Semantic Probabilistic Layers for Neuro-Symbolic Learning

Kareem Ahmed[1] **Stefano Teso**[2] **Kai-Wei Chang**[1] **Guy Van den Broeck**[1] **Antonio Vergari**[3]

[1]Computer Science Department, University of California Los Angeles, USA
[2]Department of Computer Science and Information Engineering, University of Trento, Italy
[3]School of Informatics, University of Edinburgh, United Kingdom

Abstract

We design a predictive layer for structured-output prediction (SOP) that can be plugged into any neural network guaranteeing its predictions are consistent with a set of predefined symbolic constraints. Our **S**emantic **P**robabilistic **L**ayer (SPL) can model intricate correlations, and hard constraints, over a structured output space while being amenable to end-to-end learning via maximum likelihood. SPLs combine exact probabilistic inference with logical reasoning in a clean and modular way, learning complex distributions and restricting their support to solutions of the constraint. As such, they can faithfully, and efficiently, model complex SOP tasks beyond the reach of alternative neuro-symbolic approaches. We show SPLs outperform such competitors in terms of accuracy on challenging SOP tasks including hierarchical multi-label classification, pathfinding and preference learning, while retaining *perfect* constraint satisfaction.

1 INTRODUCTION

Modularity is among the major factors that propelled the Cambrian explosion of deep learning [Goodfellow et al., 2016]. By stacking multiple *differentiable* layers together, practitioners are able to train deep classifiers in an end-to-end fashion with little effort. However, despite its flexibility, *this modular approach to learning does not guarantee that the predictions of these models conform to our expectations as to what makes sense*. On the contrary, unconstrained deep classifiers are notorious for leading to predictions that are inconsistent with the rules governing the underlying domain.

This is even more evident in, and crucial for, structured output prediction (SOP) tasks, where classifiers have to predict hundreds of mutually constrained labels [Tsochantaridis et al., 2004, Borchani et al., 2015]. Consider for example a classical SOP task such as multi-label classification (MLC) [Tsoumakas and Katakis, 2007]. Learning a multi-label classifier that disregards the correlations among labels, e.g., by considering them *fully independent* given the inputs, yields sub-optimal results [Bielza et al., 2011]. In more challenging tasks such as hierarchical MLC (HMLC) [Sorower, 2010] or pathfinding [Pogančić et al., 2019], leveraging the domain's logical constraints (encoding, e.g., the label hierarchy or acyclicity and connectedness of a path) at training time can improve prediction accuracy [Levatić et al., 2015], but it cannot guarantee that the predictions are always *consistent* with the constraints at inference time [Giunchiglia and Lukasiewicz, 2020]. fig. 1 illustrates this problem in the context of pathfinding: constraint-unaware neural networks systematically fail to predict label configurations that form a valid path. In many safety-critical scenarios such as protein function [Radivojac et al., 2013] and interaction prediction [Sacca et al., 2014], and drug discovery [De Cao and Kipf, 2018, Di Liello et al., 2020], predicting inconsistent solutions can not only be harmful but also highly expensive [Amodei et al., 2016, Giunchiglia et al., 2022].

Unsurprisingly, due to their discrete nature, injecting logical constraints into deep neural networks while retaining modularity and differentiability is extremely challenging, as demonstrated in the *neuro-symbolic learning* literature [Sarker et al., 2021]. One such attempt has been to learn neural networks that satisfy the logical constraints by explicitly minimizing a differentiable loss term, the probability that the networks violates the constraint for any given prediction. And while successful, such approaches do not guarantee consistency of the predictions at test time. More recently, researchers have proposed predictive layers that do guarantee consistency, but these are restricted to specific kinds of symbolic knowledge [Giunchiglia and Lukasiewicz, 2020, Sivaraman et al., 2020] or are intractable for even moderately complex logical constraints [Hoernle et al., 2022].

Motivated by these observations, we introduce a novel **S**emantic **P**robabilistic **L**ayer (SPL) for modeling intricate correlations, and logical constraints on the labels of the out-

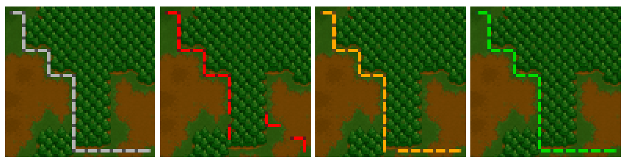


Figure 1: **Neural nets struggle with satisfying validity constraints in complex semantic SOP tasks** such as predicting the lowest-cost path from the top-left to the bottom-right corners of a Warcraft map. Even state-of-the-art neuro-symbolic approaches like the Semantic Loss (SL) Xu et al. [2018] fail to ensure consistency with hard rules (c). SPLs in contrast guarantees validity while retaining modularity, expressiveness and efficiency. See Sec. 4 for details.

put space in a modular and probabilistically sound manner. It does so by leveraging recent advancements in the literature on probabilistic circuits [Vergari et al., 2020, Choi et al., 2020]. The key features of SPL are that, on the one hand, it can be used as a *drop-in replacement* for common predictive layers such as sigmoid layers, and on the other, it *guarantees* the output's consistency with any prespecified logical constraints. Importantly, SPL supports efficient inference and – perhaps surprisingly – does not complicate training.

2 DESIGNING A PROBABILISTIC LAYER FOR NEURO-SYMBOLIC SOP

Notation. We denote scalar constants x in lower case, random variables X in upper case, vectors of constants $\boldsymbol{x}$ in bold and vectors of random variables $\mathbf{X}$ in capital boldface. $\mathbb{1}\{\varphi\}$ denotes the indicator function that evaluates to 1 if the statement φ holds and to 0 otherwise. We denote by $\boldsymbol{x} \models \mathsf{K}$ that the value assignment $\boldsymbol{x}$ satisfies the logical formula K.

Neuro-symbolic SOP. We tackle SOP tasks in which a neural net classifier must learn to associate instances $\boldsymbol{x} \in \mathbb{R}^D$ to L *interdependent* labels, identified by $\boldsymbol{y} \in \{0,1\}^L$. We assume that we can abstract any neural classifier into two components: a feature extractor f that maps inputs $\mathbf{X}$ to a M-dimensional embedding $\mathbf{Z} = f(\mathbf{X})$ and a predictive final layer that outputs the label distribution $p(\mathbf{Y} \mid \mathbf{Z})$. For example, the simplest [Mullenbach et al., 2018, Xu et al., 2018, Giunchiglia and Lukasiewicz, 2020] predictive layer in neural classifiers for SOP considers labels Y_i to be conditionally independent from each other given $\mathbf{Z}$, i.e., $p(\mathbf{Y} \mid \mathbf{Z}) = \prod_{i=1}^{L} p(Y_i \mid \mathbf{Z})$.

We are interested in dependencies between labels that can occur both as *correlations*, as in MLC [Dembczyński et al., 2012], and as encoded by a logical formula K over the labels $\mathbf{Y}$ and optionally a subset of the discrete inputs variables $\mathbf{X}$.

SPL. At a high level, SPL is a single layer that combines exact probabilistic inference with logical reasoning in a clean and modular way, learning complex distributions and restricting their support to solutions of the constraint.

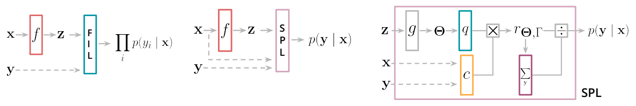


Figure 2: **A high level view of SPLs.** The predictive layer of a neural network for neuro-symbolic SOP, e.g., a FIL (**left**), can be readily replaced by a SPL (**middle**). SPLs are implemented (**right**) by multiplying a probabilistic circuit $q_{\mathbf{\Theta}}(\mathbf{Y} \mid f(\mathbf{X}))$ parameterized by (a function g of) the network's embeddings $f(\mathbf{X})$, and a constraint circuit $c_{\mathsf{K}}(\mathbf{X}, \mathbf{Y})$ embodying the symbolic knowledge. The result is normalized by efficiently marginalizing over the product circuit $r_{\mathbf{\Theta},\mathsf{K}}$, so as to guarantee fully probabilistic semantics and end-to-end differentiable learning by maximum likelihood.

Definition 2.1 (Semantic probabilistic layer (SPL))**.** Given an input configuration $\boldsymbol{x}$, a SPL decomposes the computation of the probability of a label configuration as:

$$p(\boldsymbol{y} \mid f(\boldsymbol{x})) = q_{\mathbf{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x})) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y}) / \mathcal{Z}(\boldsymbol{x}), \qquad (1)$$

where $\mathcal{Z}(\boldsymbol{x}) = \sum_{\boldsymbol{y}} q_{\mathbf{\Theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$. Here, $q_{\mathbf{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x}))$ is a module to perform probabilistic reasoning by encoding an expressive distribution over the labels parameterized by $\mathbf{\Theta}$; $c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$ is a module to ensure consistency of the predictions by encoding logical constraints K being non-zero only when K is satisfied, i.e., $c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y}) = \mathbb{1}\{(\boldsymbol{x}, \boldsymbol{y}) \models \mathsf{K}\}$; and $\mathcal{Z}(\boldsymbol{x})$ is a normalization term. fig. 2 illustrates the computational graph of our SPL at training.

3 REALIZING SPLS WITH CIRCUITS

SPLs can be realized as *circuits*, a large class of computational graphs that can represent both functions and distributions [Choi et al., 2020, Darwiche and Marquis, 2002], and subsume a plethora of tractable probabilistic models. The key idea is to leverage this single formalism to represent both an expressive joint distribution for $q_{\mathbf{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x}))$ and a compact encoding of the logical constraints for $c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$, while ensuring the exact and efficient evaluation of eq. (1). This can be achieved by ensuring that these computational graphs respect certain structural properties: *smoothness*, *decomposability*, *determinism* and *compatibility* [Darwiche and Marquis, 2002, Vergari et al., 2021]. Next, we introduce *probabilistic circuits* for modeling $q_{\mathbf{\Theta}}$ and *constraint circuits* for c_{K}, while in Sec. B we propose a more efficient implementation of SPL that utilizes a single circuit.

3.1 EXPRESSIVE DISTRIBUTIONS VIA PROBABILISTIC CIRCUITS

We start by introducing circuits for *joint* probability distributions, which we then extend to *conditional* distributions, that are used to implement $q_{\boldsymbol{\Theta}}(\mathbf{Y} \mid f(\mathbf{X}))$ in SPLs.

Definition 3.1 (Circuits)**.** A circuit h over variables $\mathbf{Y}$ is a computational graph encoding a parameterized function $h_{\boldsymbol{\Theta}}(\mathbf{Y})$ by combining different computational units: input functional units, sum units, and product units. An input functional n represents a base parametric function $h_n(\mathsf{sc}(n); \boldsymbol{\lambda})$ over some variables $\mathsf{sc}(n) \subseteq \mathbf{Y}$, its scope, and is parameterized by $\boldsymbol{\lambda}$. Sum and product units n elaborate the output of other units, denoted $\mathsf{in}(n)$. Sum units are parameterized by $\boldsymbol{\omega}$ and compute the weighted sum of their inputs $\sum_{c \in \mathsf{in}(n)} \omega_c h_c(\mathsf{sc}(n))$, while product units compute $\prod_{c \in \mathsf{in}(n)} h_c(\mathsf{sc}(n))$. The parameters $\boldsymbol{\Theta}$ of a circuit encompass parameters of input functionals ($\boldsymbol{\lambda}$) and sum units ($\boldsymbol{\omega}$).

For any input $\boldsymbol{y}$, the value of $h_{\boldsymbol{\Theta}}(\boldsymbol{y})$ can be evaluated by propagating the output of the input units through the computational graph and reading out the value of the last unit. The *support* of h is the set of all states $\boldsymbol{y}$ for which the output is non-zero, i.e., $\mathsf{supp}(h) = \{\boldsymbol{y} \in \mathsf{val}(\mathbf{Y}) \mid h(\boldsymbol{y}) \neq 0\}$.

Definition 3.2 (Probabilistic circuits (PCs))**.** A circuit q is a PC if it encodes a (possibly unnormalized) distribution, i.e., $q_{\boldsymbol{\Theta}}(\boldsymbol{y})$ is non-negative for all configurations $\boldsymbol{y}$ of $\mathbf{Y}$.

From here on, we will assume PCs to have positive sum parameters $\boldsymbol{\omega}$ and whose input units model valid distributions, e.g., Bernoullis, as these conditions are sufficient for satisfying Def. 3.2. Moreover, w.l.o.g. we will assume the sum and product units to be organized into alternating layers, and that every product unit n receives only two inputs c_1, c_2, i.e., $q_n(\mathbf{X}) = q_{c_1}(\mathbf{Y}) \cdot q_{c_2}(\mathbf{Y})$. These conditions can easily be enforced in polynomial time [Vergari et al., 2015, 2019]. We are specifically interested in smooth and decomposable PCs, as they will be enabling efficient inference in SPL.

Definition 3.3 (Smoothness & Decomposability)**.** A circuit is *smooth* if the inputs of every sum unit n depend on the same variables: $\forall\, c_1, c_2 \in \mathsf{in}(n), \mathsf{sc}(c_1) = \mathsf{sc}(c_2)$. It is *decomposable* if inputs of every product unit n depend on disjoint sets of variables: $\mathsf{in}(n) = \{c_1, c_2\}, \mathsf{sc}(c_1) \cap \mathsf{sc}(c_2) = \emptyset$.

Smooth and decomposable PCs are both expressive and efficient: they encode distributions with hundred millions of parameters and be effectively learned [Peharz et al., 2020].

As proposed by Shao et al. [2022], a (smooth and decomposable) PC $q_{\boldsymbol{\Theta}}(\mathbf{Y})$ encoding a distribution on $\mathbf{Y}$ can be turned into a (smooth and decomposable) circuit conditioned on an input $\mathbf{X}$ by having its parameters be a function of $\mathbf{X}$.

Definition 3.4 (Neural conditional circuits [Shao et al., 2020])**.** A conditional circuit $q(\mathbf{Y}; \boldsymbol{\Theta} = g(\mathbf{X}))$ models the conditional distribution $p(\mathbf{Y} \mid \mathbf{X})$ via a differentiable gating function g that maps each input $\boldsymbol{x}$ to parameters $\boldsymbol{\Theta}$.

An example of a smooth and decomposable conditional circuit is shown in fig. 3.

3.2 ENCODING LOGICAL FORMULAS WITH CONSTRAINT CIRCUITS

The next step is to translate a logical constraint K into a smooth and decomposable circuit $c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$. To this end, we employ a special type of PCs, defined as follows.

Definition 3.5 (Constraint circuits)**.** A PC c over variables $\mathbf{X} \cup \mathbf{Y}$ is a constraint circuit encoding prior knowledge K if it computes $\mathbb{1}\{(\boldsymbol{x}, \boldsymbol{y}) \models \mathsf{K}\}$ for every configuration $(\boldsymbol{x}, \boldsymbol{y})$.

As a means to realizing such a circuit, we will consider constraint circuits whose sum unit parameters are equal to 1 and input functionals that are indicator functions over their scope, e.g., $c_n(\boldsymbol{z}) = \mathbb{1}\{\boldsymbol{z} \models \varphi(n)\}$ where $\mathbf{Z}$ is the scope of the input and $\varphi(n)$ a constraint over it. Furthermore, we will require that each sum unit in the circuit be *deterministic*.

Definition 3.6 (Determinism)**.** A sum unit n is *deterministic* if its inputs have disjoint supports, i.e., $\forall\, c_1, c_2 \in \mathsf{in}(n), c_1 \neq c_2 \implies \mathsf{supp}(c_1) \cap \mathsf{supp}(c_2) = \emptyset$.

fig. 3 shows an example of a deterministic constraint circuit, and Sec. D illustrates the comilation process in detail.

3.3 EFFICIENT INFERENCE IN SPLS

What remains to be shown to complete SPLs is that the product supports efficient normalization and inferenceTo this end, we need to introduce the notion of compatibility between the two circuits [Vergari et al., 2021].

Definition 3.7 (Compatible circuits in SPLs)**.** A smooth and decomposable conditional PC $q(\mathbf{Y}; \boldsymbol{\Theta})$ is compatible over variables $\mathbf{Y}$ with a smooth and decomposable constraint circuit $c_{\mathsf{K}}(\mathbf{Y}, \mathbf{X})$ if any pair of product units $n \in p$ and $m \in m$ with the same scope over $\mathbf{Y}$ can be rearranged to be mutually compatible and decompose in the same way: $(\mathsf{sc}(n) = \mathsf{sc}(m)) \implies (\mathsf{sc}(n_i) = \mathsf{sc}(m_i),\ n_i$ and m_i are compatible) for some rearrangement of the inputs of n (resp. m) into n_1, n_2 (resp. m_1, m_2). fig. 3 shows two compatible circuits q and c.

Theorem 3.1 (Efficient inference in SPLs)**.** *If $q(\mathbf{Y}; \boldsymbol{\Theta})$ and $c_{\mathsf{K}}(\mathbf{Y}, \mathbf{X})$ are two smooth, decomposable and compatible circuits, then computing eq.* (1) *can be done in $\mathcal{O}(|q||c|)$ time. Furthermore, if they are also deterministic, then computing the MAP state can be done in $\mathcal{O}(|q||c|)$ time.*

The proof, as well as how to come up with compatible circuits can be found in Sec. F and Sec. A, resp.

4 EXPERIMENTS

We evaluate SPLs on standard neuro-symbolic SOP benchmarks such as *simple path prediction*, *preference learning* [Xu et al., 2018], *shortest path finding in Warcraft* [Pogančić et al., 2019] and *HMLC* [Giunchiglia and

Table 1: SPLs outperform all loss-based competitors in the neuro-symbolic benchmarks of [Xu et al., 2018].

	Simple Path			Preference Learning		
Architecture	Exact	Hamming	Consistent	Exact	Hamming	Consistent
MLP+FIL	5.6	85.9	7.0	1.0	**75.8**	2.7
MLP+$\mathcal{L}_{\text{SL}}$	28.5	83.1	75.2	15.0	72.4	69.8
MLP+NeSyEnt	30.1	83.0	91.6	18.2	71.5	96.0
MLP+SPL	**37.6**	**88.5**	**100.0**	**20.8**	72.4	**100.0**

Table 2: SPLs outperform competitors in pathfinding in Warcraft. Predicted paths that do not exactly match the ground truth are still valid paths and yield very close costs to the ground truth. Competitors' predictions can have higher Hamming scores but be invalid. See Sec. G.3 for more examples.

Architecture	Exact	Hamming	Consistent
ResNet-18+FIL	55.0	**97.7**	56.9
ResNet-18+$\mathcal{L}_{\text{SL}}$	59.4	**97.7**	61.2
ResNet-18+SPL	**78.2**	96.3	**100.0**

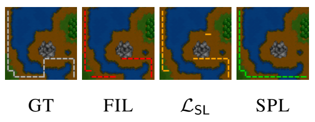


Table 3: Comparison between SPL and HMCNN [Giunchiglia and Lukasiewicz, 2020] on twelve HMLC datasets averaged over 10 runs. Best results for each dataset are in bold. Results which are not significantly worse than the competition, as determined using an unpaired Wilcoxon test, are marked in boldface. Consistency is always 100% for both approaches.

Dataset	Exact Match		Hamming Score	
	HMCNN	MLP+SPL	HMCNN	MLP+SPL
CellCycle	3.05 ± 0.11	$\mathbf{3.79 \pm 0.18}$	$\mathbf{98.26 \pm 0.00}$	97.84 ± 0.06
Derisi	1.39 ± 0.47	$\mathbf{2.28 \pm 0.23}$	$\mathbf{98.32 \pm 0.32}$	97.70 ± 0.07
Eisen	5.40 ± 0.15	$\mathbf{6.18 \pm 0.33}$	$\mathbf{98.09 \pm 0.01}$	97.30 ± 0.04
Expr	4.20 ± 0.21	$\mathbf{5.54 \pm 0.36}$	$\mathbf{98.29 \pm 0.01}$	97.87 ± 0.02
Gasch1	3.48 ± 0.96	$\mathbf{4.65 \pm 0.30}$	$\mathbf{98.37 \pm 0.31}$	97.59 ± 0.05
Gasch2	3.11 ± 0.08	$\mathbf{3.95 \pm 0.28}$	$\mathbf{98.27 \pm 0.00}$	97.94 ± 0.07
Seq	5.24 ± 0.27	$\mathbf{7.98 \pm 0.28}$	$\mathbf{98.31 \pm 0.01}$	97.66 ± 0.03
Spo	$\mathbf{1.97 \pm 0.06}$	$\mathbf{1.92 \pm 0.11}$	$\mathbf{98.23 \pm 0.00}$	98.17 ± 0.03
Diatoms	48.21 ± 0.57	$\mathbf{58.71 \pm 0.68}$	$\mathbf{99.75 \pm 0.00}$	99.64 ± 0.01
Enron	5.97 ± 0.56	$\mathbf{8.18 \pm 0.68}$	$\mathbf{94.10 \pm 0.04}$	93.19 ± 0.13
ImCLEF07A	79.75 ± 0.38	$\mathbf{86.08 \pm 0.45}$	$\mathbf{99.40 \pm 0.01}$	99.35 ± 0.03
ImCLEF07D	76.47 ± 0.35	$\mathbf{81.06 \pm 0.68}$	$\mathbf{98.06 \pm 0.02}$	$\mathbf{98.07 \pm 0.08}$

Lukasiewicz, 2020]. We compare SPLs against several state-of-the-art loss- and layer-based approaches (Sec. E) by applying them to the same base neural network architecture as feature extractor f. As we are interested in measuring how close to the ground truth and how safe the predictions of all models are, we report the percentage of Exact matches of the predicted labels, also called subset accuracy [Tsoumakas and Katakis, 2007], and the percentage of Consistent predictions, also called "Constraint" [Xu et al., 2018]. Note that, like other consistency layers, SPLs are guaranteed to always output 100% consistent predictions. Additionally, we report the Hamming score [Tsoumakas and Katakis, 2007], mainly to maintain compatibility with previous experimental settings [Xu et al., 2018, Ahmed et al., 2022]. This metric does not consider consistency of predictions and favors competitors that assume label independence and thus can minimize the per-label cross-entropy [Dembczyński et al., 2012] (table 2). Sec. G collects all experiment details.

Simple path prediction & preference learning. We start by comparing SPLs against loss-based approaches, reproducing the neuro-symbolic benchmarks of Xu et al. [2018] In the first experiment, given a source and destination node in an unweighted grid $G = (V, E)$, the neural net needs to find the shortest unweighted path connecting them. We consider a 4×4 grid. The input $(\boldsymbol{x}, \boldsymbol{y})$ is a binary vector of length $|V| + |E|$, with the first $|V|$ variables indicating the source and destination nodes, and the subsequent $|E|$ variables indicating a subgraph $G' \subseteq G$. Each label is a binary vector encoding the unique shortest path in G'.

In the preference learning taskthe input consist of the user's preference over 6 sushi types, and the model is to predict the user's preferences (a strict total order) over the remaining 4.

We employ a 5-layer and 3-layer MLP as a baseline for the simple path prediction, and preference learning, respectively, equipped with FIL layer and additionally with the Semantic Loss [Xu et al., 2018] (MLP+$\mathcal{L}_{\text{SL}}$) or its entropic extension [Ahmed et al., 2022] (MLP+NeSyEnt). Table 1 clearly shows that the increased expressiveness of SPL, coming from overparameterizing c_{K}, allows to outperform all competitors while guaranteeing consistent predictions.

Warcraft Shortest Path. Next, we evaluate SPL on the more challenging task of predicting the minimum cost path in a weighted 12×12 grid imposed over terrain maps of Warcraft II [Pogančić et al., 2019]. Each vertex is assigned a cost corresponding to the type of the underlying terrain (e.g., earth has lower cost than water). The minimum cost path between the top left and the bottom right vertices of the grid is encoded as an indicator matrix, and serves as a label. As in [Pogančić et al., 2019] we use a ResNet18 [He et al., 2016] with FIL optionally with $\mathcal{L}_{\text{SL}}$ as a baseline. Given the largest size of the compiled constraint circuit c_{K} in this case 10^{10}, we use a two-circuit implementation of SPL. Results in fig. 1 and table 2 are striking: not only SPL outperforms competitors by a large margin – approx. $+23\%$ over FIL and $+19\%$ over the SL – but also consistently delivers meaningful paths that are very close to the ground truth in terms of cost, even when they encode very different routes. See Sec. G.3 for a gallery of these examples.

Hierarchical Multi-Label Classification. Lastly, we follow the experimental setup of Giunchiglia and Lukasiewicz [2020] and evaluate SPL on 12 real-world HMLC tasks. These tasks are especially challenging due to the limited number of training samples, the large number of output classes and the sparsity of the output space. We compare our single-circuit SPL against HMCNN which was shown to outperform several other state-of-the-art HMLC approaches in Giunchiglia and Lukasiewicz [2020]. We show the effect of increasing the expressiveness of SPL via overparameterization in an ablation test in Sec. G.4. The results in Table 3 highlight that SPL significantly outperforms HMCNN in terms of exact match on 11 data sets and performs comparably on 1,while achieving nearly identical Hamming score.

References

Kareem Ahmed, Eric Wang, Kai-Wei Chang, and Guy Van den Broeck. Neuro-symbolic entropy regularization. *arXiv preprint arXiv:2201.11250*, 2022.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.

Concha Bielza, Guangdi Li, and Pedro Larranaga. Multi-dimensional classification with bayesian networks. *International Journal of Approximate Reasoning*, 52(6): 705–727, 2011.

Hanen Borchani, Gherardo Varando, Concha Bielza, and Pedro Larranaga. A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(5):216–233, 2015.

Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In Marie desJardins and Michael L. Littman, editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press, 2013. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6470`.

YooJung Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic modeling. 2020.

YooJung Choi, Tal Friedman, and Guy Van den Broeck. Solving marginal map exactly by probabilistic circuit transformations. In *International Conference on Artificial Intelligence and Statistics*, pages 10196–10208. PMLR, 2022.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

Tirtharaj Dash, Sharad Chitlangia, Aditya Ahuja, and Ashwin Srinivasan. A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1–15, 2022.

Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1-2):5–45, 2012.

Aryan Deshwal, Janardhan Rao Doppa, and Dan Roth. Learning and inference for structured prediction: A unifying perspective. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.

Luca Di Liello, Pierfrancesco Ardino, Jacopo Gobbi, Paolo Morettin, Stefano Teso, and Andrea Passerini. Efficient generation of structured objects with constrained adversarial networks. *Advances in neural information processing systems*, 33:14663–14674, 2020.

Michelangelo Diligenti, Marco Gori, Marco Maggini, and Leonardo Rigutini. Bridging logic and kernel machines. *Machine learning*, 86(1):57–88, 2012.

Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165, 2017.

Anton Dries, Angelika Kimmig, Wannes Meert, Joris Renkens, Guy Van den Broeck, Jonas Vlasselaer, and Luc De Raedt. Problog2: Probabilistic logic programming. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 312–315. Springer, 2015.

Greg Durrett and Dan Klein. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 302–312. The Association for Computer Linguistics, 2015. doi: 10.3115/v1/p15-1030. URL `https://doi.org/10.3115/v1/p15-1030`.

Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*, pages 1931–1941. PMLR, 2019.

Francesco Giannini, Michelangelo Diligenti, Marco Gori, and Marco Maggini. On a convex logic fragment for learning and reasoning. *IEEE Transactions on Fuzzy Systems*, 27(7):1407–1416, 2018.

Eleonora Giunchiglia and Thomas Lukasiewicz. Coherent hierarchical multi-label classification networks. *Advances in Neural Information Processing Systems*, 33: 9662–9673, 2020.

Eleonora Giunchiglia and Thomas Lukasiewicz. Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72: 759–818, 2021.

Eleonora Giunchiglia, Mihaela Catalina Stoian, and Thomas Lukasiewicz. Deep learning with logical constraints. *arXiv preprint arXiv:2205.00523*, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Nicholas Hoernle, Rafael-Michael Karampatsis, Vaishak Belle, and Ya'akov Gal. Multiplexnet: Towards fully satisfied logical constraints in neural networks. In *AAAI*, 2022.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.

Jurica Levatić, Dragi Kocev, and Sašo Džeroski. The importance of the label hierarchy in hierarchical multi-label classification. *Journal of Intelligent Information Systems*, 45(2):247–271, 2015.

Fayao Liu, Guosheng Lin, and Chunhua Shen. Crf learning with cnn features for image segmentation. *Pattern Recognition*, 48(10):2983–2992, 2015.

Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31, 2018.

Nicholas Mattei and Toby Walsh. PrefLib: A library for preferences. In *International conference on algorithmic decision theory*, pages 259–270. Springer, 2013.

James Mullenbach, Sarah Wiegreffe, Jon Duke, Jimeng Sun, and Jacob Eisenstein. Explainable prediction of medical codes from clinical text. *arXiv preprint arXiv:1802.05695*, 2018.

Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: Backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34, 2021.

Robert Peharz, Sebastian Tschiatschek, Franz Pernkopf, and Pedro Domingos. On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752. PMLR, 2015.

Robert Peharz, Robert Gens, Franz Pernkopf, and Pedro Domingos. On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044, 2016.

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR, 2020.

Knot Pipatsrisawat and Adnan Darwiche. New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 517–522, 2008.

Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019.

Predrag Radivojac, Wyatt T Clark, Tal Ronnen Oron, Alexandra M Schnoes, Tobias Wittkop, Artem Sokolov, Kiley Graim, Christopher Funk, Karin Verspoor, Asa Ben-Hur, et al. A large-scale evaluation of computational protein function prediction. *Nature methods*, 10(3):221–227, 2013.

Claudio Sacca, Stefano Teso, Michelangelo Diligenti, and Andrea Passerini. Improved multi-level protein–protein interaction prediction with semantic-based regularization. *BMC bioinformatics*, 15(1):1–18, 2014.

Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. Neuro-symbolic artificial intelligence: Current trends. *arXiv preprint arXiv:2105.05330*, 2021.

Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *International Conference on Probabilistic Graphical Models*, pages 401–412. PMLR, 2020.

Xiaoting Shao, Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Thomas Liebig, and Kristian Kersting. Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140:298–313, 2022.

Andy Shih and Stefano Ermon. Probabilistic circuits for variational inference in discrete graphical models. *Advances in Neural Information Processing Systems*, 33: 4635–4646, 2020.

Aishwarya Sivaraman, Golnoosh Farnadi, Todd Millstein, and Guy Van den Broeck. Counterexample-guided learning of monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:11936–11948, 2020.

Mohammad S Sorower. A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18:1–25, 2010.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104, 2004.

Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Simplifying, regularizing and strengthening sum-product network structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 343–358. Springer, 2015.

Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. Visualizing and understanding sum-product networks. *Machine Learning*, 108(4):551–573, 2019.

Antonio Vergari, YooJung Choi, Robert Peharz, and Guy Van den Broeck. Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*, 2020.

Antonio Vergari, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. A compositional atlas of tractable circuit operations for probabilistic inference. *Advances in Neural Information Processing Systems*, 34, 2021.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*, 2016.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.

Yu Zhang, Zhenghua Li, and Min Zhang. Efficient Second-Order TreeCRF for Neural Dependency Parsing. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 3295–3305. Association for Computational Linguistics, 2020a. doi: 10.18653/v1/2020.acl-main.302. URL `https://doi.org/10.18653/v1/2020.acl-main.302`.

Yu Zhang, Houquan Zhou, and Zhenghua Li. Fast and accurate neural CRF constituency parsing. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4046–4053, 2020b. doi: 10.24963/ijcai.2020/560. URL `https://doi.org/10.24963/ijcai.2020/560`.

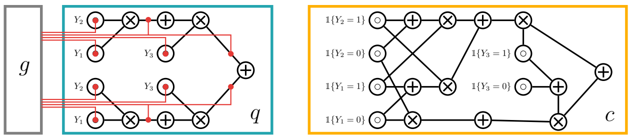


Figure 3: **Examples of circuits in SPL**. **Left**: probabilistic circuit $q_{\mathbf{\Theta}}$. Red lines indicate how the parameters $\mathbf{\Theta}$ flow from the gating function g to the various input functional and sum units of q. **Right**: constraint circuit encoding the logical constraint $(Y_{\mathsf{cat}} \implies Y_{\mathsf{animal}}) \wedge (Y_{\mathsf{dog}} \implies Y_{\mathsf{animal}})$ where labels are $Y_i \in \{Y_{\mathsf{cat}}, Y_{\mathsf{dog}}, Y_{\mathsf{animal}}\}$. Note that q and c are smooth, decomposable (Def. 3.3) and compatible (Def. 3.7) and c is deterministic (Def. 3.6). By parameterizing c via g we can obtain a single-circuit SPL (Sec. B).

A CONSTRUCTING COMPATIBLE CIRCUITS

How do we come up with compatible circuits? One option is to have a PC q that is compatible with every possible smooth and decomposable circuit c. To do so, we can represent q as a mixtures of M fully-independent models; i.e., $\sum_{i=1}^{M} \omega_i \prod_j q(Y_j; \mathbf{\Theta}_i)$. This additional sum unit can be enough to be more expressive than a FIL and already delivers more accurate predictions than any competitor, as our experiments in pathfinding show (Sec. 4). An example of such a circuit is shown in fig. 3. Another sufficient condition for compatibility is that both q and c share the exact same hierarchical scope partitioning [Vergari et al., 2021], sometimes called a vtree or variable ordering [Choi et al., 2020, Pipatsrisawat and Darwiche, 2008]. This can be done easily if one first compiles logical constraints into OBDDs or SDDs and then uses a mechanized algorithm to build q as in [Peharz et al., 2020] to create a compatible structure. Additionally, to ensure q is a deterministic PC, we could exploit the mechanized construction proposed in Shih and Ermon [2020]. Computing the exact MAP state, however, is of less concern as approximate inference algorithms, e.g., beam search decoding [Vijayakumar et al., 2016] or iterative pruning Choi et al. [2022], are nowadays a commodity in deep learning frameworks. For non-deterministic PCs, we compute the MAP state with a faster approximation by replacing non-deterministic sum units with max units Peharz et al. [2016]. This runs in time linear in the size of r, and yet delivers state-of-the-art accuracies in our experiments Sec. 4.

B A SINGLE-CIRCUIT SPL

The two-circuit design we proposed for SPLs provides a clear and theoretically-backed interface between neural networks and probabilistic and symbolic reasoning. This setup, however, can sometimes be wasteful, as it requires to compute the product of two circuits and renormalize. We circumvent this issue by designing a single-circuit implementation of SPL.

Definition B.1 (Single-circuit SPL)**.** Given an input configuration $\boldsymbol{x}$, a single-circuit SPL computes $p(\boldsymbol{y} \mid \boldsymbol{x}) = c_{\mathsf{K}}(\mathbf{Y}, \mathbf{X}; \boldsymbol{\Omega} = g(f(\mathbf{X})))$ where c_{K} is a neural conditional constraint circuit whose sum-unit parameters $\boldsymbol{\Omega}$ are non-unitary values parameterized via a gating function g.

In a nutshell, we can directly realize SPL by compiling a complex logical constraints into a deterministic constraint circuit c_{K}, as before, and then parameterizing it with a gating function of the network embeddings $f(\mathbf{X})$, i.e., allowing its sum units to be non-unitary and input dependent. Since the support of c_{K} is already restricted to exactly match the constraint K, parameterizing $\boldsymbol{\Omega}$ induces an expressive probability distribution over the label configurations that are consistent with K. We can further guarantee that the circuit's output are normalized probabilities by enforcing the parameters ω of each sum unit to form a convex combination [Peharz et al., 2015]. This can be easily done by utilizing a softmax activation function for g.

One of the advantages of the two-circuit implementation of SPLs is that the size of the circuit $q_{\boldsymbol{\Theta}}$ can be easily increased to improve the capacity of the model (Sec. 3.1). The single-circuit implementation is not as flexible, as normally the number of parameters is determined by the complexity of the constraint circuit, which depends entirely on the compilation step. In this case, one option is to *overparameterize* the neural conditional circuit by introducing additional sum units, hence allowing it to capture more modes in the distribution. We detail this process is Sec. C. A side effect of overparameterization is that it relaxes determinism, meaning that MAP inference needs to be approximated, as described in Sec. 3.3. Additionally, training a gating function to map relatively small embeddings to large parameter vectors in overparameterized circuits, can slow down training. If so, a two-circuit implementation of SPL is to be preferred.

C OVERPARAMETERIZING THE SINGLE-CIRCUIT SPL

As mentioned in Def. B.1, SPLs can be realized as a single circuit by first compiling a complex logical constraint into a deterministic constraint circuit, and then parameterizing it using a gating function of the network embeddings. Intuitively, this parameterization induces a probability distribution over the possible solutions of a logical formula encoded in the constraint circuit. The expressiveness of this distribution depends on the number of parameters of the constraint circuit, i.e., the number of weighted edges associated to sum units. As we would like to endow our single-circuit SPL with the ability to induce complex distributions, we devise *two strategies* to introduce more parameters than what the constraint circuit alone can offer: *replication* and *mixture multiplication*.

Algorithm 1 OVERPARAMETERIZE(c, k, cache, first_call)

```
 1: Input: a smooth, deterministic, and structured-decomposable circuit $c$ over variables $\mathbf{X}$, an overparameterization factor $k$, and a cache for memoization, and a flag to denote the first call
 2: Output: an overparameterized, smooth, and structured-decomposable circuit $c$ over $\mathbf{X}$
 3: if $\boldsymbol{q} \in$ cache then
 4:     return cache $[\boldsymbol{q}]$
 5: if $c$ is an input unit then
 6:     nodes $\leftarrow [c]$
 7: else if $c$ is a sum unit then
 8:     elements $\leftarrow [\,]$
 9:     //For every product unit that is an input of $c$
10:     //recursively overparameterize its inputs,
11:     //which are sum units, and take their cross (cartesian) product
12:     for $(c_L, c_R) \in$ in$(c)$ do
13:         left $\leftarrow$ OVERPARAMETERIZE$(c_L, k)$
14:         right $\leftarrow$ OVERPARAMETERIZE$(c_R, k)$
15:         elements.APPEND([CROSSPRODUCT(left, right)]
16:     in$(c) \leftarrow$ elements
17:     nodes $= [c] + [$COPY$(c)$ for $i = 1$ to $k]$
18: if first_call then
19:     //Create a sum unit whose inputs are nodes
20:     //and whose parameters are 1s.
21:     nodes $\leftarrow$ SUM(nodes, $\{1\}_{i=1}^{|\text{nodes}|}$)
22: cache$(c) \leftarrow$ nodes
23: return nodes
```

Replication works by maintaining m copies of the circuit, and taking their weighted average, i.e., introducing a sum unit that mixes them [Peharz et al., 2020]. Mixture multiplication, instead, substitutes a single local marginal distribution encoded by a sub-circuit rooted into a sum unit with k mixture models over the same scope. In practice, we create $k-1$ copies of each sum units and rewire them by computing a cross product of their inputs as in Peharz et al. [2020]. algorithm 1 formalizes this process.

As mentioned in Def. B.1, both strategies relax determinism. However, note that *they do not alter the support of the underlying distribution*. This guarantees that all the predictions will be consistent with the encoded constraint (D3) (Sec. 2).

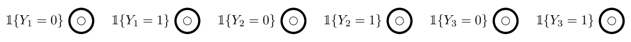
D COMPILING LOGICAL FORMULAS INTO CONSTRAINT CIRCUITS

For our experiments we use standard compilation tools to obtain a constraint circuit starting from a propositional logical formula in conjunctive normal form. Specifically, we use Graphillion[1] to compile the constraints in the Warcraft pathfinding experiment into an SDD. For all other experiments, we use PySDD[2] a python SDD compiler [Darwiche, 2011, Choi and Darwiche, 2013].

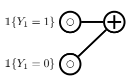
We now illustrate step-by-step one example of such a compilation for a simple logical formula. Consider the constraint circuit c in fig. 3 encoding the constraint

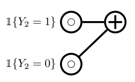
$$(Y_{\mathsf{cat}} \implies Y_{\mathsf{animal}}) \land (Y_{\mathsf{dog}} \implies Y_{\mathsf{animal}}). \tag{2}$$

Intuitively, our aim is to compile the above logical formula into a *compact* form representing all possible assignments to $Y_{\mathsf{cat}}, Y_{\mathsf{dog}}, Y_{\mathsf{animal}}$ satisfying the above constraint. We compile such a constraint by proceeding in a bottom up fashion, where bottom-up compilation can be seen as composing Boolean sub-functions whose domain is determined by a variable ordering, also called vtree (see Sec. 3.3). In this example, we assume the function $f(Y_{\mathsf{animal}}, Y_{\mathsf{cat}}, Y_{\mathsf{dog}})$ decomposes as $f_1(Y_{\mathsf{animal}}) \cdot f_2(Y_{\mathsf{dog}}) \cdot f_3(Y_{\mathsf{cat}})$ We therefore start by compiling a constraint circuit that is a function of Y_{cat} and Y_{dog}, and compose it with a constraint circuit that is a function of Y_{animal} We first introduce input functionals representing indicators associated with $Y_{\mathsf{cat}}, Y_{\mathsf{dog}}, Y_{\mathsf{animal}}$. We will denote by Y_i the indicator $\mathbb{1}\{Y_i = 1\}$ and by $\neg Y_i$ the indicator $\mathbb{1}\{Y_i = 0\}$.



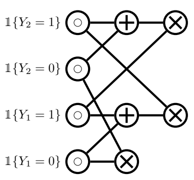
We start by disjoining the indicators Y_{cat} with $\neg Y_{\mathsf{cat}}$, and Y_{dog} with $\neg Y_{\mathsf{dog}}$. This corresponds to introducing deterministic and smooth sum units in our circuits.





These units represent *disjoint solutions* to the logical formula, meaning there exists distinct assignments, characterized by the children, that satisfy the logical constraint e.g. $Y_{\mathsf{cat}}, Y_{\mathsf{dog}}, Y_{\mathsf{animal}}$ and $Y_{\mathsf{cat}}, \neg Y_{\mathsf{dog}}, Y_{\mathsf{animal}}$ are two distinct assignments that satisfy the logical constraint.

The compilation process proceeds by conjoining the constraint circuits for $Y_{\mathsf{dog}} \lor \neg Y_{\mathsf{dog}}$ with Y_{cat}, Y_{dog} with $Y_{\mathsf{cat}} \lor \neg Y_{\mathsf{cat}}$, and $\neg Y_{\mathsf{dog}}$ with $\neg Y_{\mathsf{cat}}$.



A decomposable product units *composes* functions over disjoint sets of variables. The above three product nodes represent the Boolean functions $(Y_{\mathsf{dog}} \lor \neg Y_{\mathsf{dog}}) \land Y_{\mathsf{cat}}$, $Y_{\mathsf{dog}} \land (Y_{\mathsf{cat}} \lor \neg Y_{\mathsf{cat}})$, and $\neg Y_{\mathsf{dog}} \land \neg Y_{\mathsf{cat}}$.

[1]https://github.com/takemaru/graphillion

[2]https://github.com/wannesm/PySDD

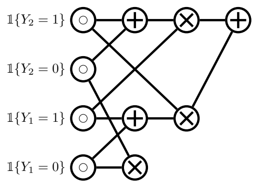
We again disjoin $(Y_{\mathsf{dog}} \lor \neg Y_{\mathsf{dog}}) \land Y_{\mathsf{cat}}$ with $Y_{\mathsf{dog}} \land (Y_{\mathsf{cat}} \lor \neg Y_{\mathsf{cat}})$, and $\neg Y_{\mathsf{dog}} \land \neg Y_{\mathsf{cat}}$ with true, the logical multiplicative identity, guaranteeing alternating sum and product units.

So far, we have compiled constraint circuits for the logical formula

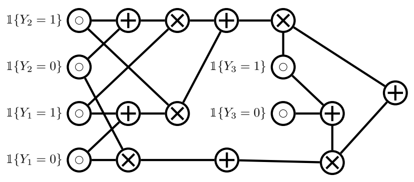
$$((Y_{\mathsf{dog}} \lor \neg Y_{\mathsf{dog}}) \land Y_{\mathsf{cat}}) \lor (Y_{\mathsf{dog}} \land (Y_{\mathsf{cat}} \lor \neg Y_{\mathsf{cat}})) \tag{3}$$

and the logical formula

$$\neg Y_{\mathsf{dog}} \land \neg Y_{\mathsf{cat}} \tag{4}$$



What remains is to conjoin eq. (3) with Y_{animal}, and eq. (4) with $\neg Y_{\mathsf{animal}}$, and disjoin the resulting constraint circuits. What we get is a a mixture distribution over the possible solutions of the constraint: If we predict there is a dog or a cat, or both, in e.g., an image, we better predict that there's an animal. On the other hand, the absence of a dog and a cat from an image implies nothing as to the presence of an animal in the image.



Compilation techniques like the one we illustrated do not, however, escape the hardness of the problem: the compiled circuit can be exponential in the size of the constraint, *in the worst case*. *In practice*, nevertheless, we can obtain compact circuits because real-life logical constraints exhibit enough structure (e.g., they encode repeated sub-problems) that can be easily exploited by a compiler. We refer to the literature of compilation for details on this [Darwiche and Marquis, 2002].

E RELATED WORKS

In this section, we position SPLs against state-of-the-art In-depth surveys on this topic can be found in [Dash et al., 2022] and [Giunchiglia et al., 2022].

Energy-based models. Deep energy-based models (EBMs) replace FILs with an unnormalized factor graph [Koller and Friedman, 2009] that captures higher-order label dependencies LeCun et al. [2006] but at the cost of foregoing probabilistic semantics and efficiency. EBMs are typically unconcerned with hard constraints. Neural approaches for segmentation [Liu et al., 2015] and parsing [Durrett and

Klein, 2015, Zhang et al., 2020a,b] remedy to this by replacing the factor graph with a full-fledged intractable (discriminative) graphical model [Koller and Friedman, 2009]. To gain efficiency, one can restrict EBMs to simpler graphical models (e.g., chains, trees), compromising expressiveness and the ability to model non-trivial logical constraints.

Loss-based methods. A prominent strategy consists of penalizing the network for producing inconsistent predictions using an auxiliary loss [Dash et al., 2022, Giunchiglia et al., 2022]. While popular, loss-based methods, however *cannot* guarantee that the predictions will be consistent at test time. Common losses include translating logical constraints into a differentiable fuzzy logic [Diligenti et al., 2012, 2017], as exemplified by DL2 Fischer et al. [2019]. Although efficient, this solution is not probabilistically sound and crucially *is not syntax-invariant*: different encodings of the same formula (e.g., conjunctive vs. disjunctive normal form) yield different losses Giannini et al. [2018], Di Liello et al. [2020]. Closer to our SPL, the Semantic Loss (SL) [Xu et al., 2018] avoids this issue by penalizing the probability θ_i associated to the i-th label by the neural network via the loss term$-\sum_{\boldsymbol{y}} \prod_i q(Y_i; \theta_i) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$. When K is compiled into a constraint circuit c_{K} one retrieves $-\mathcal{Z}(\boldsymbol{x})$ for a two-circuit version of SPL that is as expressive as FIL as it assumes independent labels via a conditional PC $\prod_i q(Y_i; \theta_i)$. The neuro-symbolic entropy (NeSyEnt) [Ahmed et al., 2022] extends $\mathcal{L}_{\mathsf{SL}}$by an entropy term that improves consistency, but continues to assume conditional independence of labels.

Consistency layers. Approaches ensuring consistency by embedding the constraints into the predictive layer as in SPLs include MultiplexNet [Hoernle et al., 2022] and HMCCN [Giunchiglia and Lukasiewicz, 2020]. MultiplexNet is able to encode only constraints in disjunctive normal form, which is problematic for generality and efficiency as neuro-symbolic SOP tasks involve an intractably large number of clauses – e.g. our pathfinding experiments involves billions of clauses. HMCCN encodes label dependencies as fuzzy relaxation and is the current state-of-the-art model for HMLC [Giunchiglia and Lukasiewicz, 2020]. A recent extension [Giunchiglia and Lukasiewicz, 2021] is also restricted to a certain family of constraints that can be represented with fuzzy logic.

Other approaches. Other common approaches to neuro-symbolic SOP require to invoke a solver to either obtain the MAP state or to compute (often only approximately) the gradient of the loss [Deshwal et al., 2019, Pogančić et al., 2019, Niepert et al., 2021]. SPLs have no such requirement. Some neuro-symbolic approaches [Sarker et al., 2021] constrain the outputs of neural networks within complex logical reasoning pipelines to solve tasks harder than neuro-symbolic SOP. For instance, DeepProblog [Manhaeve et al., 2018] uses Prolog's backward chaining algorithm for first order logical rules whose probabilistic weights are predicted by the network. In modern implementations of Problog, grounding a first order program and then compiling it into constraint circuits [Dries et al., 2015] produces a conditional circuit akin to those we use in SPLs, but in which (*i*) only input distributions are parameterized and (*ii*) increasing the parameter count is not considered straightforward.

F PROOFS

Theorem 3.1 (Efficient inference in SPLs). *If* $q(\mathbf{Y}; \boldsymbol{\Theta})$ *and* $c_{\mathsf{K}}(\mathbf{Y}, \mathbf{X})$ *are two smooth, decomposable and compatible circuits, then computing eq.* (1) *can be done in* $\mathcal{O}(|q||c|)$ *time, where* $|\cdot|$ *denotes the circuit size. Furthermore, if they are also deterministic, then computing the MAP state can be done in* $\mathcal{O}(|q||c|)$ *time. .*

We prove the first statement by first showing that the partition function $\mathcal{Z}(\boldsymbol{x})$ in eq. (1) can solved exactly in time $\mathcal{O}(|q||c|)$. It will then follow from it that computing eq. (1) can be done in $\mathcal{O}(|q||c| + |q| + |c|) \approx \mathcal{O}(|q||c|)$ where the last two additive factors derive from evaluating q and c for an input configuration $(\boldsymbol{x}, \boldsymbol{y})$.

To do so, we will exploit two ingredients: i) the product of q and c can be represented as a smooth and decomposable circuit in time $\mathcal{O}(|q||c|)$ [Vergari et al., 2021] and ii) any smooth and decomposable circuit guarantees tractable marginalization in time linear in its size [Choi et al., 2020]. The next two propositions formalize these statements.

Proposition F.1 (Tractable product of circuits). *Let* $q(\mathbf{Y}; \boldsymbol{\Theta})$ *and* $c_{\mathsf{K}}(\mathbf{Y}, \mathbf{X})$ *be two smooth, decomposable circuits that are compatible over* $\mathbf{Y}$ *then computing their product as a circuit* $r_{\boldsymbol{\Theta},\mathsf{K}}(\mathbf{X}, \mathbf{Y}) = q(\mathbf{Y}; \boldsymbol{\Theta}) \cdot c_{\mathsf{K}}(\mathbf{Y}, \mathbf{X})$ *that is decomposable over* $\mathbf{Y}$ *can be done in* $\mathcal{O}(|q||c|)$*. If both* q *and* c *are also deterministic, then* r *is as well.*

Proof. The proof directly follows from Theorem 3.2 from Vergari et al. [2021]. □

Note that $\mathcal{O}(|q||c|)$ is a loose upperbound and the size of r is in practice smaller [Vergari et al., 2021]. **??** shows as an example the circuit obtained by multiplying the PC and the constraint circuit in fig. 3.

Proposition F.2 (Tractable marginalization of circuits). *Let* $r(\mathbf{X}, \mathbf{Y})$ *be a circuit that is smooth and decomposable over* $\mathbf{Y}$ *with input functions over* $\mathbf{Y}$ *that can be tractably marginalized out. Then for any variables* $\mathbf{Y}' \subseteq \mathbf{Y}$ *and their assignment* $\boldsymbol{y}'$*, the marginalization* $\sum_{\boldsymbol{y}'} r(\boldsymbol{y}', \boldsymbol{y}'', \boldsymbol{x})$ *can be computed exactly in time linear in the size of* r*, where* $\mathbf{Y}'' = \mathbf{Y} \setminus \mathbf{Y}'$*.*

Proof. The proof follows by considering that i) the input functionals in SPLs are simple distributions such as Bernoullis and indicators and can be easily marginalized in $\mathcal{O}(1)$ and ii) that for every configuration $\boldsymbol{x}$ of variables $\mathbf{X}$,

Table 4: A comparison of the performance of single-circuit SPL with different parameters: m, the number of circuit copies in our replication strategy; *gates*, the number of layers in the gating function; and k the overparameterization factor in the mixture multiplication strategy (algorithm 1). We report the percentage of exact matches of the predicted labels on the validation set of the *HMLC* dataset, highlighting the best numbers in **boldface**. As can be seen, all datasets benefit from overparameterization.

DATASET	m: 2				m: 4				m: 8			
	GATES: 2		GATES: 4		GATES: 2		GATES: 4		GATES: 2		GATES: 4	
	k: 2	k: 4	k: 2	k: 4	k: 2	k: 4	k: 2	k: 4	k: 2	k: 4	k: 2	k: 4
CELLCYCLE	4.25	4.48	4.48	4.01	4.60	**4.83**	4.25	4.48	4.36	4.13	4.36	4.13
DERISI	2.26	2.02	2.14	2.26	**2.49**	2.26	2.38	2.38	**2.49**	2.38	2.26	**2.49**
EISEN	6.05	6.05	6.05	6.05	5.86	6.43	**6.81**	6.24	6.43	6.43	6.05	6.43
EXPR	5.42	4.83	5.18	5.30	4.83	**5.54**	**5.54**	5.18	**5.54**	5.42	5.18	5.42
GASCH1	5.56	5.79	5.67	5.91	5.44	5.67	6.03	**6.26**	5.79	5.79	**6.26**	6.03
GASCH2	4.00	4.24	4.83	**4.95**	4.12	4.00	4.12	4.36	4.24	3.53	4.24	4.59
SEQ	7.74	7.74	7.51	7.85	8.19	7.28	7.96	7.17	7.96	7.39	7.51	**8.42**
SPO	2.27	2.15	2.15	2.51	2.39	2.27	2.51	2.51	**2.87**	2.27	2.39	2.63
DIATOMS	53.71	**54.68**	50.16	51.29	53.23	52.10	49.35	48.23	52.90	52.58	46.61	47.26
ENRON	19.53	18.52	17.85	19.87	19.87	20.20	**20.54**	20.20	19.53	20.20	19.53	19.87
IMCLEF07A	86.97	87.03	86.27	86.60	87.00	**87.33**	86.50	86.70	87.07	86.90	87.00	86.83
IMCLEF07D	85.93	85.80	85.87	85.73	85.60	**86.50**	85.87	85.90	85.87	85.83	86.10	85.50

$r(\mathbf{Y}, \boldsymbol{x})$ is a circuit only over $\mathbf{Y}$ and therefore Proposition 2.1 from Vergari et al. [2021] can be directly applied. $\square$

Analogously, the second statement of theorem 3.1 follows from proposition F.1 and recalling the MAP state of a deterministic circuit can be computed in time linear in its size.

Proposition F.3 (Tractable MAP state of circuits (Choi et al. [2020]))**.** *Let $r(\mathbf{X}, \mathbf{Y})$ be a circuit that is smooth, decomposable and deterministic over $\mathbf{Y}$ then for a configuration $\boldsymbol{x}$ its MAP state $\text{argmax}_{\boldsymbol{y}}\, r(\boldsymbol{x}, \boldsymbol{y})$ can be computed in time $\mathcal{O}(|r|)$.*

G ADDITIONAL EXPERIMENTAL DETAILS

G.1 SIMPLE PATH PREDICTION AND PREFERENCE LEARNING

In the simple path prediction task, given a source and destination node in an unweighted grid $G = (V, E)$, the neural net needs to find the shortest unweighted path connecting them. We consider a 4×4 grid. The input $(\boldsymbol{x}, \boldsymbol{y})$ is a binary vector of length $|V| + |E|$, with the first $|V|$ variables indicating the source and destination nodes, and the subsequent $|E|$ variables indicating a subgraph $G' \subseteq G$. Each label is a binary vector of length $|E|$ encoding the unique shortest path in G'. For each example, we obtain G' by dropping one third of the edges in the graph G uniformly at random, filtering out the connected components with fewer than 5 nodes, to reduce degenerate cases, and then sample a source and destination node uniformly at random from G'. The dataset consists of 1600 such examples, with a 60/20/20 train/validation/test split.

In the preference learning task, given a user's ranking over a subset of items, the network has to predict the user's ranking over the remaining items. We encode an ordering over n items as a binary matrix Y_{ij}, where for each $i, j \in 1, \ldots, n$, Y_{ij} indicates whether item i is the jth element in the ordering. The input $\boldsymbol{x}$ consist of the user's preference over 6 sushi types, and the model has to predict the user's preferences (a strict total order) over the remaining 4. We use preference ranking data over 10 types of sushi for $5,000$ individuals, taken from [Mattei and Walsh, 2013], and a 60/20/20 split.

We follow Xu et al. [2018] in employing a 5-layer with 50 hidden units each and sigmoid activation functions, and 3-layer MLP with 50 hidden units each as a baseline for the simple path prediction, and preference learning, respectively. We equip this baselines with a FIL and additionally with the Semantic Loss [Xu et al., 2018] (MLP+$\mathcal{L}_{\text{SL}}$) or its entropic extension [Ahmed et al., 2022] (MLP+NeSyEnt).

We compile the logical constraints into an SDD [Darwiche, 2011] and then turn it into a constraint circuit c_{K} that is used for $\mathcal{L}_{\text{SL}}$, NeSyEnt (Sec. E) and our 1-circuit implementation of SPLs. To obtain the results for SPL in Table 1, we perform a grid search over the using the validation set for a maximum of 2000 iterations, similar to Xu et al. [2018]. We search over the learning rates in the range $\{1\times10^{-3}, 5\times10^{-3}, 1\times10^{-4}, 5\times10^{-4}\}$, the overparameterization factor k in the range $\{2, 4, 8\}$, as well as the number of circuit mixtures m in the range $\{2, 4, 8\}$, evaluating the model with the best performance on the validation.

G.2 HIERARCHICAL MULTI-LABEL CLASSIFICATION

We follow the experimental setup of Giunchiglia and Lukasiewicz [2020] and evaluate SPL on 12 real-world HMLC tasks spanning four different domains: 8 functional genomics, 2 medical images, 1 microalgea classification, and 1 text categorization. These tasks are especially challenging due to the limited number of training samples, the large number of output classes, ranging from 56 to 4130, as well as the sparsity of the output space. We used the same train-validation-test splits and experimental setup as [Giunchiglia and Lukasiewicz, 2020]. For numeric features we replaced missing values by their mean, and for categorical features by a vector of zeros, and standardized all features. We used the validation splits to determine the number of layers in the gating function in the range $\{2, 4, 8\}$, the overparameterization factor in the range $\{2, 4, 8\}$, and the number of mixtures in the range $\{2, 4, 8\}$, keeping all other hyperparameters fixed. The final models were obtained by training using a batch size of 128 and early stopping with a patience of 20 on the validation set.

G.3 WARCRAFT PATHFINDING

We evaluate SPL on the more challenging task of predicting the minimum cost path in a weighted 12×12 grid imposed over terrain maps of Warcraft II [Pogančić et al., 2019]. Our setting differs from the one proposed by Pogančić et al. [2019] in two ways: i) a node only neighbors four nodes as instead of eight, excluding the diagonals; ii) the neural network predicts the edges in the path, as opposed to the vertices, resolving ambiguities in the previous task (note that a set of vertices can *might* ambiguously encode more than one path). Each vertex is assigned a cost corresponding to the type of the underlying terrain (e.g., earth has lower cost than water). The minimum cost path between the top left and the bottom right vertices of the grid is encoded as an indicator matrix, and serves as a label.

We use Graphillion[3] to compile the path constraint, limiting our constraint to the set of paths whose length is less than 29, as determined on the training set.

As in [Pogančić et al., 2019] we use a ResNet18 [He et al., 2016] with FIL optionally with $\mathcal{L}_{\mathsf{SL}}$ as a baseline. Given the largest size of the compiled constraint circuit c_{K} in this case 10^{10}, we use a two-circuit implementation of SPL. We use the identity function as our gating function and do a grid search over only the number of mixtures in the range $\{2, 4, 8\}$ in our model, keeping all other hyperparameters as proposed in [Pogančić et al., 2019].

[3] https://github.com/takemaru/graphillion

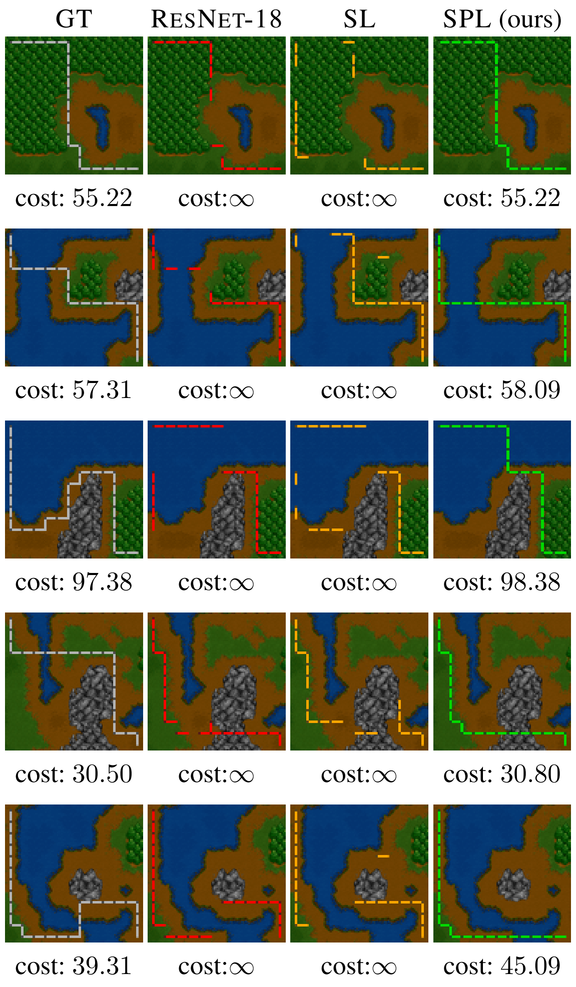


Figure 4: **More examples of shortest path predictions in SPLs and competitors.** SPLs always deliver valid paths and when these do not exact match the ground truth, they are very close in terms of their global cost. Paths from the baselines might yield a higher Hamming score (due to more overlapping edges with the ground truth) but are invalid.

G.4 A STUDY ON THE EFFECT OF OVERPARAMETERIZATION IN SPL

We now illustrate the effect that overparameterization has on the performance of the single-circuit SPL. To that end, we performed an ablation study, comparing single-circuit SPLs comprising a different number of circuit copies m for our replication strategy, a different number of layers in the gating function, denoted by *Gates*, and the overparameterization factor k as used in algorithm 1 in our mixture multiplication strategy.

We report the exact match percentage of the predicted labels on the validation set of the 12 HMLC datasets in **??**. As a general trend, we can see that our overparameterization strategies pay off and in general more mixture nodes help

($k = 4$) as well as using more replicas ($m \geq 4$). The effect of employing a deeper gating function is less striking instead, with a two-layer gating function achieving highest performances on 9 datasets.