AUXILIARY VARIABLES IN PROBABILISTIC PROGRAMS

EKANSH SHARMA AND DANIEL M. ROY

Abstract. We extend the first-order meta language of Staton et al. (2016) with a new language construct, `slice-let`. This new construct allows a user to define a random variable x, with a potentially intractable distribution, by introducing an auxiliary random variable, u, and specifying only the conditional distributions of x given u and of u given x. In effect, the distribution of x need only be defined up to some level of approximation, determined by the value of u. We outline the denotational semantics for `slice-let` and give some example programs. Finally, we briefly discuss an approximate representation of the program that can be used by an inference algorithm.

1. Introduction

Most probabilistic programming languages work by allowing users to define prior distributions by specifying exact samplers (see, e.g., (Goodman et al., 2008; Staton et al., 2016)). This tact is limiting; for example, it is known that certain conditional independences cannot be represented by exact samplers (Ackerman et al., 2017). In this paper, we consider the problem of representing distributions approximately. The setup is as follows: We want to introduce a random variable, x, whose distribution is potentially intractable. Normally, one would have to specify an exact sampler to use this random variable within a PPL. Instead, the programmer introduces an auxiliary variable, u, usually by specifying a simple conditional distribution for u given x. Instead of the user specifying the distribution of x, the user specifies the conditional distribution of x given u. The semantics of our construct assign the expression the distribution of x. Implementations, however, can use the two conditional distributions to generate approximate or exact–approximate inference algorithms. These types of algorithms are known as slice samplers. In effect, the distribution of x given u treats u as a level of approximation and one need only sample x up to that level of approximation. Potential applications include exact-approximate representation of Dirichlet Processes (Papaspiliopoulos and Roberts, 2008), Indian Buffet Processes (Teh, Görür, and Ghahramani, 2007).

2. Syntax and Semantics

We extend the first-order meta-language described in Staton et al. (2016), the syntax of which is shown in Fig. 1. This language is an idealized PPL with infinitary type and term constructor, and a constant term for all measurable functions between measurable spaces. It also contains `sample` and `observe` primitives that facilitate construction of Bayesian probabilistic models. We augment this language by adding $\texttt{slice-let}\ u = \texttt{sample}(\mu_0)\ \texttt{with}\ x = P_1\ \texttt{update}\ u = P_2$. The typing rule for `slice-let` is

$$\frac{\Gamma \vdash_d \mu_0 : \mathrm{P}(\mathbb{R}_+) \qquad \Gamma, u : \mathbb{R}_+ \vdash_p P_1 : \mathbb{A} \qquad \Gamma, x : \mathbb{A} \vdash_p P_2 : \mathbb{R}_+}{\Gamma \vdash_p \texttt{slice-let}\ u = \texttt{sample}(\mu_0)\ \texttt{with}\ x = P_1\ \texttt{update}\ u = P_2 : \mathbb{A} \times \mathbb{R}_+}$$

The construct `slice-let` generalizes the probabilistic term `let`: instead of providing sequential dependence between u and x, user provides an initial (potentially incorrect) distribution for u, $[\![\mu_0]\!](\gamma) : \Sigma_{[\![\mathbb{R}_+]\!]} \to [0,1]$ and two Markov transition kernels $[\![P_1]\!](\gamma) : [\![\mathbb{R}_+]\!] \times \Sigma_{[\![\mathbb{A}]\!]} \to [0,1]$ and $[\![P_2]\!](\gamma) : [\![\mathbb{A}]\!] \times \Sigma_{[\![\mathbb{R}_+]\!]} \to [0,1]$. Informally, the semantics of `slice-let` construct is the stationary distribution on $[\![\mathbb{A}]\!] \times [\![\mathbb{R}_+]\!]$ of the Markov chain corresponding to the kernel $P' = P'_1 P'_2$, where $P'_i = [\![P_i]\!](\gamma)$. More formally, $[\![\texttt{slice-let}\ u = \texttt{sample}(\mu_0)\ \texttt{with}\ x = P_1\ \texttt{update}\ u = P_2]\!](\gamma)$ is the unique measure $\mu : \Sigma_{[\![\mathbb{A}]\!]} \times \Sigma_{[\![\mathbb{R}_+]\!]} \to [0,1]$ such that, for all bounded measurable functions f, $\int f(x',u)\mu(\mathrm{d}x, \mathrm{d}u)P'_1(u, \mathrm{d}x') = \int f(x,u')\mu(\mathrm{d}x, \mathrm{d}u)P'_2(x, \mathrm{d}u') = \int f(x,u)\mu(\mathrm{d}x, \mathrm{d}u)$.

Types:

$$\mathbb{A}, \mathbb{B} ::= \mathbb{R} \mid 1 \mid \mathrm{P}(\mathbb{A}) \mid \mathbb{A} \times \mathbb{B} \mid \mathbb{A} + \mathbb{B}$$

Expressions:

$$\begin{aligned} t, u, v ::= {} & x \mid * \mid (t, u) \mid (i, t) \mid \pi_j(t) \mid \texttt{case}\ t\ \texttt{of}\ \{(i, x) \Rightarrow u_i\}_{i \in I} \\ & \mid f(t) \mid \texttt{sample}(t) \mid \texttt{observe}(t, u) \\ & \mid \texttt{slice-let}\ u = \texttt{sample}(\mu_0)\ \texttt{with}\ x = P_1\ \texttt{update}\ u = P_2 \\ & \mid \texttt{let}\ x = t\ \texttt{in}\ u \mid \texttt{return}(t) \end{aligned}$$

FIGURE 1. Syntax for Language

$$\begin{aligned} \texttt{slice-let}\ u = \texttt{sample}(\delta_{u_0})\ \texttt{with}\ x = \texttt{sample}(\text{Uniform}(\texttt{normal-inv}(u, \mu, \sigma))) \\ \texttt{update}\ u = \texttt{sample}(\text{Uniform}[0, \texttt{density-normal}(x, \mu, \sigma)]) \end{aligned}$$

FIGURE 2. Slice-sampling a normal distribution

$$\begin{aligned} \texttt{let truncated-sample} = \ \texttt{slice-let}\ u = \texttt{sample}(\delta_{u_0})\ \texttt{with}\ (x, n) = \texttt{truncated-sample} \\ \texttt{update}\ u = \texttt{sample}(\text{Uniform}[0, 2^{-n}]) \end{aligned}$$

FIGURE 3. Bernoulli Process w/ decaying weights

3. EXAMPLES

Fig. 2 gives a program for slice-sampling a normal distribution (Neal, 2003). Let $\mathcal{P}$ represent the program in Fig. 2. In this toy example, define $\texttt{density-normal}(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp(\frac{(x-\mu)^2}{2\sigma^2})$, define $\texttt{normal-inv}(u, \mu, \sigma) = \{x \mid \texttt{density-normal}(x, \mu, \sigma) > u\}$ and let $u_0 \in [0, \texttt{normal-density}(\mu, \mu, \sigma)]$. Denotationally, the following equation holds:

$$[\![\mathcal{P}]\!](\gamma) = [\![\texttt{sample}(\mathcal{N}(\mu, \sigma))]\!](\gamma).$$

A similar approach can be used to sample from any univariate absolutely continuous distribution (Neal, 2003).

In the second example, let $Z = (z_1, z_2, ...)$, be an infinite dimensional vector such that $z_i \overset{ind}{\sim} \text{Bernoulli}(2^{-i})$. An exact representation for such infinite dimensional object requires an infinite loop. Now, we write a function $\texttt{truncated-sample} : (0, 1) \to \{0, 1\}^\infty \times \mathbb{N}$ in the language. Define the terms $\Gamma \vdash_d t_n : \{0, 1\}^n$ as follows:

$$\begin{aligned} t_{n+1} &:= \texttt{let}\ x = \texttt{flip}(2^{-(n+1)})\ \texttt{in}\ \texttt{return}(t_n, x) \\ t_1 &:= \texttt{flip}(2^{-1}) \end{aligned}$$

Also, define the terms $\Gamma, x : \{0, 1\}^n \vdash_d \texttt{last-active}_n : \mathbb{N}$ as follows:

$$\begin{aligned} \texttt{last-active}_{n+1} := {} & \texttt{if}\ (\pi_{n+1}(x) == 1)\ \texttt{return}(n + 1) \\ & \texttt{else let}\ x = (\pi_1(x), ..., \pi_n(x))\ \texttt{in last-active}_n \\ \texttt{last-active}_1 := {} & \texttt{if}\ (\pi_1(x) == 1)\ \texttt{return}(1)\ \texttt{else return}(0). \end{aligned}$$

Then, we can write the a term $\Gamma, u : \mathbb{R}_+ \vdash_p \texttt{truncated-sample} : \{0, 1\}^\infty \times \mathbb{N}$ as follows:

$$\begin{aligned} \texttt{truncated-sample} := {} & \texttt{case}\ (\lfloor -\log_2(u) \rfloor, ())\ \texttt{of} \\ & (n, ()) \implies \texttt{let}\ x = t_n\ \texttt{in return}\ ((x, 0, 0, ...), \texttt{last-active}_n) \end{aligned}$$

Given these definitions, the program in Fig. 3 represents exactly the distribution of interest on $\{0, 1\}^\infty$.

4. Inference

There is an obvious program transformation, based on unfolding, for a language with `slice-let` that can used as an approximate representation for an inference algorithm. In general, sampling based inference algorithms in probabilistic programming languages requires exact samples from the prior distribution. Each sample is weighted by the score of the trace. For a language with `slice-let`, we allow the user to pre-select the level of approximation they want, say k. Given this approximation level, we can sample from the k-step Markov transition kernel $(P')^k$ defined earlier. As $k \to \infty$, the distribution of the sample converges in several metrics. We are also devising a MCMC algorithm for the construct.

References

Ackerman, N. L., J. Avigad, C. E. Freer, D. M. Roy, and J. M. Rute (2017). "On Computable Representations of Exchangeable Data". In: *PPS Workshop, Principles of Programming Languages (POPL 2017)*.

Goodman, N. D., V. K. Mansinghka, D. Roy, K. Bonawitz, and J. B. Tenenbaum (2008). "Church: A language for generative models". In: *In Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI 2008)*.

Neal, R. M. (2003). "Slice sampling". *Annals of statistics*, pp. 705–741.

Papaspiliopoulos, O. and G. O. Roberts (2008). "Retrospective Markov chain Monte Carlo methods for Dirichlet process hierarchical models". *Biometrika* 95.1, pp. 169–186.

Staton, S., H. Yang, F. Wood, C. Heunen, and O. Kammar (2016). "Semantics for Probabilistic Programming: Higher-order Functions, Continuous Distributions, and Soft Constraints". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '16. New York, NY, USA: ACM, pp. 525–534. ISBN: 978-1-4503-4391-6.

Teh, Y. W., D. Görür, and Z. Ghahramani (2007). "Stick-breaking construction for the Indian buffet process". In: *Artificial Intelligence and Statistics*, pp. 556–563.