IFT 6269 Final Report: Probabilistic Reasoning, from Graphs to Circuits

Breandan Considine [1 2 3]

Abstract

Graphical models (PGMs) are very expressive, but even approximate inference on belief networks (BNs) is NP-hard (Dagum & Luby, 1993). We can faithfully represent a large class of PGMs and their corresponding distributions as probabilistic circuits (PCs) (Choi et al., 2020), which are capable of exact inference in polynomial time and tractable to calibrate using SGD or EM. PCs share many algebraic properties with PGMs and can propagate statistical estimators using simple algebraic rules. In this work, we will see how to compile BNs to PCs using the approach developed by Zhao et al. (2015), and demonstrate their equivalence on a few toy inference problems.

1. Introduction

Probabilistic inference is logical framework for reasoning about uncertainty. In this work, we define a denotational and operational semantics for probabilisic inference based on constructive logic, then build a concrete implementation by translating those inference rules to the Kotlin language.

Let E be a set of events and S be a set of subsets of E. A *probability distribution* is a function $P : S \times \Sigma \to \mathbb{R}^+$ satisfying the Kolmogorov (1933) axioms, in particular:

(3) $P(S = s) \in \mathbb{R}^+, \forall s \in S.$

(4) $P(E) = 1$. ($S =$ may be omitted for brevity.)

(5) $P(X \cup Y) = P(X) + P(Y), \forall X \cap Y = \varnothing.$

Suppose we have the following probabilistic language:

$\mathcal{D} \to$ Gaussian	$\mathcal{D} \to$ Bernoulli	$\mathcal{D} \to$ Dirichlet ...
$\mathcal{V} \to A \mid \ldots \mid Z$	$\mathcal{V} \to \mathcal{V}, \mathcal{V}$	$\mathcal{P} \to \mathcal{P}(\mathcal{V}) \mid \mathcal{D}$
$\mathcal{P} \to \mathcal{P}(\mathcal{V} \mid \mathcal{V})$	$\mathcal{P} \to \mathcal{P}(\mathcal{E})$	$\mathcal{S} \to \mathcal{V} \sim \mathcal{P}$
$\mathcal{E} \to \mathcal{V} \pm \mathcal{V}$	$\mathcal{E} \to \mathcal{V}\mathcal{V}$	$\mathcal{E} \to \mathcal{V} \div \mathcal{V}$
$\mathcal{P} \to \prod_{i=1}^{n} \mathcal{P}$	$\mathcal{P} \to \sum_{i=1}^{n} \mathcal{P}$	$\mathcal{P} \to \mathcal{P} * \mathcal{P}$

[1]McGill University, School of Computer Science [2]Knowledge and Software Technology Lab [3]Mila Quebéc. Correspondence to: Breandan Considine <breandan.considine@mail.mcgill.ca>.

IFT 6269 project report - Université de Montréal

Given a distribution over a set X, we can *sample* from it to produce a single element from that set, a *random variable*:

$$\frac{\Gamma \vdash P(X) : X \times \Sigma \to \mathbb{R}^+ \qquad x \sim P(X)}{\Gamma \vdash x : (X \times \Sigma \to \mathbb{R}^+) \rightsquigarrow X} \text{ Sample}$$

The joint distribution $P(X, Y)$ is a distribution over the Cartesian product of the two sets X and Y, denoted $X \times Y$:

$$\frac{\Gamma \vdash P(X) : X \times \Sigma \to \mathbb{R}^+, P(Y) : Y \times \Sigma \to \mathbb{R}^+}{\Gamma \vdash P(X, Y) : X \times Y \times \Sigma \to \mathbb{R}^+} \text{ Joint}$$

Given a joint distribution $P(X, Y)$, if we see an event $y : Y$, this observation is called *conditioning* and the resulting distribution over X is called a *conditional distribution*:

$$\frac{\Gamma \vdash P(X, Y) : X \times Y \times \Sigma \to \mathbb{R}^+ \qquad \Gamma \vdash y : Y}{\Gamma \vdash P(X \mid Y = y) : X \times \Sigma \to \mathbb{R}^+} \text{ Cond}$$

Given a conditional distribution and its prior, to exchange the order of conditioning, we can use Bayes' (1763) rule:

$$\frac{\overbrace{P(X \mid Y)}^{\text{Likelihood}} \quad \overbrace{P(Y)}^{\text{Prior}}}{\underbrace{P(Y \mid X)}_{\text{Normalize}} \propto \underbrace{P(X \mid Y)}_{\text{Observe}} \underbrace{P(Y)}_{\text{Sample}}} \text{ Bayes}$$

When a conditional distribution $P(X \mid Y)$ does not depend on its prior, the variables are said to be *independent*.

$$\frac{P(X \mid Y) = P(X)}{X \perp Y} \text{ Indep}$$

Equivalently, when two distributions $P(X)$ and $P(Y)$ are multiplied to form a joint distribution $P(X, Y)$, we may conclude that X and Y are independent random variables:

$$\frac{P(X, Y) = P(X)P(Y)}{X \perp Y} \text{ Fact}$$

When two conditionals $P(X \mid Z), P(Y \mid Z)$ are multiplied to form a joint distribution $P(X, Y \mid Z)$, X and Y are said to be *conditionally independent given* Z:

$$\frac{P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z)}{X \perp Y \mid Z} \text{ CondIndep}$$

2. Operational Semantics

So far, we have defined a denotational semantics describing a simple type system for probability functions. But how exactly do we obtain concrete samples and estimates?

In order to sample from a univariate distribution, we can feed the output from a uniform PRNG into a specific function called a *quantile*. Unfortunately, most common distributions are defined in terms of their probability or cumulative density functions and do not have a closed form quantile, so we must approximate it by inverting the CDF:

$$\frac{\texttt{x} \sim P(X) \qquad \texttt{CDF} : x \mapsto \int P(X = x)dx}{\dfrac{\texttt{PRNG() = CDF(x)}}{\texttt{x = INVCDF(PRNG())}}\ \text{Invert}}\ \text{Draw}$$

If we have two random variables sampled from known distributions and wanted to combine them, what would be the resulting distribution? For independent RVs, we know the joint distribution is the product of their distributions:

$$\frac{P(X) \qquad P(Y)}{P(X,Y) = P(X)P(Y)}\ \text{Join}$$

But suppose we knew a formula describing how those RVs were related. All mathematical formulae are composed of algebraic operators, but the algebra of RVs is nontrivial: the distribution formed by combining two RVs, in general, cannot be obtained by the same formula on their distributions.

$$P\big(f(X,Y)\big) \neq f\big(P(X), P(Y)\big)$$

To obtain a formula for the combined distribution, we must ensure the combination of two probability distributions is also a probability distribution. Generally speaking, to combine two arbitrary RVs, we must integrate. For a dyadic function, we can take the double integral, which is known to be exchangeable under certain conditions (Fubini, 1907):

$$\frac{x \sim P(X) \qquad y \sim P(Y) \qquad z : X \times Y \to Z}{\dfrac{P(Z = z(x,y) \mid X = x, Y = y) = \int\int z(x,y)dxdy}{\int_Y \int_X f(x,y)dxdy = \int_X \int_Y f(x,y)dydx}}$$

These integrals can often be simplified by considering the specific dependence relation. For instance, the sum distribution for independent RVs is obtained by convolving (see § 7.1 for further details) their respective density functions:

$$\frac{P(X) \qquad P(Y)}{P(X+Y) = P(X) * P(Y)}\ \text{Conv}$$

More specifically, if we have a so-called *stable distribution* (Lévy, 1925), the solution is even more simple. For example, if X and Y are both known to be Gaussian RVs, we can combine their parameters μ and σ directly:

$$\frac{x \sim \mathcal{N}(\mu_x, \sigma_x^2) \qquad y \sim \mathcal{N}(\mu_y, \sigma_y^2) \qquad z = x + y}{z \sim \mathcal{N}(\mu_x + \mu_y, \sigma_x^2 + \sigma_x^2)}$$

As described in Willard (2020), there exist many simplified relations of the above variety, which can be derived through analytic integration of the corresponding density functions.

Given a joint distribution $P(X, Y)$, in order to remove a variable, we must sum or integrate over its conditional distribution. This process is known as *marginalization*, and the resulting distribution is called a *marginal distribution*:

$$\frac{\Gamma \vdash P(X,Y) : X \times Y \times \Sigma \to \mathbb{R}^+}{\Gamma \vdash P(X) : X \times \Sigma \to \mathbb{R}^+ \propto \int P(X \mid Y = y)dy}\ \text{Mar}$$

Given the joint distribution, we can obtain a conditional distribution by dividing the joint by the marginal distribution:

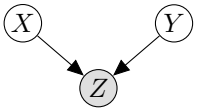
$$\frac{\Gamma \vdash P(X,Y) : X \times Y \times \Sigma \to \mathbb{R}^+}{\Gamma \vdash P(X \mid Y) : X \times \Sigma \to \mathbb{R}^+ = P(X,Y) \div P(Y)}\ \text{Con}$$

3. Probabilistic Graphical Models

Probabilistic graphical models (Jordan, 2003; Koller & Friedman, 2009) (PGMs) are a framework for probabilistic inference on distributions whose conditional dependence structure may be expressed as a graph. The graph's edges capture conditional independence relations between RVs. One type of PGM is a directed graphical model (DGM).

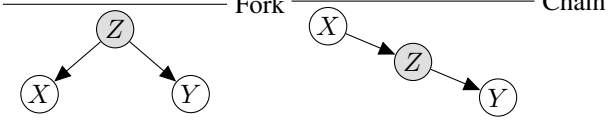
If we observe a variable with two latent variables which are not conditionally independent, we call this a *collider*.

$$\frac{X \not\perp Y \mid Z}{\text{Collide}}$$



If we observe a variable with conditionally independent latent variables, we call this a *fork* or equivalently, a *chain*.

$$\frac{X \perp Y \mid Z}{\text{Fork}} \qquad \frac{X \perp Y \mid Z}{\text{Chain}}$$



In general, we can describe a belief network (BN) is an acyclic DGM of the form:

$$P(x_1, \ldots, x_D) = \prod_{i=1}^{D} P(x_i \mid \texttt{parents}(x_i))$$

4. Probabilistic Circuits

By defining some elementary distributions and composing them by means of simple operators, we obtain a rich framework for probabilistic modeling. Recent work (Choi et al., 2020) has explored tractable models for probabilistic reasoning based on semiring algebras. Semirings are known to have many useful applications in graph theory (Dolan, 2013) and formal languages (Bernardy & Claessen, 2013).

A semiring algebra has two operators, $\oplus$ and $\otimes$, with the usual properties. In particular, distributivity holds:

$$\frac{X \otimes (Y \oplus Z)}{(X \otimes Y) \oplus (X \otimes Z)} \quad \frac{(Y \oplus Z) \otimes X}{(X \otimes Y) \oplus (X \otimes Z)} \text{ Distrib}$$

The sum product network (SPN) is a commutative semiring over univariate distributions (Friesen & Domingos, 2016):

$$PC \to v \sim \mathcal{D} \quad PC \to PC \oplus PC \quad PC \to PC \otimes PC$$

Given a BN, we can compile it to a SPN using the following procedure described by Butz et al. (2019):

---

procedure TRANSLATE(b: BN): SPN
  c ←VARIABLEELIMINATE(b) ▷ Reverse topsort.
  s ←REDISTRIBUTEPARAMS(c) ▷ AC → SPN.
  s ←COMPILEMARGINALIZED(s)
  return CANONICALIZE(s)
end procedure
procedure CANONICALIZE(s_0: SPN): SPN
  $s_1 \leftarrow$ ADDTERMINALS(s_0)
  $s_1 \leftarrow$ MERGEPRODUCTS(s_1)
  if $s_0 = s_1$ **then return** s_1 ▷ Recurse until we
  else return CANONICALIZE(s_1) ▷ reach a fixpoint.
end procedure

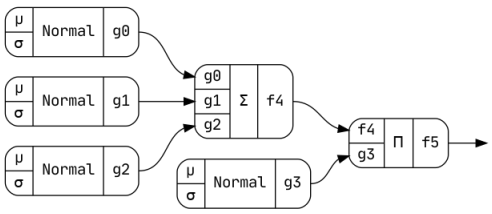
For directed models, such as GMMs and DGMs, most of the work is computing the topological ordering and canonicalization: compilation is effectively linearithmic with respect to the number of nodes in the network.

5. DSL

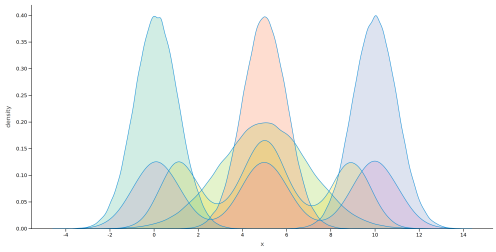
Our DSL implements these rules for Gaussian SPNs.

```
val g0 = Gaussian(mu = 0.1,  sig = 1.0)
val g1 = Gaussian(mu = 5.0,  sig = 1.0)
val g2 = Gaussian(mu = 10.0, sig = 1.0)
val g3 = Gaussian(mu = 5.0,  sig = 2.0)
val f4 = g0 + g1 + g2
val f5 = g3 * f4
compare(g0, g1, g2, g3 g4, g5).show()
```

This program forms a computation graph. The topology of this data structure is invariant to instruction reordering.



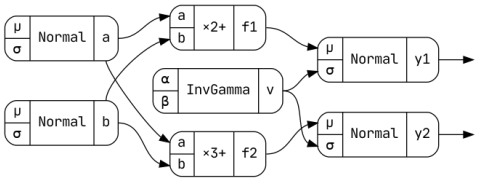
Composition allows us to build complex mixture densities:



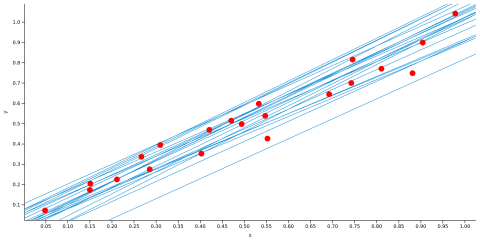
Similarly, we can compose higher-order distributions:

```
val a = Gaussian(mu = 0, sigma = 9.0)
val b = Gaussian(mu = 0, sigma = 9.0)
val v = InvGamma(alpha = 0.5, beta = 0.5)
val f1 = a * 2 + b
val f2 = a * 3 + b
val y1 = Gaussian(mu = f1, sigma = v)
val y2 = Gaussian(mu = f2, sigma = v)
compare(f1, f2, y3, y4).show()
```

The parameters of `y1`, `y2` are themselves distributions:



We can visualize this distribution in function space:



Inspired by Ellis et al. (2016), this DSL allows us to construct a distribution over simple programs, where the language itself is described by the CFG shown earlier. Not only can it describe simple distributions on continuous and categorical RVs, but distributions over simple programs, whose inputs are parameterized by recursively simpler distributions. Program synthesis then, consists of sampling from the space of probabilistic programs in our DSL.

6. Conclusion

Prior work (Considine et al., 2019; Considine, 2019) explored differentiable programming. Differentiability plays a key role in learning, but does not provide the necessary vocabulary to describe human knowledge. In order to capture human knowledge and begin to reason as humans do, programs must be able to express *uncertainty*.

What is the probability of observing local variations? How are those observations related? And how do local variations affect global behavior? In order to correctly pose these questions and begin to answer them, we must be able to reason about uncertainty (Pearl, 2014). In order to do so, we first defined some relations between probability distributions. We showed how to rewrite and factorize them in various contexts, and sample from them programmatically.

Graphs are a natural representation for both programs (Allamanis et al., 2017) and probabilistic models (Pearl, 2014). The language of linear algebra provides a unifying framework for many graph algorithms and computational tasks (Kepner & Gilbert, 2011). Recent evidence suggests probabilisitic inference is tractable for a large class of graphical models (Choi et al., 2020). And sparse matrix representations enable efficient execution of large graphs on modern graphics processors (Kepner et al., 2016).

In this work, we show how to learn a BN from data, and compare its performance with BNs using a toy dataset in which inference is empirically tractable. To do so, we lift numerical computation graph into the domain of probability kernels, and implement general-purpose combinators and domain-specific estimators for simple distributions in the algebra of random variables. By computing lazily and only lowered onto numerical values only when necessary to perform generation or inference, allows us to perform flexible algebraic rewriting to optimize for e.g. latency or numerical stability.

While currently limited to simple distributions and CPU-based sampling, in future work, we hope to accelerate our DSL and extend these ideas to the domain of program synthesis. To learn more about our work, please visit: `https://github.com/breandan/markovian`.

References

Allamanis, M., Brockschmidt, M., and Khademi, M. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017.

Bayes', T. An essay toward solving a problem in the doctrine of chances. *Philosophical Transactions*, pp. 1683–1775, 1763.

Bernardy, J.-P. and Claessen, K. Efficient divide-and-conquer parsing of practical context-free languages. In *Proceedings of the 18th ACM SIGPLAN International Conference on Functional Programming*, ICFP '13, pp. 111122, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450323260. doi: 10.1145/2500365.2500576. URL `https://doi.org/10.1145/2500365.2500576`.

Butz, C. J., Oliveira, J. S., and Peharz, R. Sum-product network decompilation. *arXiv preprint arXiv:1912.10092*, 2019.

Choi, Y., Vergari, A., and Van den Broeck, G. Probabilistic circuits: A unifying framework for tractable probabilistic models. 2020. URL `http://starai.cs.ucla.edu/papers/ProbCirc20.pdf`.

Considine, B. Programming Tools for Intelligent Systems. Master's thesis, Université de Montréal, 2019. URL `https://ndan.co/public/masters_thesis.pdf`.

Considine, B., Famelis, M., and Paull, L. Kotlin∇: A Shape-Safe eDSL for Differentiable Programming. `https://github.com/breandan/kotlingrad`, 2019.

Dagum, P. and Luby, M. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial intelligence*, 60(1):141–153, 1993. URL `https://doi.org/10.1016/0004-3702(93)90036-B`.

Dolan, S. Fun with semirings: a functional pearl on the abuse of linear algebra. In *Proceedings of the 18th ACM SIGPLAN international conference on Functional programming*, pp. 101–110, 2013.

Ellis, K., Solar-Lezama, A., and Tenenbaum, J. Sampling for bayesian program learning. 2016.

Friesen, A. and Domingos, P. The sum-product theorem: A foundation for learning tractable models. In *International Conference on Machine Learning*, pp. 1909–1918, 2016.

Fubini, G. Sugli integrali multipli. *Rend. Acc. Naz. Lincei*, 16:608–614, 1907.

Jordan, M. I. An introduction to probabilistic graphical models, 2003.

Kepner, J. and Gilbert, J. *Graph algorithms in the language of linear algebra*. SIAM, 2011.

Kepner, J., Aaltonen, P., Bader, D., Buluç, A., Franchetti, F., Gilbert, J., Hutchison, D., Kumar, M., Lumsdaine, A., Meyerhenke, H., et al. Mathematical foundations of the graphblas. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–9. IEEE, 2016.

Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Kolmogorov, A. N. Grundbegriffe der Wahrscheinlichkeitsrechnung, 1933.

Lévy, P. Calcul des probabilités. 1925.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

Pearl, J. and Paz, A. *Graphoids: A graph-based logic for reasoning about relevance relations*. University of California (Los Angeles). Computer Science Department, 1985. URL http://ftp.cs.ucla.edu/pub/stat_ser/r53-L.pdf#page=7.

Willard, B. T. miniKanren as a tool for symbolic computation in python. *arXiv preprint arXiv:2005.11644*, 2020. URL https://arxiv.org/pdf/2005.11644.pdf.

Zhao, H., Melibari, M., and Poupart, P. On the relationship between sum-product networks and Bayesian networks. In *International Conference on Machine Learning*, pp. 116–124, 2015.

7. Appendix

7.1. Combinators

The *convolution* operator $*$ is defined as the sum of a product at each point:

$$\frac{\Gamma \vdash g : S^d \to \mathbb{R}, h : S^d \to \mathbb{R}}{(g * h)(x) = \int_{S^d} f(y)g(x-y)dy} \text{ ContConv}$$

$$\frac{\Gamma \vdash g : S^d \to \mathbb{Z}, h : S^d \to \mathbb{Z}}{(g * h)(x) = \sum_{S^d} f(y)g(x-y)} \text{ DiscConv}$$

7.2. Formal Language

We can derive closed form estimators for SPNs, including expectation and variance, using the following formulae:

$$\frac{\Gamma \vdash A : \mathcal{D}, B : \mathcal{D} \qquad C = A + B}{\mathrm{Var}[C]^2 = \sqrt{\mathrm{Var}[A]^2 + \mathrm{Var}[B]^2 + 2 \cdot \mathrm{Cov}[A, B]}} \text{ VSum}$$

$$\frac{\Gamma \vdash A : \mathcal{D}, B : \mathcal{D} \qquad C = AB}{\mathrm{Var}[C]^2 = C\sqrt{\frac{\mathrm{Var}[A]^2}{A^2} + \frac{\mathrm{Var}[B]^2}{B^2} + 2\frac{\mathrm{Cov}[A,B]}{AB}}} \text{ VProd}$$

$$\frac{\Gamma \vdash A : \mathcal{D}, B : \mathcal{D} \qquad C = A + B}{\mathbb{E}[C] = \mathbb{E}[A] + \mathbb{E}[B]} \text{ ESum}$$

$$\frac{\Gamma \vdash A : \mathcal{D}, B : \mathcal{D} \qquad C = AB}{\mathbb{E}[C] = \mathbb{E}[A]\mathbb{E}[B]} \text{ EProd (IID)}$$

7.3. Probabilistic Graphical Models

Following Pearl & Paz (1985), the grammar of conditional independence statements has some equivalence relations:

$$\frac{X \perp Y \mid Z}{Y \perp X \mid Z} \text{ Sym} \qquad \frac{X \perp Y, W \mid Z}{X \perp Y \mid Z} \text{ Decomp}$$

$$\frac{X \perp Y \mid Z \qquad X \perp Z \mid Y}{X \perp Y, W \mid Z} \text{ Union}$$

$$\frac{X \perp W \mid Y, Z \qquad X \perp Z \mid Y}{X \perp W \mid Y} \text{ Contract}$$

A path between two vertices $\textcircled{A} \ldots \textcircled{B}$ is blocked if:

1. $\textcircled{A} \ldots \to \textcircled{V} \to \ldots \textcircled{B}$ or $\textcircled{A} \ldots \leftarrow \textcircled{V} \to \ldots \textcircled{B}$.
2. $\textcircled{A} \ldots \to \textcircled{V} \leftarrow \ldots \textcircled{B}$ and $\textcircled{?} \notin \texttt{desc}(\textcircled{V})$.

$\textcircled{A}$ and $\textcircled{B}$ are d-separated if all paths are blocked.