Semantic Strengthening of Neuro-Symbolic Learning

Kareem Ahmed
Computer Science Department
UCLA
ahmedk@cs.ucla.edu

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

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

Abstract

Numerous neuro-symbolic approaches have recently been proposed typically with the goal of adding symbolic knowledge to the output layer of a neural network. Ideally, such losses maximize the probability that the neural network's predictions satisfy the underlying domain. Unfortunately, this type of probabilistic inference is often computationally infeasible. Neuro-symbolic approaches therefore commonly resort to fuzzy approximations of this probabilistic objective, sacrificing sound probabilistic semantics, or to sampling which is very seldom feasible. We approach the problem by first assuming the constraint decomposes conditioned on the features learned by the network. We iteratively *strengthen* our approximation, restoring the dependence between the constraints most responsible for degrading the quality of the approximation. This corresponds to computing the mutual information between pairs of constraints conditioned on the network's learned features, and may be construed as a measure of how well aligned the gradients of two distributions are. We show how to compute this efficiently for tractable circuits. We test our approach on three tasks: predicting a minimum-cost path in Warcraft, predicting a minimum-cost perfect matching, and solving Sudoku puzzles, observing that it improves upon the baselines while sidestepping intractability.

1 Introduction

Neural networks have been established as excellent feature extractors, managing to learn intricate statistical features from large datasets. However, without a notion of the *symbolic rules* underlying any given problem domain, neural networks are often only able to achieve decent *label-level* accuracy, with a complete disregard to the structure *jointly* encoded by the individual labels. These structures may encode, for example, a path in a graph, a matching of users to their preferences, or even the solution to a Sudoku puzzle.

Neuro-symbolic approaches (De Raedt et al., 2020) hope to remedy the problem by injecting into the training process knowledge regarding the underlying problem domain, e.g. a Sudoku puzzle is characterized by the uniqueness of the elements of every row, column, and 3×3 square. This is achieved by maximizing the probability allocated by the neural network to outputs satisfying the rules of the underlying domain. Computing this quantity is, in general, a #P-hard problem (Valiant, 1979), which while tractable for a range of practical problems (Xu et al., 2018; Ahmed et al., 2022c), precludes many problems of interest.

A common approach is to side step the hardness of computing the probability *exactly* by replacing logical operators with their fuzzy t-norms, and logical implications with simple inequalities (Medina Grespan et al., 2021; van Krieken et al., 2020). This, however, does not preserve the sound probabilistic semantics of the underlying logical statement: equivalent logic statements no longer correspond to the same set of satisfying assignments, to different probability distributions, and consequently, vastly different constraint probabilities. On the other hand, obtaining a Monte Carlo estimate of the probability (Ahmed et al., 2022a) is infeasible in exponentially-sized output spaces where the valid outputs represent only a sliver of the distribution's support.

In this paper, starting from first principles, we derive a probabilistic approach to scaling probabilistic inference for neuro-symbolic learning while retaining the sound semantics of the underlying logic. Namely, we start by assuming that the probability of the constraint decomposes, conditioned on the network's learned features. That is, we assume the events encoded by the logical formula to be *mutually independent* given the learned features, and therefore, joint probability factorizes as a product of probabilities. This generalizes the prolific assumption that the prob-

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206. Copyright 2023 by the author(s).

abilities of the variables are *mutually-independent* conditioned on the network's learned features (Mullenbach et al., 2018; Xu et al., 2018; Giunchiglia and Lukasiewicz, 2020) to events over arbitrary number of atoms. This reduces the (often intractable) problem of probabilistically satisfying the constraint, the validity of a Sudoku puzzle, to the (tractable) problem of probabilistically satisfying the individual local constraints, e.g. the uniqueness of the elements of a row, column, or square. This, however, introduces inconsistencies: an assignment that satisfies one constraint might violate another, leading to misaligned gradients. More precisely, for each pair of constraints, we are interested in the penalty incurred, in terms of modeling error, by assuming the constraints to be independent when they are in fact dependent, conditioned on the features learned by the neural network. This corresponds exactly to the conditional mutual information, a quantity notoriously hard to calculate. We give an algorithm for tractably computing the conditional mutual information, given that our constraints are represented as circuits satisfying certain structural properties. Training then proceeds, where we interleave the process of learning the neural network, with the process of *semantic strengthening*, where we iteratively tightening our approximation, using the neural network to guide us to which constraints need to be made dependent.

We test our approach on three different tasks: predicting a minimum-cost path in a Warcraft terrain, predicting a minimum-cost perfect matching, as well as solving Sudoku puzzles, where we observe that our approach greatly improves upon the baselines all for a minuscule increase in computation time (our experiments are capped at 2-3, and 7 seconds per iteration for Warcraft min-cost path, MNIST perfect matching, and Sudoku, respectively), thereby sidestepping the intractability of the problem. Our code is publiclt available at github.com/UCLA-StarAI/Semantic-Strengthening.

2 Problem Statement and Motivation

We will start by introducing the notational choices used throughout the remainder of the paper, followed by a motivation of the problem.

We write uppercase letters (X, Y) for Boolean variables and lowercase letters (x, y) for their instantiation ($Y = 0$ or $Y = 1$). Sets of variables are written in bold uppercase ($\mathbf{X}$, $\mathbf{Y}$), and their joint instantiation in bold lowercase ($\boldsymbol{x}$, $\boldsymbol{y}$). A literal is a variable (Y) or its negation ($\neg Y$). A logical sentence (α or β) is constructed from variables and logical connectives ($\wedge$, $\vee$, etc.), and is also called a (logical) formula or constraint. A state or world $\boldsymbol{y}$ is an instantiation to all variables $\mathbf{Y}$. A state $\boldsymbol{y}$ satisfies a sentence α, denoted $\boldsymbol{y} \models \alpha$, if the sentence evaluates to true in that world. A state $\boldsymbol{y}$ that satisfies a sentence α is also said to be a model of α. We denote by $m(\alpha)$ the set of all models of α. The notation for states $\boldsymbol{y}$ is used to refer to an assignment, the logical sentence enforcing the assignment, or the binary output vector capturing the assignment, as these are all equivalent notions. A sentence α entails another sentence β, denoted $\alpha \models \beta$, if all worlds that satisfy α also satisfy β.

A Probability Distribution over Possible Structures Let α be a logical sentence defined over Boolean variables $\mathbf{Y} = \{Y_1, \ldots, Y_n\}$. Let $\boldsymbol{p}$ be a vector of probabilities for the same variables $\mathbf{Y}$, where $\boldsymbol{p}_i$ denotes the predicted probability of variable Y_i and corresponds to a single output of the neural network. The neural network's outputs induce a probability distribution $P(\cdot)$ over possible states $\boldsymbol{y}$ of $\mathbf{Y}$:

$$P(\boldsymbol{y}) = \prod_{i:\boldsymbol{y}\models Y_i} \boldsymbol{p}_i \prod_{i:\boldsymbol{y}\models \neg Y_i} (1 - \boldsymbol{p}_i). \tag{1}$$

Semantic Loss The semantic loss (Xu et al., 2018) is a function of the logical constraint α and a probability vector $\boldsymbol{p}$. It quantifies how close the neural network comes to satisfying the constraint by computing the probability of the constraint under the distribution $P(\cdot)$ induced by $\boldsymbol{p}$. It does so by reducing the problem of probability computation to weighted model counting (WMC): summing up the models of α, each weighted by its likelihood under $P(\cdot)$. It, therefore, maximizes the probability mass allocated by the network to the models of α

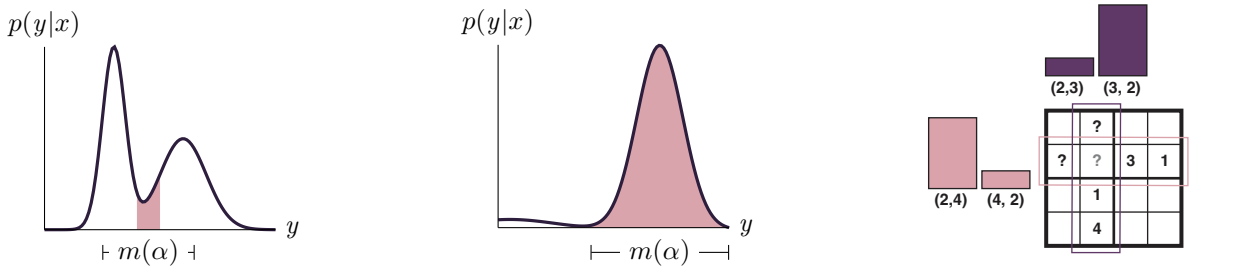
$$P(\alpha) = \mathbb{E}_{\boldsymbol{y}\sim P}\left[\mathbf{1}\{\boldsymbol{y} \models \alpha\}\right] = \sum_{\boldsymbol{y}\models\alpha} P(\boldsymbol{y}). \tag{2}$$

Taking the negative logarithm recovers semantic loss.

Computing the above expectation is generally #P-hard (Valiant, 1979): there are potentially exponentially many models of α. For instance, there are 6.67×10^{21} valid 9×9 Sudokus (Felgenhauer and Jarvis, 2005), where as the number of valid matchings or paths in a $n \times n$ grid grows doubly-exponentially in the grid size (Strehl, 2001).

A common approach resorts to *relaxing* the logical statements, replacing logical operators with their fuzzy t-norms, and implications with simple inequalities, and come in different flavors: Product (Rocktäschel et al., 2015; Li and Srikumar, 2019; Asai and Hajishirzi, 2020), Gödel (Minervini et al., 2017), and Łukasiewicz (Bach et al., 2017), which differ only in their interpretation of the logical operators. Medina Grespan et al. (2021) offer a comprehensive theoretical, and empirical, treatment of the subject matter.

While attractive due to their tractability, t-norms suffer from a few major drawbacks. First, they *lose the precise meaning of the logical statement*, i.e. the satisfying and unsatisfying assignments of the relaxed logical formula differ from those of the original logical formula. Second, the logic is no longer consistent, i.e. logical statements that



(a) Setting where satisfying assignments are only fraction of distribution support.

(b) A network allocating *most* of probability mass to satisfying assignments.

(c) Distributions over empty entries of Sudoku row and col modeled separately.

Figure 1: Estimating the probability of a constraint using sampling can fail when, (a) the set of satisfying assignments represents only a minuscule subset of the distribution's support, or, (b) when the network already largely satisfies the constraints, and consequently, we are very unlikely to sample very low-probability assignments violating the constraint. Using product t-norm, (c), to model the probability of satisfying constraints reduces the problem to satisfying the constraints locally, which can often lead to conflicting probabilities, and therefore, conflicting gradients. Here, e.g., according to the distribution over the Sudoku row, 3 is the likely value of the cell in grey, where as, according to the distribution over the Sudoku column, 4 is the likely value.

are otherwise equivalent correspond to different truth values, as the relaxations are a function of their syntax rather than their semantics. Lastly, the relaxation sacrifices sound probabilistic semantics, unlike other approaches (Xu et al., 2018; Manhaeve et al., 2018) where the output probability corresponds to the probability mass allocated to truth assignments of the logical statement, the output probability has no sound probabilistic interpretation (Medina Grespan et al., 2021).

A slightly more benign relaxation (Rocktäschel et al., 2015) only assumes that, for a constraint $\alpha = \beta_1 \wedge \ldots \wedge \beta_n$, a neural network $f(\cdot)$, and an input $\boldsymbol{x}$, the events β_i are mutually independent conditioned on the features learned by the neural network. That is, the probability of the constraint factorizes as $P(\alpha \mid f(\boldsymbol{x})) = P(\beta_1 \mid f(\boldsymbol{x})) \times \ldots \times P(\beta_n \mid f(\boldsymbol{x}))$. This recovers the true probabilistic semantics of the logical statement when $\beta_1, \ldots, \beta_n$ are over disjoint sets of variables, i.e. $\forall_{i,j}\ \mathsf{vars}(\beta_i) \cap \mathsf{vars}(\beta_j) = \emptyset$ for $i \neq j$ and can otherwise be thought of as a *tractable* approximation, the basis of which is the neural network's ability to sufficiently encode the dependencies shared between the constraints, rendering them conditionally independent given the learned features. That is assuming the neural network makes almost-deterministic predictions of the output variables given the embeddings. However, even assuming the true function being learned is deterministic, there is still the problem of an imperfect embedding giving probabilistic predictions whereby clauses are dependent.

The above relaxation reduces the *intractable* problem of satisfying the global constraint to the *tractable* problem of satisfying the local constraints, and can therefore often lead to *misaligned gradients*. Consider cell $(1, 1)$ of the Sudoku in Figure 1. Consider the two constraints asserting that the elements of row 2 and that the elements of column 2 are unique, and assume the probability distribution induced by the network over row and column assignments are as shown in Figure 1, right. This leads to opposing gradients for cell $(1, 1)$: On the one hand, the gradient from maximizing the probability of the column constraint pushes it to 2, whereas the gradient from maximizing the probability of the row constraint pushes it to 4. The problem here stems from modeling as independent two constraints that are strongly coupled, so much so that the value of one determines the value of the other.

Recently, Ahmed et al. (2022a) proposed using sampling to obtain a Monte Carlo estimate of the probability of the constraint being satisfied. This offers the convenience of specifying constraints as PyTorch functions, as well as accommodating non-differentiable elements in the training pipeline of the constraint, especially in cases where the training pipeline includes non-differentiable elements. However, when problems are intractable, this is often accompanied by a state space that is combinatorial in size, meaning that the probability of sampling a valid structure drops precipitously as a function of the size of the state space, making it near impossible to obtain any learning signal, as almost all the sampled states will necessarily violate our constraint. The same applies when the constraint is almost satisfied, meaning we never sample low-probability assignment that violate the constraint.

That is not to mention the downfalls of gradient estimators: the gradient estimator employed by Ahmed et al. (2022a) is the REINFORCE gradient estimator, which while unbiased in the limited of many samples, exhibits variances that makes it very hard to learn. Even gradient estimators that do not exhibit this problem of variance, trade off variance for bias, making it unlikely to obtain the true gradient.

3 Semantic Strengthening

We are interested in an approach that, much like the approaches discussed in Section 2 is tractable, but retains sound probabilistic semantics, and yields a non-zero gradient when the constraint is locally, or globally, violated.

Let our constraint α be given by a conjunctive normal form (CNF), $\alpha = \beta_1 \wedge \ldots \wedge \beta_n$. We start by assuming that, for a neural network $f(\cdot)$, and an input $\boldsymbol{x}$, the clauses β_i are mutually independent conditioned on the features learned by the neural network i.e. the probability of the constraint factorizes as $P(\alpha \,|\, f(\boldsymbol{x})) = P(\beta_1 \,|\, f(\boldsymbol{x})) \times \ldots \times P(\beta_n \,|\, f(\boldsymbol{x}))$, where the probability of each of the clauses, $P(\beta_i)$, can be computed tractably. This recovers the true probabilistic semantics of the logical statement when $\beta_1, \ldots, \beta_n$ are over disjoint sets of variables, i.e. $\forall_{i,j}\ \mathsf{vars}(\beta_i) \cap \mathsf{vars}(\beta_j) = \emptyset$ for $i \neq j$, and can otherwise be thought of as a *tractable* approximation, the basis of which is the neural network's ability to sufficiently encode the dependencies shared between the constraints, rendering them conditionally independent given the learned features, again, assuming the true function is deterministic, with no inherent uncertainty.

The above approximation is semantically sound in the sense that, the probability of each term $P(\beta_i)$ accounts for all the truth assignment of the clause β_i. It is also guaranteed to yield a semantic loss value of 0, and therefore a zero gradient if and only if all the clauses, β_i, are satisfied.

However, as discussed in Section 2, training the neural network to satisfy the local constraints can often be problematic: two *dependent* constraints assumed independent can often disagree on the value of their shared variables leading to opposing gradients. If we are afforded more computational resources, we can start strengthening our approximation by relaxing some of the independence assumptions made in our model.

3.1 Deriving the Criterion

The question then becomes, *which independence assumptions to relax*. We are, of course, interested in relaxing the independence assumptions that have the most positive impact on the quality of the approximation. Or, put differently, we are interested in relaxing the independence assumptions for which we incur the most penalty for assuming, otherwise dependent constraints, to be independent. For each pair of constraints β_i and β_j, for all $i \neq j$, this corresponds to the Kullback-Leibler divergence of the product of their marginals from their joint distribution, and is a measure of the modeling error we incur, in bits, by assuming the independence of the two constraints

$$D_{\mathrm{KL}}\big(P_{(X,Y)} \| P_X \cdot P_Y\big) \tag{3}$$

where X and Y are Bernoulli random variables, $X \sim P(\beta_i)$, $Y \sim P(\beta_j)$, and $(X, Y) \sim P(\beta_i, \beta_j)$, for all i, j such that $i \neq j$. Equation (3) equivalently corresponds to the *mutual information* $I(X;Y)$ given by

$$I(X;Y) = \mathbb{E}_{(X,Y)}\left[\log \frac{P_{(X,Y)}(X,Y)}{P_X(X) \cdot P_Y(Y)}\right], \tag{4}$$

between the random variables X and Y, or the measure of *dependence* between them. Intuitively, mutual information captures the information shared between X and Y: it measures how much knowing one reduces about the uncertainty of the other. When they are independent, then knowing one does not give any information about the other, and therefore the mutual information is 0. At the other extreme, one is a deterministic function of the other, and therefore, the mutual information is maximized and equals to their entropy. Note that the expectations in both Equation (3) and Equation (4) are over the joint distribution $P_{(X,Y)}$.

We would be remiss, however, to dismiss the features learned by the network, as they already encode some of the dependencies between the constraints, affording us the ability to make stronger approximations. That is, we are interested in the mutual information between all pairs of constraints β_i, β_j *conditioned* on the neural network's features. Let $\mathcal{D}$ be our data distribution, and Z be a random variable distributed according to D, we are interested in computing

$$I(X;Y \mid Z) = \mathbb{E}_Z\left[\mathbb{E}_{(X,Y)|Z}\left[\log \frac{P(x, y \mid \boldsymbol{z})}{P(x \mid \boldsymbol{z}) \cdot P(y \mid \boldsymbol{z})}\right]\right] \tag{5}$$

$$= \mathbb{E}_Z\left[\sum_{x=0}^{1}\sum_{y=0}^{1} P(x, y \mid \boldsymbol{z})\left[\log \frac{P(x, y \mid \boldsymbol{z})}{P(x \mid \boldsymbol{z}) \cdot P(y \mid \boldsymbol{z})}\right]\right], \tag{6}$$

where, as is common place, we estimate the outer expectation using Monte Carlo sampling from the data distribution.

Perhaps rather surprisingly, not withstanding the expectation w.r.t the data distribution, the quantity in Equation (5) is hard to compute. This is not only due to the intractability of the probability, which as we have already stated is #P-hard in general, but also due to the hardness of conjunction, in general. Loosely speaking, one could have constraints β_i and β_j for which the probability computation, $P(\beta_i)$ and $P(\beta_j)$ is tractable, yet computing $P(\alpha)$, where once again $\alpha = \beta_i \wedge \beta_j$, is hard (Shen et al., 2016; Khosravi et al., 2019). Intuitively, the hardness of conjunction comes from finding the intersection of the satisfying assignments without enumeration. We formalize this in Section 3.3.

3.2 The Semantic Strengthening Algorithm

For the purposes of this section, we will assume we can tractably compute the conditional mutual information in Equation (5), and proceed with giving our Semantic

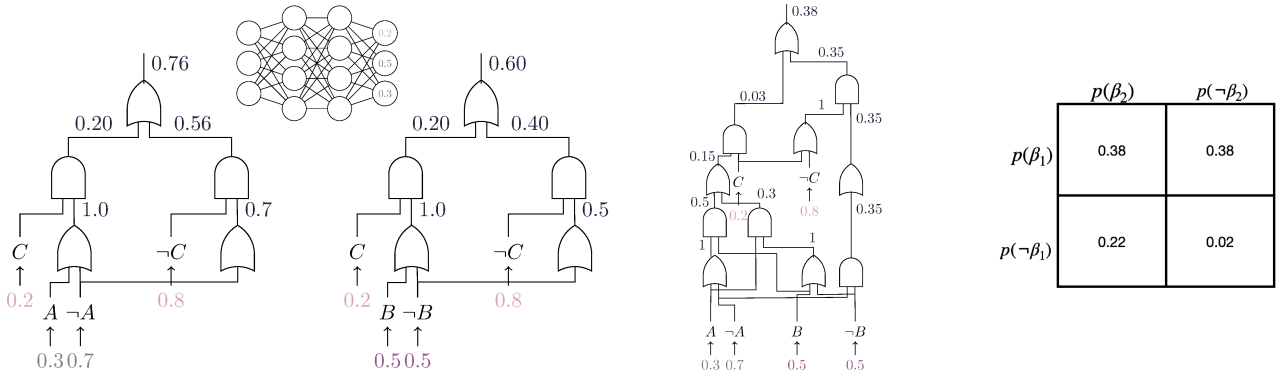


Figure 2: (Left) Example of two compatible constraint circuits parameterized by the outputs of a neural network. To compute the probability of a circuit, we plug in the output of the neural network $\boldsymbol{p}_i$ and $1 - \boldsymbol{p}_i$ for positive and negative literal i, respectively. The computation proceeds bottom-up, taking products at AND gates and summations at OR gates, and the probability is accumulated at the root of the circuit. (Right) the conjunction of the two constraint circuits, its probability, computing the probabilities required for the mutual information using the law of total probability.

Algorithm 1 $\texttt{MI}(\beta_1; \beta_2 \mid f(\boldsymbol{x}))$

Input: Two compatible constraint circuits β_1 and β_2
Output: Mutual Information of β_1 and β_2 given features

// Conjoin β_1 and β_2
$\alpha = \beta_1 \wedge \beta_2$
// Compute the probability of α, β_1 and β_2 c.f. Figure 2
$p_\alpha, p_{\beta_1}, p_{\beta_2}$ = prob(α), prob(β_1), prob(β_2)
// Calculate marginals and joint using total probability
$p_X = [1 - p_{\beta_1}, p_{\beta_1}], p_Y = [1 - p_{\beta_2}, p_{\beta_2}]$
$p_{(X,Y)} = [[1 - p_{\beta_1} - p_{\beta_2} - p_\alpha, p_{\beta_2} - p_\alpha], [p_{\beta_1} - p_\alpha, p_\alpha]]$
mi $= 0$
for x, y **in** product($[0, 1]$) **do**
    mi $+= p_{(X,Y)}[x][y] \times \log(\frac{p_{(X,Y)}[x][y]}{p_X[x] \times p_Y[y]})$
return mi

Algorithm 2 $\texttt{SemanticStrengthening}(\text{constraints}, \kappa)$

Input: Current set of constraint circuits
Output: *Strengthened* set of constraints

pwmi $= [\,]$
for β_1, β_2 **in** product(constraints) **do**
    if disjoint(vars(β_i), vars(β_j)) **then continue**
// Keep track of constraints with mutual information
    pwmi.append((MI(β_i, β_j), β_i, β_j))
// Consider only the top κ pairs of constraints
to_merge $=$ sorted(pwmi, reverse=True)[: κ]
for mi, β_1, β_2 **in** (to_merge) **do**
    constraints.remove(β_i, β_j)
    constraints.append($\beta_i \wedge \beta_j$)
return constraints

Strengthening algorithm. The idea is, simply put, to use the neural network to guide the process of relaxing the independence assumptions introduced between the constraints. Specifically, we are given an interval, η, a constraint budget, κ, and a computational budget τ. We initiate the process of training the neural network, interrupting training every η epochs, computing the conditional mutual information between pairs of constraints, considering only those pairs sharing at least one variable (e.g. the two constraints asserting the uniqueness of the first and last row, respectively, do not share variables, are therefore independent, and by definition have a mutual information of 0, so we need not consider joining them, *yet*). Subsequently, we identify the κ pairs of constraints with the highest pairwise conditional mutual information, and that therefore, have the most detrimental effect on the quality of our approximation. We detect the strongly connected components of constraints, and conjoin them: if β_1 and β_2 should be made dependent, and β_2 and β_3 should be made dependent, then β_1 , β_2 and β_3 are made dependent. We delete the old constraints from, and add the new constraints, to our set of constraints, and resume training. This process is repeated every η epochs until we have exhausted our computational budget τ. Our full algorithm is shown in Algorithm 2.

3.3 Tractably Computing the Criterion

Unlike previous approaches (Chen et al., 2018; Mesner and Shalizi, 2019; Tezuka and Namekawa, 2021), we do not need to resort to variational approximations or neural estimation to compute the mutual information, and instead appeal to the language of *tractable circuits*. That is, we appeal to knowledge compilation techniques—a class of methods that transform, or *compile*, a logical theory into a target form, *tractable circuits*, which represent functions as parameterized computational graphs. By imposing certain structural properties on these computational graphs, we enable the tractable computation of certain classes of probabilistic queries over the encoded functions. As such, circuits provide us with a language for building and reasoning

about tractable representations.

Logical Circuits More formally, a *logical circuit* is a directed, acyclic computational graph representing a logical formula. Each node n in the DAG encodes a logical subformula, denoted $[n]$. Each inner node in the graph is either an AND or an OR gate, and each leaf node encodes a Boolean literal (Y or $\neg Y$). We denote by $\mathsf{in}(n)$ the set of n's children, that is, the operands of its logical gate.

Structural Properties As already alluded to, circuits enable the tractable computation of certain classes of queries over encoded functions granted that a set of structural properties are enforced. We explicate such properties below.

A circuit is *decomposable* if the inputs of every AND gate depend on disjoint sets of variables i.e. for $\alpha = \beta \wedge \gamma$, $\mathsf{vars}(\beta) \cap \mathsf{vars}(\gamma) = \varnothing$. Intuitively, decomposable AND nodes encode local factorizations over variables of the function. For simplicity, we assume that decomposable AND gates always have two inputs, a condition that can be enforced on any circuit in exchange for a polynomial increase in its size (Vergari et al., 2015; Peharz et al., 2020).

A second useful property is *smoothness*. A circuit is *smooth* if the children of every OR gate depend on the same set of variables i.e. for $\alpha = \bigvee_i \beta_i$, we have that $\mathsf{vars}(\beta_i) = \mathsf{vars}(\beta_j)\ \forall i, j$. Decomposability and smoothness are a sufficient and necessary condition for tractable integration over arbitrary sets of variables in a single pass, as they allow larger integrals to decompose into smaller ones (Choi et al., 2020).

Furthermore, a circuit is said to be *deterministic* if, for any input, at most one child of every OR node has a non-zero output i.e. for $\alpha = \bigvee_i \beta_i$, we have that $\beta_i \wedge \beta_j = \bot$ for all $i \neq j$. Similar to decomposability, determinism induces a recursive partitioning of the function, but over the support, i.e. satisfying assignments, of the function, rather than the variables. Determinism, taken together with smoothness and decomposability, allows us to tractably compute the probability of a constraint (Darwiche and Marquis, 2002).

What remains, is to show that we can tractably conjoin two constraints. Conjoining two decomposable and deterministic circuits is NP-hard if we wish the result to also be decomposable and deterministic, which as we mentioned is a requirement for tractable probability computation (Darwiche and Marquis, 2002; Shen et al., 2016; Khosravi et al., 2019). To guarantee the tractability of the probability computation of the conjoined constraint, we will, therefore, need to introduce one last structural property, namely the notion of *compatibility* between two circuits (Vergari et al., 2021). Two circuits, c_1 and c_2 over variables $\mathbf{Y}$ are said to be compatible if (1) they are smooth and decomposable, and (2) any pair of AND nodes, $n \in c_1$ and $m \in c_2$ with the same scope over $\mathbf{Y}$ can be rearranged to be mutually compatible and decompose in the same way i.e. $\mathsf{vars}(n) = \mathsf{vars}(m) \implies \mathsf{vars}(n_i) = \mathsf{vars}(m_i)$, and n_i and m_i are compatible, for some arrangement of the inputs n_i and m_i of n and m. A sufficient condition for compatibility is that both c_1 and c_2 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). Intuitively, the two circuits should share the order in which they factorize the function over its variables. Figure 2 shows an example of smooth, decomposable, deterministic and compatible circuits.

At a high level, there exist off-the-shelf compilers utilizing SAT solvers, essentially through case analysis, to compile a logical formula into a tractable logical circuit. We are agnostic to the exact flavor of circuit so long as the properties outlined herein are respected. In our experiments, we use PySDD[1] – a Python SDD compiler (Darwiche, 2011; Choi and Darwiche, 2013).

Now that we have shown that we can tractably compute the probabilities $P(\beta_1), P(\beta_2)$ and $P(\alpha)$, we can utilize the law of total probability (c.f. Figure 2) to compute the remaining probabilities, and therefore, the mutual information. Our algorithm is shown in Algorithm 1.

4 Related Work

There has been increasing interest in combining neural learning with symbolic reasoning, a class of methods that has been termed *neuro-symbolic* methods, studying how to best combine both paradigms in a bid to accentuate their positives and mitigate their negatives. The focus of many such approaches has therefore been on making probabilistic reasoning tractable through first-order approximations, and differentiable, through reducing logical formulas into arithmetic objectives, replacing logical operators with their fuzzy t-norms, and implications with inequalities (Kimmig et al., 2012; Rocktäschel et al., 2015; Fischer et al., 2019; Pryor et al., 2022).

Diligenti et al. (2017) and Donadello et al. (2017) use first-order logic to specify constraints on outputs of a neural network. They employ fuzzy logic to reduce logical formulas into differential, arithmetic objectives denoting the extent to which neural network outputs violate the constraints, thereby supporting end-to-end learning under constraints. More recently, Xu et al. (2018) introduced semantic loss, which circumvents the shortcomings of fuzzy approaches, while supporting end-to-end learning under constraints. More precisely, *fuzzy reasoning* is replaced with *exact probabilistic reasoning*, by compiling logical formulae into structures supporting efficient probabilistic queries. Liu et al. (2023) use semantic loss to simultaneously learn a neural network and extract generalized logic rules. Differ-

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

ent from other neural-symbolic methods that require background knowledge and candidate logical rules, they aim to induce task semantics with minimal priors.

Another class of neuro-symbolic approaches have their roots in logic programming. DeepProbLog (Manhaeve et al., 2018) extends ProbLog, a probabilistic logic programming language, with the capacity to process neural predicates, whereby the network's outputs are construed as the probabilities of the corresponding predicates. This simple idea retains all essential components of ProbLog: the semantics, inference mechanism, and the implementation. Manhaeve et al. (2021) attempts to scale DeepProbLog by considering only the top-k proof paths. In a similar vein, Dai et al. (2018) combine domain knowledge specified as purely logical Prolog rules with the output of neural networks, dealing with the network's uncertainty through revising the hypothesis by iteratively replacing the output of the neural network with anonymous variables until a consistent hypothesis can be formed. Bošnjak et al. (2017) present a framework combining prior procedural knowledge, as a Forth program, with neural functions learned through data. The resulting neural programs are consistent with specified prior knowledge and optimized with respect to data.

There has recently been a plethora of approaches ensuring consistency by embedding the constraints as predictive layers, including semantic probabilistic layers (SPLs) (Ahmed et al., 2022b), MultiplexNet (Hoernle et al., 2022) and HMCCN (Giunchiglia and Lukasiewicz, 2020). Much like semantic loss (Xu et al., 2018), SPLs maintain sound probabilistic semantics, and while displaying impressive scalability to real world problems, but might struggle with encoding harder constraints. SIMPLE (Ahmed et al., 2023) proposes an SPL for the k-subset distribution, to be used as a latent space to induce a distribution over features, for which they derive a low-bias, low-variance gradient estimator. MultiplexNet is able to encode only constraints in disjunctive normal form, which is problematic for generality and efficiency as neuro-symbolic tasks often involve an intractably large number of clauses. HMCCN encodes label dependencies as fuzzy relaxation and is the current state-of-the-art model for hierarchical mutli-label classification (Giunchiglia and Lukasiewicz, 2020), but, similar to its recent extension (Giunchiglia and Lukasiewicz, 2021), is restricted to a certain family of constraints. Daniele et al. (2022) discusses how to enforce the consistency for fuzzy relaxations with general formulas.

5 Experimental Evaluation

We evaluated our approach, semantic strengthening, on several neuro-symbolic tasks, namely Warcraft minimum-cost path finding, minimum-cost perfect matching of MNIST digits, as well as the task of training neural networks to solve Sudoku puzzles. The challenge with all of the above tasks, when looked at through a neuro-symbolic lens, is the vastness of the state space: as previously mentioned, there are 6.6×10^{21} valid 9×9 Sudokus, and the number of valid matchings, or paths in a grid grows doubly-exponentially in the grid size—simply too much to enumerate. Even approaches like semantic loss which rely on circuit approaches to exploit the local structure in the problem, essentially through caching solutions to repeated subproblems, do not scale to large instances of these tasks.

As has been established in previous work (Xu et al., 2018; Ahmed et al., 2022c,b), label-level accuracy, or the accuracy of predicting individual labels is very often a poor indication of the performance of the neural network, and is often uninteresting in neuro-symbolic settings, where we are rather more interested in the accuracy of our predicted structure object *exactly* matching the ground truth, e.g., *is the prediction a shortest path?*, a metric which we denote "Exact" in our experiments, as well as the accuracy of predicting objects that are *consistent* with the constraint, e.g., *is the prediction a valid path?*, a metric which we denote "Consistent" in our experiments. Note that, unlike the other two tasks, for the case of Sudoku, these measures are one and the same: a valid Sudoku has a single *unique* solution.

In all of our experiments, we compare against two baselines: a neural network, whose architecture we specify in the corresponding experimental section, and the same neural network augmented with product t-norm, where we assume the independence of constraints throughout training.

Warcraft Shortest Path We evaluate our approach, semantic strengthening, on the challenging task of predicting the minimum-cost path in a weighted grid imposed over Warcraft terrain maps. Following Pogančić et al. (2020), our training set consists of $10,000$ terrain maps curated using the Warcraft II tileset. Each map encodes an underlying grid of dimension 12×12, where each vertex is assigned a cost depending on the type of terrain it represents (e.g. earth has lower cost than water). The shortest (minimum cost) path between the top left and bottom right vertices is encoded as an indicator matrix, and serves as label. Figure 3 shows an example input presented to the network and the input with an annotated shortest path as a groundtruth. Presented with an image of a terrain map, a convolutional neural network—similar to Pogančić et al. (2020), we use ResNet18 (He et al., 2016)—outputs a 12×12 binary matrix indicating a set of vertices. Note that the minimum-cost path is not unique: there may exist several paths sharing the same minimum cost, all of which are considered to be correct by our metrics. Table 1 shows our results.

We observe that incorporating constraints into learning improves the accuracy of predicting the optimal path from 44.80% to 50.40%, and the accuracy of predicting a *valid* path from 56.90% to 63.20%, as denoted by the "Ex-

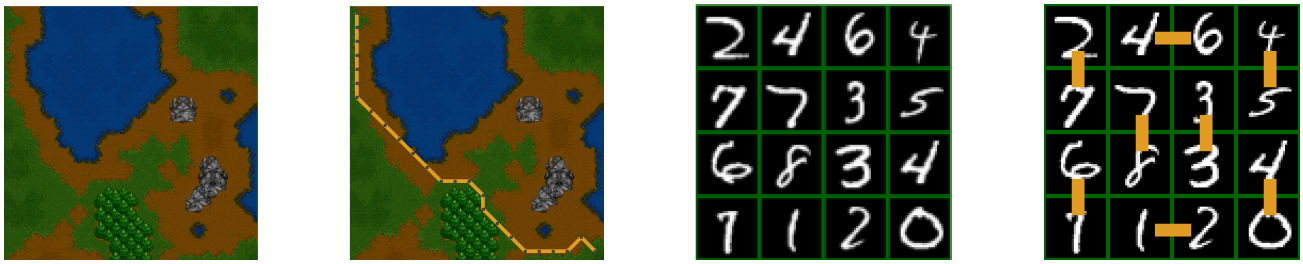


Figure 3: An example of a Warcraft terrain map (left) and an MNIST grid, and the corresponding groundtruth labels.

Table 1: Warcraft shortest path prediction results

Test accuracy %	Exact	Consistent
ResNet-18	44.80	56.90
+ Product t-norm	50.40	63.20
+ Semantic Strengthening	**61.20**	**72.70**

act" and "Consistent" metrics, respectively. Furthermore, and perhaps more interestingly, we see that our approach, *semantic strengthening*, greatly improves upon the baseline, as well as product t-norm improving the accuracy of predicting the optimal path from 44.80% and 50.40% to 61.20%, while greatly improving the accuracy of predicting a valid path from 56.90% and 63.20% to 72.70%.

MNIST Perfect Matching Our next task consists in predicting a minimum-cost perfect-matching of a set of k^2 MNIST digits arranged in a $k \times k$ grid, where diagonal matchings are not permitted. We consider the problem for the instance when $k = 10$. Similar to Pogančić et al. (2020), we generate the ground truth by considering the underlying $k \times k$ grid graph, and solving a min-cost perfect-matching problem using Blossom V (Kolmogorov, 2009), where the edge weights are given simply by reading the two vertex digits as a two-digit number, reading downwards for vertical edges, and left to right for horizontal edges. The minimum-cost perfect matching label is then encoded as an indicator vector for the subset of the selected edges. Similar to the Warcraft experiment, the grid image is input to a (pretrained) ResNet-18, which simply outputs a set of predicted edges. Table 2 shows our results.

Table 2: Perfect Matching prediction test results

Test accuracy %	Exact	Consistent
ResNet-18	9.30	10.00
+ Product t-norm	12.70	12.90
+ Semantic Strengthening	**15.50**	**18.40**

Similar to the Warcraft experiment, we observe that incorporating constraints into learning improves the accuracy of predicting the optimal perfect matching from 9.30% to 12.70%, and the accuracy of predicting a *valid* perfect matching from 10.00% to 12.90%, as denoted by the "Exact" and "Consistent" metrics, respectively. Furthermore, we see that our approach, *semantic strengthening*, greatly improves upon the baseline, as well as product t-norm improving the accuracy of predicting the optimal perfect matching from 09.30% and 12.70% to 15.50%, while greatly improving the accuracy of predicting a valid perfect matching from 10.00% and 12.90% to 18.40%.

Sudoku Lastly, we consider the task of predicting a solution to a given Sudoku puzzle. Here the task is, given a 9×9 partially-filled grid of numbers to fill in the remaining cells in the grid such that the entries each row, column, and 3×3 square are unique i.e. each of the numbers from 1 through 9 appears exactly once.

We use the dataset provided by Wang et al. (2019), consisting of 10K Sudoku puzzles, split into 9K training examples, and 1K test samples, all puzzles having 10 missing entries.

As our baseline, we follow Wang et al. (2019) in using a convolutional neural network modeled on that of Park (2018). The input to the neural network is given as a bit representation of the initial Sudoku board, along with a mask representing the bits to be learned, i.e. the bits in the empty Sudoku cells. The network interprets the bit inputs as 9 input image channels (one for each square in the board) and uses a sequence of 10 convolutional layers (each with 512 3×3 filters) to output the solution, with the mask input as a set of additional image channels in the same format as the board. Table 3 shows our results.

Table 3: Sudoku test results

Test accuracy %	Exact	Consistent
10-Layer ConvNet	16.80	16.80
+ Product t-norm	22.10	22.10
+ Semantic Strengthening	**28.00**	**28.00**

In line with our previous experiments, we observe that incorporating constraints into learning improves the accuracy of predicting correct Sudoku solutions, the "Exact" metric from 16.80% to 22.10%. Furthermore, we see that our ap-

proach, *semantic strengthening*, greatly improves upon the baseline, as well as product t-norm, improving the accuracy from 16.80% and 22.10% to 28.00%.

6 Conclusion

In conclusion, we proposed semantic strengthening, a tractable approach to neuro-symbolic learning, that remains faithful to the probabilistic semantic of the distribution defined by the neural network on a given constraint. Semantic strengthening starts by assuming the independence of the clauses in a given constraint, thereby reducing the, often intractable, problem of satisfying a global constraint, to the tractable problem of satisfying individual local constraints. It uses a principled criterion, conditional mutual information, to determine, and relax any unjustified independence assumptions most detrimental to the quality of our approximation. We have shown that we are able to greatly improve upon the baselines on three challenging tasks, where semantic strengthening was able to increase the *accuracy* and *consistency* of the model's predictions.

Acknowledgements

KA would like to thank Arthur Choi and Yoojung Choi for helpful discussions throughout the project. This work was funded in part by the DARPA Perceptually-enabled Task Guidance (PTG) Program under contract number HR00112220005, and NSF grants #IIS-1943641, #IIS-1956441, and #CCF-1837129.

References

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)*.

Ahmed, K., Teso, S., Chang, K.-W., Van den Broeck, G., and Vergari, A. (2022b). Semantic probabilistic layers for neuro-symbolic learning. In *NeurIPS*.

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

Ahmed, K., Zeng, Z., Niepert, M., and Van den Broeck, G. (2023). SIMPLE: A gradient estimator for k-subset sampling. In *ICLR*.

Asai, A. and Hajishirzi, H. (2020). Logic-Guided Data Augmentation and Regularization for Consistent Question Answering. In *ACL*.

Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. (2017). Hinge-loss markov random fields and probabilistic soft logic. *JMLR*.

Bošnjak, M., Rocktäschel, T., Naradowsky, J., and Riedel, S. (2017). Programming with a differentiable forth interpreter. In *Proceedings of the 34th ICML*.

Chen, J., Song, L., Wainwright, M., and Jordan, M. (2018). Learning to explain: An information-theoretic perspective on model interpretation. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 883–892. PMLR.

Choi, A. and Darwiche, A. (2013). Dynamic minimization of sentential decision diagrams. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, AAAI'13, page 187–194. AAAI Press.

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

Dai, W.-Z., Xu, Q.-L., Yu, Y., and Zhou, Z.-H. (2018). Tunneling neural perception and logic reasoning through abductive learning.

Daniele, A., van Krieken, E., Serafini, L., and Harmelen, F. V. (2022). Refining neural network predictions using background knowledge. *ArXiv*, abs/2206.04976.

Darwiche, A. (2011). Sdd: A new canonical representation of propositional knowledge bases. In *IJCAI*.

Darwiche, A. and Marquis, P. (2002). A knowledge compilation map. *JAIR*.

De Raedt, L., Dumančić, S., Manhaeve, R., and Marra, G. (2020). From statistical relational to neuro-symbolic artificial intelligence. In *IJCAI*.

Diligenti, M., Gori, M., and Saccà, C. (2017). Semantic-based regularization for learning and inference. *Artificial Intelligence*.

Donadello, I., Serafini, L., and d'Avila Garcez, A. (2017). Logic tensor networks for semantic image interpretation. In *IJCAI*.

Felgenhauer, B. and Jarvis, F. (2005). Enumerating possible sudoku grids.

Fischer, M., Balunovic, M., Drachsler-Cohen, D., Gehr, T., Zhang, C., and Vechev, M. (2019). DL2: Training and querying neural networks with logic. In *ICML*.

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

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *CVPR*.

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

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 32 (NeurIPS)*.

Kimmig, A., Bach, S., Broecheler, M., Huang, B., and Getoor, L. (2012). A short introduction to probabilistic soft logic. In *Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications*.

Kolmogorov, V. (2009). Blossom v: a new implementation of a minimum cost perfect matching algorithm. *Mathematical Programming Computation*, 1:43–67.

Li, T. and Srikumar, V. (2019). Augmenting neural networks with first-order logic. In *ACL*.

Liu, A., Xu, H., Van den Broeck, G., and Liang, Y. (2023). Out-of-distribution generalization by neural-symbolic joint training. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*.

Manhaeve, R., Dumancic, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). Deepproblog: Neural probabilistic logic programming. In *NeurIPS*.

Manhaeve, R., Marra, G., and De Raedt, L. (2021). Approximate Inference for Neural Probabilistic Logic Programming. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, pages 475–486.

Medina Grespan, M., Gupta, A., and Srikumar, V. (2021). Evaluating relaxations of logic for neural networks: A comprehensive study. In Zhou, Z.-H., editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2812–2818. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Mesner, O. C. and Shalizi, C. R. (2019). Conditional mutual information estimation for mixed discrete and continuous variables with nearest neighbors. *arXiv: Statistics Theory*.

Minervini, P., Demeester, T., Rocktäschel, T., and Riedel, S. (2017). Adversarial sets for regularising neural link predictors. In *UAI*.

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*.

Park, K. (2018). Can convolutional neural networks crack sudoku puzzles? https://github.com/Kyubyong/sudoku.

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

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

Pogančić, M. V., Paulus, A., Musil, V., Martius, G., and Rolinek, M. (2020). Differentiation of blackbox combinatorial solvers. In *ICLR*.

Pryor, C., Dickens, C., Augustine, E., Albalak, A., Wang, W. Y., and Getoor, L. (2022). Neupsl: Neural probabilistic soft logic.

Rocktäschel, T., Singh, S., and Riedel, S. (2015). Injecting logical background knowledge into embeddings for relation extraction. In *Proceedings of the 2015 Conference of the NAACL*.

Shen, Y., Choi, A., and Darwiche, A. (2016). Tractable operations for arithmetic circuits of probabilistic models. In Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Strehl, V. (2001). Counting domino tilings of rectangles via resultants. *Advances in Applied Mathematics*, 27(2):597–626.

Tezuka, T. and Namekawa, S. (2021). Information bottleneck analysis by a conditional mutual information bound. *Entropy*, 23.

Valiant, L. (1979). The complexity of computing the permanent. *Theoretical Computer Science*.

van Krieken, E., Acar, E., and Harmelen, F. V. (2020). Analyzing differentiable fuzzy logic operators. *ArXiv*, abs/2002.06100.

Vergari, A., Choi, Y., Liu, A., Teso, S., and Van den Broeck, G. (2021). A compositional atlas of tractable circuit operations for probabilistic inference. In *NeurIPS*.

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*.

Wang, P., Donti, P. L., Wilder, B., and Kolter, J. Z. (2019). Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR.

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 *Proceedings of the 35th ICML 2018*.