Learning Recommender Systems from Multi-Behavior Data

Chen Gao*, Xiangnan He†, Dahua Gan*, Xiangning Chen*, Fuli Feng†, Yong Li*, Tat-Seng Chua†, Depeng jin*
*Beijing National Research Center for Information Science and Technology
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
†School of Computing, National University of Singapore, Computing 1, Computing Drive, Singapore 117417

***Abstract*—Most existing recommender systems leverage the data of one type of user behaviors only, such as the purchase behavior in E-commerce that is directly related to the business KPI (Key Performance Indicator) of conversion rate. Besides the key behavioral data, we argue that other forms of user behaviors also provide valuable signal, such as views, clicks, adding a product to shop carts and so on. They should be taken into account properly to provide quality recommendation for users.**

In this work, we contribute a novel solution named NMTR (short for Neural Multi-Task Recommendation) for learning recommender systems from multiple types of user behaviors. We develop a neural network model to capture the complicated and multi-type interactions between users and items. In particular, our model accounts for the cascading relationship among behaviors (e.g., a user must click on a product before purchasing it). To fully exploit the signal in the data of multiple types of behaviors, we perform a joint optimization based on the multi-task learning framework, where the optimization on a behavior is treated as a task. Extensive experiments on two real-world datasets demonstrate that NMTR significantly outperforms state-of-the-art recommender systems that are designed to learn from both single-behavior data and multi-behavior data. Further analysis shows that modeling multiple behaviors is particularly useful for providing recommendation for sparse users that have very few interactions.

***Index Terms*—Recommendation, Collaborative Filtering, Deep Learning, Multi-behavior Data**

I. Introduction

In online information systems, users interact with a system in a variety of forms. For example, in an E-commerce website, a user can click on a product, add a product to shopping cart, purchase a product and so on.

In traditional recommender systems, only user-item interaction data of one single type of user behavior is considered for collaborative filtering, such as the purchase behavior in E-commerce and the rating behavior on movies [1], [2]. While it is particularly useful to optimize a recommender model on the data that is directly related to the business KPI, the other forms of behaviors should not be neglected, since they also provide valuable signal on a user's preference.

Existing approaches for multi-behavior recommendation can be divided into two categories. The first category is based on collective matrix factorization (CMF) [3]–[6], which extends the matrix factorization (MF) method to jointly factorize multiple behavior matrices. In MF, a user (or an item) is described as an embedding vector to encode her preference (or its property), and a user-item interaction is estimated as the inner product of the user embedding and item embedding. To correlate MF on multiple behavior matrices, it is essential to share the embedding matrix of entities of one side (e.g., items), and let the entities of the other side (e.g., users) learn different embedding matrices for different types of behaviors.

The second category approaches the problem from the perspective of learning [1], [7]. To learn recommender models from the (implicit) data of interactions, it is natural to assume that a user's interacted items should be more preferable over the non-interacted items. Bayesian Personalized Ranking (BPR) [1] is a representative method that implements the assumption of relative preference; it is then extended to address multi-behavior recommendation [7] by enriching the training data of relative preference from the multi-behavior data.

Despite effectiveness, we argue that existing models for multi-behavior recommendation suffer from three limitations.

- **Lack of behavior semantics**. Each type of behaviors has its own semantics and contexts, and more importantly, there exist strong ordinal relations among different behavior types. For example, the behaviors may represent the action sequence of a user on a product: click should happen prior to add-to-cart, and add-to-cart should happen before purchase. Moreover, the semantics make some intermediate feedback rather meaningful, such as the products that are viewed but not purchased. However, existing models have largely ignored the semantics of different behavior types.
- **Unreasonable embedding learning**. The CMF paradigm needs to enforce the entities of one side (either users or items) have different embedding matrices for different types of behaviors. From the perspective of representation learning and interpretation of latent factor models [8], [9], this setting is not so unreasonable. Specifically, a user's embedding vector represents his/her inherent interests, which should remain unchanged when the user performs different types of behaviors on items; and similarly for the item side. In addition, using separate embeddings for different behaviors may be non-optimal due to losing connections, as you mentioned in the comment. Besides, when embedding are separated, it is hard to be learned well for those sparse behaviors.
- **Incapability in modeling complicated interactions**. Existing methods largely rely on MF to estimate a user's prefer-

ence on an item. In MF, the interaction function is a fixed inner product, which is insufficient to model the complicated and multi-type interactions between users and items. This is also a major reason why these CMF methods need to enforce entities of one side to have different embedding matrices for predicting different types of behaviors; otherwise, the model could not make distinct predictions for different behavior types.

To address the above mentioned limitations in multi-behavior recommendation, we propose a new solution named **N**eural **M**ulti-**T**ask **R**ecommendation (NMTR). Briefly, our method combines the recent advance of neural collaborative filtering with multi-task learning to effectively learn from multiple types of user behaviors. Specifically, we separate the two components of embedding learning and interaction as advocated by the NCF framework. We then design that 1) a user (and an item) has a shared embedding across multiple types of behaviors, and 2) a data-dependent interaction function is learned for each behavior type. Through this way, we address the inherent limitations of CMF methods and make the model more suitable for learning from behaviors of multiple types.

Moreover, to incorporate the behavior semantics, especially the ordinal relation among behavior types, we relate the model prediction of each behavior type in a cascaded manner. To be specific, assuming we have two form of behaviors, view and purchase, which form a natural ordinal relation: view $\rightarrow$ purchase. We enforce that the prediction of a high-level behavior (i.e., purchase) comes from the prediction of the low-level behavior (i.e., view). Through this way, we can capture the underlying semantics that a user must view a product in order to purchase it.

To summarize, the main contributions of this work are as follows.

- We propose a novel neural network model tailored to learning user preference from multi-behavior data. The model shares the embedding layer for different behavior types, and learns separate interaction function for each behavior type.
- To capture the ordinal relations among behavior types, we propose to correlate the model prediction of each behavior type in a cascaded way. Furthermore, we train the whole model in a multi-task manner to make full use of multiple types of behaviors.
- To demonstrate the effectiveness of our proposal, we implement three variants of NMTR using different neural collabrative filtering models as the interaction function. Extensive experiments on two real-world datasets show that our method outperform best existing methods by 6.08% and 30.76% on two datasets, respectively.

  As a side contribution, we will release our implementation upon acceptance.

The remainder of the paper is as follows. We first formalize the problem and introduce some preliminaries in Section II. We then present our proposed method in Section III. We conduct experiments in Section IV, before reviewing related work in Section V and concluding the paper in Section VI.

II. PRELIMINARIES

We first formulate the problem to solve in this paper. Then we recapitulate the neural collaborative filtering technique [2].

Lastly, we introduce collective matrix factorization, a prevalent solution for multi-behavior recommendation.

A. Problem Formulation

In recommender systems, there typically exists a key type of user behaviors to be optimized, which we term it as the *target behavior*. For example, in an E-commerce site, the target behavior is usually purchase, since it is directly related with the conversion rate of recommendation and is the strongest signal to reflect a user's preference. Traditional collaborative filtering techniques [1], [10] focus on the target behavior only and forgo other types of user behaviors such as views, clicks, etc., which are readily available in the server logs. The focus of this work is to leverage these other types of user behaviors to improve the recommendation for the target behavior.

Let $\{\mathbf{Y}^1, \mathbf{Y}^2, ..., \mathbf{Y}^R\}$ denote the user-item interaction matrices for all the R types of behaviors. Each interaction matrix is of size $M \times N$, where M and N denote the number of users and items, respectively. Since in real-world applications, most user feedback are in the implicit form [1], [11], we assume that each entry of the interaction matrix has a value of 1 or 0:

$$y_{ui}^r = \begin{cases} 1, & \text{if } u \text{ has interacted with } i \text{ under behavior } r; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

As we have discussed in the introduction, many user behavior types in real-world applications follow an ordinal (or sequential) relationship. Without loss of generality, we assume that the behavior types have a total order and sort them from the lowest level to the highest level: $\mathbf{Y}^1 \rightarrow \mathbf{Y}^2 ... \rightarrow \mathbf{Y}^R$, where $\mathbf{Y}^R$ denotes the target behavior to be optimized.

Since the target behavior typically concerns the conversion rate, we regard it as having highest priority.

The problem of multi-behavior recommendation is then formulated as follows.

Input: The user-item interaction data of the target behavior $\mathbf{Y}^R$, and the interaction data of other behavior types $\{\mathbf{Y}^1, \mathbf{Y}^2, ..., \mathbf{Y}^{R-1}\}$.

Output: A model that estimates the likelihood that a user u will interact with an item i under the target behavior.

After obtaining the predictive model, we can use it to score all items for a user u, and select the top-ranked items as the recommendation results for u.

B. Neural Collaborative Filtering (NCF)

NCF is generic neural network framework for performing collaborative filtering (CF) on single-behavior data [2]. It applies a representation learning view [12] for CF, representing each user (and item) as an embedding vector. To predict a user's preference on an item, it feeds their embeddings into a neural network:

$$\hat{y}_{ui} = f_{\Theta}(\mathbf{p}_u, \mathbf{q}_i | \Theta), \quad (2)$$

where $\mathbf{p}_u$ and $\mathbf{q}_i$ denote the embedding vector for user u and item i, respectively; f_Θ denotes the neural network with parameters Θ, which is also called as the *interaction function*, since it is responsible for learning the interaction between user embedding and item embedding to obtain the prediction score. The model parameters of NCF can be learned in an end-to-end fashion. Specifically, the authors opt to optimize a pointwise log loss, where the positive instances are the entries of value 1 (aka., observed entries) in the user-item interaction matrix $\mathbf{Y}^R$ and the negative instances are randomly sampled from the entries of value 0 (aka., missing data).

The matrix factorization model can be seen as the special case of NCF — by specifying the interaction function f_Θ as an inner product, NCF exactly recovers MF. As such, under the NCF framework, MF can be interpreted as using a fixed, data-independent interaction function. As demonstrated in the NCF paper and its follow-up work [13], using such a fixed interaction function is suboptimal and can be improved by learning the interaction function from data. It is this evidence that motivates us to develop neural network models to address the multi-behavior recommendation task.

In the NCF paper, the authors present three instantiations of NCF, namely, GMF, MLP and NeuMF. Briefly, GMF generalizes MF by defining f_Θ as an element-wise product layer followed by a weighted output layer. MLP employes multi-layer perceptron above the concatenation of $\mathbf{p}_u$ and $\mathbf{q}_i$ to learn the interaction function. The best performance is achieved by NeuMF, which concatenates the element-wise product layer of GMF and the last hidden layer of MLP, feeding it to a weighted output layer to obtain the prediction score. Our NMTR uses NCF as a building block, and as such, any design of f_Θ can be used as a component to learn the interaction function for one behavior type in our method.

C. Collective Matrix Factorization

CMF is originally proposed to factorize multiple data matrices that have certain common entities [3]. For example, it can be used to factorize user-movie and movie-genre matrix, where movies are the common entities of the two data matrices. The idea is to correlate the multiple factorization processes by sharing the embeddings of common entities.

Nevertheless, in multi-behavior recommendation, both sides of entities are shared in data matrices of different behavior types. Directly applying CMF will fail to produce different predictions for different behavior types. To address this problem, Zhao *et al.* [6] proposed to share the item embedding matrix for all behavior types, allowing a user to learn different embedding vectors for different behavior types. To be specific, the objective function to optimize is as follows:

$$\min_{\mathbf{p}_*, \mathbf{q}_*} \sum_{r=1}^{R} \sum_{u=1}^{M} \sum_{i=1}^{N} c_{ui}^r (y_{ui}^r - {\mathbf{p}_u^r}^T \mathbf{q}_i)^2, \quad (3)$$

where c_{ui}^r denotes the importance of the entry y_{ui}^r in factorization, $\mathbf{q}_i$ denotes the embedding vector for item i that is shared by all behavior types, and $\mathbf{p}_u^r$ denotes the embedding vector for user i in reconstructing the behaviors of the r-th type. Note that we have omitted the L_2 regularization term for clarity.

As argued earlier in the introduction, this setting is not so irrational and non-interpretable as a latent factor model. Specifically, an embedding vector for a user encodes his/her latent interest, which should remain unchanged when the user seeks items of interest to consume at a particular time. Moreover, other potential limitations of existing CMF methods include the use of a fixed interaction function of inner product, and the use of squared regression loss for optimization, which may be suboptimal for item recommendation with implicit feedback [1], [2].

III. Methods

Figure 1 illustrates our proposed NMTR model. Given a user-item pair (u, i) as the input, the model aims to predict the likelihood that u will perform a behavior (of any of the R types) on item i, represented as the output of $\{\hat{y}_{ui}^1, \hat{y}_{ui}^2, ..., \hat{y}_{ui}^R\}$.

Our NMTR method is featured with four special designs:

- **Shared embedding layer**. To make it reasonable under the paradigm of representation learning, we share the embedding layer of users and items for the modeling of all behavior types.
- **Separated interaction function**. We learn different interaction functions for predicting the behaviors of different types. This is achieved by using the expressive NCF unit for each type of behaviors.
- **Cascaded predictions**. To capture the ordinal relations among behavior types, we correlate the predictions of different behavior types through cascading.
- **Multi-task learning**. To optimize the cascaded architecture, we simultaneously train the predictive models for all behavior types by performing multi-task learning.

In what follows, we present our method by elaborating the above four designs.

A. Shared Embedding Layer

In order to make our proposed model extensible, we apply one-hot encoding to encode the input of user ID and item ID. The advantage is that it can be easily extended to incorporate other features of a user and an item (e.g., user demographics and item attributes), if they are available in an application. Let $\mathbf{v}_u^U$ and $\mathbf{v}_i^I$ denote the one-hot feature vector for user u and item i. Then the embedding layer is defined as a linear fully connected layer without the bias terms:

$$\mathbf{p}_u = \mathbf{P}^T \mathbf{v}_u^U, \quad \mathbf{q}_i = \mathbf{Q}^T \mathbf{v}_i^I, \quad (4)$$

where $\mathbf{P}$ and $\mathbf{Q}$ are the user embedding matrix and item embedding matrix, respectively. When only the ID feature is used to describe a user (or an item), $\mathbf{P}$ and $\mathbf{Q}$ are of the size $M \times E$ and $N \times E$, respectively, where E denotes the embedding size; and $\mathbf{p}_u$ and $\mathbf{q}_i$ are essentially the $u-$th and $i-$th row vector of $\mathbf{P}$ and $\mathbf{Q}$, respectively.

It is worth noting that NMTR has only one embedding layer in the lower part of the model, which is to be used for the prediction of all behavior types in the upper part. Based on this design, we can interpret the model under the paradigm of representation learning, where $\mathbf{p}_u$ and $\mathbf{q}_i$ are the latent features to be learned to represent user u and item i, respectively.

B. Separated Interaction Function

Above the embedding layer is the hidden layers that model the interaction between $\mathbf{p}_u$ and $\mathbf{q}_i$ to obtain the prediction score. Since we need to predict the likelihood of multiple behavior types with the same input, it is essential to learn a separated interaction function for each type. Let f_Θ^r denote the interaction function for the r-th type of behaviors with parameters Θ, which outputs the likelihood that u will perform a behavior of the r-th type:

$$\hat{y}_{ui}^r = \sigma(f_\Theta^r(\mathbf{p}_u, \mathbf{q}_i)), \tag{5}$$

where σ denotes the sigmoid function converting the output to a probability. A good design of f_Θ^r is to have the ability and sufficient flexibility to learn the possible complicated patterns (e.g., collaborative filtering and others) in user behaviors. To achieve this, we consider the three neural network units proposed in the NCF paper [2]:

- **GMF** generalizes MF by allowing different dimensions of the embedding space to have different weights. To be specific, it first uses an element-wise product to get an interacted vector, and then project the vector to an output score with a weight vector:

$$f_\Theta^{GMF}(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{h}^T(\mathbf{p}_u \odot \mathbf{q}_i), \tag{6}$$

  where $\mathbf{h} \in \mathbb{R}^{E\times 1}$ denotes the learnable weight vector. The parameters of the GMF unit are $\Theta_{GMF} = \{\mathbf{h}\}$.
- **MLP** applies a multi-layer perceptron on the concatenation of $\mathbf{p}_u$ and $\mathbf{q}_i$ to learn the interaction function in a hierarchical and non-linear manner:

$$\begin{aligned} \mathbf{z}_1 &= ReLU(\mathbf{W}_1 \begin{bmatrix} \mathbf{p}_u \\ \mathbf{q}_i \end{bmatrix} + \mathbf{b}_1), \\ &\cdots\cdots \\ \mathbf{z}_L &= ReLU(\mathbf{W}_L \mathbf{z}_{L-1} + \mathbf{b}_L), \\ f_\Theta^{MLP}&(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{h}^T \mathbf{z}_L, \end{aligned} \tag{7}$$

  where L denotes the number of hidden layers in the multi-layer perceptron, $\mathbf{W}_x$ and $\mathbf{b}_x$ denote the weight matrix and bias vector for the x-th hidden layer, and $\mathbf{z}_x$ are the intermediate neurons. By default, the rectifier unit (ReLU) is used as the activation function for the hidden layer, which is beneficial to build deep models. The parameters of the MLP unit are $\Theta_{MLP} = \{\mathbf{h}, \{\mathbf{W}_x\}_{x=1}^L, \{\mathbf{b}_x\}_{b=1}^L\}$.
- **NeuMF** combines the advantage of the linear GMF with the nonlinear MLP to learn the interaction function:

$$f_\Theta^{NeuMF}(\mathbf{p}_u, \mathbf{q}_i) = \mathbf{h}^T \begin{bmatrix} \mathbf{p}_u \odot \mathbf{q}_i \\ \mathbf{z}_L \end{bmatrix}, \tag{8}$$

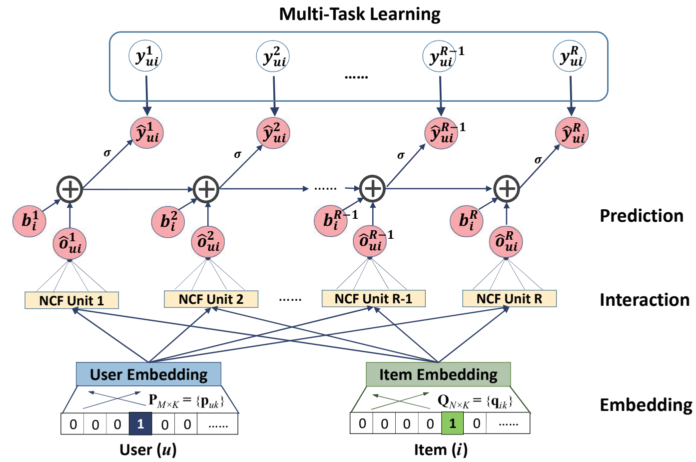


Fig. 1. Illustration of our proposed NMTR model.

  where $\mathbf{z}_L$ indicates the last hidden layer of MLP, as have been defined in Equation (7). The $\mathbf{z}_L$ is concatenated with $\mathbf{p}_u \odot \mathbf{q}_i$ as the hidden layer of NeuMF, which is then projected to a score through the weighted vector $\mathbf{h} \in \mathbb{R}^{2E\times 1}$. In the original design of NeuMF, the authors used different embedding layers for GMF and MLP. While in our method, we have only one set of embeddings for users and items. As such, we tweak the NeuMF unit by sharing the embedding layer of GMF and MLP.

Note that any of the three units can be used to model for behaviors of any type, and the optimal setting may depend on the dataset. We will empirically evaluate the performance of three NCF choices and their impact on our NMTR method in Section IV. There are many other possible designs for the NCF units, such as placing more layers above the hidden layer of NeuMF to thoroughly merge GML and MLP, among others [11], [13]. Since the focus of this paper is not to develop new NCF units for interaction learning, we leverage existing ones as the building block for our NMTR model.

C. Cascaded Predictions

Typically there are certain ordinal relations among behavior types in a real-world application, such as a user must view a product (i.e., click the product page) before she can purchase it. The existence of such relations implies that the predictive model for different behavior types should be related with each other, rather than being independent.

To encode the sequential effect, we enforce that the prediction of a behavior lies in the prediction of the precedent behavior type. Formally, we cascade the prediction of different behaviors as:

$$\begin{aligned} \hat{y}_{ui}^r &= \sigma(out_{ui}^r) \text{ for } r \in \{1, 2, ..., R\}, \\ out_{ui}^R &= out_{ui}^{R-1} + f_\Theta^R(\mathbf{p}_u, \mathbf{q}_i) + b_i^R, \\ &\cdots\cdots \\ out_{ui}^2 &= out_{ui}^1 + f_\Theta^2(\mathbf{p}_u, \mathbf{q}_i) + b_i^2, \\ out_{ui}^1 &= f_\Theta^1(\mathbf{p}_u, \mathbf{q}_i) + b_i^1, \end{aligned} \tag{9}$$

where b_i^r denotes the bias of item i in the data of the r-th behavior type, and f_Θ^r denotes the interaction function for the r-th type of behaviors, which can be any of the three NCF units as introduced before. Note that the predictive value is in a range from 0 to 1, so we adopt *sigmod* function σ to summation out_{ui} to obtain $\hat{y}_{ui}$. The item bias term can capture some discrepancy effects in different types of behaviors, for example, some items are likely to be clicked by users (e.g., products on campaign) but less likely to be purchased. Moreover, some previous work has demonstrated that incorporating item bias is more effective than incorporating user bias for learning from single-behavior implicit feedback [10].

A graphical illustration of our cascading design[1] can be found in the top part of Figure 1. Such a design is particularly useful for predicting the preference of inactive users that have few data on the target behavior. Typically, the data of low-level behaviors (e.g., clicks) is easier to collect and has a larger volume than the target behavior (e.g., purchases). By basing the prediction of target behavior on its precedent types of behaviors, we can achieve better prediction when the target behavior data of a user is insufficient to estimate f_Θ^R well.

D. Multi-Task Learning

As we have a dedicated model for each type of behaviors and the models follow a cascading prediction, it is intuitive to train the models separately by following the order of $\hat{y}_{ui}^1, \hat{y}_{ui}^2, ..., \hat{y}_{ui}^R$. Since these models share the same embedding layer and the final recommendation is based on the last target model $\hat{y}_{ui}^R$, this way can be seen as pre-training the embedding layer of the target model using other types of behaviors. We argue that such a sequential training manner does not make full use of the multi-behavior data, since it only uses precedent models to improve the next model while there is no benefit for the precedent models. A better solution could be to let the models reinforce each other.

In contrast to training the models separately, multi-task learning (MTL) is a paradigm that performs joint training on the models of different but correlated tasks, so as to obtain a better model for each task [14]. The intuition for our design of cascaded predictions is that, if we can obtain improved models for other types of behavior, the model for the target behavior can also be improved. As such, we opt for MTL that trains all models simultaneously, where the model learning for each behavior type is treated as a task.

Objective Function. Following the probabilistic optimization framework [1], [2], we first define the likelihood function for a single behavior type as:

$$P_r = \prod_{(u,i)\in\mathcal{Y}_r^+} \hat{y}_{ui}^r \prod_{(u,i)\in\mathcal{Y}_r^-} (1 - \hat{y}_{ui}^r), \tag{10}$$

[1]Note that we assume that the behaviors can form a full-order cascading relationship, while in real world the relationship might be more complicated. For example, there is no sequential relation between sharing a product to social network and adding it to cart by nature. Technically speaking, we can adapt to such partial-order relation by sorting the behaviors by their strength in reflecting user preference. To conclude, order of different types of behaviors can be sort in two manners: nature manner and preference-level based manner. We focus on the first on leave exploration of second one as future work.

where $\mathcal{Y}_r^+$ denotes the set of observed interactions in behavior matrix $\mathbf{Y}^r$, and $\mathcal{Y}_r^-$ denotes negative instances to be sampled from the unobserved interactions in $\mathbf{Y}^r$. We then get the joint probability for multiple types of behaviors as:

$$P = \prod_{r=1}^{R} P_r = \prod_{r=1}^{R} \prod_{(u,i)\in\mathcal{Y}_r^+} \hat{y}_{ui}^r \prod_{(u,i)\in\mathcal{Y}_r^-} (1 - \hat{y}_{ui}^r). \tag{11}$$

Taking the negative logarithm of the joint probability, we obtain the loss function to be minimized as:

$$L = -\sum_{r=1}^{R} \lambda_r \Big(\sum_{(u,i)\in\mathcal{Y}_r^+} \log \hat{y}_{ui}^r + \sum_{(u,i)\in\mathcal{Y}_r^-} \log(1 - \hat{y}_{ui}^r) \Big), \tag{12}$$

where we additionally include the term λ_r to control the influence of the r-th type of behaviors on the joint training. This is a hyper-parameter to be specified for different datasets, since the importance of a behavior type may vary for problems of different domains and scales. Moreover, we enforce that $\sum_{r=1}^{R} \lambda_r = 1$ to facilitate the tuning of these hyper-parameters.

Directly optimizing this joint loss function will update the parameters of models for multiple behavior types together. As such, a better embedding learned from a gradient step of the data of one type will benefit the learning of other types.

Training. Since our model is composed of nonlinear neural networks, we optimize parameters with stochastic gradient descent (SGD), a generic solver for neural network models. As most machine learning toolkits (e.g., TensorFlow, Theano, PyTorch etc.) provide the function of automatic differentiation, we omit the derivation of the derivatives of our model. Instead, we elaborate on how to form a mini-batch to facilitate faster training, since modern computing units like GPU and CPU provide acceleration for matrix-wise float operations.

To generate a mini-batch, we first sample a user-item pair (u, i) such that user u has at least one observed interaction on item i (regardless of the behavior type). We then inspect the interactions of the (u, i) pair — for each observed interaction, we sample a negative instance from u's unobserved interactions of the behavior type. As an example, if the sampled (u, i) pair has an interaction in the 1-st behavior and 2-nd behavior, we get two positive training instances y_{ui}^1 and y_{ui}^2; we then sample two items t and s that u did not interact under first two behaviors, respectively, to get two negative instances y_{ut}^1 and y_{us}^2. To put it in other words, user u has added item i to cart; item s is sampled from unobserved items in u's interaction history while item t is sampled from unobserved or only-viewed items in u's interaction history. We iterate the above sampling step until the desired size of a mini-batch is reached.

Note that we empirically find that sampling multiple negative instances to pair with a positive instance in a mini-batch can improve the performance. This finding has been reported before in optimizing neural recommender models with log loss on single-behavior data [2], [15]. As such, in our experiments, we allow a flexible tuning of the negative sampling ratio.

IV. EXPERIMENTS

In this section, we conduct extensive experiments on two real-world datasets to answer the following research questions:

- **RQ1:** How does our proposed NMTR perform as compared with state-of-the-art recommender systems that are designed for learning from single-behavior and multi-behavior data?
- **RQ2:** How do the key hyper-parameters affect NMTR's performance, and how is the effectiveness of our designed multi-task learning for the task?
- **RQ3:** Can NMTR help to address the data sparsity problem, i.e., improving recommendations for sparse users with fewer interactions of the target behavior?

In what follows, we first describe the experimental settings, and then answer the above three research questions.

A. Experimental Settings

1) Datasets and Evaluation Protocol: We experimented with two real-world E-commerce datasets that contain multiple types of user behaviors including purchases, views, adding to carts, etc.

- **Beibei Dataset**[2]**.** This dataset is collected from Beibei, the largest E-commerce platform for maternal and infant products in China. We sampled a subset of user interactions that contain views, adding to carts (abbreviated as *carts*), and purchases within the time period from 2017/06/01 to 2017/06/30.
- **Tmall Dataset**[3]**.** This is the dataset released in IJCAI-15 challenge[4], which is collected from Tmall, the largest business-to-consumer E-Commerce website in China. It records two types of user behaviors, views and purchases, within the time period from 2014/05/01 to 2014/11/30.

For both datasets, we merged the duplicated user-item interactions by keeping the earliest one, which is a common setting in existing researches [1], [16]; the rationality here is to test the performance of a method in recommending novel items that a user did not consume before. Moreover, we focused on users with more than one type of behavior. Note that in the Beibei dataset, an observed buy interaction must after adding to cart; an observed interaction of adding to cart must come after view. The Tmall dataset is similar with it. After the above pre-processing steps, we obtained the final evaluation datasets, the statistics of which are summarized in Table I. A real-world dataset with the number of users above than 10,000 is enough to conduct off-line evaluation [1], [10]. Therefore, experimental results on these two datasets are reliable and persuasive. In addition, our carefully collected Beibei dataset is very precise and are more abundant than any public dataset. We believe in that it will benefit the community after we make it public.

In the evaluation stage, given a user in the testing set, each algorithm ranks all items that the user has not interacted before.

[2]https://www.beibei.com

[3]https://www.tmall.com

[4]https://tianchi.aliyun.com/datalab/dataSet.htm?id=5

TABLE I

STATISTICS OF OUR EVALUATION DATASETS.

Dataset	User#	Item#	Purchase#	Cart#	View#
Beibei	21,716	7,977	295,622	642,622	2,412,586
Tmall	15,670	9,076	136,648	–	813,396

We applied the widely used leave-one-out technique to obtain the training set and test set, which means for every user, there is a test item her has not interacted with. We then adopted two popular metrics, *HR* and *NDCG*, to judge the performance of the ranking list:

- **HR@K:** *Hit Ratio* (HR) measures whether the test item is contained by the top-K item ranking list (1 for yes and 0 for no).
- **NDCG@K:** *Normalized Discounted Cumulative Gain* (NDCG) complements HR by assigning higher scores to the hits at higher positions of the ranking list.

2) Baselines: We compared the performance of our proposed NMTR with 9 baselines, which can be divided into two groups based on whether it models single-behavior or multi-behavior data. The compared single-behavior methods are introduced as follows.

BPR [1] *Bayesian Personalized Ranking* (BPR) is a widely used pairwise learning framework for item recommendation with implicit feedback. Same as the original paper, we used BPR to optimize the MF model.

NCF [2] *Neural Collaborative Filtering* (NCF) is a neural framework to learn interactions between the latent features of users and items. As we employed three NCF methods, named **GMF**, **MLP** and **NeuMF** to learn the interaction function for each behavior type, we evaluated how the three methods perform for single-behavior data.

The second group of five compared methods that can leverage multiple types of behavior data are as follows.

CMF [6] As have described in Section II-C, CMF decomposes the data matrices of multiple behavior types simultaneously. We adapted the method by sharing the user embeddings for factorizing different interaction matrices of various types of behaviors.

As our datasets are implicit feedback, we further augmented the method by sampling negative instances in the same way as our NMTR.

MC-BPR [7] Multi-Channel BPR [7] is the state-of-the-art solution for multi-behavior recommendation. It adapts the negative sampling rule in BPR to account for the levels of user feedback in multi-behavior data. For example on the Tmall dataset that has two behavior types — purchase and view, to generate a negative sample for a purchase interaction, it assigns different probabilities for sampling from 1) items that are viewed but not purchased, and 2) items that are not viewed. We tuned the probability distribution for sampling and reported the best results.

TABLE II
BEST PARAMETER SETTINGS OF OUR PROPOSED NMTR METHODS FOR TOP-K RECOMMENDATION

Dataset	Parameter	NMTR-GMF	NMTR-MLP	NMTR-NeuMF
Beibei	Optimzer	Adagrad	Adagrad	Adagrad
	Learning rate	0.01	0.01	0.01
	Number of layer	-	3	3
	Loss coefficient	[1/3,1/3,1/3]	[1/3,1/3,1/3]	[1/3,1/3,1/3]
	Regularization	[0,1e-5]	[0,1e-5]	[0,1e-5]
Tmall	Optimzer	Adagrad	Adagrad	Adagrad
	Learning rate	0.01	0.01	0.05
	Number of layer	-	3	3
	Loss coefficient	[0.4,0.6]	[0.5,0.5]	[0.4,0.6]
	Regularization term	[0,5e-5]	[0,1e-5]	[0,0]

MC-NCF Since Multi-Channel BPR is a generic learning method that is applicable to any differentiable recommender model, we replaced the basic MF model in it with state-of-the-art NCF models, and named this extension as MC-NCF. That is, we optimized the three NCF models with the Multi-Channel BPR learner, and named the respective methods as **MC-GMF**, **MC-MLP** and **MC-NeuMF**.

3) Parameter Settings: We implemented our NMTR and baseline methods in TensorFlow[5]. Since we have three choices of NCF units as the interaction function, we name the respective methods as **NMTR-GMF**, **NMTR-MLP** and **NMTR-NeuMF**.

We randomly selected a training instance for each user as the validation set to tune hyper-parameters. For all methods, we set the embedding size to 64, a relatively larger number that achieves good performance on our datasets.

For CMF, one important hyper-parameter is the weight of different behavior types in the joint loss. We tuned the weight for each behavior in [0, 0.2, 0.4, 0.6, 0.8, 1].

For neural network models, we initialized their parameters using the method proposed in [17]. For models that have multiple hidden layers, i.e., MLP, MC-MLP, NMTR-MLP, NeuMF, MC-NeuMF and NMTR-NeuMF, we employed a tower structure for the hidden layers same as [2], and tuned the number of layers from 1 to 5.

We set the negative sampling ratio as 4 for all methods, an empirical value that shows good performance. We tried two SGD-based optimizers, Adam [18] and Adagrad [19], and tuned the learning rate for each optimizer in [0.001, 0.005, 0.01, 0.02, 0.05].

Moreover, we applied L_2 regularization to all methods to prevent overfitting.

B. Performance Comparison (RQ1)

We first compare the top-K recommendation performance with state-of-the-art methods. We investigate the top-K performance with K setting to [50, 80, 100, 200]. Note that for a user, our evaluation protocol ranks all unobserved items in the training set [10]. Though this all-ranking protocol can be very time-consuming, the obtained results are more persuasive than ranking a random subset of negative times only (e.g., as have done in [2]). In this case, small values of K will make the results have a large variance and unstable. As such, we report results of a relatively large[6].

We report the best parameter setting for our proposed NMTR methods in Table II.

Table III shows the performance of HR@K and NDCG@K for our three NMTR methods, five multi-behavior recommendation methods, and four single-behavior methods. From the results, we have the following observations:

- **NMTR achieves the best performance.** Our proposed NMTR methods obtain the best performance in terms of HR@K and NDCG@K as compared to all baselines. The one-sample paired t-tests indicate that all improvements are statistically significant for $p < 0.05$. Among the three NMTR methods, NMTR-GMF and NMTR-NeuMF are better than NMTR-MLP, which verifies the effectiveness of the element-wise operator in learning the user-item interaction function.
  Compared with the best single-behavior baseline NeuMF, NMTR outperforms it by 9.01% in HR and 6.72% in NDCG on the Beibei dataset; and the improvements are 13.04% in HR and 9.91% in NDCG on the Tmall dataset. Compared with MC-NeuMF, which extends NeuMF on multi-behavior data with the Multi-Channel BPR [7], NMTR obtains an improvement in HR of 6.08% and 10.23% on Beibei and Tmall, respectively.
  In addition, we can observe that MF based methods (CMF, MC-BPR and BPR), achieve the worst performance on the Beibei dataset, which has more complicated and richer behaviors than the Tmall dataset. This confirms the incapability of MF in modeling complicated interactions between users and items, being inferior to the multi-layer neural networks.
- **NMTR is a better framework than MC.** For each NCF model, we find that optimizing it under our NMTR framework outperforms optimizing it under the Multi-Channel BPR framework. Specifically, NMTR-NeuMF outperforms MC-NeuMF by 6.08% on Beibei dataset and 30.76% on Tmall dataset in HR@100. Thus, we can conclude that NMTR performs better than the MC framework in adapting a single-behavior recommender model for multiple behaviors.
  To better understand the difference between two frameworks, we present the training loss and the testing performance in each training iteration in Figure 2 (for Beibei) and Figure 3 (for Tmall). In our NMTR framework, the training loss is defined as the joint loss in multi-task learning, which is a combination of the prediction loss of behaviors of multiple types. We can observe that, for both datasets,

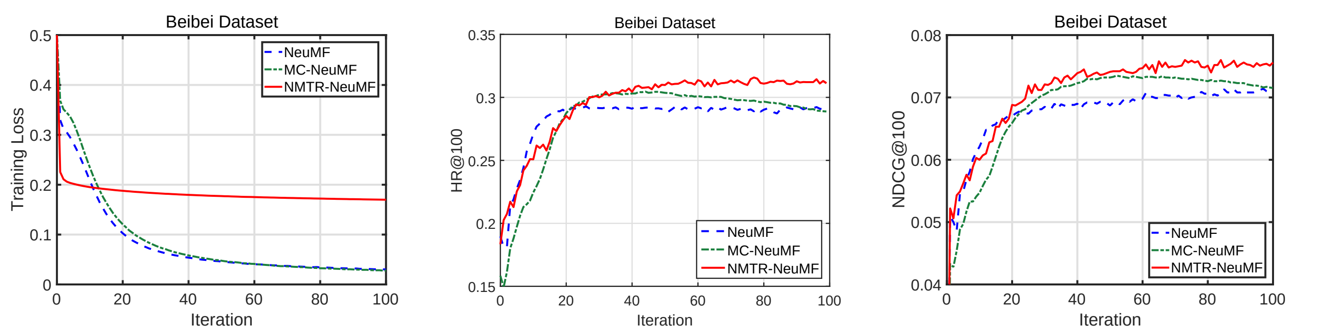
[5] https://www.tensorflow.org

[6] There is another reason to choose a relatively larger K. In practical recommender systems, the procedure of item recommendation is typically divided into two stages [20]: candidate selection and re-ranking. In this work, our proposed model are one kind of collaborative filtering (CF) methods, which are typically applied in the first stage. Since the aim of first stage is to retrieve a few hundreds of relevant items, a larger K to evaluate CF methods is more reasonable.

TABLE III
TOP-K RECOMMENDATION PERFORMANCE COMPARISON ON THE BEIBEI AND TMALL DATASETS (K IS SET TO 50, 80, 100, 200)

		Beibei Dataset							
Group	**Method**	**HR@50**	**NDCG@50**	**HR@80**	**NDCG@80**	**HR@100**	**NDCG@100**	**HR@200**	**NDCG@200**
Our NMTR Model	NMTR-GMF	0.2050	0.0590	**0.2721**	**0.0688**	0.3119	0.0741	0.4543	0.0961
	NMTR-MLP	0.1928	0.0560	0.2690	0.0676	0.3188	**0.0762**	0.4732	0.0967
	NMTR-NeuMF	**0.2079**	**0.0609**	0.2689	0.0683	**0.3193**	0.0760	**0.4766**	**0.0971**
Multi-behavior	CMF	0.1596	0.0481	0.2377	0.0606	0.2829	0.0663	0.4191	0.0850
	MC-BPR	0.1743	0.0503	0.2299	0.0604	0.2659	0.0647	0.3852	0.0786
	MC-GMF	0.1822	0.0508	0.2425	0.0611	0.2975	0.0690	0.4262	0.0891
	MC-MLP	0.1810	0.0534	0.2342	0.0598	0.2810	0.0684	0.4116	0.0834
	MC-NeuMF	0.2014	0.0577	0.2522	0.0669	0.3010	0.0719	0.4300	0.0897
Single-behavior	BPR	0.1199	0.0348	0.1686	0.0419	0.2002	0.0463	0.3039	0.0624
	GMF	0.1792	0.0475	0.2555	0.0608	0.2920	0.0665	0.4090	0.0828
	MLP	0.1711	0.0483	0.2383	0.0459	0.2679	0.0617	0.3947	0.0792
	NeuMF	0.1828	0.0573	0.2559	0.0668	0.2929	0.0714	0.4078	0.0852

		Tmall Dataset							
Group	**Method**	**HR@50**	**NDCG@50**	**HR@80**	**NDCG@80**	**HR@100**	**NDCG@100**	**HR@200**	**NDCG@200**
Our NMTR Model	NMTR-GMF	0.0778	0.0250	0.1042	0.0293	**0.1196**	0.0314	**0.1751**	0.0390
	NMTR-MLP	0.0734	0.0251	0.0884	0.0277	0.0982	0.0290	0.1672	0.0338
	NMTR-NeuMF	**0.0854**	**0.0315**	**0.1045**	**0.0347**	0.1169	**0.0366**	0.1668	**0.0428**
Multi-behavior	CMF	0.0738	0.0234	0.0940	0.0269	0.1085	0.0287	0.1565	0.0359
	MC-BPR	0.0674	0.0218	0.0928	0.0260	0.1072	0.0282	0.1597	0.0357
	MC-GMF	0.0653	0.0243	0.0778	0.0258	0.0846	0.0264	0.1084	0.0294
	MC-MLP	0.0617	0.0195	0.0784	0.0219	0.0868	0.0228	0.1122	0.0238
	MC-NeuMF	0.0711	0.0296	0.0820	0.0311	0.0894	0.0320	0.1172	0.0359
Single-behavior	BPR	0.6666	0.0200	0.0926	0.0240	0.1058	0.0263	0.1647	0.0342
	GMF	0.0742	0.0271	0.0927	0.0295	0.1027	0.0306	0.1407	0.0355
	MLP	0.0666	0.0194	0.0824	0.0220	0.0905	0.0233	0.1194	0.0273
	NeuMF	0.0760	0.0299	0.0925	0.0321	0.1013	0.0333	0.1383	0.0377

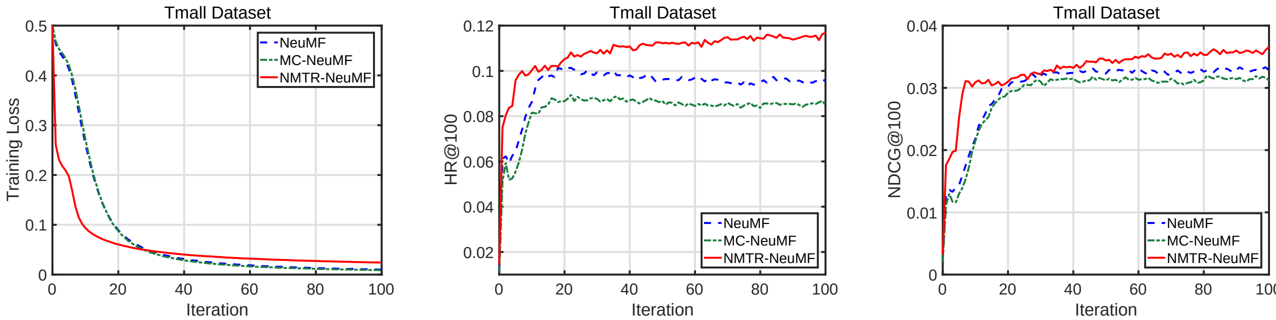


(a) Training Loss (b) HR@100 (c) NDCG@100

Fig. 2. Training loss and testing performance of NeuMF, MC-NeuMF, and NMTR-NeuMF in each iteration on Beibei

although training loss of NMTR is the highest, it essentially demonstrates the best generalization performance.
For the Beibei dataset, we find that the HR score of MC-NeuMF starts to decrease after 40 iterations, even though the L_2 regularization and dropout have been adopted. Note that in Table III, we have reported the peak performance of each baseline evaluated per iteration (such a setting is to fully explore the potential of all methods). Even so, our NMTR still outperforms MC-NeuMF by 6.08% in HR@100 and 5.70% in NDCG@100. However, on the Tmall dataset, in which only two behaviors are available and the data is of a smaller scale, MC-NeuMF fails to utilize the view behavior to improve the performance (i.e., underperforms NeuMF). In contrast, our NMTR-NeuMF outperforms NeuMF by 30.76% in HR@100 and 14.37% in NDCG@100, which are very significant improvements.

- **The performance on multiple behaviors are relevant to that on single behavior.** No matter which framework is chosen, NMTR or MC, we can observe that the performance of the multi-behavior setting is relevant to that of single-behavior. This is because that they use the same set of CF functions, which on the other hand implies that the performance on multi-behavior data maybe limited by the choice of the CF function. An empirical evidence is that NMTR-MLP performs the worst among the three NMTR methods, which can be caused by the poor performance of MLP in modeling CF effect (in single-behavior data). In addition, for some metrics, such as HR@50 and NDCG@50 on both dataset, and HR@80 and NDCG@80 on Tmall dataset, NMTR-MLP and NMTR-GMF are outperformed



(a) Training Loss (b) HR@100 (c) NDCG@100

Fig. 3. Training loss and testing performance of NeuMF, MC-NeuMF, and NMTR-NeuMF in each iteration on Tmall

by some baseline methods such as MC-NeuMF. It can be explained that MC-NeuMF's relatively better performance is due to NeuMF's best performance compared all single-behavior methods. Therefore, our NMTR-NeuMF achieves better performance than MC-NeuMF on these metrics. Moreover, another important finding is that auxiliary behaviors could adversely degrade the performance without a proper modeling. An evidence can be found in the results of the Tmall dataset, where the methods under the MC framework fail to improve the performance in general.

There are another interesting finding worth to be analyzed. Some multi-behavior baselines perform even worse than some single-behaviors baselines. This can be explained with two reasons. First, for CMF, it cannot perform better than NCF (GMF, MLP, NeuMF) since it use fixed dot product as interaction function even though it can exploit multi-behavior data. Second, for MC-BPR and MC-NCF, their sampling strategy is probable to bring noise when purchase data is too sparse.

To summarize, the extensive comparison on two real datasets verify that our proposed NMTR solution can effectively leverage multiple types of behaviors to improve the recommendation performance, i.e. our model outperforms the best baseline method by 6.08% and 30.76% on two datasets, respectively.

C. Impact of Auxiliary Behaviors and Parameters (RQ2)

In order to understand how auxiliary behavior data affect the recommendation performance, we choose the Beibei dataset for further investigation since it has more types of behaviors. Since the motivation of multi-behavior recommendation is to utilize interaction data of other types of behaviors to help improving recommendation quality on target behavior, we investigate how the data quality of auxiliary behaviors affects our NMTR model's performance. A intuitive experimental setting is that to random sample auxiliary behaviors for our utilized two datasets while keeping target behaivor (i.e. purchase) intact. Table IV shows the performance of different combinations of behavioral data. There are four sampling rules for obtaining a subset. For example, (Purchase, 50%view) means that intact purchase records are kept and we randomly select half records of view behavior for each user. As mentioned above, when investigating top-K performance, K=100 is a reasonable setting. Thus, here we evaluated the performance via two metrics: HR@100 and NDCG@100. We tuned hyper-parameters, with a similar way as Section IV-A, to report the best performance for various subsets of interaction data. From the results, we have the following two observations.

First, adding views data leads to better performance than adding carts data.

The main reason is probably that the cart data contains too similar signal with the purchase data and provides fewer new signal on user preference.

Specifically, a purchase record is often accompanied by a carting record. On the contrary, the view behaviors provide some useful intermediate feedback such as, viewed and not bought, which can effectively improve the learning on binary implicit feedback.

Second, by using only 50% of the cart and view interactions, we find that the performance is worse than the previous two experiments. Specifically, the performance of (Purchase, 50% Carting) is worse than only using purchase, while (Purchase, 50% Viewing) is better than only using purchase. There are two major reasons. On one hand, view is the weakest signal to reflect user preference and the total number of views is very large, making the missing of part of view data is acceptable. Therefore, missing of some view records shall not affect the result too much. On the other hand, random missing of carts records can bring some noises, as cart behavior is very similar with the purchase behavior, and this validates the hypothesis in [21]: those missing records of some behaviors are more likely taken as negative value rather than missing value by model.

In order to understand how hyper-parameters impact the performance, we focus on the coefficient in the joint loss function of MTL, λ_r, since it controls the weight of each type of behavior and is a key parameter of our method. There are three and two behavior types for Beibei and Tmall, respectively. For the Beibei dataset, there are three types of behaviors (view, cart and purchase), which means there are three loss coefficients λ_1, λ_2 and λ_3, respectively. Note that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, we tune the three coefficients in $[0, \frac{1}{6}, \frac{2}{6}, \frac{3}{6}, \frac{4}{6}, \frac{5}{6}, 1]$ and plot the performance of HR@100 in

TABLE IV
PERFORMANCE OF NMTR MODEL WITH DIFFERENT COMBINATION OF INTERACTION DATA ON THE BEIBEI DATASET

	Beibei Dataset							
Interaction Subset	**(Purchase, Carting)**		**(Purchase, View)**		**(Purchase, 50% Carting)**		**(Purchase, 50% View)**	
Performance	**HR@100**	**NDCG@100**	**HR@100**	**NDCG@100**	**HR@100**	**NDCG@100**	**HR@100**	**NDCG@100**
NMTR-GMF	0.2979	0.0705	0.3029	0.0726	0.2947	0.0701	0.2953	0.0698
NMTR-MLP	0.2770	0.0670	0.3140	0.0741	0.2726	0.0654	0.3058	0.0725
NMTR-NeuMF	0.2882	0.0691	0.3147	0.0743	0.2778	0.0676	0.3107	0.0737

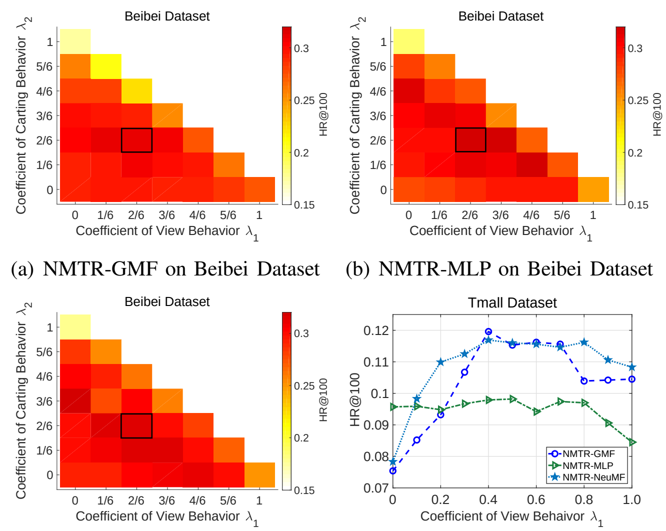
Figure 4(a), 4(b) and 4(c). When λ_1 and λ_2 are given, then value of λ_3 is determined. Therefore each block represents a setting of λ_r. And in these three figures, darker blocks means better performance. Similarly, for the Tmall dataset, there are only two types of behaviors (view and purchase), so there are two coefficients: $\lambda_1 + \lambda_2 = 1$. We tune λ_1 from 0 to 1 with step size 0.1 and plot the HR@100 performance in Figure 4(d). For both datasets, the best performance of the NMTR methods are achieved at almost the same setting, (2/6,2/6,2/6) for the Beibei dataset and about(0.4, 0.6) for the Tmall dataset, which verifies that it is not so independent on the utilized CF unit. For Beibei dataset, in Figure 4(a), 4(b) and 4(c), upper-right blocks are rather shallow since they represent a relatively small λ_3 which is the coefficient of purchase behavior. However, for Tmall dataset, in Figure 4(d), a relatively low coefficient of purchase behavior outperforms that of view behavior. We argue that it is due to size difference of auxiliary behavioral data in two datasets.

Furthermore, as mentioned in Section III-D, we utilize multi-task learning rather than sequential learning to optimize our proposed model. Then to study how multi-task learning outperforms the intuitive sequential learning in optimizing the cascaded prediction models, we compare the performance of the two training methods in Table V. Here we still adopt HR@100 and NDCG@100 as evaluation metrics. For the sequential learning, we feed the cascaded CF units for each behavior type with separated samples following the order of behaviors. We can find that training in the sequential manner achieved worse performance, which verified the effectiveness of our proposed multi-task training component. We also compare the performance on other types of behaviors of these two training manner, and the limited improvement of multi-task learning can be explained the those dense behaviors do not rely so heavily on other behaviors. Another interesting observation is that sequentially trained NMTR models perform worse than single-behavior baselines. Actually, for neural models the initialization of parameters is a key hyper-parameter, and when training NMTR models in a sequential way, other types of behaviors may bring worse initialization when starting to train the neural model with interaction data of the target behavior. Then the recommendation performance on the target behavior is not so good.

We also conduct some experiments to evaluate the model when adopting separate embedding layers for each types of behaviors. We find that it is very hard to converge in the training,

TABLE V
PERFORMANCE COMPARISON OF SEQUENTIAL TRAINING AND MULTI-TASK LEARNING ON THE BEIBEI AND TMALL DATASETS

Dataset	**Beibei**		**Tmall**	
Performance	**HR@100**	**NDCG@100**	**HR@100**	**NDCG@100**
NMTR-GMF	0.3119	0.0741	0.1196	0.0314
Sequential-GMF	0.2730	0.0672	0.0913	0.0290
NMTR-MLP	0.3188	0.0762	0.0982	0.0290
Sequential-MLP	0.2663	0.0692	0.0856	0.0226
NMTR-NeuMF	0.3193	0.0760	0.1169	0.0366
Sequential-NeuMF	0.2704	0.0658	0.0946	0.0304



(a) NMTR-GMF on Beibei Dataset (b) NMTR-MLP on Beibei Dataset

(c) NMTR-NeuMF on Beibei Dataset (d) NMTR on Beibei Dataset

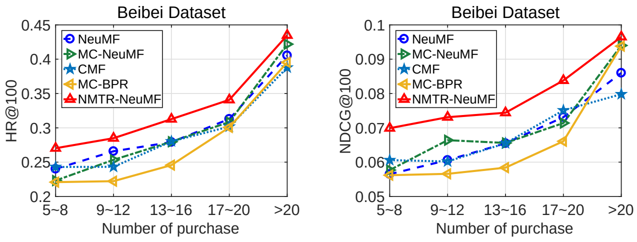
Fig. 4. HR@100 Performance of NMTR with different loss coefficient on the Beibei and Tmall datasets

which can be explained from two perspectives. First, there are too many parameters. Second, some types of behaviors are too sparse to learn their corresponding embedding layer.

In summary, our NMTR incorporates the semantics of different behavior interactions and capture the ordinal relations among them. In addition, coefficient λ_r, as a significant hyper-parameter in our NMTR model, is independent with CF unit.

D. Impact of Data Sparsity (RQ3)

Data sparsity is a big challenge for recommender systems based on implicit feedbacks [10], [22], and multi-behavior recommendation is a typical solution of it. Traditional recommender systems only focus on single behaviors, so the



(a) HR@100 (b) NDCG@100

Fig. 5. Performance of NeuMF, MC-NeuMF, CMF, MC-BPR and NMTR-NeuMF on users with different number of purchase records

data sparsity issue here is defined to sparse interactions on the target behavior. Thus, we study how our proposed NMTR model improves the recommendation for those users having few records of target behavior. Specifically, we divided all users of the Beibei dataset into several groups according to the number of purchase records: [5-8, 9-12, 13-16, 17-20, >20]. In each group, the number of users are in the range of 4000 to 5000, which eliminates the randomness of experimental results.

For each group, we compare the performance of our methods with baseline methods. For NMTR and MC models, we only plot the most competitive ones, NMTR-NeuMF and MC-NeuMF, for clarity; for baselines for single-behavior data, we also only plot the best one, NeuMF.

The results are shown in Figure 5. From the results, we can observe that when the user purchase data becomes sparser, the recommendation performance of NMTR-NeuMF decreases slower than other methods. Especially for NDCG, from fifth to first user group, NMTR-NeuMF is decreased by 27.56% while MC-BPR and MC-NeuMF is decreased by 40.09% and 38.62%. Furthermore, even in the first user group with only 5-8 purchase records, our NMTR still keeps a good recommendation performance of 0.027 for HR@100 and 0.07 for NDCG@100, which outperforms the best baseline by 11.23% and 15.35%, respectively. As a result, the performance gap between NMTR and other methods becomes larger when data become sparser. Since NMTR model learns the other type of behaviors in a reasonable way, it can achieve a good performance for users with sparse interactions. As a summary, we conclude that our proposed NMTR model solves data sparsity problem efficiently to some extent.

In conclusion, we conduct extensive experiments on two real-word datasets, which verifies that our proposed NMTR model outperform existing recommendation methods. Further studies demonstrate our model can alleviate data sparisty problem efficiently.

V. Related Work

A. Multi-Behavior Recommendation.

Multiple behaviors based Recommendation aims to leverage other data of interactions to help improving the recommendation performance on target interaction. Matrix factorization, which is widely used for recommendation of single interaction [1], [10], is already adapted to this task. Ajit *et al.* [3] first proposed a collective matrix factorization model to simultaneously factor several matrices while sharing parameters among factors when an entity participates in multiple relations, named CMF. Some other works applied the CMF to handle datasets of multiple user behaviors [4]–[6]. For example, Park *et al.* [4] applied CMF model by considering an extra item-item matrix from the users' view behavior on E-Commerce website, Zhe *et al.* [6] considered different behaviors in online social network (comment, re-share, and create-post), while Artus *et al.* [5] studied the cold-start problem in multi-relations-based recommendation task on social networks. On the other hand, Babak *et al.* [7] proposed an extension of Bayesian Personalized Ranking (BPR) [1], named Multi-channel BPR, to adapt the sampling rule from different types of behavior in training of standard BPR. As discussed in the introduction, these existing models suffer from several limitation, which is overcomed by our neural-based solution NMTR.

B. Neural Network Based Recommendation.

Salakhutdinov *et al.* [23] proposed a Restricted Boltzmann Machines to predict explicit ratings, which was the first to apply neural network to recommender system. Recently, lots of works utilize neural network to extract the auxiliary information and features in recommender system, such as textual [24], [25], visual [26], [27], audio [28], [28] and video [29]. Rather than these other side features, some other works make use of recurrent neural network to model temporal features in recommender system [30]–[32].

More recently, He *et al.* [2] proposed a neural network architectures for collaborative filtering, named Neural Collaborative Filtering (NCF), which learns the user-item interaction function using neural networks. It has been extended to adapt to different recommendation scenarios [11], [33]. For example, Wang *et al.* [11] applied NCF to model user-item interaction in both information domain and social domain, and Chen *et al.* [33] combined NCF with attention mechanism to recommend videos and images. There are also some works modeling user sequential behaviors via adopting recurrent neural network [20], [22]. Different from them, we focus on the candidate-selection stage in typical recommendation and ignore temporal information.

Our work extends the architecture of NCF to a multi-task learning framework, which aims to solve the problem of learning recommender systems from multi-behavior data.

C. Multi-task Learning for Recommendation.

In multi-task learning (MTL) framework, various related tasks can share common representations, while training in parallel. Traditional multi-task learning works are mainly based on matrix regularization [14], [34] and neural-based approach [35], [36]. To the best of our knowledge, [37] is the first work to apply multi-task learning to recommender system, which built a MTL framework to limit the similarity between users and similarity between items. Bansal *et al.* [38] proposed a gated-recurrent-units based MTL network which share the embedded representation of texts and output personalized text for different users. In contrast, our work adapts MTL our task to effectively learn from multiple user behaviors.

VI. Conclusion and future work

In this work, we designed a recommendation system to exploit multiple types of user behaviors. We proposed a neural network method named NMTR, which combines the recent advances of NCF modeling and the efficacy of multi-task learning. We conducted extensive experiments on two real-world datasets and demonstrated the effectiveness of our NMTR method on multiple recommender models.

This work makes the first step towards understanding how to integrate the rich semantics of users' multiple behaviors into recommender systems. With increasing kinds of user behaviors on the Web, we believe multi-behavior recommendation is an important topic and will attract more attention in the future.

As for future work, we will perform online evaluation of our NMTR method through A/B tests, and focus more on the practical issues of online learning and incremental learning. On the other hand, we will study multi-behavior recommendation in the scenarios that user behaviors cannot form a full-order cascading relation. These behaviors not only contain the normal interactions between users and items, but may also include social interactions among users, such as sharing, following, etc. It is interesting to investigate how to integrate these heterogeneous kinds of user behaviors into a unified recommendation framework. Lastly, we will study time-aware models to capture the evolution of user preference in multi-behavior recommendation.

References

[1] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," in *UAI*, 2009, pp. 452–461.

[2] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *WWW*, 2017, pp. 173–182.

[3] A. P. Singh and G. J. Gordon, "Relational learning via collective matrix factorization," in *SIGKDD*. ACM, 2008, pp. 650–658.

[4] C. Park, D. Kim, J. Oh, and H. Yu, "Do also-viewed products help user rating prediction?" in *WWW*, 2017, pp. 1113–1122.

[5] A. Krohn-Grimberghe, L. Drumond, C. Freudenthaler, and L. Schmidt-Thieme, "Multi-relational matrix factorization using bayesian personalized ranking for social network data," in *WSDM*, 2012, pp. 173–182.

[6] Z. Zhao, Z. Cheng, L. Hong, and E. H. Chi, "Improving user topic interest profiles by behavior factorization," in *WWW*, 2015, pp. 1406–1416.

[7] B. Loni, R. Pagano, M. Larson, and A. Hanjalic, "Bayesian personalized ranking with multi-channel user feedback," in *RecSys*, 2016, pp. 361–364.

[8] Y. Zhang, G. Lai, M. Zhang, Y. Zhang, Y. Liu, and S. Ma, "Explicit factor models for explainable recommendation based on phrase-level sentiment analysis," in *SIGIR*, 2014, pp. 83–92.

[9] R. Heckel, M. Vlachos, T. Parnell, and C. Dünner, "Scalable and interpretable product recommendations via overlapping co-clustering," in *ICDE*, 2017, pp. 1033–1044.

[10] S. Kabbur, X. Ning, and G. Karypis, "Fism: factored item similarity models for top-n recommender systems," in *SIGKDD*, 2013, pp. 659–667.

[11] X. Wang, X. He, L. Nie, and T.-S. Chua, "Item silk road: Recommending items from information domains to social users," in *SIGIR*, 2017.

[12] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *TPAMI*, vol. 35, no. 8, pp. 1798–1828, 2013.

[13] T. Bai, J. Wen, J. Zhang, and W. X. Zhao, "A neural collaborative filtering model with interaction-based neighborhood," in *CIKM*, 2017, pp. 1979–1982.

[14] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *SIGKDD*, 2004, pp. 109–117.

[15] X. Chen, Y. Zhang, Q. Ai, H. Xu, J. Yan, and Z. Qin, "Personalized key frame recommendation," in *SIGIR*, 2017, pp. 315–324.

[16] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *NIPS*, 2008, pp. 1257–1264.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015, pp. 1026–1034.

[18] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[19] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *JMLR*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[20] W. Wang, H. Yin, S. Sadiq, L. Chen, M. Xie, and X. Zhou, "Spore: A sequential personalized spatial item recommender system," in *ICDE*, 2016, pp. 954–965.

[21] H. Steck, "Training and testing of recommender systems on data missing not at random," in *SIGKDD*, 2010, pp. 713–722.

[22] H. Yin, L. Chen, W. Wang, X. Du, Q. V. H. Nguyen, and X. Zhou, "Mobi-sage: A sparse additive generative model for mobile app recommendation," in *ICDE*, 2017, pp. 75–78.

[23] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted boltzmann machines for collaborative filtering," in *ICML*, 2007, pp. 791–798.

[24] L. Zheng, V. Noroozi, and P. S. Yu, "Joint deep modeling of users and items using reviews for recommendation," in *WSDM*, 2017, pp. 425–434.

[25] L. Tang and E. Y. Liu, "Joint user-entity representation learning for event recommendation in social network," in *ICDE*, 2017, pp. 271–280.

[26] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *SIGIR*, 2015, pp. 43–52.

[27] S. Wang, Y. Wang, J. Tang, K. Shu, S. Ranganath, and H. Liu, "What your images reveal: Exploiting visual contents for point-of-interest recommendation," in *WWW*, 2017, pp. 391–400.

[28] A. Van den Oord, S. Dieleman, and B. Schrauwen, "Deep content-based music recommendation," in *NIPS*, 2013, pp. 2643–2651.

[29] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *RecSys*, 2016, pp. 191–198.

[30] S. Wu, W. Ren, C. Yu, G. Chen, D. Zhang, and J. Zhu, "Personal recommendation using deep recurrent neural networks in netease," in *ICDE*, 2016, pp. 1218–1229.

[31] S. Okura, Y. Tagami, S. Ono, and A. Tajima, "Embedding-based news recommendation for millions of users," in *SIGKDD*, 2017, pp. 1933–1942.

[32] Y. Song, A. M. Elkahky, and X. He, "Multi-rate deep learning for temporal recommendation," in *SIGIR*, 2016, pp. 909–912.

[33] J. Chen, H. Zhang, X. He, L. Nie, W. Liu, and T.-S. Chua, "Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention," in *SIGIR*, 2017, pp. 335–344.

[34] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," in *NIPS*, 2007, pp. 41–48.

[35] Y.-G. Jiang, Z. Wu, J. Wang, X. Xue, and S.-F. Chang, "Exploiting feature and class relationships in video categorization with regularized deep neural networks," *TPAMI*, vol. 40, no. 2, pp. 352–364, 2018.

[36] Y. Yang and T. Hospedales, "Deep multi-task representation learning: A tensor factorisation approach," in *ICLR*, 2017.

[37] X. Ning and G. Karypis, "Multi-task learning for recommender system," in *ACML*, 2010, pp. 269–284.

[38] T. Bansal, D. Belanger, and A. McCallum, "Ask the gru: Multi-task learning for deep text recommendations," in *RecSys*, 2016, pp. 107–114.