# Tractable Probabilistic Neurosymbolic AI Using Boolean Tensor Factorizations

**Rik Adriaensen**[1] **Jaron Maene**[1] **Luc De Raedt**[1,2]

[1]KU Leuven, Belgium
[2]Örebro University, Sweden

## Abstract

Probabilistic logic programming offers a principled way of combining deep learning and formal reasoning, but its inference is $\#P$-hard. A simple factorization assumption introduced by TractOR makes probabilistic reasoning tractable for large relations at the cost of limiting expressivity. We reinterpret this approach using Boolean tensor factorizations to identify and resolve three practical limitations: it is restricted to symmetric relations, expresses only a limited class of distributions, and cannot reason across components.

## 1 INTRODUCTION

Probabilistic logic programming offers a principled paradigm for neurosymbolic AI (NeSy) [Manhaeve et al., 2021, Yang et al., 2020, Li et al., 2023], by combining deep learning and formal reasoning. Neural networks predict probabilities and probabilistic logic programs reason with them. Naturally, parameterizing the probabilities in a probabilistic logic program by neural networks results in a NeSy framework with clear semantics.

However, NeSy systems based on probabilistic logic programming have two problems. First, exact inference reduces to weighted model counting, which is $\#P$-hard [Valiant, 1979]. Second, they model probabilistic relations using independent random variables, so all dependencies must be fully described in the logic [van Krieken et al., 2024].

TractOR [Friedman and Van den Broeck, 2020] shows that representing probabilistic relations with a simple factorization rather than using independent random variables makes probabilistic reasoning tractable and allows learning dependencies. Applying this factorization to probabilistic NeSy resolves both problems but introduces limitations of its own, as we will show.

This paper reinterprets TractOR's factorization using Boolean tensor factorizations, identifies three practical limitations of it: (1) symmetric relations, (2) limited expressivity and (3) no interaction between components, and proposes a solution to each limitation: (1) untying the embeddings, (2) using a noisy-or instead of a mixture and (3) components with higher ranks.

## 2 PRELIMINARIES

**Probabilistic Datalog.** We assume basic familiarity with Datalog and otherwise refer to Appendix A and Abiteboul et al. [1995]. Probabilistic Datalog [Fuhr, 1995] extends Datalog with a distribution over interpretations via probabilistic facts. A *probabilistic fact* has the form $\mathsf{p_a} :: \mathsf{a}$, where $\mathsf{a}$ is a ground atom and $\mathsf{p_a} \in [0, 1]$ its associated probability. A *probabilistic Datalog program* is a pair $(\mathcal{PF}, \mathcal{R})$ where $\mathcal{PF}$ is a set of probabilistic facts and $\mathcal{R}$ a set of rules.

Each probabilistic fact corresponds to an independent Bernoulli variable. This induces a distribution over subsets of facts $w$ called *(possible) worlds*, defined as follows.

$$\Pr(w) = \prod_{\mathsf{a} \in w} \mathsf{p_a} \prod_{\mathsf{a} \in \mathcal{F} \setminus w} (1 - \mathsf{p_a}) \tag{1}$$

The *success probability* $\Pr(\mathsf{q})$ of a *query* atom $\mathsf{q}$ is defined under the *distribution semantics* [Sato, 1995] as the probability that it is entailed by the program $w \cup \mathcal{R}$, where $w$ is a randomly sampled world. The Iverson brackets $[\![.]\!]$ used below denote 1 if the condition holds and 0 otherwise.

$$\Pr(\mathsf{q}) = \sum_{w \subseteq \mathcal{F}} [\![w \cup \mathcal{R} \models \mathsf{q}]\!] \Pr(w) \tag{2}$$

The typical inference approach [Fierens et al., 2015] is to find a query formula over the probabilistic facts whose models are exactly the worlds $w$ in which the atom $\mathsf{q}$ is entailed by $w \cup \mathcal{R}$. The success probability then equals the weighted

model count (WMC) of this formula. When the Datalog program does not contain negation or recursion, this formula can be written as a union of conjunctive queries (UCQ) [Abiteboul et al., 1995].

The complexity of computing the WMC of a formula as a function of the number of probabilistic facts is called *data complexity*, in contrast to *query complexity*, where the facts are fixed and query size varies. Dalvi and Suciu [2013] show that every UCQ formula is either safe (polynomial data complexity) or unsafe ($\#P$-hard). They provide a lifted inference algorithm that computes the WMC in polynomial time for all safe formulas and correctly identifies (and fails on) unsafe formulas.

**TractOR.** TractOR [Friedman and Van den Broeck, 2020], a probabilistic relational embedding model, answers arbitrary queries with polynomial data complexity by making a simple factorization assumption. It introduces a new probabilistic fact $\mathsf{f}_1(\mathsf{t})$ for every predicate $\mathsf{t}$ and $\mathsf{f}_2(\mathsf{x})$ for every constant $\mathsf{x}$. Then it assumes that every binary predicate $\mathsf{t}$ factorizes as

$$\mathsf{t}(\mathsf{X},\mathsf{Y}) :- \mathsf{f}_1(\mathsf{t}),\mathsf{f}_2(\mathsf{X}),\mathsf{f}_2(\mathsf{Y}). \tag{3}$$

Tractability is then assured by the following theorem, as the UCQ representation of any query will now contain at most one variable per atom.

**Theorem 1** *[Friedman and Van den Broeck, 2020, Thm. 4] Suppose that $\varphi$ is a UCQ with at most one variable per atom. Then $\varphi$ is safe.*

To improve expressivity, they work with a mixture of $k$ such simple models using uniform mixing weights. The parameters of $\mathsf{f}_1(\mathsf{t})$ and $\mathsf{f}_2(\mathsf{x})$ across the mixture components act as embeddings.

## 3 BOOLEAN TENSOR FACTORIZATIONS IN DATALOG

TractOR's factorization is closely related to Boolean tensor factorizations. Hence, we introduce the relevant concepts [Miettinen, 2011] and show how to express them in Datalog.

A *Boolean tensor* is a multidimensional array $\mathbf{T} \in \mathbb{B}^{I_1 \times \cdots \times I_m}$ with all operations defined over the Boolean algebra. The *order* of the tensor equals the number of dimensions $m$, also called ways or modes. An $m$-ary relation $\mathsf{t}$ over a domain $\mathcal{D}$ can be represented as a Boolean tensor $\mathbf{T}$ of order $m$, assuming a fixed ordering of the domain.

We use lowercase symbols (e.g., $\mathsf{t}$) for logical predicates and the corresponding uppercase bold symbols (e.g., $\mathbf{T}$) for their tensor representation, such that $\forall \bar{\mathsf{c}} \in \mathcal{D}^m : \mathsf{t}(\bar{\mathsf{c}}) \iff \mathbf{T}(\bar{\mathsf{c}}) = 1$. As all Boolean tensors in this paper represent relations, we index tensors directly with domain elements, assuming an implicit mapping to tensor indices.

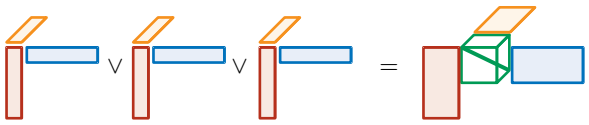

Figure 1: The Boolean CP factorization visualized.

A *rank*-1 Boolean tensor is an outer product of Boolean vectors. This rule expresses that $\mathbf{T}$ is a rank-1 tensor.

$$\mathsf{t}(\mathsf{X},\mathsf{Y},\mathsf{Z}) :- \mathsf{f}_1(\mathsf{X}),\mathsf{f}_2(\mathsf{Y}),\mathsf{f}_3(\mathsf{Z}) \tag{4}$$

The Boolean *canonical polyadic* (CP) factorization decomposes a tensor into a disjunction of $r$ rank-1 component tensors, where $r$ is the (Boolean) rank of the factorization. The Datalog rules $\mathsf{t}(\mathsf{X},\mathsf{Y},\mathsf{Z}) :- \mathsf{f}_1(\mathsf{X},1),\mathsf{f}_2(\mathsf{Y},1),\mathsf{f}_3(\mathsf{Z},1)$ and $\mathsf{t}(\mathsf{X},\mathsf{Y},\mathsf{Z}) :- \mathsf{f}_1(\mathsf{X},2),\mathsf{f}_2(\mathsf{Y},2),\mathsf{f}_3(\mathsf{Z},2)$ express that $\mathbf{T}$ is a rank-2 tensor[1]. The $r$ vectors for each mode $n$ can be collected as the columns of a *factor matrix* $\mathbf{F}_n$ corresponding to the predicate $\mathsf{f}_\mathsf{n}$ in the Datalog rules. The CP factorization can then be written as $\mathbf{I}_r \times_1 \mathbf{F}_1 \times_2 \mathbf{F}_2 \times_3 \mathbf{F}_3$, with $\mathbf{I}_r$ the size $r$ identity tensor and $\times_n$ the tensor product along the $n^{\text{th}}$ mode (see Figure 1). The numbers in the second argument position of $\mathsf{f}_\mathsf{n}$, indicating the columns, form the *latent domain* $\mathcal{D}_{\text{lat}} = \{1,\ldots,r\}$ where $r$ is the rank of the factorization. The two rules from before can now be written as a single rule where the variable $\mathsf{l}$ ranges over the latent domain containing two elements.

$$\mathsf{t}(\mathsf{X},\mathsf{Y},\mathsf{Z}) :- \mathsf{f}_1(\mathsf{X},\mathsf{l}),\mathsf{f}_2(\mathsf{Y},\mathsf{l}),\mathsf{f}_3(\mathsf{Z},\mathsf{l}) \tag{5}$$

**Example 1** *Consider the third-order Boolean tensor over the domain $\{\mathsf{a},\mathsf{b},\mathsf{c}\}$ representing the relation*

$$\mathsf{t} = \{(\mathsf{a},\mathsf{a},\mathsf{a}),(\mathsf{c},\mathsf{a},\mathsf{a}),(\mathsf{a},\mathsf{c},\mathsf{a}),(\mathsf{c},\mathsf{c},\mathsf{a}),(\mathsf{a},\mathsf{a},\mathsf{b}),(\mathsf{b},\mathsf{a},\mathsf{b}),$$
$$(\mathsf{a},\mathsf{b},\mathsf{b}),(\mathsf{b},\mathsf{b},\mathsf{b}),(\mathsf{a},\mathsf{a},\mathsf{c}),(\mathsf{c},\mathsf{a},\mathsf{c}),(\mathsf{a},\mathsf{c},\mathsf{c}),(\mathsf{c},\mathsf{c},\mathsf{c})\}.$$

*Then there exists a rank-2 Boolean CP factorization of this relation shown below as a program and using tensors.*

$$\mathsf{f}_1(\mathsf{a},1).\mathsf{f}_1(\mathsf{a},2).\mathsf{f}_1(\mathsf{b},2).\mathsf{f}_1(\mathsf{c},1).\mathsf{f}_2(\mathsf{a},1).\mathsf{f}_2(\mathsf{a},2).$$
$$\mathsf{f}_2(\mathsf{b},2).\mathsf{f}_2(\mathsf{c},1).\mathsf{f}_3(\mathsf{a},1).\mathsf{f}_3(\mathsf{b},2).\mathsf{f}_3(\mathsf{c},1).$$
$$\mathsf{t}(\mathsf{X},\mathsf{Y},\mathsf{Z}) :- \mathsf{f}_1(\mathsf{X},\mathsf{l}),\mathsf{f}_2(\mathsf{Y},\mathsf{l}),\mathsf{f}_3(\mathsf{Z},\mathsf{l}).$$

$$\mathbf{T} = \begin{array}{c} \\ \mathsf{a} \\ \mathsf{b} \\ \mathsf{c} \end{array}\!\!\begin{array}{c} \begin{array}{ccc}\mathsf{a}&\mathsf{b}&\mathsf{c}\end{array} \\ \left[\begin{array}{ccc} 1&0&1 \\ 0&0&0 \\ 1&0&1 \end{array}\right] \end{array} \begin{array}{c} \begin{array}{ccc}\mathsf{a}&\mathsf{b}&\mathsf{c}\end{array} \\ \left[\begin{array}{ccc} 1&1&0 \\ 1&1&0 \\ 0&0&0 \end{array}\right] \end{array} \begin{array}{c} \begin{array}{ccc}\mathsf{a}&\mathsf{b}&\mathsf{c}\end{array} \\ \left[\begin{array}{ccc} 1&0&1 \\ 0&0&0 \\ 1&0&1 \end{array}\right] \end{array}$$

$$= \mathbf{I}_2 \times_1 \begin{array}{c} \begin{array}{cc}1&2\end{array} \\ \left[\begin{array}{cc} 1&1 \\ 0&1 \\ 1&0 \end{array}\right] \end{array} \times_2 \begin{array}{c} \begin{array}{cc}1&2\end{array} \\ \left[\begin{array}{cc} 1&1 \\ 0&1 \\ 1&0 \end{array}\right] \end{array} \times_3 \begin{array}{c} \begin{array}{cc}1&2\end{array} \\ \left[\begin{array}{cc} 1&0 \\ 0&1 \\ 1&0 \end{array}\right] \end{array} \begin{array}{c} \\ \mathsf{a} \\ \mathsf{b} \\ \mathsf{c} \end{array}$$

[1] W.l.o.g., we factorize a single relation, as the tensors of multiple relations of the same arity can always be stacked along a new mode creating a single higher-order relation (see Appendix B.1).

Each rank-1 component in the factorization covers a subset of the relation called a tile. Therefore, the CP factorization corresponds to a tiling of the reconstructed relation. See Appendix B for further discussion on Boolean tensor factorizations and their connection to tilings.

When the same factor matrix is used for different modes in the same factorization, we call these modes or embeddings *tied*. For example, in TractOR's factorization (Equation 3) the second and third mode are tied.

# 4 TRACTABLE PROBABILISTIC DATALOG

We now show how modeling probabilistic relations with tensor factorizations rather than fully materializing them leads to tractable inference.

We say a program is *factorized* if all probabilistic facts are factor predicates of the same CP factorization. Hence, a factorized program contains one rule defining the factorization (Equation 5) and possibly others over the relations reconstructed by the factorization. We assume every possible probabilistic fact over the factor predicates is present, the number of which is polynomial in the domain sizes (i.e., $\mathcal{O}(m|\mathcal{D}||\mathcal{D}_{\text{lat}}|)$). Since we use the fragment of Datalog equivalent to UCQ, any formula derived from a factorized program can be written as a UCQ containing only factor predicates. Grounding all variables ranging over the latent domain results in a formula where every atom contains at most one variable. As the increase in formula size depends only on the size of the latent domain, the theorem below follows from Theorem 1.

**Theorem 2** *For a fixed latent domain $\mathcal{D}_{\text{lat}}$, answering an arbitrary query over a factorized negation-free non-recursive probabilistic Datalog program is polynomial in the size of the original domain $\mathcal{D}$.*

However, answering an arbitrary query remains intractable in the latent domain size. In practice, this intractability is prohibitive, even for low-rank factorizations where this domain is small. TractOR avoids this problem by using a mixture of rank-1 factorized programs. That is, it takes $k$ factorized programs with a rank-1 factorization rule (in which case Equation 5 reduces to Equation 4), and the probability of a query $\mathsf{q}$ is the average over the answers of these programs. That is, $\Pr_{\text{TractOR}}(\mathsf{q}) = \sum_{j=1}^{k} \frac{1}{k} \Pr_j(\mathsf{q})$ with $\Pr_j(\mathsf{q})$ the predicted success probability of the $j^{\text{th}}$ rank-1 program.

**Learning factorizations.** Factorized programs are a tractable subset of probabilistic Datalog, so we can learn their parameters with any standard approach. However, to achieve the claimed tractability, we rewrite the query as a UCQ and use the lifted inference algorithm from Dalvi and Suciu [2013] to compile a polynomial-sized circuit computing its WMC. The circuit is end-to-end differentiable, so we can learn the parameters using gradient descent. This is a gradient-based approach to learning the factorization.

On a task where the domain consists of subsymbolic inputs, e.g., a citation graph where every entity is a text document, the parameters of a factorized program can be predicted by neural networks. Current neurosymbolic frameworks usually predict relations over, e.g., documents directly, whereas here they would predict the latent unary properties of each document (each column of the factor matrices represents one such property), from which these relations can be reconstructed. We leave such an extension for future work.

## 4.1 LIMITATIONS AND SOLUTIONS

We identify three limitations of TractOR's factorization, propose a solution for each and use experiments to illustrate them. We experiment on two tasks: a toy example containing two simple relations $\mathsf{t}_1$ and $\mathsf{t}_2$ (see Figure 3, ground truth) and the countries knowledge graph [Bouchard et al., 2015], which contains the locatedIn and neighborOf relations over all countries, 23 subregions, and 5 continents of the world. On both tasks, we first train a mixture or noisy-or over rank-1 programs with only the factorization rule, learning an approximation of the given relations. Then, we add various rules to the rank-1 programs to derive the symmetric closure, intersection, union, and composition of the learned relations. We report binary cross-entropy (BCE) to measure the quality of the computed success probabilities and mean average precision (mAP) to measure the correct ranking of true and false facts. For further experimental details, including the rules used, we refer to Appendix C.

**1. Symmetric relations only.** Tying modes restricts tiles to be symmetric in these modes. For example, TractOR's factorization (Equation 3) can only represent symmetric relations because the final two modes are tied, as also mentioned in the paper [Friedman and Van den Broeck, 2020].

Untying the modes (i.e., a separate $\mathsf{f}_\mathsf{n}$ for each argument position) allows representing non-symmetric relations, although this increases the number of parameters from $\mathcal{O}(k|\mathcal{D}|)$ to $\mathcal{O}(mk|\mathcal{D}|)$. Figure 2 (top) shows on the toy task that this has a significant impact even when factorizing very simple relations.

**2. Limited expressivity.** Each rank-1 program defines a distribution over possible worlds in which the reconstructed relation $\mathsf{t}$ is a single tile, so the success probability of $\mathsf{t}(\mathsf{a},\mathsf{b})$ equals the marginal probability that $(\mathsf{a},\mathsf{b})$ lies in that tile. A mixture of the predicted success probability of these rank-1 programs can only assign probability one to two facts $\mathsf{t}(\mathsf{a}_1,\mathsf{b}_1)$ and $\mathsf{t}(\mathsf{a}_2,\mathsf{b}_2)$ if all rank-1 programs assign only probability mass to tiles covering both of these facts. As

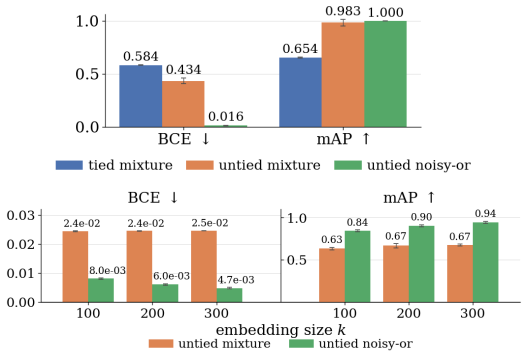


Figure 2: The results for training the rank-1 programs on the toy task (top) and countries task (bottom).

any tile covering both $\mathsf{t}(\mathsf{a}_1, \mathsf{b}_1)$ and $\mathsf{t}(\mathsf{a}_2, \mathsf{b}_2)$ (with $\mathsf{a}_1 \neq \mathsf{a}_2$ and $\mathsf{b}_1 \neq \mathsf{b}_2$) also covers $\mathsf{t}(\mathsf{a}_1, \mathsf{b}_2)$ and $\mathsf{t}(\mathsf{a}_2, \mathsf{b}_1)$ it is then impossible to *not* assign probability one to the latter two facts at the same time. These mixtures can therefore not represent arbitrary success probabilities even with an arbitrary number of components. When learning the factorization's parameters using binary cross-entropy loss, as is common for probabilistic logic programs, this inexpressivity moreover leads to local minima that separate positive from negative queries but fail to learn the correct probabilities of these queries.

Combining the success probabilities of the rank-1 programs using a noisy-or, i.e., $\Pr(\mathsf{q}) = 1 - \prod_{j=1}^{k}(1 - \Pr_j(\mathsf{q}))$, instead of a mixture, resolves the expressivity problem. Whereas the mixture treats the programs as mutually exclusive, selecting one program's prediction, the noisy-or treats them as independent, so a fact is derived whenever at least one program predicts it. This does allow representing arbitrary success probabilities given an arbitrary number of components, as we could define a separate tile for every true reconstructed ground fact. Figure 2 shows that using the noisy-or is better at both learning the correct marginals and correctly ranking true and false facts. Figure 3 (top) shows on the toy task that the noisy-or better approximates the ground-truth queries than the mixture.

**3. No interaction between components.** Finally, when adding rules to rank-1 programs, they are not applied across components. Since every program reconstructs a single tile, rule bodies only combine facts from the same tile. Whether this limits what is derivable depends on the rule. For example, applying symmetry to each tile makes the whole relation symmetric, but composing two facts from different tiles is not possible.

The third limitation would be fully resolved by a single program of sufficiently high rank to approximate the target relation, at the cost of intractability in the rank (Theorem 2). A noisy-or over low-rank components would instead trade tractability against which facts are derivable, which we leave to future work. Table 1 compares our approach, which adds the rules to each rank-1 program, against a baseline that first reconstructs the relation with the noisy-or and then applies the rules. The baseline is infeasible for the larger countries knowledge graph, whereas our approach does scale to this larger task (see Appendix C.1). The factorized program perfectly applies the symmetry rule but does not consistently predict the success probabilities for the other rules, as this might require reasoning across tiles. For example, Figure 3 (bottom) shows that the composition rule is correctly applied within a single tile ($\mathsf{composition}_2$, composing $\mathsf{t}_2$ facts) but not across tiles ($\mathsf{composition}_1$, composing $\mathsf{t}_1$ facts).

Table 1: The results for applying rules to learned relations for the toy task using the untied noisy-or program. Adding the rules to the factorized program is compared to a baseline which first reconstructs the learned factorization and then applies the rules.

| | baseline | | factorized | |
|---|---|---|---|---|
| rule | BCE ↓ | mAP ↑ | BCE ↓ | mAP ↑ |
| symmetry | 0.025 | 1.000 | 0.019 | 1.000 |
| intersection | 0.012 | 1.000 | 1.832 | 1.000 |
| union | 0.029 | 1.000 | 0.581 | 1.000 |
| composition | 0.017 | 1.000 | 2.416 | 0.851 |

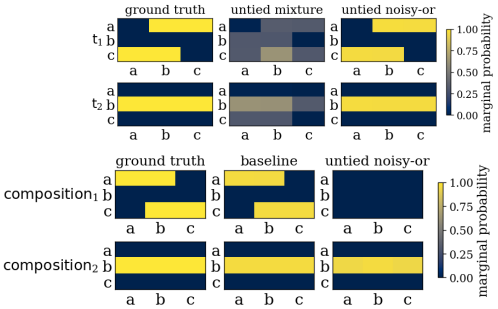


Figure 3: The success probabilities predicted by the noisy-or and mixture over rank-1 programs for the learned predicates $\mathsf{t}_1$ and $\mathsf{t}_2$ (top) and derived composition facts (bottom).

## 5 CONCLUSION

The mixture of rank-1 factorizations proposed by TractOR [Friedman and Van den Broeck, 2020] can be applied to probabilistic Datalog for tractable probabilistic neurosymbolic reasoning. However, three limitations restrict the practical use of this subset of probabilistic Datalog: it can only represent symmetric relations, expresses a limited class of distributions, and does not allow reasoning across mixture components. We proposed a solution to each: untying the embeddings, replacing the mixture with a noisy-or, and using components with a higher rank.

**Acknowledgements**

This research received funding from the Flemish Government under the "Flanders Artificial Intelligence Research program". This research is co-funded by the European Union (ERC, DeepLog, 101142702). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Luc De Raedt is also supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

**References**

Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. ISBN 0-201-53771-0.

Guillaume Bouchard, Sameer Singh, and Théo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *Proceedings of the 2015 AAAI Spring Symposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches*, 2015.

Nilesh Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59(6), January 2013. ISSN 0004-5411. doi: 10.1145/2395116.2395119.

Daan Fierens, Guy Van Den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, May 2015. Publisher: Cambridge University Press.

Tal Friedman and Guy Van den Broeck. Symbolic querying of vector spaces: Probabilistic databases meets relational embeddings. In Jonas Peters and David Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 1268–1277. PMLR, 03–06 Aug 2020.

Norbert Fuhr. Probabilistic datalog—a logic for powerful retrieval methods. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '95, page 282–290, New York, NY, USA, 1995. Association for Computing Machinery. ISBN 0897917146. doi: 10.1145/215206.215372.

Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In Einoshin Suzuki and Setsuo Arikawa, editors, *Discovery Science*, pages 278–289, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30214-8.

Ziyang Li, Jiani Huang, and Mayur Naik. Scallop: A language for neurosymbolic programming. *Proc. ACM Program. Lang.*, 7(PLDI), June 2023. doi: 10.1145/3591280.

Jaron Maene, Vincent Derkinderen, and Pedro Zuidberg Dos Martires. KLay: Accelerating arithmetic circuits for neurosymbolic AI. In *The Thirteenth International Conference on Learning Representations*, 2025.

Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298:103504, 2021. ISSN 0004-3702. doi: 10.1016/j.artint.2021.103504.

Pauli Miettinen. Boolean tensor factorizations. In *2011 IEEE 11th International Conference on Data Mining*, pages 447–456, 2011. doi: 10.1109/ICDM.2011.28.

Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In Leon Sterling, editor, *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, pages 715–729. MIT Press, 1995.

Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979. doi: 10.1137/0208032. URL `https://doi.org/10.1137/0208032`.

M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, October 1976. ISSN 0004-5411. doi: 10.1145/321978.321991.

Emile van Krieken, Pasquale Minervini, Edoardo M. Ponti, and Antonio Vergari. On the independence assumption in neurosymbolic learning. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing neural networks into answer set programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1755–1762. International Joint Conferences on Artificial Intelligence Organization, 7 2020. doi: 10.24963/ijcai.2020/243. Main track.

# Tractable Probabilistic Neurosymbolic AI Using Boolean Tensor Factorizations (Supplementary Material)

**Rik Adriaensen**[1] **Jaron Maene**[1] **Luc De Raedt**[1,2]

[1]KU Leuven, Belgium
[2]Örebro University, Sweden

## A DATALOG

A *term* is either a constant $\mathsf{c}$ or a variable $\mathsf{X}$. An *atom* has the form $\mathsf{a}(\bar{\mathsf{c}})$, where $\mathsf{a}$ is a predicate of arity $m$, and $\bar{\mathsf{c}} = \mathsf{c}_1, \ldots, \mathsf{c}_m$ are its arguments. A *rule* is an expression of the form $\mathsf{h} :\!-\ \mathsf{b}_1, \ldots, \mathsf{b}_n.$, where $\mathsf{h}$ is an atom called the *head*, and $\mathsf{b}_1, \ldots, \mathsf{b}_n$ are atoms forming the *body*. The rule states that the head is true whenever all atoms in the body are true. A *fact* is a ground rule with an empty body representing an atom that is unconditionally true. An expression (atom, rule, query) is *ground* if it does not contain variables. A non-ground expression can be *grounded* by substituting all its variables by constants.

A *Datalog program* $\mathcal{P}$ is a pair $(\mathcal{F}, \mathcal{R})$, where $\mathcal{F}$ is a set of facts and $\mathcal{R}$ is a set of rules with non-empty bodies. The set of all constants occurring in a program is called the *domain* $\mathcal{D}$, also known as the Herbrand universe. The set of all predicates occurring in the program is denoted by $\mathrm{Pred}(\mathcal{P})$. Throughout the paper, we will assume that the predicates occurring in $\mathcal{F}$ do not occur in the heads of the rules $\mathcal{R}$ and there is no recursion in the rule set. Rules are sometimes called conjunctive queries (CQ) and multiple rules for the same head predicate form a union of conjunctive queries (UCQ), which represents a disjunction of conjunctions.

The *Herbrand base* $\mathcal{B}$ is the set of all possible ground atoms that can be formed using predicates in $\mathrm{Pred}(\mathcal{P})$ and constants in $\mathcal{D}$. A *(Herbrand) interpretation* is a set of ground atoms that are considered true; all other ground atoms in $\mathcal{B}$ are considered false. The interpretation of a predicate is called a relation, but they are often used interchangeably. A ground rule is *satisfied* by an interpretation if either the body is false or the head is true. A non-ground rule is satisfied if all its ground instances are satisfied. An interpretation is a *(Herbrand) model* of a program if it satisfies all rules in the program. The *least (Herbrand) model* $\mathcal{M}_\mathcal{P}$ is the unique model that is a subset of all other models [Van Emden and Kowalski, 1976] and it determines the intended semantics of the program. A ground query $\mathsf{q}$ is *entailed by* a program $\mathcal{P}$, denoted $\mathcal{P} \models \mathsf{q}$, if $\mathsf{q}$ is true in the least model of $\mathcal{P}$.

**Example 2** *Consider the following program $\mathcal{P}$ with domain $\mathcal{D} = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$.*

$$\mathsf{likes}(\mathsf{a},\mathsf{a}).\ \mathsf{likes}(\mathsf{a},\mathsf{b}).\ \mathsf{likes}(\mathsf{b},\mathsf{a}).$$
$$\mathsf{likes}(\mathsf{b},\mathsf{b}).\ \mathsf{likes}(\mathsf{b},\mathsf{c}).\ \mathsf{likes}(\mathsf{c},\mathsf{b}).\ \mathsf{likes}(\mathsf{c},\mathsf{c}).$$
$$\mathsf{path}_3(\mathsf{X},\mathsf{Y}) :\!-\ \mathsf{likes}(\mathsf{X},\mathsf{X}_1), \mathsf{likes}(\mathsf{X}_1,\mathsf{X}_2), \mathsf{likes}(\mathsf{X}_2,\mathsf{Y}).$$

*The Herbrand base $\mathcal{B}$ is* $\{\mathsf{likes}(\mathsf{a},\mathsf{a}), \mathsf{likes}(\mathsf{a},\mathsf{b}), \ldots, \mathsf{path}_3(\mathsf{c},\mathsf{b}), \mathsf{path}_3(\mathsf{c},\mathsf{c})\}$. *The least model of $\mathcal{P}$ is* $\mathcal{M}_\mathcal{P} = \mathcal{B} \setminus \{\mathsf{likes}(\mathsf{a},\mathsf{c}), \mathsf{likes}(\mathsf{c},\mathsf{a})\}$.

## B BOOLEAN TENSOR FACTORIZATIONS

### B.1 FACTORIZING MULTIPLE RELATIONS

To factorize multiple relations of the same arity, their Boolean tensors can be stacked along a new mode, and the resulting higher-order tensor can be factorized in the usual way. This corresponds to reification in Datalog, where every atom

$\mathsf{t}(\mathsf{c}_1, \ldots, \mathsf{c}_m)$ containing one of the chosen predicates is replaced by $\mathsf{stack}(\mathsf{t}, \mathsf{c}_1, \ldots, \mathsf{c}_m)$, with $\mathsf{stack}$ a fresh predicate. This preserves the semantics of the program. We allow predicates to appear as arguments in the paper, implicitly assuming reification. For example, $\mathsf{t}(\mathsf{X}, \mathsf{Y}) :- \mathsf{f}_1(\mathsf{t}), \mathsf{f}_2(\mathsf{X}), \mathsf{f}_3(\mathsf{Y})$ is syntactic sugar for $\mathsf{stack}(\mathsf{t}, \mathsf{X}, \mathsf{Y}) :- \mathsf{f}_1(\mathsf{t}), \mathsf{f}_2(\mathsf{X}), \mathsf{f}_3(\mathsf{Y})$.

### B.2 CLUSTERINGS AND TILINGS

It is common to interpret the Boolean factor matrices $\mathbf{F}_n$ as clusterings, where each column defines a different cluster [Miettinen, 2011]. Each column corresponds to a latent constant in $\mathcal{D}_{\text{lat}}$, so latent constants can be thought of as clusters containing all domain elements with a 1 in the corresponding column. For example, the fact $\mathsf{f}_\mathsf{n}(\mathsf{c}, 1)$ can be interpreted as $\mathsf{c}$ being in cluster 1 under clustering $\mathsf{n}$.

The corresponding interpretation of the CP decomposition is that of a tiling [Geerts et al., 2004, Miettinen, 2011]. Each rank-1 component tensor corresponds to an all-ones subtensor called a tile. Together, the $r$ tiles must cover all ones of the reconstructed tensor and none of the zeros. A tile has the same order as the reconstructed tensor but contains only a subset of its indices along each mode. These subsets are defined by the corresponding clusters in the factor matrices. Formally, a tiling is defined as shown below.

**Definition 1** *(Tiling) Given an $m$-ary relation $\mathsf{t} \subseteq \mathcal{D}^m$, a* tile *$\tau(d_1, \ldots, d_m)$ is the Cartesian product $d_1 \times \cdots \times d_m$ of $m$ subsets of the domain, that is, $d_n \subseteq \mathcal{D}$ for each $n \in \{1, \ldots, m\}$. A tile* covers $\mathsf{t}$ *if all tuples in the tile are part of the relation, that is, $\forall(\mathsf{c}_1, \ldots, \mathsf{c}_m) \in \tau(d_1, \ldots, d_m) : \mathsf{t}(\mathsf{c}_1, \ldots, \mathsf{c}_m)$. A set of tiles forms a* tiling *of relation $\mathsf{t}$ if the union of all the tiles is equal to the relation $\mathsf{t}$.*

## C EXPERIMENTAL DETAILS

Both tasks consist of two steps: learn only the factorization and apply the rules. For the first step, we train the parameters of a factorized program containing only the factorization rule (e.g., $\mathsf{t}(\mathsf{X}, \mathsf{Y}) :- \mathsf{f}_1(\mathsf{t}, \mathsf{I}), \mathsf{f}_2(\mathsf{X}, \mathsf{I}), \mathsf{f}_3(\mathsf{Y}, \mathsf{I})$ for each probabilistic relation $\mathsf{t}$). Afterwards, we add a copy of the rules shown in Equation 6 (substituting $\mathsf{t}$) for every probabilistic relation learned in the experiment.

$$\begin{aligned}
&\mathsf{symmetric}(\mathsf{X}, \mathsf{Y}) :- \mathsf{t}(\mathsf{X}, \mathsf{Y}).\\
&\mathsf{symmetric}(\mathsf{X}, \mathsf{Y}) :- \mathsf{t}(\mathsf{Y}, \mathsf{X}).\\
&\mathsf{composition}(\mathsf{X}, \mathsf{Z}) :- \mathsf{t}(\mathsf{X}, \mathsf{Y}), \mathsf{t}(\mathsf{Y}, \mathsf{Z}).\\
&\mathsf{union}(\mathsf{X}, \mathsf{Y}) :- \mathsf{t}(\mathsf{X}, \mathsf{Z}).\\
&\mathsf{union}(\mathsf{X}, \mathsf{Y}) :- \mathsf{t}(\mathsf{Y}, \mathsf{Z}).\\
&\mathsf{intersection}(\mathsf{X}, \mathsf{Y}) :- \mathsf{t}(\mathsf{X}, \mathsf{Z}), \mathsf{t}(\mathsf{Y}, \mathsf{Z}).
\end{aligned} \tag{6}$$

As a baseline for the rule applications, we first reconstruct the learned relations as the predicted success probabilities of the mixture/noisy-or over rank-1 programs and add them as explicit probabilistic relations to a probabilistic program. When learning the factorizations, we use the hyperparameters reported in Table 2. We repeat all experiments across five random seeds and report the means. For the countries experiment, we add a regularization term to the loss with regularization weight $\lambda$ that ensures the $L_1$ norm of each column of the factor matrices is larger than one to ensure no components are unused. This is essential to make the performance improve with increasing rank.

Table 2: Hyperparameters for the toy and countries experiments, where $k$ refers to the number of rank-1 programs.

| Experiment | lr | epochs | $k$ | loss | $\lambda$ |
|---|---|---|---|---|---|
| Toy | 1 | 1000 | 3 | BCE | - |
| Countries | 0.1 | 600 | $\{100, 200, 300\}$ | BCE | 0.1 |

We train and evaluate the programs similar to Fierens et al. [2015] by compiling them once into an end-to-end differentiable circuit implemented with an extension of KLay [Maene et al., 2025].

### C.1 ADDITIONAL RESULTS

In this section, we include some additional experimental results. Table 3 shows the performance of adding different rules to the factorized programs (untied noisy-or) trained on the countries dataset for an increasing number of rank-1 programs $k$. Figure 4 shows the application of all four rules on the toy relations, further illustrating that these rule applications are limited to single tiles.

Table 3: The results for adding rules to the rank-1 programs using the untied noisy-or after training the factorization on the countries task.

| | $k = 100$ | | $k = 200$ | | $k = 300$ | |
|---|---|---|---|---|---|---|
| | BCE ↓ | mAP ↑ | BCE ↓ | mAP ↑ | BCE ↓ | mAP ↑ |
| symmetry | 0.010 | 0.873 | 0.008 | 0.931 | 0.006 | 0.963 |
| intersection | 0.513 | 0.779 | 0.428 | 0.846 | 0.402 | 0.876 |
| union | 0.121 | 1.000 | 0.082 | 1.000 | 0.058 | 1.000 |
| composition | 0.074 | 0.398 | 0.087 | 0.400 | 0.094 | 0.439 |

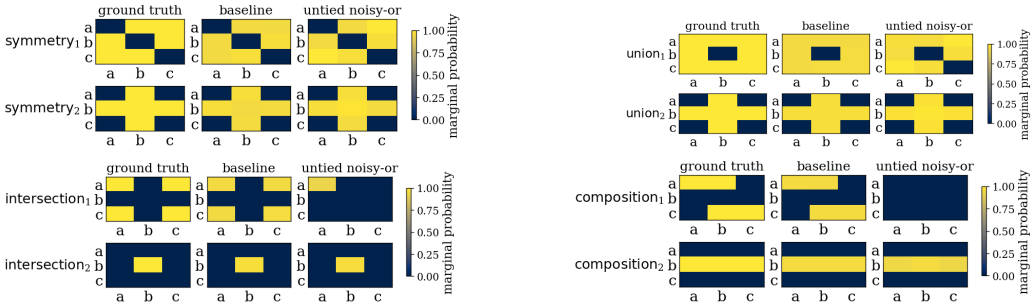


Figure 4: Success probabilities predicted by the baseline and the factorized programs for the derived symmetry, union, intersection and composition facts.