Logic Tensor Networks for Top-N Recommendation

Tommaso Carraro[1,2,*], Alessandro Daniele[2], Fabio Aiolli[1] and Luciano Serafini[2]

[1]Department of Mathematics, University of Padova, Via Trieste, 63, 35131 Padova, Italy
[2]Data and Knowledge Management, Fondazione Bruno Kessler, Via Sommarive, 18, 38123 Povo, Italy

Abstract

Despite being studied for more than twenty years, state-of-the-art recommendation systems still suffer from important drawbacks which limit their usage in real-world scenarios. Among the well-known issues of recommender systems, there are data sparsity and the cold-start problem. These limitations can be addressed by providing some background knowledge to the model to compensate for the scarcity of data. Following this intuition, we propose to use Logic Tensor Networks (LTNs) to tackle the top-n item recommendation problem. In particular, we show how LTNs can be used to easily and effectively inject commonsense recommendation knowledge inside a recommender system. We evaluate our method on MindReader, a knowledge graph-based movie recommendation dataset containing plentiful side information. In particular, we perform an experiment to show how the benefits of the knowledge increase with the sparsity of the dataset. Eventually, a comparison with a standard Matrix Factorization approach reveals that our model is able to reach and, in many cases, outperform state-of-the-art performance.

Keywords

recommender systems, top-n recommendation, logic tensor networks, neural-symbolic integration

1. Introduction

Recommender system (RS) technologies are nowadays an essential component for e-services (e.g., Amazon, Netflix, Spotify). Generally speaking, an RS aims at providing suggestions for items (e.g., movies, songs, news) that are most likely of interest to a particular user [1]. Since the first appearance of RSs in early 2000, Collaborative Filtering (CF) [2, 3, 4] has affirmed of being the standard recommendation approach. In particular, Latent Factor models, and especially Matrix Factorization (MF), have dominated the CF scene [5, 6, 7] for years, and this has been further emphasized with the deep learning rise [8, 9, 10, 11, 12].

Despite their success, state-of-the-art models still suffer from important drawbacks, which limit their applicability in real-world scenarios. Among the most crucial problems, there are data sparsity and the cold-start problem [13, 1]. Data sparsity leads to datasets where the density of ratings is usually less than 1%, while cold-start makes the recommendation challenging for new users and items. One way to address these limitations is to provide additional information to the models to compensate for the scarcity of data. Following this intuition, methods based on Tensor

NeSy 2022: 16th International Workshop on Neural-Symbolic Learning and Reasoning, Cumberland Lodge, Windsor, UK
*Corresponding author.
tcarraro@fbk.eu (T. Carraro); daniele@fbk.eu (A. Daniele); aiolli@math.unipd.it (F. Aiolli); serafini@fbk.eu (L. Serafini)
0000-0002-3043-1456 (T. Carraro); 0000-0001-9441-0729 (A. Daniele); 0000-0002-5823-7540 (F. Aiolli); 0000-0003-4812-1031 (L. Serafini)

Factorization [14] and Factorization Machines [15, 16] have been proposed recently. These models allow to effectively extend the user-item matrix by adding new dimensions containing content (e.g., movie genres, demographic information) and/or contextual side information (e.g., location, time). Though these techniques have been shown to improve the recommendation performance, they are usually specifically designed for one type of side information (e.g., the user or item content) and lack explainability [17, 18]. Novel recommendation datasets (e.g., [19]) provide manifold side information (e.g., ratings on movie genres, actors, directors), and hence models which can exploit all the available information are required.

Neural-Symbolic Integration (NeSy) [20] and Statistical Relational Learning (SRL) [21] represent good candidates to incorporate knowledge with learning. These two branches of Artificial Intelligence study approaches for the integration of some form of prior knowledge, usually expressed through First-Order Logic (FOL), with statistical models. The integration has been shown beneficial to address data scarcity [22].

In this paper, we propose to use a Logic Tensor Network (LTN) [23] to inject commonsense knowledge into a standard Matrix Factorization model for the top-n item recommendation task. LTN is a NeSy framework that allows using logical formulas to instruct the learning of a neural model. We propose to use the MindReader dataset [19] to test our model. This dataset includes a variety of information, such as users' tastes across movie genres, actors, and directors. In this work, we show how LTN can naturally and effectively exploit all this various information to improve the generalization capabilities of the MF model. In addition, an experiment that drastically reduces the density of the training ratings reveals that our model can effectively mitigate the sparsity of data, outperforming the standard MF model, especially in the most challenging scenarios.

2. Related works

The integration of logical reasoning and learning in RSs is still in its early stages. Among the NeSy approaches for RSs, the most prominent is NCR [24]. In this work, the recommendation problem is formalized into a logical reasoning problem. In particular, the user's ratings are represented using logical variables, then, logical operators are used to construct formulas that express facts about them. Afterward, NCR maps the variables to *logical* embeddings and the operators to neural networks which act on those embeddings. By doing so, each logical expression can be equivalently organized as a neural network, so that logical reasoning and prediction can be conducted in a continuous space. In [25], the idea of NCR is applied to knowledge graphs for RSs, while [26] uses a NeSy approach to tackle the explainability of RSs.

The seminal approach that successfully applied SRL to RSs has been HyPER [27], which is based on Probabilistic Soft Logic (PSL) [28]. In particular, HyPER exploits the expressiveness of FOL to encode knowledge from a wide range of information sources, such as multiple user and item similarity measures, content, and social information. Then, Hinge-Loss Markov Random Fields are used to learn how to balance the different information types. HyPER is highly related to our work since the logical formulas that we use resemble the ones used in HyPER. After HyPER, other SRL approaches have been proposed for RSs [29, 30].

3. Background

This section provides useful notation and terminology used in the remainder of the paper.

3.1. Notation

Bold notation is used to differentiate between vectors, e.g., $\mathbf{x} = [3.2, 2.1]$, and scalars, e.g., $x = 5$. Matrices and tensors are denoted with upper case bold notation, e.g., $\mathbf{X}$. Then, $\mathbf{X}_i$ is used to denote the i-th row of $\mathbf{X}$, while $\mathbf{X}_{i,j}$ to denote the position at row i and column j. We refer to the set of users of a RS with $\mathcal{U}$, where $|\mathcal{U}| = n$. Similarly, the set of items is referred to as $\mathcal{I}$ such that $|\mathcal{I}| = m$. We use $\mathcal{D}$ to denote a dataset. $\mathcal{D}$ is defined as a set of N triples $\mathcal{D} = \{(u, i, r)^{(j)}\}_{j=1}^{N}$, where $u \in \mathcal{U}$, $i \in \mathcal{I}$, and $r \in \mathbb{N}$ is a rating. We assume that a user u cannot give more than one rating to an item i, namely $\nexists r_1, r_2 \in \mathbb{N}, r_1 \neq r_2 : \{(u, i, r_1)\} \cup \{(u, i, r_2)\} \subseteq \mathcal{D}$. $\mathcal{D}$ can be reorganized in the so-called user-item matrix $\mathbf{R} \in \mathbb{N}^{n \times m}$, where users are on the rows and items on the columns, such that $\mathbf{R}_{u,i} = r$ if $(u, i, r) \in \mathcal{D}$, 0 otherwise.

3.2. Matrix Factorization

Matrix Factorization (MF) is a Latent Factor Model that aims at factorizing the user-item matrix $\mathbf{R}$ into the product of two lower-dimensional rectangular matrices, denoted as $\mathbf{U}$ and $\mathbf{I}$. $\mathbf{U} \in \mathbb{R}^{n \times k}$ and $\mathbf{I} \in \mathbb{R}^{m \times k}$ are matrices containing the users' and items' latent factors, respectively, where k is the number of latent factors. The objective of MF is to find $\mathbf{U}$ and $\mathbf{I}$ such that $\mathbf{R} \approx \mathbf{U} \cdot \mathbf{I}^\top$. An effective way to learn the latent factors is by using gradient-descent optimization. Given the dataset $\mathcal{D}$, a MF model seeks to minimize the following loss function:

$$\boldsymbol{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(u,i,r) \in \mathcal{D}} ||\tilde{r} - r||^2 + \lambda ||\boldsymbol{\theta}||^2 \tag{1}$$

where $\tilde{r} = \mathbf{U}_u \cdot \mathbf{I}_i^\top$ and $\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{I}\}$. The first term of Equation (1) is the Mean Squared Error (MSE) between the predicted and target ratings, while the second one is an $L2$ regularization term. λ is an hyper-parameter to set strength of the regularization.

3.3. Logic Tensor Networks

Logic Tensor Networks [23] (LTNs) are a Neural-Symbolic framework that enables effective integration of deep learning and logical reasoning. It allows to define a knowledge base composed of a set of logical axioms and to use them as the objective of a neural model. To define the knowledge base, LTN uses a specific first-order language, called Real Logic, which forms the basis of the framework. It is fully differentiable and has a concrete semantics that allows mapping every symbolic expression into the domain of real numbers. Thanks to Real Logic, LTN can convert logical formulas into computational graphs that enable gradient-based optimization based on fuzzy logic semantics.

Real Logic is defined on a first-order language $\mathscr{L}$ with a signature that contains a set $\mathscr{C}$ of constant symbols, a set $\mathscr{X}$ of variable symbols, a set $\mathscr{F}$ of functional symbols, and a set $\mathscr{P}$ of predicate symbols. A term is constructed recursively from constants, variables, and functional symbols. An expression formed by applying a predicate symbol to some term(s) is

called an atomic formula. Complex formulas are constructed recursively using connectives (i.e., $\neg, \wedge, \vee, \implies, \leftrightarrow$) and quantifiers (i.e., $\forall, \exists$).

To emphasize the fact that symbols are grounded onto real-valued features, we use the term *grounding*[1], denoted by $\mathcal{G}$. In particular, each individual is grounded as a tensor of real features, functions as real functions, and predicates as real functions that specifically project onto a value in the interval $[0, 1]$. A variable x is grounded to a *sequence* of n_x individuals from a domain, with $n_x \in \mathbb{N}^+, n_x > 0$. As a consequence, a term $t(x)$ or a formula $\mathrm{P}(x)$, constructed recursively with a free variable x, will be grounded to a sequence of n_x values too. Afterward, connectives are grounded using fuzzy semantics, while quantifiers using special aggregation functions. In this paper, we use the *product configuration*, which is better suited for gradient-based optimization [31]. Specifically, conjunctions are grounded using the product t-norm T_{prod}, negations using the standard fuzzy negation N_S, implications using the Reichenbach implication I_R, and the universal quantifier using the generalized mean w.r.t the error values ME_p. The other connectives and quantifiers are not used in this paper, hence not reported.

$$
\begin{aligned}
\mathrm{T}_{prod}(u, v) &= u * v,\ u, v \in [0, 1] \\
\mathrm{I}_R(u, v) &= 1 - u + u * v,\ u, v \in [0, 1] \\
\mathrm{N}_S(u) &= 1 - u,\ u \in [0, 1] \\
\mathrm{ME}_p(u_1, \dots, u_n) &= 1 - (\frac{1}{n} \sum_{i=1}^{n} (1 - u_i)^p)^{\frac{1}{p}}, p \geq 1, u_1, \dots, u_n \in [0, 1]
\end{aligned}
$$

Connective operators are applied element-wise to the tensors in input, while aggregators aggregate the dimension of the tensor in input that corresponds to the quantified variable. Real Logic provides also a special type of quantification, called diagonal quantification, denoted as $\mathrm{Diag}(x_1, \dots, x_n)$. It applies only to variables that have the same number of individuals (i.e., $n_{x_1} = n_{x_2} = \cdots = n_{x_n}$) and allows to quantify over specific tuples of individuals, such that the i-th tuple contains the i-th individual of each of the variables in the argument of Diag. An intuition about how these operations work in practice is given in Appendix D.

Given a Real Logic knowledge base $\mathcal{K} = \{\phi_1, \dots, \phi_n\}$, where $\phi_1, \dots, \phi_n$ are closed formulas, LTN allows to learn the grounding of constants, functions, and predicates appearing in them. In particular, if constants are grounded as embeddings, and functions/predicates onto neural networks, their grounding $\mathcal{G}$ depends on some learnable parameters $\boldsymbol{\theta}$. We denote a parametric grounding as $\mathcal{G}(\cdot|\boldsymbol{\theta})$. In LTN, the learning of parametric groundings is obtained by finding parameters $\boldsymbol{\theta}^*$ that maximize the satisfaction of $\mathcal{K}$:

$$
\boldsymbol{\theta}^* = \mathrm{argmax}_{\boldsymbol{\theta}} \mathrm{SatAgg}_{\phi \in \mathcal{K}} \mathcal{G}(\phi|\boldsymbol{\theta}) \tag{2}
$$

where, $\mathrm{SatAgg} : [0, 1]^* \mapsto [0, 1]$ is a formula aggregating operator, often defined using ME_p.

Because Real Logic grounds expressions in real and continuous domains, LTN attaches gradients to every sub-expression and consequently learns through gradient-descent optimization.

[1]Notice that this is different from the common use of the term *grounding* in logic, which indicates the operation of replacing the variables of a term or formula with constants or terms containing no variables. To avoid confusion, we use the synonym *instantiation* for this purpose.

4. Method

Our approach uses a Logic Tensor Network to train a basic Matrix Factorization (MF) model for the top-n item recommendation task. The LTN is trained using a Real Logic knowledge base containing commonsense knowledge facts about the movie recommendation domain. This section formalizes the knowledge base used by our model, how the symbols appearing in it are grounded in the real field, and how the learning of the LTN takes place.

4.1. Knowledge base

The Real Logic knowledge base that our model seeks to maximally satisfy is composed of the following axioms.

$$\phi_1 : \forall \text{Diag}(user, movie, rating)(\text{Sim}(\text{Likes}(user, movie), rating)) \tag{3}$$

$$\begin{aligned}\phi_2 : \forall(user, movie, genre)(&\neg \text{LikesGenre}(user, genre) \wedge \text{HasGenre}(movie, genre) \\ \implies &\text{Sim}(\text{Likes}(user, movie), rating\_))\end{aligned} \tag{4}$$

where *user*, *movie*, *rating*, and *genre* are variable symbols to denote the users of the system, the items of the system, the ratings given by the users to the items, and the genres of the movies, respectively. *rating*_ is a constant symbol denoting the negative rating. Likes(u, m) is a functional symbol returning the prediction for the rating given by user u to movie m. Sim(r_1, r_2) is a predicate symbol measuring the similarity between two ratings, r_1 and r_2. LikesGenre(u, g) is a predicate symbol denoting whether the user u likes the genre g. HasGenre(m, g) is a predicate symbol denoting whether the movie m belongs to the genre g.

Notice the use of the diagonal quantification on Axiom (3). When *user*, *movie*, and *rating* are grounded with three sequences of values, the i-th value of each variable matches with the values of the other variables. This is useful in this case since the dataset $\mathscr{D}$ comes as a set of triples. Diagonal quantification allows forcing the satisfaction of Axiom (3) for these triples only, rather than any combination of users, items, and ratings in $\mathscr{D}$.

4.2. Grounding of the knowledge base

The grounding allows to define how the symbols of the language are mapped onto the real field, and hence how they can be used to construct the architecture of the LTN. In particular, given $\mathscr{D} = \{(u, m, r)\}_{j=1}^{N}$, $\mathscr{G}(user) = \langle u^{(j)} \rangle_{j=1}^{N}$, namely *user* is grounded as a sequence of the N user indexes in $\mathscr{D}$. $\mathscr{G}(movie) = \langle m^{(j)} \rangle_{j=1}^{N}$, namely *movie* is grounded as a sequence of the N movie indexes in $\mathscr{D}$. $\mathscr{G}(rating) = \langle r^{(j)} \rangle_{j=1}^{N}$ with $r^{(j)} \in \{0, 1\}\ \forall j$, namely *rating* is grounded as a sequence of the N ratings in $\mathscr{D}$, where 0 denotes a negative rating and 1 a positive one. $\mathscr{G}(rating\_) = 0$, namely *rating*_ is grounded as the negative rating. $\mathscr{G}(genre) = \langle 1, \ldots, N_g \rangle$, namely *genre* is grounded as a sequence of N_g genre indexes, where N_g is the number of genres appearing in the movies of $\mathscr{D}$. $\mathscr{G}(\text{Likes} \mid \mathbf{U}, \mathbf{I}) : u, m \mapsto \mathbf{U}_u \cdot \mathbf{I}_m^\top$, namely Likes is grounded onto a function that takes as input a user index u and a movie index m and returns the prediction of the MF model for user at index u and movie at index m, where $\mathbf{U} \in \mathbb{R}^{n \times k}$ and $\mathbf{I}^{m \times k}$ are the matrices of the users'

and items' latent factors, respectively. $\mathcal{G}(\text{LikesGenre}) : u, g \mapsto \{0, 1\}$, namely LikesGenre is grounded onto a function that takes as input a user index u and a genre index g and returns 1 if the user u likes the genre g in the dataset, 0 otherwise. Similarly, $\mathcal{G}(\mathit{HasGenre}) : m, g \mapsto \{0, 1\}$, namely HasGenre is grounded onto a function that takes as input a movie index m and a genre index g and returns 1 if the movie m belongs to genre g in the dataset, 0 otherwise. Finally, $\mathcal{G}(\text{Sim}) : \tilde{r}, r \mapsto \exp(-\alpha||\tilde{r} - r||^2)$, namely Sim is grounded onto a function that computes the similarity between a predicted rating $\tilde{r}$ and a target rating r. The use of the exponential allows to treat Sim as a predicate since the output is restricted in the interval $[0, 1]$. The squared is used to give more penalty to larger errors in the optimization. α is an hyper-parameter to change the smoothness of the function.

Intuitively, Axiom (3) states that for each user-movie-rating triple in the dataset $\mathcal{D} = \{(u, m, r)^{(j)}\}_{j=1}^{N}$, the prediction computed by the MF model for the user u and movie m should be similar to the target rating r provided by the user u for the movie m. Instead, Axiom (4) states that for each possible combination of users, movies, and genres, taken from the dataset, if the user u does not like a genre of the movie m, then the prediction computed by the MF model for the user u and movie m should be similar to the negative rating $rating\_$, namely the user should not like the movie m. By forcing the satisfaction of Axiom (3), the model learns to factorize the user-item matrix using the ground truth, while Axiom (4) acts as a kind of regularization for the latent factors of the MF model.

4.3. Learning of the LTN

The objective of our LTN is to learn the latent factors in $\mathbf{U}$ and $\mathbf{I}$ such that the axioms in the knowledge base $\mathcal{K} = \{\phi_1, \phi_2\}$ are maximally satisfied, namely $\text{argmax}_{\boldsymbol{\theta}} \text{SatAgg}_{\phi \in \mathcal{K}} \mathcal{G}_{(user,movie,rating) \leftarrow \mathcal{D}}(\phi|\boldsymbol{\theta})$[2], where $\boldsymbol{\theta} = \{\mathbf{U}, \mathbf{I}\}$. In practice, this objective corresponds to the following loss function:

$$\boldsymbol{L}(\boldsymbol{\theta}) = (1 - \text{SatAgg}_{\phi \in \mathcal{K}} \mathcal{G}_{(user,movie,rating) \leftarrow \mathcal{B}}(\phi|\boldsymbol{\theta})) + \lambda||\boldsymbol{\theta}||^2 \tag{5}$$

where $\mathcal{B}$ denotes a batch of training triples randomly sampled from $\mathcal{D}$. An $L2$ regularization term has been added to the loss to prevent overfitting. Hyper-parameter λ is used to define the strength of the regularization. Notice that the loss does not specify how the variable *genre* is grounded. Its grounding depends on the sampled batch $\mathcal{B}$. In our experiments, we grounded it with the sequence of genres of the movies in the batch.

It is worth highlighting that the loss function depends on the semantics used to approximate the logical connectives, quantifiers, and formula aggregating operator. In our experiments, we used the *stable product configuration*, a stable version of the product configuration introduced in [23]. Then, we selected ME_p as formula aggregating operator, with $p = 2$.

[2]In the notation, $(user, movie, rating) \leftarrow \mathcal{D}$ means that variables *user*, *movie*, and *rating* are grounded with the triples taken from the dataset $\mathcal{D}$, namely *user* takes the sequence of user indexes, *movie* the sequence of movie indexes, and *rating* the sequence of ratings.

5. Experiments

This section presents the experiments we have performed with our method. They have been executed on an Apple MacBook Pro (2019) with a 2,6 GHz 6-Core Intel Core i7. The model has been implemented in Python using PyTorch. In particular, we used the `LTNtorch`[3] library. Our source code is available at URL[4].

5.1. Dataset

In our experiments, we used the MindReader [19] dataset. It contains 102,160 explicit ratings collected from 1,174 real users on 10,030 entities (e.g., movies, actors, movie genres) taken from a knowledge graph in the movie domain. The explicit ratings in the dataset can be of three types: *like* (1), *dislike* (−1), or *unknown* (0). The dataset is subdivided in 10 splits. In our experiments, we used split 0. Each split has a training set, a validation set, and a test set. The training set contains both ratings given on movies and on the other entities, while validation and test sets contain only ratings given on movies. The validation and test sets are built in such a way to perform a *leave-one-out* evaluation. In particular, for each user of the training set, one random positive movie rating is held out for the validation set, and one for the test set. The validation/test example of the user is completed by adding 100 randomly sampled negative movie ratings from the dataset. To improve the quality of the dataset, we removed the unknown ratings. Moreover, we removed the top 2% of popular movies from the test set to see how the model performs on non-trivial recommendations, as suggested in [19]. Afterward, we considered only the training ratings given on movies and movie genres since our model uses only this information. After these steps, we converted the negative ratings from -1 to 0. Our final dataset contains 962 users, 3,034 movies, 164 genres, 16,351 ratings on movies, and 10,889 ratings on movie genres. The density of the user-movie ratings is 0.37%.

5.2. Experimental setting

In our experiments, we compared the performance of three models: (1) a standard MF model trained on the movie ratings of MindReader using Equation (1), denoted as MF, (2) a LTN model trained on the movie ratings of MindReader using Equation (5) with $\mathscr{K} = \{\phi_1\}$, denoted as LTN, and (3) a LTN model trained on the movie and genre ratings of MindReader using Equation (5) with $\mathscr{K} = \{\phi_1, \phi_2\}$, denoted as $\text{LTN}_{\text{genres}}$. To compare the performance of the models, we used two widely used ranking-based metrics, namely `hit@k` and `ndcg@k`, explained in Appendix A. In our experiments, we used the following procedure: (1) we generated additional training sets by randomly sampling the 80%, 60%, 40%, and 20% of the movie ratings of each user from the entire training set, referred to as 100%. Then, (2) for each training set $Tr \in \{100\%, 80\%, 60\%, 40\%, 20\%\}$ and for each model $m \in \{\text{MF}, \text{LTN}, \text{LTN}_{\text{genres}}\}$: (2*a*) we performed a grid search of model m on training set Tr to find the best hyper-parameters on the validation set using `hit@10` as validation metric; then, (2*b*) we tested the performance of the best model on the test set in terms of `hit@10` and `ndcg@10`. We repeated this procedure 30 times using seeds from 0 to 29. The test metrics

[3] https://github.com/logictensornetworks/LTNtorch
[4] https://github.com/tommasocarraro/LTNrec

have been averaged across these runs and reported in Table 1. Due to computational time, the grid search has been computed only for the first run. Starting from the second run, step (2*a*) is replaced with the training of model *m* on the training set *Tr* with the best hyper-parameters found during the first run. A description of the hyper-parameters tested in the grid searches as well as the training details of the models is explained in Appendix B.

6. Results

A comparison between MF, LTN, and $\text{LTN}_{\text{genres}}$ is reported in Table 1. The table reports the performance of the three models on a variety of tasks with different sparsity.

Table 1
Test `hit@10` and `ndcg@10` averaged across 30 runs. Standard deviations are between brackets.

% of training ratings	Metric	MF	LTN	$\text{LTN}_{\text{genres}}$
100%	`hit@10`	$0.4499_{(0.0067)}$	$0.4636_{(0.0040)}$	$\mathbf{0.4642}_{(0.0054)}$
	`ndcg@10`	$0.1884_{(0.0028)}$	$0.1899_{(0.0014)}$	$\mathbf{0.1905}_{(0.0022)}$
80%	`hit@10`	$0.4459_{(0.0057)}$	$0.4585_{(0.0066)}$	$\mathbf{0.4616}_{(0.0069)}$
	`ndcg@10`	$0.1864_{(0.0023)}$	$0.1881_{(0.0023)}$	$\mathbf{0.1894}_{(0.0025)}$
60%	`hit@10`	$0.4274_{(0.0107)}$	$0.4475_{(0.0087)}$	$\mathbf{0.4487}_{(0.0080)}$
	`ndcg@10`	$0.1798_{(0.0039)}$	$0.1853_{(0.0034)}$	$\mathbf{0.1862}_{(0.0031)}$
40%	`hit@10`	$0.3983_{(0.0105)}$	$0.4087_{(0.0117)}$	$\mathbf{0.4322}_{(0.0102)}$
	`ndcg@10`	$0.1692_{(0.0047)}$	$0.1726_{(0.0052)}$	$\mathbf{0.1807}_{(0.0049)}$
20%	`hit@10`	$0.2956_{(0.0196)}$	$\mathbf{0.3764}_{(0.0170)}$	$0.3761_{(0.0160)}$
	`ndcg@10`	$0.1367_{(0.0093)}$	$0.1594_{(0.0069)}$	$\mathbf{0.1598}_{(0.0068)}$

By looking at the table, it is possible to observe that LTN outperforms MF in all the five tasks. In particular, for the dataset with 20% of training ratings, the improvement is drastic (27.33% on `hit@10`). We want to emphasize that the two models only differ in the loss function. This demonstrates that the loss based on fuzzy logic semantics of LTN is beneficial to deal with the sparsity of data. Then, with the addition of knowledge regarding the users' tastes across the movie genres, it is possible to further improve the results, as shown in the last column of the table. $\text{LTN}_{\text{genres}}$ outperforms the other models on almost all the tasks. For the dataset with the 20% of the ratings, the `hit@10` of $\text{LTN}_{\text{genres}}$ is slightly worse compared to LTN. This could be related to the quality of the training ratings sampled from the original dataset. This is also suggested by the higher standard deviation associated with the datasets with higher sparsity. For considerations about the training times of the models refer to Appendix C.

7. Conclusions

In this paper, we proposed to use Logic Tensor Networks to tackle the top-n recommendation task. We showed how, by design, LTN permits to easily integrate side information inside a recommendation model. We compared our LTN models with a standard MF model, in a variety of tasks with different sparsity, showing the benefits provided by the background knowledge, especially when the task is challenging due to data scarcity.

References

[1] F. Ricci, L. Rokach, B. Shapira, Recommender Systems: Introduction and Challenges, Springer US, Boston, MA, 2015, pp. 1–34. URL: https://doi.org/10.1007/978-1-4899-7637-6_1. doi:10.1007/978-1-4899-7637-6_1.

[2] X. Su, T. M. Khoshgoftaar, A survey of collaborative filtering techniques, Adv. in Artif. Intell. 2009 (2009). URL: https://doi.org/10.1155/2009/421425. doi:10.1155/2009/421425.

[3] Y. Koren, R. Bell, Advances in Collaborative Filtering, Springer, Boston, MA, 2011, pp. 145–186. URL: https://doi.org/10.1007/978-0-387-85820-3_5. doi:10.1007/978-0-387-85820-3_5.

[4] F. Aiolli, Efficient top-n recommendation for very large scale binary rated datasets, in: Proceedings of the 7th ACM Conference on Recommender Systems, RecSys '13, Association for Computing Machinery, New York, NY, USA, 2013, p. 273–280. URL: https://doi.org/10.1145/2507157.2507189. doi:10.1145/2507157.2507189.

[5] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 263–272. doi:10.1109/ICDM.2008.22.

[6] X. Ning, G. Karypis, Slim: Sparse linear methods for top-n recommender systems, in: 2011 IEEE 11th International Conference on Data Mining, 2011, pp. 497–506. doi:10.1109/ICDM.2011.134.

[7] M. Polato, F. Aiolli, Boolean kernels for collaborative filtering in top-n item recommendation, Neurocomput. 286 (2018) 214–225. URL: https://doi.org/10.1016/j.neucom.2018.01.057. doi:10.1016/j.neucom.2018.01.057.

[8] D. Liang, R. G. Krishnan, M. D. Hoffman, T. Jebara, Variational autoencoders for collaborative filtering, in: Proceedings of the 2018 World Wide Web Conference, WWW '18, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2018, p. 689–698. URL: https://doi.org/10.1145/3178876.3186150. doi:10.1145/3178876.3186150.

[9] I. Shenbin, A. Alekseev, E. Tutubalina, V. Malykh, S. I. Nikolenko, RecVAE: A new variational autoencoder for top-n recommendations with implicit feedback, in: Proceedings of the 13th International Conference on Web Search and Data Mining, ACM, 2020. URL: https://doi.org/10.1145%2F3336191.3371831. doi:10.1145/3336191.3371831.

[10] H. Steck, Embarrassingly shallow autoencoders for sparse data, in: The World Wide Web Conference, WWW '19, Association for Computing Machinery, New York, NY, USA, 2019, p. 3251–3257. URL: https://doi.org/10.1145/3308558.3313710. doi:10.1145/3308558.3313710.

[11] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, T.-S. Chua, Neural collaborative filtering, in: Proceedings of the 26th International Conference on World Wide Web, WWW '17, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2017, p. 173–182. URL: https://doi.org/10.1145/3038912.3052569. doi:10.1145/3038912.3052569.

[12] T. Carraro, M. Polato, F. Aiolli, Conditioned variational autoencoder for top-n item recommendation, 2020. URL: https://arxiv.org/abs/2004.11141. doi:10.48550/ARXIV.2004.11141.

[13] M. Polato, F. Aiolli, Exploiting sparsity to build efficient kernel based collaborative filtering for top-n item recommendation, Neurocomputing 268 (2017) 17–26. URL: https://www.sciencedirect.com/science/article/pii/S0925231217307592. doi:https://doi.org/10.1016/j.neucom.2016.12.090, advances in artificial neural networks, machine learning and computational intelligence.

[14] P. Bhargava, T. Phan, J. Zhou, J. Lee, Who, what, when, and where: Multi-dimensional collaborative recommendations using tensor factorization on sparse user-generated data, in: Proceedings of the 24th International Conference on World Wide Web, WWW '15, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2015, p. 130–140. URL: https://doi.org/10.1145/2736277.2741077. doi:10.1145/2736277.2741077.

[15] S. Rendle, Factorization machines, in: 2010 IEEE International Conference on Data Mining, 2010, pp. 995–1000. doi:10.1109/ICDM.2010.127.

[16] X. Xin, B. Chen, X. He, D. Wang, Y. Ding, J. Jose, Cfm: Convolutional factorization machines for context-aware recommendation, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 3926–3932. URL: https://doi.org/10.24963/ijcai.2019/545. doi:10.24963/ijcai.2019/545.

[17] Y. Zhang, X. Chen, Explainable recommendation: A survey and new perspectives, Foundations and Trends® in Information Retrieval 14 (2020) 1–101. URL: https://doi.org/10.1561%2F1500000066. doi:10.1561/1500000066.

[18] T. Carraro, M. Polato, F. Aiolli, A look inside the black-box: Towards the interpretability of conditioned variational autoencoder for collaborative filtering, in: Adjunct Publication of the 28th ACM Conference on User Modeling, Adaptation and Personalization, UMAP '20 Adjunct, Association for Computing Machinery, New York, NY, USA, 2020, p. 233–236. URL: https://doi.org/10.1145/3386392.3399305. doi:10.1145/3386392.3399305.

[19] A. H. Brams, A. L. Jakobsen, T. E. Jendal, M. Lissandrini, P. Dolog, K. Hose, Mindreader: Recommendation over knowledge graph entities with explicit user ratings, CIKM '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 2975–2982. URL: https://doi.org/10.1145/3340531.3412759. doi:10.1145/3340531.3412759.

[20] T. R. Besold, A. d. Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kuehnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, G. Zaverucha, Neural-symbolic learning and reasoning: A survey and interpretation, 2017. URL: https://arxiv.org/abs/1711.03902. doi:10.48550/ARXIV.1711.03902.

[21] L. D. Raedt, K. Kersting, Statistical Relational Learning, Springer US, Boston, MA, 2010, pp. 916–924. URL: https://doi.org/10.1007/978-0-387-30164-8_786. doi:10.1007/978-0-387-30164-8_786.

[22] A. Daniele, L. Serafini, Neural networks enhancement with logical knowledge, 2020. URL: https://arxiv.org/abs/2009.06087. doi:10.48550/ARXIV.2009.06087.

[23] S. Badreddine, A. d'Avila Garcez, L. Serafini, M. Spranger, Logic tensor networks, Artificial Intelligence 303 (2022) 103649. URL: https://doi.org/10.1016%2Fj.artint.2021.103649. doi:10.1016/j.artint.2021.103649.

[24] H. Chen, S. Shi, Y. Li, Y. Zhang, Neural collaborative reasoning, in: Proceedings of the Web Conference 2021, ACM, 2021. URL: https://doi.org/10.1145%2F3442381.3449973.

doi:10.1145/3442381.3449973.

[25] H. Chen, Y. Li, S. Shi, S. Liu, H. Zhu, Y. Zhang, Graph collaborative reasoning, in: Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining, WSDM '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 75–84. URL: https://doi.org/10.1145/3488560.3498410. doi:10.1145/3488560.3498410.

[26] Y. Xian, Z. Fu, H. Zhao, Y. Ge, X. Chen, Q. Huang, S. Geng, Z. Qin, G. de Melo, S. Muthukrishnan, Y. Zhang, Cafe: Coarse-to-fine neural symbolic reasoning for explainable recommendation, in: Proceedings of the 29th ACM International Conference on Information Knowledge Management, CIKM '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1645–1654. URL: https://doi.org/10.1145/3340531.3412038. doi:10.1145/3340531.3412038.

[27] P. Kouki, S. Fakhraei, J. Foulds, M. Eirinaki, L. Getoor, Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems, RecSys '15, Association for Computing Machinery, New York, NY, USA, 2015, p. 99–106. URL: https://doi.org/10.1145/2792838.2800175. doi:10.1145/2792838.2800175.

[28] A. Kimmig, S. Bach, M. Broecheler, B. Huang, L. Getoor, A short introduction to probabilistic soft logic, Mansinghka, Vikash, 2012, pp. 1–4. URL: https://lirias.kuleuven.be/retrieve/204697.

[29] R. Catherine, W. Cohen, Personalized recommendations using knowledge graphs: A probabilistic logic programming approach, in: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 325–332. URL: https://doi.org/10.1145/2959100.2959131. doi:10.1145/2959100.2959131.

[30] M. Gridach, Hybrid deep neural networks for recommender systems, Neurocomputing 413 (2020) 23–30. URL: https://www.sciencedirect.com/science/article/pii/S0925231220309966. doi:https://doi.org/10.1016/j.neucom.2020.06.025.

[31] E. van Krieken, E. Acar, F. van Harmelen, Analyzing differentiable fuzzy logic operators, Artificial Intelligence 302 (2022) 103602. URL: https://doi.org/10.1016%2Fj.artint.2021.103602. doi:10.1016/j.artint.2021.103602.

A. Metrics

To validate and test our models, we selected two widely used ranking-based metrics, namely `hit@k` and `ndcg@k`. They are defined as follows:

- `hit@k`: Hit Ratio measures whether a testing item is placed in the top-k positions of the ranking, considering the presence of an item as a hit;
- `ndcg@k`: Normalized Discounted Cumulative Gain measures the quality of the recommendation based on the position of the target item in the ranking. In particular, it uses a monotonically increasing discount to emphasize the importance of higher ranks versus lower ones.

Formally, let us define $\omega(r)$ as the item at rank r, $\mathbb{I}[\cdot]$ as the indicator function, and I_u as the set of held-out items for user u. `hit@k` for user u is defined as

$$\text{hit}\,@k(u,\omega) := \mathbb{I}\left[\left(\sum_{r=1}^{k}\mathbb{I}\left[\omega(r)\in I_u\right]\right)\geq 1\right].$$

Truncated discounted cumulative gain (`dcg@k`) for user u is defined as

$$\text{dcg@}k(u,\omega) := \sum_{r=1}^{k}\frac{2^{\mathbb{I}[\omega(r)\in I_u]}-1}{\log(r+1)}.$$

`ndcg@k` is the `dcg@k` linearly normalized to $[0,1]$ after dividing by the best possible `dcg@k`, where all the held-out items are ranked at the top. Notice that in this paper $|I_u| = 1$.

B. Training details

The hyper-parameters tested during the grid searches explained in Section 5.2 vary depending on the model. For all the models, we tried a number of latent factors $k \in \{1, 5, 10, 25\}$, regularization coefficient $\lambda \in \{0.001, 0.0001\}$, batch size in $\{32, 64\}$, and whether it was better to add users' and items' biases to the model. For LTN and $\text{LTN}_{\text{genres}}$, we tried $\alpha \in \{0.05, 0.1, 0.2\}$ for the predicate Sim and used $p = 2$ for the aggregator ME_p of Axiom (3). For $\text{LTN}_{\text{genres}}$, we tried $p \in \{2, 5\}$ for the aggregator ME_p of Axiom (4). Notice that $\lim_{p\to\infty}\text{ME}_p(u_1,\dots,u_n) = \min\{u_1,\dots,u_n\}$. Intuitively, p offers flexibility to account for outliers in the data. The higher the p, the more focus the model will have on the outliers.

For all the models, the latent factors $\mathbf{U}$ and $\mathbf{I}$, for users and items, respectively, have been randomly initialized using the *Glorot* initialization, while the biases with values sampled from a normal distribution with 0 mean and unitary variance. All the models have been trained for 200 epochs by using the Adam optimizer with a learning rate of 0.001. For each training, we used early stopping to stop the learning if after 20 epochs no improvements were found on the validation metric (i.e., `hit@10`).

C. Training time

A comparison of the training times required by the models on the different datasets is presented in Table 2. The models have been trained for 200 epochs with a learning rate of 0.001, batch size of 64, one latent factor (i.e., $k = 1$), without bias terms, and without early stopping. The other hyper-parameters do not affect training time. In particular, LTN_{genres} increases the time complexity considerably. This is due to Axiom 4, which has to be evaluated for each possible combination of users, items, and genres. This drawback can limit the application of LTN_{genres} in datasets with a higher number of users and items. However, it is possible to boost training time using GPUs or by designing logical axioms which make use of diagonal quantification.

Table 2
Training time in seconds.

% of training ratings	MF	LTN	LTN_{genres}
100%	26.99	50.87	247.30
80%	22.52	37.79	213.62
60%	18.31	28.97	145.86
40%	15.60	20.09	97.43
20%	8.12	10.68	50.85

D. Intuition of Real Logic grounding

In Real Logic, differently from first-order logic, a variable x is grounded as a *sequence* of n_x individuals (i.e., tensors) from a domain, with $n_x \in \mathbb{N}^+, n_x > 0$. As a direct consequence, a term $t(x)$ or a formula $\mathrm{P}(x)$, with a free variable x, is grounded to a sequence of n_x values too. For example, $\mathrm{P}(x)$ returns a vector in $[0,1]^{n_x}$, namely $\langle \mathrm{P}(x_i) \rangle_{i=1}^{n_x}$, where x_i is the i-th individual of x. Similarly, $t(y)$ returns a matrix in $\mathbb{R}^{n_y \times z}$, assuming that t maps to individuals in $\mathbb{R}^z$. This formalization is intuitively extended to terms and formulas with arity greater than one. In such cases, Real Logic organizes the output tensor in such a way that it has a dimension for each free variable involved in the expression. For instance, $t_2(x,y)$ returns a tensor in $\mathbb{R}^{n_x \times n_y \times z}$, assuming that t_2 maps to individuals in $\mathbb{R}^z$. In particular, at position (i,j) there is the evaluation of $t_2(x_i, y_j)$, where x_i denotes the i-th individual of x and y_j the j-th individual of y. Similarly, $\mathrm{P}_2(x,y)$ returns a tensor in $[0,1]^{n_x \times n_y}$, where at position (i,j) there is the evaluation of $\mathrm{P}(x_i, y_j)$.

The connective operators are applied element-wise to the tensors in input. For instance, $\neg \mathrm{P}_2(x,y)$ returns a tensor in $[0,1]^{n_x \times n_y}$, where at position (i,j) there is the evaluation of $\neg \mathrm{P}_2(x_i, y_j)$, namely N_S (i.e., $\neg$) is applied to each truth value in the tensor $\mathrm{P}_2(x,y) \in [0,1]^{n_x \times n_y}$. For binary connectives, the behavior is similar. For instance, let Q be a predicate symbol and u a variable. Then, $\mathrm{P}_2(x,y) \wedge \mathrm{Q}(x,u)$ returns a tensor in $[0,1]^{n_x \times n_y \times n_u}$, where at position (i,j,k) there is the evaluation of the formula on the i-th individual of x, j-th individual of y, and k-th individual of u.

The quantifiers aggregate the dimension that corresponds to the quantified variable. For instance, $\forall x \mathrm{P}_2(x,y)$ returns a tensor in $[0,1]^{n_y}$, namely the aggregation is performed across the dimension of x. Since y is the only free variable remaining in the expression, the output has

one single dimension, corresponding to the dimension of y. Specifically, the framework computes $\mathrm{P}_2(x, y) \in [0, 1]^{n_x \times n_y}$ first, then it aggregates the dimension corresponding to x. Similarly, $\forall(x, y)\, \mathrm{P}_2(x, y)$ returns a scalar in $[0, 1]$, namely the aggregation is performed across the dimensions of both variables x and y. In the case of diagonal quantification, the framework behaves differently. For instance, $\forall\, \mathrm{Diag}(w, v)\, \mathrm{P}_2(w, v)$, where w and v are two variables with the same number of individuals $n_w = n_v$, returns a scalar in $[0, 1]$, which is the result of the aggregation of n_w truth values, namely $\mathrm{P}_2(w_1, v_1), \mathrm{P}_2(w_2, v_2), \ldots, \mathrm{P}_2(w_{n_w}, v_{n_v})$. Without diagonal quantification (i.e., $\forall(w, v)\, \mathrm{P}_2(w, v)$), the framework performs an aggregation across the dimensions of both variables, involving n_w^2 values, namely $\mathrm{P}_2(w_1, v_1), \mathrm{P}_2(w_1, v_2), \ldots, \mathrm{P}_2(w_{n_w}, v_{n_v-1}), \mathrm{P}_2(w_{n_w}, v_{n_v})$. Intuitively, $\forall(w, v)$ aggregates all the values in $[0, 1]^{n_w \times n_v}$, while $\forall\, \mathrm{Diag}(w, v)$ aggregates only the values in the diagonal.