Neural Collaborative Reasoning

Hanxiong Chen
Rutgers University
New Brunswick, NJ, US
hanxiong.chen@rutgers.edu

Shaoyun Shi
Tsinghua University
Beijing, China
shisy17@mails.tsinghua.edu.cn

Yunqi Li
Rutgers University
New Brunswick, NJ, US
yunqi.li@rutgers.edu

Yongfeng Zhang
Rutgers University
New Brunswick, NJ, US
yongfeng.zhang@rutgers.edu

ABSTRACT

Existing Collaborative Filtering (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 associative relevance patterns in data, so that a user embedding can be matched with relevant item embeddings using designed or learned similarity functions. However, 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 cognitive reasoning in data.

In this paper, we propose to advance Collaborative Filtering (CF) to Collaborative Reasoning (CR), which means that each user knows part of the reasoning space, and they collaborate for reasoning in the space to estimate preferences for each other. Technically, we propose a Neural Collaborative Reasoning (NCR) framework to bridge learning and reasoning. Specifically, we integrate the power of representation learning and logical reasoning, where representations capture similarity patterns in data from perceptual perspectives, and 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 reasoning architecture, which learns logical operations such as AND ($\wedge$), OR ($\vee$) and NOT ($\neg$) as neural modules for implication reasoning ($\rightarrow$). In this way, logical expressions can be equivalently organized as neural networks, so that logical reasoning and prediction can be conducted in a continuous space. Experiments on real-world datasets verified the advantages of our framework compared with both shallow, deep and reasoning models.

KEYWORDS

Collaborative Filtering; Collaborative Reasoning; Recommender Systems; Cognitive Reasoning; Cognitive Intelligence

ACM Reference Format:
Hanxiong Chen, Shaoyun Shi, Yunqi Li, and Yongfeng Zhang. 2021. Neural Collaborative Reasoning. In *Proceedings of the Web Conference 2021 (WWW '21), April 19–23, 2021, Ljubljana, Slovenia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3442381.3449973

1 INTRODUCTION

Collaborative Filtering (CF) is an important approach to recommender systems [12, 45]. 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 [44] and Item-based CF [49], consider the row and column vectors in the original 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 [30], 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.

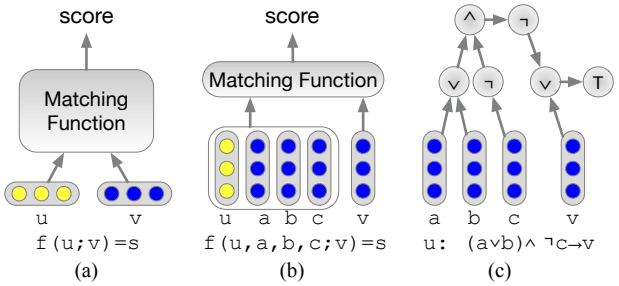


Figure 1: An overview of the fundamental structure of different collaborative filtering algorithms.

Researchers have further explored CF 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, 25, 29], and heterogeneous information sources can be used to enrich the embeddings [54], such as text [58], image [19], and knowledge graphs [2, 52]. We can also explicitly consider a user's behavior history to learn better embeddings (Figure 1(b)), such as in sequential recommendation [6, 21, 24, 32]. Another approach is to learn better matching functions. For example, using vector translation instead of inner product for matching [18], or learning the matching function based on metric learning [22] and neural networks [7, 20, 51]. However, whether complex neural matching functions are better than simple matching functions is controversial [8, 9, 14, 43].

Similarity matching-based CF methods have been adopted 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

This paper is published under the Creative Commons Attribution 4.0 International (CC-BY 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.
WWW '21, April 19–23, 2021, Ljubljana, Slovenia
© 2021 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY 4.0 License.
ACM ISBN 978-1-4503-8312-7/21/04.
https://doi.org/10.1145/3442381.3449973

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. In a broader sense, the community has realized the importance of advancing AI from perception to cognition tasks [4, 34, 50]. As a representative cognitive reasoning task, we hope an intelligent recommendation system would be able to conduct logical reasoning over the data to predict user's future behaviors for personalized recommendation.

To achieve this goal, we propose Neural Collaborative Reasoning (NCR), which defines the recommendation problem as a differentiable (i.e., neural) logical reasoning problem based on the wisdom of the crowd (i.e., collaborative). Specifically, each user's behavior record is considered as a Horn clause, 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 contributes to 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 each user. More specially, we propose a neural logic reasoning architecture, which integrates the power of embedding learning and logical reasoning in a shared model. The model learns basic logical operations such as AND ($\wedge$), OR ($\vee$), and NOT ($\neg$) as neural modules based on logic regularization. As a result, the recommendation problem can be formalized as estimating the probability that a Horn clause is True (T), 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. Finally, the Horn clause can be identically transformed into a neural architecture using the logical neural modules, as shown in Figure 1(c), which 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.

The key contributions of the paper are as follows:

- We propose a novel neural collaborative reasoning framework to bridge symbolic logical reasoning and continuous embedding learning for recommendation.
- We propose to adopt Horn clause for implication reasoning in recommendation, which naturally fits with the prediction nature of recommendation tasks.
- We propose a neural logic reasoning architecture, which dynamically constructs the network structure according to the given logical expression, and enables logic priors to be added to the neural network.
- We conduct experiments on several real-world recommendation datasets to analyze the behavior of our framework.

The following part of the paper will include related work (section 2), preliminaries (section 3), our framework (section 4), experiments (section 5), as well as conclusions and future work (section 6).

2 RELATED WORK

Collaborative Filtering (CF) has been an important approach 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 [12, 53, 55].

Early approaches to CF consider the user-item rating matrix and conduct rating prediction with user-based [27, 44] or item-based [33, 49] 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 [30], non-negative matrix factorization [31], and probabilistic matrix factorization [38]. 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 [7, 18, 20, 22, 51], while the representation learning approach learns rich user/item representations and adopts a simple matching function (e.g., inner product) for efficient matching score calculation [2, 35, 52, 54, 58]. However, there exist debates over whether complex matching functions are better than simple functions [8, 9, 14, 43]. Another important direction is learning to rank for recommendation, which learns the relative ordering of items instead of the absolute preference scores. A representative method is Bayesian personalized ranking (BPR) [42], which is a pair-wise learning to rank method. It is also further generalized to take other information sources such as images [19].

Although many CF approaches have been developed for recommendation tasks, existing methods mostly model recommendation as a perception task based on similarity matching instead of a cognition task based on cognitive/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. Integrating logical reasoning and neural networks has been considered in several research contexts. According to [5], connectionism in AI can date back to 1943 [36], which is arguably the first neural-symbolic system for Boolean logic. More recently, it is shown that argumentation frameworks, abductive reasoning, and normative multi-agent systems can also be represented by neural symbolic frameworks [5, 10, 11, 15, 23]. Another approach to integrating machine learning and logical reasoning is Markov logic networks [41, 46, 56], which combines probabilistic graphical models with first-order logic. It leverages domain knowledge and logic rules to learn graph structure for inference, which is effective for reasoning on knowledge graphs [41].

The most related work to ours is neural logic reasoning [50], which adopts neural logic modules for solving logical equations and (non-personalized) recommendation. However, our work is different on three aspects: we build neural models for logical reasoning based on the implication form of Horn clauses, which is a more natural way of making logical predictions in recommendation tasks; we develop a personalized recommendation model while the model in [50] can only conduct non-personalized recommendation;

[1]Material implication ($\rightarrow$) can be represented by basic operations: $x \rightarrow y \Leftrightarrow \neg x \vee y$

we explore if and how different neural logic structures influence the prediction performance of the neural logic models.

More broadly, researchers have realized the importance of advancing AI from perception to cognition tasks [4, 34]. 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 logical laws 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:

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

Different from neural logic reasoning [50], besides these operations and laws, we introduce another secondary logical operation $x \rightarrow y$ called material implication, which is fundamental to logic reasoning with Horn clauses, and as we will show later, it naturally fits into the prediction task of personalized recommendation. This operation can be equivalently transformed using basic operations:

$$x \rightarrow y \Leftrightarrow \neg x \vee y \tag{2}$$

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 [37], and propose a neural-symbolic framework for logical reasoning in a continuous space. Similar to [50], 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 (toy example in Figure 1(c), and more details later), which evaluates the T/F value of the expression in a latent space.

4 NEURAL COLLABORATIVE REASONING

We present our Neural Collaborative Reasoning (NCR) framework in this section, which encapsulates logical reasoning into a dynamic neural architecture. We first formalize recommendation into a logical reasoning problem in Horn clause form. Then we introduce how to dynamically assemble the literals and logical operations into a neural network for recommendation. After that, we present the logical regularizers, which regularize the behavior of neural modules to conduct the expected logical operation. At the end of the section, we provide the learning algorithm for model training.

4.1 Reasoning with Implicit Feedback

One fundamental goal of personalized recommendation is to predict a user's future behavior given the existing behaviors. We first consider users' implicit feedbacks, i.e., we only know if a user has interacted with an item, but do not know if the user likes or dislikes the interacted item. Suppose a user u's interaction history contains r items $v_1, v_2, \cdots, v_r$, and we want to predict if an item v_x is to be recommended for the user. We need to define a function in first-order logic to encode the interactions into a logic space. Then the problem of recommending item v_x or not reduces to the problem of deciding if the following Horn clause is True or False:

$$I(u, v_1) \wedge I(u, v_2) \wedge \cdots \wedge I(u, v_r) \rightarrow I(u, v_x) \tag{3}$$

In this example, $I(u, v_i)$ is an encoding function to be learned that shows user u *interacted* with item v_i. We can also learn this function for different meanings based on different scenarios and training data. Intuitively, we use Horn clause to depict if the user's existing behaviors together would imply the user's preference on a new item v_x. In model training, each user's interaction history is represented as a logical expression. Since the number of interactions and the interacted items of different users are different, the logical expressions from all users combined represent a diverse set of training rules. Intuitively, each user contributes to part of the logical space (i.e., the user's logical expression), and they collectively estimate a reasoning model of the space to make predictions for each other, thus noted as neural collaborative reasoning. We will introduce how to learn the model in the next section.

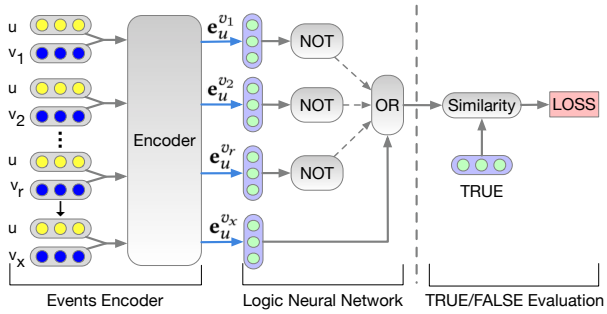


Figure 2: Implementation of the NCR framework. The gray boxes with yellow or blue circles represent user or item embeddings; Blue boxes with green circles represent event embeddings in the logical space, where the encoder is a neural network that encodes user-item interactions to events; NOT and OR are neural logic modules; Dashed arrows mean that the order of the inputs are randomly shuffled in each round.

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:

$$\neg\big(I(u, v_1) \wedge I(u, v_2) \wedge \cdots \wedge I(u, v_r)\big) \vee I(u, v_x) \tag{4}$$

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

$$\big(\neg I(u, v_1) \vee \neg I(u, v_2) \vee \cdots \vee \neg I(u, v_r)\big) \vee I(u, v_x) \tag{5}$$

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 NCR framework in this section, and we will experiment with different forms in later sections.

For simplicity in notation, in the following parts of the paper, we use an *event* e to denote an interaction. Let $\mathcal{U}$ and $\mathcal{V}$ be the set of users and items. Suppose the interaction history of user $u \in \mathcal{U}$ is $\{v_1, v_2, \cdots, v_r\}$, and then we represent these interactions $I(u, v_1), I(u, v_2), \cdots, I(u, v_r)$ as events $e_u^{v_1}, e_u^{v_2}, \cdots, e_u^{v_r}$, where $e_u^{v_i}$ means user u interacted with item v_i. As a result, the personalized recommendation problem becomes predicting if the observed events $e_u^{v_1}, e_u^{v_2}, \cdots, e_u^{v_r}$ would imply a new event $e_u^{v_x}$ by deciding if the following statement is true:

$$e_u^{v_1} \wedge e_u^{v_2} \wedge \cdots \wedge e_u^{v_r} \rightarrow e_u^{v_x} \tag{6}$$

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

$$(\neg e_u^{v_1} \vee \neg e_u^{v_2} \vee \cdots \vee \neg e_u^{v_r}) \vee e_u^{v_x} \tag{7}$$

where $e_u^{v_x}$ is for $I(u, v_x)$. As we will illustrate later, the embedding $\mathbf{e}_u^v$ for event e_u^v will consider both user u and item v, so that the model can make personalized recommendations tailored to a user. Note that we do not make use of the time information in the above modeling, as a result, the ordering of the observed events in the left side of Eq.(6) does not matter. Making use of the time ordering information for sequential NCR will be considered as a future work.

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 explicit feedback signals are very informative for the recommendation task, as a result, it would be very beneficial if we can take the explicit feedback into the logical reasoning procedure.

Fortunately, it is very easy to extend the above formalization for reasoning with explicit feedback. In particular, we change the predicate function $I(u, v)$ to $L(u, v)$, which means if the user *likes* the interacted item. Then, we can extend the definition of event e_u^v to describe the user attitude towards an interacted item. More specifically, we use e_u^v to represent that user u interacted with item v with positive feedback, and use $\neg e_u^v$ to show that user u interacted with item v with negative feedback. Still take the previous example, suppose user u interacted with items $v_1, v_2, \cdots, v_r$, and gave positive feedback on $v_1, v_2, \cdots$, while negative feedback on v_r (could be negative feedback on more items), then if or not to recommend v_x depends on the T/F value of the following statement:

$$e_u^{v_1} \wedge e_u^{v_2} \wedge \cdots \wedge \neg e_u^{v_r} \rightarrow e_u^{v_x} \tag{8}$$

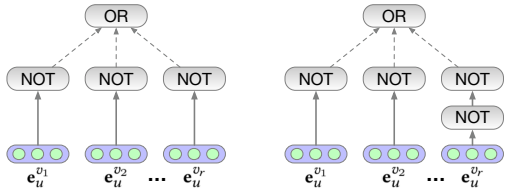
which is equivalently written as:

$$(\neg e_u^{v_1} \vee \neg e_u^{v_2} \vee \cdots \vee \neg\neg e_u^{v_r}) \vee e_u^{v_x} \tag{9}$$

Here, we keep the double negation on $e_u^{v_r}$ to make sure the negation modular will be adequately trained in the neural network, which will be explained with more details 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 we will generalize to explicit feedback cases.



(a) Implicit Feedback **(b) Explicit Feedback**

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

Suppose a user u interacted with $v_1, v_2, \cdots, v_r$, and our model needs to predict if item v_x would be interacted by u. We first use a simple two-layer neural 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:

$$\mathbf{e}_u^v = \mathbf{W}_2\phi(\mathbf{W}_1 \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} + \mathbf{b}_1) + \mathbf{b}_2 \tag{10}$$

where $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ are the user and item latent embedding vectors in d-dimensional space; $\mathbf{W}_1, \mathbf{W}_2$ and $\mathbf{b}_1, \mathbf{b}_2$ are the weight matrices and bias terms to be learned; $\mathbf{e}_u^v \in \mathbb{R}^n$ 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:

$$(\neg \mathbf{e}_u^{v_1} \vee \neg \mathbf{e}_u^{v_2} \vee \cdots \vee \neg \mathbf{e}_u^{v_r}) \vee \mathbf{e}_u^{v_x} \tag{11}$$

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 $\mathbf{F}$ ($\mathbf{F} = \neg\mathbf{T}$). The $\mathbf{T}$ vector is randomly initialized and kept unchanged during model training, and the $\mathbf{F}$ vector is calculated by $\neg\mathbf{T}$. We expect that the final event vector of the expression would be close to $\mathbf{T}$ if v_x 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 a two-layer neural network (Eq.(10)). For example, the $\text{AND}(\cdot,\cdot)$ 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 $\text{NOT}(\cdot)$ module, we can calculate the negated events $\neg\mathbf{e}_u^{v_1}, \neg\mathbf{e}_u^{v_2}, \cdots, \neg\mathbf{e}_u^{v_r}$. After that, we combine the negated events and the candidate event embedding $\mathbf{e}_u^{v_x}$ into the $\text{OR}(\cdot,\cdot)$ module, so as to generate the final event embedding of the whole logical expression. The $\text{OR}(\cdot,\cdot)$ 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. $\mathbf{e}_u^v$ and $\mathbf{e}_u^{v'}$. The output vector $\mathbf{e}_u^{v,v'}$ is treated as the representation of event $\mathbf{e}_u^v \vee \mathbf{e}_u^{v'}$ in the logical representation space. The next event vector $\mathbf{e}_u^{v''}$ and the previous output $\mathbf{e}_u^{v,v'}$ 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,

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	$\neg \mathrm{T} = \mathrm{F}$	$r_1 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 + Sim(\mathrm{NOT}(\mathbf{x}), \mathbf{x})$
	Double Negation	$\neg(\neg x) = x$	$r_2 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{NOT}(\mathrm{NOT}(\mathbf{x})), \mathbf{x})$
AND	Identity	$x \wedge \mathrm{T} = x$	$r_3 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{AND}(\mathbf{x}, \mathbf{T}), \mathbf{x})$
	Annihilator	$x \wedge \mathrm{F} = \mathrm{F}$	$r_4 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{AND}(\mathbf{x}, \mathbf{F}), \mathbf{F})$
	Idempotence	$x \wedge x = x$	$r_5 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{AND}(\mathbf{x}, \mathbf{x}), \mathbf{x})$
	Complementation	$x \wedge \neg x = \mathrm{F}$	$r_6 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{AND}(\mathbf{x}, \mathrm{NOT}(\mathbf{x})), \mathbf{F})$
OR	Identity	$x \vee \mathrm{F} = x$	$r_7 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{OR}(\mathbf{x}, \mathbf{F}), \mathbf{x})$
	Annihilator	$x \vee \mathrm{T} = \mathrm{T}$	$r_8 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{OR}(\mathbf{x}, \mathbf{T}), \mathbf{T})$
	Idempotence	$x \vee x = x$	$r_9 = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{OR}(\mathbf{x}, \mathbf{x}), \mathbf{x})$
	Complementation	$x \vee \neg x = \mathrm{T}$	$r_{10} = \frac{1}{\lvert\mathcal{X}\rvert}\sum_{\mathbf{x}\in\mathcal{X}} 1 - Sim(\mathrm{OR}(\mathbf{x}, \mathrm{NOT}(\mathbf{x})), \mathbf{T})$
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	$x \vee y = y \vee x$ $x \wedge y = y \wedge x$	

can be represented by the following equations:

$$\begin{aligned} \neg \mathbf{e}_u^{v_i} &= \mathrm{NOT}(\mathbf{e}_u^{v_i}), \forall i \in \{1, 2, \cdots, r\} \\ \mathbf{Exp} &= \mathrm{OR}\left(\neg \mathbf{e}_u^{v_1}, \neg \mathbf{e}_u^{v_2}, \cdots, \neg \mathbf{e}_u^{v_r}, \mathbf{e}_u^{v_x}\right) \end{aligned} \tag{12}$$

The final output **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 **Exp** is close to the constant true vector (**T**) in the logical space. As stated before, the true vector is randomly initialized at the beginning and it is never updated during model training, which serves as the anchor vector for all other latent vectors in the logical space. 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 **Exp** and **T** vectors. In this work, we use the most simple cosine similarity measure:

$$Sim(\mathbf{Exp}, \mathbf{T}) = \frac{\mathbf{Exp} \cdot \mathbf{T}}{\|\mathbf{Exp}\|\|\mathbf{T}\|} \tag{13}$$

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 Regularization

We have defined three logical neural modules. However, by now they are just plain neural networks. We need to guarantee that each logical module is really performing the expected logical operation in the latent space. To achieve this goal, similar to [50], 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 (e.g., the $\mathbf{e}_u^{v_i}$ in Eq.(12)), or any intermediate event during the logical neural network calculation (e.g., the output $\neg \mathbf{e}_u^{v_i}$ by feeding $\mathbf{e}_u^{v_i}$ to the NOT$(\cdot)$ module), or the final event embedding **Exp** of the logical expression. Let $\mathcal{X}$ be the set of all event embeddings, and let $Sim(\cdot,\cdot)$ be a similarity function, which is cosine similarity in our implementation. As noted before, **T** is the constant anchor vector representing true, which is randomly initialized and never updated during model learning, and **F** is the vector representing false, which is obtained through NOT(**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:

$$\mathbf{x} \equiv \mathrm{NOT}(\mathrm{NOT}(\mathbf{x})) \tag{14}$$

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 similarity between the output of NOT(NOT(**x**)) and **x**, which is equal to minimizing:

$$1 - Sim\left(\mathrm{NOT}(\mathrm{NOT}(\mathbf{x})), \mathbf{x}\right) \tag{15}$$

To make sure that the logic neural modules can not only perform the expected operation on the initial input events, but also on all of the intermediate hidden events as well as the final event, we apply the regularizer to all of the event embeddings $\mathcal{X}$ in the logical space, which gives us the final regularizer for the double negation law:

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

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 NOT(**x**) and **x**, such as **T** and **F**. The final regularizer considering all laws is:

$$\mathcal{L}_{logicReg} = \sum\nolimits_i r_i \tag{17}$$

The associative and commutative laws cannot 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 use the pair-wise learning algorithm [42] for model training. Specifically, we conduct negative sampling on each given expression during the training process. Suppose we observed that user u interacted with r items $v_{i-1}, v_{i-2}, \cdots, v_{i-r}$ and then the user interacted with item v_i. We 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:

$$\begin{aligned} C_{ui}^{+} &= \neg e_u^{v_{i-1}} \vee \neg e_u^{v_{i-2}} \vee \cdots \vee \neg e_u^{v_{i-r}} \vee e_u^{v_i} \\ C_{uj}^{-} &= \neg e_u^{v_{i-1}} \vee \neg e_u^{v_{i-2}} \vee \cdots \vee \neg e_u^{v_{i-r}} \vee e_u^{v_j} \end{aligned} \tag{18}$$

where C_{ui}^{+} is the expression for the observed ground-truth interaction, and C_{uj}^{-} is the expression for the negative sampled interaction. Then, we have a pair of truth evaluation results:

$$\begin{aligned} s_{ui}^{+} &= Sim\big(\text{LNN}(\mathbf{e}_u^{v_{i-1}}, \mathbf{e}_u^{v_{i-2}}, \cdots, \mathbf{e}_u^{v_{i-r}}, \mathbf{e}_u^{v_i}), \mathbf{T}\big) \\ s_{uj}^{-} &= Sim\big(\text{LNN}(\mathbf{e}_u^{v_{i-1}}, \mathbf{e}_u^{v_{i-2}}, \cdots, \mathbf{e}_u^{v_{i-r}}, \mathbf{e}_u^{v_j}), \mathbf{T}\big) \end{aligned} \tag{19}$$

where LNN is the logic neural network structure as shown in Figure 2. Then we calculate $s_{uij} = \alpha \cdot (s_{ui}^{+} - s_{uj}^{-})$ to represent the difference between these two expressions and apply an optimization algorithm to maximize this difference, where α is an amplification factor to amplify the difference ($\alpha = 10$ in our experiments). In implementation, we sample n negative items for each user-item pair. We use $\mathcal{V}_u^{+}$ to represent the observed example set for user u, and $\mathcal{V}_u^{-}$ to represent the sampled negative example set for user u, where $v_i \in \mathcal{V}_u^{+}$ and $v_j \in \mathcal{V}_u^{-}$. The loss function can be written as:

$$\mathcal{L}_{ncr} = -\sum_{u \in \mathcal{U}} \sum_{v_i \in \mathcal{V}_u^{+}} \sum_{v_j \in \mathcal{V}_u^{-}} \ln \sigma(s_{uij}) + \lambda_\Theta \|\Theta\|_2^2 \tag{20}$$

where Θ represents all of the parameters of the model, including the user and item embeddings, the parameters of the event encoder network, and the parameters of the neural modules; λ_Θ is the ℓ_2-norm regularization coefficient; $\sigma(\cdot)$ is the logistic sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$. Maximizing s_{uij} is equivalent to minimizing $\mathcal{L}_{ncr}$. The pseudo-code for calculating the logic neural network loss is given in Appendix A.2.

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

$$\mathcal{L} = \mathcal{L}_{ncr} + \lambda_r \mathcal{L}_{logicReg} \tag{21}$$

where λ_r is the coefficient for the logic regularizers. We apply the same coefficient to all of the logic regularizers since they are equally important to regularize the logical behavior of the model. We apply back propagation [47] to optimize the parameters.

5 EXPERIMENTS

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

- **RQ1**: What is the performance of the NCR framework in terms of personalized recommendation tasks? Does it outperform state-of-the-art models? (Section 5.5)
- **RQ2**: If and 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)

5.1 Experiment 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.

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%

ML100k [17]. This is a frequently used dataset maintained by Grouplens. It includes 100,000 movie ratings ranging from 1 to 5 from 943 users to 1,682 movies.

Amazon 5-core [35]. 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.

Our NCR framework can be implemented in two ways: reasoning with implicit feedback or with explicit feedback. Following common practice, when reasoning with implicit feedback, we only consider the user interaction information and ignore the ratings, while for reasoning with explicit feedback, we consider 1-3 ratings as negative feedback and 4-5 ratings as positive feedback.

Since our baseline models include some sequential recommendation models, according to the suggestions of [57], we use leave-one-out setting under temporal ordering, i.e., the last interaction of each user is put into the test set, the second-to-last interaction of each user is put into the validation set, and other interactions of each user constitute the training set. Details of the dataset pre-processing procedure are provided in the Appendix A.1.

5.2 Baselines

According to the suggestions in [9, 43], we consider both shallow and deep models as baselines. We compare with two representative shallow models (BPR-MF and SVD++), two deep models (DMF and NeuMF), two session-based models (GRU4Rec and STAMP), as well as one state-of-the-art reasoning-based model (NLR).

- **BPR-MF** [42]: The Bayesian Personalized Ranking model, which is a pair-wise ranking model for recommendation. We

use Biased Matrix Factorization (Bias-MF) [30] as the prediction function under the BPR framework, which considers user, item and global bias terms for matrix factorization. We denote the final model as BPR-MF.

- **SVD++** [28]: Also a matrix factorization based method, which extends Singular Value Decomposition (SVD) by considering user history interactions when modeling the users.
- **DMF** [51]: Deep Matrix Factorization is a deep model for recommendation, which uses multiple non-linear layers to process the raw user-item interaction matrix.
- **NeuMF** [20]: A neural network-based collaborative filtering algorithm, which employs a non-linear prediction network for user and item matching.
- **GRU4Rec** [21]: A sequential/session-based recommendation model, which uses Recurrent Neural Network (RNN) to capture the sequential dependencies in users' historical interactions for prediction and recommendation.
- **STAMP** [40]: A sequential/session-based recommendation model based on the attention mechanism, which captures a user's long-term and short-term preferences.
- **NLR** [50]: Neural Logic Reasoning, which proposes a Logic-Integrated Neural Network (LINN) to take logical constraints and neural modeling for reasoning and prediction.

In the experiments, we test three versions of our model:

- **NCR-I**: Neural Collaborative Reasoning with Implicit feedback, which only uses the interaction information for model learning.
- **NCR-E**: Neural Collaborative Reasoning with Explicit feedback, which adopts the explicit feedback for model learning.
- **NCR-E w/o LR**: Neural Collaborative Reasoning with Explicit feedback but without Logical Regularization, i.e., we remove the logical regularizers from NCR-E by setting $\lambda_r = 0$ in Eq.(21).

We train the pair-wise ranking methods based on 1:1 negative sampling, i.e., for each interacted item $v_i \in \mathcal{V}_u^+$, we sample one negative item $v_j \in \mathcal{V}_u^-$ that the user did not interact with. Source code of our model and the baselines are available on GitHub.[2]

5.3 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.

According to the suggestions of [57], we use *real-plus-N* [3, 48] to calculate the measures. More specifically, for each user-item pair in the validation and test set, we randomly sample 100 irrelevant items, and we rank these 101 items for ranking evaluation.

5.4 Experimental Settings

We use the same train, validation and test datasets for our model and baseline methods in experiments. For fair comparison, for all models including our model and baselines, we tune each model's parameter to its own best performance on the validation set based on NDCG@5. Eventually, all other models except for DMF are set with an embedding size of 64, while for DMF, the embedding size is 128. More details of the parameter settings are shown in Appendix A.3. For our NCR model, the number of layers for all neural modules are set to 2. We apply ReLU non-linear activation function between layers. The learning rate is 0.001, and the weight of ℓ_2-regularization coefficient (λ_Θ in Eq.(20)) is 0.0001 for ML100k dataset and 0.00001 for Amazon datasets. The default logical regularization coefficient (λ_r in Eq.(21)) that we use to report the results is 0.1, and we tune the parameter to see its effect in Section 5.6. We optimize the models using mini-batch Adam [26] with a batch size of 128. More implementation details of the models, as well as the hardware and software settings are provided in Appendix A.3.

[2] https://github.com/rutgerswiselab/NCR

5.5 Performance of the NCR Framework (RQ1)

To evaluate the performance of the NCR framework, we report the results, including NDCG and Hit-Ratio (HR), on all of the datasets in Table 3, where **NCR-I** represents our model with implicit feedback, and **NCR-E** represents our model with explicit feedback. Besides, we use underline to highlight the best result among the matching-based baselines (i.e., the first six baselines) in each column, and use the dagger symbol † to highlight the best result among all baselines, including the reasoning-based NLR baseline. Bold number shows the best result of the whole column.

We first compare with the matching-based models. We see that among the first six baselines, GRU4Rec and STAMP achieve the best performance in most cases. Since the two models use implicit feedback for model training, for fairness in comparison, we use our implicit model NCR-I to compare with the two baselines. The results show that NCR-I is better than the two baselines in most cases. Notice that our model shuffles the input variables in every epoch, which means that we actually did not use the item ordering information. However, our neural collaborative reasoning approach can still outperform the two session-based models on most of the measures. When using explicit feedback for model learning, we see that our NCR-E model achieves even better performance. The Improvement[1] row in Table 3 shows the percentage improvement of NCR-E against the best matching-based model.

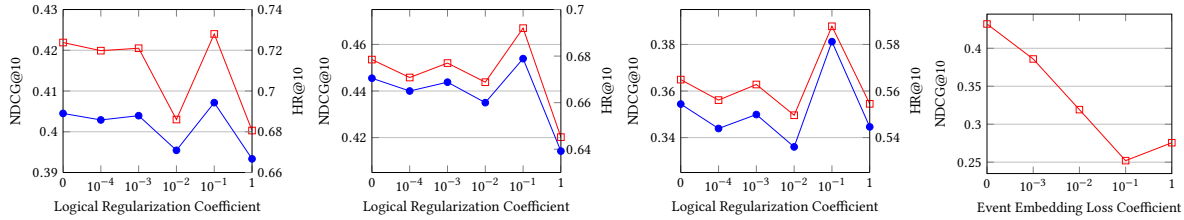
We then compare with the reasoning-based baseline, i.e, the NLR model. Since NLR uses explicit feedback for model learning, we use our explicit model NCR-E to compare with NLR. We see that NCR-E is better than NLR on all metrics. The underlying reason may be two fold. First is that NLR is a non-personalized model, which does not learn personalized user embeddings, but only relies on the logical relationship among items for recommendation. However, our NCR model is personalized by considering both user and item in the event embedding. Another reason is that NCR adopts Horn clauses for reasoning, which is a more natural and straightforward logical language for prediction tasks such as recommendation. The Improvement[2] row in Table 3 shows the percentage improvement of NCR-E against the NLR model.

5.6 The Effect of Logical Regularization (RQ2)

In this section, we answer the question that if the logical regularizers would help to improve the performance. We tune the logical regularizer coefficient λ_r in Eq.(21) from 10^{-5} to 1. The corresponding NDCG@10 and HR@10 are shown in Figure 4(a)-(c). We can see

Table 3: Results of recommendation performance on three datasets with metrics NDCG (N) and Hit Ratio (HR). We use underline (number) to show the best result among the matching-based baselines (i.e., the first six baselines), and we use dagger (number†) to show the best result among all baselines including the reasoning-based NLR method. We use bold font to mark the best result of the whole column. We use one star (*) to indicate that the performance is significantly better than the best matching-based baselines, and use two stars (**) to indicate that the performance is significantly better than all baselines including the NLR baseline. The significance is at 0.05 level based on paired t-test. Improvement^{1} shows our model improvement over the best matching-based result (i.e., over number), while improvement^{2} shows our model improvement over NLR.

	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
BPR-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	0.5455	0.6843†	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	0.3564	0.4122	0.5134	0.6856†	0.4038	0.4459	0.5287	0.6688	0.3154	0.3551	0.4284	0.5511
STAMP	0.3560	0.4070	0.5159†	0.6730	0.3935	0.4366	0.5246	0.6577	0.3095	0.3489	0.4196	0.5430
NLR	0.3602†	0.4151†	0.5102	0.6795	0.4191†	0.4591†	0.5506†	0.6739	0.3475†	0.3852†	0.4623†	0.5788†
NCR-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
NCR-E w/o **LR**	0.3671	0.4219	0.5180	0.6890	0.4126	0.4535	0.5444	0.6705	0.3272	0.3649	0.4377	0.5544
NCR-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***
Improvment^{1}	5.50%	2.86%	5.76%	1.27%	5.37%	4.73%	2.86%	0.70%	10.94%	9.21%	8.29%	5.46%
Improvment^{2}	4.39%	2.14%	6.71%	2.66%	1.53%	1.72%	1.91%	2.26%	0.69%	0.67%	0.35%	0.41%



(a) ML100k **(b) Movies and TV** **(c) Electronics** **(d) λ_e on ML100k**

Figure 4: (a)-(c): NDCG@10 (red squared line) and HR@10 (blue circled line) on three datasets according to the increase of the logical regularization coefficient λ_r. (d): NDCG@10 when increasing the event embedding loss coefficient λ_e on ML100k.

that the best performance would be reached by assigning the logical regularization coefficient to 0.1. 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 constraint is too weak or too strong, the performance of the model would be negatively influenced.

In Table 3, **NCR-E w/o LR** shows the performance of our explicit feedback model without using logical regularizers (i.e., setting $\lambda_r = 0$). By comparing **NCR-E w/o LR** and **NCR-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 0.05 level except for NDCG@10 on the ML100k dataset. This result shows that logical regularizers do help to improve the recommendation performance in our framework.

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 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 experessions. 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 many 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

Table 4: Ranking performance under different logical structures. "*" indicates significance at 0.05 level under paired t-test.

	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
NLR	0.3529	0.4066	0.5113	0.6763	0.4191	0.4591	0.5506	0.6739	0.3475	0.3852	0.4623	0.5788
[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]**NCR-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*

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.

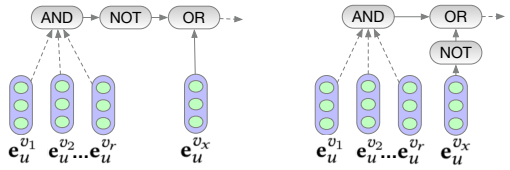
To answer the question, we explore two alternative network structures. One is a logically equivalent model (**EqModel**), i.e., we still use the logical expression $\mathbf{e}_u^{v_1} \wedge \mathbf{e}_u^{v_2} \wedge \cdots \wedge \mathbf{e}_u^{v_r} \rightarrow \mathbf{e}_u^{v_x}$ to model the task. However, it is represented as $\neg(\mathbf{e}_u^{v_1} \wedge \mathbf{e}_u^{v_2} \wedge \cdots \wedge \mathbf{e}_u^{v_r}) \vee \mathbf{e}_u^{v_x}$, instead of $(\neg\mathbf{e}_u^{v_1} \vee \neg\mathbf{e}_u^{v_2} \vee \cdots \vee \neg\mathbf{e}_u^{v_r}) \vee \mathbf{e}_u^{v_x}$ that we used before (Eq.(11)). Figure 5(a) shows the network structure of the logically equivalent model. One can see that although this model is logically equivalent to NCR, 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 logical expression $\mathbf{e}_u^{v_x} \rightarrow \mathbf{e}_u^{v_1} \wedge \mathbf{e}_u^{v_2} \wedge \cdots \wedge \mathbf{e}_u^{v_r}$, which is equivalent to $\neg\mathbf{e}_u^{v_x} \vee (\mathbf{e}_u^{v_1} \wedge \mathbf{e}_u^{v_2} \wedge \cdots \wedge \mathbf{e}_u^{v_r})$, to build the neural structure. Figure 5(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, based on 5-round random experiments, we provide the p-value under paired t-test between the EqModel and the original NCR-E model (p-value^{1,3} in the table), as well as between the CMPModel and NCR-E (p-value^{2,3} in the table). For easy reference, we copy the results of GRU4Rec, which is the best matching-based baseline model, as well as the results of NLR, which is the reasoning-based model from Table 3 to Table 4.

We have two key observations from the results in Table 4. First, we see that both NCR-E and the 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 NCR-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 NCR-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., NCR-E is not significantly better than EqModel), while NCR-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 NCR-E and EqModel are logically equivalent, they



(a) EqModel **(b) CMPModel**

Figure 5: Comparison between the network structure that (a) follows the logical prior and (b) violates the logical prior.

achieve comparable performance when the training data is sufficient. However, NCR-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 NCR-E. As a result, NCR-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, to explore if the recommendation task can be modeled based on Boolean logic, we apply a constraint that any event embedding can only be either the **T** vector or the **F** vector. 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 $\mathbf{e}_u^v$ 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 embedding $\mathbf{e}_u^v$ and the **T** vector, otherwise, if the user has a negative feedback on the item, we minimize the MSE between $\mathbf{e}_u^v$ and the **F** vector. The event embedding loss function is:

$$\mathcal{L}_{event} = \sum_u \sum_{v \in \mathcal{V}_u^+} \text{MSE}(\mathbf{e}_u^v, \mathbf{G}) \tag{22}$$

where $\mathcal{V}_u^+$ is the set of user u's interacted items, and $\mathbf{G}$ 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 current loss function in Eq.(21) to achieve the following new loss function:

$$\mathcal{L}_{Boolean} = \mathcal{L}_{ncr} + \lambda_r \mathcal{L}_{logicReg} + \lambda_e \mathcal{L}_{event} \tag{23}$$

where λ_e is the coefficient of the event embedding loss. By adding this loss, the model tries to polarize the event embeddings to either **T** or **F**. The model would then conduct reasoning in an approximate binary space when the event embedding loss $\mathcal{L}_{event}$ is minimized to a very small number (0.0001 in our experiment). We present the results of NDCG@10 on ML100k with $\lambda_e \in [0.001, 0.01, 0.1, 1]$ in Figure 4(d). Results on other datasets are similar.

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

6 CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a Neural Collaborative Reasoning (NCR) framework, which models recommendation as a reasoning task by integrating logical structures and neural networks for personalized recommendation. Experiments show that our method provides significant improvement on the ranking performance. 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.

Our work provides a very fundamental framework to integrate learning and reasoning, which conducts neural logic reasoning on top of learned vector representations. This inspires a wide scope of possibilities for future work. In this work, we only used the user interaction information for collaborative reasoning, while in the future, it is interesting to consider contextual and multimodal information for reasoning. Besides, the modularized design of our framework makes it promising to integrate with Neural Architecture Search (NAS) for Automatic Machine Learning (AutoML) [13], Program Synthesis [16] and Explainable AI [55]. Finally, except for the recommendation task, the framework can also be extended to other tasks such as predicate logic reasoning, vision and language reasoning, knowledge graph reasoning, graph neural networks, social networks, and domain-specific applications such as medical and legal research where cognitive reasoning is significantly required, which are very promising directions to explore in the future.

ACKNOWLEDGMENTS

We thank the reviewers for the reviews and suggestions. This work was supported in part by NSF IIS-1910154 and IIS-2007907. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

Algorithm 1 Neural Collaborative Reasoning Loss (Eq.(20))

Input: Training user set U, item set V, model parameters Θ, ℓ_2-regularization coefficient λ_Θ
Output: Model loss

procedure CalcNCRLoss
  Randomly initialize Θ
  $Loss \leftarrow 0$
  for $u \in U$ **do**
    $V_{hist} \leftarrow$ drawHistory(u) ▷ obtain history of user u
    $v_i \leftarrow$drawTarget(u) ▷ obtain the target item of user u
    $V^- \leftarrow$ Sample(u, v_i, V_{hist}, V, n) ▷ get n negative samples
    $E \leftarrow$ ENCODE(V_{hist}) ▷ get history event embeddings
    $\mathbf{e}_i \leftarrow$ ENCODE(v_i) ▷ get target event embedding
    $s^+_{ui} \leftarrow$ Sim($LNN(E, \mathbf{e}_i), \mathbf{T}$) ▷ get target event score
    for $v_j \in V^-$ **do**
      $\mathbf{e}_j \leftarrow$ ENCODE(v_j)▷ get negative event embedding
      $s^-_{uj} \leftarrow$ Sim($LNN(E, \mathbf{e}_j), \mathbf{T}$) ▷ negative event score
      $s_{uij} \leftarrow \alpha \cdot (s^+_{ui} - s^-_{uj})$▷ get the final pair-wise score
      $Loss \leftarrow Loss + \ln \sigma(s_{uij})$ ▷ update loss
  $Loss \leftarrow -Loss + \lambda_\Theta \|\Theta\|_2^2$ ▷ update loss
  return $Loss$ ▷ return loss

APPENDIX

A.1 Data Preprocessing

We transform the rating information, which comes with 1 to 5 ratings, to 0 and 1, which represents the negative or positive explicit feedback. 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 dataset by timestamp. For each user and item pair in the dataset, we select its corresponding most recent n history interactions to build the logical expression. Here we set the length of history to 5, which means that each user-item pair comes with 5 history interactions. For those items from the earliest 5 interactions of the corresponding user, we put them into the training dataset. Users with less than 5 interactions are put into the 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 the validation set and the 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.

A.2 Pseudo-Code to Calculate the NCR Loss

The pseudo-code for calculating the NCR loss in Eq.(20) is shown in Algorithm 1.

A.3 Additional Experimental Settings

We carefully tune the parameters for all baseline models to reach their best performance. For DMF, we implemented the model with two-layer neural networks to model users and items, respectively. Since the authors claimed that the increment of vector size would help to improve the performance, we tune the hidden vector size and set it to 128 to reach the best performance; For NeuMF, we use

a three-layer multi-layer perceptron network with layer sizes 32, 16, 8 as mentioned by the author. The final output layer has only one layer with dimension 64; For both the GRU4Rec and STAMP models, we use 5 as the history length, which is the same as our model in 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 NCR model to guarantee that the differences in the reported results only result from the variation of the neural network structure.

For training process, early-stopping is conducted according to the performance on the validation set. Training examples that share the same neural network structure are put into the same mini-batch for better efficiency. 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 of the neural network parameters for DMF, NeuMF, GRU4Rec, STAMP, NLR, NCR, CMPModel and EqModel are initialized from a normal distribution with 0 mean and 0.01 standard deviation. Our framework is implemented with PyTorch [39] v1.4 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] Qingyao Ai, Vahid Azizi, Xu Chen, and Yongfeng Zhang. 2018. Learning heterogeneous knowledge base embeddings for explainable recommendation. *Algorithms* 11, 9 (2018), 137.

[3] Alejandro Bellogin, Pablo Castells, and Ivan Cantador. 2011. Precision-oriented evaluation of recommender systems: an algorithmic comparison. In *Proceedings of the fifth ACM conference on Recommender systems*. 333–336.

[4] Yoshua Bengio. 2019. From System 1 Deep Learning to System 2 Deep Learning. In *NeurIPS'2019*.

[5] 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).

[6] 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.

[7] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *DLRS: Proceedings of the 1st RecSys workshop on deep learning for recommender systems*. 7–10.

[8] Maurizio Ferrari Dacrema, Simone Boglio, Paolo Cremonesi, and Dietmar Jannach. 2021. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems (TOIS)* (2021).

[9] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *RecSys*. 101–109.

[10] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. 2019. Bridging machine learning and logical reasoning by abductive learning. In *NeurIPS*. 2815–2826.

[11] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural logic machines. *ICLR*.

[12] 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.

[13] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research* 20, 55 (2019), 1–21.

[14] Maurizio Ferrari Dacrema, Federico Parroni, Paolo Cremonesi, and Dietmar Jannach. 2020. Critically Examining the Claimed Value of Convolutions over User-Item Embedding Maps for Recommender Systems. In *CIKM*. 355–363.

[15] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. 2012. *Neural-symbolic learning systems: foundations and applications*. Springer Sci. & Bus. Media.

[16] Sumit Gulwani, Oleksandr Polozov, Rishabh Singh, et al. 2017. Program synthesis. *Foundations and Trends® in Programming Languages* 4, 1-2 (2017), 1–119.

[17] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm TIST* 5, 4 (2016), 19.

[18] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *RecSys*. 161–169.

[19] Ruining He and Julian McAuley. 2016. VBPR: Visual Bayesian Personalized Ranking from Implicit Feedback. In *AAAI*.

[20] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *WWW*. 173–182.

[21] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and D Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR*.

[22] Cheng-Kang Hsieh, Longqi Yang, Yin Cui, Tsung-Yi Lin, Serge Belongie, and Deborah Estrin. 2017. Collaborative metric learning. In *In WWW*. 193–201.

[23] Zhengyao Jiang and Shan Luo. 2019. Neural Logic Reinforcement Learning. *Proceedings of the 36th International Conference on Machine Learning* (2019).

[24] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE.

[25] 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.

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

[27] 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.

[28] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *SIGKDD*. ACM, 426–434.

[29] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. *KDD* (2009).

[30] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[31] Daniel D Lee and H Sebastian Seung. 2001. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*. 556–562.

[32] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *CIKM*. 1419–1428.

[33] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon.Com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing* (2003).

[34] Gary Marcus. 2020. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177* (2020).

[35] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR*. ACM.

[36] WS McCulloch and Walter Pitts. 1943. A Logical Calculus of the Idea Immanent in Neural Nets. *Bulletin ofMathematical Biophysics* 5 (1943), 115–133.

[37] 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.

[38] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *Advances in neural information processing systems*. 1257–1264.

[39] 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).

[40] Qiao Liu, Yifu Zeng, Refuoe Mokhosi, Haibin Zhang. 2018. STAMP: short-term attention/memory priority model for session-based recommendation. In *KDD*.

[41] Meng Qu and Jian Tang. 2019. Probabilistic logic neural networks for reasoning. In *Advances in Neural Information Processing Systems*. 7710–7720.

[42] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.

[43] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. *RecSys* (2020).

[44] 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.

[45] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to Recommender Systems Handbook. *Springer US* (2011).

[46] Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine learning* 62, 1-2 (2006), 107–136.

[47] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.

[48] Alan Said and Alejandro Bellogín. 2014. Comparative recommender system evaluation: benchmarking recommendation frameworks. In *RecSys*.

[49] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *WWW*. ACM, 285–295.

[50] Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, and Yongfeng Zhang. 2020. Neural Logic Reasoning. In *CIKM*. ACM.

[51] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems.. In *IJCAI*.

[52] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative Knowledge Base Embedding for Recommender Systems. In *KDD*.
[53] Shuai Zhang, Lina Yao, and Aixin Sun. 2019. Deep learning based recommender system: A survey and new perspectives. *Comput. Surveys* 52, 1 (2019), 1–38.
[54] 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.
[55] Yongfeng Zhang and Xu Chen. 2020. Explainable recommendation: A survey and new perspectives. *Foundations and Trends in Information Retrieval* 14, 1 (2020).
[56] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. 2020. Efficient probabilistic logic reasoning with graph neural networks. *ICLR* (2020).
[57] Wayne Xin Zhao, Junhua Chen, Pengfei Wang, Qi Gu, and Ji-Rong Wen. 2020. Revisiting Alternative Experimental Settings for Evaluating Top-N Item Recommendation Algorithms. *CIKM* (2020).
[58] Lei Zheng, Vahid Noroozi, and Philip S. Yu. 2017. Joint deep modeling of users and items using reviews for recommendation. In *WSDM*.