Semantic Probabilistic Layers for Neuro-Symbolic Learning

Kareem Ahmed
CS Department
UCLA
ahmedk@cs.ucla.edu

Stefano Teso
CIMeC and DISI
University of Trento
stefano.teso@unitn.it

Kai-Wei Chang
CS Department
UCLA
kwchang@cs.ucla.edu

Guy Van den Broeck
CS Department
UCLA
guyvdb@cs.ucla.edu

Antonio Vergari
School of Informatics
University of Edinburgh
avergari@ed.ac.uk

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 all 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 empirically demonstrate that SPLs outperform these competitors in terms of accuracy on challenging SOP tasks including hierarchical multi-label classification, pathfinding and preference learning, while retaining *perfect* constraint satisfaction. Our code is made publicly available on Github at github.com/KareemYousrii/SPL.

1 Introduction

Modularity is among the major factors that propelled the Cambrian explosion of deep learning [35]. 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 of what makes sense*. On the contrary, unconstrained deep classifiers are notorious for leading to predictions that are inconsistent with the logical constraints governing an 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 [73, 8]. Consider for example a classical SOP task such as multi-label classification (MLC) [74]. 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 [6]. In more challenging tasks such as hierarchical MLC (HMLC) [71] or pathfinding [60], 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 [44], but it cannot guarantee that the predictions are always *consistent* with the constraints at inference time [32]. 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

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

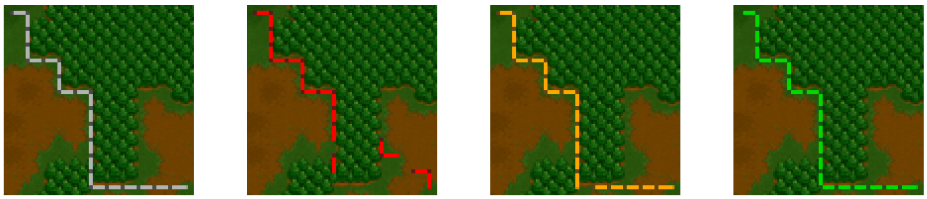


Figure 1: **Neural nets struggle with satisfying validity constraints in complex semantic SOP tasks** such as predicting the lowest-cost path from the top-left to the bottom-right corners of a Warcraft map. Even state-of-the-art neuro-symbolic approaches like the Semantic Loss [80] fail to ensure consistency with hard rules (c). SPLs in contrast guarantees validity while retaining modularity, expressiveness and efficiency. See Sec. 5 for complete experimental details and additional results.

scenarios such as protein function [63] and interaction prediction [65], and drug discovery [20, 24], predicting inconsistent solutions can not only be harmful but also highly expensive [5, 34].

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 [66]. One such attempt has been to learn neural networks that satisfy the logical constraints by explicitly minimizing a differentiable loss term encoding the probability that the networks violates the constraint for a 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 [32, 70] or become intractable for even moderately complex logical constraints [38].

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 output space in a modular and probabilistically sound manner. It does so by leveraging recent advancements in the literature on probabilistic circuits [76, 13]. The key features of SPL are that, on the one hand, it can be used as a *drop-in replacement* for common predictive layers of deep nets like sigmoid layers, and on the other, it *guarantees* the output's consistency with any prespecified logical constraints. Importantly, SPL also supports efficient inference and – perhaps surprisingly – does not complicate training.

Contributions. Summarizing, we: (*i*) Identify six desiderata that neuro-symbolic predictors ought to satisfy to flexibly and reliably support real-world SOP tasks, and show that state-of-the-art approaches fall short of one or more of them (Table 1); (*ii*) Introduce SPL, a novel semantic probabilistic layer that leverages probabilistic circuits to satisfy all six desiderata, i.e., that can be plugged into neural networks to ensure predictions to comply to given logical constraints, while retaining efficiency, expressivity, differentiability, and fully probabilistic semantics; (*iii*) We provide empirical evidence of the effectiveness of SPL in several challenging neuro-symbolic SOP tasks, such as HMLC and pathfinding, where it outperforms state-of-the-art neuro-symbolic approaches, often by a noticeable margin. We implemented SPLs in PyTorch [54], and our code is made publicly available on Github at github.com/KareemYousrii/SPL.

2 Designing a probabilistic layer for neuro-symbolic SOP

Notation. In the following, 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 $\boldsymbol{x}$ to variables $\mathbf{X}$ satisfies a 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 the vector $\boldsymbol{y} \in \{0,1\}^L$. 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, and yet widely adopted [51, 80, 32], 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 refer to this as *fully independent layer* (FIL). In a FIL,

Table 1: **SPL is the only approach to satisfy all the desiderata for neuro-symbolic SOP.** An in-depth discussion of all competitors can be found in Sec. 4.

	Losses			Layers				
Desideratum	DL2 [29]	SL [80]	NeSyEnt [3]	FIL	EBM [43]	MultiplexNet [38]	CCN [33]	SPL *(ours)*
(D1) Probabilistic	✗	✓	✓	✓	✗	✓	✗	✓
(D2) Expressive	✗	✗	✗	✗	✓	✗	✗	✓
(D3) Consistent	✗	✗	✗	✗	✗	✓	✓	✓
(D4) General	✓	✓	✓	✗	✓	✓	✗	✓
(D5) Modular	✓	✓	✓	✓	✓	✓	✓	✓
(D6) Efficient	✓	✓	✓	✓	✗	✗	✓	✓

$p(Y_i = y_i \mid \boldsymbol{z})$ is computed as $\sigma(\mathbf{w}_i^\top \boldsymbol{z})$ where $\mathbf{w}_i \in \mathbb{R}^M$ is a vector of parameters and $\sigma(x)$ is the logistic sigmoid function $1/(1+e^{-x})$.

We are interested in dependencies between labels that can occur both as *correlations*, as is the case in MLC [22], and as *logical constraints* encoded by logical formulas. For example, in a HMLC task [32] one logical constraint can encode the fact that observing a label for the class cat and dog, implies observing the label for their superclass animal

$$(Y_{\mathsf{cat}} = 1 \implies Y_{\mathsf{animal}} = 1) \wedge (Y_{\mathsf{dog}} = 1 \implies Y_{\mathsf{animal}} = 1). \tag{1}$$

Specifically, we assume symbolic knowledge to be supplied in the form of constraints encoded as a logical formula denoted as K and defined over the labels $\mathbf{Y}$ and optionally over a subset of the discrete input variables in $\mathbf{X}$, if any (e.g., in our experiments, the predicted simple path is constrained to lie within the subset of edges appearing in the input graph, see Sec. 5). On the other hand, we expect a model to learn the label correlations from data. We call such task *neuro-symbolic SOP*.

Desiderata for neuro-symbolic SOP. To tackle this setting, we seek an algorithmic strategy for replacing the predictive layer in any neural network classifier with little effort, with the aim of injecting complex symbolic knowledge and allowing for flexible probabilistic reasoning. We formalize these observations into the following six desiderata for our predictive layer:

- D1. **Probabilistic**: The layer should enjoy sound probabilistic semantics, and deliver normalized probabilistic predictions to facilitate maximum-likelihood learning and sound decision making by virtue of calibrated probabilistic predictions
- D2. **Expressive**: It should be able to compactly encode intricate *correlations* between labels.
- D3. **Consistent**: It should always output predictions that are consistent with the prespecified symbolic knowledge, i.e., for all $\boldsymbol{x}$ and $\boldsymbol{y}$, if $(\boldsymbol{x}, \boldsymbol{y}) \not\models \mathsf{K}$ then $p(\boldsymbol{y} \mid \boldsymbol{x}) = 0$.
- D4. **General**: It should support rich *logical constraints* over the labels expressed in some formal language, e.g., propositional logic.
- D5. **Modular**: It should be applicable to any off-the-shelf (and possibly pretrained) neural network in a modular fashion, enabling end-to-end learning and rapid prototyping.
- D6. **Efficient**: The time required by the predictor to compute a prediction should be linear in the size of the predictor and of the hard constraint representation.

For example, FILs are clearly probabilistic (D1), modular (D5), and efficient (D6), but at the cost of being incapable of modeling intricate correlations and logical constraints and thus generating inconsistent predictions (D2–D4) (see also Fig. 1). Table 1 summarizes how the other popular and effective approaches to neuro-symbolic SOP nowadays fall short of one of more desiderata as well. We discuss this in detail in Sec. 4. To the best of our knowledge, our proposed *semantic probabilistic layers* (SPLs) are the first algorithmic solution to satisfy all above desiderata.

SPL. At a high level, SPL realizes the above desiderata in 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.

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:

$$p(\boldsymbol{y} \mid f(\boldsymbol{x})) = q_{\boldsymbol{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x})) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y}) / \mathcal{Z}(\boldsymbol{x}) \qquad \text{where} \quad \mathcal{Z}(\boldsymbol{x}) = \sum\nolimits_{\boldsymbol{y}} q_{\boldsymbol{\Theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y}). \tag{2}$$

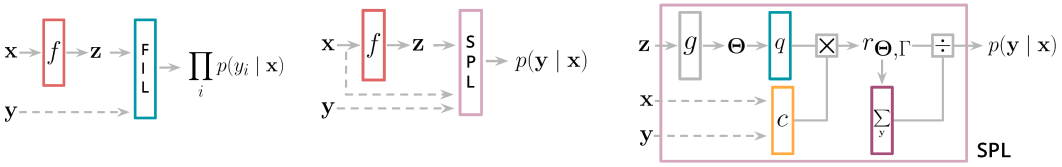


Figure 2: **A high level view of SPLs.** The predictive layer of a neural network for neuro-symbolic SOP, e.g., a FIL (**left**), can be readily replaced by a SPL (**middle**). SPLs are implemented (**right**) by multiplying together a probabilistic circuit $q_{\boldsymbol{\Theta}}(\mathbf{Y} \mid f(\mathbf{X}))$ 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 $r_{\boldsymbol{\Theta},\mathsf{K}}$, so as to guarantee fully probabilistic semantics and end-to-end differentiable learning by maximum likelihood.

Here, $q_{\boldsymbol{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x}))$ is a module to perform probabilistic reasoning by encoding an expressive distribution over the labels parameterized by $\boldsymbol{\Theta}$; $c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$ is a module to ensure consistency of the predictions by encoding logical constraints K and 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 renormalization term, also called the partition function. It is worth noting that this amounts to taking a product of experts [37] which is, in general, hard.

Fig. 2 illustrates the computational graph of our SPL at training time. In order to satisfy all D1-D6, we will realize both $q_{\boldsymbol{\Theta}}$ and c_{K} as *circuits* [76, 13], constrained computational graphs that enable tractable computations. Differently from FILs, $q_{\boldsymbol{\Theta}}$ in SPLs can encode an expressive joint distributions over the labels and therefore attain full expressiveness by scaling the number of parameters $\boldsymbol{\Theta}$ (D2). Consistency is guaranteed by the component c_{K}: by multiplying it to the joint probability of a label configuration the resulting product distribution $r_{\boldsymbol{\Theta},\mathsf{K}}(\boldsymbol{y}, \boldsymbol{x}) = q_{\boldsymbol{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x})) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$ will have its support effectively cut by K, and thus cannot allocate any probability mass to inconsistent predictions (D3). Additionally, c_{K} will allow to encode general propositional logical constraints in a compact computational graph (D4). Lastly, the product $r_{\boldsymbol{\Theta},\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$ is fully differentiable and allows SPL to be an off-the-shelf replacement for other predictive layers (see Fig. 2) and enables end-to-end learning (D5). By renormalizing $r_{\boldsymbol{\Theta},\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y})$ and outputting normalized probabilities, SPL enables the exact computation of gradients for $\boldsymbol{\Theta}$, which can therefore be trained by maximum likelihood (D1).

Thanks to recent advancements in the literature on circuits, we can compute the partition function $\mathcal{Z}(\boldsymbol{x})$ efficiently in time linear in the size of $r_{\boldsymbol{\Theta},\mathsf{K}}$, thus preserving efficiency (D6) and not compromising on the other desiderata. This will also yield correct (and consistent) predictions at test time, when an SPL computes the MAP state $\boldsymbol{y}^* = \operatorname{argmax}_{\boldsymbol{y}} r_{\boldsymbol{\Theta},\mathsf{K}}(\boldsymbol{y}, \boldsymbol{x}) / \sum_{\boldsymbol{y}} r_{\boldsymbol{\Theta},\mathsf{K}}(\boldsymbol{y}, \boldsymbol{x})$. The next section clarifies *how* to implement the modules of SPL as circuits while satisfying these desiderata.

3 Realizing SPLs with tractable circuit representations

The components of SPLs are *circuits*, a large class of computational graphs that can represent both functions and distributions [13, 18]. Circuits subsume many tractable generative and discriminative probabilistic models—from Chow-Liu and latent tree models [14, 11], to hidden Markov models (HMMs) [62], sum-product networks (SPNs) [61], decision trees [41, 15], and deep regressors [40]—as well as many compact representations of logical formulas, such as (ordered) binary decision diagrams [4], sentential decision diagrams (SDDs) [17] and others [18].

The key idea behind SPLs is to leverage this single formalism to represent both an expressive joint distribution for $q_{\boldsymbol{\Theta}}(\boldsymbol{y} \mid f(\boldsymbol{x}))$ 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. (2). This can be achieved by ensuring that these computational graphs abide certain structural properties: *smoothness*, *decomposability*, *determinism* and *compatibility* [18, 75]. Next, we introduce *probabilistic circuits* for modeling $q_{\boldsymbol{\Theta}}$ (Sec. 3.1) and *constraint circuits* for c_{K} (Sec. 3.2), while in Sec. 3.4 we propose a more efficient implementation of SPL utilizing a single circuit.

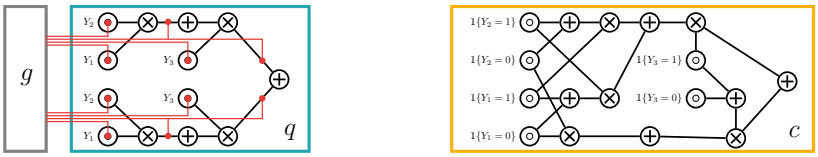


Figure 3: **Examples of circuits in SPL**. **Left**: a neural conditional probabilistic circuit $q_{\boldsymbol{\Theta}}$. Red lines indicate how the output of g parameterizes the input distribution parameters $\boldsymbol{\lambda}$ and the sum unit parameters $\boldsymbol{\omega}$ of q, both indicated as red dots. **Right**: constraint circuit encoding the logical constraint of Eq. (1) 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. 3.4). **Both:** circuits q and c are compatible, as product units with the same scope decompose in the same way. E.g., consider the first two product units of q and c, right to left and top to bottom. Both units decompose $\{Y_3, Y_2, Y_1\}$ into Y_3 and Y_2, Y_1.

3.1 Representing expressive distributions with probabilistic circuits

We start by introducing circuits for *joint* probability distributions, and then extend the discussion to *conditional* distributions, which we use 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 three kinds of 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 $\mathsf{sc}(n) \subseteq \mathbf{Y}$, called its scope, and it 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 the parameters of all input functionals ($\boldsymbol{\lambda}$) and the parameters of 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 $\boldsymbol{y}$ of $\mathbf{Y}$ for which the output is non-zero, i.e., $\mathsf{supp}(h) = \{\boldsymbol{y} \mid h(\boldsymbol{y}) \neq 0\}$.

Definition 3.2 (Probabilistic circuits (PCs))**.** A circuit q is a PC if it encodes a (possibly unnormalized) probability 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 [77, 78]. We are specifically interested in smooth and decomposable PCs, as they will be enabling efficient inference in SPL (Sec. 3.3).

Definition 3.3 (Smoothness & Decomposability)**.** A circuit is *smooth* if for every sum unit n, its inputs 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 the 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 can encode distributions with hundred millions of parameters and be effectively learned by gradient ascent [58]. The structure of their computational graph can be either specified manually [61, 58, 56] or acquired automatically from data [77, 64, 16], e.g., by first learning a latent tree model and then compiling the latter into a circuit [46]. These circuits are competitive with intractable models such as variational autoencoders and normalizing flows scores on several benchmarks [45].

As proposed by Shao et al. [68], any (smooth and decomposable) PC $q_{\boldsymbol{\Theta}}(\mathbf{Y})$ encoding a *joint* distribution over the labels $\mathbf{Y}$ can be turned into a (smooth and decomposable) *conditional* circuit, conditioned by input variables $\mathbf{X}$, by letting its parameters be a function of $\mathbf{X}$.

Definition 3.4 (Neural conditional circuits [67])**.** A conditional circuit $q(\mathbf{Y}; \boldsymbol{\Theta} = g(\mathbf{X}))$ models the conditional distribution $p(\mathbf{Y} \mid \mathbf{X})$ via a differentiable function g that maps every input configuration $\boldsymbol{x}$ to the set of parameters of $\boldsymbol{\Theta}$ of p, also called the *gating function*.

An example of a smooth and decomposable conditional circuit is shown in Fig. 3. This design immediately allows us to implement $q_{\boldsymbol{\Theta}}(\mathbf{Y} \mid f(\mathbf{X}))$ in SPL as a conditional PC whose gating function maps the feature embedding space $\mathbb{R}^K$ to the parameter space $\mathbb{R}_+^{|\boldsymbol{\Theta}|}$, realizing $q(\mathbf{Y}; \boldsymbol{\Theta} = g(f(\mathbf{X})))$. As such, the gating function g creates a clean interface between any pre-trained feature extractor f and the PC q (Fig. 3). While one can devise g in several ways, we strive for simplicity in our experiments and adopt vanilla multi-layer perceptrons (MLPs) whose final activations are either sigmoids, if they have to predict the parameters $\boldsymbol{\lambda}$ of the Bernoulli input distributions of q, or softmax, if they output the sum unit parameters $\boldsymbol{\omega}$ (Def. 3.1).

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 $\{(\boldsymbol{x}, \boldsymbol{y}) \models \mathsf{K}\}$ for every configuration $(\boldsymbol{x}, \boldsymbol{y})$.

As a practical way to realize such a circuit, we will consider constraint circuits that have all sum unit parameters 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 require each sum unit in it to 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. Thanks to determinism, we can readily translate classical compact representations for logical formulas such as (ordered) binary decision diagrams [4, 9] and sentential decision diagrams (SDDs) [17] into constraint circuits as defined above. This becomes evident when they are written in the language of negation normal form [18] and their *and* gates (resp. *or* gates) are replaced with product units (resp. sum units) [13]. A logic constraint can therefore be represented as a constraint circuit for SPLs, by utilizing any of the many tools available for OBDDs [72] or SDDs [10, 53]. Sec. B illustrates in detail how to compile the example constraint of Eq. (1) into the constraint circuit of Fig. 3 in this way.

The worst-case size of the constraint circuit depends on a) the algorithm employed for compilation and, b) the local structure of the constraints, rather than the number of labels. For example, in our Warcraft experiment (see Sec. 5), we have a label configuration space over edges in a 12×12 grid, yielding $2^{12^2} = 2^{144} \approx 10^{43}$ states. However, only 10^{10} configurations satisfy the constraint that these edge labels form a valid path in the grid. If our compilation algorithm were to simply enumerate these configurations, putting them in a logical OR (as done in some neuro-symbolic learners such as MultiplexNet [38]), the size of the constraint circuit, denoted as $|c|$, would be 10^{10}. However, by using recent advancements in compiling logical formulas into constraints circuits, we can can greatly reduce the circuit size. For example, compiler we use [1] generates circuits whose size is worst-case exponential in the treewidth of the CNF representation of the logical formula, but typically much smaller. See Sec. 5 and Sec. F for details.

3.3 Efficient inference in SPLs

As discussed above, PCs can be expressive (D2) and are modular (D5), while constraint circuits ensure consistency (D3) for general constraints (D4). What remains to be shown to complete SPLs is that the product supports efficient normalization (D1) and inference (D6), specifically that it allows for the efficient evaluation of the normalization constant of $r_{\boldsymbol{\Theta},\mathsf{K}}$, and its MAP state. To this end, we need to introduce the notion of compatibility between the two circuits [75].

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 $n \in q$ and $m \in c_{\mathsf{K}}$ 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). The two circuits q and c shown in Fig. 3 are compatible.

Table 2: SPLs outperform all loss-based competitors in the neuro-symbolic benchmarks of [80].

	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 (*ours*)	**37.6**	**88.5**	**100.0**	**20.8**	72.4	**100.0**

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. (2) 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 can be found in Sec. A. 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., $\sum_{i=1}^{M}\omega_i\prod_j q(Y_j;\boldsymbol{\Theta}_i)$. 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. 5). 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 [75], sometimes called a vtree or variable ordering [13, 59].

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 [58] to create a compatible structure. Additionally, to ensure q is a deterministic PC, we could exploit the mechanized construction proposed in Shih and Ermon [69]. Computing the exact MAP state, however, is of less concern as approximate inference algorithms, e.g., beam search decoding [79] or iterative pruning [12], 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 [55]. This runs in time linear in the size of r, and yet delivers state-of-the-art accuracies in our experiments Sec. 5.

3.4 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 3.8 (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 (D4) 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 (D3), parameterizing $\boldsymbol{\Omega}$ induces an expressive probability distribution over the label configurations that are consistent with K (D2). We can further guarantee that the circuit's output are normalized probabilities (D1, D6) by enforcing the parameters ω of each sum unit to form a convex combination [57]. 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. In such cases, a two-circuit implementation of SPL is to be preferred.

4 Related works

In this section, we position SPLs against state-of-the-art approaches for enforcing constraints on neural network predictions. In-depth surveys on this topic can be found in [19] and [34].

Energy-based models. Deep energy-based models (EBMs) replace FILs with an unnormalized factor graph [42] that captures higher-order label dependencies [43] (D2) but at the cost of foregoing probabilistic semantics (D1) and efficiency (D6). EBMs are typically unconcerned with hard constraints (D3). Neural approaches for segmentation [47] and parsing [28, 82, 83] remedy to this by replacing the factor graph with a full-fledged intractable (discriminative) graphical model [42]. To gain efficiency, one can restrict EBMs to simpler graphical models (e.g., chains, trees), compromising expressiveness (D2) and the ability to model non-trivial logical constraints (D3, D4).

Loss-based methods. A prominent strategy consists of penalizing the network for producing inconsistent predictions using an auxiliary loss [19, 34]. 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 [25, 26], as exemplified by DL2 [29]. Although efficient (D6), this solution is not probabilistically sound (D1) and crucially *is not syntax-invariant*: different encodings of the same formula (e.g., conjunctive vs. disjunctive normal form) yield different losses [31, 24]. Closer to our SPL, the Semantic Loss (SL) [80] avoids this issue by penalizing the the probability θ_i associated to the i-th label by the neural network via the loss term

$$\mathcal{L}_{\mathsf{SL}} \propto -\sum_{y \models \mathsf{K}} \prod_{\boldsymbol{y} \models Y_i} \theta_i \prod_{\boldsymbol{y} \not\models Y_i} (1-\theta_i) = -\sum_{\boldsymbol{y} \models \mathsf{K}} \prod_i p(Y_i \mid \boldsymbol{x}) = -\sum_{\boldsymbol{y}} \prod_i q(Y_i; \theta_i) \cdot c_{\mathsf{K}}(\boldsymbol{x}, \boldsymbol{y}).$$

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) [3] extends $\mathcal{L}_{\mathsf{SL}}$ by an entropy term that improves (but still does not guarantee) consistency. It still makes the same independence assumptions over labels (D2).

Consistency layers. Approaches ensuring consistency by embedding the constraints into the predictive layer as in SPLs include MultiplexNet [38] and HMCCN [32]. MultiplexNet is able to encode only constraints in disjunctive normal form, which is problematic for generality (D4) and efficiency (D6) 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 [32]. HMCCN and even its recent extension [33] are restricted to only certain constraints that can be exactly encoded with fuzzy logic easily. SPLs instead can express constraints encoded as arbitrary propositional logical formulas (D4).

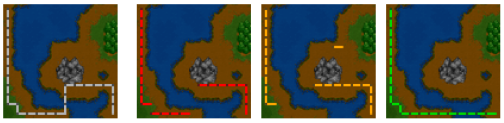
Other approaches. Other common approaches to neuro-symbolic SOP require to invoke a solver to either obtain the MAP state or to compute (often only approximately) the gradient of the loss [23, 60, 52]. SPLs have no such requirement. Some neuro-symbolic approaches [66] constrain the outputs of neural networks within complex logical reasoning pipelines to solve tasks harder than neuro-symbolic SOP. For instance, DeepProblog [48] 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 [27] 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 a straightforward operation. Scallop [39] provides a more scalable approach to deepproblog by considering only the top-k proofs. We leave to future work how we could quickly compile only a specific query as DeepProblog/Scallop do, to deal with first-order representations efficiently.

5 Experiments

We evaluate SPLs on standard neuro-symbolic SOP benchmarks such as *simple path prediction*, *preference learning* [80], *shortest path finding in Warcraft* [60] and *HMLC* [32]. We compare SPLs against several state-of-the-art loss- and layer-based approaches (Sec. 4) 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 [74], and the percentage of Consistent predictions, also called "Constraint" [80]. Note that, like other consistency layers,

Table 3: 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. More examples in Sec. D.3.

ARCHITECTURE	EXACT	HAMMING	CONSISTENT
RESNET-18+FIL	55.0	**97.7**	56.9
RESNET-18+$\mathcal{L}_{\mathsf{SL}}$	59.4	**97.7**	61.2
RESNET-18+SPL (*ours*)	**78.2**	96.3	**100.0**



GROUND TRUTH FIL $\mathcal{L}_{\mathsf{SL}}$ SPL

SPLs are guaranteed to always output 100% consistent predictions. Additionally, we report the HAMMING score [74], mainly to maintain compatibility with previous experimental settings [80, 3]. This metric does not consider consistency of predictions and naturally favors competitors that assume label independence and thus can minimize the per-label cross-entropy [22] (Table 3). Sec. D collects all experimental details such as architectures and hyperparameters used for each experiment.

In Sec. F we provide the average timings for compiling logical formulas into circuits—carried out once, and reused in all subsequent experiments, for parameterizing the conditional circuits, computing the MAP-state of SPL and the loss function at training time (including the cost of computing the product circuit r and its normalization). All these timings, compilation excluded, are reported per batch. We compare to the timings of baselines such as semantic loss and neuro-symbolic entropy, where applicable, to which SPL is highly competitive.

Simple path prediction & preference learning. We start by comparing SPLs against loss-based approaches, reproducing the neuro-symbolic benchmarks of Xu et al. [80] for simple path prediction and preference learning. 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 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, filter 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 [49], and a 60/20/20 split.

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 [80] (MLP+$\mathcal{L}_{\mathsf{SL}}$) or its entropic extension [3] (MLP+NESYENT). We compile the logical constraints into an SDD [17] and then turn it into a the same constraint circuit c_{K} that is used for $\mathcal{L}_{\mathsf{SL}}$, NESYENT (Sec. 4) and our 1-circuit implementation of SPLs. Table 2 clearly shows that the increased expressiveness of SPL, coming from overparameterizing c_{K}, allows to outperform all competitors while guaranteeing consistent predictions, as expected.

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 [60]. 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 [60] we use a ResNet18 [36] 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. Results in Fig. 1 and Table 3 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. D.3 for a gallery of these examples. Concerning times, SPLs are able to compute the likelihood in a mere 14 seconds per batch even on a 10^{10} valid configuration space (Sec. F).

Table 4: Comparison between SPL and HMCNN [32] 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	$\mathbf{1.97 \pm 0.06}$	1.92 ± 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	$\mathbf{81.06 \pm 0.68}$	$\mathbf{98.06 \pm 0.02}$	$\mathbf{98.07 \pm 0.08}$

Hierarchical Multi-Label Classification. Lastly, we follow the experimental setup of Giunchiglia and Lukasiewicz [32] 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. Fig. 3 shows an example of a hierarchy of classes. These tasks are especially challenging due to the limited number of training samples, the large number of output classes, ranging from 56 to 499, as well as the sparsity of the output space. The larger datasets yield a label space of 2^{499} configurations, but we can compile them in seconds into compact constraints circuits of size $\approx$ 108KB (Sec. F).

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 as well as the overparameterization, keeping all other hyperparameters fixed. The final models were obtained by training using a batch size of 128 and early stopping on the validation set. 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 [32]. We study the effect of increasing the expressivenss of SPL via overparameterization in Sec. D.4. The results in Table 5 highlight that SPL significantly outperforms HMCNN in terms of exact match on 11 data sets performing comparably on 1,

6 Conclusion

SPLs offer the first clear interface for integrating complex probabilitistic reasoning and logical constraints on top of any neural network classifier while retaining efficient inference and training. They improve by a noticeable margin the current state-of-the-art on challenging neuro-symbolic SOP benchmarks such as pathfinding and HMLC. This opens up a number of interesting research directions. First, SPLs can be extended to incorporate logical constraints over multiple networks and representable by first-order formulas [48], which we plan to explore in future works, making the circuit construction pipeline totally transparent to users [2] while possibly automatically learning constraints from data [21, 50]. Second, we are interested in leveraging SPLs to inject scalable logical constraints into large language models [7] thus equipping them with probabilistic reasoning [30, 81].

Acknowledgments and Disclosure of Funding

The authors would like to thank Arthur Choi for helpful discussions on compiling the constraints for the Warcraft Shortest Path task, and Andreas Grivas for proofreading a draft manuscript. The research of ST was partially supported by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. This work was funded in part by the DARPA Perceptually-enabled Task Guidance (PTG) Program under contract number HR00112220005,, NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, Samsung, CISCO, and a Sloan Fellowship.

References

[1] (2017). Pysdd. In *Recent Trends in Knowledge Compilation, Report from Dagstuhl Seminar 17381*.

[2] Ahmed, K., Li, T., Ton, T., Guo, Q., Chang, K.-W., Kordjamshidi, P., Srikumar, V., Van den Broeck, G., and Singh, S. (2022a). Pylon: A pytorch framework for learning with constraints. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence (Demo Track)*.

[3] Ahmed, K., Wang, E., Chang, K.-W., and den Broeck, G. V. (2022b). Neuro-symbolic entropy regularization. In *The 38th Conference on Uncertainty in Artificial Intelligence*.

[4] Akers, S. B. (1978). Binary decision diagrams. *IEEE Transactions on computers*, 27(06):509–516.

[5] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

[6] Bielza, C., Li, G., and Larranaga, P. (2011). Multi-dimensional classification with bayesian networks. *International Journal of Approximate Reasoning*, 52(6):705–727.

[7] Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., et al. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

[8] Borchani, H., Varando, G., Bielza, C., and Larranaga, P. (2015). A survey on multi-output regression. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 5(5):216–233.

[9] Bryant, R. E. and Meinel, C. (2002). Ordered binary decision diagrams. In *Logic synthesis and verification*, pages 285–307. Springer.

[10] Choi, A. and Darwiche, A. (2013). Dynamic minimization of sentential decision diagrams. In desJardins, M. and Littman, M. L., editors, *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. AAAI Press.

[11] Choi, M. J., Tan, V. Y., Anandkumar, A., and Willsky, A. S. (2011). Learning latent tree graphical models. *Journal of Machine Learning Research*, 12:1771–1812.

[12] Choi, Y., Friedman, T., and Van den Broeck, G. (2022). Solving marginal map exactly by probabilistic circuit transformations. In *International Conference on Artificial Intelligence and Statistics*, pages 10196–10208. PMLR.

[13] Choi, Y., Vergari, A., and Van den Broeck, G. (2020). Probabilistic circuits: A unifying framework for tractable probabilistic modeling.

[14] Chow, C. and Liu, C. (1968). Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467.

[15] Correia, A. H. C., Peharz, R., and de Campos, C. P. (2020). Joints in random forests. In *NeurIPS*.

[16] Dang, M., Vergari, A., and Van den Broeck, G. (2022). Strudel: A fast and accurate learner of structured-decomposable probabilistic circuits. *International Journal of Approximate Reasoning*, 140:92–115.

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

[18] Darwiche, A. and Marquis, P. (2002). A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264.

[19] Dash, T., Chitlangia, S., Ahuja, A., and Srinivasan, A. (2022). A review of some techniques for inclusion of domain-knowledge into deep neural networks. *Scientific Reports*, 12(1):1–15.

[20] De Cao, N. and Kipf, T. (2018). MolGAN: An implicit generative model for small molecular graphs. *ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*.

[21] De Raedt, L., Passerini, A., and Teso, S. (2018). Learning constraints from examples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

[22] Dembczyński, K., Waegeman, W., Cheng, W., and Hüllermeier, E. (2012). On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88(1-2):5–45.

[23] Deshwal, A., Doppa, J. R., and Roth, D. (2019). Learning and inference for structured prediction: A unifying perspective. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*.

[24] Di Liello, L., Ardino, P., Gobbi, J., Morettin, P., Teso, S., and Passerini, A. (2020). Efficient generation of structured objects with constrained adversarial networks. *Advances in neural information processing systems*, 33:14663–14674.

[25] Diligenti, M., Gori, M., Maggini, M., and Rigutini, L. (2012). Bridging logic and kernel machines. *Machine learning*, 86(1):57–88.

[26] Diligenti, M., Gori, M., and Sacca, C. (2017). Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165.

[27] Dries, A., Kimmig, A., Meert, W., Renkens, J., Broeck, G. V. d., Vlasselaer, J., and Raedt, L. D. (2015). Problog2: Probabilistic logic programming. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 312–315. Springer.

[28] Durrett, G. and Klein, D. (2015). 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.

[29] Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. (2019). Dl2: Training and querying neural networks with logic. In *International Conference on Machine Learning*, pages 1931–1941. PMLR.

[30] Geva, M., Gupta, A., and Berant, J. (2020). Injecting numerical reasoning skills into language models. *arXiv preprint arXiv:2004.04487*.

[31] Giannini, F., Diligenti, M., Gori, M., and Maggini, M. (2018). On a convex logic fragment for learning and reasoning. *IEEE Transactions on Fuzzy Systems*, 27(7):1407–1416.

[32] Giunchiglia, E. and Lukasiewicz, T. (2020). Coherent hierarchical multi-label classification networks. *Advances in Neural Information Processing Systems*, 33:9662–9673.

[33] Giunchiglia, E. and Lukasiewicz, T. (2021). Multi-label classification neural networks with hard logical constraints. *Journal of Artificial Intelligence Research*, 72:759–818.

[34] Giunchiglia, E., Stoian, M. C., and Lukasiewicz, T. (2022). Deep learning with logical constraints. *arXiv preprint arXiv:2205.00523*.

[35] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[36] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[37] Hinton, G. E. (1999). Products of experts. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*.

[38] Hoernle, N., Karampatsis, R.-M., Belle, V., and Gal, Y. (2022). Multiplexnet: Towards fully satisfied logical constraints in neural networks. In *AAAI*.

[39] Huang, J., Li, Z., Chen, B., Samel, K., Naik, M., Song, L., and Si, X. (2021). Scallop: From probabilistic deductive databases to scalable differentiable reasoning. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 25134–25145. Curran Associates, Inc.

[40] Khosravi, P., Choi, Y., Liang, Y., Vergari, A., and Van den Broeck, G. (2019). On tractable computation of expected predictions. In *Advances in Neural Information Processing Systems*, pages 11169–11180.

[41] Khosravi, P., Vergari, A., Choi, Y., Liang, Y., and den Broeck, G. V. (2020). Handling missing data in decision trees: A probabilistic approach. In *Proceedings of The Art of Learning with Missing Values, Workshop at ICML*.

[42] Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

[43] LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. (2006). A tutorial on energy-based learning. *Predicting structured data*, 1(0).

[44] Levatić, J., Kocev, D., and Džeroski, S. (2015). The importance of the label hierarchy in hierarchical multi-label classification. *Journal of Intelligent Information Systems*, 45(2):247–271.

[45] Liu, A., Mandt, S., and Van den Broeck, G. (2022). Lossless compression with probabilistic circuits. *International Conference of Learning Representations*.

[46] Liu, A. and Van den Broeck, G. (2021). Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34.

[47] Liu, F., Lin, G., and Shen, C. (2015). Crf learning with cnn features for image segmentation. *Pattern Recognition*, 48(10):2983–2992.

[48] Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). Deepproblog: Neural probabilistic logic programming. *Advances in Neural Information Processing Systems*, 31.

[49] Mattei, N. and Walsh, T. (2013). PrefLib: A library for preferences. In *International conference on algorithmic decision theory*, pages 259–270. Springer.

[50] Morettin, P., Kolb, S., Teso, S., and Passerini, A. (2020). Learning weighted model integration distributions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5224–5231.

[51] Mullenbach, J., Wiegreffe, S., Duke, J., Sun, J., and Eisenstein, J. (2018). Explainable prediction of medical codes from clinical text. *arXiv preprint arXiv:1802.05695*.

[52] Niepert, M., Minervini, P., and Franceschi, L. (2021). Implicit MLE: Backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34.

[53] Oztok, U. and Darwiche, A. (2015). A top-down compiler for sentential decision diagrams. In Yang, Q. and Wooldridge, M. J., editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 3141–3148. AAAI Press.

[54] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[55] Peharz, R., Gens, R., Pernkopf, F., and Domingos, P. (2016). On the latent variable interpretation in sum-product networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(10):2030–2044.

[56] Peharz, R., Lang, S., Vergari, A., Stelzner, K., Molina, A., Trapp, M., Broeck, G. V. d., Kersting, K., and Ghahramani, Z. (2020a). Einsum networks: Fast and scalable learning of tractable probabilistic circuits. In *International Conference of Machine Learning*.

[57] Peharz, R., Tschiatschek, S., Pernkopf, F., and Domingos, P. (2015). On theoretical properties of sum-product networks. In *Artificial Intelligence and Statistics*, pages 744–752. PMLR.

[58] Peharz, R., Vergari, A., Stelzner, K., Molina, A., Shao, X., Trapp, M., Kersting, K., and Ghahramani, Z. (2020b). Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Uncertainty in Artificial Intelligence*, pages 334–344. PMLR.

[59] Pipatsrisawat, K. and Darwiche, A. (2008). New compilation languages based on structured decomposability. In *AAAI*, volume 8, pages 517–522.

[60] Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. (2019). Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*.

[61] Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE.

[62] Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16.

[63] Radivojac, P., Clark, W. T., Oron, T. R., Schnoes, A. M., Wittkop, T., Sokolov, A., Graim, K., Funk, C., Verspoor, K., Ben-Hur, A., et al. (2013). A large-scale evaluation of computational protein function prediction. *Nature methods*, 10(3):221–227.

[64] Rahman, T., Kothalkar, P., and Gogate, V. (2014). Cutset networks: A simple, tractable, and scalable approach for improving the accuracy of chow-liu trees. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 630–645. Springer.

[65] Sacca, C., Teso, S., Diligenti, M., and Passerini, A. (2014). Improved multi-level protein–protein interaction prediction with semantic-based regularization. *BMC bioinformatics*, 15(1):1–18.

[66] Sarker, M. K., Zhou, L., Eberhart, A., and Hitzler, P. (2021). Neuro-symbolic artificial intelligence: Current trends. *arXiv preprint arXiv:2105.05330*.

[67] Shao, X., Molina, A., Vergari, A., Stelzner, K., Peharz, R., Liebig, T., and Kersting, K. (2020). Conditional sum-product networks: Imposing structure on deep probabilistic architectures. In *International Conference on Probabilistic Graphical Models*, pages 401–412. PMLR.

[68] Shao, X., Molina, A., Vergari, A., Stelzner, K., Peharz, R., Liebig, T., and Kersting, K. (2022). Conditional sum-product networks: Modular probabilistic circuits via gate functions. *International Journal of Approximate Reasoning*, 140:298–313.

[69] Shih, A. and Ermon, S. (2020). Probabilistic circuits for variational inference in discrete graphical models. *Advances in Neural Information Processing Systems*, 33:4635–4646.

[70] Sivaraman, A., Farnadi, G., Millstein, T., and Van den Broeck, G. (2020). Counterexample-guided learning of monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:11936–11948.

[71] Sorower, M. S. (2010). A literature survey on algorithms for multi-label learning. *Oregon State University, Corvallis*, 18:1–25.

[72] Toda, T. and Soh, T. (2016). Implementing efficient all solutions SAT solvers. *ACM J. Exp. Algorithmics*, 21(1):1.12:1–1.12:44.

[73] Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the twenty-first international conference on Machine learning*, page 104.

[74] Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13.

[75] Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. (2021). A compositional atlas of tractable circuit operations for probabilistic inference. *Advances in Neural Information Processing Systems*, 34.

[76] Vergari, A., Choi, Y., Peharz, R., and Van den Broeck, G. (2020). Probabilistic circuits: Representations, inference, learning and applications. In *Tutorial at the The 34th AAAI Conference on Artificial Intelligence*.

[77] Vergari, A., Di Mauro, N., and Esposito, F. (2015). 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.

[78] Vergari, A., Di Mauro, N., and Esposito, F. (2019). Visualizing and understanding sum-product networks. *Machine Learning*, 108(4):551–573.

[79] Vijayakumar, A. K., Cogswell, M., Selvaraju, R. R., Sun, Q., Lee, S., Crandall, D., and Batra, D. (2016). Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*.

[80] Xu, J., Zhang, Z., Friedman, T., Liang, Y., and Van den Broeck, G. (2018). A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR.

[81] Zhang, X., Bosselut, A., Yasunaga, M., Ren, H., Liang, P., Manning, C. D., and Leskovec, J. (2021). Greaselm: Graph reasoning enhanced language models. In *International Conference on Learning Representations*.

[82] Zhang, Y., Li, Z., and Zhang, M. (2020a). Efficient Second-Order TreeCRF for Neural Dependency Parsing. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J. R., 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.

[83] Zhang, Y., Zhou, H., and Li, Z. (2020b). Fast and accurate neural CRF constituency parsing. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 4046–4053.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Please see section 1 for a summary of our contributions
   (b) Did you describe the limitations of your work? [Yes] Please see Supplementary
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Please see Supplementary
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Please see supplementary material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please see section 5 for details regarding the dataset splits. All other details are included in the supplementary.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We reported error bars and significance testing for the hierarchical multi-label classification experiment, please see 5
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Please see supplementary material
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A 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. (2) 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. (2) can solved exactly in time $\mathcal{O}(|q||c|)$. It will then follow from it that computing Eq. (2) 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 $\mathcal{O}(|q||c|)$ [75] and ii) any smooth and decomposable circuit guarantees tractable marginalization in time linear in its size [13]. The next two propositions formalize these statements.

Proposition A.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. [75]. □

Note that $\mathcal{O}(|q||c|)$ is a loose upperbound and the size of r is in practice smaller [75].

Proposition A.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 $\mathbf{X}$, $r(\mathbf{Y},\boldsymbol{x})$ is a circuit only over $\mathbf{Y}$ and therefore Proposition 2.1 from Vergari et al. [75] can be directly applied. □

Analogously, the second statement of Theorem 3.1 follows from Proposition A.1 and by recalling that the MAP state of a deterministic circuit can be computed in time linear in its size.

Proposition A.3 (Tractable MAP state of circuits (Choi et al. [13]))**.** *Let $r(\mathbf{X},\mathbf{Y})$ be a circuit that is smooth and decomposable and deterministic over $\mathbf{Y}$ then for a configuration $\boldsymbol{x}$ its MAP state $\operatorname{argmax}_{\boldsymbol{y}} r(\boldsymbol{x},\boldsymbol{y})$ can be computed in time $\mathcal{O}(|r|)$.*

B Compiling logical formulas into 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] [1] a python SDD compiler [17, 10].

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

$$(Y_{\mathsf{cat}} \implies Y_{\mathsf{animal}}) \wedge (Y_{\mathsf{dog}} \implies Y_{\mathsf{animal}}). \tag{3}$$

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

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

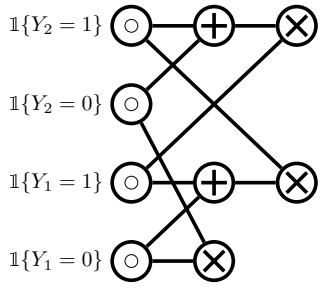
[2] https://github.com/wannesm/PySDD

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 $Y_{\mathsf{dog}} \vee \neg Y_{\mathsf{dog}}$ with Y_{cat}, Y_{dog} with $Y_{\mathsf{cat}} \vee \neg Y_{\mathsf{cat}}$, 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 $(Y_{\mathsf{dog}} \vee \neg Y_{\mathsf{dog}}) \wedge Y_{\mathsf{cat}}$, $Y_{\mathsf{dog}} \wedge (Y_{\mathsf{cat}} \vee \neg Y_{\mathsf{cat}})$, and $\neg Y_{\mathsf{dog}} \wedge \neg Y_{\mathsf{cat}}$.

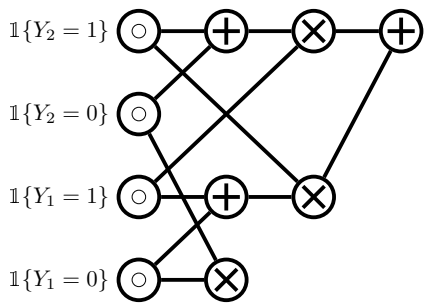
We again disjoin $(Y_{\mathsf{dog}} \vee \neg Y_{\mathsf{dog}}) \wedge Y_{\mathsf{cat}}$ with $Y_{\mathsf{dog}} \wedge (Y_{\mathsf{cat}} \vee \neg Y_{\mathsf{cat}})$, and $\neg Y_{\mathsf{dog}} \wedge \neg Y_{\mathsf{cat}}$ with true, the logical multiplicative identity, guaranteeing alternating sum and product nodes, as mentioned in Sec. 3.1.

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

$$((Y_{\mathsf{dog}} \vee \neg Y_{\mathsf{dog}}) \wedge Y_{\mathsf{cat}}) \vee (Y_{\mathsf{dog}} \wedge (Y_{\mathsf{cat}} \vee \neg Y_{\mathsf{cat}})) \tag{4}$$

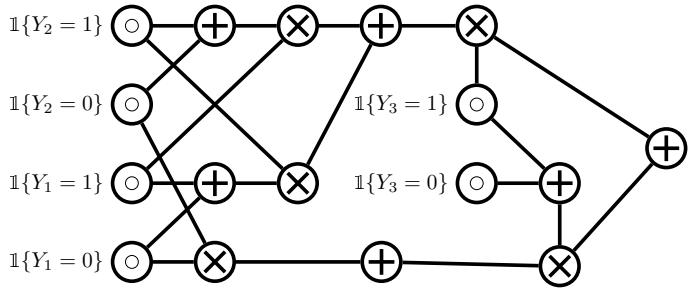
and the logical formula

$$\neg Y_{\mathsf{dog}} \wedge \neg Y_{\mathsf{cat}} \tag{5}$$



What remains is to conjoin Eq. (4) with Y_{animal}, and Eq. (5) 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 [18].

C Overparameterizing the single-circuit SPL

As mentioned in Def. 3.8, 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*.

Replication works by maintaining m copies of the circuit, and taking their weighted average, i.e., introducing a sum unit that mixes them [58]. 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. [58]. Algorithm 1 formalizes this process.

As mentioned in Def. 3.8, 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 Additional experimental details

D.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 [49], and a 60/20/20 split.

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

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
Output: an overparameterized, smooth, and structured-decomposable circuit c over $\mathbf{X}$
if $\boldsymbol{q} \in$ cache **then**
  return cache $[\boldsymbol{q}]$
if c is an input unit **then**
  nodes $\leftarrow [c]$
else if c is a sum unit **then**
  elements $\leftarrow [\,]$
  //For every product unit that is an input of c
  //recursively overparameterize its inputs,
  //which are sum units, and take their cross (cartesian) product
  for $(c_L, c_R) \in \mathsf{in}(c)$ **do**
    left $\leftarrow$ OVERPARAMETERIZE(c_L, k)
    right $\leftarrow$ OVERPARAMETERIZE(c_R, k)
    elements.APPEND([CROSSPRODUCT(left, right)]
  $\mathsf{in}(c) \leftarrow$ elements
  nodes $= [c] + [\text{COPY}(c)$ **for** $i = 1$ **to** $k]$
if first_call **then**
  //Create a sum unit whose inputs are nodes
  //and whose parameters are 1s.
  nodes $\leftarrow$ SUM(nodes, $\{1\}_{i=1}^{|\text{nodes}|}$)
$\mathsf{cache}(c) \leftarrow$ nodes
return nodes

We follow Xu et al. [80] 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 [80] (MLP+$\mathcal{L}_{\mathsf{SL}}$) or its entropic extension [3] (MLP+NESYENT).

We compile the logical constraints into an SDD [17] and then turn it into a constraint circuit c_{K} that is used for $\mathcal{L}_{\mathsf{SL}}$, NESYENT (Sec. 4) and our 1-circuit implementation of SPLs. To obtain the results for SPL in Table 2, we perform a grid search over the using the validation set for a maximum of 2000 iterations, similar to Xu et al. [80]. We search over the learning rates in the range $\{1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-4}, 5 \times 10^{-4}\}$, 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 set.

D.2 Hierarchical Multi-Label Classification

We follow the experimental setup of Giunchiglia and Lukasiewicz [32] 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 [32]. 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.

D.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 [60]. Our setting differs from the one proposed by Pogančić et al. [60] in two ways: i) a node only neighbors four nodes as instead of eight, excluding

Table 5: 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

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 [60] we use a ResNet18 [36] 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 [60].

D.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 Table 5. 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.

E Ethical Considerations

SPLs are meant as a module to be added on top of neural networks, and as such it does not significantly alter the ethical risk of the underlying model and target application. One exception is if the symbolic constraint is wrong (because e.g. it was encoded by a non-expert), in which case enforcing consistency - as SPLs do - may lead to mistakes or bias in the model's predictions.

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

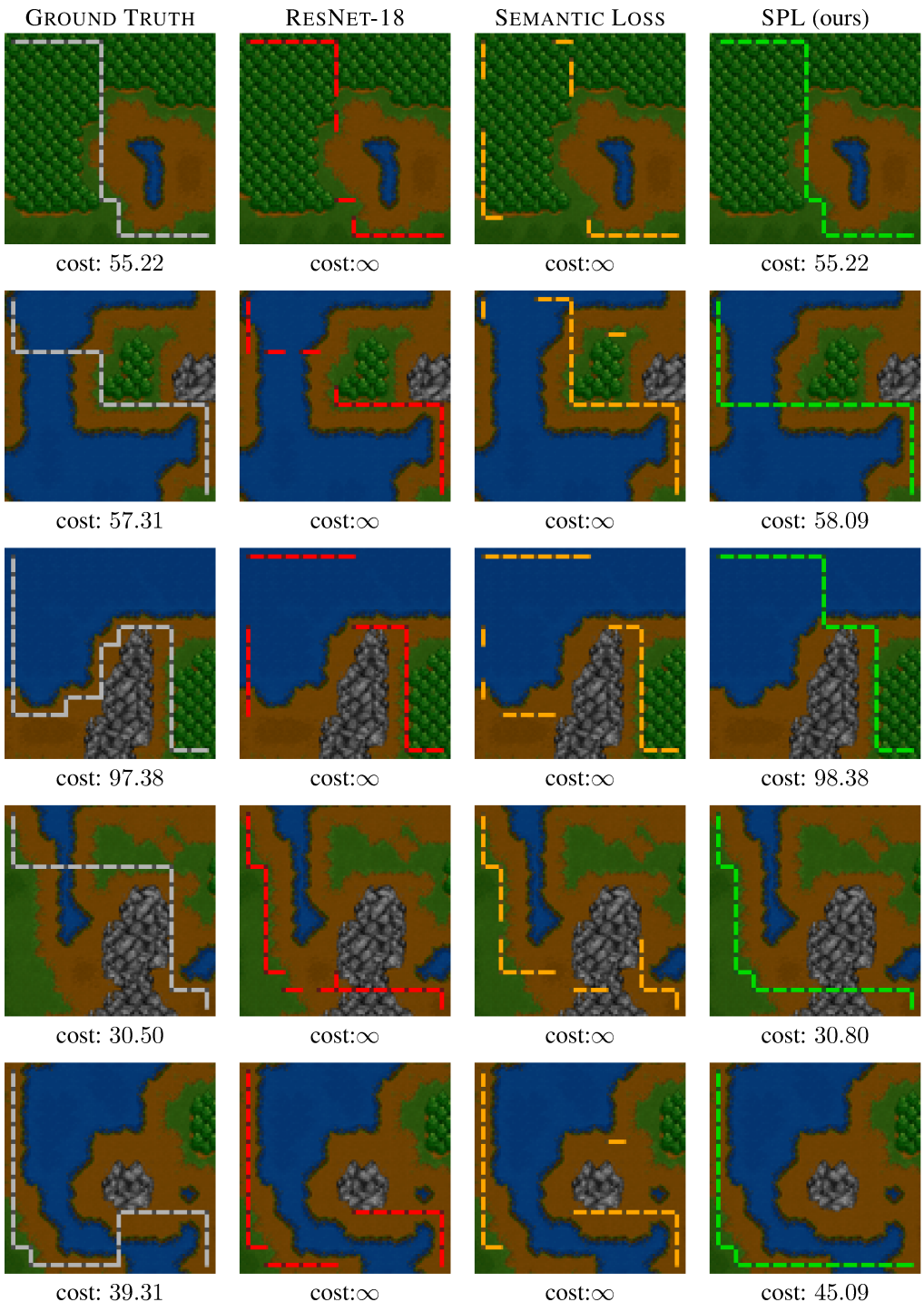


Figure 4: **More examples of shortest path predictions in SPLs and competitors.** SPLs always deliver valid paths and even when these do not exact match the ground truth, they are very close in terms of their global cost. Paths from the baselines might yield a higher Hamming score (as they have more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

F Timings

Table 6: A comparison of the timings of the different methods used throughout our experiments. All timings are in seconds. The timings for HMLC datasets are obtained by averaging over the timings of an entire epoch. All other timings are the average over three function calls. An empty cell, denoted by a dash, indicates the method was not used for that dataset, and therefore its timing is unavailable.

DATASET	COMPILATION	$\mathcal{L}_{\mathsf{SL}}$	NESYENT	SPLS		
				PARAMETERIZE	CROSS-ENTROPY	MAP
CELLCYCLE	68	-	-	0.03	0.41	0.74
DERISI	68	-	-	0.01	0.21	0.37
EISEN	29	-	-	0.01	0.16	0.28
EXPR	68	-	-	0.00	0.11	0.19
GASCH1	68	-	-	0.02	0.42	0.77
GASCH2	68	-	-	0.03	0.40	0.74
SEQ	66	-	-	0.01	0.22	0.36
SPO	67	-	-	0.03	0.40	0.74
DIATOMS	8	-	-	0.00	0.09	0.14
ENRON	0.04	-	-	0.01	0.16	0.28
IMCLEF07A	0.35	-	-	0.00	0.06	0.11
IMCLEF07D	0.08	-	-	0.00	0.05	0.10
WARCRAFT	457	16.30	-	0.21	14.11	15.59
PREFERENCE	[80]	0.024	0.035	0.00	0.00	0.01
SIMPLE PATH	[80]	0.34	0.49	0.00	0.13	0.19