A New Perspective on Learning Context-Specific Independence

Yujia Shen YUJIAS@CS.UCLA.EDU
Arthur Choi AYCHOI@CS.UCLA.EDU
Adnan Darwiche DARWICHE@CS.UCLA.EDU
Computer Science Department, University of California, Los Angeles

Abstract

Local structure such as context-specific independence (CSI) has received much attention in the probabilistic graphical model (PGM) literature, as it facilitates the modeling of large complex systems, as well as for reasoning with them. In this paper, we provide a new perspective on how to learn CSIs from data. We propose to first learn a functional and parameterized representation of a conditional probability distribution (CPD), such as a neural network. Next, we quantize this continuous function, into an arithmetic circuit representation that facilitates efficient inference. In the first step, we can leverage the many powerful tools that have been developed in the machine learning literature. In the second step, we exploit more recently-developed analytic tools from explainable AI, for the purposes of learning CSIs. Finally, we contrast our approach, empirically and conceptually, with more traditional variable-splitting approaches, that search for CSIs more explicitly.

Keywords: context-specific independence, Bayesian networks, arithmetic circuits.

1. Introduction

Context-specific independence (CSI) is a type of local structure that facilitates the modeling of large and complex systems, by allowing one to represent in a succinct way conditional distributions that would otherwise be infeasible to represent (Boutilier et al., 1996). Further, local structure such as CSI can be exploited by modern classes of inference algorithms to perform reasoning in Bayesian networks whose treewidths are too large for more traditional inference algorithms (Darwiche, 2003; Chavira and Darwiche, 2008; Shen et al., 2016; Poole and Zhang, 2003). Traditional representations of CSI use data structures such as decision trees, decision graphs, rules, default tables, etc. (Friedman and Goldszmidt, 1998; Chickering et al., 1997; Larkin and Dechter, 2003; Koller and Friedman, 2009; Pensar et al., 2015). Algorithms for learning these representations are typically search-based, where we iteratively search for variables that split the data into partitions, until the resulting distribution becomes (sufficiently) independent of the remaining variables.[1]

We propose here a new perspective on learning CSIs, resulting in a new context-specific representation for conditional probability distributions (CPDs) that we call Functional Context-Specific CPDs, or just FoCS CPDs. FoCS CPDs generalize rule CPDs, where a context is typically defined as a partial instantiation of the variables. More recently, the Conditional Probabilistic Sentential Decision Diagram (or Conditional PSDD) was proposed, which generalizes term-based rules to arbitrary propositional sentences (Shen et al., 2018). FoCS CPDs generalize this further so that an arbitrary function can be used to define the scope of a context, say one defined by a neural network.

1. Exact algorithms for learning CSI, such as (Koivisto and Sood, 2005; Hyttinen et al., 2018), typically cannot scale to problems where variables may have many parents, as we consider in this paper.

The first significance of this new representation is that it allows us to immediately leverage powerful machine learning systems that have been developed in recent years, for the purposes of learning CSIs. The second significance is that efficient probabilistic reasoning can be enabled, by exploiting recently developed analytic tools from the domain of eXplainable Artificial Intelligence (XAI),[2] which allows us to extract a decision graph representation of a context-specific CPD, but one that facilitates exact inference, i.e., a conditional PSDD (Shen et al., 2018, 2019, 2016).

This paper is organized as follows. In Section 2, we review functional and context-specific representations of CPDs. In Section 3 we propose the FoCS CPD. In Section 4 we propose an algorithm to learn FoCS CPDs from data, and in Section 5 we show how to reason with them. We empirically compare FoCS CPDs with functional and context-specific representations in Section 6, and we provide a case study on "learning to decode" in Section 7. Finally, we conclude in Section 8.

2. Representations of CPDs

A Bayesian network (BN) has two main components: (1) a directed acyclic graph (DAG) and (2) a set of conditional probability tables (CPDs) (Pearl, 1988; Darwiche, 2009; Koller and Friedman, 2009; Murphy, 2012). Typically, CPDs are represented using tabular data structures, although this becomes impractical when a variable has many parents. In this section, we review two alternative representations of interest: functional representations (such as noisy-or models and neural networks) and context-specific representations, such as tree CPDs and rule CPDs.

In what follows, we use upper case letters (X) to denote variables and lower case letters (x) to denote their values. Variable sets are denoted by bold-face upper case letters ($\mathbf{X}$) and their instantiations by bold-face lower case letters ($\mathbf{x}$). Generally, we use X to denote a variable in a Bayesian network and $\mathbf{U}$ to denote its parents. We further refer to $X\mathbf{U}$ as a family. We thus denote a network parameter using the form $\theta_{x|\mathbf{u}}$, which represents the conditional probability $Pr(X=x|\mathbf{U}=\mathbf{u})$.

2.1 Functional Representations

To specify a CPD using a table, one must specify a parameter $\theta_{x|\mathbf{u}}$ for all family instantiations $x\mathbf{u}$, the number of which is exponential in the number of variables in the family $X\mathbf{U}$. In a functional representation of a CPD, one has a parametrized function $f(X\mathbf{U};\theta)$ that *computes* the probability $Pr(x|\mathbf{u})$ from a parameter vector θ that can be much smaller than the size of an explicit table. For example, the well-known noisy-or model implicitly specifies a conditional distribution using a number of parameters that is only linear in the number of parents (Pearl, 1988; Darwiche, 2009).

Other functional representations include logistic functions (Frey, 1998; Vomlel, 2006) as well as neural networks (Bengio and Bengio, 2000; Kingma and Welling, 2014). Consider the following conditional distribution for a variable X with parents U_1U_2, where each variable is binary (0/1): $Pr(x=1 \mid u_1u_2) = \sigma(\theta_0 + \theta_1 u_1 + \theta_2 u_2)$, where $\sigma(a) = [1 + \exp\{-a\}]^{-1}$ is the sigmoid function and where $\theta_0, \theta_1, \theta_2$ are parameters. This functional CPD has the following tabular representation:

u_1, u_2	$1,1$	$1,0$	$0,1$	$0,0$
$Pr(X=1 \mid u_1u_2)$	$\sigma(\theta_0+\theta_1+\theta_2)$	$\sigma(\theta_0+\theta_1)$	$\sigma(\theta_0+\theta_2)$	$\sigma(\theta_0)$
$Pr(X=0 \mid u_1u_2)$	$1-\sigma(\theta_0+\theta_1+\theta_2)$	$1-\sigma(\theta_0+\theta_1)$	$1-\sigma(\theta_0+\theta_2)$	$1-\sigma(\theta_0)$

In general, if we have n binary parents U, then the tabular representation will have 2^n free parameters, whereas the functional logistic representation will have only $n+1$ parameters.

2. https://www.darpa.mil/program/explainable-artificial-intelligence.

While functional representations allow us to compactly specify a CPD, they become unwieldly once we need to perform any reasoning. For example, the following result shows that computing the Most Probable Explanation (MPE) is intractable when using a logistic representation of a CPD, even when the parents are independent. The proof follows by reduction from the knapsack problem.[3]

Theorem 1 *Consider a prior probability $Pr(\mathbf{U})$ over n variables and a conditional probability $Pr(X \mid \mathbf{U})$. If the prior probability is fully factorized and $Pr(X \mid \mathbf{U})$ is represented by a logistic function using $n+1$ parameters, it is NP-complete to compute $\operatorname{argmax}_{\mathbf{u}} Pr(\mathbf{u} \mid x)$.*

2.2 Context-Specific CPDs

A *decision-tree CPD*, or just *tree CPD*, represents a conditional distribution of a variable X given its parents $\mathbf{U}$ in a Bayesian network (Friedman and Goldszmidt, 1998; desJardins et al., 2005). It is composed of a decision tree over variables $\mathbf{U}$, and at each leaf of the decision tree is a CPD column, which we denote by $\Theta_{X|.}$. A *CPD column* $\Theta_{X|.}$ is a distribution over variable X for some given context. A *decision-graph CPD* is a representation like a decision tree, but where equivalent leaves with equivalent CPD columns $\Theta_{X|.}$ are merged together in a single node (Chickering et al., 1997). This decision graph can be further simplified by iteratively merging decision nodes whose children are equivalent. A *rule CPD* is another representation of a conditional distribution that uses rules to define the CPD. A rule is composed of two parts: a context, which is typically a partial instantiation $\mathbf{v}$ of the parent variables $\mathbf{U}$, and a CPD column $\Theta_{X|.}$. A set of rules specify a rule CPD if the contexts $\mathbf{v}$ represent a mutually-exclusive and exhaustive partitioning of the instantiations of $\mathbf{U}$.

A decision-tree CPD specifies a set of rules, where each leaf represents a rule with the same CPD column $\Theta_{X|.}$ assigned to the leaf, where the context $\mathbf{v}$ is found by taking the value of each variable that was branched on, on the path from the root to the leaf. A decision-graph CPD also specifies a set of rules in a similar way, except that we relax the requirement that the context be specified by a partial instantiation, but now as a disjunction of partial instantiations, one for each path that can reach the leaf from the root. The rule CPD can be generalized further by allowing the context to be specified as an arbitrary propositional sentence. Such a rule CPD can be realized using the recently proposed Conditional Probabilistic Sentential Decision Diagram (Conditional PSDD) (Shen et al., 2018).[4] While they enable more succinct representations of conditional distributions, Conditional PSDDs also facilitate the ability to reason with them (Shen et al., 2016, 2019).

3. Functional Context-Specific CPDs

Next, we propose a generalized rule CPD where the scope of a rule is defined, not just by a partial instantiation of its parents, or just by a propositional sentence, but more generally by some function.

Definition 2 *A Functional Context-Specific (FoCS) CPD represents a conditional distribution of a variable X given its parents $\mathbf{U}$, and is defined by a tuple $(f, k, \{I_i\}_{i=1}^k, \{\Theta_{X|I_i}\}_{i=1}^k)$. The function*

3. Say we have a knapsack problem with capacity W and items i, each with weight w_i and value v_i. Each item i has a random binary variable U_i, with a prior $(Pr(X=0), Pr(X=1)) \propto (\exp\{0\}, \exp\{v_i\})$, and with a logistic parameter of weight w_i. The bias parameter is $-W$. By scaling the weights of the logistic function, we approach a step function, and the MPE given evidence $X=0$ obtains a maximum value selection of items within capacity.

4. In a Conditional PSDD, the contexts are represented using a shared SDD (Darwiche, 2011), and the CPD columns are represented using a shared PSDD (Kisa et al., 2014).

f maps each parent configuration to a real number. The k intervals I_i form a mutually-exclusive and exhaustive partition of $\mathbb{R}$. Each interval I_i has a corresponding CPD column $\Theta_{X|I_i}$.

A FoCS CPD, which we denote by $\Phi_{X|\mathbf{U}}$, induces a conditional distribution $Pr(X|\mathbf{U})$ where:

$$Pr(X \mid \mathbf{u}) = \begin{cases} \Theta_{X|I_1} & \text{if } f(\mathbf{u}) \in I_1 \\ \vdots & \vdots \\ \Theta_{X|I_k} & \text{if } f(\mathbf{u}) \in I_k \end{cases}. \tag{1}$$

Each interval I_i defines a context Δ_i. The models of the context are the parent configurations whose function values fall inside I_i. Since the intervals are mutually exclusive and exhaustive, the contexts form a partition of the parent instantiations, hence the conditional distribution is well-defined. Further, the k intervals of a FoCS CPD induces $k-1$ monotonically increasing thresholds, $T_1, \cdots, T_{k-1}$. These thresholds in turn induce intervals $(-\infty, T_1)$, $[T_1, T_2), \cdots, [T_{k-1}, +\infty)$. Hence, we will refer to intervals and thresholds interchangeably.

For example, consider a FoCS CPD with function $\sigma(4{\cdot}U_1U_2{-}2{\cdot}U_1{-}2{\cdot}U_2{+}1)$, and two intervals $I_0 = (-\infty, 0.5)$ and $I_1 = [0.5, \infty)$, with their respective CPT columns $\Theta_{X|I_0} = (0.2, 0.8)$ and $\Theta_{X|I_1} = (0.9, 0.1)$. By enumerating all four instantiations over U_1, U_2, one obtains the following rule CPD that is equivalent to our FoCS CPD:

$\Theta_{X\|\Delta_i}$	Δ_i
$(0.2, 0.8)$	if $U_1 \oplus U_2 \equiv 1$
$(0.9, 0.1)$	if $U_1 \oplus U_2 \equiv 0$

.

4. Learning

In this section, we show how to (1) learn the parameters of a FoCS CPD when the contexts are known, and (2) how to learn the contexts of a FoCS CPD, using parameter learning as a sub-routine.

4.1 Learning the Parameters

Given a dataset $\mathcal{D}$, the log likelihood of a set of Bayesian network parameters Θ is

$$LL(\mathcal{D} \mid \Theta) = \textstyle\sum_{X\mathbf{U}} CLL(\mathcal{D}_{X\mathbf{U}} \mid \Theta_{X|\mathbf{U}})$$

which is the sum of the conditional log likelihoods of the CPDs $\Theta_{X|\mathbf{U}}$ given the datasets $\mathcal{D}_{\mathbf{XU}}$ projected onto the families $X\mathbf{U}$. The local conditional log likelihoods is given by

$$CLL(\mathcal{D}_{X\mathbf{U}} \mid \Theta_{X|\mathbf{U}}) = \textstyle\sum_{i=1}^{N} \log \theta_{x_i|\mathbf{u}_i}$$

which we can optimize independently. Let $\mathcal{D}\#(\mathbf{y})$ denote the number of instances in the dataset $\mathcal{D}$ compatible with the partial instantiation $\mathbf{y}$. The parameters $\theta^\star_{x|\mathbf{u}}$ that optimize the conditional log likelihood is given by $\theta^\star_{x|\mathbf{u}} = \frac{\mathcal{D}\#(x\mathbf{u})}{\mathcal{D}\#(\mathbf{u})}$ which further represent the maximum-likelihood estimates; for more details, see, e.g., (Darwiche, 2009; Koller and Friedman, 2009; Murphy, 2012).

Analagous estimates can be obtained for a Bayesian network with FoCS CPDs $\Phi_{X|\mathbf{U}}$ with contexts Δ_i and the corresponding CPD columns $\Theta_{X|\Delta_i}$. Namely, we have the maximum-likelihood estimates $\theta^\star_{x|\Delta_i} = \frac{\mathcal{D}\#(x,\Delta_i)}{\mathcal{D}\#(\Delta_i)}$. That is, we count the number of instances compatible with both x and the context Δ_i, and then normalize by the total number of instances compatible with the context Δ_i.

	U_1	U_2	X
$\mathbf{d}_1$	0	0	1
$\mathbf{d}_2$	1	0	0
$\mathbf{d}_3$	1	1	0
$\mathbf{d}_4$	0	1	1
$\mathbf{d}_5$	1	1	1

(a)

	U_1	U_2	f
$\mathbf{d}_2$	1	0	$\sigma(-2)$
$\mathbf{d}_4$	0	1	$\sigma(-1)$
$\mathbf{d}_3$	1	1	$\sigma(1)$
$\mathbf{d}_5$	1	1	$\sigma(1)$
$\mathbf{d}_1$	0	0	$\sigma(2)$

(b)

T	$\Delta_{\leq T}$	$\Delta_{>T}$
$-\infty$	$\{\}$	$\{\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_1\}$
$\sigma(-2)$	$\{\mathbf{d}_2\}$	$\{\mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_1\}$
$\sigma(-1)$	$\{\mathbf{d}_2, \mathbf{d}_4\}$	$\{\mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_1\}$
$\sigma(1)$	$\{\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_5\}$	$\{\mathbf{d}_1\}$
$\sigma(2)$	$\{\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_1\}$	$\{\}$

(c)

Figure 1: (a) a dataset, (b) sorted by MLP output, (c) different thresholds and the resulting partitions.

Each maximum likelihood estimates $\theta^{\star}_{x|\Delta_i}$ for a FoCS CPD $\Phi_{X|\mathbf{U}}$ can be computed using a single pass of the dataset $\mathcal{D}$, and also proportional to the time it takes to test whether a given example $\mathbf{d}$ from the dataset is compatible with Δ_i. As contexts of a FoCS CPD are specified by a function and a set of intervals $\{I_i\}_{i=1}^{k}$, it suffices to evaluate the function at the given partial instantiation $\mathbf{u}$ and then test whether the function value lies in the corresponding internal I_i.

4.2 Learning the Contexts

Next, we propose a simple algorithm to learn the contexts of a FoCS CPD from a given dataset $\mathcal{D}$. Based on the definition of a FoCS CPD, its context is defined by a function, that maps parent configurations to a number, and a set of intervals described by thresholds $\{T_i\}_{i=1}^{k-1}$. The context Δ_i consists of parent configurations whose function value fall inside the interval I_i.

Our approach has two steps: (1) we first learn a functional CPD using a multi-layer perceptron (MLP), like the CPDs we discussed in Section 2.1, and then (2) we iteratively learn thresholds on the output of the MLP. From Theorem 1, we know MPE inference using a functional CPD is in general intractable, like the one we shall learn in Step (1). As we shall discuss in Section 5, the FoCS CPDs we obtain from Step (2) shall give us a way to approach this apparent intractability.

First, we learn a functional representation of the conditional distribution $Pr(X \mid \mathbf{U})$. We use an MLP, which we denote by $f_x(\mathbf{u})$, to estimate the conditional probabilities $Pr(x \mid \mathbf{u})$ for some distinguished state x of a variable X; for simplicity, we assume X is binary (0/1). The MLP is trained using feature-label pairs $(\mathbf{u}, x)$ for all family instantiations $x, \mathbf{u}$ that appear in the original dataset $\mathcal{D}$. In our experiments, we used cross entropy as a loss function.

Our next step is to obtain a FoCS CPD $\Phi_{X|\mathbf{U}}$ from the MLP $f_x(\mathbf{U})$ that we have just learned. Our approach is based on learning a threshold on the output of our MLP, which in turn induces a partition of the input space. By iteratively learning additional thresholds, we can further refine our partitioning. Suppose for now that we learn a single threshold T, which yields the following contexts: $\Delta_{\leq T} = \{\mathbf{u} \mid f_x(\mathbf{u}) \leq T\}$ and $\Delta_{>T} = \{\mathbf{u} \mid f_x(\mathbf{u}) > T\}$ where $\Delta_{\leq T} = \neg\Delta_{>T}$ relative to all parent instantiations $\mathbf{u}$, i.e., we have a partitioning.

Consider Figure 1, which highlights a simple example. In Figure 1a, we have a small dataset. Suppose that we learn the following MLP f from this dataset: $f(u_1, u_2) = \sigma(6u_1u_2 - 4u_1 - 3u_2 + 2)$. In Figure 1b, we have sorted this dataset by the value of $f(\mathbf{u})$; remember that the sigmoid function σ is a monotonic non-decreasing function, i.e., $\sigma(x) \leq \sigma(y)$ iff $x \leq y$. Note that for any two consecutive output values f_i and f_j in the sorted list of Figure 1b, any chosen threshold $T \in (f_i, f_j]$ will result in the same partitioning. For example a threshold $T = \sigma(0)$ results in the same partition as threshold $T = \sigma(0.9)$, yielding the sets $\Delta_{\leq T} = \{\mathbf{d}_2, \mathbf{d}_4\}$ and $\Delta_{>T} = \{\mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_1\}$ (note that $\mathbf{d}_3$ and $\mathbf{d}_5$ represent the same parent instantiation $u_1 = 1, u_2 = 1$).

For each threshold T, we can learn the resulting parameters $\theta_{X|\Delta_{\leq T}}$ and $\theta_{X|\Delta_{>T}}$ using a single pass over the dataset, and then compute the resulting conditional log likelihood. If N is the size of the dataset $\mathcal{D}$, then it suffices to check N possible threshold values, plus one additional threshold $T = -\infty$ that ensures that $\Delta_{\leq T}$ is empty, and that $\Delta_{>T}$ contains all of the examples. We then simply pick the single threshold that maximizes the conditional log likelihood. Finally, one can amortize the complexity of computing the conditional log likelihoods for all possible thresholds, hence requiring a single pass over the dataset $\mathcal{D}$ overall.

We can refine the partition further by recursing on each partition, and finding an additional threshold within each partition, using the same algorithm we described above. We can continue to recurse and refine our partition, until validation likelihood falls or does not improve enough.

5. Reasoning

In general, if we use a purely functional representation of a CPD, then inference becomes intractable, as given by Theorem 1. Alternatively, we seek next to obtain tractable FoCS CPDs, first using recently proposed analytic tools from the domain of explainable AI (XAI).

5.1 Marginal Inference via Knowledge Compilation

Recently, in the domain of XAI, Choi et al. (2019) showed how a binary neural network (BNN) can be formally analyzed and verified using symbolic tools from the domain of Knowledge Compilation (Darwiche and Marquis, 2002). A BNN is a neural network with binary inputs and a binary output. Such a neural network represents a Boolean function. Consider for example a linear classifier f: $1.15 \cdot U_1 + 0.95 \cdot U_2 - 1.05 \cdot U_3 \geq 0.52$. Here, U_1, U_2, U_3 are binary (0/1) inputs, and the classifier outputs 1 if this threshold test passes and it outputs 0 otherwise. We can enumerate all possible inputs and record the classifier output $f(u_1, u_2, u_3)$, leading to the following truth table:

U_1	U_2	U_3	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0

U_1	U_2	U_3	f
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

The original numerical classifier is thus equivalent to the Boolean function $[\neg U_3 \wedge (U_1 \vee U_2)] \vee [U_3 \wedge U_1 \wedge U_2]$. Previously, Chan and Darwiche (2003) showed how to extract the Boolean function of a given linear classifier, which includes neurons with step activations as a special case. More recently, Choi et al. (2019) showed that one can compose the Boolean functions of binary neurons and aggregate them to obtain the Boolean function of a binary neural network. By compiling this Boolean function into a tractable logical representation, such as an Ordered Binary Decision Diagram (OBDD), then certain queries and transformations can be performed in time that is polynomial in the size of the resulting circuit (Darwiche and Marquis, 2002; Darwiche, 2011).

Here, we use the algorithm proposed by Choi et al. (2019), to compile a binary neural network into an SDD circuit, in order to compile a FoCS CPD into a Conditional PSDD. First, if we threshold the output of an MLP with step-activations, then it corresponds to a binary neural network. Hence, we can compile each FoCS CPD context Δ_i into an SDD. Second, it is straightforward to compile a

CPD column $\Theta_{X|\cdot}$ into a PSDD (Shen et al., 2018). It is then straightforward to aggregate all of the context SDDs and CPD column PSDDs into a single Conditional PSDD (Shen et al., 2018). If we obtain the CPDs of a Bayesian network as a Conditional PSDD, then we can employ the algorithms in (Shen et al., 2019, 2016), in order to compute marginals in the Bayesian network.[5]

5.2 MPE Inference via Mixed-Integer Linear Programming

Consider the most probable explanation (MPE) query in a Bayesian network: $\operatorname{argmax}_{\mathbf{x}\sim\mathbf{e}} Pr(\mathbf{x})$, where $\mathbf{x}$ is a complete instantiation of the network variables, $\mathbf{e}$ is the observed evidence, and $\sim$ denotes compatiability between $\mathbf{x}$ and $\mathbf{e}$ (they set common variables to the same values). Computing the MPE is an NP-complete problem (Shimony, 1994). Theorem 1 shows that MPE is still NP-complete with independent parents $\mathbf{U}$ and a common observed child X with a functional CPD. However, MPE is still easier than computing marginals, which is a PP-complete problem (Roth, 1996). Hence, compiling to conditional PSDD may be overkill if we only care about MPEs.

Given a FoCS CPD, it suffices to apply a mixed-integer linear programming (MILP) solver to the task of solving an MPE query, i.e., to compute $\operatorname{argmax}_{\mathbf{u}} Pr(\mathbf{u}, x)$ where $Pr(X|\mathbf{U})$ is represented with a FoCS CPD. First, the log of the MPE is a linear function of the log parameters of the network parameters, which we use as the objective of the MILP. Using a FoCS CPD for a binary variable X, an observation x effectively adds another term to the objective function, which depends on the context implied by the input $\mathbf{u}$. This can be incorporated to the MILP after observing that an MLP with step activations can be reduced to an MILP, as in (Narodytska et al., 2018; Griva et al., 2008).

6. Experiments

Here, we empirically evaluate the FoCS CPD and its learning algorithm that we proposed in Sections 3 & 4. In particular, we evaluate our effectiveness at learning conditional distributions, in comparison to other functional and context-specific representations. We shall subsequently evaluate the reasoning algorithms proposed in Section 5, via a case study in Section 7.

We evaluate two sets of benchmarks, one synthetic, and one real-world. We consider two baselines: (1) a functional CPD representation, using a multi-layer perceptron (MLP), and (2) a context-specific CPD representation, namely a tree CPD, which was learned using ID3. We compare each representation based on their (negated) conditional log likelihood (CLL); lower is better. When learning any CPD column, we further use Laplace (add-one) smoothing.

We trained an MLP $f_x(\mathbf{U})$ to predict the value of variable X given an instantiation of the parents $\mathbf{U}$. We used a single hidden layer of 16 neurons with ReLU activations, whose parameters were learned with cross-entropy loss. The MLP was trained using the Adam optimizer in TENSORFLOW. We trained a tree CPD using the ID3 algorithm of the toolkit SCIKIT-LEARN. In our experiments, we learned decision trees of gradually increasing complexity (measured by counting decision tree leaves), by gradually increasing the bound on tree depth, which is a parameter of the ID3 algorithm.

Finally, to obtain a FoCS CPD, we use the learning algorithm described in Section 4, using the MLP that we trained above. In our experiments, we also gradually increased the number of contexts created. With $k-1$ thresholds created, we create k contexts, which we compare with the number of contexts found by the tree CPD (i.e., the number of decision tree leaves).

5. Note that while multiplying two PSDDs is a tractable operation, multiplying n PSDDs may not be.

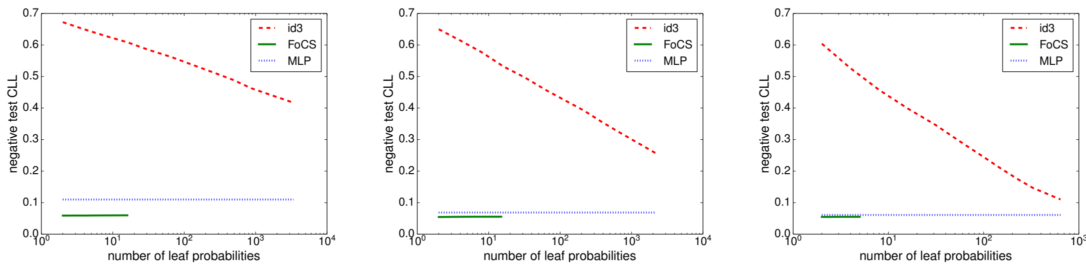

Figure 2: Number of contexts vs. CLL, for $k \in \{2, 4, 8\}$ from left to right (synthetic benchmark).

Synthetic Benchmark. In our first experiment, we simulated training data from a conditional distribution exhibiting many CSIs. That is, we simulated data where the value of variable X depends only on the *cardinality* of its parents $\mathbf{U}$, where X and $\mathbf{U}$ are binary (0/1). If the fraction of parents set to 1 is at most $\frac{1}{s}$, then $Pr(x|\mathbf{u}) = 0.05$; otherwise $Pr(x|\mathbf{u}) = 0.95$. In our experiments, we simulated parent instantations $\mathbf{u}$ such that the two different contexts ($\leq \frac{1}{s}$ and $> \frac{1}{s}$) had the same probability of being generated. We assumed that each FoCS CPD had 16 parent variables, and each was trained using $16,384$ examples, and tested with an additional $16,384$ examples.

In Figure 2, we plot results for $s \in \{2, 4, 8\}$. On the x-axis we increase the number of contexts for the tree CPD (by increasing the bound on depth) and for the FoCS CPD (by adding more thresholds); the MLP is a functional CPD without any explicit CSIs, and hence is a flat line on each plot. We make the following observations in Figure 2: (1) as we increase the number of contexts of the tree CPD (ID3), the better the CLL, (2) the MLP and the FoCS CPD perform similarly, and both obtain better CLLs than the tree CPD, and (3) the FoCS CPD obtains a good CLL using only a small number of contexts, and obtains a better CLL than the MLP that it was created from.

It is well-known that decision trees cannot succinctly represent certain (Boolean) functions. For example, a decision tree must be complete, using 2^n leaves, to represent the parity function over n variables. This is also the case for cardinality constraints over n variables, which have less succinct decision trees for $s = 2$ and succinct decision trees for $s = 1$ or $s = n$. We see this pattern as well in Figure 2, as the performance of ID3 more closely approaches that of MLP and our FoCS CPD.

Compared to the MLP, our FoCS model estimates much fewer parameters—once we are given k contexts Δ_i, then we simply need to estimate the k corresponding CPD columns $\Theta_{X|\Delta_i}$. The fact that our learning appears to converge almost immediately, suggests that our learning algorithm is indeed learning the context-sensitivies inherent in the cardinality constrained data that we simulated. In contrast, the MLP does not search for CSIs. Hence, this explains the ability of our FoCS model to obtain better CLLs than the MLP that our FoCS model was based on.

Real-World Benchmark. Next, we consider a real-world dataset: MNIST digits. This dataset is composed of 28×28 pixel grayscale images, which we binarized to black-and-white. We consider one-vs-all classification, where the parents $\mathbf{U}$ represent the input image, and the child X represents whether the input is of a particular digit d ($X = \mathsf{true}$) or some other digit from 0 to 9 ($X = \mathsf{false}$).

Figure 3 highlights the result. Our FoCS model consistently estimates the conditional distribution more accurately using fewer contexts compared to the tree CPD model. This provides strong evidence that exploiting the structure from a learned functional model is more efficient than searching for the structure of contexts by variable-splitting as done when learning a decision tree. Again, given that our FoCS model appears to converge relatively quickly, this suggests that our learning

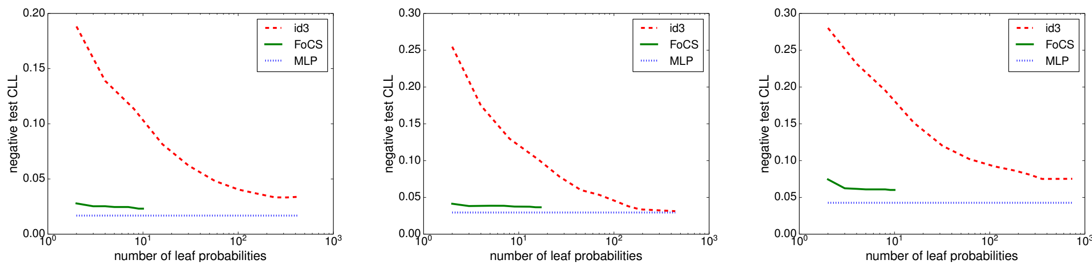

Figure 3: Number of contexts vs. CLL, for digits $d \in \{0, 1, 2\}$ from left-to-right (MNIST).

algorithm is able to learn the CSIs from data (a CSI may represent here a partial instantiation of the input pixels that will almost guarantee the classification of a digit). However, a small number of contexts may not be enough to obtain the performance of an MLP. Note that there may be only a limited number of contexts that our learning algorithm can discover, if the training data is not diverse enough, hence why the curve for our FoCS model stops earlier than that for the tree CPD.[6]

7. Case Study: Learning to Decode

In this section, we show through a simple case study how FoCS CPDs allow us to reason about and learn from complex processes. Our case study is done in the context of channel coding, where our goal is to encode data in a way that allows us to detect and correct for any errors that may occur after transmission through a noisy channel. In subsequent experiments, we show, using our proposed learning algorithm, how one can learn to decode encoded messages, without knowing the original code that was used to encode them!

7.1 Channel Coding: A Brief Introduction

Consider the following problem. Say you have a message represented using n bits $U_0, \ldots, U_{n-1}$ that can be either 0 or 1. We want to transmit this message across a noisy channel, where there is a chance that each bit U_i might be corrupted by noise (say flipped from 0 to 1, or from 1 to 0). To improve the reliability of this process we can send additional bits, say m of them $X_0, \ldots, X_{m-1}$. We refer to the original bits $\mathbf{u}$ as the *message* and the redundant bits $\mathbf{x}$ as the *encoding* (or alternatively the *channel input*). We further refer to the encoding process as the *code*. The *channel output* are the bits $\mathbf{y}$ received from the noisy channel. Finally, there is a *decoding* process that attempts to detect and correct any errors in the channel output.

One simple example of a code, is the repetition code, which sends l additional copies of the message across the noisy channel (say 3 copies total). At the channel output, one detects an error if any of the 3 copies reports a discrepency among the corresponding bits. One can attempt to correct for the error by taking a majority vote. Repetition codes are among the simplest type of error-correcting code. More sophisticated codes include *turbo codes* and *low density parity check codes*, whose decoders were shown to be instances of loopy belief propagation in a Bayesian network;

6. For example, if the MLP is trained to the point where it obtains 100% confidence in most of the training examples (which is not unlikely for datasets such as MNIST), then we would not be able to split any of the resulting contexts).

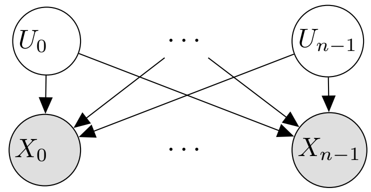

Figure 4: The Bayesian network modeling the encoding process.

see (Frey and MacKay, 1997) for a short perspective. Another common type of code uses parity checks among randomly selected sets of message bits, as redundant bits in the encoding.

7.2 Experiments

We can model a message encoder using a Bayesian network like the one in Figure 4. Root nodes U_i represent the bits (0/1) to be encoded, and the leaf nodes X_i represent the encoded bits (0/1) that are to be transmitted across a noisy channel. For simplicity, we assume that both the message and the encoding are composed of n bits. We assume the message bits U_i are marginally independent. Each of the encoded bits X_i can in general depend on any of the message bits U_i, depending on the particular code being used. At the same time, we can model the noisy channel, which may flip a bit from 0 to 1 or from 1 to 0, with some probability. The traditional reasoning task would be, given a code, i.e., the conditional distributions $Pr(X_i \mid \mathbf{U})$, and a message $\mathbf{x}$ received over the noisy channel, to find the most likely message $\mathbf{u}$ that was originally encoded.

Consider for example a simple code where we record the parity of every pair of adjacent message bits. Such an encoding, can be represented using the following CPD on the left:

$$Pr(X_i{=}1 \mid \mathbf{U}) = \begin{cases} 100\% & \text{if } U_i \oplus U_{i+1 \bmod n} = 1 \\ 0\% & \text{if } U_i \oplus U_{i+1 \bmod n} = 0 \end{cases} \qquad Pr(X_i{=}1 \mid \mathbf{U}) = \begin{cases} 95\% & \text{if } U_i \oplus U_{i+1 \bmod n} = 1 \\ 5\% & \text{if } U_i \oplus U_{i+1 \bmod n} = 0 \end{cases}.$$

We can further model the noise in the channel with the above modified CPD on the right.

Given a set of message/encoding pairs $(\mathbf{u}, \mathbf{x})$ we can try to learn the code used to encode the messages, i.e., learn the conditional distributions $Pr(X_i \mid \mathbf{U})$. We represent each conditional distribution using a FoCS CPD, as a tabular representation would be intractable for this type of problem: the table would have a number of entries that is exponential in n, and we would need at least as much data to learn the parameters.

We learn a FoCS CPD, as in Section 4, starting with an MLP with a single hidden layer containing 8 neurons with sigmoid activations. We convert the sigmoid activations to step activations for the purposes of reducing it to MILP, in order to perform MPE inference as described in Section 5. From the MLP, we obtain a FoCS CPD by learning one threshold, which yields 2 different contexts.

Once we have learned a FoCS model from data, we then try to decode an encoded message. That is, given an encoded message $\mathbf{x}$, we find the most likely original encoding via: $\mathrm{argmax}_{\mathbf{u}}\, Pr(\mathbf{u} \mid \mathbf{x})$, which is an MPE query. We used the MILP solver GUROBI (Gurobi Optimization, 2020) with the CVXPY optimizer to solve these MILP problems, in our experiments, which we present next.

To obtain a training and testing set of messages $\mathbf{u}$ we sampled bits at random with $Pr(u_i) = 0.8$. To obtain a set of encoded messages $\mathbf{x}$, we used a code where each encoded bit took the parity of three consecutive bits (there are n such encoded bits). We assume that the channel has a 5% chance of flipping an transmitted bit. We simulated datasets of size $2^{14} = 16,384$, and performed 5-fold cross validation. The following table summarizes our results.

n	word accuracy	bit accuracy	Hamming error	time (s)
10	0.750 ± 0.003	0.902 ± 0.002	0.974 ± 0.021	0.247 ± 0.001
15	0.651 ± 0.005	0.900 ± 0.002	1.493 ± 0.044	0.469 ± 0.004
20	0.578 ± 0.005	0.905 ± 0.001	1.886 ± 0.037	1.047 ± 0.036
25	0.493 ± 0.007	0.905 ± 0.001	2.371 ± 0.043	11.382 ± 0.549
30	0.414 ± 0.006	0.901 ± 0.003	2.963 ± 0.099	140.190 ± 11.539

From top-to-bottom, each row represents increasing message sizes. We report word accuracy (the percentage of instances where the original message was successfuly decoded from the encoding without error), bit accuracy (the percentage of bits that were decoded without error), Hamming error (the average number of incorrect bits in a decoded message), and time (in seconds).

We make a few observations. Bit accuracy remains consistent around 90%, for all message sizes n. Word accuracy falls, as expected, since it becomes more difficult to decode the entire message without error, the longer the message gets. Note that a 41.4% for $n = 30$ is quite good compared to the expected word accuracy one would have obtained by composing a message estimate from most-likely-bit estimates at 90% accuracy, which would be $0.9^{30} = 4.24\%$. When we consider the hamming error, even if there were an error in the decoding, only a few bits were incorrect on average. Finally, we see that inference time appears to grow exponentially as n grows. This is also expected as decoding is in general an NP-hard problem.

8. Conclusion

We proposed here the FoCS CPD model, for representing CSIs in conditional distributions. We proposed an algorithm for learning the parameters as well as the contexts of FoCS CPDs. We showed how efficient inference can be enabled using FoCS CPDs, by leveraging tools from knowledge compilation and optimization. We highlighted some of the advantages of FoCS CPDs compared to more traditional functional and context-sensitive CPD representations. Finally, we provided a case study showing how FoCS CPDs enable us to "learn how to decode."

Acknowledgments

This work has been partially supported by grants from NSF IIS-1910317, ONR N00014-18-1-2561, and DARPA N66001-17-2-4032.

References

S. Bengio and Y. Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. *Trans. Neur. Netw.*, 11(3):550–557, 2000.

C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *UAI*, pages 115–123, 1996.

H. Chan and A. Darwiche. Reasoning about Bayesian network classifiers. In *UAI*, 2003.

M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *AIJ*, 172(6–7):772–799, 2008.

D. M. Chickering, D. Heckerman, and C. Meek. A bayesian approach to learning bayesian networks with local structure. In *UAI*, pages 80–89, 1997.

A. Choi, W. Shi, A. Shih, and A. Darwiche. Compiling neural networks into tractable Boolean circuits. In *AAAI Spring Symposium on Verification of Neural Networks (VNN)*, 2019.

A. Darwiche. A differential approach to inference in Bayesian networks. *JACM*, 2003.

A. Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, pages 819–826, 2011.

A. Darwiche and P. Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

M. desJardins, P. Rathod, and L. Getoor. Bayesian network learning with abstraction hierarchies and context-specific independence. In *ECML*, pages 485–496, 2005.

B. J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

B. J. Frey and D. J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In *NIPS*, pages 479–485, 1997.

N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Learning in graphical models*, pages 421–459. Springer, 1998.

I. Griva, S. G. Nash, and A. Sofer. *Linear and Nonlinear Optimization*. SIAM, 2008.

L. Gurobi Optimization. Gurobi optimizer reference manual, 2020.

A. Hyttinen, J. Pensar, J. Kontinen, and J. Corander. Structure learning for Bayesian networks over labeled DAGs. In *PGM*, pages 133–144, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *KR*, 2014.

M. Koivisto and K. Sood. Computational aspects of Bayesian partition models. In *ICML*, pages 433–440, 2005.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

D. Larkin and R. Dechter. Bayesian inference in the presence of determinism. In *AISTATS*, 2003.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

N. Narodytska, S. P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh. Verifying properties of binarized deep neural networks. In *AAAI*, 2018.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. MK, 1988.

J. Pensar, H. Nyman, T. Koski, and J. Corander. Labeled directed acyclic graphs: a generalization of context-specific independence in directed graphical models. *DMKD*, 29(2):503–533, 2015.

D. Poole and N. L. Zhang. Exploiting contextual independence in probabilistic inference. *JAIR*, 18:263–313, 2003.

D. Roth. On the hardness of approximate reasoning. *AIJ*, 82(1-2):273–302, 1996.

Y. Shen, A. Choi, and A. Darwiche. Tractable operations for arithmetic circuits of probabilistic models. In *NIPS*, 2016.

Y. Shen, A. Choi, and A. Darwiche. Conditional PSDDs: Modeling and learning with modular knowledge. In *AAAI*, 2018.

Y. Shen, A. Goyanka, A. Darwiche, and A. Choi. Structured bayesian networks: From inference to learning with routes. In *AAAI*, 2019.

S. E. Shimony. Finding MAPs for belief networks is NP-hard. *Artif. Intell.*, 68(2):399–410, 1994.

J. Vomlel. Noisy-or classifier. *Int. J. Intell. Syst.*, 21(3):381–398, 2006.