Work

Nowadays, many approaches have been proposed to model the sequential patterns of users. [Zimdars *et al.*, 2001] first described a model based on Markov chain to model the sequential patterns of users. The main idea of Markov chain models is to predict the user's next action by estimating the probability of a item-to-item transition matrix. However, these models often suffered from the data sparsity problem. In order to solve this, Factorized personalized Markov chains (FPMC) [Rendle *et al.*, 2010] subsumed the matrix factorization(MC) with Markov chains to generate more training data for each user. Different from the FPMC, [Koren, 2009] only used the MC to make sequential recommendation. The transition matrix generated by it will change over time to suit the drifting users preference. After that, lots of neural network models were proposed, such as [Hidasi *et al.*, 2015] using recurrent neural network (RNN) to compress the user history records into a fixed vector and then use it to make prediction. [Chen *et al.*, 2018] adopted memory network to store and manipulate the users' previous behaviors in a more explicit and effective way. By using the external user memory matrix instead of RNN, the performance got consistently improvements. In addition, to model not only the point-level but also the union-level sequential patterns, [Tang and Wang, 2018] used the convolutional neural network(CNN) to embed the items rated by a user into a "image" and learn the sequential patterns from the local features of the image. More recently, [Yuan *et al.*, 2019] proposed NextItNet that applied 1D dilated CNN and residual blocks to model both short- and long- sessions. However, all these models use a static embedding to represent each item, but they ignore a fact that different users may be interested in different aspects of a item.

5 Conclusion

In this paper, we proposed two generic building blocks to construct end-to-end neural models for sequential recommendations. Specifically, Dynamic Item Block (DIB) was proposed to resolve the issue that traditional approaches treated item representation as relatively static and fixed, taking no consideration of similar users. Besides, Prediction Enhancing Block (PEB) was devised to enhance the model prediction by projecting user representation into a multi-scale space, whereby many predictions can be made and then aggregated to improve the reliability of predictions. We applies these two building blocks into GRU and memory network, respectively. The experimental results on four real-world datasets demonstrated that our approaches gained significant improvements relative to other state-of-the-art methods.

Acknowledgments

This work is supported by the JD Grapevine Plan and Fundamental Research Funds for the Central Universities under Grant No. N181705007, by the National Natural Science Foundation of China under Grants No. 61772125, No. 61702084 and No. 61702090.

References

[Chen *et al.*, 2018] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 108–116, 2018.

[Guo *et al.*, 2018] Guibing Guo, Shichang Ouyang, Fajie Yuan, and Xingwei Wang. Approximating word ranking and negative sampling for word embedding. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2018.

[Gutmann and Hyvärinen, 2012] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13(Feb):307–361, 2012.

[He and McAuley, 2016a] Ruining He and Julian McAuley. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 191–200, 2016.

[He and McAuley, 2016b] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*, pages 507–517, 2016.

[He *et al.*, 2017] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pages 173–182, 2017.

[Hidasi *et al.*, 2015] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

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

[Koren, 2009] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[McAuley *et al.*, 2015] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52, 2015.

[Rendle *et al.*, 2009] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461, 2009.

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

[Tang and Wang, 2018] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 565–573, 2018.

[Yining Wang, 2013] Yuanzhi Li Di He Wei Chen Tie-Yan Liu Yining Wang, Liwei Wang. A theoretical analysis of normalized discounted cumulative gain (ndcg) ranking measures. In *Proceedings of the 26th Annual Conference on Learning Theory*, 2013.

[Yuan *et al.*, 2016] Fajie Yuan, Guibing Guo, Joemon M Jose, Long Chen, Haitao Yu, and Weinan Zhang. Lambdafm: learning optimal ranking with factorization machines using lambda surrogates. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 227–236. ACM, 2016.

[Yuan *et al.*, 2019] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 582–590. ACM, 2019.

[Zhang *et al.*, 2018] Quangui Zhang, Longbing Cao, Chengzhang Zhu, Zhiqiang Li, and Jinguang Sun. Coupledcf: Learning explicit and implicit user-item couplings in recommendation for deep collaborative filtering. In *IJCAI*, pages 3662–3668, 2018.

[Zimdars *et al.*, 2001] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. Using temporal data for making recommendations. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 580–588, 2001.