On Constraint Definability in Tractable Probabilistic Models

Anonymous authors

Abstract

Incorporating constraints is a major concern in probabilistic machine learning. A wide variety of problems require predictions to be integrated with reasoning about constraints, from modelling routes on maps to approving loan predictions. In the former, we may require the prediction model to respect the presence of physical paths between the nodes on the map, and in the latter, we may require that the prediction model respect fairness constraints that ensure that outcomes are not subject to bias. Broadly speaking, constraints may be probabilistic, logical or causal, but the overarching challenge is to determine if and how a model can be learnt that handles all the declared constraints. To the best of our knowledge, treating this in a general way is largely an open problem. In this paper, we consider a mathematical inquiry on how the learning of sum-product networks, is possible while incorporating constraints, establishing a connection between probabilistic constraints and the model’s parameters.

1. Introduction

Incorporating constraints is a major concern in data mining and probabilistic machine learning [Raedt et al., 2010, Friedman and Van den Broeck, 2019, Kisa et al., 2014]. A wide variety of problems require the prediction to be integrated with reasoning about constraints, from modelling routes on maps [Shen et al., 2018, Xu et al., 2018] to approving loan predictions [Mahoney and Mohen, 2007]. That is, when modelling routes, we may require the prediction model to respect the presence of physical paths between nodes on the map, in the sense of disallowing impossible or infeasible paths. Analogously, when approving loans, we may have categorical requirements that loans should not be approved for those with a criminal record, but we may also have conditional constraints for eliminating bias, e.g., the prediction should not penalize the individual based on gender.

Broadly, background information may come in different forms, including independency [Zemel et al., 2013, Zafar et al., 2015] constraints and logical formulas [Kisa et al., 2014, Xu et al., 2018], but of course the challenge is if and how we are able to provide (or learn) a model that is able to handle all the declared constraints. To the best of our knowledge, this is largely an open problem, at least in the sense of providing a general solution to a certain class of probabilistic models.
In addition to incorporating prior knowledge as constraints for training a probabilistic model, a second and equally significant way to utilize constraints is in order to enforce a set of properties on the resulting models. For example, historic data on college admissions exhibit a clear bias based on gender or race [Leonard and Jiang, 1999, Silverstein, 2000]. More generally, there is an abundance of data that reflect historical or cultural biases, prompting the rapid development of the area of fair machine learning [Zafar et al., 2015, Hardt et al., 2016]. Roughly, the idea is to place a constraint (e.g. a formalization that captures, for example, demographic parity [Zafar et al., 2015] or equality of opportunity [Hardt et al., 2016]) on the predictions of the resulting model so that biased behaviour is not exhibited.

In this paper, we consider a mathematical enquiry on the definability of constraints while training/learning a probabilistic model. Note however that performing inference on probabilistic models is a computationally intractable problem [Bacchus et al., 2009], requiring additional, often computationally intensive, subroutines in order to approximate inference. This has given rise to tractable probabilistic models (TPMs) [Kisa et al., 2014, Poon and Domingos, 2011] where conditional or marginal distributions can be computed in time linear in the size of the model. Although initially limited to low tree-width models [Bach and Jordan, 2002], recent tractable models such as sum product networks (SPNs) [Gens and Domingos, 2013, Poon and Domingos, 2011] and probabilistic sentential decision diagrams (PSDDs) [Kisa et al., 2014, Liang et al., 2017] are derived from arithmetic circuits (ACs) and knowledge compilation approaches, more generally [Choi and Darwiche, 2017, Darwiche, 2002], which exploit efficient function representations and also capture high tree-width models. These models can also be learnt from data [Gens and Domingos, 2013, Liang et al., 2017] which leverage the efficiency of inference. Consider that in classical structure learning approaches for graphical models, once learned, inference would have to be approximated, owing to its intractability. In that regard, such models offer a robust and tractable framework for learning and inferring from data. Owing to these properties and their increasing popularity for a wide range of applications [Choi et al., 2015, Liang and Van den Broeck, 2019, Poon and Domingos, 2011] and several extensions have been explored as well [Molina et al., 2018, Shen et al., 2018]. We focus on SPNs in this work, but our approach could be extended to other TPMs. We aim at targeting PSDDs in future work, since they already allow for incorporating logical constraints, so an extension to handling probabilistic and causal constraints could lead to significant advantages.

We are organised as follows. We first review the recent advances in constrained machine learning. Then we briefly review SPNs, and some preliminaries on constrained optimisation. We then turn to our main results. Finally, we conclude with discussions.

2. Related work and Context

During the last years, there have been ongoing attempts to address the problem of incorporating constraints during training or in prediction. For example, [Xu et al., 2018] examine the problem of imposing certain structure in the outcome of a classification algorithm. They approach this by adding an additional term in the objective function, one accounting for the probability of a state satisfying the given constraint. [Marquez Neila et al., 2017] consider the case of training a neural network under some constraints. They create two variants of this problem, one where results from optimization theory are utilized in order to efficiently solve the problem, under hard constraints, as well as a relaxation of this problem, with soft constraints [Gill et al., 1981, Fletcher, 1987], where terms corresponding to the constraints are added into the objective function.
Alternative ways to utilize prior knowledge have been proposed as well, such as [Stewart and Ermon, 2017]. In this work, the authors propose a framework for the semi-supervised training of neural networks. The key insight is that pre-existing knowledge can be used to create a regularizer, prompting the network to satisfy this information.

Data mining is an other field that utilizes constraints. For example, [Raedt et al., 2010] attempt to develop a structured way to apply constrained programming techniques in pattern mining or rule discovery.

Introducing constraints as a way to control a model’s complexity has been explored as well. [Friedman and Van den Broeck, 2019] consider an approach where they constrain the expected value of a quantity, modelled using open-world probabilistic databases [Ceylan et al., 2016]. By doing that, they go on to show how this constraint strengthens the semantics of such databases.

Our contribution lies in introducing an approach for training generative models under probabilistic constraints. We borrow concepts from optimization theory and develop a paradigm related to [Marquez Neila et al., 2017]. A key difference is that their approach, although similar in spirit, takes into account constraints that are expressed in terms of the model’s outcomes. Thus, they correspond to functional relationships that the output variables should respect, so, consequently, they are not of a probabilistic nature. In contrast, our approach provides a way for incorporating probabilistic constraints across all variables. Indeed, in the following sections, we will provide insights about the link between these constraints and the system of equations they induce.

In our proposed framework we suggest to utilize tractable probabilistic models [Poon and Domingos, 2011, Kisa et al., 2014], where conditional or marginal distributions can be computed in time linear in the size of the model, so we can efficiently answer the conditional or marginal queries that come up when incorporating constraints. Specifically, we will base our presentation on sum-product networks (SPNs) [Poon and Domingos, 2011]. SPNs are instances of arithmetic circuits (ACs) [Choi and Darwiche, 2017] that compactly represent the network polynomial [Darwiche, 2003] of a Bayesian network (BN).

In this paper we explore the following: can SPNs be used in order to train generative models subject to probabilistic and causal constraints? We demonstrate how to incorporate various types of probabilistic relationships into the model, specifically targeting hard and soft constraints.

3. Background

In this section we will briefly review SPNs, some causality related concepts, as well as some optimization approaches.

3.1 SPNs

SPNs are rooted directed graphical models that provide for an efficient way of representing the network polynomial [Darwiche, 2000] of a BN [Poon and Domingos, 2011], as a multilinear function \( \sum_x f(x) \prod_{n=1}^N \mathbb{1}_{x_n} \). Here \( f(\cdot) \) is the (possibly unnormalized) probability distribution of the BN, \( x \) is a vector containing all the variables of the model, i.e., \( x_1, \cdots, x_N \), the summation is over all possible states, and \( \mathbb{1}_{x_n} \) is the indicator function. An SPN \( S \) over Boolean variables \( x_1, \cdots, x_N \) has leaves corresponding to indicators \( \mathbb{1}_{x_1}, \cdots, \mathbb{1}_{x_a} \) and \( \mathbb{1}_{\bar{x}_1}, \cdots, \mathbb{1}_{\bar{x}_a} \) and whose internal nodes are sums and products.
Any edge exiting a sum node has a non-negative weight assigned to it. The value of a product node is the product of its children, \( \sum_{u_j \in \text{Ch}(u_i)} w_{ij} \mathcal{S}_j(x) \), where \( \text{Ch}(u_i) \) is the set containing the children of node \( u_i \), and \( \mathcal{S}_j \) is the sub-SPN rooted at node \( u_j \). SPNs can represent a wide class of models, including weighted mixtures of univariate distributions; see [Poon and Domingos, 2011] for discussions.

3.2 Causality

Causal inference is an approach where, apart from probabilistic information, extra information about the mechanism governing the variables’ interactions are encoded into the model. This allows reasoning about more complex queries, such as interventions and counterfactuals [Pearl, 2009b]. These can be seen as extending standard probabilities with the ability to infer what happens if a variable is forced to attain a value, by an external intervention, or what would happen had a variable obtained a different value from the one it obtained in the actual world.

The usual setting is to represent the set of probabilistic dependencies through a BN, but on top of that encode the specific mechanism that determines the value of each variable, too. In this sense, it is more general than just having a BN, since we not only possess a distribution over the variables, but also a set of equations.

An interesting remark is that, although the structural equations are essential for the specification of the model, it turns out that once you have a fully specified probabilistic distribution, it is possible to answer interventional queries without possessing the functional equations [Pearl, 2009a]. In our approach we are going to utilize the following formula to compute the effect of intervening on a variable, \( A \), on the rest of the model’s variables, \( X_{-A} \) [Pearl, 2009b]:

\[
\text{Pr}(X_{-A} | do(A = \alpha)) = \frac{\text{Pr}(X_{-A}, A = \alpha)}{\text{Pr}(A = \alpha | pa_A)}
\]

where \( pa_A \) denotes the set of \( A \)'s parents.

3.3 Optimization

Constrained optimization is a discipline concerned with developing techniques allowing for optimizing functions under a set of constraints. For example, Figure 1 depicts the problem of minimizing a function, while requiring the solution to belong in an ellipse. One of the most common ways to
address that, is to transform the objective function, so it takes the constraints into account. The problem of interest is to maximize the likelihood of a model (with a vector of parameters \( w \)), \( L(w) \) under constraints \( C_i(w) = 0, 1 \leq i \leq N \), so:

\[
\max_w L(w), \quad s.t. \ C_1(w) = 0, \cdots, C_N(w) = 0
\]

The transformed objective function, \( \Lambda \), introduces a number of auxiliary variables, as many as the constraints, \( \lambda_1, \cdots, \lambda_N \), and takes the following form

\[
\Lambda(w, \lambda_1, \cdots, \lambda_N) = L(w) + \sum_{n=1}^{N} \lambda_n C_n(w).
\]

It can be shown that all of the solutions of the original problem correspond to stationary points of the new objective function [Protter, 1985].

There are various numerical methods to solve this problem, such as projected gradient descent, where an initial vector \( w^{(0)} \) is updated incrementally, and then gets projected onto the surface defined by the constraints, until it converges to a solution of the problem. Furthermore, in cases where the objective function is in a special form, such as a quadratic polynomial, other approaches might be more efficient. See [Marquez Neila et al., 2017] for a more extensive discussion on the subject.

Optimization problems like the above require all of the feasible solutions to satisfy the constraints. These constraints are referred to as hard. Alternative formulations of the problem could yield feasible solutions not satisfying the constraints. These constraints are called soft, because instead of demanding the solutions to adhere to them, we introduce a penalty term in the objective function, for each time they get violated. For example, if all of the \( C_i(w) = 0, 1 \leq i \leq N \) were treated as soft constraints, then after setting \( \lambda_1, \cdots, \lambda_N \) to some value reflecting the cost of violating the corresponding constraint, the soft version of the problem would be to maximize the function \( L(w) + \sum_{n=1}^{N} \lambda_n C_n(w) \), so each time some \( C_i \) is not equal to zero, it induces a penalty. In this case, all \( \lambda_i \) are treated as hyperparameters, so they are specified before the optimization takes place. Furthermore, now we are interested in the maxima of this function, as opposed to the case of hard constraints, where we were interested in the stationary points of the transformed function.

4. Main Results

The majority of contemporary machine learning models rely on maximum likelihood (ML) estimation for setting the values of their parameters. The approaches we discussed earlier transform the optimization objective, enhancing the resulting model with additional properties. One limitation, in such a setting, is that the constraints are expressed in terms of the parameters, directly [Marquez Neila et al., 2017]. This is useful in situations where we require some parameters to be equal to each other, or their difference to exceed some threshold. However, in most models it is not clear how probabilistic relationships can be expressed in term of the parameters, making it difficult to utilize the existing approaches in order to achieve our goal.

Our approach is motivated from such formulations, but appeals on the following idea: since the majority of machine learning models are differentiable and utilize ML estimation, if we could find a class of models where it is feasible to uncover a correspondence between parameters and probabilities, then we could use constrained optimization approaches, in order to equip the model with additional properties. The modeller would provide the constraints in terms of the variables modelling the domain, i.e. the random variable in a generative model; arguably, this is a natural and intuitive way to express domain knowledge.

Most probabilistic constraints are expressed as an equality between probabilities. For example, if we want to incorporate the assumption that “\( A \) is independent of \( B \)”, we have to ensure that the
equality \( \Pr(A, B) = \Pr(A) \Pr(B) \) holds in the trained model. Due to space limitations, the proofs are presented in the appendix.

### 4.1 Conditional constraints

We will start with presenting the case of constraining the likelihood so it enforces the various conditional distributions of some variables to be equal. Formally, assume a variable \( Y \), a variable \( A \), whose values we would like to condition on, and a set of variables, \( X \). We are interested in modelling the joint distribution of these variables, but we would also like to incorporate some background knowledge into the model, specifically we would like it to satisfy the condition \( \Pr(Y|A = \alpha, X) = \Pr(Y|A = \alpha', X) \), where we assume that \( A \) is a binary variable, in order to make the presentation easier to follow. In this equation we do not explicitly specify the values of the variables in \( X \), rather we want the condition to hold regardless of their specific instantiation. We could also be interested in constraints of the form \( \Pr(Y|A = \alpha) = \Pr(Y|A = \alpha') \), where in this case we do not condition on \( X \). Constraints similar to this appear in the fair AI literature [Zemel et al., 2013, Zafar et al., 2015, Hardt et al., 2016, Grgić-Hlača et al., 2016], where the objective is to eliminate bias, such as racial discrimination, from predictive models, by enforcing an appropriate set of conditions.

An additional remark about the flexibility of expressing constraints in this form can be seen when considering context-specific properties [Zhang and Poole, 1999]. In the above formulation we left the values of \( X \) unspecified, but there might be cases where it is known that some properties hold only when some of the remaining variables acquire specific values. To take such information into account we should just adapt the constraint so some or all of the variables in \( X \) are set to their corresponding values, for example, such a constraint could look like \( \Pr(Y|A = \alpha, X = x) = \Pr(Y|A = \alpha', X = x) \).

As we have stated above, we are going to use SPNs to model the data, due to their provable tractability and applicability in a wide range of problems and the fact that a clean connection between probabilistic queries and the model’s parameters can be established. As we will see, this is crucial for our approach, in the sense that in the general case it is not clear how to achieve this connection, but the polynomial representation of SPNs allow us to uncover it and use it to train such a model under a set of probabilistic constraints.

The following results establishes the relationship between probabilistic constraints and the parameters of an SPN, \( w \). (An analogous statement applies to the other variants discussed above.)

**Theorem 1.** Let \( S \) be an SPN representing the joint distribution of variables \( X_1, \ldots, X_n \). Let \( X_i, X_j \) be two binary variables, then the constraint \( \Pr(X_i|X_j = 0) = \Pr(X_i|X_j = 1) \) is equivalent to a multivariate linear system of two equations on the SPN’s parameters.

The proofs for this and subsequent theorems, as well as the systems of equations, are provided in the appendix. These constraints are expressed in terms of the model’s parameters. They are multivariate linear polynomials, since in each of the resulting equations, there are two products, where the terms that appear in one factor don’t appear on the other one. Intuitively, the idea is to express the constraints in terms of the network polynomial and then equate the coefficients of the monomials.
4.2 Interventional constraints

A more complex class of distributions, used extensively in causal modelling [Pearl, 2009a], are interventional ones. They represent the probability of a variable after an external intervention on another variable. It is not always possible to estimate them using observational distributions, but when assuming that all of the model’s variables are observed, then it is possible to express the interventional distribution in terms of the observational one [Pearl, 2009b]. For the rest of this section we will make the closed-world assumption, meaning that there are no unobserved confounders between the variables. Incidentally, causal modelling concepts have gained prominence in the machine learning literature [Pearl, 2018, Zhang et al., 2017].

The new objective is to train a model while incorporating constraints of the form \( \Pr(X_{-A}|do(A = \alpha)) = \Pr(X_{-A}|do(A = \alpha')) \), where \( X_{-A} \) denotes the set of all the model’s variables, excluding \( A \). Constraints of this kind have powerful implications regarding the causal mechanisms between \( A \) and the rest of the variables. This could be seen clearly when considering similar constraints to the one above, such as \( \Pr(X_{-A}|do(A = \alpha)) = \Pr(X_{-A}) \), which means that setting \( A \) to a certain value does not influence the distribution of the rest of the variables. Intuitively, this means that \( A \) has no causal influence on any of the remaining variables.

As we have mentioned in a previous section, we will base our approach on a well known formula connecting the interventional to the observational distribution [Pearl, 2009b]:

\[
\Pr(X_{-A}|do(A = \alpha)) = \frac{\Pr(X_{-A}, A = \alpha)}{\Pr(A = \alpha|pa_A)}
\]

Depending on the application, it is possible there is enough background knowledge available to specify \( pa_A \). There might be other applications though, where this is not an option, due to the complexity of the problem or insufficient a priori information. In these cases, methods from the field of feature selection [Guyon and Elisseeff, 2003] could be utilized. The aim of these methods is to identify the Markov Blanket of a set of variables, so it is closely related to specifying the parents of a variable. Conditioning on the Markov Blanket, instead of just the parents, can serve as an approximation of the desired distribution, so there is a wide range of methods [Zhang et al., 2011, Peters et al., 2016, Zheng et al., 2018] for performing this step. Assuming we possess the parents of the variable of interest, we can show the following:

**Theorem 2.** Let \( S \) be an SPN representing the joint distribution of variables \( X_1, \cdots, X_n \). Let \( X_i \) be a binary variable, then the constraint \( \Pr(X_{-i}|do(X_i = 0)) = \Pr(X_{-i}|do(X_i = 1)) \) is equivalent to a multivariate linear system of equations on the SPN’s parameters.

4.3 Independence constraints

The last kind of constraints we will present are those enforcing independence between variables. There are some already existing approaches, such as [Xu et al., 2018], allowing for incorporating rules expressed as propositional formulas within the model, in order for example to impose certain structure to the outcome variable, but doing the same with probabilistic ones still poses a major challenge.

Using reasoning analogous to the previous cases, it is possible to incorporate conditional independence or context specific information within the model. Although similar in spirit, since usually
both of them relies on conditioning, each one provides different insights about the problem at hand. So, for example, conditional constraints could be of the form: if we know the value of a variable, \( z \), then \( A \) and \( B \) are independent. On the other hand, context specific independence is stronger, since it might state that only when \( Z = z \) we know that \( A \) and \( B \) are independent. However, it is not difficult to see that each of these independencies can be expressed as \( \Pr(A, B|Z) = \Pr(A|Z) \Pr(B|Z) \) and \( \Pr(A, B|Z = z) = \Pr(A|Z = z) \Pr(B|Z = z) \), respectively.

Assuming, as before, that the objective is to train an SPN satisfying constraints like the above, we can show that it amounts to optimizing a function over a set of multivariate quadratic polynomial constraints.

**Theorem 3.** Let \( S \) be an SPN representing the joint distribution of variables \( X_1, \ldots, X_n \). Let \( X_i, X_j \) be two binary variables, then the constraint \( \Pr(X_i, X_j) = \Pr(X_i) \cdot \Pr(X_j) \) is equivalent to a multivariate quadratic system of four equations on the SPN’s parameters.

---

**Algorithm 1:** Training with soft constraints

**Input:** Set \( D \) of instances over variables \( X \), a set of soft constraints \( C_n, \alpha \)

**Output:** An SPN with learned structure and parameters

1. \( S \leftarrow \text{GenerateDenseSPN}(X) \);
2. \( \text{InitializeWeights}(S) \);
3. repeat
4. \( \text{Sample a mini batch } M \);
5. for all \( d \in D \) do
6. \( \text{UpdateWeights}(S + \alpha \sum_n C_n, \text{Inference}(S + \alpha \sum_n C_n, d)) \);
7. end for;
8. until convergence
9. \( S \leftarrow \text{PruneZeroWeights}(S) \);
10. return \( S \)

---

5. Applying the framework

In this section we will demonstrate how to derive the system of equations that correspond to a single constraint. Let’s assume we would like to train an SPN, \( S \), over three binary variables, \( X_1, X_2, X_3 \), satisfying the property that \( X_1 \) and \( X_2 \) are independent. The canonical polynomial of \( S \) [Darwiche, 2003] is:

\[
S(X_1, X_2, X_3, \neg X_1, \neg X_2, \neg X_3) = \theta_1 X_1 X_2 X_3 + \theta_2 \neg X_1 X_2 X_3 + \theta_3 X_1 \neg X_2 X_3 \\
+ \theta_4 \neg X_1 \neg X_2 X_3 + \theta_5 X_1 \neg X_2 \neg X_3 + \theta_6 \neg X_1 X_2 \neg X_3 + \theta_7 X_1 X_2 \neg X_3 + \theta_8 \neg X_1 \neg X_2 \neg X_3 \tag{1}
\]

where each \( \theta_i \) is equal to the probability of the specific configuration of \( X_1, X_2, X_3 \) following it, so, for example, in the term \( \theta_5 X_1 \neg X_2 \neg X_3 \), \( \theta_5 = \Pr(X_1, \neg X_2, \neg X_3) \)

The joint probability of, say, \( X_1, X_2 \) is given by (1), after substituting both \( X_3, \neg X_3 \) by 1, so \( \Pr(X_1, X_2) = S(X_1, X_2, 1, \neg X_1, \neg X_2, 1) \). In the same way, \( \Pr(X_1) = S(X_1, 1, 1, \neg X_1, 1, 1) \) and \( \Pr(X_2) = S(1, X_2, 1, 1, \neg X_2, 1) \).
At this point, it is time to utilize the condition we would like to enforce, \( \text{Pr}(X_1, X_2) = \text{Pr}(X_1) \cdot \text{Pr}(X_2) \). Substituting these probabilities by the corresponding polynomial, yields the following:

\[
(\theta_1 + \theta_7)X_1X_2 + (\theta_3 + \theta_5)X_1\neg X_2 + (\theta_2 + \theta_6)\neg X_1X_2 + (\theta_4 + \theta_8)\neg X_1\neg X_2 = \\
(\theta_1 + \theta_3 + \theta_5 + \theta_7) \cdot (\theta_1 + \theta_2 + \theta_6 + \theta_7)X_1X_2 + (\theta_1 + \theta_3 + \theta_5 + \theta_7) \cdot (\theta_3 + \theta_4 + \theta_5 + \theta_8)X_1\neg X_2 + \\
(\theta_2 + \theta_4 + \theta_6 + \theta_8) \cdot (\theta_1 + \theta_2 + \theta_6 + \theta_7)\neg X_1X_2 + (\theta_2 + \theta_4 + \theta_6 + \theta_8) \cdot (\theta_3 + \theta_4 + \theta_5 + \theta_8)\neg X_1\neg X_2
\]

This is an equivalence between polynomials, so all the coefficients must be equal, meaning that:

\[
\theta_1 + \theta_7 = (\theta_1 + \theta_3 + \theta_5 + \theta_7) \cdot (\theta_1 + \theta_2 + \theta_6 + \theta_7), \quad \theta_3 + \theta_5 = (\theta_1 + \theta_3 + \theta_5 + \theta_7) \cdot (\theta_3 + \theta_4 + \theta_5 + \theta_8) \\
\theta_2 + \theta_6 = (\theta_2 + \theta_4 + \theta_6 + \theta_8) \cdot (\theta_1 + \theta_2 + \theta_6 + \theta_7), \quad \theta_4 + \theta_8 = (\theta_2 + \theta_4 + \theta_6 + \theta_8) \cdot (\theta_3 + \theta_4 + \theta_5 + \theta_8)
\]

At this point, since each \( \theta_i \) has probabilistic semantics, we perform a sanity check, by rewriting the system in terms of these probabilities. This will provide some insights on the underlying constraints, as well as some hints on alternative ways to incorporate the constraints in the model.

\[
\begin{align*}
\theta_1 + \theta_7 &= \text{Pr}(X_1, X_2), \quad \theta_2 + \theta_6 = \text{Pr}(\neg X_1, X_2), \quad \theta_3 + \theta_5 = \text{Pr}(X_1, \neg X_2), \quad \theta_4 + \theta_8 = \text{Pr}(\neg X_1, \neg X_2) \\
\theta_1 + \theta_3 + \theta_5 + \theta_7 &= \text{Pr}(X_1), \quad \theta_2 + \theta_4 + \theta_6 + \theta_7 = \text{Pr}(X_2), \\
\theta_3 + \theta_4 + \theta_5 + \theta_8 &= \text{Pr}(\neg X_2), \quad \theta_2 + \theta_4 + \theta_6 + \theta_8 = \text{Pr}(\neg X_1)
\end{align*}
\]

Substituting all these quantities to the original system, we get the following constraints:

\[
\begin{align*}
\text{Pr}(X_1, X_2) &= \text{Pr}(X_1) \cdot \text{Pr}(X_2), \quad \text{Pr}(X_1, \neg X_2) = \text{Pr}(X_1) \cdot \text{Pr}(\neg X_2), \\
\text{Pr}(\neg X_1, X_2) &= \text{Pr}(\neg X_1) \cdot \text{Pr}(X_2), \quad \text{Pr}(\neg X_1, \neg X_2) = \text{Pr}(\neg X_1) \cdot \text{Pr}(\neg X_2)
\end{align*}
\]

If they are incorporated as soft constraints, new terms are added in the objective function, but since all of them are differentiable, any standard optimization algorithm could be utilized to train the model. Algorithm 1 is based on the original LearnSPN algorithm [Poon and Domingos, 2011], but it is extended so now it allows for incorporating soft constraints. The only addition is that we included some extra terms in the objective function, weighted by the hyperparameter \( \alpha \). We would like to note that for this version we do not have to explicitly compute the constraints in terms of the parameters. This follows from the observation that each probabilistic expression corresponds to a sub-SPN, so adding these sub-SPNs to the original SPN, \( S \), results in another SPN, so we can treat the whole expression \( S + \alpha \sum_n C_n \) as a new SPN and use the LearnSPN algorithm. If, for example, the constraint is \( \text{Pr}(X_1, X_2) = \text{Pr}(X_1) \cdot \text{Pr}(X_2) \), then the corresponding sub-SPNs are those representing the distributions of \( \{X_1, X_2\}, \{X_1\}, \{X_2\} \), which can be readily obtained from \( S \).

In contrast, if they are treated as hard constraints, projected gradient descent or approaches like the one developed in [Marquez Neila et al., 2017] would need to be used to train the SPN. Algorithm 2 is based on the LearnSPN algorithm, again, but this time it is adapted in order to train an SPN under hard constraints. The modification lies on the fact that after the weights are updated, then they are projected on the space defined by the constraints, see [Marquez Neila et al., 2017]. For this variant our results are essential, since the equations cannot be handled implicitly, as was possible with soft constraints. Our approach, as seen in this example, provides a way to recover exactly these equations, so training with hard constraints can be made possible. In our opinion, although incorporating hard constraints is more involving, it is worth exploring this approach, since using soft constraints, as in [Xu et al., 2018], does not guarantee the resulting model will satisfy them.
Algorithm 2: Training with hard constraints

Input: Set D of instances over variables X, a set of constraints C_n,

Output: An SPN with learned structure and parameters

1. S ← GenerateDenseSPN(X);
2. InitializeWeights(S);
3. repeat;
4. Sample a mini batch M;
5. for all d ∈ D do;
6. UpdateWeights(S, Inference(S,d));
7. Project the weights to the space defined by C_n;
8. end for;
9. until convergence
10. S ← PruneZeroWeights(S);
11. return S

6. Conclusions

In the previous sections we presented an approach allowing to train SPNs under probabilistic constraints. SPNs are tractable models, meaning that probabilistic inference is efficient, since marginal or conditional queries can be computed in time linear in its size. This is an appealing property, because otherwise additional steps, such as MCMC sampling, would be necessary in order to perform inference. Taking that into account, SPNs can not only incorporate probabilistic assumptions, but they can also easily compute such queries in polynomial time.

An other interesting point is that our work could be seen as related to the work that has been done in the field of Fairness in AI, but from a generative modelling point of view. The main objective in the field is to formalize criteria leading to fair predictions, and train models satisfying these criteria. For example, enforcing a condition such as \( \Pr(\hat{y} = 1|a = 0) = \Pr(\hat{y} = 1|a = 1) \), where \( a \) is a protected binary attribute and \( \hat{y} \) is the model’s prediction, has been proposed [Zemel et al., 2013]. In our setting there is no predicted variable, so this condition cannot be applied. However, an analogous condition could be utilized when dealing with generative modelling, such as \( \Pr(y = 1|a = 0) = \Pr(y = 1|a = 1) \).

In this work we provided a way to equip SPNs with background information. This adds to the growing literature on constraints and machine learning that is emerging recently. The key difference in our results is that it is proven for generative models, unlike the majority of the existing work, as well as it exhibits how the model’s intrinsic architecture can be utilized to do so, allowing us to recover a system of equations. We hope the results of this paper will lead to a new range of applications making use of tractable generative models that allow the incorporation of non-trivial probabilistic prior knowledge. Extending this work to additionally incorporate logical background knowledge is the most immediate next step.

References


7. Appendix

7.1 Proofs of the theorems

**Theorem 1.** Let \( S \) be an SPN representing the joint distribution of variables \( X_1, \cdots, X_n \). Let \( X_i, X_j \) be two binary variables, then the constraint \( \Pr(X_i|X_j = 0) = \Pr(X_i|X_j = 1) \) is equivalent to a multivariate linear system of two equations on the SPN’s parameters.

**Proof:** Let \( S(x) = \sum_x f(x) \prod_{i=1}^N 1_{x_i} \) be the network polynomial of an SPN. The equality \( \Pr(X_i|X_j = 0) = \Pr(X_i|X_j = 1) \) can be rewritten as follow:

\[
\Pr(X_i, X_j = 1) = \Pr(X_i|X_j = 0) \Pr(X_j = 1) = \frac{\Pr(X_i, X_j = 1)}{\Pr(X_j = 1)} = \frac{\Pr(X_i, X_j = 0)}{\Pr(X_j = 0)}
\]

Next, we express the above probabilities in terms of \( S \) (where \( X \) corresponds to the assignment \( X = 1 \), and \( \neg X \) to \( X = 0 \)):

\[
\begin{align*}
\Pr(X_i, X_j = 1) &= \sum_{x_i = x_j = 1} f(x) \cdot 1_{x_i} + \sum_{x_i = x_j = 0} f(x) \cdot 1_{\neg x_i} \\
\Pr(X_i, X_j = 0) &= \sum_{x_i = x_j = 1} f(x) \cdot 1_{x_i} + \sum_{x_i = x_j = 0} f(x) \cdot 1_{\neg x_i} \\
\Pr(X_j = 1) &= \sum_{x_j = 1} f(x) \\
\Pr(X_j = 0) &= \sum_{x_j = 0} f(x)
\end{align*}
\]

We now substitute these equations to (2) to get that:

\[
\begin{align*}
\sum_{x_j = 0} f(x) \cdot \sum_{x_i = x_j} f(x) \cdot 1_{x_i} + \sum_{x_j = 1} f(x) \cdot \sum_{x_i = x_j} f(x) \cdot 1_{\neg x_i} &= \\
\sum_{x_j} f(x) \cdot \sum_{x_i = x_j} f(x) \cdot 1_{x_i} + \sum_{x_j = 1} f(x) \cdot \sum_{x_i = x_j} f(x) \cdot 1_{\neg x_i}
\end{align*}
\]

This is an equality between polynomials, meaning that the coefficients must be equal, so:

\[
\begin{align*}
\sum_{x_j = 0} f(x) \cdot \sum_{x_i = x_j} f(x) &= \sum_{x_i} f(x) \cdot \sum_{x_j = 1} f(x) \cdot 1_{x_i} \\
\sum_{x_j = 0} f(x) \cdot \sum_{x_i = x_j} f(x) &= \sum_{x_i} f(x) \cdot \sum_{x_j = 1} f(x) \cdot 1_{\neg x_i}
\end{align*}
\]

These constraints are expressed in terms of the model’s parameters and they are, clearly, multivariate polynomials, specifically linear ones, since, in each equations, there are two products, so if we look, for example, at the ones in the first equation, \( \sum_{x_j} f(x) \cdot \sum_{x_i = x_j} f(x) \) and \( \sum_{x_i = x_j} f(x) \cdot \sum_{x_j} f(x) \), the terms that appear in one factor don’t appear on the other one, since the summation is performed over disjoint sets.

**Theorem 2.** Let \( S \) be an SPN representing the joint distribution of variables \( X_1, \cdots, X_n \). Let \( X_i \) be a binary variable, then the constraint \( \Pr(X_{-i}|\text{do}(X_i = 0)) = \Pr(X_{-i}|\text{do}(X_i = 1)) \) is equivalent to a multivariate linear system of equations on the SPN’s parameters.
**Proof:** We will prove this, following the same reasoning as in the previous proof, so we first need to rewrite the given constraint:

\[
\begin{align*}
\Pr(X_\cdot |do(X_i = 0)) &= \Pr(X_\cdot |do(X_i = 1)) \\
\Rightarrow \Pr(X_\cdot, X_i = 0) &= \Pr(X_\cdot, X_i = 1) \\
\Rightarrow \Pr(X_i = 0|pa_{X_i}) &= \Pr(X_i = 1|pa_{X_i}) \\
\Rightarrow \Pr(X_\cdot, X_i = 0) \cdot \Pr(X_i = 1|pa_{X_i}) &= \Pr(X_\cdot, X_i = 1) \cdot \Pr(X_i = 0|pa_{X_i}) \\
\Rightarrow \Pr(X_\cdot, X_i = 0) &= \Pr(X_\cdot, X_i = 1) \cdot \Pr(X_i = 0|pa_{X_i}) \
\end{align*}
\]

The next step is to express these probabilities in terms of the network polynomial and substitute them to the above expression. Since these computations are lengthy and routine, we will not present them here. The important observation is that it is not difficult to see that we end up with a system of multivariate polynomials, in this case, too. To prove they are linear ones as well, it suffices to note that in both products \(\Pr(X_\cdot, X_i = 0)\) \(\cdot\) \(\Pr(X_i = 1|pa_{X_i})\) and \(\Pr(X_\cdot, X_i = 1)\) \(\cdot\) \(\Pr(X_i = 0, pa_{X_i})\), the set of parameters involved in the first factor is disjoint with the one appearing in the second factor, since the parameters that remain after setting \(X_i = 0\) vanish when setting \(X_i = 1\) (and vice versa). \(\square\)

**Theorem 3.** Let \(S\) be an SPN representing the joint distribution of variables \(X_1, \cdots, X_n\). Let \(X_i, X_j\) be two binary variables, then the constraint \(\Pr(X_i, X_j) = \Pr(X_i) \cdot \Pr(X_j)\) is equivalent to a multivariate quadratic system of four equations on the SPN’s parameters.

**Proof:** To prove this result it is not necessary to rewrite the given constraint, so we can start with expressing these probabilities in terms of \(S\):

\[
\begin{align*}
\Pr(X_i, X_j) &= \sum_{x_{i \cdot} x_j} f(x) \cdot x_{i \cdot} \cdot x_j + \sum_{x_i \cdot, x_j} f(x) \cdot \neg x_i \cdot \neg x_j \\
&\quad + \sum_{x_i \cdot, \neg x_j} f(x) \cdot x_i \cdot \neg x_j + \sum_{x_i \cdot, \neg x_j} f(x) \cdot \neg x_i \cdot x_j \\
\Pr(X_i) &= \sum_{x_i} f(x) \cdot x_i + \sum_{x_i} f(x) \cdot \neg x_i \\
\Pr(X_j) &= \sum_{x_j} f(x) \cdot x_j + \sum_{x_j} f(x) \cdot \neg x_j \\
\end{align*}
\]

Next, we substitute these quantities to the constraint’s equation, so we get that:

\[
\begin{align*}
&\sum_{x_{i \cdot} x_j} f(x) \cdot x_{i \cdot} \cdot x_j + \sum_{x_i \cdot, x_j} f(x) \cdot \neg x_i \cdot x_j + \sum_{x_i \cdot, x_j} f(x) \cdot x_i \cdot \neg x_j \\
&+ \sum_{x_i \cdot, \neg x_j} f(x) \cdot \neg x_i \cdot \neg x_j = \sum_{x_i} f(x) \cdot \sum_{x_j} f(x) \cdot x_i \cdot x_j \\
&+ \sum_{x_i} f(x) \cdot \sum_{x_j} f(x) \cdot x_i \cdot \neg x_j + \sum_{x_i} f(x) \cdot \sum_{x_j} f(x) \cdot \neg x_i \cdot x_j \\
&+ \sum_{x_i} f(x) \cdot \sum_{x_j} f(x) \cdot \neg x_i \cdot \neg x_j
\end{align*}
\]
Equating the coefficients we get the following system of equations:

\[
\sum_{x_i, x_j} f(x) = \sum_{x_i} f(x) \cdot \sum_{x_j} f(x) \\
\sum_{x_i, x_j} f(x) = \sum_{x_i} f(x) \cdot \sum_{x_j} f(x) \\
\sum_{x_i, \neg x_j} f(x) = \sum_{x_i, x_j} f(x) \cdot \sum_{x_i, \neg x_j} f(x) \\
\sum_{\neg x_i, x_j} f(x) = \sum_{\neg x_i, \neg x_j} f(x) \cdot \sum_{\neg x_i, x_j} f(x)
\]

Each of these equations correspond to a multivariate polynomial, as in all the previous cases, but this time they are quadratic, instead. This is because, in each equation, the sums appearing on the right hand side have some terms in common. For example, looking at the first equation, the assignment setting all the variables equal to 1 is compatible with both summations, so the term \(f(x_1, \ldots, x_n)\) appears in both of them. Clearly, by multiplying them we end up with a squared parameter. □