Neural Logic Reasoning

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

Hanxiong Chen*
Rutgers University
New Brunswick, USA
hanxiong.chen@rutgers.edu

Weizhi Ma
Tsinghua University
Beijing, China
mawz12@hotmail.com

Jiaxin Mao
Tsinghua University
Beijing, China
maojiaxin@gmail.com

Min Zhang
Tsinghua University
Beijing, China
z-m@tsinghua.edu.cn

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

ABSTRACT

Recent years have witnessed the success of deep neural networks in many research areas. The fundamental idea behind the design of most neural networks is to learn similarity patterns from data for prediction and inference, which lacks the ability of cognitive reasoning. However, the concrete ability of reasoning is critical to many theoretical and practical problems. On the other hand, traditional symbolic reasoning methods do well in making logical inference, but they are mostly hard rule-based reasoning, which limits their generalization ability to different tasks since difference tasks may require different rules. Both reasoning and generalization ability are important for prediction tasks such as recommender systems, where reasoning provides strong connection between user history and target items for accurate prediction, and generalization helps the model to draw a robust user portrait over noisy inputs.

In this paper, we propose Logic-Integrated Neural Network (LINN) to integrate the power of deep learning and logic reasoning. LINN is a dynamic neural architecture that builds the computational graph according to input logical expressions. It learns basic logical operations such as AND, OR, NOT as neural modules, and conducts propositional logical reasoning through the network for inference. Experiments on theoretical task show that LINN achieves significant performance on solving logical equations and variables. Furthermore, we test our approach on the practical task of recommendation by formulating the task into a logical inference problem. Experiments show that LINN significantly outperforms state-of-the-art recommendation models in Top-K recommendation, which verifies the potential of LINN in practice.

KEYWORDS

Neural Networks; Machine Learning; Machine Reasoning; Collaborative Reasoning; Cognitive AI

* This work was conducted when Shaoyun Shi was visiting at Rutgers University. The first two authors contributed equally to the work.

Permission to make digital or hard copies of all or part 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 components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411949

ACM Reference Format:
Shaoyun Shi, Hanxiong Chen, Weizhi Ma, Jiaxin Mao, Min Zhang, Yongfeng Zhang. 2020. Neural Logic Reasoning. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411949

1 INTRODUCTION

Deep neural networks have shown remarkable success in many fields, such as computer vision, natural language processing, information retrieval, and data mining. The design philosophy of most neural network architectures is learning statistical similarity patterns from large scale training data. For example, representation learning approaches learn vector representations from image or text for prediction, while metric learning approaches learn similarity functions for matching and inference.

Though neural networks usually have good generalization ability on dataset with similar distribution, the design philosophy of these approaches makes it difficult for neural networks to conduct logical reasoning in many theoretical and practical problems. However, logical reasoning is an important ability for intelligence, and it is critical to many theoretical tasks such as solving logical equations, as well as practical tasks such as medical decision support systems, legal assistants, and personalized recommender systems. For example, in recommendation tasks, reasoning can help to model complex relationships between users and items (e.g., user likes item A but not item B → user likes item C), especially for those rare patterns, which is usually difficult for neural networks to capture.

One typical way to conduct reasoning is through logical inference. In fact, logical inference based on symbolic reasoning was the dominant approach to AI before the emerging of machine learning approaches, and it served as the underpinning of many expert systems in Good Old Fashioned AI (GOFAI). However, traditional symbolic reasoning methods for logical inference are mostly hard rule-based reasoning, which are not easily applicable to many real-world tasks due to the difficulty in defining the rules. For example, in personalized recommendation systems, if we consider each item as a symbol, we can hardly capture all of the relationships between the items based on manually designed rules. Besides, personalized user preferences bring various interacting sequences, which may result in conflicting reasoning rules, e.g., one user may like both item A and B, while another user who liked A may dislike B. Such noise in data makes it very challenging to design recommendation rules that properly generalize to new data.

built dynamically according to the input logic expression. We further leverage logic regularizers over the neural modules to guarantee that each module conducts the expected logical operation.

3.1 Logic Operations as Neural Modules

An expression of propositional logic consists of logic constants (`True` or `False`, noted as T or F), logic variables (v), and basic logic operations (negation $\neg$, conjunction $\wedge$, and disjunction $\vee$). In LINN, the logic operations are learned as three neural modules. Leshno et al. [23] proved that multi-layer feed-forward networks with non-polynomial activation function can approximate any function. This provides us theoretical support for leveraging neural modules to learn the logic operations. Similar to most neural models in which input variables are learned as vector representations, in our framework, T, F and all logic variables are represented as vectors with the same dimension. Formally, suppose we have a set of logic expressions $E = \{e_i\}_{i=1}^m$ and their values $Y = \{y_i\}_{i=1}^m$ (either T or F), and they are constructed by a set of variables $V = \{v_i\}_{i=1}^n$, where $|V| = n$ is the number of variables. An example logic expression e would look like $(v_i \wedge v_j) \vee \neg v_k = T$.

We use bold font to represent vectors, e.g. $\mathbf{v}_i$ is the vector representation of variable v_i, and $\mathbf{T}$ is the vector representation of logic constant T, where the vector dimension is d. $\text{AND}(\cdot,\cdot)$, $\text{OR}(\cdot,\cdot)$, and $\text{NOT}(\cdot)$ are three neural modules. For example, $\text{AND}(\cdot,\cdot)$ takes two vectors $\mathbf{v}_i, \mathbf{v}_j$ as inputs, and the output $\mathbf{v} = \text{AND}(\mathbf{v}_i, \mathbf{v}_j)$ is the representation of $v_i \wedge v_j$, a vector of the same dimension d as $\mathbf{v}_i$ and $\mathbf{v}_j$. The three modules can be implemented by various neural structures, as long as they are able to learn the logical operations.

Figure 1(a) is an example of the LINN architecture corresponding to the expression $(v_i \wedge v_j) \vee \neg v_k$. The lower box shows how the framework constructs a logic expression. Each intermediate vector represents part of the logic expression, and finally, we have the vector representation of the whole logic expression $\mathbf{e} = (\mathbf{v}_i \wedge \mathbf{v}_j) \vee \neg\mathbf{v}_k$. In this way, although the size of the variable embedding matrix grows linearly with the total number of logic variables, the total number of parameters in the operator networks keeps the same. Besides, since the expressions are organized as tree structures, we can calculate it recursively from leaves to the root.

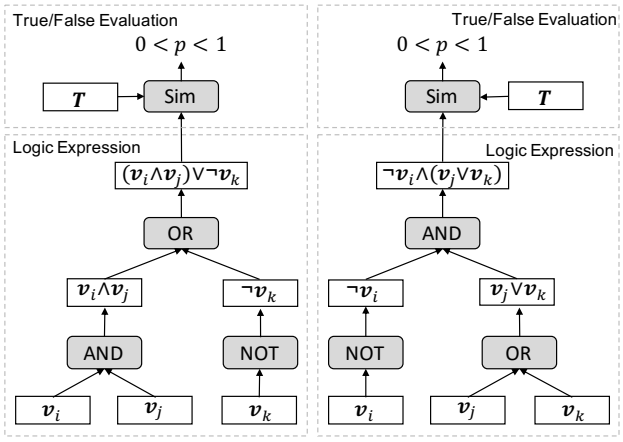
To evaluate the T/F value of the expression, we calculate the similarity between the expression vector and the $\mathbf{T}$ vector, as shown in the upper box, where T, F are short for logic constants `True` and `False`, respectively, and $\mathbf{T}$, $\mathbf{F}$ are their vector representations. Here $Sim(\cdot,\cdot)$ is also a neural module to calculate the similarity between two vectors and output a similarity value between 0 and 1. The output $p = Sim(\mathbf{e}, \mathbf{T})$ evaluates how likely LINN considers the expression to be true.

LINN can be trained with different task-specific loss functions for different tasks. For example, training LINN on a set of expressions and predicting T/F values of other expressions can be considered as a classification problem, and we can adopt cross-entropy loss:

$$L_t = L_{ce} = -\sum_{e_i \in E} y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \tag{1}$$

and for the top-n recommendation task, a pair-wise training loss based on Bayesian Personalized Ranking (BPR) [31] can be used:

$$L_t = L_{bpr} = -\sum_{e^+} \log\left(sigmoid(p(e^+) - p(e^-))\right) \tag{2}$$



(a) $(v_i \wedge v_j) \vee \neg v_k$ **(b)** $\neg v_i \wedge (v_j \vee v_k)$

Figure 1: Examples of the Logic-Integrated Neural Network (LINN) architecture. LINN learns each variable as a vector embedding, and learns each logic operation (i.e., $\wedge, \vee, \neg$) as a logic regularized neural module (i.e., AND, OR, NOT). For a given logic expression, e.g., $(v_i \wedge v_j) \vee \neg v_k$ in the left figure, LINN dynamically assembles the neural architecture according to the logic expression so as to learn the embedding of the whole expression. It then compares the expression with the constant True vector to evaluate the expression. For different input expressions, e.g., $\neg v_i \wedge (v_j \vee v_k)$ in the right figure, LINN reuses the variable embeddings and logic modules to assemble the architecture. In this way, LINN has the ability to model a compositional number of logic expressions.

where e_+ are the expressions corresponding to the positive interactions in the dataset, and e^- are the sampled negative expressions.

3.2 Logical Regularization over Neural Modules

So far, we only learned the logic operations AND, OR, NOT as neural modules, but did not explicitly guarantee that these modules implement the expected logic operations. For example, any variable or expression $\mathbf{w}$ conjuncted with false should result in false, i.e. $\mathbf{w} \wedge \mathbf{F} = \mathbf{F}$, and a double negation should result in itself $\neg(\neg\mathbf{w}) = \mathbf{w}$. Here we use $\mathbf{w}$ instead of $\mathbf{v}$ in the previous section, because $\mathbf{w}$ could either be a single variable (e.g., $\mathbf{v}_i$) or an expression in the middle of the calculation flow (e.g., $\mathbf{v}_i \wedge \mathbf{v}_j$). A LINN that aims to implement logic operations should satisfy the basic logic rules. Although the neural architecture may implicitly learn the logic operations from data, it would be better if we apply constraints to guide the learning of logical operations. To achieve this goal, we define logic regularizers to regularize the behavior of the modules, so that they perform certain logical operations. A complete set of the logical regularizers are shown in Table 1.

The regularizers are categorized by the three operations. To form the logical regularizers, these equations of laws are translated into the modules and variables in LINN, i.e. r_i's in the table. It should be noted that these logical rules are not considered in the whole vector space $\mathbb{R}^d$, but in the vector space defined by LINN. Suppose the set of all input variables as well as intermediate states and final expressions observed during training process is represented as W,

there are few works considering high-order recommendation rules or explanations for recommendation.

It should be noted that the above recommendation model is non-personalized, i.e., we do not learn user embeddings for prediction and recommendation, but instead only rely on item relationships for recommendation. However, as we will show in the following experiments, such a simple non-personalized model outperforms state-of-the-art personalized neural sequential recommendation models, which implies the great power and potential of reasoning in recommendation tasks. Extending the model to personalized versions will be considered in the future.

5.1 Experimental Settings

5.1.1 Training and Evaluation. The models are evaluated on the Top-K Recommendation task. We use the pair-wise learning strategy [31] to train the model, which is a commonly used training strategy in many ranking tasks. In detail, we use the positive interactions in the datasets to train the baseline models and use the expressions corresponding to the positive interactions to train our LINN model. For each positive interaction v^+, we randomly sample an item the user dislikes or has never interacted with before as the negative sample v^- in each epoch. Then the loss function of the base models is:

$$L = -\sum_{v^+} \log\left(sigmoid(p(v^+) - p(v^-))\right) + \lambda_\Theta \|\Theta\|_F^2 \qquad (10)$$

where $p(v^+)$ and $p(v^-)$ are the predictions of v^+ and v^-, respectively, and $\lambda_\Theta \|\Theta\|_F^2$ is ℓ_2-regularization. The loss function encourages the predictions of positive interactions to be higher than the negative samples. For our LINN, suppose the logic expression with v^+ as the target item is $e^+ = (\cdot \wedge v^+) \vee \cdots \vee (\cdot \wedge v^+)$, then the negative expression is $e^- = (\cdot \wedge v^-) \vee \cdots \vee (\cdot \wedge v^-)$, which has the same history interactions as e^+ but replaces v^+ with v^-. Then the final loss function of our LINN model is:

$$\begin{aligned} L = &-\sum_{e^+} \log\left(sigmoid(p(e^+) - p(e^-))\right) \\ &+ \lambda_l \sum_i r_i + \lambda_\ell \sum_{w \in W} \|\mathbf{w}\|_F^2 + \lambda_\Theta \|\Theta\|_F^2 \end{aligned} \qquad (11)$$

where $p(e^+)$ and $p(e^-)$ are the predictions of e^+ and e^-, respectively, and other parts are the logic, vector length and ℓ_2-regularizers as mentioned before. In Top-K evaluation, we sample 100 v^- for each v^+ and evaluate the rank of v^+ in these 101 candidates, which is a common way in many previous works. Recommendation performance is evaluated on two metrics (mean of all evaluation samples):

- **nDCG@K**: larger is better. It is one of the most popular ranking metrics in information retrieval, which is higher when the results are more similar to the ideal ranking. We consider the top-10 results, i.e., $K = 10$.
- **Hit@K**: larger is better. It is the percentage of ranking lists that include at least one positive item in the top-K highest ranked items. We use $K = 1$, which means that we evaluate whether the top-ranked item is positive.

5.1.2 Datasets. Experiments are conducted on two publicly available datasets:

- **ML-100k** [13]. It is maintained by Grouplens[2], which has been used by researchers for many years. It includes 100,000 ratings ranging from 1 to 5 from 943 users and 1,682 movies.
- **Amazon Electronics** [15]. Amazon Dataset[3] is a public e-commerce dataset. It contains reviews and ratings of items given by users on Amazon – a popular e-commerce website. We use a subset in the area of Electronics, containing 1,689,188 ratings ranging from 1 to 5 from 192,403 users and 63,001 items, which is larger and much more sparse than the ML-100k dataset.

Table 3: Statistics of the Datasets

Dataset	#User	#Item	#Positive	#Negative
ML-100k	943	1,682	55,375	44,625
Electronics	192,403	63,001	1,356,067	333,121

The statics of two datasets are summarized in Table 3. The ratings are transformed into 0 and 1. Ratings equal to or higher than 4 ($r_{i,j} \geq 4$) are transformed to 1, which means positive attitudes (like). Other ratings ($r_{i,j} \leq 3$) are converted to 0, which means negative attitudes (dislike). Then, each user's interactions are sorted by time, and each positive interaction of the user is translated to a logic expression in the way mentioned above (Equation 9). We ensure that the expressions corresponding to a user's earliest 5 positive interactions are in the training set. For those users with no more than 5 positive interactions, all the expressions are in the training set. For the remaining data, the last two positive expressions of every user are distributed into the validation set and test set, respectively (test set is preferred if there remains only one expression for the user). All the other expressions are in the training set. This way of data partition and evaluation is usually called the Leave-One-Out setting in personalized recommendation research.

5.1.3 Baselines. We compare LINN with the following baselines:

- **BPRMF** [31]. This is a Matrix Factorization (MF)-based model that applies a pairwise ranking loss. It makes recommendations by the dot product result of user and item vectors plus biases, which is popular and competitive for item recommendation.
- **SVD++** [21]. This model integrates both explicit and implicit feedback, which explicitly models the user history interactions. It has been verified as a powerful model of personalized recommendation in many previous works.

Not only traditional recommendation models, but also some recent neural recommendation models are compared to LINN. In this work, we select the following neural models, which are more powerful compared with other neural matching-based models [6].

- **STAMP** [25]. The Short-Term Attention/Memory Priority model, which uses the attention mechanism to model both short-term and long-term user preferences. This is one of the state-of-the-art recommendation models that consider the recent user interactions for recommendation.
- **GRU4Rec** [16]. A sequential recommendation model that applies Gated Recurrent Unit (GRU) [4] to derive the ranking scores.
- **NARM** [24]. This model utilizes GRU and the attention mechanism to consider the importance of different interactions, which

[2] https://grouplens.org/datasets/movielens/100k/
[3] http://jmcauley.ucsd.edu/data/amazon/index.html

Table 4: Performance on the recommendation task

	ML-100k			Amazon Electronics		
	nDCG@10	Hit@1	time/epoch	nDCG@10	Hit@1	time/epoch
BPRMF [31]	0.3664 ± 0.0017	0.1537 ± 0.0036	4.9s	0.3514 ± 0.0002	0.1951 ± 0.0004	112.1s
SVD++ [21]	0.3675 ± 0.0024	0.1556 ± 0.0044	30.4s	0.3582 ± 0.0004	0.1930 ± 0.0006	469.3s
STAMP [25]	0.3943 ± 0.0016	0.1706 ± 0.0022	8.3s	0.3954 ± 0.0003	0.2215 ± 0.0003	352.7s
GRU4Rec [16]	0.3973 ± 0.0016	0.1745 ± 0.0038	7.1s	0.4029 ± 0.0009	0.2262 ± 0.0009	225.0s
NARM [24]	*0.4022 ± 0.0015*	0.1771 ± 0.0016	9.6s	0.4051 ± 0.0006	0.2292 ± 0.0005	268.8s
LINN-R_l	0.4022 ± 0.0027	*0.1783 ± 0.0043*	20.7s	*0.4152 ± 0.0014*	*0.2396 ± 0.0019*	498.0s
LINN	**0.4064 ± 0.0015***	**0.1850 ± 0.0053***	30.7s	**0.4191 ± 0.0012***	**0.2438 ± 0.0014***	754.9s

* Significantly better than the best of other results (italic ones) with $p < 0.05$

improves the performance of recommendation. It is a state-of-the-art sequential recommendation model.

In our experiments, for LINN and all baseline models utilizing the sequential information (recent interactions), at most 10 previous interactions right before the target positive item are considered.

5.1.4 Parameters. All the models, including baselines, are trained with Adam [20] in mini-batches at the size of 128. The learning rate is 0.001 and early-stopping is conducted according to the performance on the validation set. Models are trained at most 100 epochs. To prevent models from overfitting, we use both the ℓ_2-regularization and dropout. The weight of ℓ_2-regularization λ_Θ is set between 1×10^{-7} to 1×10^{-4} and dropout ratio is set to 0.2. Vector sizes of the variables and the user/item vectors in the recommendation are 64. We run the experiments with five different random seeds and report the average results and standard errors. For LINN, λ_l is set to 1×10^{-6} on ML-100k and 1×10^{-5} on Electronics. λ_ℓ is set to 1×10^{-4} on ML-100k and 1×10^{-6} on Electronics. Note that LINN has similar time and space complexity with baseline models, and each experiment run can be finished in 6 hours (several minutes on small datasets) with a single GPU (NVIDIA GeForce GTX 1080Ti). [4]

5.2 Overall Performance

The overall performance of models on two datasets are shown in Table 4. From the results, STAMP achieves the best performance among the non-sequential baselines because it utilizes the attention mechanism to model both long-term and short-term user preferences. Models utilizing sequential information, such as GRU4Rec, and NARM, are significantly better than other non-sequential baselines, which is also observed by other works [16, 24]. It shows that sequential information is helpful to improve the recommendation performance. NARM performs better than GRU4Rec because it utilizes two attention networks to model the importance of interactions for users' local and global preferences. Although the personalized recommendation is not a standard logical inference problem, logical inference still helps in this task, which is shown by the results – we see that on both the ML100k and Electronics, LINN achieves the best performance. LINN makes more significant improvements on the Electronics dataset because the dataset is more sparse, and integrating logic inference is more beneficial than only using collaborative filtering. Note that without logic regularizers, LINN-R_l still achieves comparable performance with NARM on ML-100k and is significantly better on Electronics.

Our LINN model is simple in nature, which only learns three lightweight modules without personalized user embedding for all of the prediction tasks. It is interesting to see that such a simple approach outperforms state-of-the-art complex models such as NARM, which adopts a bi-attention-network and learns the user purpose features for personalization. This implies the power of reasoning based on modularized logic networks, and indicates a new way of designing neural networks and modeling structural data. We expect the performance of LINN will be further improved when adding advanced modeling techniques into the architecture, such as personalization, attention mechanism, and user's long/short-term preferences, which will be explored in the future.

We also show the training time per epoch of different models. Models explicitly modeling user history are generally slower than BPRMF, among which SVD++ takes more time for taking the full user history as inputs. Because of the dynamic computational graph, we simply pad meaningless tokens when forming the batches for LINN if the batch size is not fully filled, which causes some redundant computation (detailed implementation can be found in our code). Although LINN is slower than most of the baselines, they are at comparative levels, and the time cost is acceptable. Better implementations, such as parallelized optimization and reducing redundant computation (similar to GRU4Rec), may help improve the efficiency of LINN, which we will explore in the future.

5.3 Weight of Logic Regularizers

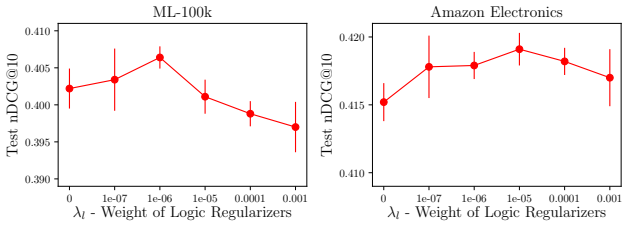


Figure 4: Performance on different logic regularizer weights.

Results of using different weights of logical regularizers verify that logical inference is helpful in making recommendations, as shown in Figure 4. Recommendation tasks can be considered as a logical inference task on the history of user behaviors, since certain user interactions may imply a high probability of interacting with

[4] Codes are provided on GitHub at https://github.com/rutgerswiselab/NLR

Table 5: Results about negative interactions in sequence

	ML-100k		Amazon Electronics	
	nDCG@10	Hit@1	nDCG@10	Hit@1
STAMP	0.3943	0.1706	0.3954	0.2215
with $-\mathbf{v}$	0.3757	0.1651	0.3901	0.2171
with $\mathbf{v}^-$	0.3803	0.1646	0.3936	0.2186
with $g(\mathbf{v})$	0.3745	0.1618	0.3899	0.2173
GRU4Rec	0.3973	0.1745	0.4029	0.2262
with $-\mathbf{v}$	0.3909	0.1741	0.3924	0.2187
with $\mathbf{v}^-$	0.4006	0.1794	0.4050	0.2286
with $g(\mathbf{v})$	0.3843	0.1681	0.3946	0.2206
NARM	*0.4022*	0.1771	*0.4051*	*0.2292*
with $-\mathbf{v}$	0.3917	0.1762	0.3978	0.2234
with $\mathbf{v}^-$	0.3964	*0.1807*	0.4029	0.2262
with $g(\mathbf{v})$	0.3842	0.1743	0.3942	0.2206
LINN	**0.4064***	**0.1850***	**0.4191***	**0.2438***
w/o negative	0.3990	0.1822	0.4172	0.2433

* Significantly better than the best baselines (italic ones) with $p < 0.05$

another item. On the other hand, learning the representations of users and items are more complicated than solving pure logical equations, since the model should have sufficient generalization ability to cope with noisy or even conflicting input expressions. Thus LINN, as an integration of logic inference and neural representation learning, performs well on the recommendation task. The weights of logical regularizers should not be too large because recommendation is not a pure logic reasoning problem but instead a reasoning problem on noisy data. Thus too large logical regularization weights may limit the expressiveness power and lead to a drop in performance.

5.4 Negative Interactions in Sequences

It should be noted that LINN can naturally model negative feedbacks in the user history through the negation operation $\neg$. As far as we know, except for using the negative interactions as the negative training samples, there is few work considering the negative interactions in user interaction sequences. Although considering negative interactions as negative samples helps to learn the item similarities and their relationships, removing them from user's historical sequence is not ideal for modeling user preference, because negative items indeed possess rich information about the user preference. LINN provides a straight-forward way to model the negative interactions in the sequence. For fair comparison, we enhance the STAMP, GRU4Rec, and NARM models in three ways so that they can also take sequences of both positive and negative interactions as inputs:

(1) Use the negative of the item vector $-\mathbf{v}$ instead of $\mathbf{v}$ as the input of RNN or attention networks in STAMP, GRU4Rec and NARM when the item is a negative interaction in the user's interaction sequence.
(2) Build an extra item embedding matrix, so that each item has two vectors $\mathbf{v}$ and $\mathbf{v}^-$ for representing positive and negative interactions on the item, respectively.

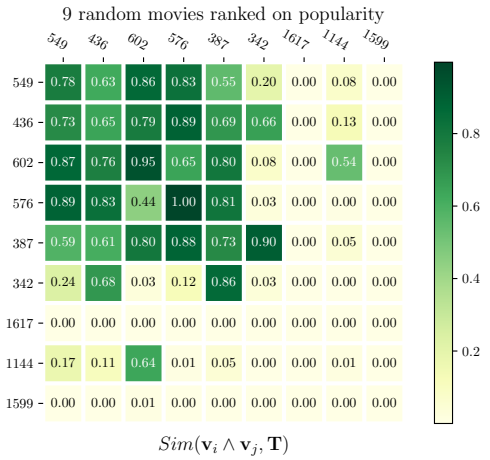


Figure 5: Heatmap of the co-occurrence probability by LINN.

(3) When there comes a negative interaction, we first use a two-layer feed-forward network $g(\cdot)$ (similar as the NOT module in LINN) to transform the item vector $\mathbf{v}$ into its negative representation $g(\mathbf{v})$.

We try our best to tune the parameters, and the results are shown in Table 5. We see that only GRU4Rec with an extra negative embedding matrix can slightly benefit from the negative interactions in the sequence. The reason may be that $-\mathbf{v}$ makes a too strong assumption that positive and negative preferences must have opposite representations in the vector space, while it may not be valid in non-linear neural networks and thus negatively affect the model performance. Using an unconstrained network $g(\cdot)$ only increases the parameter space and does not help in learning the meaning of *negative* interactions. A separate embedding matrix for negative interactions is better than these two methods, but it also weakens the relationship between the two representations of the same item. However, LINN achieves significantly better performance by using logic expressions to model the negative interactions in sequences. Note that even without these negative interactions, LINN still performs much better than GRU4Rec and NARM on Electronics, and achieves significantly better Hit@1 on ML-100k.

5.5 Item Co-occurrence

To better understand what LINN has learned, we randomly select 9 movies in ML-100k and use LINN to predict their co-occurrence (i.e., liked by the same user). The movies are ranked based on their popularity (interaction number) in the training set. More than 30 users interact with the first 3 movies, and less than 5 users interact with the last three. We use LINN to calculate the $Sim(\mathbf{v}_i \wedge \mathbf{v}_j, \mathbf{T})$ of each item pair, which means the probability of liking v_j if a user likes v_i. The results are shown in Figure 5, and we can draw the following conclusions:

- Although we do not design a symmetric network for AND or OR, LINN successfully learns that $\mathbf{v}_i \wedge \mathbf{v}_j$ is close to $\mathbf{v}_j \wedge \mathbf{v}_i$. This is shown by the observation that symmetric positions in the heatmap matrix have similar values.
- LINN learns that popular item pairs have higher probabilities to be consumed by the same user, which corresponds to the top left corner of the heatmap matrix.

- LINN can discover potential relationships between items. For example, item No.602 is a popular musical movie with song and dance, and item No.1144 is an unpopular drama movie, but they belong to the similar type of movies. We see that LINN is able to learn that they have a high probability of being liked by the same user.

Note that we do not use any content or category information of the movies to train the models. It is not easy for models to learn the item relationships solely based on interactions, especially on sparse data, while LINN does well based on the power of deep learning and logic reasoning. Our future work will consider modeling features and knowledge with logic-integrated neural networks, and we believe it has the potential to improve both transparency and effectiveness of deep neural networks.

6 CONCLUSIONS AND FUTURE WORK

In this work, we propose a Logic-Integrated Neural Network (LINN) framework, which introduces logic reasoning into deep neural networks. In particular, we use latent vectors to represent the logic variables, and the logic operations are represented as neural modules regularized by logic rules. The integration of logical inference and neural network enables the model to have the abilities of both representation learning and logical reasoning, which reveals a promising direction to design deep neural networks. Experiments on theoretical task show that LINN works well on logic reasoning problems such as solving logical equations and variables. We further apply LINN to recommendation tasks effortlessly and achieve significant performance, which shows the prospect of LINN on practical tasks.

In this work, we focused on propositional logic reasoning with neural networks, while in the future, we will further explore predicate logic reasoning based on our LINN architecture, which can be easily extended by learning predicate operations as neural modules. We will also explore the possibility of encoding knowledge graph reasoning based on LINN, which will further contribute to the explainable AI research.

7 ACKNOWLEDGEMENT

This work is supported in part by the Rutgers faculty support program, and in part by the National Key Research and Development Program of China (2018YFC0831900), Natural Science Foundation of China (61672311, 61532011), and Tsinghua University Guoqiang Research Institute. The project is also supported in part by China Postdoctoral Science Foundation and Dr. Weizhi Ma has been supported by Shuimu Tsinghua Scholar Program.

REFERENCES

[1] Yoshua Bengio. 2019. From System 1 Deep Learning to System 2 Deep Learning. In *Thirty-third Conference on Neural Information Processing Systems*.
[2] Armin Biere. 2008. PicoSAT essentials. *Journal on Satisfiability, Boolean Modeling and Computation* 4 (2008), 75–97.
[3] Antony Browne and Ron Sun. 2001. Connectionist inference models. *Neural Networks* 14, 10 (2001), 1331–1355.
[4] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.
[5] Ian Cloete and Jacek M Zurada. 2000. Knowledge-based neurocomputing. (2000).
[6] 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.
[7] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. 2019. Neural logic machines. *ICLR* (2019).
[8] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*. ACM, 413–422.
[9] Artur S Garcez, Lus C Lamb, and Dov M Gabbay. 2008. Neural-Symbolic Cognitive Reasoning. (2008).
[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] Alex Graves and Jürgen Schmidhuber. 2005. 2005 Special Issue: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5-6 (2005), 602–610.
[12] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2018. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems*. 2026–2037.
[13] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. *Acm Trans. on Interactive Intelligent Systems (TIIS)* 5, 4 (2016), 19.
[14] John Haugeland. 1989. *Artificial intelligence: The very idea*. MIT press.
[15] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *proceedings of the 25th international conference on world wide web*. 507–517.
[16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. *International Conference on Learning Representations* (2016).
[17] Steffen Hölldobler, Yvonne Kalinke, Fg Wissensverarbeitung Ki, et al. 1994. Towards a new massively parallel computational model for logic programming. In *In ECAI'94 workshop on Combining Symbolic and Connectioninst Processing*.
[18] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. Inferring and Executing Programs for Visual Reasoning. In *ICCV*. IEEE, 3008–3017.
[19] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1746–1751.
[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[21] Yehuda Koren. 2008. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 426–434.
[22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.
[23] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks* 6, 6 (1993), 861–867.
[24] Jing Li, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Tao Lian, and Jun Ma. 2017. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 1419–1428.
[25] 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.
[26] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.
[27] Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5, 4 (1943), 115–133.
[28] Allen Newell. 1980. Physical symbol systems. *Cognitive science* 4, 2 (1980), 135–183.
[29] An-Te Nguyen, Nathalie Denos, and Catherine Berrut. 2007. Improving new user recommendations with rule-based induction on cold user data. In *Proceedings of the 2007 ACM conference on Recommender systems*. 121–128.
[30] J Ross Quinlan. 1991. Knowledge acquisition from structured data: using determinate literals to assist search. *IEEE Expert* 6, 6 (1991), 32–37.
[31] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*.
[32] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.
[33] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. 2019. Learning a SAT solver from single-bit supervision. *In Proceedings of the 7th International Conference on Learning Representations* (2019).
[34] Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*. 2316–2325.
[35] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. 2018. Neural-symbolic vqa: Disentangling reasoning from vision and language understanding. In *NIPS*. 1031–1042.
[36] Yongfeng Zhang and Xu Chen. 2020. Explainable recommendation: A survey and new perspectives. *Foundations and Trends in Information Retrieval* (2020).
[37] Yongfeng Zhang, Guokun Lai, Min Zhang, Yi Zhang, Yiqun Liu, and Shaoping Ma. 2014. Explicit factor models for explainable recommendation based on phrase-level sentiment analysis. In *SIGIR*. 83–92.