Title
Unifying Probabilistic Reasoning, Learning, and Classification with Circuit Representations

Author
Liang, Yitao

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Unifying Probabilistic Reasoning, Learning, and Classification with Circuit Representations

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Computer Science

by

Yitao Liang

2021

ABSTRACT OF THE DISSERTATION

Unifying Probabilistic Reasoning, Learning,
and Classification with Circuit Representations

by

Yitao Liang
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2021
Professor Guy Van den Broeck, Chair

On one side, symbolic methods represent our knowledge of the world, and when coupled with probabilistic reasoning, one can infer the uncertainty of unknown facts conditioned on the knowledge of known evidence. On the other side, statistical machine learning finds and infers a predictive function from vast amounts of data. To automate complex decision making, both are crucially needed. While their unification has been long pursued, the two up to now still largely remain disparate from one another. This dissertation demonstrates circuit representations are a promising symbolic-statistical synthesis. In particular, we study circuit representations across the dimensions of tractable probabilistic reasoning with and without logical constraints, structure learning from data, and classifying on image domains, namely, the main tasks across symbolic and statistical methods. And more importantly, we show those dimensions are unified for circuit representations by leveraging the same syntactic properties.

The dissertation of Yitao Liang is approved.

Yizhou Sun

Stefano Soatto

Kai-Wei Chang

Guy Van den Broeck, Committee Chair

University of California, Los Angeles

2021

To Chenyu,

and our reunion.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I certainly did not expect pursing a Ph.D. degree would have been easy. However, having to finish my Ph.D. studies amid a global pandemic is definitely a challenge and up to now still feels unreal. All the ups and downs have altogether made this an unforgettable chapter of my life.

First and foremost, I am deeply blessed and honored to have Guy Van den Broeck as my advisor, who has taught me to be a creative researcher, an independent thinker, a responsible collaborator, and more importantly a good person thinking of the well-being of others. Thank you for guiding me to focus on big long-standing questions, encouraging me to carry on from setbacks, and persistently challenging me to achieve what is beyond my imagination. You have been more than just a research advisor, but also a great mentor in life. From you, I have learned the great importance of passion, grit, and dedication.

I also would like to thank every other member of my committee, Prof. Stefano Soatto, Prof. Kai-Wei Chang, and Prof. Yizhou Sun, for sharing of their experiences and their valuable suggestions and advice over the years. They have each brought me new insightful perspectives about my research and have greatly inspired me to re-think about my research in a bigger picture.

I always consider my self extremely lucky to have wonderful collaborators and have made incredible friends in LA: Kareem Ahmed, Jessa Bekker, Chen Chen, Robert Chen, YooJung Choi, Meihua Dang, Tal Friedman, Shunqi Gao, Steven Holtzen, Tong He, Tianbai Jia, Kuan-Hao Huang, Liunian Li, Anji Liu, Yang Liu, Tao Meng, Yujia Shen, Andy Shih, Erin Talvitie, Pasha Khosravi, Antonio Vergari, Zhe Zeng, Chi Zhang, Honghua Zhang, and Jieyu Zhao. I owe each of you a special thank you for celebrating the highs and staying with me for the lows. I will miss you.

I am fortunate to have Chenyu. Your smile always melts my heart. You are the inspiration and happiness of my life. Even long distance cannot reduce your caring by the slimmest amount. Your company has made this journey warm.

Lastly, I could not be more grateful to the unconditional love, care, sacrifice, and support from my parents.

VITA

2013 – 2016 B.A. (Computer Science) Franklin & Marshall College

2016 – 2019 M.S. (Computer Science) University of California, Los Angeles

PUBLICATIONS

Yitao Liang, Guy Van den Broeck. Learning Logistic Circuits. *In the Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI), 2019.*

Jingyi Xu, Zilu Zhang, Tal Friedman, **Yitao Liang**, Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. *In the Proceedings of the 35th International Conference on Machine Learning (ICML), 2018.*

Yitao Liang, Jessa Bekker, Guy Van den Broeck. Learning the Structure of Probabilistic Sentential Decision Diagrams. *In the Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Yitao Liang, Guy Van den Broeck. Towards Compact Interpretable Models: Learning and Shrinking Probabilistic Sentential Decision Diagrams. *In the Proceddings of IJCAI-17 Workshop on Explainable AI (XAI)*, 2017.

CHAPTER 1

Introduction

In every era of human history, people have innovated all sorts of tools to reduce manual labor. Since the dawn of the Information Revolution, scientists and engineers have begun to be not satisfied with inventing custom tools, but in their earnest belief develop autonomous agents with minimum human supervision that could potentially simulate general intelligence. To date, advances in artificial intelligence (AI) have remarkably improved the productivity and spurred new breakthroughs in many domains.

However, no one can give a precise and exact definition about what is intelligence. To be honest, intelligence is a very broad concept. Quantifying uncertainties and making appropriate planning is an embodiment of intelligence; reasoning about unknown facts based on current observations is also an embodiment of intelligence; perceiving and classifying different objects certainly is another embodiment of intelligence. The examples can go on and go on. Partly due to this reason, for most of its history, AI research has been naturally divided into different subfields, each with their own technical considerations, potentially philosophically-different pursuing goals, and use of varying tools.

On one side, in the past decade, we have witnessed tremendous progress and success in data-driven models, such as deep neural networks [LBH15]. Fueled by the concurrent advances in computation power and collection of large-scale datasets [DDS09], machines' performance in assorted high-profile challenges (e.g., Atari games, Go, object detection, etc.) have been elevated to or even surpassing human level [SZ14, MKS15, LMT16, SHM16, HGD17, SSS17]. Those success stories collectively demonstrate a competitive paradigm to approach learning in unstructured environ-

ments. On the other side, researchers have not stopped their relentless development of knowledge-driven or reasoning-driven models, such as different variants of probabilistic graphical models and arithmetic circuits [Dar09, PD11, KAD14]. Their success demonstrates a tractable approach in quantifying uncertainties and inferring knowledge of hidden facts in both unstructured and structured domains.

This dissertation sees the opportunity to blend the two paradigms together, such that potentially benefits and advances from both sides can be enjoyed simultaneously. Yet, with years of developments, even these two paradigms alone have each become an umbrella to cover assorted topics. To put this dissertation into context, we would like to start with clarifying what exactly we try to unify here. First of all, it is no surprise one cannot represent all sorts of knowledge, as knowledge exists in different forms as well. Some knowledge may even not suit any explicit form. In this dissertation, we focus on symbolic knowledge represented as logical sentences. Logic is natural to use when describing rule-based knowledge bases. For example, one can straightforwardly use logical sentences to describe courses' prerequisite requirements, such as the following [KAD14].

$$P \ \vee \ L$$
$$A \implies P$$
$$K \implies (A \vee L)$$

This knowledge base describes three requirements: (i) Must take at least Probability (P) or Logic (L); (ii) Probability (P) is a prerequisite for AI (A); (iii) To take Knowledge Representation (K), one must have taken either AI (A) or Logic (L) before. This knowledge base compactly describes the relationship between different courses and dictates what course sequence is allowed and what is not allowed. Relationships and what is valid are the two main aspects of a problem's structure. And across this dissertation, we will repeatedly leverage logic to tackle problems with structures.

With existing logical knowledge, one can deduce new knowledge. We want to be clear that this dissertation is not focused on this type of reasoning. Instead, we focus on probabilistic reasoning. In any realistic domains, knowledge rarely holds for 100% of the time. In fact, the very act of

preparing knowledge can involve leaving some facts unchecked and many facts crudely summarized. This means rules will often have exceptions. For example, professors can grant permissions to override prerequisites. To deal with exceptions and quantify our uncertainties of knowledge, one needs to introduce probability into knowledge. Now, instead of deducing new knowledge, how to infer the probability of other facts conditioned on the knowledge of some observed facts is imperative. Roughly speaking, probabilistic reasoning is dedicated to this. How observing the evidence of some facts affects the probabilities of other facts defines a joint distribution. In this dissertation, all reasoning is done with respect to a given distribution. However, introducing probability into knowledge alone has nothing to say about how we can do probabilistic reasoning efficiently. An implicit theme behind our unified representations is that they must be tractable, such that commonly used probabilistic reasoning can all be computed in polynomial time with respect to the size of the representations.

Guaranteeing tractability is not enough. Not any given distribution is desirable, as quantifying probabilities itself is tricky. Unless one has a justified (prior) belief of the uncertainties, capturing probabilities based on the occurrences in the observed data is favored. This leads to the second and third dimension of this dissertation, learning and classification. Here, learning specifically refers to generative learning (i.e., learning a joint distribution over some input features) and classification refers to discriminative learning (i.e., learning the conditional probability of a class given some input features). These are the two main approaches of statistical machine learning. To achieve both efficient probabilistic reasoning and competitive statistical machine learning, representations' structure now not only needs to encode logical knowledge but also needs to induce complex distributions that are close to the data when coupled with parameters. This imposes a unique challenge about how to leverage the representations' syntactic properties that guarantee their tractability in probabilistic reasoning to make structure learning amenable to data.

In short, this dissertation focuses on the synthesis of logic knowledge and its accompanying probabilistic reasoning with data-driven statistical machine learning. By unifying probabilistic reasoning, learning, and classification, one now has a straightforward pipeline that starts from

encoding complex spaces' domain-specific structural constraints, and ends with a competitive statistical model fitting the data well. Moreover, the resulting representations could still retain the flexibility to manipulate and answer assorted fundamental questions regarding the learned distribution. Considering logic knowledge is commonly captured by a representation's structure, to do this symbolic-statistical synthesis, we start from circuit representations with strong syntactic properties that can compactly encode logical constraints and extend their frontier to do well in machine learning tasks. This dissertation is also structured in response to our direction of synthesis (i.e., from probabilistic reasoning to more machine learning). In particular, this dissertation is presented as the following.

In Chapter 2, we review circuit representations. We first show how they naturally encode logical sentences, and then demonstrate how they can be converted into a probabilistic model by parameterizing their edges. We end this chapter with a brief discussion about probabilistic circuit representations' unique advantage in representing distributions that are subject to logical constraints.

In Chapter 3, we consider the probabilistic reasoning tasks for circuit representations. We start with walking through examples about how probabilistic circuits can be transformed into a computation graph to efficiently compute some common probabilistic reasoning queries. After this warm-up, we push the frontier of probabilistic circuits' tractable reasoning. In particular, we are the first to study the tractability of circuit representations from the point view of information measure, and propose a polynomial-time recursion algorithm to efficiently compare the probabilistic distributions captured by two circuit representations.

In Chapter 4, we switch the gear and start focusing on the learning aspect of circuit representations. In particular, we propose the first structure learning algorithm for a particular dialect of probabilistic circuit representations, called probabilistic sentential decision diagrams (PSDD). With our proposed learning algorithm, we show circuit representations are amenable to learning from data. Furthermore, we also demonstrate that this learning algorithm retains circuit representations' ability to disallow possible worlds that are not consistent with structured spaces' inherent logical constraints, which is beyond the reach of other representations.

In Chapter 5, we continue our efforts in advancing the learning capacity of circuit representations. Unlike the previous chapter which is about generative learning, this one investigates the other dominant learning type, namely discriminative learning. We assign new semantics to our circuit representations and call them logistic circuits, forming a a discriminative counterpart to probabilistic circuits. We show that parameter learning for logistic circuits is convex optimization, and that a simple local search can induce strong logistic circuit structures from data as well.

In Chapter 6, we revisit the knowledge encoding and probabilistic reasoning aspect of circuit representations. However, this time, instead of focusing on structured space over input features, we focus on leveraging constraints on representations' output space. In particular, we develop a novel circuit-based methodology for using symbolic knowledge in deep learning.

We conclude with a summary of this dissertation in Chapter 7.

CHAPTER 2

Probabilistic Circuits

In this chapter, we review a tractable deep representation of probabilistic distribution, called *Probabilistic Circuits*. In particular, we will go through not only its semantics (i.e., how a probabilistic distribution is represented), its syntax (i.e., how tractability is guaranteed by its structural properties), but also the motivation behind its invention and its connection with symbolic knowledge representation as well.

2.1 Background

Modeling usually involves representing a problem's structure. And probabilistic graphical models (PGMs) are invented to capture structures [Dar09]. For example, during this difficult time with a pandemic raging around the globe, a person with fever would potentially need to go through several different tests to figure out the true cause. A simplified probabilistic model that captures conditional independence could illustrate why this is the case; consider Figure 1.1. The figure is a directed acyclic graph (DAG). To be more specific, it shows a Bayesian network, probably the most famous probabilistic graphical model, over five variables, IA (influenza A), IB (influenza B), S (strep throat), C (Covid-19), and F (fever). Most PGMs are represented as DAGs. A line between two entities on the figure indicates their conditional dependence relationship. As the figure shows, influenza A and influenza B, along with strep throat and Covid-19 can all independently infect a person. Furthermore, they all can cause fever. There is slight difference in terms of symptoms between them. Yet the difference tends to be too subtle that even medical experts may not be able to tell. Given this, the only reliable option is to do several independent tests. Although in the figure

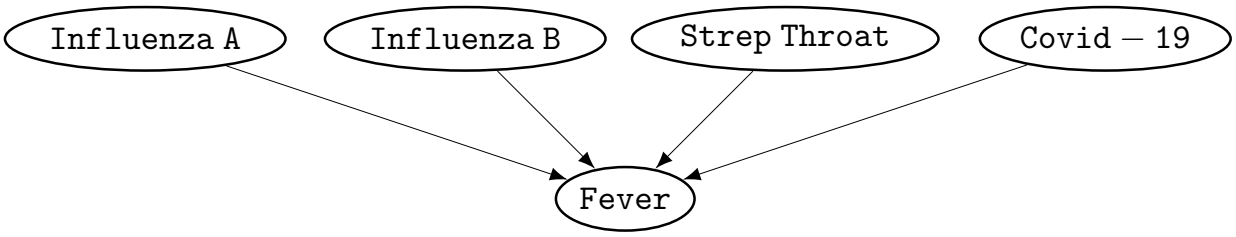


Figure 2.1: Conditional Independences between Fever and Potential Causes

we use solid lines, the dependence is by nature stochastic, such as whether having a fever is also correlated to many other hidden factors (e.g., sensitivity of individuals' immune systems). With the conditional independence, the joint probability can be decomposed into a multiplication over five pieces as follows.

$$\Pr(IA, IB, S, C, F) = \Pr(IA)\Pr(IB)\Pr(S)\Pr(C)\Pr(F \mid IA, IB, S, C)$$

Granted that PGMs are effective in representing structures, a key limitation in PGMs' learning and inference is the difficulty to calculate the partition function or the probability of evidence. What is a partition function or the probability of evidence? Extending from the previous equation from, more generally speaking, a PGM represents a joint probability distribution as a normalized product of factors:

$$\Pr(\mathbf{X} = \mathbf{x}) = \frac{1}{Z}\prod_k \theta_k(\mathbf{X}_{\{k\}}).$$

Those factors are conditionally independent from one another. Z is the partition function; its purpose is to normalize all the computed probabilities, such that the sum of the probabilities of every possible event is one.

If one is careful in observing, letters are bold, different from the previous equation. And this is done on purpose to distinguish a single variable and a set of random variables. The rule of thumb for notation is as follow. An uppercase letter X denotes a random variable and a lowercase letter x denotes an assignment to X. Literals X or $\neg X$ respectively assign true or false to variable X. Sets of variables $\mathbf{X}$ and joint assignments $\mathbf{x}$ are denoted in bold. An assignment $\mathbf{x}$ that satisfies logical

Table 2.1: A summary of important notations.

Notations	Meaning
Uppercase letters (e.g., X)	A random variable
Literals X and $\neg X$	True or False assignment to the Boolean variable X
Lowercase letters (e.g., x)	An assignment to X
Bold uppercase letters (e.g., $\mathbf{X}$)	A set of random variables
Bold lowercase letters (e.g., $\mathbf{x}$)	Joint assignments to $\mathbf{X}$
$\mathbf{x} \models \alpha$	$\mathbf{x}$ satisfies the logical sentence α
$\mathbf{XY}$ or $\mathbf{X} \cup \mathbf{Y}$	The union of sets $\mathbf{X}$ and $\mathbf{Y}$

sentence α is denoted $\mathbf{x} \models \alpha$. A complete assignment to all considered variables is called a possible world, or interchangeably an example/sample. Concatenations of sets represent their union. A summary of notations is also provided in 2.1 for easy reference. Sometimes those notations can be overloaded and denote other things. When this happens, we will make sure their overloaded meaning is clarified.

To refocus on this specific equation, $\mathbf{X}_{\{k\}}$ is a subset of variables and θ_k is a potential function over it. For most PGMs, Z is computationally difficult to obtain:

$$Z = \sum_{\mathbf{x}} \prod_{k} \theta_k(\mathbf{X}_{\{k\}}).$$

As this equation shows, a partition function potentially enumerates an exponential number of terms with respect to the number of input variables: $\mathbf{X}_{\{k\}}$ first already has an exponential number of possible combinations, and secondly every possible joint assignment $\mathbf{x}$ needs to be enumerated which is again an exponential number. What makes the matter even worse is that, depending on the specific probabilistic query, this Z-function needs to adapt. For example, when some variables are already observed and we are interested in the probability over another subset of variables, then this Z only needs to sum over all $\mathbf{x}$ then are consistent with the observation. Given what is observed

can be referred to as evidence, in this setting Z is more commonly referred to as the probability of evidence. This means, we even could not try to settle on pre-computing one Z. In fact, what we need is a model that is either directly in an easy-to-compute form or can be easily transformed into such a form.

2.2 A General Circuit Language

The assorted reasons outlined above spur the development of tractable probabilistic models. Current advances stem from two lines of work. First, probabilistic graphical model learning has long targeted sparse models [MJ00, NB04, CG07]. Second, the field of knowledge compilation studies tractable representations, such as *arithmetic circuits* (ACs) for probability distributions [Dar03], and NNF circuits for Boolean functions [DM02]. The superior tractability of these circuits derives from their ability to capture local structure and determinism [BFG96], which makes compilation to circuits a state-of-the-art technique for probabilistic inference [DDC08, CKD13].

Circuits have long been used to represent logical sentences. A logical circuit is a directed acyclic graph (DAG) representing a Boolean function, as depicted in Figure 2.2 (ignoring parameters for now). Each inner node is either an AND gate or an OR gate.[1] A leaf (input) node represents a Boolean literal, that is, X or $\neg X$, where the node can only be satisfied if X is set to 1 (true) respectively 0 (false). Boolean literals are combined into logical sentences through those inner AND and OR gates. For example, the right most OR gate represents the logical sentence C xor D.

Recently, circuits have also become the chosen target representation for tractable learners [LD08, LR13, GD13, BDC15], spurring innovation in arithmetic circuit dialects such as *sum-product networks* (SPNs) [PD11, PGD14] and *cutset networks* [RKG14]. In recent years, a large number of tractable probabilistic models have been proposed as a target representation for generative learning of a joint probability distribution: arithmetic circuits [LD08], weighted SDD [BDC15], PSDD [KAD14], cutset networks [RKG14] and sum-product networks (SPNs) [PD11]. These representa-

[1] We consider negation-normal-form circuits where no negation is allowed except at the leafs/inputs [DM02].

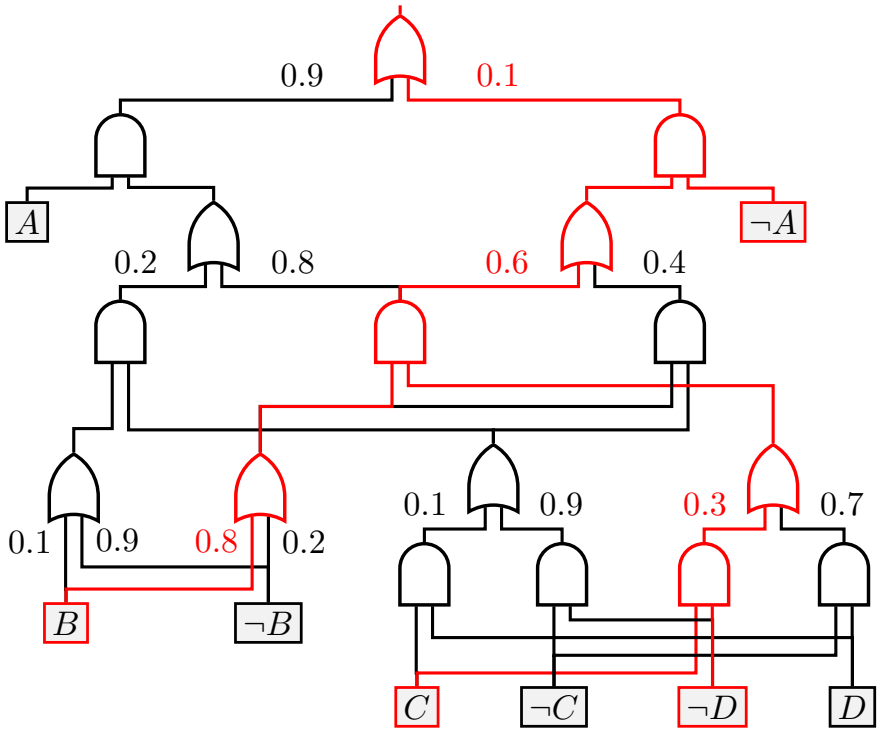


Figure 2.2: An example probabilistic circuit over 4 variables. The parameters on the hot wires are used to compute the joint probability of example $A = 0,\ B = 1,\ C = 1,\ D = 0$. The probability for this particular examples is 0.0096.

tions have various syntactic properties. Some put probabilities on terminals, others on edges. Some use logical notation (AND, OR), others use arithmetic notation ($\times$,$+$). While closely related, these representations differ significantly in the types of tractable queries and operations they support. Nevertheless, they are all circuit languages built around the properties of decomposability, and/or smoothness and/or determinism.

To summarize their similarities, we propose a simple probabilistic circuit language, where now the parameters are assumed to be normalized probabilities. For our purpose, only presenting the definition of probabilistic circuits is enough here. For a detailed survey over probabilistic circuits, we refer readers to [CVB].

Definition 1 (Probabilistic Circuit)**.** *A probabilistic circuit node n defines the following joint dis-*

tribution.

- *If n is a leaf/terminal (input) node, then* $\Pr_n(\mathbf{x}) = [\mathbf{x} \models n]$.[2]
- *If n is an AND gate with children $c_1, \ldots, c_m$, then*

$$\Pr{}_n(\mathbf{x}) = \prod_{i=1}^{m} \Pr{}_{c_i}(\mathbf{x}).$$

- *If n is an OR gate with (child node, wire parameter) inputs $(c_1, \theta_1), \ldots, (c_m, \theta_m)$, then*

$$\Pr{}_n(\mathbf{x}) = \sum_{i=1}^{m} \Pr{}_{c_i}(\mathbf{x}) \cdot \theta_i.$$

This definition has several advantages. First, it makes a clear correspondence between logical notation (AND, OR), and arithmetic notation ($\times$, $+$). Based on this definition, it is straightforward to figure out that AND gates are exchangeable with product node, and OR gates are exchangeable with sum nodes. Second, it highlights the computation graph aspect of probabilistic circuits; it is written in an induction manner starting from the bottom leaf (input) node, abstracting away the delicacies that are irrelevant to the represented joint distribution. With this definition, one can easily plug in an example and with a few steps of computation in head arrive at the probability of that example. We encourage everyone to try this process and check whether given the probabilistic circuit presented in Figure 2.2, example $A = 0, B = 1, C = 1, D = 0$ indeed has a probability of 0.96%. We will elaborate more about this computation procedure in Chapter 3 and discuss about how this procedure handles partial assignments and other more complicated situations.

This definition can also be extended to float-value inputs; the change is also straightforward. Instead of checking whether the input example $\mathbf{x}$ satisfies the leaf node (i.e., a literal), we make every leaf node represent a single-variable distribution[CVB]; for example, a univariate Gaussian distribution is a popular choice [PD11]. However, doing so will inadvertently break probabilistic circuits' inherent connection with logic and symbolic knowledge representation. Given this, we

[2]Every leaf node is a literal, which corresponds to a specific true/false assignment to a variable. For examples, Literal $\neg X$ assigns false (i.e., 0) to variable X. $\Pr_n(\mathbf{x}) = 1$ if $\mathbf{x}$ satisfies the literal node n, otherwise $\Pr_n(\mathbf{x})$ = 0.

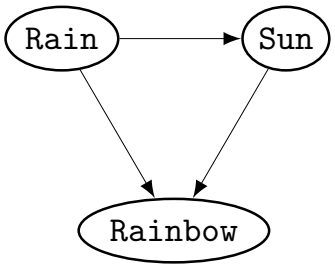
will not explore this extended definition here. There does exist an alternative interpretation about what the parameters mean in the context of logic when the inputs are float values, and we will discuss this alternative interpretation in Chapter 5.

Still, this definition does not directly expose the important syntactic properties (i.e., decomposability, smoothness, and determinism) that guarantee probabilistic circuits' tractability. To properly introduce them in detail, in the next section we specifically target one dialect of probabilistic circuits, called probabilistic sentential decision diagrams (PSDDs), and use this specific representation to take a deep dive into properties.

2.3 A Specific Tractable Representation

We choose probabilistic sentential decision diagrams (PSDDs) [KAD14] out of two considerations. First, PSDDs make use of all three properties mentioned above (i.e., decomposability, smoothness, and determinism) and hence are the perfect candidate to illustrate what those properties are and how they impact a representation. Second, owing to those intricate structural properties, PPSDDs are perhaps the most powerful circuit proposed to date. PSDDs support closed-form parameter learning, MAP inference, complex queries [BDC15], and even efficient multiplication of distributions [SCD16], which are all instrumental to fundamentally understanding complex data yet are all increasingly rare.

There exist some syntactic and semantic differences between PSDDs and the earlier probabilistic circuit definition. Some of them are trivial and can be reconciled with straightforward transformation. We will also go through the transformation steps when we encounter the differences in the following discussion. Figure 2.3c shows an example probabilistic sentential decision diagram (PSDD). As the figure shows, similar to many probabilistic graphical models, a PSDD is a parameterized directed acyclic graph (DAG). Each inner node is either a logical AND gate with two inputs, or a logical OR gate with an arbitrary number of inputs. The types of nodes usually alternate from layer to layer, but it is not a requirement. Each terminal (input) node is a univariate



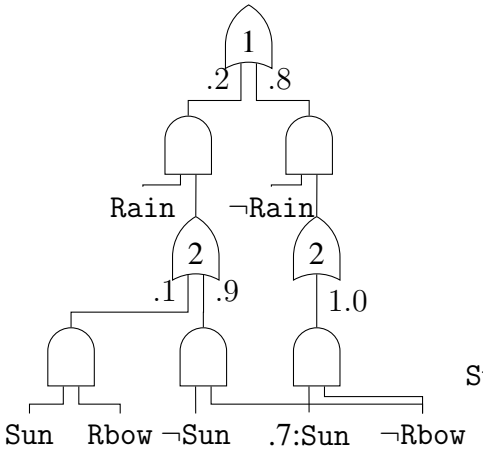
(a) Bayes net

$$\Pr(\texttt{Rain}) = 0.2,$$

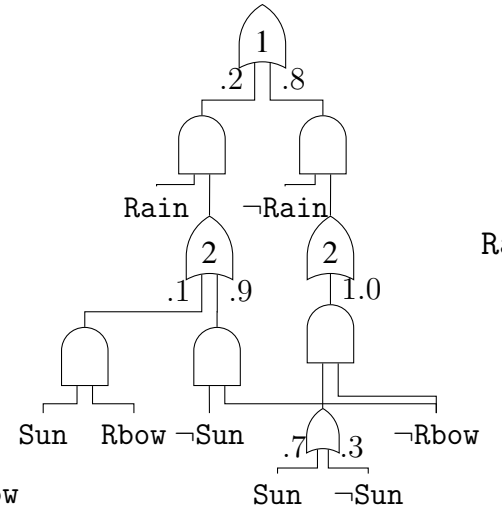
$$\Pr(\texttt{Sun} \mid \texttt{Rain}) = \begin{cases} 0.1 \text{ if } \texttt{Rain} \\ 0.7 \text{ if } \neg\texttt{Rain} \end{cases}$$

$$\Pr(\texttt{Rbow} \mid \texttt{R}, \texttt{S}) = \begin{cases} 1 \text{ if } \texttt{Rain} \wedge \texttt{Sun} \\ 0 \text{ otherwise} \end{cases}$$

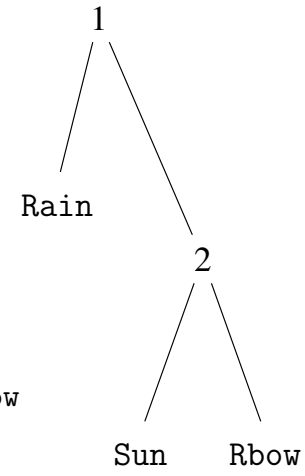
(b) Conditional probabilities



(c) Equivalent PSDD circuit



(d) In probabilistic circuit form



(e) PSDD's vtree

Figure 2.3: A Bayesian network and its equivalent PSDD.

distribution, which could either be X when X is always true, $\neg X$ when it is always false, or $(\theta : X)$ when it is true with probability θ. A decision node is the combination of an OR gate with its AND gate inputs. We refer to the left input of an AND gate as its prime (denoted p) and the right one as its sub (denoted s). The n wires in each decision node are annotated with a normalized probability distribution $\theta_1, \ldots, \theta_n$. Alternatively, we refer to a decision node's labeled AND gates as its elements and represent the decision node itself as a set of elements $\{(p_1, s_1, \theta_1), \ldots, (p_n, s_n, \theta_n)\}$. One may immediately notice two differences from this description and Definition 1. To start from the simpler one, PSDDs require AND gates to always have two child inputs, whereas probabilistic circuits do not impose restrictions on the number of children an AND gate can have. This restriction is largely inherited from the knowledge compilation tradition adopted by sentential decision diagrams (SDDs). When representing a probabilistic distribution, PSDDs are not affected by this in terms of expressiveness. An AND gate with multiple children can be simulated by several AND gates with two children stacked together. Speaking of the trickier difference, PSDDs' terminal nodes represent a univariate distribution instead of a particular assignment to the corresponding variable. Yet, this difference can also be reconciled by adding an additional OR gate. When the leaf node is either X or $\neg X$, no transformation is required, as the meaning is consistent with our definition 1. When it is $(\theta : X)$, as illustrated in Figure 2.3d, we can replace it with an OR gate. This OR gate has two child inputs, one representing the true assignment X and the other the false assignment $\neg X$. Their corresponding wires are assigned with the parameters θ and $1 - \theta$ respectively. Now this new OR gate represents a univariate distribution, identical to the original terminal node $(\theta : X)$.

2.3.1 Syntactic Properties

According to the semantics we will detail later, each PSDD node represents a probability distribution over the random variables that appear below it. To identify what random variables a PSDD node is defined with respect to, we need a *variable tree (vtree)*, which is omitted in the probabilistic circuit definition. A vtree is a full binary tree, whose leaves are labeled with variables;

see Figure 2.3e. Besides identifying random variables, it is also critical to enforce the *decomposability* property every AND gate must respect. An AND gate is *decomposable*, meaning that its inputs represent a distribution over disjoint sets of variables. The internal vtree nodes split variables into those appearing in the left subtree $\mathbf{X}$ and those in the right subtree $\mathbf{Y}$. This implies that the corresponding PSDD decision nodes (i.e., OR gates) must have primes (i.e., left input nodes) ranging over $\mathbf{X}$ and subs (i.e., right input nodes) over $\mathbf{Y}$. We say the corresponding PSDD nodes are *normalized* for the vtree node. Figure 2.3c labels decision nodes with the vtree node they are normalized for.

Each decision node must be *deterministic*, meaning that for any single possible world (i.e., an joint assignment over all considered variables), it can have at most one prime assign a non-zero probability to that world. In other words, the supports of all distributions represented by primes must be disjoint within the same decision node. We further assume that all elements assign a non-zero probability to at least one world.[3]

2.3.2 Semantics and Interpretability

Each PSDD node represents a probability distribution, starting with the terminal nodes' univariate distributions. This inductive property is clearly demonstrated in our probabilistic circuit definition (Definition 1) as well: an AND gate (i.e., product node) combines two context-specific independent distributions over two disjoint sets of variables, and an OR gate (i.e., sum node) constructs a mixture of distributions over the same set of variables. To be more specific, each decision node n (i.e., an OR gate) normalized for a vtree node with $\mathbf{X}$ and $\mathbf{Y}$ in its left and right subtrees respectively, represents a distribution over $\mathbf{XY}$ as $\Pr_n(\mathbf{XY}) = \sum_i \theta_i \Pr_{p_i}(\mathbf{X}) \Pr_{s_i}(\mathbf{Y})$. Note an OR gate is always over the same set of variables as every one of its child input node. Under these semantics, the PSDD in Figure 2.3c represents the same distribution as the Bayesian network in Figure 2.3a.

[3] In the original definition this was not required. In fact, primes were required to be exhaustive, which can necessitate a zero-probability element [KAD14]. This is an artifact from defining PSDD as an extension of SDDs, which require exhaustiveness to support negation or disjunction. These logical operations are not used for our (probabilistic) purpose.

Each PSDD node's distribution has an intricate support over which it defines a non-zero probability. We refer to this support as the *base* of node n, written $[n]$. The base of a node can alternatively be defined as a logical sentence using the recursion $[n] = \bigvee_i [p_i] \wedge [s_i]$, where $[X] = X$, $[\neg X] = \neg X$, and $[\theta : X] = \text{true}$.

From a top-down perspective, a decision node presents a choice between its prime bases $[p_i]$: at most one is true in each world. Thus, the PSDD is a decision diagram branching on which sentence $[p_i]$ is true. This generalizes decision trees or binary decision diagrams which only branch on the value of a single variable. To reach node n, all the primes on a path to n must be satisfied; they are the sub-context of n. The disjunction of all n's sub-contexts is its *context* γ_n. This notion lets us precisely characterize PSDD parameter semantics and make each parameter interpretable: they are conditional probabilities in root node r's distribution:

$$\theta_i = \Pr_r([p_i] \mid \gamma_n).$$

Note this also means the independence relationship assumed between $\mathbf{X}$ and $\mathbf{Y}$ (such that we can directly multiply the two's distribution together) for every AND gate is indeed context-specific independence. Context-specific independence is a more refined concept than conditional independence: conditional independence requires $\mathbf{X}$ to be independent from $\mathbf{Y}$ given all possible assignments to $\mathbf{Z}$, whereas context-specific independence only requires $\mathbf{X}$ to be independent from $\mathbf{Y}$ given a particular assignment to $\mathbf{Z}$ (i.e., a context).

2.3.3 Distributions in the Presence of Logical Constraints

PSDDs are the probabilistic extensions to sentential decision diagrams (SDDs), which are a target representation to compactly represent logical constraints. Given this, PSDDs can naturally represent complex probabilistic distributions in structured spaces that are subject to complex logical constraints, disallowing large numbers of possible worlds [KVC14]. In this context, knowledge compilation algorithms can build PSDD structures without looking at the data; the structure is obtained when compiling the constraints as a SDD. To provide a concrete example of a struc-

tured space, consider the Boolean variables A_{ij} for $i, j \in \{1, ..., n\}$. Here, i represents an item and j represents its position in a total ranking of n items. The unstructured space consists of 2^{n^2} possible worlds of the n^2 Boolean variables. The structured space of interest only consists of the subset that corresponds to total rankings over n items. The size of this structured space is only $n!$ as the remaining assignments do not correspond to valid, total rankings (e.g., an assignment that places two items in the same position, or one item in two different positions) [CVD15a]. When structures are large enough, parameter estimation is shown sufficient to learn distributions over game traces [CTD16], configurations, and yield state-of-the-art results learning preference distributions [CVD15b].

CHAPTER 3

Tractable Reasoning with Probabilistic Circuits

This chapter addresses this first big question in this dissertation — how to do exact probabilistic reasoning about distributions in polynomial time using probabilistic circuits.

3.1 Background

Tractability is a vague concept; to make it concrete, one has to specify for which specific type of probabilistic reasoning a probabilistic representation is tractable. In practice, when a representation is tractable for a reasoning query, it means the exact answer to that query can be computed in polynomial time complexity with respect to the the size of the representation (which is often measured in terms of the number of parameters, nodes or edges). This also means a representation can be tractable for one type of probabilistic reasoning query, but not for another. However, one almost always pursues a probabilistic representation that supports the most number of tractable probabilistic reasoning queries, as this gives users a certificate to use this representation flexibly for assorted applications without worrying about the computational overhead.

As mentioned in the last chapter, probabilistic circuits are perhaps the most powerful tractable probabilistic representations proposed to date, as they support the most number of tractable probabilistic reasoning queries. Because of this, probabilistic circuits are commonly used as the compilation targets of intractable graphical models[SCD16]. Given an arbitrary graphical model (e.g., a Bayesian network or a Markov network), one can first construct a probabilistic circuit that represents the same joint distribution as the given graphical model. One usually refers to this step

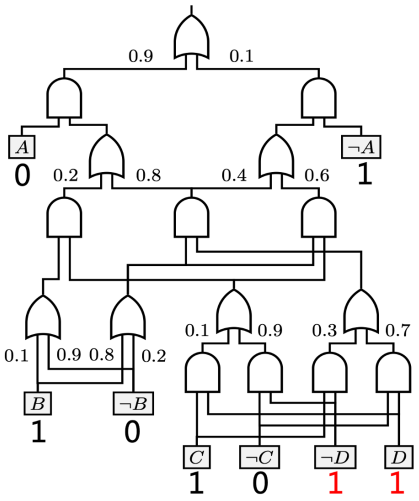
as compilation. After the probabilistic circuit is obtained, probabilistic queries that are originally NP-hard or even #P-hard to compute for the intractable graphical model, can now be answered exactly in polynomial time. Note this does not mean we have found a systematic solution to reduce NP-hard problems to the P space, as the compilation step itself can be NP-hard.

To demonstrate that probabilistic circuits are indeed one of the most powerful tractable representations, in the following sections, we start with reviewing some most commonly used probabilistic reasoning queries and watching in action with examples how probabilistic circuits compute them. We do not intend to be exhaustive here. For readers who are interested in learning more details about all the probabilistic reasoning queries that have been proved to be tractable for probabilistic circuits, we kindly refer them to articles [KAD14, SCD16, CSD17, CVB]. After this gentle warm-up, we switch the gear and explore the first study of probabilistic circuits' tractability property in information measure, a new frontier of tractable reasoning.

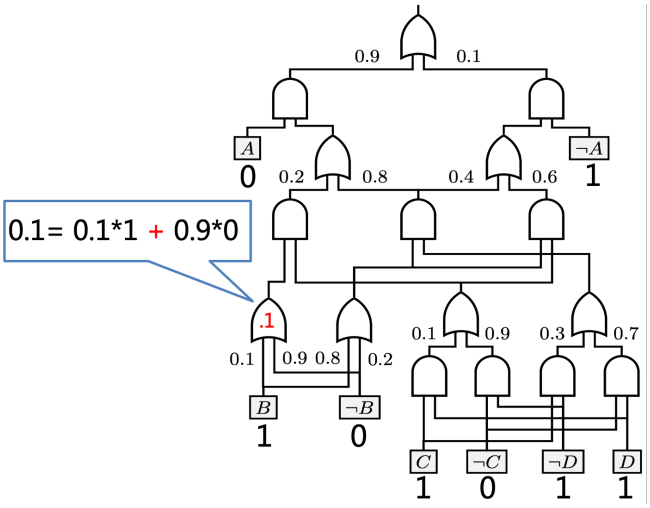
3.2 A Straightforward Bottom-Up Pass for Query Computation

In this section, we demonstrate the procedure that computes the most commonly used probabilistic queries, namely, joint, marginal, conditional probabilities, and most probable explanation (MPE). To efficiently compute each query, we need to first convert a probabilistic circuit to a computation graph. Recall that our definition of probabilistic circuits (i.e., Definition 1) essentially already summarizes how this computation graph works; it is a bottom-up evaluation pass starting from the leaf (input) nodes and the output of the root node tells us the computed query result. For the 4 types of probabilistic queries mentioned above, their time complexity is linear in terms of the size of the given probabilistic circuits.

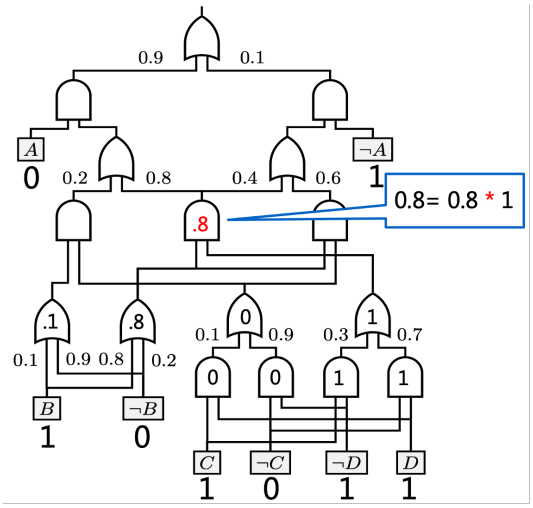
The only slightly tricky and less obvious part in Definition 1 is the output values of the leaf (input) nodes, especially when the input is a partial assignment to the considered variables. The general rule is that the output of a leaf (input) node is 1, if the literal this leaf node represents is consistent with the (partial) assignment; otherwise, the output of that leaf (input) node is 0. In



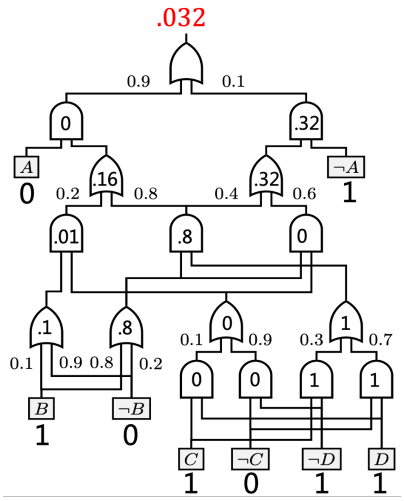
(a) The output values of leaf nodes. The leaf nodes whose variables do not appear in the partial assignment are color in red.



(b) The output values of OR gates are computed by summing the output values of their input nodes. This OR-to-summation transformation is colored in red.



(c) The output values of AND gates are computed by multiplying the output values of their input nodes. This AND-to-multiplication transformation is colored in red.



(d) The output value of the root node (colored in red), which is the exact answer to our particular marginal query.

Figure 3.1: An illustration of the bottom-up procedure to compute the marginal probability for the partial assignment $A = 0, B = 1, C = 1$. The probability for this particular query is 0.032.

the following, we will observe in action with examples about this and how the computation graph works after setting the output values of the leaf (input) nodes. Note in this section, we focus on the computation aspect of probabilistic circuits. Proving this computation procedure is correct involves a lot more work and needs to leverage some other properties not introduced in this dissertation. We do not tend to go into detail about this and refer the readers with interest to [KAD14].

3.2.1 Joint, Marginal and Conditional Probability

Joint probabilities are the most straightforward ones to compute, as we are always given a complete assignment to all considered variables. The leaf (input) nodes' output values equal the corresponding variables' values in the assignment, if the leaf nodes represent a true literal (i.e., X); the leaf nodes' output values equal one minus the corresponding variables' values in the assignment, if the leaf nodes represent a false literal (i.e., $\neg X$).

When asked to compute marginal probabilities, we are given a partial assignment, which means some considered variables have assignments and others do not. Given the partial assignments reflect what we can observe, they are also commonly called as evidence. For the leaf nodes whose variables appear in the partial assignments, their output values are computed the same as if we were given a complete assignment. For the leaf nodes whose variables do not appear in the partial assignments, their output values are always one. An intuitive explanation for this is that, since we do not observe any evidence (i.e., partial assignments) that refutes the literal situations represented by those leaf nodes, they are always consistent and hence their output values equal one. After obtaining the output values of leaf nodes, one just needs to straightforwardly follow the AND-to-multiplication, and OR-to-summation transformation to finish the computation. The whole process is also illustrated in Figure 3.1.

One can further easily compute conditional probabilities by first computing the marginals:

$$\Pr(\mathbf{x} \mid \mathbf{e}) = \frac{\Pr(\mathbf{x} \cup \mathbf{e})}{\Pr(\mathbf{e})}.$$

Note both the numerator and the denominator are marginal probabilities, computing a conditional

probability is as tractable as computing a marginal probability, except now we need to run this bottom-up evaluation procedure twice.

3.2.2 Most Probable Explanation

Most Probable Explanation (MPE) is a query that computes the most likely complete assignment that is consistent with the observed evidence. To compute MPE using probabilistic circuit involves one different transformation step compared to the computation of joint, marginal and conditional probabilities. To be specific, OR gates are transformed into weighted max nodes. In other words, the output values of an OR gate corresponds to the maximum product of the output values of its input nodes and the associated wire parameters. Other aspects of the bottom-up computation procedure remain the same. After having evaluated the probability of the MPE, we can obtain the MPE assignment by reversely traversing the probabilistic circuit. For each OR gate, one follows the input wire which results in the weighted max. For each AND gate, one follows all of the input wires. The traversal stops when it hits leaf nodes. The concatenation of literals in the leaf nodes is the MPE assignment. Figure 3.2 illustrates this process under the observed evidence $A = 1$. The computed MPE probability is 0.4032 and the computed MPE assignment is $A = 1, B = 1, C = 0, D = 1$.

3.3 More Complex Reasoning: A Case Study of Intersectional Divergence

The previous section only considers probabilistic reasoning queries with respect to one single probabilistic circuit. In this section instead, we advance the frontier and study probabilistic reasoning queries with respect to two probabilistic circuits. In particular, we study the question about how to compare two probabilistic circuits' represented distributions.

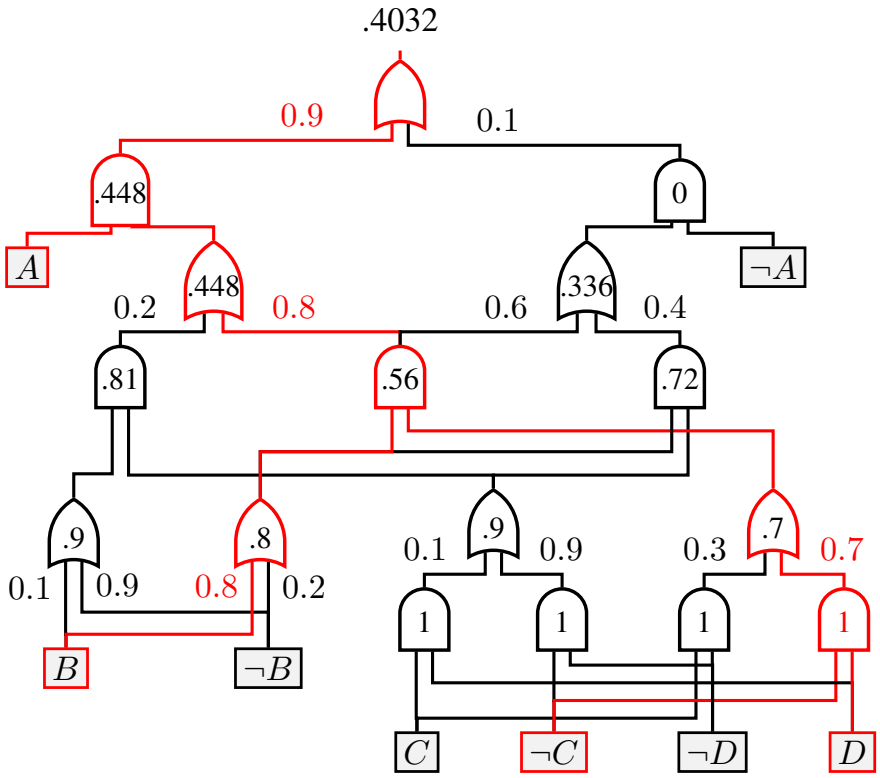
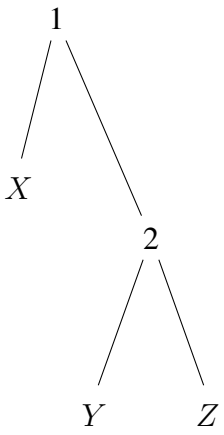


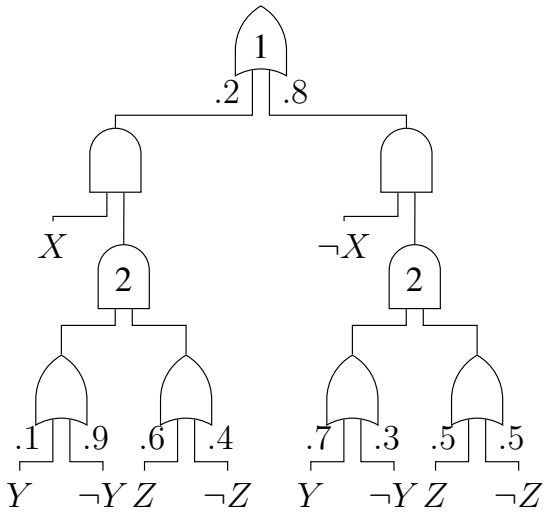
Figure 3.2: An illustration of maximum probable explanation (MPE) query under the evidence $A = 1$. Each OR gate is transformed into a weighted-maximum computation node. The hot wires are visited in the reverse traversal that obtains the MPE assignment.

3.3.1 Intersectional Divergence

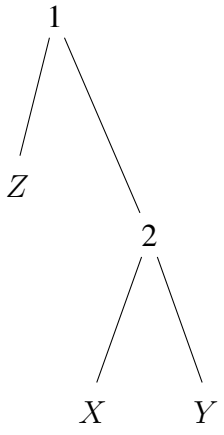
Inspired by the Kullback-Leibler divergence (KL-divergence) between two distributions, we aim to calculate the divergence between two probabilistic circuits. As the two probabilistic circuit may not share the same base or even the same set of variables, we have to impose more restrictions for the divergence to be well defined. To be specific, the new proposed divergence is only calculated with respect to the joint assignments that have non-zero probabilities induced by both probabilistic circuits, otherwise the inherent logarithm operation carries no mathematical merit. Intuitively, we can consider it as the KL-divergence computed only on the intersection between the distribution supports of the two probabilistic circuits. Given this intuition, we call it *intersectional divergence*.



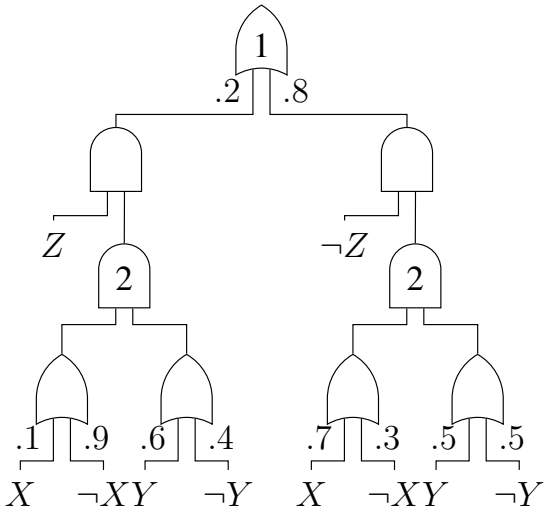
(a) A vtree with variable order X, Y and Z.



(b) A probabilistic circuit with variable order X, Y, Z.



(c) A vtree with variable order Z, X and Y.



(d) A probabilistic circuit with variable order Z, X, Y.

Figure 3.3: An illustration of two probabilistic circuits over the same set of variables but are normalized with respect to different vtrees.

Definition 2. *(Intersectional Divergence D_I) The intersectional divergence between two distributions p and q is defined as the following.*

$$D_I(p \parallel q) \stackrel{def}{=} \sum_{\mathbf{x} \models [p] \wedge [q]} p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})}$$

Here, $[p]$ represents the *base* of the distribution p. Note that this definition is independent from probabilistic circuits and hence can also be used between any two distributions. For distributions with the same base, intersectional divergence is equivalent to KL-divergence. Note that $D_I(p \parallel q) = D_{KL}(p \parallel q)$ when $[p] \models [q]$ and $[p] \equiv [q]$ in particular. In general, the relation between intersectional divergence and KL divergence is as follows.

Corollary 1. *The general relation between intersectional divergence and Kullback-Leibler divergence (KL-Divergence) is as follows. Note that this relation is generally applicable and is not restricted to distributions represented by probabilistic circuits.*

$$D_{KL}\left(p(.|[q]) \parallel q\right) = \frac{D_I(p \parallel q)}{p([q])} - \log p([q])$$

$$D_{KL}\left(p(.|[q]) \parallel q(.|[p])\right) = \frac{D_I(p \parallel q)}{p([q])} - \log \frac{q([p])}{p([q])}$$

From this corollary, it is obvious that the computation of intersectional divergence between a pair of arbitrary distributions over the same set of variables cannot be tractable. Then what about between a pair of arbitrary probabilistic circuits? It turns out that vtree plays a critical role here. To get some valuable intuition and insights, let us consider the two example probabilistic circuits in Figure 3.3. Both probabilistic circuits represent a joint distribution over the same three variables, X, Y, and Z. Yet, they are normalized for different vtrees and hence having different variable orderings. Because of this, starting from the roots' input nodes and downwards, the two probabilistic circuits' intermediate nodes do not represent distributions over the same set of variables. Intersectional divergence is obviously and should be non-applicable between those intermediate nodes. This means that the computation of intersectional divergence between these two example

probabilistic circuits cannot be decomposed into smaller cases using the intermediate results from non-root nodes. And on the root level, to do the computation, one has to enumerate all possible joint assignments. Given the number of possible joint assignments is exponential to the number of considered variables, this computation process cannot be tractable.

3.3.2 A Quadratic Recursion Algorithm

As demonstrated earlier, the intersectional divergence definition itself does not necessarily help us compute it efficiently. From our observation of computing intersectional divergence between two probabilistic circuits respecting different vtrees, we notice that this computation can only become tractable if we can "break it down" to the leaf nodes, where it reduces to trivial computation. As shown in the following derivation steps, two probabilistic circuits sharing the same vtree is indeed the key property that enables a recursive decomposition, as it ensures that the pairs of nodes considered by the algorithm depend on exactly the same set of variables.

Theorem 1. *Given two probabilistic circuits which are normalized with respect to the same vtree* $m : \{(p_1, s_1, \theta_1), (p_2, s_2, \theta_2)...(p_k, s_k, \theta_k)\}, n : \{(r_1, t_1, \beta_1), (r_2, t_2, \beta_2)...(r_l, t_l, \beta_l)\}$*, the intersectional divergence between* m *and* n *can be computed efficiently in a recursive manner:*

$$D_I(m \parallel n) = \sum_{i,j} s_i([t_j])p_i([r_j])D_{KL}(\theta_i \parallel \beta_j) + \theta_i s_i([t_j])D_I(p_i \parallel r_j) + \theta_i p_i([r_j])D_I(s_i \parallel t_j)$$

where $D_{KL}(\theta_i \parallel \beta_j) = \theta_i \log \frac{\theta_i}{\beta_j}$.

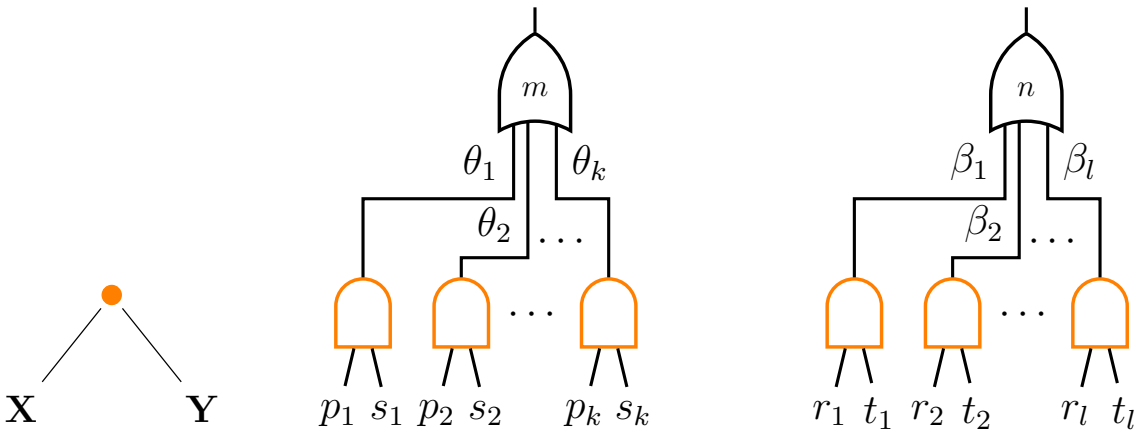
Proof.

$$
\begin{aligned}
D_I(m \parallel n) &\overset{\text{def}}{=} \sum_{\mathbf{xy} \models [m] \wedge [n]} m(\mathbf{xy}) \log \frac{m(\mathbf{xy})}{n(\mathbf{xy})} \\
&= \sum_{i,j} \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} \sum_{\mathbf{y} \models [s_i] \wedge [t_j]} p_i(\mathbf{x}) s_i(\mathbf{y}) \theta_i \log \frac{p_i(\mathbf{x}) s_i(\mathbf{y}) \theta_i}{r_j(\mathbf{x}) t_j(\mathbf{y}) \beta_j} \\
&= \sum_{i,j} \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} \sum_{\mathbf{y} \models [s_i] \wedge [t_j]} p_i(\mathbf{x}) s_i(\mathbf{y}) \theta_i \left\{ \log \frac{p_i(\mathbf{x}) \theta_i}{r_j(\mathbf{x}) \beta_j} + \log \frac{s_i(\mathbf{y})}{t_j(\mathbf{y})} \right\} \\
&= \sum_{i,j} \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} p_i(\mathbf{x}) \theta_i \left\{ \log \frac{p_i(\mathbf{x}) \theta_i}{r_j(\mathbf{x}) \beta_j} \sum_{\mathbf{y} \models [s_i] \wedge [t_j]} s_i(\mathbf{y}) \;\; + \sum_{\mathbf{y} \models [s_i] \wedge [t_j]} s_i(\mathbf{y}) \log \frac{s_i(\mathbf{y})}{t_j(\mathbf{y})} \right\} \\
&= \sum_{i,j} \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} p_i(\mathbf{x}) \theta_i \left\{ \log \frac{p_i(\mathbf{x}) \theta_i}{r_j(\mathbf{x}) \beta_i} s_i([t_j]) + D_I(s_i \parallel t_j) \right\} \\
&= \sum_{i,j} \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} p_i(\mathbf{x}) \theta_i s_i([t_j]) \left\{ \log \frac{\theta_i}{\beta_j} + \log \frac{p_i(\mathbf{x})}{r_j(\mathbf{x})} \right\} + p_i(\mathbf{x}) \theta_i D_I(s_i \parallel t_j) \\
&= \sum_{i,j} \theta_i s_i([t_j]) \left\{ \log \frac{\theta_i}{\beta_j} \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} p_i(\mathbf{x}) + \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} p_i(\mathbf{x}) \log \frac{p_i(\mathbf{x})}{r_j(\mathbf{x})} \right\} \\
&\quad + \theta_i D_I(s_i \parallel t_j) \sum_{\mathbf{x} \models [p_i] \wedge [r_j]} p_i(\mathbf{x}) \\
&= \sum_{i,j} \theta_i s_i([t_j]) \left\{ \log \frac{\theta_i}{\beta_j} p_i([r_j]) + D_I(p_i \parallel r_j) \right\} + \theta_i D_I(s_i \parallel t_j) p_i([r_j]) \\
&= \sum_{i,j} s_i([t_j]) \theta_i \log \frac{\theta_i}{\beta_j} p_i([r_j]) + \theta_i s_i([t_j]) D_I(p_i \parallel r_j) + \theta_i D_I(s_i \parallel t_j) p_i([r_j]) \\
&= \sum_{i,j} s_i([t_j]) p_i([r_j]) D_{KL}(\theta_i \parallel \beta_j) + \theta_i s_i([t_j]) D_I(p_i \parallel r_j) + \theta_i p_i([r_j]) D_I(s_i \parallel t_j)
\end{aligned}
$$

□

Corollary 2. *Let m and n be the root nodes of two different probabilistic circuits with the same vtree. Let e_m and e_n be their respective number of edges. Then, the intersectional divergence between the distributions captured by n and m respectively can be computed exactly in time complexity of $O(e_m e_n)$.*

This decomposition process is also visualized in Figure 3.4. To make sure this recursion al-



(a) A vtree.

(b) Two different probabilistic circuits.

Figure 3.4: An illustration of two probabilistic circuits that are normalized with respect to the same vtree. AND gates are colored in orange as the vtree node they correspond to.

gorithm can run in linear time complexity, we need to cache many intermediate results and this process is formalized in Algorithm 1.

3.4 Parallel Computation

A linked node representation is an intuitive data structure for circuits. However, it has the drawback that it makes computations sparse, making it harder to leverage parallelism to speed up computation. To optimize performance during inference and learning, we translate the circuit's DAG into a layered computational graph, starting with the input layer. Each layer only depends on the previous layers. Since the computations on the nodes in the same layer can be cached in one large vector, we can simultaneously parallelize our computation over them on the one hand, and training examples or inference task data on the other hand. Additionally, we can build customized kernels to accelerate computation on both CPUs and GPUs (using SIMD and CUDA kernels respectively). Experiments show that CPU parallelism gives significant speed-ups, which even become an order of magnitude faster with GPU parallelism, all using the same underlying data structures [DKL21].

Algorithm 1: `intersectional-divergence`(m, n)

input: Probabilistic circuits m and n that respect the same vtree.

output: Intersectional divergence $D_I(m, n)$

note: `pr-constraint`$(a, [b])$ is the probability of $[b]$ in PSDD a's induced distribution [CVD15b].

main:

if $(m, n) \in$ in `cache` **then**
  return `cache`$[(m, n)]$
else
  $\rho \leftarrow 0$
  for each element (p_i, s_i, θ_i) in PSDD m **do**
    for each element (r_j, t_j, β_j) in PSDD n **do**
      $\rho_{11} \leftarrow$ `pr-constraint`$(s_i, [t_j])$
      $\rho_{12} \leftarrow$ `pr-constraint`$(p_i, [r_j])$
      $\rho_{13} \leftarrow \theta_i \log \frac{\theta_i}{\beta_j}$
      $\rho_{21} \leftarrow$ `intersectional-divergence`(p_i, r_j)
      $\rho_{31} \leftarrow$ `intersectional-divergence`(s_i, t_j)
      $\rho \leftarrow \rho + \rho_{11}\rho_{12}\rho_{13} + \theta_i\rho_{11}\rho_{21} + \theta_i\rho_{12}\rho_{31}$
    end for
  end for
  `cache`$[(m, n)] \leftarrow \rho$
  return ρ
end if

3.5 Discussion

We are not the only ones to consider advancing the frontier of tractable reasoning of probabilistic circuits by leveraging the decomposition process. Multiplication of two probabilistic circuits can also only be tractable, if the two share the same vtree. A similar "break it down" recursion algorithm is proposed to support efficient multiplication [SCD16]. In Chapter 5, we will present how probabilistic circuits can be converted into a classifier or a regressor. Sharing the same vtree will be proved to be critical again in guaranteeing efficient probabilistic reasoning between a probabilistic circuit and a classification/regression circuit [KCL19].

CHAPTER 4

Structure Learning of Probabilistic Circuits

In the last chapter, we have demonstrated probabilistic circuits' ability to support assorted complex probabilistic reasoning queries. However, reasoning can only be done with respect to the probabilistic distribution represented by the given probabilistic circuit. This means the quality of the computed answers to those queries largely depends on how accurately the given probabilistic circuit represents the true data distribution. This chapter addresses this second big challenge in this dissertation by introducing our proposed method to effectively learn a probabilistic circuit from data.

4.1 Background

Tractable learning aims to induce complex, yet tractable probability distributions from data [MV16]. The learned tractable model serves as a certificate to the user that any query that arises can always be answered efficiently. And when the learning algorithm is competitive, the learned model can indeed represent a distribution that is close to the true data distribution. This means the answers to the assorted reasoning queries can indeed help users analyse and fundamentally understand the data. Efforts in tractable learning have achieved great success inducing complex joint distributions from data without constraints, while guaranteeing efficient exact probabilistic inference; for instance, by learning arithmetic circuits (ACs) or sum-product networks (SPNs). As introduced at the end of Chapter 2, knowledge compilation algorithms can build probabilistic circuit structures that are consistent with the logical constraints without looking at the data. In other words, in all prior probabilistic circuit applications that are subject to logical constraints, the learner is given a

logical sentence α that encodes domain knowledge (e.g., a constraint encoding rankings or game traces [CTD16]). Using knowledge compilation, sentence α is first transformed into an logical circuit, and second into a probabilistic circuit by parameter learning. Prior work does not perform data-oriented structure learning: no data is used to come up with PSDD structures.

These observations raise two questions: (i) are probabilistic circuits amenable to tractable learning when no logical constraints or compiled circuit are available a priori, and (ii) can we still learn probabilistic circuits that are subject to logical constraints while also fitting the data well; that is, perform true structure learning? To answer both questions, we target the most powerful dialect of probabilistic circuits, PSDDs, and develop LEARNPSDD, which is the first structure learning algorithm for PSDDs. It uses local operations on the PSDD circuit that maintain the desired circuit properties, while steadily increasing model fit. LEARNPSDD is supported by a vtree learning algorithm that captures the data's independencies in a tree structure, which we empirically show to be an essential step of the learning process. Moreover, using expectation maximization on top of LEARNPSDD, we show competitive results on the standard tractable learning benchmarks. When additionally performing bagging, our PSDD learner reports state-of-the-art results on six datasets. Finally, the proposed algorithm is general and retains the ability to learn in logically constrained probability spaces. Here, we empirically show that LEARNPSDD is able to refine the circuits compiled from constraints, yielding superior likelihood scores.

4.2 Parameter Learning

Before tackling the hard question of structure learning, let us first review how PSDDs' parameters are learned from data when a structure is given. Thanks to PSDDs' syntactic properties, the maximum-likelihood estimate for each PSDD parameter is calculated in closed form by observing the fraction of complete examples flowing through the relevant wire. More precisely, out of all the examples that agree with the node context γ_n, the parameter estimate is the fraction of examples

that also agrees with the prime base $[p_i]$ [KAD14]:

$$\hat{\theta}_i = \frac{\mathcal{D}\#(\gamma_n, [p_i])}{\mathcal{D}\#(\gamma_n)}. \tag{4.1}$$

To prevent overfitting, Laplace smoothing is used.

This maximum-likelihood guarantee gives us a great opportunity to tackle the structure learning problem, as now it is straightforward to compare which structure is better. Without this guarantee, we can easily fall into the trap that a competitive structure plus an ill-fitted set of parameters still only induces a distribution that is further away from the data than an ill-fitted structure plus a competitive set of parameters.

4.3 Vtree Learning

To learn a vtree from data, it is important to understand the assumptions that are implied by a choice of vtree. PSDDs recursively decompose the distribution by conditioning it on the prime bases $[p_i]$. Specifically, each decision node decomposes the distribution into independent distributions over $\mathbf{X}$ and $\mathbf{Y}$, guided by the vtree.

Proposition 1. *[KAD14] Prime and sub variables are independent in PSDD n, given a prime base:*

$$\begin{aligned}\Pr_n(\mathbf{XY} \mid [p_i]) &= \Pr_n(\mathbf{X} \mid [p_i]) \Pr_n(\mathbf{Y} \mid [p_i]) \\ &= \Pr_{p_i}(\mathbf{X}) \Pr_{s_i}(\mathbf{Y}).\end{aligned}$$

Independence given a logical sentence is called *context-specific independence* [BFG96]. Which context-specific independencies can be exploited, as specified by the vtree, has a crucial impact on PSDD size.

Prior work always obtains its vtree from compiling logical constraints into SDD circuits [CD13], disregarding the dependencies that are implied by this choice. We propose a novel method that does induce vtrees based on the independencies found in the data.

A common way to quantify the level of independence between two sets of variables is their mutual information:

$$\text{MI}(\mathbf{X}, \mathbf{Y}) = \sum_{\mathbf{x}} \sum_{\mathbf{y}} \Pr(\mathbf{xy}) \log \frac{\Pr(\mathbf{xy})}{\Pr(\mathbf{x}) \Pr(\mathbf{y})}.$$

Intuitively, low mutual information suggests that $\mathbf{X}$ and $\mathbf{Y}$ are almost independent, and that the data distribution can be approximated by a PSDD that satisfies Proposition 1 using only a small number of primes in each decision node. Therefore, we let mutual information guide the learner: our objective is to induce a vtree that minimizes the mutual information between the $\mathbf{X}$ and $\mathbf{Y}$ variables as they are split in each internal vtree node. Additionally, we will aim to balance the vtrees. We observe that this tends to produce smaller PSDDs in practice.

However, estimating mutual information between large $\mathbf{X}$ and $\mathbf{Y}$ requires estimating an exponential number of terms $\Pr(\mathbf{xy})$, each of which is hard to estimate accurately from data. Therefore, we approximate mutual information by average pairwise mutual information:

$$\text{pMI}(\mathbf{X}, \mathbf{Y}) = \text{avg}_{X \in \mathbf{X}, Y \in \mathbf{Y}} \text{ MI}(\{X\}, \{Y\}).$$

We present two algorithms for optimizing a vtree's pMI.

Top-down vtree induction starts with the full variable set and recursively finds splits. Every step divides the variables into two equally-sized subsets with minimal pMI. Finding splits is reduced to a balanced min-cut problem, for which optimized solvers exist [Kar13].

Bottom-up vtree induction starts with singleton sets of variables at the bottom of the vtree. For each level of the vtree, it pairs two vtrees of the level below, maximizing the pMI of the pairs, in order to minimize the pMI of future pairings at higher levels. Finding pairings of vtrees reduces to the minimum-cost perfect matching problem, for which optimized solvers exist [Kol09].

Both methods greedily solve the same problem. The difference lies in the direction of the greedy optimization. Top-down induction begins at the root and will therefore get the best splits at the higher levels. Bottom-up starts from the leaves and will therefore get the best pairings at the lower levels. Section 4.7 will present an empirical comparison showing that bottom-up induction

outperforms the top-down approach. Intuitively, most interactions occur between small numbers of variables, which makes the lower levels of the vtree more important.

4.4 Structure Learning Algorithm

This section presents the first algorithm to learn PSDD structure from data. The objective is to obtain a compact structure that approximates the data distribution well.

We propose three operations, split, clone, and merge, that change the PSDD structure while keeping the PSDD syntactically sound and the base of the root node unaltered. The soundness criteria guarantee that the learned PSDD follows the syntactic definitions described in Chapter 2. Not changing the root node's base guarantees that any constraint (i.e., domain knowledge) that is encoded in the PSDD remains intact. Our learner applies these operations greedily to optimize a score function.

4.4.1 Structure Change Operations

Three operations are introduced to change the PSDD structure to represent a different distribution over the same base. Those three can be roughly divided into two groups. The first two, split and clone, belong to one group, as they both expand the PSDD structure by introducing more nodes and their associated wires; merge forms its own group, as in contrast, it shrinks the PSDD structure by deleting a sub-part rooted at some intermediate decision node and rewiring its parents to take inputs from some other intermediate decision node that represents a similar distribution with a smaller structure size.

The split operation splits an element (AND node) into multiple elements by constraining the prime. The elements are split based on a mutually exclusive (disjoint) and exhaustive set of partial assignments to the prime variables. This ensures that the decision node remains deterministic. Indeed, for any assignment to the prime variables that had a non-zero probability in the element

Algorithm 2: Split(n, i, Zs, m)

Input: n, i: the ith element of node n to split, Zs: mutually exclusive and exhaustive set of variable assignments, m: depth of PartialCopy

Result: The ith element of node n is split on Zs.

```
1 n2c = ∅                                          // maps nodes to copies
2 RemoveElement(n, (p_i, s_i))
3 foreach z ∈ Zs do
4     PartialCopy(p_i, z, m, n2c)
5     PartialCopy(s_i, true, m, n2c)
6     AddElement(n, (n2c[p_i], n2c[s_i]))
```

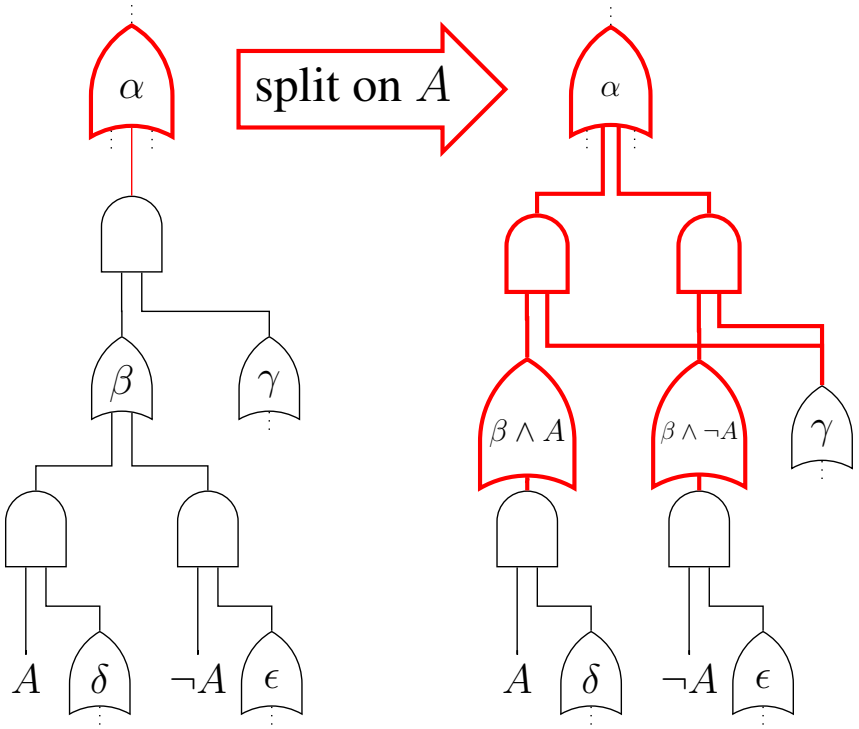


Figure 4.1: Minimal Split. Nodes labels are their base.

Algorithm 3: Clone(n, P, m)

Input: n: node to clone, P: parent nodes and elements to redirect to clone, m: depth of PartialCopy

Result: Parents P are redirected to the clone of n.

1 $n2c = \emptyset$ `// maps nodes to copies`

2 PartialCopy$(n,$ true$,\ m,\ n2c)$

3 **foreach** $(\pi, i) \in P$ **do** Update$(\pi, (i, n, n2c[n]))$

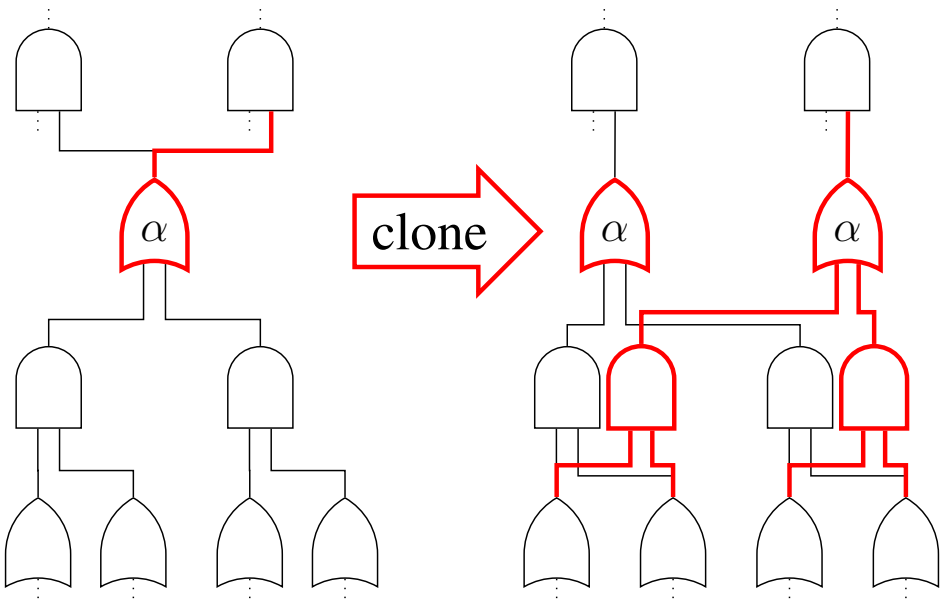


Figure 4.2: Minimal Clone. Base α does not change.

Algorithm 4: PartialCopy($n, \mathbf{z}, m, n2c$)

Input: n: node to copy, $\mathbf{z}$: variable assignment, m: depth of copy, $n2c$: map of nodes to copy

Result: constrained copy of n in $n2c$

```
1  E = ∅
2  X and Y are the partition variables of n
3  z_p = ∃_Y z ; z_s = ∃_X z
4  for i ← 1 to n do
5      if z_p ⊨ [p_i] ∧ z_s ⊨ [s_i] then
6          p' = p_i ; s' = s_i
7          if m > 0 or [p_i] ⇏ z_p then
8              if p_i ∉ n2c then PartialCopy(p_i, z_p, m − 1, n2c)
9              p' = n2c[p_i]
10         if m > 0 or [s_i] ⇏ z_s then
11             if s_i ∉ n2c then PartialCopy(s_i, z_s, m − 1, n2c)
12             s' = n2c[s_i]
13         E = E ∪ [(p', s')]
14 n2c[n] = NewNode(E)
```

before the split, there can be at most one element after the split that assigns a non-zero probability to it. To execute a split (Figure 4.1, Algorithm 2), a new element is created for each partial assignment, where the new prime is a copy of the original prime constrained by the assignment. The new sub is an unconstrained copy. The original element is removed from its decision node.

The clone operation makes a copy of a node and redirects some of the parents to the copy (Figure 4.2, Algorithm 3).

Both operations need to make partial copies of a decision node and its descendants. Our algo-

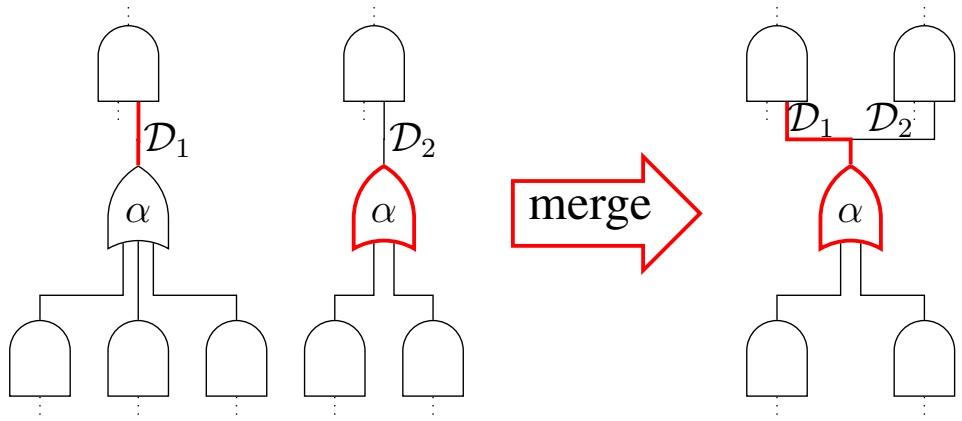


Figure 4.3: Merge. To-be-merged probabilistic circuit nodes are normalized for the same vtree node and have the same base α. The probabilistic circuit node with a smaller number of descendants is retained. The parents of the bigger one are rewired to the smaller one.

rithm can perform these copies up to some specified depth m. A minimal operation ($m = 0$) copies as few nodes as possible, and a complete operation copies all nodes. Any non-minimal operation ($m > 0$) is equivalent to multiple minimal operations. The complete description of the partial-copy algorithm is as the following. A copy of a node creates a new fold for that node and its descendants up to a specified level (Algorithm 4, lines 8,11). The elements of the copy beyond the specified level redirect to nodes of the original PSDD (line 6). Optionally, the copy can be constrained to a partial assignment for some variables. In this case, only descendants that agree with the assignment are kept in the copy (line 5) and nodes beyond the specified level may have to be copied to enforce the constraint (lines 7, 10).

The merge operation takes as input two PSDD decision nodes that respect the same vtree and have the same base. It removes the larger node and redirects its parents to the remaining one; see Figure 4.3. The parameters of the modified substructure need to be re-estimated on the union of the datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ that flows through the original nodes.

4.4.2 Validity of Operations

Before really making use of those operations to propose a competitive structure learning algorithm, we first need to make sure using them will not make PSDDs invalid.

Proposition 2. *Splits and clones maintain a PSDD's syntactic properties and do not alter the base of its root.*

Definition 3 (Valid PSDD node)**.** *A PSDD node n that is normalized for a vtree node v is valid if: (1) all primes p_i are valid nodes and normalized for the left child of v; (2) all subs s_i are valid nodes and normalized for the right child of v; (3) the primes are mutual exclusive: $\forall i \neq j, [p_i] \wedge [p_j] = \bot$; (4) all elements are satisfiable: $\forall i, [p_i] \wedge [s_i] \neq \bot$.*

A valid operation keeps the PSDD syntactically sound and does not alter the base of the root node.

Lemma 1 (PartialCopy$(n, \mathbf{z}, m, n2c)$ is valid)**.** *If the following conditions are satisfied: (1) n is valid; (2) $n2c$ is valid (this means that it only contains entries $n \rightarrow n'$ where n and n' are normalized for the same vtree, valid and $[n'] = [n] \wedge \mathbf{z}_n$, where $\mathbf{z}_n$ is the projection of the assignment $\mathbf{z}$ to the variables in the vtree of node n); (3) $\mathbf{z}$ only contains variables in the vtree of n and is satisfiable in n: $\mathbf{z} \vDash [n]$.*

Proof. Proof by induction.

Note the following preconditions hold and we use them in our proof: (1) $n2c$ includes n; (2) $n2c$ is valid.

Base case: $m = 0$ and $[n] \rightarrow \mathbf{z}$. In the base case, only one decision node is added according to $n2c$ which is n', the copy of n. Because n and n' have the same elements, n' is valid and $[n] = [n']$. Because $\mathbf{z}$ is implied by $[n]$, $[n'] = [n] \wedge \mathbf{z}_n$.

Induction step: To use the inductive assumption, we first show that the preconditions hold for the calls of PartialCopy. Because n is valid, so are its primes and subs. By induction and

precondition, $n2c$ is valid. Finally, $\mathbf{z}_p$ and $\mathbf{z}_s$ only contain the relevant variables because the others are forgotten using existential quantification.

The first postcondition is satisfied because n is added to $n2c$ in line 14. In terms of the second postcondition, we consider 3 cases: (i) The entry is already in $n2c$ when PartialCopy is called, then it is valid because of the precondition. (ii) The entry is added by a recursive call of PartialCopy, then it is valid because of induction. (iii) The entry is $n \to n'$, where n' has an element (p'_i, s'_i) for every element $(p_i, s_i) \in n$, except for those that do not agree with the assignment: $p_i \wedge \mathbf{z}_p = \bot$ or $s_i \wedge \mathbf{z}_s = \bot$. p'_i and s'_i are normalized for the correct vtrees because they either are the original children, or they come from $n2c$ which is valid by the precondition and induction.

We proceed to prove the mutual exclusivity of the copied primes, the satisfiability of the copied elements, and the correctness of the base of the copied decision node.

The primes of n' are mutually exclusive:

$$\begin{aligned}
[p'_i] \wedge [p'_j] &= [p_i] \wedge \mathbf{z}_p \wedge [p_j] \wedge \mathbf{z}_p \\
&= [p_i] \wedge [p_j] \wedge \mathbf{z}_p \\
&= \bot
\end{aligned}$$

All elements of n' are satisfiable because all the elements of n are satisfiable and elements that would become unsatisfied by conditioning on $\mathbf{z}$ are removed.

The base of n' is the base of n constraint by $\mathbf{z}$:

$$\begin{aligned}
[n'] &= \bigvee_{i \in n : \mathbf{z}_p \vDash [p_i] \wedge \mathbf{z}_s \vDash [s_i]} [p'_i] \wedge [s'_i] \\
&= \bigvee_{i \in n} [p_i] \wedge \mathbf{z}_p \wedge [s_i] \wedge \mathbf{z}_s \\
&= \mathbf{z} \wedge \bigvee_{i \in n} [p_i] \wedge [s_i] \\
&= \mathbf{z} \wedge [n]
\end{aligned}$$

□

Proposition 3 (Split(n, i, Zs, m) is valid)**.** *If the following conditions are satisfied: (1) n is valid. (2) All $\mathbf{z} \in Zs$ only contain variables of the left children of n's vtree and are satisfiable in the ith element of n: $\mathbf{z} \vDash [p_i] \wedge [s_i]$. (3) All $\mathbf{z} \in Zs$ are mutually exclusive and exhaustive.*

Proof. Note that the following postconditions hold and we use them in our proof: (1) n is valid; (2) the base of n is not altered: $[n] = [n_{\text{old}}]$.

The primes and subs of n are normalized for the correct vtree because n is valid and $n2c$ is valid (Lemma 1).

The primes of n are mutually exclusive if: (i) the original primes are mutually exclusive, (ii) the new primes are mutually exclusive, and (iii) every pair of an original prime and a new prime is mutually exclusive.

All the elements of n are satisfiable, because the precondition states that all the assignments must be satisfiable in the split element.

The original base of n is $[n_{\text{old}}] = \bigvee_j [p_j] \wedge [s_j]$. After the split, the base is:

$$
\begin{aligned}
[n] &= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \bigvee_{\mathbf{z} \in Zs} [p_{i,\mathbf{z}}] \wedge [s_i] \\
&= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \bigvee_{\mathbf{z} \in Zs} [p_i] \wedge \mathbf{z} \wedge [s_i] \\
&= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \Big([p_i] \wedge [s_i] \wedge \bigvee_{\mathbf{z} \in Zs} \mathbf{z} \Big) \\
&= \bigvee_{j \neq i} [p_j] \wedge [s_j] \vee \Big([p_i] \wedge [s_i] \Big) \\
&= [n_{\text{old}}]
\end{aligned}
$$

□

Proposition 4 (Clone(n, P, m) is valid)**.** *If the following conditions are satisfied: (1) n is valid; (2) $\forall (\pi, i) \in P$, n is either $p_{\pi,i}$ or $s_{\pi,i}$.*

Proof. Note the following postconditions hold and we use them in our proof: (1) $\forall(\pi, i) \in P$, π is valid; (2) $\forall(\pi, i) \in P$, the base of π is not altered: $[\pi] = [\pi_{\text{old}}]$. Because of lemma 1 and the preconditions, n' is a valid node with the same vtree and base as n. Redirecting the parents to this node therefore keeps the parents valid and also remains the base as unaltered. □

Proposition 5 (Merge(n, m) is valid)**.** *If the following conditions are satisfied: (1) n and m are normalized for the same vtree node; (2)n and m have the same base.*

Proof. n and m are valid nodes in the first place. Given this and the preconditions, redirecting the parents of one node to the other keeps the parents valid. □

4.4.3 Locality of Splits and Clones

Splits and clones are local operations. Only the node that is modified, the parents that are redirected and the copied descendants are affected. Furthermore, key properties of an operation, such as the required change in PSDD structure and the improvement in likelihood, are typically not affected by operations elsewhere in the PSDD.

Local operations have four desirable properties. First, the complexity of executing an operation is bounded by the number of elements it affects; cheap operations are thus possible in large PSDDs. Second, the difference in PSDD size after an operation can be easily obtained; it is the difference in the affected elements.

Third, the difference in likelihood can be computed by only looking at the elements that are affected. Indeed, [KAD14, long version] proves that the log-likelihood decomposes over the PSDD elements as follows.

Proposition 6. *The log-likelihood of PSDD r given data $\mathcal{D}$ is a sum of log-likelihood contributions*

per node:[1]

$$\ln \mathcal{L}(r|\mathcal{D}) = \ln \Pr_r(\mathcal{D}) = \sum_{n \in r} \sum_{i \in n} \ln \theta_{n,i}\, \mathcal{D}\#(\gamma_n, [p_{n,i}]),$$

where $\mathcal{D}\#(\gamma_n, [p_{n,i}])$ *is the number of examples that satisfy the node context of* n *and the base of* n*'s prime* $p_{n,i}$*.*

Fourth, we would like to simulate candidate operations before committing to execute them. Because size and likelihood changes are not affected by other operations, we can cache their values when considering a large number of candidate operations during structure search.

Local operations support principled tractable learning, using exact estimates of likelihood and tractability (size). Many other learners, especially traditional ones, are required to approximate the likelihood and have no ability to reliably determine the tractability of a learned model.

4.4.4 LEARNPSDD Algorithm

We build on our split and clone operations to create the first PSDD structure learning algorithm called LEARNPSDD[2]. It incrementally improves the structure of an existing PSDD to better fit the data. In every step, the structure is changed by executing an operation. Learning continues until the log-likelihood on validation data stagnates, or a desired time or size limit is reached. The operation to execute is greedily chosen based on the best likelihood improvement per size increment:

$$\text{score} = \frac{\ln \mathcal{L}(r' \mid \mathcal{D}) - \ln \mathcal{L}(r \mid \mathcal{D})}{\text{size}(r') - \text{size}(r)}$$

where r is the original and r' the updated PSDD.

The algorithm needs to be provided with an initial PSDD and vtree. It can take any PSDD, even one that encodes domain knowledge in its base, as is done in existing applications of PSDDs. It can also be a trivial, maximally uninformative PSDD n whose base $[n] =$ true and whose

[1] *This equation treats terminal nodes as degenerate decision nodes with primes* X *and* $\neg X$*, and subs true and false* .

[2] Open-source code and experiments are available at `https://github.com/UCLA-StarAI/LearnPSDD`.

distribution factorizes completely over the variables. The vtree can either come from compiling those constraints, or can be learned from data as described in Section 4.3.

In each iteration, LEARNPSDD considers one clone per node and one split per element. The clone is the best clone for that node where at most k parents are moved to the copy. The split is the best split with the partial assignments limited to one prime variable. Only the scores of the operations that use nodes affected by the previous iteration's operation need to be recalculated.

The operation depth parameter m is fixed during learning. The larger this parameter, the more elements are added and the larger the log-likelihood improvement per operation. A large m speeds up the learning but learns larger PSDDs, which are more prone to overfitting.

4.4.5 Implementation Details

We discuss the implementation details of LEARNPSDD.

Data In The Nodes The training data is explicitly kept in PSDD nodes during learning. Every node contains a bitset that indicates which examples agree with the context of that node. This speeds up parameter estimation and log-likelihood calculations, which are needed for every execution and simulation of an operation. For simulation of an operation, a bitmask is used to represent the examples that are moved to a copy.

Unique Node Cache To avoid duplicate calculations when doing inference, PSDD should not have duplicate nodes. This is accomplished using the unique-node technique, where a cache of the nodes is kept and it is checked every time before creating a new node [MT12]. In general, two nodes are considered equal if they have the same (p, s, θ) elements. During learning, however, we adapt this by considering two nodes different if they might evolve to a different structure, based on the training data that it contains. There are two reasons for a node not to change. First, if the node's base is a complete assignment, i.e., if all descendants of this node have only one element, then there are possible LEARNPSDD operations. A clone would be useless in this case because

all the parameters would remain as 1. Second, if the node contains no data. Such a node cannot contribute to the log-likelihood and has therefore no reason to change.

The number of added nodes is no longer a local characteristic of an operation, as it depends on the nodes available in the cache. To cope with this, we consider nodes that can be cached as free nodes: they are not counted in the score. This makes sense because if the node is already in the cache, it does not need to be added, otherwise adding it to the cache can make subsequent operations less expensive to simulate or execute.

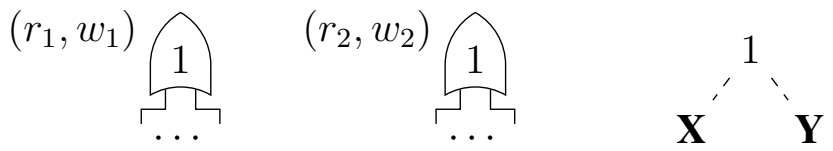
SDDs In The Nodes SDDs are kept in the nodes to represent their base. This is not really needed, because the base is implied by the structure of the PSDD. However, during structure learning, PSDDs grow bigger, while SDDs do not. Therefore, if the base needs to be checked, doing this on the SDD is more efficient. Note that before any structure learning is done, the SDD is larger than the PSDD because SDD's primes need to be exhaustive and therefore the SDD may have elements for subs that represent false. However, PSDDs are expected to grow larger than the corresponding SDDs during structure learning.

4.5 Ensembles and Structural EM

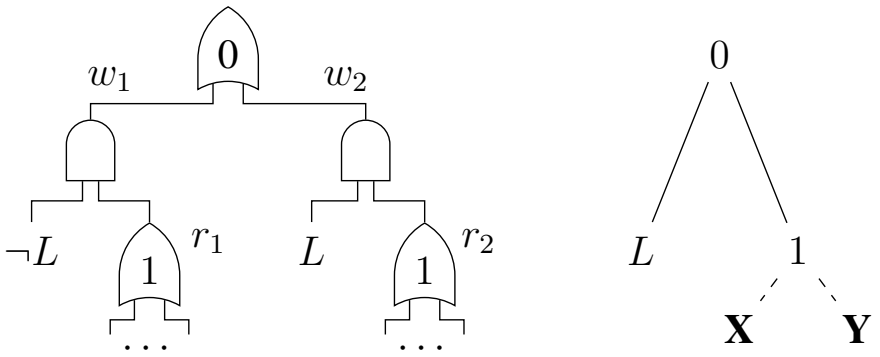
This section extends LEARNPSDD to induce mixtures of PSDDs. A mixture of PSDDs $\mathcal{M}$ is a set of pairs (r_i, w_i) where each r_i is a component PSDD and w_i is its mixture weight. A mixture of n PSDDs must have $\sum_{i=1}^{n} w_i = 1$. We further assume that all r_i are normalized for the same vtree. A mixture of PSDDs $\mathcal{M}$ represents the probability distribution $\Pr_{\mathcal{M}}(\mathbf{X}) = \sum_{i=1}^{n} w_i \Pr_{r_i}(\mathbf{X})$.

An ensemble of PSDDs is equivalent to a single PSDD with latent variables. More precisely, by adding $\lceil \log(n) \rceil$ Boolean variables $\mathbf{L}$ to the top of the vtree (encoding an n-valued latent component identifier), and mixing between the component PSDDs with an additional decision node, one can capture the distribution $\Pr_{\mathcal{M}}(\mathbf{X})$ in a single PSDD circuit. Figure 4.4 depicts this reduction.

Because the latent variables $\mathbf{L}$ are not observable, the mixture weights w_i cannot be learned



(a) Mixture of two PSDDs with outline of a common vtree.



(b) Equivalent single PSDD and vtree with latent variable L.

Figure 4.4: Representing ensembles as a single PSDD.

from data in closed form. Instead, we appeal to expectation maximization (EM) for optimizing the likelihood $\mathcal{L}(\mathcal{M} \mid \mathcal{D})$ given dataset $\mathcal{D} = \left\{\mathbf{d}^{(1)}, \mathbf{d}^{(2)}, \ldots, \mathbf{d}^{(M)}\right\}$.

We propose EM-LEARNPSDD, a variant of the (soft) structural EM algorithm [Fri98], to learn the structure and parameters of ensembles of PSDDs. In soft EM, each example $\mathbf{x}^{(j)}$ takes part in each component (that is, each PSDD r_i) with weight $\alpha_{i,j}$, resulting in weighted datasets $\bar{\mathcal{D}}_i$ for each component. Weights $\alpha_{i,j}$ represent the probability that example $\mathbf{x}^{(j)}$ belongs to distribution r_i, and therefore $\sum_i \alpha_{i,j} = 1$ for all j.

EM-LEARNPSDD consists of two nested learners: an outer EM for structure learning and an inner EM for parameter learning. The outer E-step is the inner learner. The outer M-step uses LEARNPSDD to improve the structure of all PSDD components given the weighted datasets $\bar{\mathcal{D}}_i$. It also updates the component weights as

$$w_i = \sum_{j=1}^{M} \alpha_{i,j} / \sum_{k=1}^{n} \sum_{j=1}^{M} \alpha_{k,j}.$$

The inner E-step redistributes the data over components r_i. For every example $\mathbf{d}^{(j)}$, it updates the weights in each component's weighted dataset $\bar{\mathcal{D}}_i$ as

$$\alpha_{i,j} = \frac{\Pr_{r_i}(\mathbf{d}^{(j)})}{\sum_{k=1}^{n} \Pr_{r_k}(\mathbf{d}^{(j)})}.$$

The inner M-step learns the parameters in r_i from $\bar{\mathcal{D}}_i$ using closed-form estimates (employing a weighted version of Equation 4.1). Internal EM steps alternate until convergence, or for a maximum number of iterations. We find empirically that a maximum of 3 inner EM iterations is sufficient to improve the parameters and warrants moving to another iteration of the outer EM structure learner.

The initial weighted datasets $\bar{\mathcal{D}}_i$ are found by k-means clustering on $\mathcal{D}$. These clusters are softened by weighting an example in the cluster with $1 - (n-1)\epsilon$ and one not in the cluster with ϵ (default 0.05). K-means clustering empirically provides a better starting point for EM-LEARNPSDD, as worlds (i.e., examples) belonging to the same component distribution tend to be closer in Euclidean distance.

4.6 Related Work

Sentential Decision Diagrams (SDD) are tractable representations that are closely related to the PSDD. Despite being a purely logical circuit, one can reduce statistical models to a weighted model counting task on an SDD encoding [CKD13]. [BDC15] learn Markov networks that have a compact SDD for weighted model counting. The learning algorithm uses bottom-up compilation to incrementally add factors to the SDD. It selects features based on a likelihood vs. size trade-off. Adding features is a global modification and requires all parameters to be re-learned by convex optimization.

PSDDs can be reduced to sum-product networks (SPNs), which are a syntactic variation on arithmetic circuits (ACs). A PSDD can be turned into an equivalent SPN by replacing AND nodes by products and OR nodes by sums. Several learning algorithms for SPNs exist. LearnSPN induces

an SPT (an SPN tree structure) by splitting on latent variables [GD13]. O-SPN and L-SPN improve this algorithm by merging parts of the SPT back into a DAG [RG16b]. [VDE15] describe various improvements to reduce overfitting in LearnSPN. SearchSPN shares with LEARNPSDD that it uses local operators (a combination of a type of minimal split on a latent variable with a type of minimal clone) [DV15].

Probabilistic decision graphs (PDGs) have a variable forest that defines the dependencies between variables, much like vtrees [JNS06]. To induce a variable forest, one learns small PDGs for different forests and chooses the best one. PDG structure learning applies split, merge, and redirect operations to the graph in a fixed order, much like L-SPN and O-SPN.

Beyond these, there is a vast literature on tractable learning algorithms that are less related to LEARNPSDD, include ACBN [LD08], ACMN [LR13], ID-SPN [RL14] and ECNet [RG16a].

4.7 Experiments

We evaluate the performance of LEARNPSDD and EM-LEARNPSDD, and provide deeper insights into PSDD learning. Section 4.7.2 evaluates how vtrees affect the learner. Section 4.7.3 to 4.7.6 demonstrate that PSDDs are amenable to learning in probability spaces without logical constraints. Section 4.7.7 illustrates that we can further shrink the learned PSDDs to make them more interpretable without sacrificing the data fit. Lastly, Section 4.7.8 shows that LEARNPSDD is able to capture a logically constrained probabilistic space while also fitting the data well.

4.7.1 Setup

We evaluate our learners on a standard benchmark suite [VD12]. This suite consists of 20 real-world datasets. Their important attributes are summarized in Table 4.1. These datasets have been used in various previous works for evaluating the performance of assorted tractable model learners [GD13, LR13, ABG15, RL14, RG16a]. These datasets do not assume any prior domain knowledge,

Table 4.1: Important attributes of the 20 standard density estimation benchmark datasets.

Dataset	|Var|	|Train|	|Valid|	|Test|
NLTCS	16	16181	2157	3236
MSNBC	17	291326	38843	58265
KDD	64	1800992	19907	34955
Plants	69	17412	2321	3482
Audio	100	15000	2000	3000
Jester	100	9000	1000	4116
Netflix	100	15000	2000	3000
Accidents	111	12758	1700	2551
Retail	135	22041	2938	4408
Pumsb-Star	163	12262	1635	2452
DNA	180	1600	400	1186
Kosarek	190	33375	4450	6675
MSWeb	294	29441	32750	5000
Book	500	8700	1159	1739
EachMovie	500	4524	1002	591
WebKB	839	2803	558	838
Reuters-52	889	6532	1028	1530
20NewsGrp.	910	11293	3764	3764
BBC	1058	1670	225	330
AD	1556	2461	327	491

and are not associated with any logical constraints. Our experiments run for 24 hours or until convergence on the validation set, whichever happens earlier. They run on servers with 16-core 2.6GHz Intel Xeon CPUs and 256GB RAM.

4.7.2 Impact Of Vtrees

Because a PSDD's structure is so strongly constrained by the vtree it is normalized for, we would expect vtrees to play a crucial role in determining the size of a PSDD, and more importantly, the

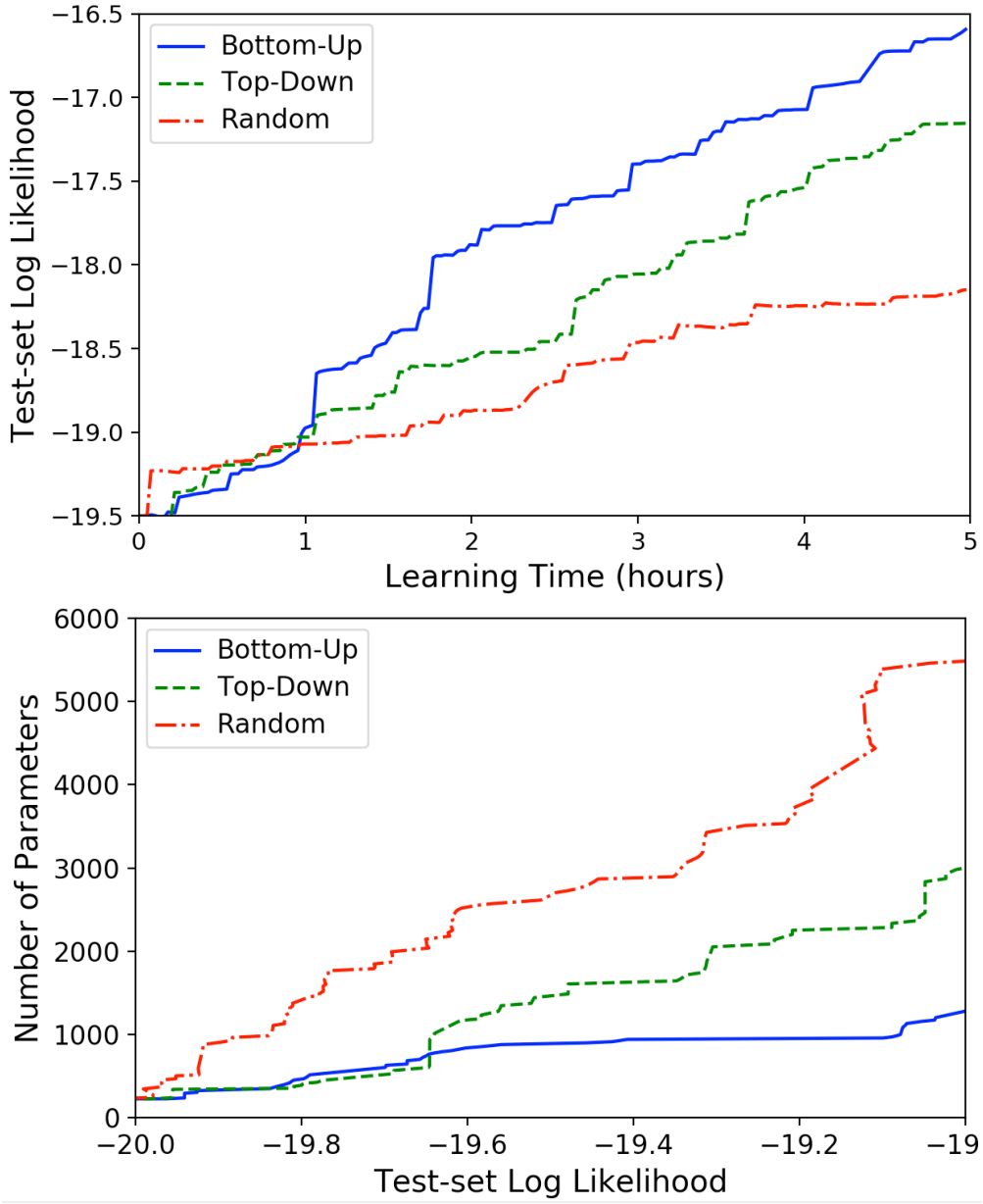


Figure 4.5: Bottom-up-induced vtrees result in better PSDDs, with higher likelihood and fewer parameters (bottom figure) and are learned in less time (top figure). This particular comparison figure is generated by running experiments on the dataset Plant. A similar pattern can also be observed in other datasets.

quality of the probability distributions we can learn with the given data and PSDD size available. To evaluate vtrees' impact and the vtree learners described in Section 4.3, we generate 3 vtrees per dataset: (i) using top-down induction, (ii) using bottom-up induction, and (iii) a balanced vtree with a random variable ordering. We run LEARNPSDD for five hours, with the operation depth parameter m set to 3, using these 3 vtrees. We compare the learned PSDDs in terms of their quality (log-likelihood) as a function of learning time, and their size (number of parameters) as a function of quality.

Figure 4.5 shows the experimental results for a representative dataset (plants). As expected, bottom-up induction learns superior vtrees, followed by top-down and finally random. The PSDD with a better vtree achieves a higher log-likelihood and is more tractable (smaller). Moreover, a better vtree reduces learning time. In all three cases, LEARNPSDD starts from a trivial initial PSDD. Hence, the log-likelihood is the same for all vtrees at the start of learning. Bottom-up vtree induction is used for the remaining experiments.

4.7.3 Evaluation Of LEARNPSDD

As discussed in Section 4.6, the approach LEARNPSDD takes is closely related to SearchSPN. We therefore evaluate LEARNPSDD's performance in comparison to SearchSPN. The operation copy depth m of LEARNPSDD is fixed to 3. As shown in Table 4.2, LEARNPSDD achieves a better testset log-likelihood than SearchSPN (or ties) in 5 datasets while being competitive in most of the other datasets. In general, LEARNPSDD's performance is weaker than SearchSPN. This is expected, because SearchSPN uses many thousands of latent variables, while LEARNPSDD uses none, and PSDDs are necessarily more restrictive than SPNs.

4.7.4 Evaluation Of EM-LEARNPSDD

The structural EM algorithm that augments LEARNPSDD essentially decomposes the optimization problems for parameter and structure learning. With a new layer of modeling power, EM-

Table 4.2: Comparison among LEARNPSDD, EM-LEARNPSDD, SearchSPN, merged L-SPN and merged O-SPN in terms of performance (log-likelihood) and model size (number of parameters). Sizes for SearchSPN are not reported in the original paper. We use the following notation: (1) LL: Average test-set log-likelihood; (2) Size: Number of parameters in the learned model; (3) † denotes a better LL between LEARNPSDD and SearchSPN; (4) * denotes a better LL between LEARNPSDD and EM-LEARNPSDD; (5) Bold likelihoods denote the best LL among EM-LEARNPSDD, SearchSPN, merged L-SPN and merged O-SPN.

Dataset	|Var|	LEARNPSDD		EM-LEARNPSDD		SearchSPN	Merged L-SPN		Merged O-SPN	
		LL	Size	LL	Size	LL	LL	Size	LL	Size
NLTCS	16	$-6.03^{\dagger *}$	3170	$\mathbf{-6.03}^*$	2147	-6.07	-6.04	3988	-6.05	1152
MSNBC	17	$-6.05^{\dagger}$	8977	$\mathbf{-6.04}^*$	3891	-6.06	-6.46	2440	-6.08	9478
KDD	64	$-2.16^{\dagger}$	14974	$\mathbf{-2.12}^*$	9182	-2.16	-2.14	6670	-2.19	16608
Plants	69	-14.93	13129	-13.79^*	13951	$-13.12^{\dagger}$	$\mathbf{-12.69}$	47802	-13.49	36960
Audio	100	-42.53	13765	-41.98^*	9721	$-40.13^{\dagger}$	$\mathbf{-40.02}$	10804	-42.06	6142
Jester	100	-57.67	11322	-53.47^*	7014	$-53.08^{\dagger}$	$\mathbf{-52.97}$	10002	-55.36	4996
Netflix	100	-58.92	10997	-58.41^*	6250	$-56.91^{\dagger}$	$\mathbf{-56.64}$	11604	-58.64	6142
Accidents	111	-34.13	10489	-33.64^*	6752	$-30.02^{\dagger}$	$\mathbf{-30.01}$	13322	-30.83	6846
Retail	135	-11.13	4091	$\mathbf{-10.81}^*$	7251	$-10.97^{\dagger}$	-10.87	2162	-10.95	3158
Pumsb-Star	163	-34.11	10489	-33.67^*	7965	$-28.69^{\dagger}$	$\mathbf{-24.11}$	17604	-24.34	18338
DNA	180	-89.11^*	6068	-92.67	14864	$\mathbf{-81.76}^{\dagger}$	-85.51	4320	-87.49	1430
Kosarek	190	$-10.99^{\dagger}$	11034	-10.81^*	10179	-11.00	$\mathbf{-10.62}$	5318	-10.98	6712
MSWeb	294	$-10.18^{\dagger}$	11389	-9.97^*	14512	-10.25	$\mathbf{-9.90}$	16484	-10.06	12770
Book	500	-35.90	15197	-34.97^*	11292	$-34.91^{\dagger}$	$\mathbf{-34.76}$	11998	-37.44	11916
EachMovie	500	-56.43^*	12483	-58.01	16074	$-53.28^{\dagger}$	$\mathbf{-52.07}$	15998	-58.05	19846
WebKB	839	-163.42	10033	-161.09^*	18431	$-157.88^{\dagger}$	$\mathbf{-153.55}$	20134	-161.17	10046
Reuters-52	889	-94.94	10585	-89.61^*	9546	$-86.38^{\dagger}$	$\mathbf{-83.90}$	46232	-87.49	28334
20NewsGrp.	910	-161.41	12222	-161.09^*	18431	$-153.63^{\dagger}$	$\mathbf{-154.67}$	43684	-161.46	29016
BBC	1058	-260.83	10585	$\mathbf{-253.19}^*$	20327	$-252.13^{\dagger}$	-253.45	21160	-260.59	8454
AD	1556	-30.49^*	9666	-31.78	9521	$-16.97^{\dagger}$	-16.77	49790	$\mathbf{-15.39}$	31070

LEARNPSDD is expected to learn more effectively. Therefore, we reduce the depth parameter m to learn smaller PSDD components while retaining a high-quality mixture overall. This experiment uses a combination of minimal operations (80%) and operations with a depth of 3 (20%). Minimal operations get chosen most often, resulting in smaller circuits for the same number of operations. We determine the number of components by conducting a grid search over $\{3, 5, 7, 9\}$ on validation data and report the best result for each dataset. EM-LEARNPSDD surpasses or ties the performance of LEARNPSDD in 17 datasets and it learns smaller models in 13 datasets; see Table 4.2. EM-LEARNPSDD is superior to LEARNPSDD in 12 datasets by being more accurate and more tractable at the same time.

4.7.5 Comparison with SPN Learners

SPNs have been demonstrated to be quite effective for tractable learning in probability spaces that are not subject to logical domain constraints. SPN learners have generated state-of-the-art results in the 20 benchmark datasets [RL14, RG16b]. Specifically, merged L-SPN and O-SPN are the first few SPN structure learners that consider a heuristic merging strategy and therefore produce SPNs that have a significant advantage in size with no loss in performance. In fact, merging for SPNs shows an improvement in test-set log-likelihood for most datasets [RG16b]. We will demonstrate the effect of merging on PSDDs in Section 4.7.7. We compare our EM-LEARNPSDD with merged L-SPN and merged O-SPN.

As shown in Table 4.2, EM-LEARNPSDD is competitive with merged L-SPN and O-SPN. This result is surprising because PSDDs are much more restrictive than SPNs. EM-LEARNPSDD outperforms O-SPN on likelihood in 11 datasets, learns smaller models in 14 datasets, and wins on both measures in 6 datasets; EM-LEARNPSDD outperforms L-SPN on likelihood in 6 datasets, learns smaller models in 14 datasets and wins on both in 2 datasets.

Table 4.3: Comparison of test-set log-likelihood between LearnPSDD and the state of the art († denotes the best).

Datasets	|Var|	LEARNPSDD Ensemble	Best-to-Date
NLTCS	16	$-5.99^{\dagger}$	-6.00
MSNBC	17	$-6.04^{\dagger}$	$-6.04^{\dagger}$
KDD	64	$-2.11^{\dagger}$	-2.12
Plants	69	-13.02	$-11.99^{\dagger}$
Audio	100	-39.94	$-39.49^{\dagger}$
Jester	100	-51.29	$-41.11^{\dagger}$
Netflix	100	$-55.71^{\dagger}$	-55.84
Accidents	111	-30.16	$-24.87^{\dagger}$
Retail	135	$-10.72^{\dagger}$	-10.78
Pumsb-Star	163	-26.12	$-22.40^{\dagger}$
DNA	180	-88.01	$-80.03^{\dagger}$
Kosarek	190	$-10.52^{\dagger}$	-10.54
MSWeb	294	-9.89	$-9.22^{\dagger}$
Book	500	-34.97	$-30.18^{\dagger}$
EachMovie	500	-58.01	$-51.14^{\dagger}$
WebKB	839	-161.09	$-150.10^{\dagger}$
Reuters-52	889	-89.61	$-80.66^{\dagger}$
20NewsGrp.	910	-155.97	$-150.88^{\dagger}$
BBC	1058	-253.19	$-233.26^{\dagger}$
AD	1556	-31.78	$-14.36^{\dagger}$

4.7.6 Comparison with the State of the Art

In this section, we demonstrate that we can achieve near state-of-the art performance using our EM-LEARNPSDD algorithm. It was shown in previous studies that bagged ensembles with expectation maximization can significantly improve results on many of the 20 datasets [RG16a, RG16b]. We therefore build bagging ensembles on top of EM-LEARNPSDD. The result is still equivalent to a single PSDD, by a translation similar to the one shown in Figure 4.4 for mixture models, except the w_is for bagging represent a uniform distribution. Our goal with this experiment is to match or exceed the state-of-the-art. This is a very strong baseline, consisting of five competitive tractable model learners: (1) ACMN [LR13], (2) ID-SPN [RL14], (3) SPN-SVD [ABG15], (4) ECNet [RG16a] and (6) Merged L-SPN [RG16b].

When fixing the number of bags to 10, EM-LEARNPSDD is competitive with the state of the art and surpasses it on 6 out of 20 datasets; see Table 4.3.

Overall, the experiments outlined so far have incrementally demonstrated that the PSDD structure learning algorithms proposed in this paper (LEARNPSDD and EM-LEARNPSDD) perform competitively in classical probability spaces without domain constraints. This is despite the fact that PSDDs are more tractable and have more syntactic properties than their alternatives.

4.7.7 Effects of Merging

Both the interpretability and the tractability of a learned PSDD depend critically on its size. In LEARNPSDD, this is largely a function of the depth of splits and clones. This subsection reports our work on controlling the size of the PSDD during learning by merging similar substructures with an algorithm called MERGEPSDD. This helps find the right trade-off between the number of parameters and the data fit, and eliminates the need to tune the depth parameter.

MERGEPSDD needs an initial learned PSDD. The default approach is to learn a large PSDD using LEARNPSDD. MERGEPSDD iterates over vtree nodes bottom-up. In each iteration, all pairwise combinations of PSDD decision nodes that respect the considered vtree node and that

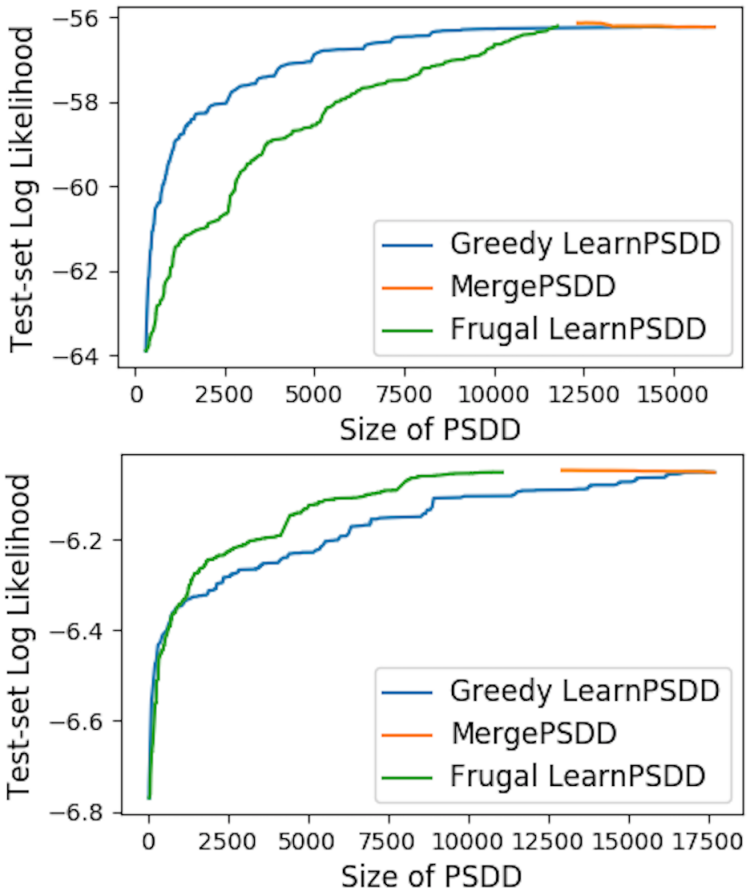


Figure 4.6: MERGEPSDD prunes away "unnecessary" PSDD structures while slightly improving performance. Top: on dataset Jester. Bottom: on dataset MSNBC.

Table 4.4: Number of parameters in PSDDs learned by LEARNPSDD using frugal or greedy operations, and MERGEPSDD. LL is the desired test-set log-likelihood.

Dataset	Target LL	Frugal LEARNPSDD	Greedy LEARNPSDD	MERGEPSDD
NLTCS	-6.08	7491	31669	23471
MSNBC	-6.05	11074	17687	12943
KDD	-2.19	13814	29429	18921
Plants	-16.98	12021	12398	11574
Audio	-44.64	5804	5494	5389
Jester	-56.21	11774	16149	12349

also have the same base are the candidates for the merge operation. The pairwise combination that yields the lowest intersectional divergence is chosen. To be more careful, the chosen merge operation is simulated in advance and only executed if the likelihood on validation data does not decrease. We evaluate the effectiveness of MERGEPSDD with a focus on size decrements. For each dataset in Table 4.4, two different PSDDs are learned, one using greedy operations (complete split) and the other using frugal operations (a mixture of 80% minimal operations and 20% depth-3 operations). Greedy LEARNPSDD aims to maximize the learning speed, but some PSDD size may be wasted. Frugal LEARNPSDD aims to have a better balance between size and learning speed. LEARNPSDD runs for 24 consecutive hours or until reaching the desired test-set log-likelihoods, whichever happens earlier. As expected, the model learned by greedy LEARNPSDD is significantly larger than the one learned by frugal LEARNPSDD.

MERGEPSDD is applied on the model learned by greedy LEARNPSDD. MERGEPSDD run sfor 6 consecutive hours or until all potential merge operations are exhausted, whichever happens first. As shown in Figure 4.6, MERGEPSDD effectively shrinks the gap in the size of the models learned by greedy LEARNPSDD and frugal LEARNPSDD. A full result on all 6 datasets that MERGEPSDD runs on is reported in Table 4.4. It shows that MERGEPSDD is able to effectively

reduce PSDD size, making the models more tractable and interpretable, without sacrificing too much in terms of model quality, by virtue of our KL-divergence heuristic.

4.7.8 Evaluation in Constrained Space

PSDDs pay for their desirable properties, such as their ability to encode domain knowledge into their base, and ability to answer complex queries, by being a more restrictive representation. The experiments so far do not directly exploit these desirable properties, to allow for a comparison with other tractable learners. They therefore only experience the restrictiveness. However, the next experiments show that in practical domains, and spaces with domain constraints in particular, having these desirable properties can be a great advantage.

Many real-world datasets contain discrete multi-valued data, instead of being only binary. The straightforward way to use general ACs for multi-valued domains, is to introduce a binary variable for each value of the multi-valued variable. Unfortunately, in the learned distribution, it will then be possible for a multi-valued variable to have multiple values simultaneously. PSDDs can easily cope with this by encoding into the base that binary variables belonging to the same multi-valued variable must be mutually exclusive, and at least one must be true.

To assess the advantage of PSDD in this setting, we compare three learning approaches: (i) LEARNPSDD without domain constraints, (ii) parameter learning on an SDD that is compiled from the constraints (as in prior work, for example [KAD14]), and (iii) applying LEARNPSDD on the initial PSDD obtained from (ii). We use the same vtree in all settings and run LEARNPSDD for 5 hours. We conduct the experiments on two real-world datasets from the UCI repository: Adult and CoverType. Continuous features are discretized into four equal-sized bins. Adult has 14 original (125 binary) variables and CoverType has 12 original (84 binary) variables. Adult and CoverType respectively contain 32,562 and 581,012 examples.

As expected, learning structure on top of the constraints yields the best models. Interestingly, only using the constraints to come up with the SDD structure strongly outperforms unconstrained

Table 4.5: Incorporating domain constraints improves the quality of the learned distributions. Compared settings: (i) unconstrained LEARNPSDD, (ii) constrained PSDD (no LEARNPSDD), and (iii) constrained LEARNPSDD.

Dataset	No Constraint	PSDD	LEARNPSDD
Adult	-18.41	-14.14	-12.86
CovType	-14.39	-8.81	-7.32

structure learning, which shows that ignoring constraints complicates learning significantly. The improvement is due to the fact that the probabilities of many impossible assignments (given the multi-valued constraint) are set to 0 and hence the probabilities of the remaining assignments correspondingly increase.

4.8 Discussion

The two questions raised at the beginning of this chapter both receive a strong positive answer. LEARNPSDD is an effective algorithm for learning PSDD structures. It achieves some state-of-the-art results learning classical probability distributions that are not subject to constraints. Moreover, it can just as easily induce structure over logically constrained spaces without losing any domain-specific information.

CHAPTER 5

Logistic Circuits: Discriminative Learning of Probabilistic Circuits

In the last chapter, we have demonstrated that probabilistic circuits are amenable to generative learning. However, for many applications, one is instead more interested in knowing the most likely label for the given input. This task is commonly known as classification. To do well in classifications, learning the complete joint distribution over the input feature variables plus the label variable (i.e., generative learning) may not be necessary or even desirable. It is commonly believed that directly learning the conditional probability of the label variable given the input features (i.e., discriminative learning) could yield better classification accuracy [NJ02]. Considering this, we dedicate this chapter to the third big challenge in this dissertation — how to learn a competitive classifier from data while retaining many of our circuit representations' properties. To be specific, this chapter proposes a novel classification representation called *logistic circuits*, which are the discriminative counterpart of probabilistic circuits.

5.1 Background

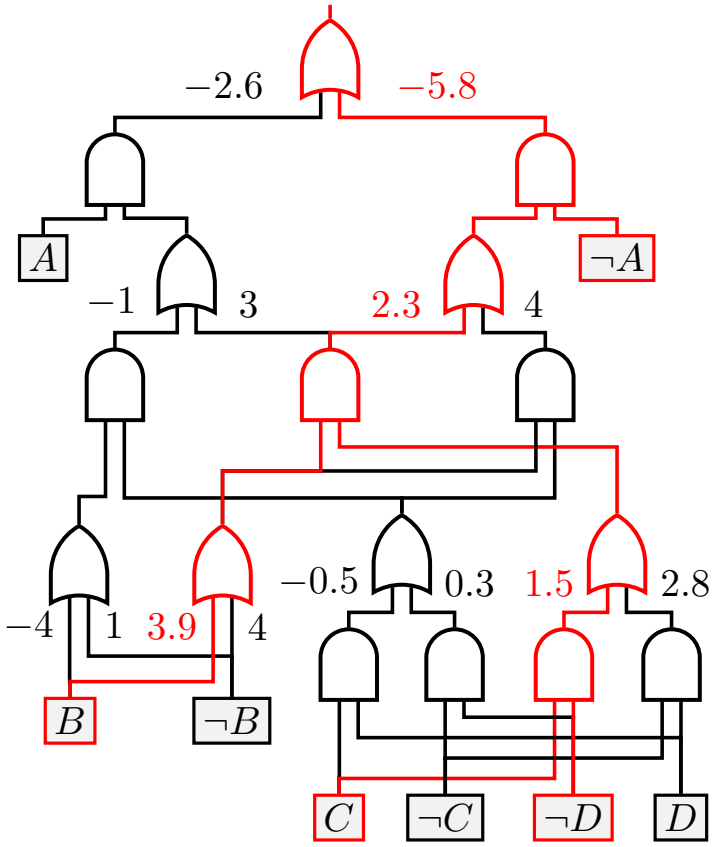
As demonstrated in the last chapter, circuit representations are a promising synthesis of symbolic and statistical methods in AI. They are "deep" layered data structures with statistical parameters, yet they also capture intricate structural knowledge. Besides LEARNPSDD's success in learning tractable joint distributions possibly subject to logical constraints, there exist other competitive generative learning algorithms proposed for different dialects of probabilistic circuits

[LD08, RKG14, PD11], which have also been discussed extensively in the last chapter. Collectively, these approaches achieve the state of the art in discrete density estimation and vastly outperform classical probabilistic graphical model learners [GD13, RL14, ABG15, RG16b]. However, we have not observed the same success when deploying circuit representations for *classification or discriminative learning*. Probabilistic circuit classifiers significantly lag behind the performance of neural networks [Ben18].

To further advance the frontier of symbolic-statistical synthesis and address this performance lag, we propose a new classification representation called *logistic circuits*, which shares many syntactic properties with the representations mentioned earlier. One can view logistic circuits as the discriminative counterpart to probabilistic circuits. Owing to their elegant properties, learning the parameters of a logistic circuit can be reduced to a logistic regression problem and is therefore convex. Learning logistic circuit structure is reduced to a simple local search problem using primitives from the probabilistic circuit learning literature introduced in the last chapter.

We run experiments on standard image classification benchmarks (MNIST and Fashion) and achieve accuracy higher than much larger multiple-layer-perceptions (the most classic neural network architecture) and even convolutional neural networks with an order of magnitude more parameters. For example, logistic circuits obtain 99.4% accuracy on MNIST. Compared to other tractable learners on MNIST, and the state-of-the-art discriminative SPN learner in particular [PVS18], our logistic circuit learner cuts the error rate by a factor of three and a factor of 25 compared to the best generative learner for classification [ABG15]). Furthermore, we show our learner is highly data efficient, managing to still learn well with limited data.

This chapter proceeds as follows. Section 5.2 introduces the syntax and semantics of logistic circuits. Sections 5.3 and 5.4 describe our parameter and structure learning algorithms, which Section 5.5 evaluates empirically. Section 5.6 elaborates on the connection with tractable generative models, after which we conclude with discussion about related and future work.



(a) A logistic circuit

(b) Weights and classification probabilities for select examples

A	B	C	D	$g_r(ABCD)$	$\Pr(Y = 1 \mid ABCD)$
1	0	1	1	-3.1	4.31%
0	1	1	0	1.9	86.99%
1	1	1	0	5.8	99.70%

Figure 5.1: A logistic circuit with example classifications.

5.2 Representation

This section introduces the logistic circuit representation.

5.2.1 Logistic Circuits

This paper proposes *logistic circuits* for classification. Syntactically, they are logical circuits where every AND is decomposable and every OR is deterministic. However, logistic circuits further associate real-valued parameters $\theta_1, \ldots, \theta_m$ with the m input wires to every OR gate. For example, the root OR node in Figure 5.1a associates parameters -2.6 and -5.8 with its two inputs.

To give semantics to logistic circuits, we first characterize how a particular complete assignment $\mathbf{x}$ (one data example) propagates through the circuit.

Definition 4 (Boolean Circuit Flow)**.** *Consider a deterministic OR gate n. The Boolean flow $f(n, \mathbf{x}, c)$ of a complete assignment $\mathbf{x}$ between parent n and child c is*

$$f(n, \mathbf{x}, c) = \begin{cases} 1 & \textit{if } \mathbf{x} \models c \\ 0 & \textit{otherwise} \end{cases}$$

For example, under the assignment $A = 0$, $B = 1$, $C = 1$, $D = 0$, the root node in Figure 5.1a has a Boolean circuit flow of 0 with its left child and 1 with its right child. Note that the determinism property guarantees that under every OR gate, for a given example $\mathbf{x}$, at most one wire has a flow of 1, and the rest has a flow of 0.

We are now ready to define the logistic circuit semantics.

Definition 5 (Logistic Circuit Semantics)**.** *A logistic circuit node n defines the following weight function $g_n(\mathbf{x})$.*

- *If n is a leaf (input) node, then $g_n(\mathbf{x}) = 0$.*

– *If n is an AND gate with children $c_1, \ldots, c_m$, then*

$$g_n(\mathbf{x}) = \sum_{i=1}^{m} g_{c_i}(\mathbf{x}).$$

– *If n is an OR gate with (child node, wire parameter) inputs $(c_1, \theta_1), \ldots, (c_m, \theta_m)$, then*

$$g_n(\mathbf{x}) = \sum_{i=1}^{m} f(n, \mathbf{x}, c_i) \cdot \left(g_{c_i}(\mathbf{x}) + \theta_i\right).$$

At root node r with weight function $g_r(\mathbf{x})$, the logistic circuit defines the posterior distribution on class variable Y as

$$\Pr(Y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp\left(-g_r(\mathbf{x})\right)}. \tag{5.1}$$

Using Boolean circuit flow, this definition essentially collects all the parameters on wires with flow 1 that reach the root, in order to then make a prediction. This is illustrated in Figure 5.1a by coloring red the gates and wires whose parameters and weight function are propagated upward for the example assignment $A = 0$, $B = 1$, $C = 1$, $D = 0$. The logistic circuit in Figure 5.1a defines the same posterior predictions as the table in Figure 5.1b. Specifically, for the example assignment, the weight function simply sums the parameters colored in red: $-5.8 + 2.3 + 3.9 + 1.5 = 1.9$. We then apply the logistic function (Eq. 5.1) to get the classification probability

$$\Pr(Y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-1.9)} = 86.99\%.$$

5.2.2 Real-Valued Data

The semantics given so far assume Boolean inputs $\mathbf{x}$, which is a rather restrictive assumption and prohibits many machine learning applications. Next, we augment the logistic circuit semantics such that they can classify examples with continuous variables.

We interpret real-valued variables $q \in [0, 1]$ as parameterizing an (independent) Bernoulli distribution (cf. [XZF18]). Each continuous variable represents the probability of the corresponding Boolean random variable X. For example, with $\mathbf{q}$ setting $A = 0.4$, $B = 0.8$, $C = 0.2$, and

$D = 0.7$, the probability of $\neg A \wedge D$ would be $(1 - 0.4) \cdot 0.7 = 0.42$. The same distribution defines a probability for each logical sentence, and therefore each node in the logistic circuit. This allows us to generalize Boolean flow as follows.

Definition 6 (Probabilistic Circuit Flow)**.** *Consider a deterministic OR gate n. Let $\mathbf{q}$ be a vector of probabilities, one for each variable in $\mathbf{X}$. The probabilistic flow $f(n, \mathbf{q}, c)$ of vector $\mathbf{q}$ between parent n and child c is*

$$f(n, \mathbf{q}, c) = \Pr_{\mathbf{q}}(c \mid n) = \frac{\Pr_{\mathbf{q}}(c \wedge n)}{\Pr_{\mathbf{q}}(n)} = \frac{\Pr_{\mathbf{q}}(c)}{\Pr_{\mathbf{q}}(n)},$$

where $\Pr_{\mathbf{q}}(.)$ is the fully-factorized distribution where each variable in $\mathbf{X}$ has the probability assigned by $\mathbf{q}$.

Logistic circuit semantics now support continuous data (after normalizing to $[0, 1]$), simply by replacing Boolean flow with probabilistic flow in Definition 5. Note that probabilistic circuit flow has Boolean circuit flow as a special case, when $\mathbf{q}$ happens to be binary. Furthermore, due to the determinism and decomposability properties, the probabilities in Definition 6 can be computed efficiently, together with all probabilistic circuit flows and weight functions in the logistic circuit. We defer the discussion of these computational details to Section 5.3.4. In the rest of this paper, we will abuse notation and have $\mathbf{x}$ refer to Boolean inputs as well as continuous inputs $\mathbf{q}$ interchangeably.

5.3 Parameter Learning

A natural next question is how to learn logistic circuit parameters from complete data, for a fixed given circuit structure (structure learning is discussed in Section 5.4). Furthermore, we ask whether those learned parameters are guaranteed to be optimal, globally minimizing a loss function. We address these questions by showing how parameter learning can be reduced to logistic regression on a modified set of features, owing to logistic circuits' strong properties.

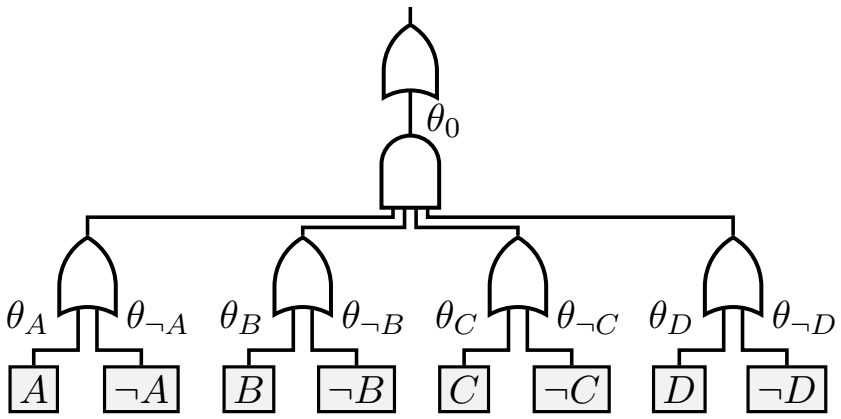


Figure 5.2: Logistic regression represented as a logistic circuit.

5.3.1 Special Cases

Before presenting the general reduction, we briefly discuss two special cases that establish some intuition.

5.3.1.1 Linear Weight Functions

Consider a vanilla logistic regression model on input variables (features) $\mathbf{X}$. Does there exist an equivalent logistic circuit with the same weight function? For sample $\mathbf{x}$, logistic regression with parameters $\boldsymbol{\theta}$ would have weight function $\mathbf{x} \cdot \boldsymbol{\theta}$. Following Definition 5, we obtain such a simple weight function (linear in the input variables) by placing OR gates over complementary pairs of literals and associating a θ parameter which each wire (see Figure 5.2).[1] A large parent AND gate collects these variable-wise weights into a single linear sum. Finally, an OR gate at the root adds the bias term regardless of the input.

Proposition 7. *For each classical logistic regression model, there exists an equivalent logistic circuit model.*

[1]The negated variable inputs and parameters $\theta_{\neg X}$ are redundant, but we keep them for the sake of consistency. Alternatively, we can fix $\theta_{\neg X} = 0$ for all X to remove this redundancy.

5.3.1.2 Boolean Flow Indicators

Next, let us consider a special case that makes no assumptions about circuit structure, but that requires the inputs to be fully binary. Such a circuit would have Boolean flows through every wire. Instead of working with the input variables $\mathbf{X}$, we can introduce new features that are indicator variables, telling us how the example propagates through the circuit, and which wires have a Boolean flow that reaches the circuit root. The circuit flows (indicators) decide which parameters are summed into the weight function; this process has been implicitly revealed in Figure 5.1a. By introducing such indicators, we can always obtain a linear weight function of composite features that are extracted from sample $\mathbf{x}$. Next, we generalize this idea of introducing wire features to arbitrary logistic circuits.

5.3.2 Reduction to Logistic Regression

We will now consider the most general case, with continuous input data and no assumptions on the circuit structure.

Proposition 8. *Any logistic circuit model can be reduced to a logistic regression model over a particular feature set.*

Corollary 3. *Logistic circuit cross-entropy loss is convex.*

To prove Proposition 8, we need to rewrite the classification distribution in Definition 5 as follows.

$$\Pr(Y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp(-\mathbb{x} \cdot \boldsymbol{\theta})}.$$

Here, $\mathbb{x}$ is some vector of features extracted from the raw example $\mathbf{x}$. This feature vector can only depend on $\mathbf{x}$; not on the parameters $\boldsymbol{\theta}$. Thus, the fundamental question is whether we can decompose $g_n(\mathbf{x})$ into $\mathbb{x} \cdot \theta$ for all nodes n. We prove this to be true by induction:

- Base case: n is a leaf (input) node. It is obvious g_n can be expressed as $\mathbb{x} \cdot \theta$ since g_n always equals 0.

– Induction step: assume g of all the nodes under node n can be expressed as $\mathbb{x} \cdot \theta$. We need to consider two cases:

1. If n is an AND gate having (w.l.o.g.) two children, prime p and sub s. Given $g_p = \mathbb{x}_p \cdot \theta_p$ and $g_s = \mathbb{x}_s \cdot \theta_s$,

$$\begin{aligned} g_n &= \mathbb{x}_p \cdot \theta_p + \mathbb{x}_s \cdot \theta_s \\ &= \begin{bmatrix} \mathbb{x}_p \\ \mathbb{x}_s \end{bmatrix} \cdot \begin{bmatrix} \theta_p \\ \theta_s \end{bmatrix}. \end{aligned}$$

2. If n is an OR gate with (child node, wire parameter) inputs $\{(c_1, \theta_1), \ldots, (c_m, \theta_m)\}$. Given $g_{c_i} = \mathbb{x}_{c_i} \cdot \theta_{c_i}$,

$$\begin{aligned} g_n &= \sum_i f(n, \mathbf{x}, c_i) \cdot (\mathbb{x}_{c_i} \cdot \theta_{c_i} + \theta_i) \\ &= \begin{bmatrix} f(n, \mathbf{x}, c_1) \cdot \mathbb{x}_{c_1} \\ f(n, \mathbf{x}, c_1) \\ \vdots \\ f(n, \mathbf{x}, c_m) \cdot \mathbb{x}_{c_m} \\ f(n, \mathbf{x}, c_m) \end{bmatrix} \cdot \begin{bmatrix} \theta_{c_1} \\ \theta_1 \\ \vdots \\ \theta_{c_m} \\ \theta_m \end{bmatrix}. \end{aligned}$$

Note that this proof holds true regardless of whether the input sample $\mathbf{x}$ is binary or real-valued. With this proof, it is obvious that learning the parameters of a logistic circuit is equivalent to logistic regression on features $\mathbb{x}$. We refer readers to [Ren05] for a detailed proof that logistic regression is convex.

Given this correspondence, any convex optimization technique can now be brought to bear on the problem of learning the parameters of a logistic circuit. In particular, we use stochastic gradient descent for this task.

5.3.3 Global Circuit Flow Features

In the proof of Proposition 8, features $\mathbb{x}$ are computed recursively by induction. However, it is not clear what these features represent, and how they are connected to the input data. In this subsection, we assign semantics to those extracted features. They are the *global circuit flow* of the observed example through the circuit. Global circuit flow is defined with respect to the root of a logistic circuit.

Definition 7 (Global Circuit Flow)**.** *Consider a logistic circuit over variables* $\mathbf{X}$ *rooted at OR gate* r*. The global circuit flow* $f_r(n, \mathbf{x}, c)$ *of input* $\mathbf{x}$ *between parent* n *and child* c *is defined inductively as follows. The global circuit flow between root* r *and its child* c *is the (local) probabilistic circuit flow:* $f_r(r, \mathbf{x}, c) = f(r, \mathbf{x}, c)$*. Then, for any node* n *with parents* $v_1, \ldots, v_m$*, we have that*

- *if* n *is an AND gate, global flow from child* c *is*

$$f_r(n, \mathbf{x}, c) = \sum_{i=1}^{m} f_r(v_i, \mathbf{x}, n),$$

- *if* n *is an OR gate, global flow from child* c *is*

$$f_r(n, \mathbf{x}, c) = f(n, \mathbf{x}, c) \cdot \sum_{i=1}^{m} f_r(v_i, \mathbf{x}, n).$$

The red wires in Figure 5.1a have a global circuit flow of 1 for the given Boolean input. In general, global circuit flow assigns a continuous probability value to each wire. Based on global circuit flow, we postulate the following alternative semantics for logistic circuits.

Definition 8 (Logistic Circuit Alternative Semantics)**.** *Let* $\mathcal{W}$ *be the set of all wires* (n, θ, c) *between OR gates* n *and children* c *with parameters* θ*. Then, a logistic circuit rooted at node* r *defines the weight function*

$$g_r(\mathbf{x}) = \sum_{(n,\theta,c)\in\mathcal{W}} f_r(n, \mathbf{x}, c) \cdot \theta.$$

Note that the definition of global circuit flows, as well as our alternative semantics, follow a top-down induction. In contrast, the original semantics in Definition 5 follow a bottom-up induction. We resolve this discrepancy next.

Proposition 9. *The features $\mathbb{x}$ constructed in the proof of Proposition 8 are equivalent to global flows $f_r(n, \mathbf{x}, c)$.*

Corollary 4. *The bottom-up semantics of Definition 5 and the top-down semantics of Definition 8 are equivalent.*

In the following, we prove this proposition by induction.

- Base case: the inputs of the root r are either leaf nodes or AND gates whose inputs are leaf nodes. By definition, for the root's input wires, their local circuit flow equals their global circuit flow. According to the decomposition matrix of g_n in the proof of Proposition 8, the features associated with the root's input wires are equivalent to their local circuit flow. By transitivity, we prove logistic circuits' features are equivalent to its global circuit flow vector in the base case.

- Induction step: assume the proposition holds for all OR gates in a given logistic circuit except the root r. Again, the root's inputs can be either leaf nodes or AND gates. It is obvious that for the root's input wires, their associated features are equivalent to their global circuit flow, as this has been proven in the base case. So we only need to focus on the wires of the sub logistic circuits rooted on those AND gates. The inputs to those AND gates can either be leaf nodes or OR gates. As the wires between AND gates and their leaf children do not have parameters, the correctness of the proposition does not get affected by them. We can narrow our focus again. Now let us consider an OR gate n, which is an input to some of those aforementioned AND gates $\{e_1, \ldots, e_m\}$. By our induction assumption, its features are equivalent to the global circuit flows defined with respect to n; in other words, $\mathbb{x}_n = f_n$. After propagating $\mathbb{x}_n$ upwards to the root, we get $\sum_{i=1}^{m} f(r, \mathbf{x}, e_1) \cdot \mathbb{x}_n$. The sum of the global flow on all output wires of n is $F_r(n) = \sum_{i=1}^{m} f(r, \mathbf{x}, e_1)$. Since $F_r(n)$ is propagated throughout the whole sub logistic circuit rooted at n, the global circuit flow in this sub logistic circuit with respect to the root r is $F_r(n) \cdot f_n = \sum_{i=1}^{m} f(r, \mathbf{x}, e_1) \cdot f_n$. Therefore, the constructed features are equivalent to the global circuit flows.

Recall that without parameters, a logistic circuit is simply a logical circuit, which means that gates in a logistic circuit have real meaning: they correspond to some logical sentence. Hence, the values of global circuit flow features $\mathbb{x}$ correspond to probabilities of these logical sentences according to the input vector $\mathbf{x}$. This provides us with a precious opportunity to assign meaning to the features learned by logistic circuits. We will revisit this point in Section 5.5.4, where we also visualize some global circuit flow features.

Algorithm 5: Node probabilities from a real-valued sample $\mathbf{x}$.

Input: A vector of probabilities $\mathbf{x}$.

Result: $\Pr_{\mathbf{x}}(n)$: the node probability of n for $\mathbf{x}$.

```
for n in the circuit's nodes, children before parents do
    if n is a leaf with variable X then
        if n is X then
            Pr_x(n) = x(X)
        else
            Pr_x(n) = 1 − x(X)
    else if n is an AND gate then
        Pr_x(n) := 1
        for c in inputs of n do
            Pr_x(n) *= Pr_x(c)
    else
        // n is an OR gate
        Pr_x(n) := 0
        for c in inputs of n do
            Pr_x(n) += Pr_x(c)
```

Algorithm 6: Features $\mathbb{x}$ from a real-valued sample $\mathbf{x}$.

Input: Node probabilities $\Pr_{\mathbf{x}}(\cdot)$.

Result: Real-valued feature vector $\mathbb{x}$.

for n *in all nodes, parents before children* **do**
    $v(n) := 0$
$v(\text{root}) := 1$
for n *in all non-leaf nodes, parents before children* **do**
    if n *is an OR gate* **then**
        for c *in inputs of* n **do**
            $\mathbb{x}(n, c) := v(n) \cdot \Pr_{\mathbf{x}}(c) \ / \ \Pr_{\mathbf{x}}(n)$
            $v(c) \mathrel{+}= \mathbb{x}(n, c)$
    else
        `// n is an AND gate`
        for c *in inputs of* n **do**
            $v(c) \mathrel{+}= v(n)$

5.3.4 Computing Global Flow Features Efficiently

While logistic circuit parameter learning is convex, we would like to also guarantee that the required feature computation is tractable. This section discusses efficient methods to calculate global flow features $\mathbb{x}$ (i.e., $f_r(n, \mathbf{x}, c)$) from training samples $\mathbf{x}$ offline, before parameter learning.

As is clear from Definition 6, circuit flows make extensive use of node probabilities. We design our computation to consist of two parts, and dedicate the first part to the calculation of node probabilities. The first part is a bottom-up linear pass over the circuit starting with leaf nodes whose probabilities are directly provided by the input sample. The second part makes use of these node probabilities to calculate the global circuit flow features in linear time. It is a top-down implementa-

tion of the recursion in Definition 7. Note that these computations correspond to the partial derivative computations used in arithmetic circuits for the purpose of probabilistic inference [Dar03].

Our algorithm is completely compatible with fast vector arithmetic: instead of inputting one single sample each time, one can directly supply the algorithms with a vector of samples (e.g., a mini batch), and this yields significant speedups.

We calculate node probabilities in a bottom-up induction on the structure of the sentence; see Algorithm 5.

- Base case: n is a leaf (input) node. The node probability is directly defined in $\mathbf{x}$: $\Pr_{\mathbf{x}}(n) = \mathbf{x}(X)$ if n is X; $\Pr_{\mathbf{x}}(n) = 1 - \mathbf{x}(X)$ if n is $\neg X$ (lines 2-6 in Algorithm 5).
- Induction step: given that the node probabilities for all the leaves have been calculated, we move upward to intermediate nodes and the root, where there are two cases.
  * n is an AND gate with inputs $\{c_1, \ldots, c_m\}$. Since in a logistic circuit every AND gate is decomposable, by independence of the conjuncts, $\Pr_{\mathbf{x}}(n) = \prod_{i=1}^{m} \Pr_{\mathbf{x}}(c_i)$ (lines 7-10 in Algorithm 5).
  * n is an OR gate with input nodes $\{c_1, \ldots, c_m\}$. Since every OR gate is deterministic, the probabilistic events defined at each child within the same OR parent do not intersect with each other. By mutual exclusivity, $\Pr_{\mathbf{x}}(n) = \sum_i \Pr_{\mathbf{x}}(c_i)$ (lines 11-14 in Algorithm 5).

Node probabilities $\Pr_{\mathbf{x}}(\cdot)$ are used in Algorithm 6 to obtain the final feature vector.

To compute the final feature vector, We perform a top-down pass starting from the root OR gate; see Algorithm 6. After visiting an OR gate, the method first calculates its associated global circuit flows from its inputs; see Line 7 in Algorithm 6. These newly calculated global flows then get passed down and are accumulated on those child gates for later use on the descendent gates (Line 8). After visiting an AND gate, there is no new global circuit flow to be calculated. Hence,

the algorithm directly accumulates the flows passed to those AND gates to their children (Line 10-11).

Note that instead of inputing one single sample at a time, one can directly supply Algorithm 5 and 6 with a vector of samples. Our proposed calculation method is completely compatible with matrix operations, and by doing so, one can expect a large speedup.

5.4 Structure Learning

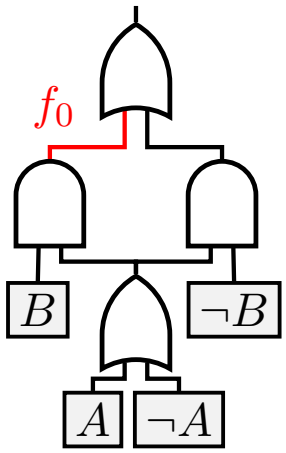
This section presents an algorithm to learn a compact logical circuit structure for logistic circuits from data. For simplicity of designing the primitive operations, we assume AND gates always have two inputs (prime and sub).

5.4.1 Learning Primitive

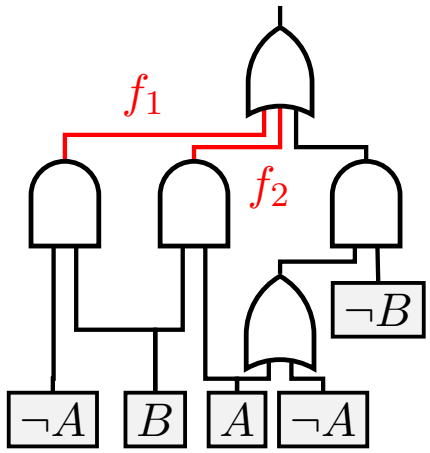
The split operation was first introduced to modify the structure of PSDD circuits [LBV17]. We adopt it here with minor changes[2] as the primitive operation for our structure learning algorithm. Splitting an AND gate happens by imposing two additional constraints that are *mutually exclusive* and *exhaustive*, in particular by making two opposing variable assignments. Executing a split creates partial copies of the gate and some of its decedents. Furthermore, one can choose to duplicate additional nodes up to a fixed depth (3 in our experiments). We refer readers to [LBV17] for further details on the algorithm for executing splits.

Splits are ideal primitives to change the classifier induced by a logistic circuit: they directly affect the circuit flows (see Figure 5.3). By imposing constraints on AND gates, splits alter the node probabilities associated with the affected AND gates. Following Definition 6, the circuit flows on the wires out of those AND gates adapt accordingly. While Figure 5.3 focuses on the immediately affected wires, the effect of a split on circuit flows can propagate downward for several levels,

[2]Compared to the splits in LearnPSDD [LBV17], we do not limit constraints to be on primes.



(a) Before split of f_0 on A



(b) After split of f_0 on A

A	B	f_0	f_1	f_2
1	1	1	0	1
0	1	1	1	0
0.5	0.6	0.6	0.30	0.30
0.4	0.8	0.8	0.48	0.32

(c) Circuit flow before and after the split.

Figure 5.3: A split changes the circuit flow.

depending on the depth of node duplication. Still the effects of a split on both the structure of a logistic circuit and the circuit flows are very local and contained in the sub-circuit rooted at the OR parent of the split AND gate. However, its effect on the parameters is global. Once a split is executed, the whole parameter set needs to be re-trained.

5.4.2 Learning Algorithm

The overall structure learning algorithm for logistic circuits, built on top of the split operation, proceeds as follows. Iteratively, one split is executed to change the structure, followed by parameter learning. We only consider single-variable split constraints and first select which AND gate to split, followed by a selection of which variable to split on.

When using gradient descent, one hopes that the parameter on the AND gate output consistently has its partial derivatives pointing in the same direction for all training examples. This will steadily push the parameter to a large magnitude. If this is not the case, we will use splits to alter the flow of examples through the circuit. Specifically, those AND gates whose associated output parameter has a large variance of its partial derivative (that is, the derivative of the loss function w.r.t. that parameter) requires splitting for the parameters to improve. We simply select the AND gate whose output parameter has the highest training variance.

Given an AND gate to split, we consider candidate variables X to execute the split with. We construct two sets of training examples that affect this node: in one group, each example is weighted by the marginal probability of X; in the other, with the marginal probability of $\neg X$. Next, we calculate the within-group weighted variances of the partial derivatives. The variable with the smallest weighted variances gets picked, as this suggests the split will introduce new parameters with gradients that align in one direction.

5.5 Empirical Evaluation

In this section, we empirically evaluate the competitiveness of our learner on three aspects: classification accuracy, model complexity, and data efficiency.[3] Moreover, we visualize the most important active feature with regards to the given sample to provide local interpretation for why the learned logistic circuit makes such classification.

[3]Open-source code and experiments are available at `https://github.com/UCLA-StarAI/LogisticCircuit`.

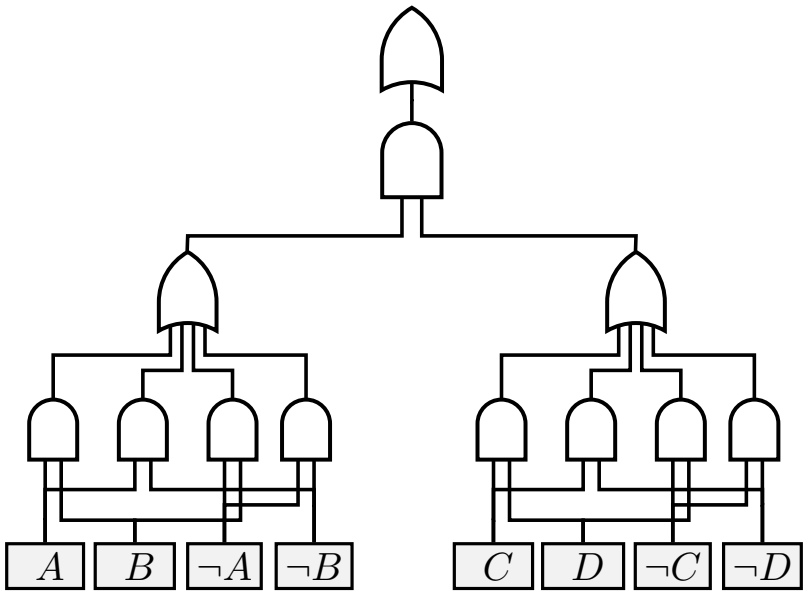


Figure 5.4: Initial structure of logistic circuits with 4 pixels.

5.5.1 Setup & Data Preprocessing

We choose MNIST and FASHION[4] as our testbeds. Since logistic circuits are intended for binary classification, we use the standard "one vs. rest" approach to construct an ensemble multi-class classifier such that our method can be evaluated on these two datasets. When running the binary logistic circuit, we transform pixels that are smaller than their mean plus 0.05 standard deviation to 0 and the rest to 1. When running the real-valued version, we transform pixels to $[0, 1]$ by dividing them by 255. The learned structure with the highest F1 score on validation after 48 hours of running is used for evaluation. All experiments are run on single CPUs.

All experiments in this paper start with an initial structure where every pixel has two corresponding leaf nodes, one for the pixel being true and the other false. Pixels are paired up by AND gates; an AND gate is created for every joint assignment to the pair. AND gates for the same pair share one OR gate parent. After this, OR gates are paired with AND gates and every AND gate is connected to its own OR gate parent until we reach the root. Figure 5.4 is an example of the initial

[4]A dataset of Zalando's images, intended as a more challenging drop-in replacement of MNIST [XRV17].

Table 5.1: Classification accuracy of logistic circuits along with commonly used existing models.

Accuracy % on Dataset	MNIST	FASHION
Baseline: Logistic Regression	85.3	79.3
Baseline: Kernel Logistic Regression	97.7	88.3
Random Forest	97.3	81.6
3-layer MLP	97.5	84.8
RAT-SPN [PVS18]	98.1	89.5
SVM with RBF Kernel	98.5	87.8
5-Layer MLP	99.3	89.8
Logistic Circuit (binary)	97.4	87.6
Logistic Circuit (real-valued)	99.4	91.3
CNN with 3 conv layers	99.1	90.7
Resnet [HZR16]	99.5	93.6

structure with 4 pixels. Note that our structure learning algorithm is compatible with other initial structures and one can create ad-hoc ones tailored to different applications.

The reported kernel logistic regression is based on the pixel n-grams implemented in Vowpal Wabbit [LLS07]. The reported random forest has 500 decision trees. The reported SVM with RBF Kernel uses hyper-parameters $C = 8, \gamma = 0.05$ on MNIST and $C = 4, \gamma = 25$ on Fashion. The reported 3-layer MLP has layers of size 784-1000-500-250-10 respectively. The reported 5-layer MLP has layers of size 784-1000-500-250-2000-250-10 respectively. The reported CNN with 3 convolutional layers uses 3-by-3 padded filters in the convolutional layers.

5.5.2 Classification Accuracy

Table 5.1 summarizes the classification accuracy on test data. Learning a logistic circuit on the binary data is on par with a 3-layer MLP; the real-valued version outperforms 5-layer MLPs and

Table 5.2: Number of parameters of logistic circuits in context with existing SGD-based models, when achieving the classification accuracy reported in Table 5.1.

Number of Parameters	MNIST	Fashion
Baseline: Logistic Regression	<1K	<1K
Baseline: Kernel Logistic Regression	1,521 K	3,930K
Logistic Circuit (real-valued)	182K	467K
Logistic Circuit (binary)	268K	614K
3-layer MLP	1,411K	1,411K
RAT-SPN [PVS18]	8,500K	650K
CNN with 3 conv layers	2,196K	2,196K
5-Layer MLP	2,411K	2,411K
Resnet [HZR16]	4,838K	4,838K

even CNNs with 3 convolutional layers. The fact that logistic circuits achieve better accuracy than CNNs is surprising, since logistic circuits do not use convolutions, which are specifically designed to exploit image invariances.

In addition, we would like to emphasis our comparison with two of the baselines. As parameter learning of logistic circuits is equivalent to logistic regression, one can view structure learning of logistic circuits as a process of constructing composite features from raw samples. The significant improvement over standard logistic regression demonstrates the effectiveness of our method in extracting valuable features; using kernel logistic regression can only partially bridge the gap in performance, yet as shown later, it does so at the cost of introducing many more parameters.

Besides extracting informative composite features, from our experience of running the reported experiments, the structure learning process could also impose implicit regularization on the learned logistic circuits. To speed up the whole learning pipeline, parameter learning is early stopped before parameters are converged to the global optimum. This means the final obtained parameters could

Table 5.3: Comparison of logistic circuits with MLPs when trained with different percentages of the dataset.

Accuracy % with % of Training Data	MNIST			FASHION		
	100%	10%	2%	100%	10%	2%
5-layer MLP	99.3	**98.2**	94.3	89.8	86.5	80.9
CNN with 3 Conv Layers	99.1	98.1	95.3	90.7	87.6	83.8
Logistic Circuit (Binary)	97.4	96.9	94.1	87.6	86.7	83.2
Logistic Circuit (Real-Valued)	**99.4**	97.8	**96.1**	**91.3**	**87.8**	**86.0**

be different from what one would get through a complete parameter learning on the final structure. This implicit regularization tends to help logistic circuits generalize in unseen data points. We hypothesize the embedded early stopping causes some subsets of the parameters less tuned than the others, which renders the learned logistic circuits less sensitive to nuance features. Yet, the exact mechanism of this implicit regularization remains to be fully investigated, and a clear understanding of it could lead to a better-designed structure learning algorithm in the future.

Refocusing on logistic circuits' classification accuracy, we also want to call attention to our comparison with RAT-SPN, the current state of the art in discriminative learning for any dialect of probabilistic circuits. SPN is another form of circuit representation, with less restrictive structure. Parameter learning in SPN is not convex and generally requires other techniques such as EM or non-convex optimization. The empirical observation that our method achieves significantly better classification accuracy than RAT-SPN demonstrates that in structure learning, imposing more restrictions on the model's structural syntax may be beneficial. The syntactic restriction of logistic circuits requires decomposability and determinism; without them, convex parameter learning does not appear to be possible. As structure learning is built on top of parameter learning, a well-behaved parameter learning loss with a unique optimum can provide more informative guidance about how to adapt the structure, leading to a more competitive structure learning algorithm overall.

5.5.3 Model Complexity & Data Efficiency

Table 5.2 summarizes the size of all compared models when achieving the reported accuracy. We can conclude that logistic circuits are significantly smaller than the alternatives, despite attaining higher accuracy.

We design the next set of experiments to specifically investigate how well our structure learning algorithm performs under the setting where the number of training samples is limited. We have two additional sets of experiments, where only 2% and 10% of the original training data is supplied. Table 5.3 summarizes the performance in this limited-data setting. We mainly compare against a 5-layer MLP and CNN with 3 convolutional layers, whose performance is on par with our method under the full-data setting. As summarized in Table 5.3, except on MNIST with 10% training samples, real-valued logistic circuits achieve the best classification accuracy. Moreover, in both versions of logistic circuits, when the available training samples are reduced from 100% to 2%, the accuracy only drops by around 3% when evaluating on MNIST; around 5% on Fashion. In contrast, a much larger drop occurs for 5-layer MLP and CNN. Specifically, MLP's accuracy drops by 5% (9%) while CNN's accuracy drops by 4% (7%) on MNIST (Fashion). This small magnitude of accuracy decrease illustrates how data efficient our proposed structure learning algorithm is.

Except on MNIST with 10% training samples, real-valued logistic circuits achieve the best classification accuracy. From a top-down perspective, each OR gate of a logistic circuit presents a weighted choice between its wires. Hence, one can view a logistic circuit as a decision diagram. Under this perspective, splits refine OR gates' branching rules. As each branching rule naturally applies to multiple samples, we hypothesize that the splits selected by our structure learning algorithm reflect the general conditional feature information present in the dataset.

5.5.4 Local Explanation

Next, we aim to share some insights about how to explain the learned logistic circuit. Specifically, we investigate the question: "Why does the logistic circuit classify a given sample $\mathbf{x}$ as y?" Since

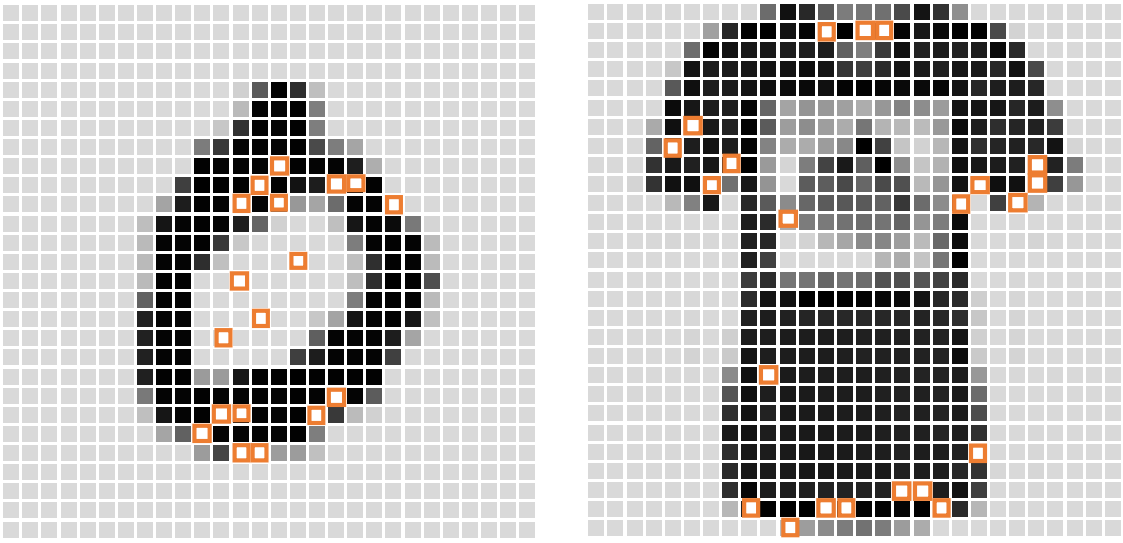


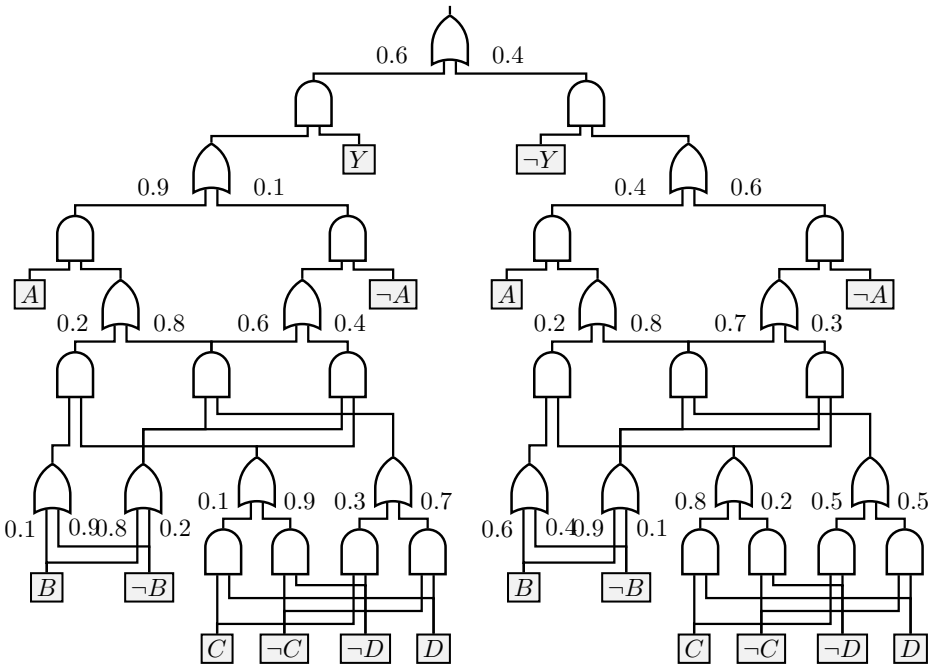
Figure 5.5: Visualization of the single compositional feature that contributes most to the classification probability with regards to the input image. Features are marked in orange. Left: a digit 0 from MNIST. Right: a t-shirt from Fashion.

any logistic circuit can be reduced to a logistic regression classifier, we can easily find the active global flow feature that contributes most to the given sample's classification probability. That is, the feature that maximizes $\mathbb{x} \cdot \theta$. We visualize one such feature for MNIST data and one for Fashion in Figure 5.5 by marking the variables used in their corresponding logical sentences.

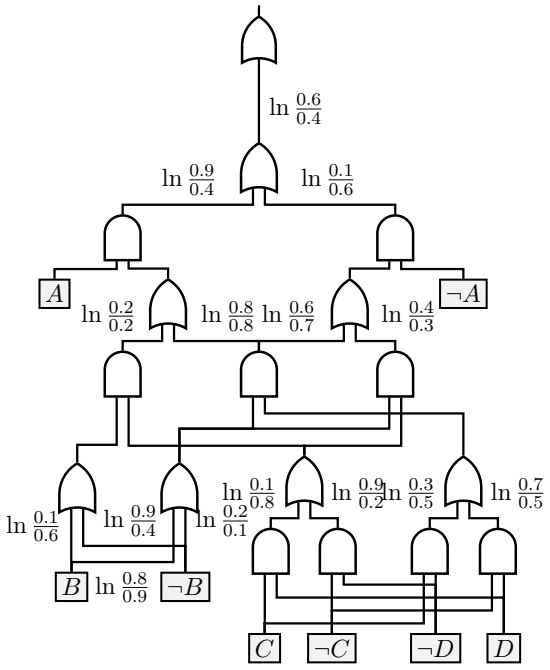
5.6 Connection to Probabilistic Circuits

Recall our logistic circuit definition (Definition 5) looks very similar to our probabilistic circuit definition (Definition 1). Except from adding a logistic circuit over the output of the circuit root, the main difference between these two definitions is that the two's parameters clearly have different meaning. However, is there some inherent connection between those parameters. In this section, we study this problem.

Figure 5.6a shows a probabilistic circuit for the joint distribution $\Pr(Y, A, B, C, D)$. This tractable circuit language is a relaxation of PSDDs [KAD14] and a specific type of SPN [PD11]



(a) Probabilistic circuit for joint distribution $\Pr(Y, A, B, C, D)$



(b) Logistic circuit for $\Pr(Y = 1 \mid A, B, C, D)$

Figure 5.6: A probabilistic circuit with parallel structures under class variable Y and its equivalent logistic circuit for predicting Y.

where determinism holds throughout. It is also a type of arithmetic circuit. We are now ready to connect logistic and probabilistic circuits. It is well known that logistic regression is the discriminative counterpart of a naive Bayes generative model [NJ02]. A similar correspondence holds between our logistic and probabilistic circuits.

Proposition 10. *Consider a probabilistic circuit whose structure is of the form $(Y \wedge \alpha) \vee (\neg Y \wedge \beta)$, where sub-circuits α and β are structurally identical. Then, there exists an equivalent logistic circuit for the conditional probability of Y in the probabilistic circuit. Moreover, this logistic circuit has structure $\vee\alpha$ and its parameters can be computed in closed form as log-ratios of probabilistic circuit probabilities.*

We first depict this correspondence intuitively in Figure 5.6. The logistic circuit has the same structure as the two halves of the probabilistic circuit, and its parameters are computed from the probabilistic circuit probabilities. The distributions $\Pr(Y = 1 \mid A, B, C, D)$ represented by the circuits in Figures 5.6a and 5.6b are identical.

Formal Correspondence Next, we present the formal proof of this correspondence for binary $\mathbf{x}$. Recall that in our circuits, only the input wires of OR gates are parameterized. Let $\mathcal{W}_\delta$ be the set that contains all these wires in circuit δ:

$$\mathcal{W}_\delta = \{(n, c) \mid c \text{ is a gate with parent OR gate } n\}.$$

After expanding the equations in Definition 1 and following the top-down definition of global circuit flow (i.e., following Definition 7), one finds that the joint distribution induced by a probabilistic circuit δ can be rewritten as

$$\Pr{}_\delta(\mathbf{x}) = \prod_{(n,c)\in\mathcal{W}_\delta} f_\delta(n, \mathbf{x}, c) \cdot \theta^\delta_{(n,c)}.$$

We will exploit this finding in the derivation of the conditional distribution induced by the probabilistic circuit $\gamma = (Y \wedge \alpha) \vee (\neg Y \wedge \beta)$.

$$\begin{aligned}
&\Pr_\gamma(Y = 1 \mid \mathbf{x}) \\
&= \frac{\Pr_\gamma(Y=1)\Pr_\alpha(\mathbf{x})}{\Pr_\gamma(Y=0)\Pr_\beta(\mathbf{x}) + \Pr(Y=1)\Pr_\alpha(\mathbf{x})} \\
&= \frac{1}{1 + \frac{\Pr_\gamma(Y=0)\Pr_\beta(\mathbf{x})}{\Pr_\gamma(Y=1)\Pr_\alpha(\mathbf{x})}} \\
&= \frac{1}{1 + \frac{\Pr_\gamma(Y=0)\prod_{(n,c)\in\mathcal{W}_\beta} f_\beta(n,\mathbf{x},c)\theta^\beta_{(n,c)}}{\Pr_\gamma(Y=1)\prod_{(n,c)\in\mathcal{W}_\alpha} f_\alpha(n,\mathbf{x},c)\theta^\alpha_{(n,c)}}}
\end{aligned}$$

As stated in Proposition 10 and shown in Figure 5.6, sub-circuits α and β share the same structure. Therefore, we can further simplify this equation as follows.

$$\begin{aligned}
&\Pr_\gamma(Y = 1 \mid \mathbf{x}) \\
&= \frac{1}{1 + \frac{\Pr_\gamma(Y=0)}{\Pr_\gamma(Y=1)}\prod_{(n,c)\in\mathcal{W}_\alpha} f_{\vee\alpha}(n,\mathbf{x},c)\frac{\theta^\beta_{(n,c)}}{\theta^\alpha_{(n,c)}}} \\
&= \frac{1}{1 + \exp\left[-g(\mathbf{x})\right)]} = \Pr_{\vee\alpha}(Y = 1 \mid \mathbf{x})
\end{aligned}$$

where

$$g(\mathbf{x}) = \log\frac{\Pr_\gamma(Y=1)}{\Pr_\gamma(Y=0)} + \sum_{(n,c)\in\mathcal{W}_\alpha} f_{\vee\alpha}(n,\mathbf{x},c)\log\frac{\theta^\alpha_{(n,c)}}{\theta^\beta_{(n,c)}} \tag{5.2}$$

$$= \theta^{\vee\alpha}_{root} + \sum_{(n,c)\in\mathcal{W}_\alpha} f_{\vee\alpha}(n,\mathbf{x},c)\cdot\theta^{\vee\alpha}_{(n,c)}. \tag{5.3}$$

The transformation from Equation 5.2 to 5.3 expresses the logistic circuit parameters as the log-ratios of probabilistic circuit probabilities. For example, the class priors captured in the output wires of α and β are now combined as a log-ratio to form the bias term for $\vee\alpha$, expressed by the root parameter.

This proof also provides us with a new perspective to understand the semantics of the learned parameters. The parameters represent the log-odds ratio of the features given different classes. Note that by Bayes' theorem, a naive Bayes model would derive its induced distribution in a sequence

of steps similar to the ones above, resulting in Equation 5.2. Given this correspondence, one can also view our proposed structure learning method as a way to construct meaningful features for a naive Bayes classifier. We know that after training, naive Bayes classifiers are equivalent to logistic regression classifiers (as in Equation 5.3).

5.7 Cooperation with Probabilistic Circuits

The counterpart relationship demonstrated and proved in the previous section turns out to be critical in improving logistic circuits' robustness. In particular, consider we are given a trained logistic circuit, but when predicting for new examples, some of the input features are missing. Missing features are pervasive in real-world applications, due to assorted reasons; for example, the noisy nature of the environment, unreliable or broken sensors, difficulty of gathering data, etc. [DSX10]. Yet, not just for logistic circuit, missing features at the prediction time can still be a challenge for most classifiers and regressors.

The existing common approach to deal with missing features is to substitute the missing features with one or several imputed values. However, to obtain the imputed values, one tends to make unrealistic assumptions about the feature distribution. One popular assumption is that features are independent from one another, which is too ideal and rarely holds. A more principled approach is to directly compute the expected prediction of the classifier with respect to the feature distribution. Formally, it is

$$\mathop{\mathbb{E}}_{\mathbf{x}_m \sim \Pr(\mathbf{X}_m \mid \mathbf{x}_o)} \left[\mathcal{F}(\mathbf{x}_m \mathbf{x}_o) \right].$$

Here, $\mathcal{F}$ is a predictor (i.e., classifier or regressor) and we partition features into those that are missing $\mathbf{X}_m$ and those are given and observed $\mathbf{X}_o$. This approach is guaranteed to be unbiased, as it considers all possible (partial) assignments to the missing features altogether. The major bottleneck that prevents this principled approach to be widely adopted is its computational cost. For most classifiers and regressors, it is intractable [KLC19]. However, when $\mathcal{F}$ is a logistic circuit and we use a probabilistic circuit to represent the feature distribution, this computation becomes

tractable under certain conditions. To be specific, two conditions need to be satisfied: (i) the logistic circuit is used as a regressor (i.e., no logistic function on the top); (ii) the logistic circuit and the probabilistic circuit share the same vtree [KCL19]. Essentially, we require a probabilistic circuit and its discriminative counterpart. For such a pair of logistic and probabilistic circuit, this expected prediction can be exactly computed in quadratic time with respect to the size of the circuits. It is a recursion algorithm, and the intuition behind it and how it "breaks down" the computation to intermediate nodes is similar to the intersection divergence algorithm presented in Chapter 3. Indeed and again, the two circuits sharing the same vtree is the property that enables a polynomial time recursive decomposition.

5.8 Related Work

[GD12] proposed the first parameter learning algorithm for discriminative SPNs, using MPE inference as a sub-routine. Without the support of the determinism property, parameter learning of general SPNs is a relatively harder question than its logistic circuit counterpart, since it is non-convex. [ABG15] boost the accuracy of SPNs on MNIST to 97.6% by extracting more representative features from raw inputs based on the Hilbert-Schmidt independence measure. [PVS18] further improved the classification ability of SPNs by drastically simplifying SPN structure requirements and utilizing a loss objective that hybrids cross-entropy (discriminative learning) with log-likelihood (generative learning).

[RL16] developed a discriminative structure learning algorithm for arithmetic circuits. The method updates the circuit that represents a corresponding conditional random field (CRF) model by adding features conditioned on arbitrary evidence to the model. This work further relaxes decomposability and smoothness properties of ACs for a more compact representation. However, it targets the setting where there are a large number of output variables, not single-variable classification.

All the aforementioned literature conforms to a common trend of abandoning properties of the

chosen circuit representations for easier structure learning and better prediction accuracy. They argue that those special syntactic restrictions complicate the learning process. On the contrary, this paper chooses perhaps the most structure-restrictive circuit as the target representation. Instead of relaxing the target representation's syntactical requirements, our proposed method fully leverages the valuable properties that stem from these restrictions, and in particular convexity.

5.9 Discussion

We have presented logistic circuits, a novel circuit-based classification model with convex parameter learning, and a simple structure learning procedure based on local search. Logistic circuits outperform much larger classifiers and perform well in a limited data regime. Compared to other symbolic, circuit-based approaches, logistic circuits present a leap in performance on image classification benchmarks. Promising future efforts to further improve logistic circuits' performance include support for convolution, parameter tying, and structure sharing.

CHAPTER 6

Improving Semi-Supervised Learning by Reasoning about Constraints

After demonstrating that circuit representations are amenable to both generative and discriminative learning from data in the previous two chapters, we switch gear again and revisit the symbolic dimension of circuit representations. Again, circuit representations will be used to represent logic constraints. However, this time constraints are not imposed on the input feature space. How to efficiently reason and effectively learn in that type of structure space has been well discussed in Chapter 3 and Chapter 4. In this chapter, we focus on the constraints imposed on the representations' output space. Furthermore, instead of using circuit representations as a standalone learner, this chapter is dedicated to the question about how we can leverage them to incorporate symbolic knowledge into deep neural networks. We specifically choose semi-supervised image classification as the target task to show that even a small addition of supposedly simple symbolic knowledge can yield a significant boost to the prediction accuracy.

6.1 Background

The work that will be introduced in this chapter is largely motivated by two observations. First, knowledge is ubiquitous, yet sometimes can be easily overlooked. For example, for classification, people tend to dismiss the constraint embedded in the task setup: every example has one and exactly one class. This is a simple constraint, yet can be surprisingly powerful in the semi-supervised learning setting. We aim to demonstrate through this case study that regardless of how seemingly

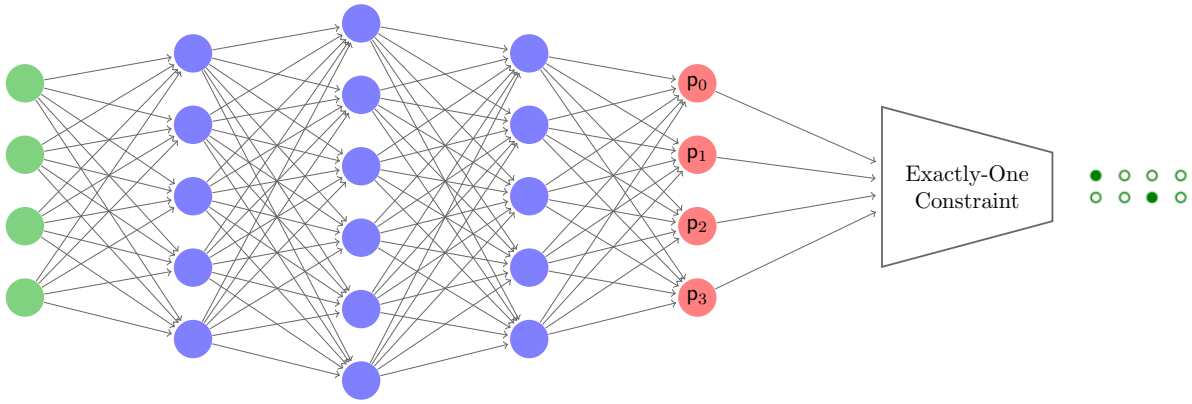


Figure 6.1: Outputs of a neural network feed into semantic loss functions for exactly-one constraint. This simple yet often-overlooked constraint surprisingly yields significant improvement for semi-supervised classification tasks.

trivial the knowledge is, it may still be able to help the learner to go a long way when used in the right domains. Secondly, most neuro-symbolic approaches aim to simulate or learn symbolic reasoning in an end-to-end deep neural network, or capture symbolic knowledge in a vector-space embedding. This choice is partly motivated by the need for smooth *differentiable* models; adding symbolic reasoning code (e.g., SAT solvers) to a deep learning pipeline destroys this property. Unfortunately, while making reasoning differentiable, the precise logical meaning of the knowledge is often lost.

Considering these two observations, we take a distinctly unique approach, and tackle the problem of differentiable but sound logical reasoning from first principles. Starting from a set of intuitive axioms, we derive the differentiable *semantic loss* which captures how well the outputs of a neural network match a given constraint. This function precisely captures the *meaning* of the constraint, and is independent of its *syntax*.

Given this is the first time we use neural networks in this dissertation, we introduce an additional notation: the output row vector of a neural net is denoted $\mathbf{p}$. Given we focus on semi-supervised image classification tasks, we assume each value in $\mathbf{p}$ represents the probability of an output and falls in $[0, 1]$. Figure 6.1 illustrates how exactly-one constraint is passed into a deep neu-

ral network. The constraint states that for a set of indicators $\mathbf{X} = \{X_1, \ldots, X_n\}$, one and exactly one of those indicators must be true, with the rest being false. This is enforced through a logical constraint α by conjoining sentences of the form $\neg X_1 \vee \neg X_2$ for all pairs of variables (at most one variable is true), and a single sentence $X_1 \vee \cdots \vee X_n$ (at least one variable is true).

6.2 Semantic Loss

In this section, we formally introduce semantic loss. We begin by giving the definition and our intuition behind it. This definition itself provides all of the necessary mechanics for enforcing constraints. We also show that semantic loss is not just an arbitrary definition, but rather is defined uniquely by a set of intuitive assumptions. After stating the assumptions formally, we then provide an axiomatic proof of the uniqueness of semantic loss in satisfying these assumptions.

6.2.1 Definition

Semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$ is a function of a sentence α in propositional logic, defined over variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, and a vector of probabilities p for the same variables $\mathbf{X}$. Element p_i denotes the predicted probability of variable X_i, and corresponds to a single output of the neural net. For example, the semantic loss between the one-hot constraint from the previous section, and a neural net output vector p, is intended to capture how close the prediction p is to having exactly-one output set to true (i.e. 1), and all others set to false (i.e. 0), regardless of which output is correct. The formal definition of this is as follows:

Definition 9 (Semantic Loss)**.** *Let* p *be a vector of probabilities, one for each variable in* $\mathbf{X}$*, and let* α *be a sentence over* $\mathbf{X}$*. The semantic loss between* α *and* p *is*

$$\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \propto -\log \sum_{\mathbf{x} \models \alpha} \prod_{i : \mathbf{x} \models X_i} \mathsf{p}_i \prod_{i : \mathbf{x} \models \neg X_i} (1 - \mathsf{p}_i).$$

Intuitively, the semantic loss is proportional to a negative logarithm of the probability of generating a state that satisfies the constraint, when sampling values according to p. Hence, it is the

self-information (or "surprise") of obtaining an assignment that satisfies the constraint [Jon79].

6.2.2 Derivation from First Principles

In this section, we begin with a theorem stating the uniqueness of semantic loss, as fixed by a series of axioms. These axioms present the semantics behind this loss function and hence justifies its name. To be more specific, those axioms shed light into how this loss function connects to and retains the logical semantics of the constraints, besides being computed with respect to the logical constraint.

Theorem 2 (Uniqueness)**.** *The semantic loss function in Definition 9 satisfies all axioms in the following and is the only function that does so, up to a multiplicative constant.*

Recall logical implication intuitively describes which constraint is harder to be satisfied than the other when presented with two logical sentences. To retain logical meaning, we first need to guarantee that the magnitude of this loss function is related to the logical implication relationship. Towards this, we postulate that semantic loss is monotone in the order of implication.

Axiom 1 (Monotonicity)**.** *If* $\alpha \models \beta$*, then semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{p})$ *for any vector* p*.*

Intuitively, as we add stricter requirements to the logical constraint, going from β to α and making it harder to satisfy, semantic loss cannot decrease. For example, when β enforces the output of an neural network to encode a subtree of a graph, and we tighten that requirement in α to be a path, semantic loss cannot decrease, since every path is also a tree and any solution to α is a solution to β. A direct consequence is that if α and β are logically equivalent, they must incur exactly the same loss for the same probability vector p.

Proposition 11 (Semantic Equivalence)**.** *If* $\alpha \equiv \beta$*, then semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{p})$ *for any vector* p*.*

Given true sentences are the weakest constraints, if we do not want them to incur any loss, semantic loss much be non-negative given its monotonicity property.

Given semantic loss is to guide the probability vector p to more satisfy the constraint α, we would also like to guarantee that when p satisfies the constraint, semantic loss is zero. For example, when our constraint α requires that the output vector encodes an arbitrary total ranking, and the vector correctly represents a single specific total ranking, there is no semantic loss. From Definition 9, it is clear p can only have either 1 or 0 in this situation. Essentially, p is now a complete binary assignment to the output vector. And to avoid confusion, we use $\mathbf{x}$ to highlight this fact.

Proposition 12 (Satisfaction)**.** *If* $\mathbf{x} \models \alpha$*, then semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x}) = 0$.

As a special case, logical literals (X or $\neg X$) constrain a single variable to take on a value, and thus play a role similar to the labels used in supervised learning. Such constraints require an even tighter correspondence: semantic loss must act like a classical loss function (i.e., cross entropy).

Axiom 2 (Label-Literal Correspondence)**.** *Semantic loss of a single literal is proportionate to the cross-entropy loss for the equivalent data label:* $\mathrm{L}^{\mathrm{s}}(X, p) \propto -\log(p)$ *and* $\mathrm{L}^{\mathrm{s}}(\neg X, p) \propto -\log(1 - p)$.

To retain logical meaning is not enough, it also needs to be differentiable such that it can work well with gradient-based learning and optimization methods.

Axiom 3 (Differentiability)**.** *For any fixed* α*, semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$ *is monotone in each probability in* p*, continuous and differentiable.*

We have now presented the most important axioms of semantic loss. Yet, additional axioms are required to allow us to prove the following form of the semantic loss for a state $\mathbf{x}$. For conciseness, those additional axioms are only presented in the next subsection.

Lemma 2. *For state* $\mathbf{x}$ *and vector* p*, we have* $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathsf{p}) \propto -\sum_{i:\mathbf{x}\models X_i} \log \mathsf{p}_i - \sum_{i:\mathbf{x}\models\neg X_i} \log(1 - \mathsf{p}_i)$.

Lemma 2 falls short as a full definition of semantic loss for arbitrary sentences. The next subsection makes the notion of semantic loss precise by stating one additional axiom. It is based on the observation that the state loss of Lemma 2 is proportionate to a log-probability. In particular,

it corresponds to the probability of obtaining state $\mathbf{x}$ after independently sampling each X_i with probability p_i. We have now derived the semantic loss function from first principles, and arrived at Definition 9. Moreover, we can show that Theorem 2 holds - that it is the only choice of such a loss function.

6.2.3 Details of the Derivation

This subsection provides further details on our axiomatization of semantic loss. We detail here a complete axiomatization of semantic loss, which will involve restating some axioms and propositions from the previous sections. Readers with interest are highly encouraged to continue reading. However, skipping it would not affect the understanding of the rest of this chapter either.

The first axiom says that there is no loss when the logical constraint α is always true (it is a logical tautology), independent of the predicted probabilities p.

Axiom 4 (Truth)**.** *Semantic loss of a true sentence is zero:* $\forall \mathsf{p}, \mathrm{L}^{\mathrm{s}}(\mathit{true}, \mathsf{p}) = 0$.

Next, when enforcing two constraints on disjoint sets of variables, we want the ability to compute semantic loss for the two constraints separately, and sum the results for their joint semantic loss.

Axiom 5 (Additive Independence)**.** *Let* α *be a sentence over* $\mathbf{X}$ *with probabilities* p*. Let* β *be a sentence over* $\mathbf{Y}$ *disjoint from* $\mathbf{X}$ *with probabilities* q*. Semantic loss between sentence* $\alpha \wedge \beta$ *and the joint probability vector* $[\mathsf{p}\,\mathsf{q}]$ *decomposes additively:* $\mathrm{L}^{\mathrm{s}}(\alpha \wedge \beta, [\mathsf{p}\,\mathsf{q}]) = \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) + \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{q})$.

It directly follows from Axioms 4 and 5 that the probabilities of variables that are not used on the constraint do not affect semantic loss. Proposition 13 formalizes this intuition.

Proposition 13 (Locality)**.** *Let* α *be a sentence over* $\mathbf{X}$ *with probabilities* p*. For any* $\mathbf{Y}$ *disjoint from* $\mathbf{X}$ *with probabilities* q*, semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, [\mathsf{p}\,\mathsf{q}]) = \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$.

Proof. Follows from the additive independence and truth axioms. Set $\beta = \text{true}$ in the additive independence axiom, and observe that this sets $\mathrm{L}^{\mathrm{s}}(\beta, \mathsf{q}) = 0$ because of the truth axiom. □

We have shown semantic loss is monotone in the order of implication in the earlier subsection, here we finish proving it is non-negative.

Proposition 14 (Non-Negativity)**.** *Semantic loss is non-negative.*

Proof. Because $\alpha \models$ true for all α, the monotonicity axiom implies that $\forall \mathsf{p}, \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\text{true}, \mathsf{p})$. By the truth axiom, $\mathrm{L}^{\mathrm{s}}(\text{true}, \mathsf{p}) = 0$, and therefore $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq 0$ for all choices of α and p. □

A state $\mathbf{x}$ is equivalently represented as a data vector, as well as a logical constraint that enforces a value for every variable in $\mathbf{X}$. When both the constraint and the predicted vector represent the same state (for example, $X_1 \wedge \neg X_2 \wedge X_3$ vs. $[1\,0\,1]$), there should be no semantic loss.

Axiom 6 (Identity)**.** *For any state* $\mathbf{x}$*, there is zero semantic loss between its representation as a sentence, and its representation as a deterministic vector:* $\forall \mathbf{x}, \mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) = 0$.

One can also use the axioms above to derive that any vector satisfying the constraint must incur zero loss; see Proposition 12 in the previous subsection. We repeat this proposition first, followed by our proof.

Proposition 15 (Satisfaction)**.** *If* $\mathbf{x} \models \alpha$*, then semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x}) = 0$.

Proof of Proposition 15. The monotonicity axiom specializes to say that if $\mathbf{x} \models \alpha$, we have that $\forall \mathsf{p}, \mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$. By choosing p to be $\mathbf{x}$, this implies $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) \geq \mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x})$. From the identity axiom, $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) = 0$, and therefore $0 \geq \mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x})$. Proposition 14 bounds the loss from below as $\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x}) \geq 0$. □

Next, we have the symmetry axioms.

Axiom 7 (Value Symmetry)**.** *For all* p *and* α*, we have that* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\bar{\alpha}, 1 - \mathsf{p})$ *where* $\bar{\alpha}$ *replaces every variable in* α *by its negation.*

Axiom 8 (Variable Symmetry)**.** *Let* α *be a sentence over* $\mathbf{X}$ *with probabilities* p*. Let* π *be a permutation of the variables* $\mathbf{X}$*, let* $\pi(\alpha)$ *be the sentence obtained by replacing variables* x *by* $\pi(x)$*, and let* $\pi(\mathsf{p})$ *be the corresponding permuted vector of probabilities. Then,* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\pi(\alpha), \pi(\mathsf{p}))$.

The value and variable symmetry axioms together imply the equality of the multiplicative constants in the label-literal duality axiom for all literals.

Lemma 3. *There exists a single constant K such that $\mathrm{L}^{\mathrm{s}}(X, p) = -K \log(p)$ and $\mathrm{L}^{\mathrm{s}}(\neg X, p) = -K \log(1-p)$ for any literal x.*

Proof. Value symmetry implies that $\mathrm{L}^{\mathrm{s}}(X_i, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\neg X_i, 1 - \mathsf{p})$. Using label-literal correspondence, this implies $K_1 \log(p_i) = K_2 \log(1 - (1 - p_i))$ for the multiplicative constants K_1 and K_2 that are left unspecified by that axiom. This implies that the constants are identical. A similar argument based on variable symmetry proves equality between the multiplicative constants for different i. □

Finally, this allows us to prove the following form of semantic loss for a state $\mathbf{x}$, which has been shown towards the end of the previous subsection as well.

Lemma 4. *For state $\mathbf{x}$ and vector p, we have $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathsf{p}) \propto -\sum_{i:\mathbf{x} \models X_i} \log \mathsf{p}_i - \sum_{i:\mathbf{x} \models \neg X_i} \log(1 - \mathsf{p}_i)$.*

Proof of Lemma 4. A state $\mathbf{x}$ is a conjunction of independent literals, and therefore subject to the additive independence axiom. Each literal's loss in this sum is defined by Lemma 3. □

The following and final axiom requires that semantic loss is proportionate to the logarithm of a function that is additive for mutually exclusive sentences.

Axiom 9 (Exponential Additivity)**.** *Let α and β be mutually exclusive sentences (i.e., $\alpha \wedge \beta$ is unsatisfiable), and let $f^s(K, \alpha, \mathsf{p}) = K^{-\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})}$. Then, there exists a positive constant K such that $f^s(K, \alpha \vee \beta, \mathsf{p}) = f^s(K, \alpha, \mathsf{p}) + f^s(K, \beta, \mathsf{p})$.*

We are now able to state and prove the main uniqueness theorem.

Theorem 3 (Uniqueness)**.** *The semantic loss function in Definition 9 satisfies all axioms presented above and is the only function that does so, up to a multiplicative constant.*

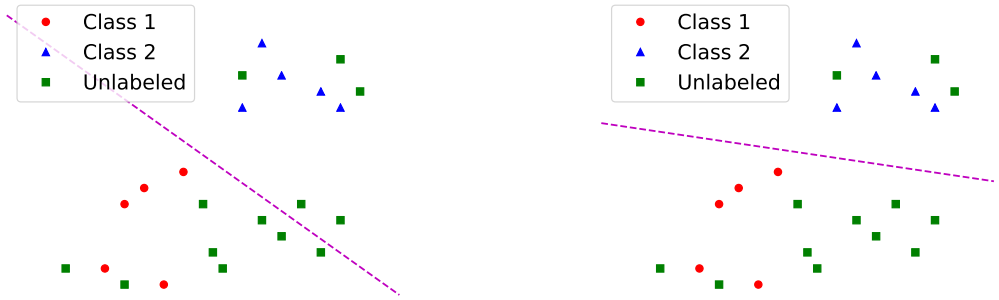
Proof of Theorem 3. The truth axiom states that $\forall \mathsf{p}, f^s(K, \text{true}, \mathsf{p}) = 1$ for all positive constants K. This is the first Kolmogorov axiom of probability. The second Kolmogorov axiom for $f^s(K, ., \mathsf{p})$ follows from the additive independence axiom of semantic loss. The third Kolmogorov axiom (for the finite discrete case) is given by the exponential additivity axiom of semantic loss. Hence, $f^s(K, ., \mathsf{p})$ is a probability distribution for some choice of K, which implies the definition up to a multiplicative constant. □

6.3 A Case Study in Exactly-One Constraint

The most straightforward constraint that is ubiquitous in classification is mutual exclusion over one-hot-encoded outputs. That is, for a given example, exactly one class and therefore exactly one binary indicator must be true. The machine learning community has made great strides on this task, due to the invention of assorted deep learning representations and their associated regularization terms [KSH12, HZR16]. Many of these models take large amounts of labeled data for granted, and big data is indispensable for discovering accurate representations [HTF09]. To sustain this progress and alleviate the need for more labeled data, there is a growing interest in utilizing unlabeled data to augment the predictive power of classifiers [SE17, BBM04]. This section shows why semantic loss naturally qualifies for this task.

6.3.1 Illustrative Examples

To illustrate the benefit of semantic loss in the semi-supervised setting, we begin our discussion with a small toy example. Consider a binary classification task; see Figure 6.2. Ignoring the unlabeled examples, a simple linear classifier learns to distinguish the two classes by separating the labeled examples (Figure 6.2a). However, the unlabeled examples are also informative, as they must carry some properties that give them a particular label. This is the crux of semantic loss for semi-supervised learning: a model must confidently assign a consistent class even to unlabeled data. Encouraging the model to do so results in a more accurate decision boundary (Figure 6.2b).



(a) Trained w/o semantic loss

(b) Trained with semantic loss

Figure 6.2: Binary classification toy example: a linear classifier without and with semantic loss.

6.3.2 Method

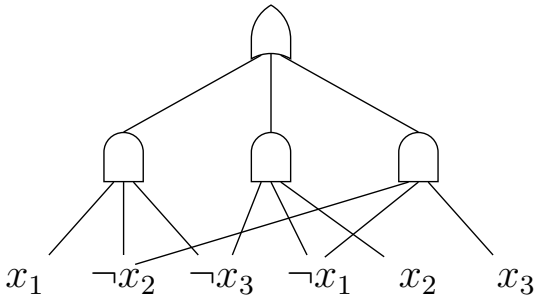
Our proposed method intends to be generally applicable and compatible with any feedforward neural net. Semantic loss is simply another regularization term that can directly be plugged into an existing loss function. More specifically, with some weight w, the new overall loss becomes

$$\text{existing loss} + w \cdot \text{semantic loss}.$$

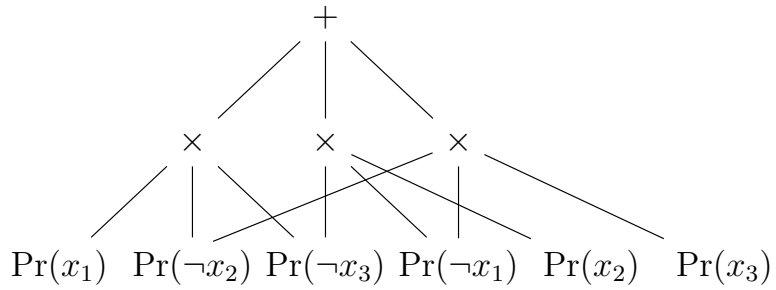
Concretely, for the exactly-one constraint used in n-class classification, semantic loss reduces to

$$\mathrm{L}^{\mathrm{s}}(\text{exactly-one}, \mathsf{p}) \propto -\log \sum_{i=1}^{n} \mathsf{p}_i \prod_{j=1, j\neq i}^{n} (1 - \mathsf{p}_j),$$

where values p_i denote the probability of class i as predicted by the neural net. To efficiently compute semantic loss, one needs to leverage tractable reasoning; see Chapter 3 for more details about how to efficiently evaluate a circuit representation. Tractable reasoning can reduce the time complexity to compute semantic loss for the exactly-one constraint from $O(n^2)$ to $O(n)$; see Figure 6.3 to recall how circuits are transformed into an efficient computation graph, Since exactly-one constraint is simple, this computation speedup is not significantly noticeable. In general, for arbitrary constraints α, tractable reasoning has to be deployed to make sure semantic loss does not incur expensive computation overhead.



(a) A compiled decomposable and deterministic circuit for the exactly-one constraint with 3 variables.



(b) The corresponding computation graph for the exactly-one constraint with 3 variables.

Figure 6.3: An illustration how to compactly represent exactly-one constraint for three output variables as circuits. To efficiently compute the likelihood to satisfy this constraint, we transform AND gates to multiplication and OR gates to summation. Note this transformation has also been introduced in Chapter 3.

6.4 Empirical Evaluation

We evaluate semantic loss in the semi-supervised setting by comparing it with several competitive models on three benchmark datasets, namely, MNIST, FASHION, and Cifar-10.[1]

6.4.1 Experiment Setup

As most semi-supervised learners build on a supervised learner, changing the underlying model significantly affects the semi-supervised learner's performance. For comparison, we add semantic loss to almost identical base models used in ladder nets [RBH15], which currently achieves state-of-the-art results on semi-supervised MNIST and CIFAR-10 [Kri09]. Specifically, the MNIST base model is a fully-connected multilayer perceptron (MLP), with layers of size 784-1000-500-250-250-250-10. On CIFAR-10, it is a 10-layer convolutional neural network (CNN) with 3-by-3 padded filters. After every 3 layers, features are subject to a 2-by-2 max-pool layer with strides of 2. Furthermore, we use ReLu [NH10], batch normalization [IS15], and Adam optimization [KB15] with a learning rate of 0.002. Table 6.1 shows the slight architectural difference between the CNN used in ladder nets and ours. The major difference lies in the choice of ReLu. Note that we add standard padded cropping to preprocess images and an additional fully connected layer at the end of the model, neither is used in ladder nets. We only make those slight modification so that the baseline performance reported by [RBH15] can be reproduced.

Validation sets are used for tuning the weight associated with semantic loss, the only hyper-parameter that causes noticeable difference in performance for our method. We perform a grid search over $\{0.001, 0.005, 0.01, 0.05, 0.1\}$ to find the optimal value. Empirically, 0.005 always gives the best or nearly the best results and we report its results on all experiments. For the sake of fairness, we subject ladder nets to a small-scale parameter tuning as well in case its performance is more volatile.

[1]The code to reproduce all experiments in this chapter can be found at `https://github.com/UCLA-StarAI/Semantic-Loss/`.

For all our experiments, we use the standard 10,000 held-out test examples provided in the original datasets and randomly pick 10,000 from the standard 60,000 training examples (50,000 for CIFAR-10) as validation set. For values of N that depend on the experiment, we retain N randomly chosen labeled examples from the training set, and remove labels from the rest. We balance classes in the labeled samples to ensure no particular class is over-represented. Images are preprocessed for standardization and Gaussian noise is added to every pixel ($\sigma = 0.3$).

6.4.2 Experiment Results

In the following we report our results on three benchmark datasets.

MNIST The permutation invariant MNIST classification task is commonly used as a testbed for general semi-supervised learning algorithms. This setting does not use any prior information about the spatial arrangement of the input pixels. Therefore, it excludes many data augmentation techniques that involve geometric distortion of images, as well as convolutional neural networks.

When evaluating on MNIST, we run experiments for 20 epochs, with a batch size of 10. Experiments are repeated 10 times with different random seeds. Table 6.2 compares semantic loss to three baselines and state-of-the-art results from the literature. The first baseline is a purely supervised MLP, which makes no use of unlabeled data. The second is the classic self-training method for semi-supervised learning, which operates as follows. After every 1000 iterations, the unlabeled examples that are predicted by the MLP to have more than 95% probability of belonging to a single class, are assigned a pseudo-label and become labeled data.

Additionally, we construct a third baseline by replacing the semantic loss term with the entropy regularizor described in [GB05] as a direct comparison for semantic loss. With the same amount of parameter tuning, we find that using entropy achieves an accuracy of 96.27% with 100 labeled examples, and 98.32% with 1000 labelled examples, both are slightly worse than the accuracies reached by semantic loss. Furthermore, to our best knowledge, there is no straightforward method to generalize entropy loss to the settings of complex constraints, where semantic loss is clearly

Table 6.1: Specifications of CNNs in Ladder Net and our proposed method.

CNN in Ladder Net	CNN in this paper
Input 32×32 RGB image	
	Resizing to 36×36 with padding; Cropping Back
Whitening Contrast Normalization Gaussian Noise with std. of 0.3	
3×3 conv. 96 BN LeakyReLU	3×3 conv. 96 BN ReLU
3×3 conv. 96 BN LeakyReLU	3×3 conv. 96 BN ReLU
3×3 conv. 96 BN LeakyReLU	3×3 conv. 96 BN ReLU
2×2 max-pooling stride 2 BN	
3×3 conv. 192 BN LeakyReLU	3×3 conv. 192 BN ReLU
3×3 conv. 192 BN LeakyReLU	3×3 conv. 192 BN ReLU
3×3 conv. 192 BN LeakyReLU	3×3 conv. 192 BN ReLU
2×2 max-pooling stride 2 BN	
3×3 conv. 192 BN LeakyReLU	3×3 conv. 192 BN ReLU
1×1 conv. 192 BN LeakyReLU	3×3 conv. 192 BN ReLU
1×1 conv. 10 BN LeakyReLU	1×1 conv. 10 BN ReLU
Global meanpool BN	
	Fully connected BN
10-way softmax	

Table 6.2: MNIST. Previously reported test-set accuracies followed by baselines and semantic loss results ($\pm$ stddev).

Accuracy % with # of used labels	100	1000	ALL
AtlasRBF [PRA14]	91.9 ($\pm$0.95)	96.32 ($\pm$0.12)	98.69
Deep Generative [KMJ14]	96.67($\pm$0.14)	97.60 ($\pm$0.02)	99.04
Virtual Adversarial [MMK16]	97.67	98.64	99.36
Ladder Net [RBH15]	**98.94** ($\pm$0.37)	**99.16** ($\pm$0.08)	99.43 ($\pm$0.02)
Baseline: MLP, Gaussian Noise	78.46 ($\pm$1.94)	94.26 ($\pm$0.31)	99.34 ($\pm$0.08)
Baseline: Self-Training	72.55 ($\pm$4.21)	87.43 ($\pm$3.07)	
Baseline: MLP with Entropy Regularizer	96.27 ($\pm$0.64)	98.32 ($\pm$0.34)	99.37 ($\pm$0.12)
MLP with Semantic Loss	98.38 ($\pm$0.51)	98.78 ($\pm$0.17)	99.36 ($\pm$0.02)

defined and can be easily deployed.

Lastly, we create a fourth baseline by constructing a constraint-sensitive loss term in the style of [HML16], using a simple extension of Probabilistic Soft Logic (PSL) [KBB12]. PSL translates logic into continuous domains by using soft truth values, and defines functions in the real domain corresponding to each Boolean function. This is normally done for Horn clauses, but since they are not sufficiently expressive for our constraints, we apply fuzzy operators to arbitrary sentences instead. We are forced to deal with a key difference between semantic loss and PSL: encodings in fuzzy logic are highly sensitive to the syntax used for the constraint (and therefore violate Proposition 11). We selected two reasonable encodings. The first encoding is:

$$(\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3)$$

The second encoding is:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$$

Both encodings extend to cases whether the number of variables is arbitrary.

The norm functions used for these experiments are as described in [KBB12], with the loss for an interpretation I being defined as follows:

$$x_1 \wedge x_2 = max\{0, I(x_1) + I(x_2) - 1\}$$
$$x_1 \vee x_2 = min\{I(x_1) + I(x_2), 1\}$$
$$\neg x_1 = 1 - I(x_1)$$

The first encoding results in a constant value of 1, and thus could not be used for semi-supervised learning. The second encoding empirically deviates from 1 by < 0.01, and since we add Gaussian noise to the pixels, no amount of tuning was able to extract meaningful supervision. Thus, we do not report these results.

When given 100 labeled examples ($N = 100$), MLP with semantic loss gains around 20% improvement over the purely supervised baseline. The improvement is even larger (25%) compared to self-training. Considering *the only change is an additional loss term*, this result is very encouraging. Comparing to the state of the art, ladder nets slightly outperform semantic loss by 0.5% accuracy. This difference may be an artifact of the excessive tuning of architectures, hyperparameters, and learning rates that the MNIST dataset has been subject to. In the coming experiments, we extend our work to more challenging datasets, in order to provide a clearer comparison with ladder nets. Before that, we want to share a few more thoughts on how semantic loss works. A classical softmax layer interprets its output as representing a categorical distribution. Hence, by normalizing its outputs, softmax enforces the same mutual exclusion constraint enforced in our semantic loss function. However, there does not exist a natural way to extend softmax loss to unlabeled samples. In contrast, semantic loss does provide a learning signal on unlabeled samples, by forcing the underlying classifier to make a decision and construct a confident hypothesis for all data. However, for the fully supervised case ($N =$ all), semantic loss does not significantly affect accuracy. Because the MLP has enough capacity to almost perfectly fit the training data, where the constraint is always satisfied, semantic loss is almost always zero. This is a direct consequence of Proposition 12.

Table 6.3: FASHION. Test accuracy comparison between MLP with semantic loss and ladder nets.

Accuracy % with # of used labels	100	500	1000	ALL
Ladder Net [RBH15]	81.46 ($\pm$0.64)	85.18 ($\pm$0.27)	86.48 ($\pm$0.15)	90.46
Baseline: MLP, Gaussian Noise	69.45 ($\pm$2.03)	78.12 ($\pm$1.41)	80.94 ($\pm$0.84)	89.87
MLP with Semantic Loss	**86.74** ($\pm$0.71)	**89.49** ($\pm$0.24)	**89.67** ($\pm$0.09)	89.81

Table 6.4: CIFAR. Test accuracy comparison between CNN with Semantic Loss and ladder nets.

Accuracy % with # of used labels	4000	ALL
CNN Baseline in Ladder Net	76.67 ($\pm$0.61)	90.73
Ladder Net [RBH15]	79.60 ($\pm$0.47)	
Baseline: CNN, Whitening, Cropping	77.13	90.96
CNN with Semantic Loss	**81.79**	90.92

FASHION The FASHION [XRV17] dataset consists of Zalando's article images, aiming to serve as a more challenging drop-in replacement for MNIST. Arguably, it has not been overused and requires more advanced techniques to achieve good performance. As in the previous experiment, we run our method for 20 epochs, whereas ladder nets need 100 epochs to converge. Again, experiments are repeated 10 times and Table 6.3 reports the classification accuracy and its standard deviation (except for $N =$ all where it is close to 0 and omitted for space).

Experiments show that utilizing semantic loss results in a very large 17% improvement over the baseline when only 100 labels are provided. Moreover, our method compares favorably to ladder nets, except when the setting degrades to be fully supervised. Note that our method already nearly reaches its maximum accuracy with 500 labeled examples, which is only 1% of the training dataset.

CIFAR-10 To show the general applicability of semantic loss, we evaluate it on CIFAR-10. This dataset consists of 32-by-32 RGB images in 10 classes. A simple MLP would not have enough

representation power to capture the huge variance across objects within the same class. To cope with this spike in difficulty, we switch our underlying model to a 10-layer CNN as described earlier. We use a batch size of 100 samples of which half are unlabeled. Experiments are run for 100 epochs. However, due to our limited computational resources, we report on a single trial. Note that we make slight modifications to the underlying model used in ladder nets to reproduce similar baseline performance, which have been explained in Section 6.4.1.

As shown in Table 6.4, our method compares favorably to ladder nets. However, due to the slight difference in performance between the supervised base models, a direct comparison would be methodologically flawed. Instead, we compare the net improvements over baselines. In terms of this measure, our method scores a gain of 4.66% whereas ladder nets gain 2.93%.

6.5 Related Work

Incorporating symbolic background knowledge into machine learning is a long-standing challenge [SMK95]. It has received considerable attention for structured prediction in natural language processing, in both supervised and semi-supervised settings. For example, *constrained conditional models* extend linear models with constraints that are enforced through integer linear programming [CRR08, CSR13]. Constraints have also been studied in the context of probabilistic graphical models [MD08, GGT10]. [KVC14] utilize a circuit language called the *probabilistic sentential decision diagram* to induce distributions over arbitrary logical formulas. They learn generative models that satisfy preference and path constraints [CVD15a, CTD16], which we study in a discriminative setting.

Various deep learning techniques have been proposed to enforce either arithmetic constraints [PKD15, MSF17] or logical constraints [RSR15, HML16, DRR16, SE17, MDR17, DGS17, DSG17] on the output of a neural network. The common approach is to reduce logical constraints into differentiable arithmetic objectives by replacing logical operators with their fuzzy t-norms and logical implications with simple inequalities. A downside of this fuzzy relaxation is that the logical sen-

tences lose their precise meaning. The learning objective becomes a function of the syntax rather than the semantics, whereas our proposed semantic loss still retains the semantic meaning of the constraints it captures. Moreover, these relaxations are often only applied to Horn clauses. One alternative is to encode the logic into a factor graph and perform loopy belief propagation to compute a loss function [NR17], which is known to have issues in the presence of complex logical constraints [SG14].

Finally, the objective of semantic loss to increase the confidence of predictions on unlabeled data is related to information-theoretic approaches to semi-supervised learning [GB05, EA10], and approaches that increase robustness to output perturbation [MMK16]. A key difference between semantic loss and these information-theoretic losses is that semantic loss generalizes to arbitrary logical output constraints that are much more complex.

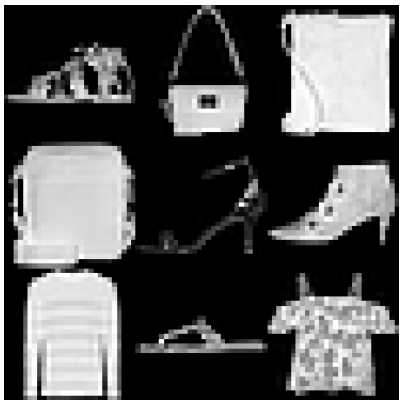
6.6 Discussion

The experiments so far have demonstrated the competitiveness and general applicability of our proposed method on semi-supervised learning tasks. It surpasses the previous state of the art (ladder nets) on FASHION and CIFAR-10, while being close on MNIST. Considering the simplicity of our method, such results are encouraging. Figure 6.4 illustrates the effect of semantic loss on FASHION pictures whose correct label was hidden from the learner. Pictures 6.4a and 6.4b are correctly classified by the supervised base model, and on the first set it is confident about this prediction ($\mathsf{p}_i > 0.8$). Semantic loss rarely diverts the model from these initially correct labels. However, it bootstraps these unlabeled examples to achieve higher confidence in the learned concepts. With this additional learning signal, the model changes its beliefs about Pictures 6.4b. Even on confidently misclassified Pictures 6.4d, semantic loss is able to remedy the mistakes of the base model.

Indeed, a key advantage of semantic loss is that it only requires a simple additional loss term, and thus incurs almost no computational overhead. Conversely, this property makes our method

(a) Confidently Correct

(b) Unconfidently Correct



(c) Unconfidently Incorrect

(d) Confidently Incorrect

Figure 6.4: FASHION pictures grouped by how confidently and correctly the supervised base model classifies them. With semantic loss, the final semi-supervised model predicts all correctly and confidently.

sensitive to the underlying model's performance. Without the underlying predictive power of a strong supervised learning model, we do not expect to see the same benefits we observe here.

Another unique advantage of semantic loss is that it can work with any constraints imposed on the output space of a deep learning model, as long as the constraints can be represented as logical sentences. In this chapter, we have downplayed this advantage, as we would like to highlight the fact that knowledge and constraints are everywhere and can be overlooked. In deriving semantic loss, we have repeated using "path" as an example. In fact, we have solid experiments to demonstrate that by first encoding all valid paths as logical sentences, we can achieve an almost 10-fold improvement in predicting the shortest path on a grid environment compared to standard MLP neural networks. Readers with interest are highly encouraged to go through the original semantic loss paper to have a more complete insight on how deep structured predictions for highly complex output spaces can significantly benefit from semantic loss [XZF18].

An interesting direction for future work is to come up with effective approximations of semantic loss, for settings where even the methods we have described are not sufficient. There are several potential ways to proceed with this, including hierarchical abstractions, relaxations of the constraints, or projections on random subsets of variables.

CHAPTER 7

Conclusion

In this dissertation, towards the goal of symbolic-statistical synthesis, we advocate circuit representations. Circuit representations have been extensively studied in the community of knowledge representation, specifically as a target form to compactly represent logical sentences. This makes them a natural candidate to encode structural knowledge of complex spaces; for example, what is valid and what is not in a domain.

Existing efforts have laid a great foundation on how to parameterize circuits with strong syntactic properties to represent a joint distribution over input features (i.e., random variables). We follow this line and advance the frontier of tractable probabilistic reasoning with circuit representations. Specifically, we study the problem of exactly computing the KL-divergence between two circuits. We invent a recursion algorithm whose time complexity is quadratic with respect to the circuit size for this purpose. Variable ordering is an important syntactic property for circuits. In deriving the algorithm, it is clear that sharing the same variable ordering is the key that makes this computation tractable.

We next study the structure learning problem for probabilistic circuits. Structure learning is motivated by two considerations. First, given circuit representations' delicate syntactic properties, it is infeasible to manually design a circuit structure that induces complex data distribution. Second, to have meaningful probabilistic reasoning that corresponds to real happenings, it is imperative to learn probabilistic circuits from data. With our invented algorithm, we demonstrate that those syntactic properties are not obstructions, but rather assistance in the learning process. They help reduce the structure learning problem to local search. Moreover, by first compiling an initial

structure from domain-specific logical constraints, one can straightforwardly learn over structured spaces, a unique advantage that is rendered possible by symbolic-statistical synthesis.

Given probabilistic circuits are for generative learning, to complete our studies on learning, we further invent the discriminative counterparts of probabilistic circuits. We demonstrate that, thanks to the inherent syntactic properties, their parameter learning is convex optimization. A simple search can induce strong structures from data for those discriminative circuits as well, and their performance on standard image classification benchmarks is on par with deep neural networks.

Lastly, we study a practical question about how to incorporate circuit-based symbolic knowledge into the current deep learning framework. Through reasoning about how likely the deep learning's outputs satisfy the circuit-encoded constraints, we invent a novel semantic loss. Our approach significantly improves deep learning's prediction accuracy in semi-supervised settings and deep structured domains with complex output spaces.

Overall, we demonstrate probabilistic reasoning, learning, and classification are unified for circuit representations through the same syntactic properties. We hope that this dissertation can shine a light on the opportunities of bridging the latest advances in probabilistic reasoning and deep statistical machine learning.

REFERENCES

[ABG15] Tameem Adel, David Balduzzi, and Ali Ghodsi. "Learning the Structure of Sum-Product Networks via an SVD-based Algorithm." *UAI*, pp. 32–41, 2015.

[BBM04] Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. "Integrating constraints and metric learning in semi-supervised clustering." In *ICML*, p. 11. ACM, 2004.

[BDC15] Jessa Bekker, Jesse Davis, Arthur Choi, Adnan Darwiche, and Guy Van den Broeck. "Tractable Learning for Complex Probability Queries." In *NIPS*, Dec 2015.

[Ben18] Rodrigo Benenson. "What is the class of this image?" `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`, 2018.

[BFG96] Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. "Context-specific independence in Bayesian networks." In *UAI*, pp. 115–123, 1996.

[CD13] Arthur Choi and Adnan Darwiche. "Dynamic Minimization of Sentential Decision Diagrams." In *AAAI*, 2013.

[CG07] Anton Chechetka and Carlos Guestrin. "Efficient Principled Learning of Thin Junction Trees." In *NIPS*, pp. 273–280, 2007.

[CKD13] Arthur Choi, Doga Kisa, and Adnan Darwiche. "Compiling Probabilistic Graphical Models using Sentential Decision Diagrams." In *ECSQARU*, pp. 121–132, 2013.

[CRR08] Ming-Wei Chang, Lev-Arie Ratinov, Nicholas Rizzolo, and Dan Roth. "Learning and Inference with Constraints." In *AAAI*, pp. 1513–1518, 2008.

[CSD17] Arthur Choi, Yujia Shen, and Adnan Darwiche. "Tractability in structured probability spaces." In *Advances in Neural Information Processing Systems 30*, 2017.

[CSR13] Kai-Wei Chang, Rajhans Samdani, and Dan Roth. "A Constrained Latent Variable Model for Coreference Resolution." In *EMNLP*, 2013.

[CTD16] Arthur Choi, Nazgol Tavabi, and Adnan Darwiche. "Structured Features in Naive Bayes Classification." In *AAAI*, pp. 3233–3240, 2016.

[CVB] YooJung Choi, Antonio Vergari, and Guy Van den Broeck. "Probabilistic circuits: A unifying framework for tractable probabilistic models. sep 2020." *URL http://starai.cs.ucla.edu/papers/ProbCirc20.pdf*.

[CVD15a] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. "Tractable learning for structured probability spaces: A case study in learning preference distributions." In *IJCAI*, 2015.

[CVD15b] Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. "Tractable Learning for Structured Probability Spaces: A Case Study in Learning Preference Distributions." In *IJCAI*, 2015.

[Dar03] Adnan Darwiche. "A Differential Approach to Inference in Bayesian Networks." *J. ACM*, **50**(3):280–305, May 2003.

[Dar09] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

[DDC08] Adnan Darwiche, Rina Dechter, Arthur Choi, Vibhav Gogate, and Lars Otten. "Results from the Probablistic Inference Evaluation of UAI-08." 2008.

[DDS09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[DGS17] Michelangelo Diligenti, Marco Gori, and Claudio Sacca. "Semantic-based regularization for learning and inference." *JAIR*, **244**:143–165, 2017.

[DKL21] Meihua Dang, Pasha Khosravi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. "Juice: A julia package for logic and probabilistic circuits." In *AAAI (demo track)*, 2021.

[DM02] Adnan Darwiche and Pierre Marquis. "A knowledge compilation map." *Journal of AI Research*, **17**:229–264, 2002.

[DRR16] Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. "Lifted Rule Injection for Relation Embeddings." In *EMNLP*, pp. 1389–1399, 2016.

[DSG17] Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. "Logic Tensor Networks for Semantic Image Interpretation." In *IJCAI*, pp. 1596–1602, 2017.

[DSX10] Ofer Dekel, Ohad Shamir, and Lin Xiao. "Learning to classify with missing and corrupted features." *Machine learning*, **81**(2):149–178, 2010.

[DV15] Aaron Dennis and Dan Ventura. "Greedy Structure Search for Sum-Product Networks." *IJCAI*, 2015.

[EA10] Ayse Erkan and Yasemin Altun. "Semi-Supervised Learning via Generalized Maximum Entropy." In *AISTATS*, volume PMLR, pp. 209–216, 2010.

[Fri98] Nir Friedman. "The Bayesian Structural EM Algorithm." *UAI*, pp. 129–138, 1998.

[GB05] Yves Grandvalet and Yoshua Bengio. "Semi-supervised learning by entropy minimization." In *NIPS*, 2005.

[GD12] Robert Gens and Pedro Domingos. "Discriminative learning of sum-product networks." In *NIPS*, pp. 3239–3247, 2012.

[GD13] Robert Gens and Pedro Domingos. "Learning the structure of sum-product networks." In *ICML*, pp. 873–880, 2013.

[GGT10] Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. "Posterior regularization for structured latent variable models." *JMLR*, **11**(Jul):2001–2049, 2010.

[HGD17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In *ICCV*, 2017.

[HML16] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. "Harnessing deep neural networks with logic rules." In *ACL*, 2016.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "Overview of supervised learning." In *The elements of statistical learning*, pp. 9–41. Springer, 2009.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In *CVPR*, June 2016.

[IS15] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In *ICML*, pp. 448–456, 2015.

[JNS06] Manfred Jaeger, Jens D Nielsen, and Tomi Silander. "Learning probabilistic decision graphs." *IJAR*, **42**(1):84–100, 2006.

[Jon79] Douglas Samuel Jones. *Elementary information theory*. Clarendon Press, 1979.

[KAD14] Doga Kisa, Guy Van den Broeckand Arthur Choi, and Adnan Darwiche. "Probabilistic sentential decision diagrams." In *KR*, pp. 1–10, 2014.

[Kar13] George Karypis. "METIS A Software Package for Partitioning Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 5.1.0." Technical report, University of Minnesota, 2013.

[KB15] Diederik P Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization." In *ICLR*, 2015.

[KBB12] Angelika Kimmig, Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. "A Short Introduction to Probabilistic Soft Logic." In *NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012.

[KCL19] Pasha Khosravi, YooJung Choi, Yitao Liang, Antonio Vergari, and Guy Van den Broeck. "On Tractable Computation of Expected Predictions." In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[KLC19] Pasha Khosravi, Yitao Liang, YooJung Choi, and Guy Van den Broeck. "What to Expect of Classifiers? Reasoning about Logistic Regression with Missing Features." In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[KMJ14] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. "Semi-supervised Learning with Deep Generative Models." In *NIPS*, 2014.

[Kol09] Vladimir Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm." *Mathematical Programming Computation*, **1**(1):43–67, 2009.

[Kri09] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images." In *University of Toronto Technical Report*, 2009.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classifcation with Deep Convolutional Neural Networks." In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, pp. 1097–1105, 2012.

[KVC14] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. "Probabilistic sentential decision diagrams: Learning with massive logical constraints." In *ICML Workshop on Learning Tractable Probabilistic Models (LTPM)*, 2014.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." *nature*, **521**(7553):436–444, 2015.

[LBV17] Yitao Liang, Jessa Bekker, and Guy Van den Broeck. "Learning the Structure of Probabilistic Sentential Decision Diagrams." In *UAI*, 2017.

[LD08] Daniel Lowd and Pedro Domingos. "Learning Arithmetic Circuits." In *UAI*, 2008.

[LLS07] John Langford, Lihong Li, and Alex Strehl. "Vowpal Wabbit Open Source Project." Technical report, Yahoo!, 2007.

[LMT16] Yitao Liang, Marlos C Machado, Erik Talvitie, and Michael Bowling. "State of the art control of atari games using shallow reinforcement learning." In *AAMAS*, 2016.

[LR13] Daniel Lowd and Amirmohammad Rooshenas. "Learning Markov Networks With Arithmetic Circuits." In *AISTATS*, pp. 406–414, 2013.

[MD08] Robert Mateescu and Rina Dechter. "Mixed deterministic and probabilistic networks." *Annals of mathematics and artificial intelligence*, **54**(1-3):3, 2008.

[MDR17] Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. "Adversarial sets for regularising neural link predictors." In *UAI*, 2017.

[MJ00] Marina Meila and Michael I Jordan. "Learning with mixtures of trees." *JMLR*, **1**:1–48, 2000.

[MKS15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning." *nature*, **518**(7540):529–533, 2015.

[MMK16] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. "Distributional Smoothing with Virtual Adversarial Training." In *ICLR*, 2016.

[MSF17] Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. "Imposing Hard Constraints on Deep Networks: Promises and Limitations." *arXiv preprint arXiv:1706.02025*, 2017.

[MT12] Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 2012.

[MV16] Nicola Di Mauro and Antonio Vergari. "PGM Tutorial on Learning Sum-Product Networks." 2016.

[NB04] Mukund Narasimhan and Jeff A. Bilmes. "PAC-learning bounded tree-width Graphical Models." In *UAI*, 2004.

[NH10] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In *ICML*, pp. 807–814. Omnipress, 2010.

[NJ02] Andrew Y Ng and Michael I Jordan. "On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes." In *NIPS*, pp. 841–848, 2002.

[NR17] Jason Naradowsky and Sebastian Riedel. "Modeling Exclusion with a Differentiable Factor Graph Constraint." In *ICML (Workshop Track)*, 2017.

[PD11] Hoifung Poon and Pedro Domingos. "Sum-product networks: A new deep architecture." In *UAI*, 2011.

[PGD14] Robert Peharz, Robert Gens, and Pedro Domingos. "Learning selective sum-product networks." In *LTPM*, 2014.

[PKD15] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. "Constrained convolutional neural networks for weakly supervised segmentation." In *ICCV*, pp. 1796–1804, 2015.

[PRA14] Nikolaos Pitelis, Chris Russell, and Lourdes Agapito. "Semi-supervised learning using an unsupervised atlas." In *ECML-PKDD*, pp. 565–580. Springer, 2014.

[PVS18] Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. "Probabilistic Deep Learning using Random Sum-Product Networks." *ArXiv*, June 2018.

[RBH15] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. "Semi-supervised learning with ladder networks." In *NIPS*, 2015.

[Ren05] Jason D. M. Rennie. "Regularized Logistic Regression is Strictly Convex." Technical report, MIT, 2005.

[RG16a] Tahrima Rahman and Vibhav Gogate. "Learning Ensembles of Cutset Networks." In *AAAI*, pp. 3301–3307, 2016.

[RG16b] Tahrima Rahman and Vibhav Gogate. "Merging strategies for sum-product networks: From trees to graphs." In *UAI*, 2016.

[RKG14] Tahriman Rahman, Prasanna Kothalkar, and Vibhav. Gogate. "Cutset Networks: A Simple, Tractable, and Scalable Approach for Improving the Accuracy of Chow-Liu Trees." In *ECML PKDD*, pp. 630–645, 2014.

[RL14] Amirmohammad Rooshenas and Daniel Lowd. "Learning Sum-Product Networks with Direct and Indirect Variable Interactions." In *ICML*, pp. 710–718, 2014.

[RL16] Amirmohammad Rooshenas and Daniel Lowd. "Discriminative structure learning of arithmetic circuits." In *AISTATS*, pp. 1506–1514, 2016.

[RSR15] Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. "Injecting Logical Background Knowledge into Embeddings for Relation Extraction." In *HLT-NAACL*, 2015.

[SCD16] Yujia Shen, Arthur Choi, and Adnan Darwiche. "Tractable Operations for Arithmetic Circuits of Probabilistic Models." In *NIPS*, 2016.

[SE17] Russell Stewart and Stefano Ermon. "Label-Free Supervision of Neural Networks with Physics and Domain Knowledge." In *AAAI*, pp. 2576–2582, 2017.

[SG14] David Smith and Vibhav Gogate. "Loopy belief propagation in the presence of determinism." In *AISTATS*, 2014.

[SHM16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature*, **529**(7587):484–489, 2016.

[SMK95] Ashwin Srinivasan, S Muggleton, and RD King. "Comparing the use of background knowledge by inductive logic programming systems." In *ILP*, pp. 199–230, 1995.

[SSS17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge." *Nature*, **550**(7676):354–359, 2017.

[SZ14] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In *ICLR*, 2014.

[VD12] Jan Van Haaren and Jesse Davis. "Markov Network Structure Learning: A Randomized Feature Generation Approach." In *AAAI*, pp. 1148–1154, 2012.

[VDE15] Antonio Vergari, Nicola Di Mauro, and Floriana Esposito. "Simplifying, regularizing and strengthening sum-product network structure learning." In *ECML PKDD*, pp. 343–358, 2015.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms.", 2017.

[XZF18] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. "A Semantic Loss Function for Deep Learning with Symbolic Knowledge." In *ICML*, 2018.