

Neural Collaborative Reasoning

Hanxiong Chen

Dept. of Computer Science
Rutgers University
hanxiong.chen@rutgers.edu

Shaoyun Shi

Dept. of Computer Science
Tsinghua University
shisy17@mails.tsinghua.edu.cn

Yunqi Li

Dept. of Computer Science
Rutgers University
yunqi.li@rutgers.edu

Yongfeng Zhang

Dept. of Computer Science
Rutgers University
yongfeng.zhang@rutgers.edu

ABSTRACT

Collaborative Filtering (CF) has been an important approach to recommender systems. However, existing CF methods are mostly designed based on the idea of matching, i.e., by learning user and item embeddings from data using shallow or deep models, they try to capture the relevance patterns in data, so that a user embedding can be matched with appropriate item embeddings using designed or learned similarity functions. We argue that as a cognition rather than a perception intelligent task, recommendation requires not only the ability of pattern recognition and matching from data, but also the ability of logical reasoning in the data.

Inspired by recent progress on neural-symbolic machine learning, we propose a neural collaborative reasoning framework to integrate the power of embedding learning and logical reasoning, where the embeddings capture similarity patterns in data from perceptual perspectives, and the logic facilitates cognitive reasoning for informed decision making. An important challenge, however, is to bridge differentiable neural networks and symbolic reasoning in a shared architecture for optimization and inference. To solve the problem, we propose a Modularized Logical Neural Network architecture, which learns basic logical operations such as AND, OR, and NOT as neural modules based on logical regularizer, and learns logic variables as vector embeddings. In this way, each logic expression can be equivalently organized as a neural network, so that logical reasoning and prediction can be conducted in a continuous space. Experiments on several real-world datasets verified the advantages of our framework compared with both traditional shallow and deep models.

KEYWORDS

Collaborative Filtering; Recommender Systems; Neural-Symbolic Learning; Neural Logic Reasoning; Collaborative Reasoning

1 INTRODUCTION

Collaborative Filtering (CF) is an important approach to recommender systems [8, 38]. By leveraging the wisdom of crowd, CF methods predict a user's future preferences based on his or her previous records. Many existing CF methods are designed based on the fundamental idea of similarity matching, with either designed or learned matching functions, as illustrated in Figure 1(a). For example, early CF algorithms, such as User-based CF [37] and Item-based CF [41], consider the row and column vectors in the original

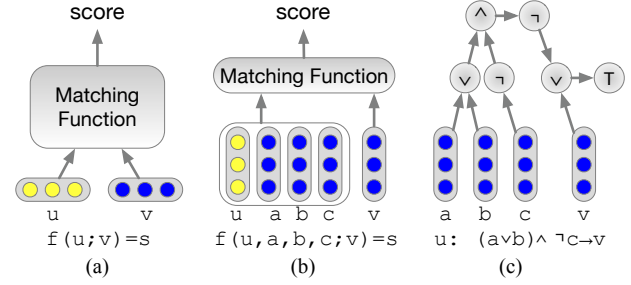


Figure 1: An overview of the fundamental structure of different collaborative filtering algorithms.

user-item rating matrix as the user and item representations (i.e., embedding), and a manually designed weighted average function is used as the matching function $f(\cdot)$ to calculate the relevance score between each user u and a candidate item v . The advance of machine learning has further extended CF methods for improved accuracy. One prominent example is Matrix Factorization (MF) techniques for CF [25], which takes inner product as the matching function $f(\cdot)$, and learns the user and item embeddings in the inner product space to fit ground-truth user-item interactions.

Researchers have further explored CF algorithms under the similarity matching framework. One approach is to learn better embeddings. For example, context-aware CF integrates context information such as time and location to learn informative embeddings [1, 20, 24], and various heterogeneous information sources can be used to enrich the embeddings [44, 50], such as text [51], image [14], and knowledge graphs [45, 48]. We can also explicitly consider a user's behavior history to learn better embeddings (Figure 1(b)), such as in sequential recommendation [5, 16]. Another approach is to use better matching functions. For example, use vector translation instead inner product for similarity matching [13], or learn a distance function in linear space based on metric learning [18], or even learn a universal matching function in non-linear space based on neural networks [15].

Similarity matching-based CF methods have been successful in many real-world recommender systems. However, as a cognition rather than a perception task, recommendation requires not only the ability of pattern learning and matching, but also the ability of cognitive reasoning, because a user's future behavior may not be simply driven by its similarity with the user's previous behaviors, but instead by the user's cognitive reasoning procedure about what to do next. For example, if a user has purchased a laptop before, this does not lead to the user purchasing similar laptops in the future, rather, one would expect the user to purchase further equipment such as a laptop bag. Such a reasoning procedure may exhibit certain logical structures, such as $(a \vee b) \wedge \neg c \rightarrow v$, as shown in Figure 1(c), which means that if the user likes a or b , and does not like c , then he/she would probability like v . The AI community has realized

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, and collaborations, contact the owner/authors(s).

© 2020 Copyright held by the owner/author(s).

the importance of advancing AI techniques from perception to cognition tasks [2]. As a representative cognitive reasoning task, we hope an intelligent recommendation system would be able to discover and leverage the logical relations in data, so as to reason over the data for the prediction of user’s future behaviors.

To achieve this goal, we propose neural collaborative reasoning, which defines the recommendation problem as a differentiable (i.e., neural) logical reasoning problem based on the wisdom of the crowd (i.e., collaborative). In particular, each user’s behavior record is considered as a logical implication rule, such as $(a \vee b) \wedge \neg c \rightarrow v$ in the above example, meaning that the user liked item v given his/her previous preferences on a , b and c . In this sense, each user knows part of the whole logical space, so that we can conduct collaborative logical reasoning based on the collective information from all users to estimate the preferences for new users. More specially, we propose a Neural Logic Recommendation (NLR) framework based on Modularized Logical Neural Networks (MLNN), which integrates the power of embedding learning and logical reasoning in a shared architecture. The model learns each basic logical operation such as AND (\wedge), OR (\vee), and NOT (\neg) as a neural module based on logical regularizers. In this way, the recommendation problem can be formalized as estimating the probability that a future behavior v is true given user’s previous behaviors, such as $(a \vee b) \wedge \neg c \rightarrow v$ in the example. Based on the definition of material implication (\rightarrow)¹, this reduces to the T/F evaluation of $\neg((a \vee b) \wedge \neg c) \vee v$, which only includes basic logical operations. As a result, the logical expression can be identically transformed into a neural architecture based on the logical neural modules, as shown in Figure 1(c), which finally decides the T/F value of the expression. In this way, differentiable neural networks and symbolic reasoning are bridged in a shared architecture for optimization and inference.

To the best of our knowledge, this is one of the first work to bridge logical reasoning and embedding learning for personalized recommendation. The key contributions of our paper are as follows:

- We propose a novel neural logic recommendation framework to bridge symbolic logical reasoning and continuous embedding learning in a neural architecture.
- We introduce logical regularizers as constraints to regularize the behavior of neural networks.
- We dynamically construct the network structure according to the given logical expression, which enables logic priors to be added to the neural network structure.
- We conduct experiments on several real-world recommendation datasets to analyze the behavior of our framework.

In the following, we first review some related work in Section 2. In section 3, we introduce some background knowledge, and then explain the details of our framework in Section 4. We describe the experimental results on different datasets to verify the effectiveness of our approach in Section 5, and finally, Section 6 concludes the work with possible future directions to explore.

2 RELATED WORK

Collaborative Filtering (CF) has been important approaches to recommender systems. Due to its long-time research history and the

wide scope of literature, it is hardly possible to cover all CF algorithms, so we review some representative methods in this section, and a more comprehensive review can be seen in [8, 49].

Early approaches to CF consider the user-item rating matrix and conduct rating prediction with user-based [22, 37] or item-based [27, 41] collaborative filtering methods. With the development of dimension reduction methods, latent factor models such as matrix factorization are later widely adopted in recommender systems, such as singular value decomposition [25], non-negative matrix factorization [26], and probabilistic matrix factorization [33]. In these approaches, each user and item is learned as a latent vector to calculate the matching score of the user-item pairs.

Recently, the development of deep learning and neural network models has further extended collaborative filtering methods for recommendation. The relevant methods can be broadly classified into two sub-categories: similarity learning approach, and representation learning approach. The similarity learning approach adopts simple user/item representations (such as one-hot) and learns a complex matching function (such as a prediction network) to calculate user-item matching scores [13, 15, 18], while the representation learning approach learns rich user/item representations and adopts a simple matching function (e.g., inner product) for matching score calculation [44, 50, 51]. Another important research direction is learning to rank for recommendation, which learns the relative ordering of items instead of the absolute preference scores. The most representative method on this direction is perhaps Bayesian personalized ranking (BPR) [36], which is a pair-wise learning to rank method. It is also further generalized to take other information sources such as images [14].

Although many CF approaches have been successfully applied in recommendation tasks, existing methods mostly model recommendation as a perception task based on similarity matching, instead of a cognition task based on logical reasoning. However, users’ future behaviors may not be simply driven by the similarity with their previous behavior, but a concrete reasoning procedure about what to do next. Though few research work is seen in the recommendation literature, integrating logical reasoning and neural networks has been considered in other research contexts.

According to [3], connectionism in AI can date back to 1943 [31], which is arguably the first neural-symbolic system for Boolean logic. Since then, most neural-symbolic systems focused on representing, computing, and learning languages other than classical propositional logics [4, 6, 9, 17, 42]. Neural-symbolic learning has also been applied to knowledge extraction, representation and reasoning tasks [10, 43]. More recently, it has been shown that argumentation frameworks, abductive reasoning, and normative multi-agent systems can also be represented by neural symbolic frameworks [3, 11]. It is encouraging to see that a variety of forms of reasoning can be realized by the same framework that specializes in different ways [3]. Another approach to integrating machine learning and logical reasoning is Markov Logic Networks (MLN) [39], which combines probabilistic graphical models with first-order logic. It leverages domain knowledge and logic rules to learn graph structure and conduct inference, which has been shown to be effective for reasoning on knowledge graphs [35].

In a broader sense, the AI community has realized the importance of advancing AI from perception to cognition tasks [2], and

¹Material implication can be represented by basic logical operations: $x \rightarrow y \Leftrightarrow \neg x \vee y$

Table 1: Logical laws and the corresponding equation that each logical module should satisfy in our neural architecture. Some of the laws are guaranteed by adding an explicit logical regularizer into the training loss function, while others are guaranteed by randomly shuffling the logic variables during model training. $Sim(\cdot, \cdot)$ represents a similarity measure function.

	Law	Equation	Logical Regularizer r_i
NOT	Negation	$\neg T = F$	$r_1 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 + Sim(\text{NOT}(x), x)$
	Double Negation	$\neg(\neg x) = x$	$r_2 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{NOT}(\text{NOT}(x)), x)$
AND	Identity	$x \wedge T = x$	$r_3 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{AND}(x, T), x)$
	Annihilator	$x \wedge F = F$	$r_4 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{AND}(x, F), F)$
	Idempotence	$x \wedge x = x$	$r_5 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{AND}(x, x), x)$
	Complementation	$x \wedge \neg x = F$	$r_6 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{AND}(x, \text{NOT}(x)), F)$
OR	Identity	$x \vee F = x$	$r_7 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{OR}(x, F), x)$
	Annihilator	$x \vee T = T$	$r_8 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{OR}(x, T), T)$
	Idempotence	$x \vee x = x$	$r_9 = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{OR}(x, x), x)$
	Complementation	$x \vee \neg x = T$	$r_{10} = \frac{1}{ \mathcal{X} } \sum_{x \in \mathcal{X}} 1 - Sim(\text{OR}(x, \text{NOT}(x)), T)$
AND/OR	Associativity	$x \vee (y \vee z) = (x \vee y) \vee z$ $x \wedge (y \wedge z) = (x \wedge y) \wedge z$	Random Shuffling of Logic Variables
	commutativity	$x \vee y = y \vee x$ $x \wedge y = y \wedge x$	

researchers have been exploring the possibility of integrating deep learning with logical inference [7, 19, 29, 46]. However, as far as we know, there has been no work on neural logic reasoning approaches to recommendation systems. As a representative cognition task, we hope future intelligent recommendation systems can model higher-level cognitive intelligence for informed planning, reasoning, and decision making.

3 PRELIMINARIES

In this section, we briefly introduce some logical operators and basic laws of logic used in this work. We start from propositional logic, which includes three basic operations: AND (conjunction), OR (disjunction), and NOT (negation). Each variable, such as x , is called a *literal*. A flat operation over literals, such as $(x \wedge y)$, is called a *clause*, while operations over clauses, such as $(x \wedge y) \vee (a \wedge b \wedge c)$, is called an *expression*. Each logical operation should satisfy some basic laws in propositional logic, for example, the double negation law of NOT: $\neg(\neg x) = x$. We list some laws used in our work in Table 1. Another useful law not listed in the table is the De Morgan’s Law, which states that:

$$\begin{aligned} \neg(x \wedge y) &\Leftrightarrow \neg x \vee \neg y \\ \neg(x \vee y) &\Leftrightarrow \neg x \wedge \neg y \end{aligned} \quad (1)$$

Besides these operations and laws, we need another secondary logical operation $x \rightarrow y$, which is called material implication. This operation can be equivalently transformed using basic operations:

$$x \rightarrow y \Leftrightarrow \neg x \vee y \quad (2)$$

Propositional logic is a very useful language for symbolic reasoning. However, the symbolic nature of the language makes it difficult to be “learned” from data based on continuous optimization. To solve the problem, we borrow the idea of distributed representation learning [32], and propose a neural-symbolic framework for logical reasoning in a continuous space. Specifically, each literal x is learned as a vector embedding \mathbf{x} , and each logical operation (e.g., \wedge)

is learned as a neural module (e.g., $\mathbf{z} = \text{AND}(\mathbf{x}, \mathbf{y})$). As a result, an expression can be organized as a neural architecture (Figure 1(c)), which evaluates the T/F value of the expression in a latent space.

4 NEURAL LOGIC RECOMMENDATION

In this section, we present our Neural Logic Recommendation (NLR) framework, which encapsulates logical reasoning into a dynamic neural architecture. To achieve this goal, we first formalize recommendation into a logical reasoning problem. Then we introduce how to dynamically assemble the literals and logical operations into a neural network based on the given logical expression. After that, we present the logical regularizers, which regularize the behavior of neural modules to conduct certain logical operations. At the end of the section, we provide the learning algorithm to train our model.

4.1 Reasoning with Implicit Feedback

One fundamental goal of personalized recommendation is to predict a user’s future behavior given the user’s previous behaviors. We first consider users’ implicit feedbacks, i.e., we only know if a user has interacted with an item or not, but do not know if the user likes or dislikes the interacted item. Suppose a user’s interaction histories are items a, b and c (could be more), then the problem of recommending item v or not, reduces to the problem of deciding if the following implication statement is True or False:

$$a \wedge b \wedge c \rightarrow v \quad (3)$$

It should be noted that a logic variable in the expression (e.g., a) means the event that “user likes item a ”, while $\neg a$ means the event that “user dislikes item a ”. As a result, each variable represents a proposition, which enables us to conduct propositional inference over the variables. Intuitively, we use the simple conjunctive logical expression to decide if the user’s previous behaviors together would imply that the user likes a new item v . We will introduce how to learn the proposition embeddings in the next section.

Based on the definition of material implication (Eq.(2)), the above statement can be rewritten by only using the basic logical operations AND (\wedge), OR (\vee) and NOT (\neg), which is shown as follows:

$$\neg(a \wedge b \wedge c) \vee v \quad (4)$$

Based on De Morgan's Law (Eq.(1)), this can be further rewritten into a statement using only two basic operations \neg and \vee :

$$(\neg a \vee \neg b \vee \neg c) \vee v \quad (5)$$

One can see that the same logical statement (e.g., Eq.(3)) can be written into logically identical but literally different forms (Eq.(4) and Eq.(5)). As a result, a natural question to ask is which form should we use to build the neural architecture. As we will show in the experiments later, it is beneficial if the neural network only needs to train two logical operation modules (\neg and \vee) instead of all three modules (\neg , \wedge and \vee). As a result, we choose Eq.(5) as the basic logical form to introduce the NLR framework in this section, and we will experiment with different forms in later sections.

The above introduced formalization maps recommendation to a reasoning problem based on implicit feedbacks. However, the formalization is not personalized because user's personalized information is not encoded into the reasoning process. In the following, we further extend *items* to *events* for personalized recommendation.

Suppose we have a user $u_1 \in \mathcal{U}$ and a set of items $v_1, v_2, v_3, v_4 \in \mathcal{V}$, where \mathcal{U} and \mathcal{V} are user and item sets. Let the interaction history of u_1 be $\{v_1, v_2, v_3\}$, and then we represent these interactions $\langle u_1, v_1 \rangle, \langle u_1, v_2 \rangle, \langle u_1, v_3 \rangle$ as events e_1^1, e_2^1, e_3^1 , where e_j^i means user u_i interacted with item v_j before. As a result, the personalized recommendation problem becomes predicting if the previous events e_1^1, e_2^1, e_3^1 would imply a new event e_4^1 by deciding if the following statement is true:

$$e_1^1 \wedge e_2^1 \wedge e_3^1 \rightarrow e_4^1 \quad (6)$$

which, still, can be re-written using two logical operations:

$$(\neg e_1^1 \vee \neg e_2^1 \vee \neg e_3^1) \vee e_4^1 \quad (7)$$

where e_4^1 is for $\langle u_1, v_4 \rangle$. As we will illustrate later, the embedding of event e_j^i will consider both user u_i and item v_j , so that the model can make recommendations tailored to a particular user.

4.2 Reasoning with Explicit Feedback

Sometimes users will not only interact with items, but also will tell us if she likes or dislikes the item. Such implicit feedback signals are very informative for the recommendation task, as a result, it would be very beneficial if we can take the explicit feedbacks into the logical reasoning procedure.

Fortunately, it is very easy to extend the above formalization for reasoning with explicit feedback. In particular, we extend the definition of event e_j^i to describe the user attitude towards an interacted item. Specifically, we use e_j^i to represent that user u_i interacted with item v_j with positive feedback, and use $\neg e_j^i$ to show that user u_i interacted with item v_j with negative feedback. Still take the previous example, if user u_1 interacted with v_1, v_2, v_3 , and gave positive feedback on v_1, v_2 , while negative feedback on v_3 , then if or not to recommend v_4 depends on the T/F of the following statement:

$$e_1^1 \wedge e_2^1 \wedge \neg e_3^1 \rightarrow e_4^1 \quad (8)$$

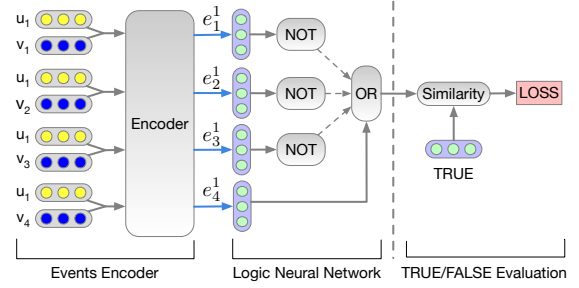


Figure 2: Implementation of our NLR framework. The gray boxes with yellow or blue circles represent latent embeddings, e.g., user embedding u_1 , and item embeddings $v_1 \sim v_4$; blue boxes with green circles represent event embeddings in the logical space, where the encoder is a multi-layer neural network that encodes user-item interactions; NOT and OR are neural logic modules; Dashed arrows mean that the order of encoded events are randomly shuffled each round.

which is equivalently written as:

$$(\neg e_1^1 \vee \neg e_2^1 \vee \neg e_3^1) \vee e_4^1 \quad (9)$$

Here, we preserve the double negation to make sure the negation modular will be adequately trained in the neural network, which will be explained with more detail in the following subsection.

4.3 Logical Modules

We have introduced how to formalize a recommendation task into a logical reasoning procedure. Now we introduce how to build the neural architecture based on the given logical expression. We use the implicit feedback case as a running example in this section, and later will generalize to explicit feedback cases.

Suppose a user u_1 interacted with v_1, v_2, v_3 , and our model needs to predict if item v_4 would be interacted by u_1 . We first use a multi-layer perceptron (MLP) network to encode the user and item interactions into event vectors. The following equation shows encoding a pair of user and item vectors into one event vector with a two-layer neural network:

$$\mathbf{e}_j^i = \mathbf{W}_2^T \phi(\mathbf{W}_1 \begin{bmatrix} \mathbf{u}_i \\ \mathbf{v}_j \end{bmatrix} + \mathbf{b}_1) + \mathbf{b}_2 \quad (10)$$

where $\mathbf{u}_i, \mathbf{v}_j \in \mathbb{R}^d$ are randomly initialized user and item vectors in d dimensional space; $\mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{n \times d}$ are the weight matrices; $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^n$ are bias terms; $\mathbf{e}_j^i \in \mathbb{R}^n$ represents the encoded event vector, and $\phi(\cdot)$ is the rectified linear unit (ReLU) activation function: $\phi(x) = \max(0, x)$.

With these event vectors, the next step is to construct the neural architecture to model the logical expression in Eq.(7), which is now shown as vector representations based on event embeddings:

$$(\neg \mathbf{e}_1^1 \vee \neg \mathbf{e}_2^1 \vee \neg \mathbf{e}_3^1) \vee \mathbf{e}_4^1 \quad (11)$$

Our goal is to calculate the above logical expression in a continuous representation space, and the space is characterized by two constant vectors \mathbf{T} and \mathbf{F} ($\mathbf{F} = \neg \mathbf{T}$). We expect that the final event vector of the expression would be close to \mathbf{T} if v_4 should be recommended, and \mathbf{F} otherwise. To achieve this goal, we represent each logical operation \wedge, \vee, \neg as a neural module $\text{AND}(\cdot, \cdot), \text{OR}(\cdot, \cdot)$, and $\text{NOT}(\cdot)$, where each neural module is also an MLP network

(Eq.(10)). For example, the AND(\cdot, \cdot) module takes two event embeddings \mathbf{e}_1 and \mathbf{e}_2 as input, and outputs a new event embedding, which represents the event that both \mathbf{e}_1 and \mathbf{e}_2 happens.

Based on the event embeddings and logical modules, we can then assemble a neural architecture for Eq.(11), as shown in Figure 2. By sending each input event embedding into the NOT(\cdot) module, we can calculate the negation of the events. After that, we combine both the negated events and the event embedding \mathbf{e}_4^1 into the OR(\cdot, \cdot) module, so as to generate the final event embedding of the whole logical expression. The OR(\cdot, \cdot) operation can only take two event embeddings at each time, to calculate the joint embedding of more than two events using OR, we first feed in two events, e.g. \mathbf{e}_j^i and $\mathbf{e}_{j'}^i$. The output vector $\mathbf{h}_{j,j'}^i$ is treated as the representation of event $\mathbf{e}_j^i \vee \mathbf{e}_{j'}^i$ in the logical representation space. The next event vector $\mathbf{e}_{j''}^i$ and the previous output $\mathbf{h}_{j,j'}^i$ will be sent to this OR neural module again to get the embedding of three disjunction events. We conduct this recurrently until the entire OR expression is calculated. The process, which is shown in Figure 2, can be represented by the following equations:

$$\begin{aligned} \neg \mathbf{e}_j^1 &= \text{NOT}(\mathbf{e}_j^1), \forall j \in \{1, 2, 3\} \\ \mathbf{Exp} &= \text{OR}(\neg \mathbf{e}_1^1, \neg \mathbf{e}_2^1, \neg \mathbf{e}_3^1, \mathbf{e}_4^1) \end{aligned} \quad (12)$$

The final output \mathbf{Exp} is the vector representation of the logical Expression in Eq.(11). To determine if the expression represents true or false, we examine if the final event embedding \mathbf{Exp} is close to the constant true vector (\mathbf{T}) in the logical space. The true vector is a randomly initialized anchor vector, which sets a basis for all of the high dimensional latent vectors in the logical space. It would never be updated after its initialization. If a vector represents true, then the vector should be close to this true vector, otherwise it should be far from the true vector. Any measure can be used to compare the \mathbf{Exp} and \mathbf{T} vectors. In this work, we use the most simple cosine measure:

$$P(\mathbf{Exp} = \mathbf{T}) = \frac{\mathbf{Exp} \cdot \mathbf{T}}{\|\mathbf{Exp}\| \|\mathbf{T}\|} \quad (13)$$

The above illustration is based on implicit feedback reasoning. To conduct reasoning based on explicit feedbacks such as Eq.(9), we only need to slightly modify the neural architecture in Figure 2. In particular, positive events are still fed into the neural network as before, but negative events will pass through an extra NOT(\cdot) module before feeding into the original architecture, as illustrated in Figure 3. In this design, we preserve the double negation structure instead of deleting both of them to make sure the negation module can be adequately optimized in the model training procedure.

Since the number of variables (i.e., interactions) and the numbers of negative feedbacks vary for different users, the length and structure of the logical expression would be different. As a result, the neural structure are different for different users, which will be dynamically assembled according to the input expression.

4.4 Logical Regularizer

We have defined three logical neural modules. However, by now they are just plain MLP neural networks. We need to guarantee that each logical module is really performing the expected logical

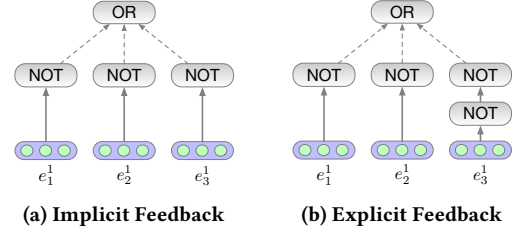


Figure 3: Reasoning over implicit (a) and explicit (b) feedbacks. The figure only shows the Logic Neural Network portion of Figure 2, other parts of the model are unchanged.

operation in the latent space. To achieve this goal, we add logical regularizer to the neural modules to constrain their behaviors. The regularizers and their corresponding laws are listed in Table 1.

Let \mathbf{x} represent an event embedding, which could be the original user-item interaction event, or any intermediate event during the logical neural network calculation, or the final event embedding of the logical expression. Let \mathcal{X} be the set of all event embeddings, and let $\text{Sim}(\cdot, \cdot)$ be a similarity function, which is cosine similarity in our implementation. As noted before, \mathbf{T} is the constant vector representing true, which is randomly initialized and never updated during model learning, and \mathbf{F} is the vector representing false, which is obtained through $\text{NOT}(\mathbf{T})$.

We take the double negation rule r_2 as an example to explain the basic idea of logical regularizers. For the NOT(\cdot) module to perform negation operation in the latent space, we require it to satisfy the double negation law:

$$\mathbf{x} \equiv \text{NOT}(\text{NOT}(\mathbf{x})) \quad (14)$$

which means that any event embedding, if negated for twice, should return to itself. To constrain the behavior of the negation module, we design a regularizer to maximize the cosine similarity between the output of $\text{NOT}(\text{NOT}(\mathbf{x}))$ and \mathbf{x} , which is equivalent to minimizing:

$$1 - \text{Sim}(\text{NOT}(\text{NOT}(\mathbf{x})), \mathbf{x}) \quad (15)$$

To make sure that the neural module can perform the same operation not only to the initial input event vectors, but also to all the intermediate hidden events and the final event, we apply the regularizer to all the event embeddings \mathcal{X} in the logical space, which gives us the final regularizer for the double negation law:

$$r_2 = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} 1 - \text{Sim}(\text{NOT}(\text{NOT}(\mathbf{x})), \mathbf{x}) \quad (16)$$

where $|\mathcal{X}|$ is the size of the entire event space. We would not introduce the details for all of the regularizers listed in Table 1, since they are designed in similar ways. The only difference is the regularizer for negation law r_1 , where we conduct one plus the similarity instead of one minus similarity, because we want to maximize the distance between $\text{NOT}(\mathbf{x})$ and \mathbf{x} , such as \mathbf{T} and \mathbf{F} . The final regularizer considering all laws is:

$$\mathcal{L}_{\text{logicReg}} = \sum_i r_i \quad (17)$$

The associative and commutative law can not be easily represented as regularizers. Instead, we randomly shuffle the order of the input events every iteration during the training process to make the learned AND and OR modules satisfy these two laws.

4.5 Learning Algorithm

In this work, we employ the pair-wise learning algorithm [36] for model training. Specifically, we conduct negative sampling on each given expression during the training process. Suppose we know that user u interacted with items $v_{i-1}, v_{i-2}, v_{i-3}$ right before item v_i . Then we can sample another item $v_j \in \mathcal{V}$ that the user did not interact with. Based on this, we build the structured neural network in terms of the following two expressions:

$$\begin{aligned} C_{ui} &= \neg e_{v_{i-1}}^u \vee \neg e_{v_{i-2}}^u \vee \neg e_{v_{i-3}}^u \vee e_{v_i}^u \\ C_{uj}^- &= \neg e_{v_{i-1}}^u \vee \neg e_{v_{i-2}}^u \vee \neg e_{v_{i-3}}^u \vee e_{v_j}^u \end{aligned} \quad (18)$$

where $C_{u,i}$ is the expression for our observed data, and $C_{u,j}^-$ is the expression for negative sampled data. Then, we have a pair of truth evaluation results:

$$\begin{aligned} p_{u,i}^+ &= P(\text{LNN}(\mathbf{e}_{v_{i-1}}^u, \mathbf{e}_{v_{i-2}}^u, \mathbf{e}_{v_{i-3}}^u, \mathbf{e}_{v_i}^u) = \mathbf{T}) \\ p_{u,j}^- &= P(\text{LNN}(\mathbf{e}_{v_{i-1}}^u, \mathbf{e}_{v_{i-2}}^u, \mathbf{e}_{v_{i-3}}^u, \mathbf{e}_{v_j}^u) = \mathbf{T}) \end{aligned} \quad (19)$$

where LNN is the logic neural network structure as in Figure 2. Then we calculate $\hat{p}_{uij} = p_{u,i}^+ - p_{u,j}^-$ to represent the difference between these two expressions, and apply an optimization algorithm to maximize this difference. In real implementation, we sample n negative items for each user-item pair. We use \mathcal{V}^- to represent the negative sample set, where $\mathcal{V}^- \subset \mathcal{V}$, $v_j \in \mathcal{V}^-$ and $v_i \notin \mathcal{V}^-$. The loss function can be written as:

$$\mathcal{L}_{lnn} = - \sum_{u \in \mathcal{U}} \sum_{v_i \in \mathcal{V}} \sum_{v_j \in \mathcal{V}^-} \ln \sigma(\hat{p}_{uij}) + \lambda_{\Theta} \|\Theta\|_2^2 \quad (20)$$

where Θ represents the parameter vector of the model, λ_{Θ} is the weight for Frobenius norm of the model parameters, and $\sigma(\cdot)$ is the logistic sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$. Maximizing \hat{p}_{uij} is equivalent to minimizing \mathcal{L}_{lnn} . The pseudo-code for calculating the logic neural network loss is given in Appendix A.2.

Now we can combine the logic regularizer together with our pairwise learning loss to get the final loss function:

$$\mathcal{L} = \mathcal{L}_{lnn} + \lambda_r \mathcal{L}_{logicReg} \quad (21)$$

where λ_r is the weight for logic regularizers. We apply the same weight to all the logic regularizers since they are equally important to regularize the logical behavior of the model. We apply back propagation [40] to optimize the parameters.

5 EXPERIMENTS

As the key motivation of this work is to develop a novel neural logic recommendation framework to harness the power of collaborative filtering and logical reasoning for personalized recommendation, we aim to answer the following research questions via experiments.

- **RQ1:** What is the performance of the NLR framework in terms of ranking tasks? Does it outperform state-of-the-art models? (Section 5.5)
- **RQ2:** How does the logic regularizer help to improve the performance? (Section 5.6)
- **RQ3:** Does the logical prior over the neural network structure help to improve the performance? (Section 5.7)
- **RQ4:** Can we model the recommendation problem with pure Boolean logic? (Section 5.8)

Table 2: Statistics of the datasets in our experiments.

Dataset	#Users	#Items	#Interaction	Density
ML100k	943	1,682	100,000	6.30%
Movies & TV	123,961	50,053	1,697,533	0.027%
Electronics	192,404	63,002	1,689,188	0.014%

5.1 Dataset

We experiment with three publicly available datasets. The statistics of the datasets are summarized in Table 2. The size of these three datasets ranges from 10K up to million level, and they cover movies as well as e-commerce recommendation scenarios.

ML100k [12]. A popular dataset maintained by Grouplens, which has been used by many researchers. It includes 100,000 movie ratings ranging from 1 to 5 from 943 users and 1,682 movies.

Amazon 5-core [30]. This is the Amazon e-commerce dataset, which includes user, item and rating information spanning from May 1996 to July 2014. Compared with ML100k, this is a relatively sparse dataset. It covers 24 different categories, and we take **Movies and TV** and **Electronics**, which are two million-scale datasets.

Following common practice, we consider 1-3 ratings as negative feedback, and 4-5 ratings as positive feedback. Details of the dataset pre-processing procedure are provided in the Appendix A.1.

5.2 Evaluation Metrics

To evaluate the top- K recommendation performance, we use standard metrics such as Normalized Discounted Cumulative Gain at rank K (NDCG@ K) and Hit Ratio at rank K (HR@ K). In our experiments, the result of all metrics are averaged over all users.

5.3 Baselines

To examine the effectiveness of our proposed neural logic recommendation model, we compare with two traditional shallow methods (Biased-MF and SVD++), two deep models (DMF and NeuMF), as well as two session-based models (GRU4Rec and STAMP).

- **Biased-MF** [25]: This is a well-known recommendation algorithm by adding user, item and global bias terms into matrix factorization.
- **SVD++** [23]: It is also a matrix factorization based method, which extends Singular Value Decomposition (SVD) by considering user history interactions when modeling the users.
- **DMF** [47]: Deep Matrix Factorization is a deep model for recommendation, which uses multiple non-linear layers to process the raw user-item interaction matrix.
- **NeuMF** [15]: A neural network-based collaborative filtering algorithm, which employs a non-linear prediction network for user and item matching.
- **GRU4Rec** [16]: A session-based recommendation model, which uses RNN to capture the sequential dependencies in user’s historical interactions.
- **STAMP** [28]: A session-based recommendation model based on attention mechanism, which can capture user’s long-term and short-term preferences.

We train all of the models with pair-wise methods based on negative sampling. For each user-item pair in the validation and test set, we randomly sample 100 negative items for ranking evaluation.

Table 3: Results of recommendation performance on three datasets with metrics NDCG (N) and Hit Ratio (HR). The strongest baselines are marked with underlines. We use bold font to mark the best results from all models for each comparison. We use * to indicate that the performance is significantly better than the best baseline based on paired t -test at level of 0.05.

	ML100k				Movies and TV				Electronics			
	N@5	N@10	HR@5	HR@10	N@5	N@10	HR@5	HR@10	N@5	N@10	HR@5	HR@10
Biased-MF	0.3024	0.3659	0.4501	0.6486	0.3962	0.4392	0.5346	0.6676	0.3092	0.3472	0.4179	0.5354
SVD++	0.3087	0.3685	0.4586	0.6433	0.3918	0.4335	0.5224	0.6512	0.2775	0.3172	0.3848	0.5077
DMF	0.3023	0.3661	0.4480	0.6450	0.4006	0.4455	<u>0.5455</u>	<u>0.6843</u>	0.2775	0.3143	0.3783	0.4922
NeuMF	0.3002	0.3592	0.4490	0.6316	0.3791	0.4211	0.5134	0.6429	0.3026	0.3358	0.4031	0.5123
GRU4Rec	<u>0.3564</u>	<u>0.4122</u>	0.5134	<u>0.6856</u>	<u>0.4038</u>	<u>0.4459</u>	0.5287	0.6688	<u>0.3154</u>	<u>0.3551</u>	<u>0.4284</u>	<u>0.5511</u>
STAMP	0.3560	0.4070	<u>0.5159</u>	0.6730	0.3935	0.4366	0.5246	0.6577	0.3095	0.3489	0.4196	0.5430
NLR-I	0.3697	0.4219	0.5265	0.6890	0.4152	0.4550	0.5479	0.6709	0.3226	0.3604	0.4331	0.5500
NLR-E_{mod}	0.3671	0.4219	0.5180	0.6890	0.4126	0.4535	0.5444	0.6705	0.3272	0.3649	0.4377	0.5544
NLR-E	0.3760*	0.4240*	0.5456*	0.6943*	0.4255*	0.4670*	0.5611*	0.6891	0.3499*	0.3878*	0.4639*	0.5812*
Improvement	5.50%	2.86%	5.76%	1.27%	5.37%	4.73%	2.86%	0.70%	10.94%	9.21%	8.29%	5.46%

5.4 Experimental Settings

We use the same train, validation and test datasets for our model and baseline methods in experiments. All the model’s parameters are tuned on the validation set based on NDCG@5. For fair comparison, all models, except for DMF, are set with an embedding size of 64 and optimized using mini-batch Adam [21] with a batch size of 128. For our NLR model, the number of layers for all MLP modules are set to 2. We apply ReLU non-linear activation function between layers. The learning rate is 0.001, the weight of ℓ_2 regularizer is 0.0001 for ML100k dataset and 0.00001 for Amazon datasets. The logical regularizer weight we use to report the results is 0.1. The implementation details of baseline models, as well as the hardware and software settings are provided in Appendix A.3.

5.5 Performance of NLR framework (RQ1)

To evaluate the performance of our NLR framework, we report the results, including NDCG and hit ratio (HR), on all the datasets in Table 3. We use NLR-I to represent our model with implicit feedbacks as the input, while NLR-E means our model with explicit feedbacks (negative/positive). It shows that our model consistently outperforms all the comparative methods under the given metrics. We give the relative improvement of our model against the strongest baseline results in the last row of the table. From the comparison, we notice that the shallow matrix factorization based algorithms – Biased-MF and SVD++ – have similar performance on most of the measures. They are powerful and efficient by training with pair-wise loss for ranking tasks. Deep neural models such as DMF and NeuMF also reach a reasonable performance on some tasks.

Since our model make use of user historical interactions, it is meaningful to compare our model with session-based recommendation methods. For both GRU4Rec and STAMP, they use implicit feedbacks for training the model and make use of sequential information. For fairness concern, we compare our implicit model NLR-I with the two baselines. The reported data shows that our model is able to compete with the session-based models. What we need to notice is that our model shuffles the input variables every epoch. That means we actually did not use the order information.

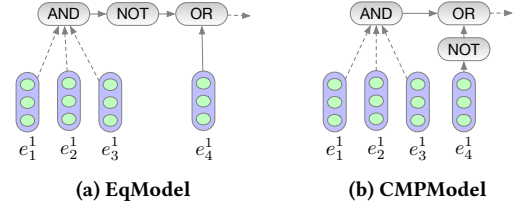


Figure 4: Comparison between the network structure that follows the logical prior (a) and violates the logical prior (b).

However, our logic neural network can still beat the other two session-based models significantly in most of the measures even using less information.

5.6 The Effect of Logical Regularizers (RQ2)

In this section, we answer the question that if the logic regularizers would help to improve the performance. We tune the logical regularizer weight λ_r from 10^{-5} to 1. The corresponding NDCG@10 and HR@10 are given in Figure 5(a)-(c). We can see that the best performance would be reached by assigning 0.1 to the weight of logic regularizers. This result shows that it is useful to apply logical constraints to the neural networks to improve the recommendation performance. However, the constraints need to be carefully adjusted. If the constraints too weak or too strong, the performance of the recommendation model would be negatively influenced.

In Table 3, **NLR-E_{mod}** shows the performance of our explicit feedback model without using logical regularizers (i.e., $\lambda_r = 0$). By comparing **NLR-E_{mod}** and **NLR-E**, we can see that the recommendation performance improves by using logical regularizers. We also conduct paired t -test between the two models, and the improvements are significant at the level of 0.05 except for NDCG@10 on ML100k dataset. This result shows that logical regularizers do help to improve the recommendation performance.

5.7 The Effect of Logic Prior over Structure (RQ3)

Our logical neural network structure is characterized by two important features: modularity and logical regularization. Modularity

Table 4: Ranking performance under different logical structures. "* indicates the significance at the level of 0.05**

	ML100k				Movies and TV				Electronics			
	N@5	N@10	HR@5	HR@10	N@5	N@10	HR@5	HR@10	N@5	N@10	HR@5	HR@10
GRU4Rec	0.3564	0.4122	0.5134	0.6856	0.4038	0.4459	0.5287	0.6688	0.3154	0.3551	0.4284	0.5511
¹ EqModel	0.3664	0.4224	0.5318	0.7070	0.4105	0.4521	0.5429	0.6686	0.3249	0.3626	0.4355	0.5518
² CMPModel	0.3551	0.4144	0.5106	0.6932	0.4100	0.4506	0.5417	0.6670	0.3165	0.3541	0.4252	0.5416
³ NLR-E	0.3760	0.4240	0.5456	0.6943	0.4255	0.4670	0.5611	0.6891	0.3499	0.3878	0.4639	0.5812
p -value ^{1,3}	0.0825	0.0606	0.1073	0.0547	0.0156*	0.0230*	0.0212*	0.0197*	0.0015*	0.0021*	0.0010*	0.0009*
p -value ^{2,3}	0.0099*	0.0250*	0.0258*	0.4668	0.0108*	0.0103*	0.0057*	0.0048*	0.0022*	0.0019*	0.0023*	0.0018*

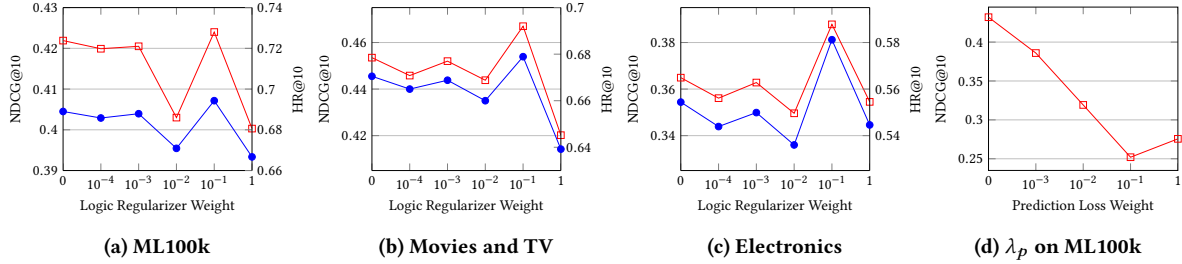


Figure 5: (a)-(c): NDCG@10 (Red) and HR@10 (Blue) according to the increase of logic regularizer weight λ_r . (d): NDCG@10 according to the increase of prediction loss weight λ_p on ML100k.

means that we dynamically assemble the neural structure according to the logical expression. Each network module is responsible for a specific operation, and the entire network structure varies in terms of the logical expressions. As a result, different user and item interaction histories would result in different network structures during both training and testing, and this is a big difference between our framework and traditional deep learning models whose network structures are static.

As we mentioned in Section 4.1, the same logical statement (e.g., Eq.(3)) can be written into logically identical but literally different expressions (Eq.(4) and (5)), and different expressions will result in different network structures in our model. In the previous modeling and experiments, we used the two operation (\neg , \vee) expression to build the network structure (Figure 2). However, it would be interesting to see what happens if we use other logically identical but literally different expressions to build the network structure.

To answer the question, we explore two alternative network structures. One is a logically equivalent model (**EqModel**), i.e., we still use the logical expression $\mathbf{e}_1^1 \wedge \mathbf{e}_2^1 \wedge \mathbf{e}_3^1 \rightarrow \mathbf{e}_4^1$ to model our task. However, it is represented as $\neg(\mathbf{e}_1^1 \wedge \mathbf{e}_2^1 \wedge \mathbf{e}_3^1) \vee \mathbf{e}_4^1$, instead of $(\neg\mathbf{e}_1^1 \vee \neg\mathbf{e}_2^1 \vee \neg\mathbf{e}_3^1) \vee \mathbf{e}_4^1$ that we used before (Eq.(11)). Figure 4(a) shows the network structure of the logically equivalent model. One can see that although this model is logically equivalent to NLR, the neural structures are different. Besides, the original network only needs to train two modules (\neg , \vee), while the new network needs to train all three modules (\neg , \wedge , \vee).

Another model is a logically nonequivalent model, noted as a comparative model (**CMPModel**). We apply the logic expression $\mathbf{e}_4^1 \rightarrow \mathbf{e}_1^1 \wedge \mathbf{e}_2^1 \wedge \mathbf{e}_3^1$, which is equivalent to $\neg\mathbf{e}_4^1 \vee (\mathbf{e}_1^1 \wedge \mathbf{e}_2^1 \wedge \mathbf{e}_3^1)$, to build the neural structure. Figure 4(b) shows the network structure of the CMPModel. One can see that the model attempts to use future events to predict the previous events, which violates our logical intuition about the recommendation task.

In Table 4, we give the p -value of paired t -test between the EqModel and the original NLR-E model, as well as between the CMPModel and NLR-E, based on 5-round random experiments. For each model, the reported results are the best score among the repeated experiments. We copy the results of GRU4Rec, which is overall the best baseline model in Table 3, for easy reference.

We have two key observations from the results in Table 4. First, we see that both NLR-E and EqModel consistently outperform the GRU4Rec baseline, while the CMPModel is generally not better than the baseline. Besides, the CMPModel is significantly worse than the original NLR-E model (shown by p -value^{2,3} in the table). This observation shows that a correct and reasonable logical structure is important to the performance of the model.

Another observation comes by comparing NLR-E and EqModel. By looking at the p -value between the two models (i.e., p -value^{1,3}), we see that the two models are comparable on ML100k dataset (i.e., NLR-E is not significantly better than EqModel), while NLR-E is indeed significantly better than EqModel on the two Amazon datasets. The underlying reason may arise from two factors – the complexity of model, and the sufficiency of data. As shown in Table 2, the MovieLens dataset is 2 magnitudes denser than the Amazon datasets. Since NLR-E and EqModel are logically equivalent, they achieve comparable performance when the training data is sufficient. However, NLR-E only needs to train two neural models (\neg , \vee), while EqModel has to train three modules (\neg , \wedge , \vee), thus EqModel has a higher model complexity than NLR-E. As a result, NLR-E achieves better performance when the training data is sparse.

From this experiment, we can learn that it is important to use a reasonable logical prior to construct the model for a specific task. In addition, when there are multiple logically equivalent structures, we tend to use a simpler network structure (i.e., fewer modules) instead of a complex one.

5.8 Boolean Logic Modeling (RQ4)

In our model, we did not apply constraints to the event embeddings, as a result, they can be learned as flexible vectors in the logical space. In this experiment, we apply a constraint that any event embedding can only be either T or F vector, to explore if the recommendation task can be modeled based on Boolean logic. To do so, we assume that the encoder network is a prediction network, which predicts if a user would give positive feedback to an item.

We first concatenate the user u and item v 's embeddings, and feed it into the encoder network. Before further feeding the encoded event embedding \mathbf{e}_v^u into the logic neural network, we calculate the mean square error (MSE) between this event embedding and the T or F vector, where the MSE between two vectors \mathbf{x} and \mathbf{y} is defined as $\text{MSE}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \|\mathbf{x} - \mathbf{y}\|_2^2$ (n is the dimension of the vector).

If the user has a positive feedback on the item, we minimize the MSE between the event vector and the T vector, otherwise with the F vector. The prediction loss function is written as:

$$\mathcal{L}_{\text{predict}} = \text{MSE}(\mathbf{e}_v^u, \mathbf{G}) \quad (22)$$

where \mathbf{G} represents the ground-truth vector, which is T or F, depending on the user likes or dislikes the item. Then, we add this loss to the model loss function in Eq.(21) to achieve the following new loss function:

$$\mathcal{L}_{\text{Boolean}} = \mathcal{L}_{\text{Inn}} + \lambda_r \mathcal{L}_{\text{logicReg}} + \lambda_p \mathcal{L}_{\text{predict}} \quad (23)$$

where λ_p is the weight of the prediction loss. By adding this loss, the model tries to polarize the event embeddings to either T or F. The logic network would then conducts reasoning in an approximate binary space when the prediction loss is minimized to a very small number (0.0001 in our experiment). We present the results of NDCG@10 on ML100k with $\lambda_p \in [0.001, 0.01, 0.1, 1]$ in Figure 5(d).

From the results, we can see that the ranking performance heavily drops with the increase of the prediction loss weight. A large λ_p would limit the expressiveness power of the latent embeddings, which further limits the ability of logic neural network to properly model the recommendation task. This experiment shows that it is important to blend the power of embedding learning into logical reasoning for accuracy decision making in a logical space.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel Neural Logic Recommendation (NLR) framework, which models recommendation as a reasoning task by integrating logical structures and neural networks for collaborative filtering. Experiments show that our method provides significant improvement over representative baselines on recommendation tasks. We conducted further experiments to explore the behavior of our model under different settings, so as to understand why the model achieves good performance. Results show that appropriate logical regularization is helpful to the recommendation performance. Though not a key focus of this work, our framework may bring better interpretability to recommender systems, since the decision making procedure is more transparent. We will explore this interpretability perspective in the future.

APPENDIX

A.1 Data Preprocessing

We transform the rating information, which come with 1 to 5 ratings, to 0 and 1, which represents the negative or positive explicit feedbacks. Ratings equal to or higher than 4 are mapped to 1 (positive), while those equal to or lower than 3 are transformed to 0 (negative). Then we sort the datasets by time. For each user and item pair in the dataset, we select its corresponding most recent n history interactions to build the logic expression. Here we set the length of history to 5, which means each user item pair comes with 5 history interactions. For those items from the earliest 5 interactions of each user, we put them into the training data. Users with less than 5 interactions are put in training dataset. We conduct leave-one-out operation to create the validation set and test set, which means that the last two interactions of each user are assigned to validation set and test set, respectively. Test sets are preferred if there remains only one expression for the user. For the models with implicit feedback as input, we simply ignore the rating information in the experiments, since the entire historical interactions are used. There is no difference between implicit feedback dataset and explicit feedback dataset except for the rating information.

A.2 Pseudo Code to Calculate the Logic Neural Network Loss

Algorithm 1 Logic Neural Network Loss

Input: Training User set U , item set V , model parameters Θ , L_2 regularizer weight λ_Θ

Output: Model loss

```

1: procedure CALCLNNLOSS
2:   Initialize  $\Theta$ 
3:    $Loss \leftarrow 0$ 
4:   for  $u \in U, v_i \in V$  do
5:      $V_{hist} \leftarrow \text{drawHistory}(v_i)$   $\triangleright$  obtain history of  $\langle u, v_i \rangle$ 
6:      $V^- \leftarrow \text{Sample}(v_i, V, n)$   $\triangleright$  get  $n$  negative samples of  $v_i$ 
7:      $E \leftarrow \text{ENCODE}(V_{hist})$   $\triangleright$  obtain event vectors
8:      $\mathbf{e}_i \leftarrow \text{ENCODE}(v_i)$ 
9:      $p_{ui}^+ \leftarrow P(\text{LNN}(E, \mathbf{e}_i) = \text{True})$ 
10:    for  $v_j \in V^-$  do
11:       $\mathbf{e}_j \leftarrow \text{ENCODE}(v_j)$ 
12:       $p_{uj}^- \leftarrow P(\text{LNN}(E, \mathbf{e}_j) = \text{True})$ 
13:       $\hat{p}_{uij} \leftarrow p_{ui}^+ - p_{uj}^-$ 
14:       $Loss \leftarrow Loss + \ln \sigma(\hat{p}_{uij})$ 
15:    $Loss \leftarrow -Loss + \lambda_\Theta \|\Theta\|_2^2$ 
16:   return  $Loss$ 

```

A.3 Additional Experimental Settings

For DMF, we implemented the model with two two-layer neural networks for modeling users and items respectively. Since the author claimed that the increment of vector size would help to improve the performance, we tune the hidden vector size to 128 to reach the best performance on all the tasks; For NeuMF, we use a 3-layer MLP with the sizes 32, 16, 8 as mentioned by the author. The final output layer has only 1 layer with dimension 64; For both the GRU4Rec

and STAMP, we use 5 as the history length, which is the same as our model used for the experiments. The size of hidden state vectors of both models are 64; For the CMPModel and EqModel, we apply the same parameters as our NLR model to guarantee that the differences in the reported results are resulted from the variation of the structure of the neural networks.

For training process, early-stopping is conducted according to the performance on the validation set. We run the experiments with five different random seeds and report the best results of each model. The p-value of paired *t*-test are calculated based on these 5 round random experiments. We use both the ℓ_2 -regularization and dropout to prevent overfitting. The weight of ℓ_2 -regularization λ_Θ is set between 1×10^{-6} to 1×10^{-4} and dropout ratio is set to 0.2. Vector sizes of the variables and the user/item vectors are 64. The maximum training epoch is set to 100.

All the neural network parameters for DMF, NeuMF, GRU4Rec, STAMP, NLR, CMPModel and EqModel are initialized from a normal distribution with 0 mean and 0.01 standard deviation. Our framework is implemented with PyTorch [34] version 1.4 on an NVIDIA Geforce 2080Ti GPU. The operating system is Ubuntu 16.04 LTS.

REFERENCES

- [1] Gediminas Adomavicius, Bamshad Mobasher, Francesco Ricci, and Alex Tuzhilin. 2011. Context-Aware Recommender Systems. *Recommender systems handbook* (2011), 217–253.
- [2] Yoshua Bengio. 2019. From System 1 Deep Learning to System 2 Deep Learning. In *NeurIPS'2019*.
- [3] Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902* (2017).
- [4] Antony Browne and Ron Sun. 2001. Connectionist inference models. *Neural Networks* 14, 10 (2001), 1331–1355.
- [5] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *WSDM*. 108–116.
- [6] Ian Cloete and Jacek M Zurada. 2000. *Knowledge-based neurocomputing*. MIT press.
- [7] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural logic machines. In *Proceedings of the Seventh International Conference on Learning Representations*.
- [8] Michael D Ekstrand, John T Riedl, Joseph A Konstan, et al. 2011. Collaborative filtering recommender systems. *Foundations and Trends® in Human-Computer Interaction* 4, 2 (2011), 81–173.
- [9] Artur S d'Avila Garcez, Luis C Lamb, and Dov M Gabbay. 2008. *Neural-symbolic cognitive reasoning*. Springer Science & Business Media.
- [10] Artur S Avila Garcez and Gerson Zaverucha. 1999. The connectionist inductive learning and logic programming system. *Applied Intelligence* 11, 1 (1999), 59–77.
- [11] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. 2012. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.
- [12] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm TIST* 5, 4 (2016), 19.
- [13] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *RecSys*. 161–169.
- [14] Ruining He and Julian McAuley. 2016. VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. In *AAAI*.
- [15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.
- [16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and D Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.
- [17] Steffen Hölldobler, Yvonne Kalinke, Fg Wissensverarbeitung Ki, et al. 1991. Towards a new massively parallel computational model for logic programming. In *In ECAI'94 workshop on Combining Symbolic and Connectionist Processing*. Citeseer.
- [18] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *In WWW*. 193–201.
- [19] Zhengyao Jiang and Shan Luo. 2019. Neural Logic Reinforcement Learning. *Proceedings of the 36th International Conference on Machine Learning* (2019).
- [20] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering. *RecSys* (2010), 79–86.
- [21] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
- [22] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. 1997. GroupLens: applying collaborative filtering to Usenet news. *Commun. ACM* 40, 3 (1997), 77–87.
- [23] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*. ACM, 426–434.
- [24] Yehuda Koren. 2009. Collaborative Filtering with Temporal Dynamics. *KDD* (2009), 447–455.
- [25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.
- [26] Daniel D Lee and H Sebastian Seung. 2001. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*. 556–562.
- [27] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* 7, 1 (Jan. 2003), 76–80. <https://doi.org/10.1109/MIC.2003.1167344>
- [28] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, and Haibin Zhang. 2018. STAMP: short-term attention/memory priority model for session-based recommendation. In *SIGKDD*. ACM, 1831–1839.
- [29] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepprolog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*. 3749–3759.
- [30] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. ACM.
- [31] WS McCulloch and Walter Pitts. 1943. A Logical Calculus of the Idea Immanent in Neural Nets. *Bulletin of Mathematical Biophysics* 5 (1943), 115–133.
- [32] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NeurIPS*. 3111–3119.
- [33] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.
- [34] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. (2017).
- [35] Meng Qu and Jian Tang. 2019. Probabilistic logic neural networks for reasoning. In *Advances in Neural Information Processing Systems*. 7710–7720.
- [36] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the 25th conference on uncertainty in artificial intelligence*. AUAI Press, 452–461.
- [37] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. 1994. GroupLens: an open architecture for collaborative filtering of netnews. In *CSCW*. ACM, 175–186.
- [38] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to Recommender Systems Handbook. *Springer US* (2011).
- [39] Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine learning* 62, 1-2 (2006), 107–136.
- [40] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.
- [41] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*. ACM, 285–295.
- [42] Lokendra Shastri. 2007. SHRUTI: A neurally motivated architecture for rapid, scalable inference. In *Perspectives of Neural-Symbolic Integration*. Springer.
- [43] Geoffrey G Towell and Jude W Shavlik. 1994. Knowledge-based artificial neural networks. *Artificial intelligence* 70, 1-2 (1994), 119–165.
- [44] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. 2015. Collaborative deep learning for recommender systems. In *KDD*. ACM, 1235–1244.
- [45] Hongwei Wang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. DKN: Deep knowledge-aware network for news recommendation. In *WWW*. 1835–1844.
- [46] Wenya Wang and Sinno Jialin Pan. 2020. Integrating Deep Learning with Logic Fusion for Information Extraction. *AAAI*.
- [47] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems.. In *IJCAI*. 3203–3209.
- [48] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *KDD*.
- [49] Shuai Zhang, Lina Yao, and Aixun Sun. 2019. Deep learning based recommender system: A survey and new perspectives. *Comput. Surveys* 52, 1 (2019), 1–38.
- [50] Yongfeng Zhang, Qingyao Ai, Xu Chen, and W Bruce Croft. 2017. Joint representation learning for top-n recommendation with heterogeneous information sources. In *CIKM*. ACM, 1449–1458.
- [51] Lei Zheng, Vahid Noroozi, and Philip S. Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *WSDM*.