Extensible Probabilistic Programming

Eyal Dechter & Matt Johnson

May 20, 2013

1 Overview of Probabilistic Programming

The goal of probabilistic programming is to enable the user to perform probabilistic inference by writing down a probability distribution in a purely declarative fashion. Thus, probabilistic program is to probabilistic inference what logic programming is to first order logic: instead of specifying what the computer should do, the user specifies constraints on the computer's output, leaving hidden the mechanism and algorithm by which the computer finds the answer. In logic programming, the user writes down a set of logical statements and can then ask various queries of the system. Similarly, in probabilistic programming, the goal is to be able to accommodate various queries from the declared probability distribution, such as sampling from the distribution and reporting statistics related to the distribution.

How does one declare a probability distribution? Various alternatives exist in the literature. One class of probabilistic programming languages extends logic programming to accommodate probabilities; these include Markov Logic [7] and Independent Choice Logic [6]. Another class of languages captures directly the structure of a "graphical model," a representation of probability distributions as a graph in which component distributions represent nodes and edges represent dependencies; for an exmple of this, see BUGS [3] or JAGS [5].

Sampling procedures or "generative processes" are an alternative way to represent probability distributions simply as the output of a stochastic sampling procedure, i.e., a computer program with random primitives. A large number of languages, including the one we present here, use functional or imperative languages with stochastic primitives as the backbone of a probabilistic programming language. In general, one also needs to be able to condition on one or more of the variables in the program taking on a specific value. These languages usually contain explicit constructs that enable such conditioning. As examples of these languages see BLOG [4] or Church [2].

This last approach is by far the most flexible of the three presented above. It accommodates, after all, the distribution over the output of any program that one could write in, for example, scheme. How can we perform inference for arbitrary programs? A special case of this problem, after all, is to observe the output of any deterministic scheme function and return a satistfying set of inputs.

In the next two section, we will describe the Metropolis-Hastings algorithm (known as MH) which can, in principle, be used to perform inference over arbitrary program traces. However, MH is a slow and approximate algorithm, with no guarantee of accuracy for any finite amount of running time. Specialized inference algorithms for particular families of probability distributions exist throughout the statistical inference literature. Discrete graphical models over large numbers of variables can be solved exactly and efficiently if the model has low "tree-width." Continuous probability distribution whose gradients we can evaluate can be sampled from efficiently with Hamiltonian Monte Carlo [1]. And collections of probability distributions that are "conjugate" to another can often be queried very quickly; such conjugate pairs of distributions exist for any distribution with the "exponential family" distributions.

One of the goals of this project is to show how an MH based probabilistic program can be extended to use specialized inference algorithms.

2 Rejection Sampling

We briefly note that a simple universal inference algorithm exists that is simpler than MH algorithm we describe in the next section: rejection sampling. In rejection sampling, we can get unbiased samples from the posterior distribution of any of the random variables in the program by sampling program traces from the prior and keeping only the traces that agree with the `emit` call. Although the rejection sampling algorithm is guaranteed to give exact samples from the posterior, it is not practical for all but the simplest of probability distributions because in most cases the probability of the event on which we are conditioning is vanishingly small. In effect, trying to perform inference in a probabilistic program by rejection sampling would be akin to trying to find a solution to an AMB problem by making random nondeterministic choices and hoping that you hit a valid solution.

3 The Metropolis-Hastings Algorithm

The Metropolis-Hastings Algorithm is a general purpose algorithm for sampling $x \sim p(x)$ in cases where we only have direct access to $p^*(x) \propto p(x)$. MH works by constructing a Markov chain over the state space X whose stationary distribution is $p(x)$. Thus, to specify the algorithm, we need to specify the transition operator $T(x' \mid x)$, i.e. the probability of choosing location x' as the next state given that the current state is x.

To do this, we need to define a proposal distribution $Q(x' \mid x)$. This distribution is up to the user, and is often a relatively local transition. For example, in continuous domains, $Q(x' \mid x) \sim N(x - x', \sigma)$ is a common choice.

Once we have proposed a transition to x', we decide whether to accept that proposal based on the relative value of $p^*(x)Q(x' \mid x)$ vs. $p^*(x')Q(x \mid x')$. Formally, the probability of transitioning from x to x' is given by

$$T(x' \mid x) = Q(x' \mid x) \min\left\{1, \frac{p^*(x')Q(x \mid x')}{p^*(x)Q(x' \mid x)}\right\}. \quad (1)$$

It is easy to show that, under mild conditions, this transition rule results in a Markov chain whose stationary distribution is $p(x)$.

4 The structure of a probabilistic program

The probabilistic programming language we designed is written as regular scheme code except for one special construct, `emit`. Although it is itself just a regular scheme function, semantically, `(emit x x-val)` conditions the probability distribution defined by the rest of the code on the fact that variable `x` is equal to `x-val`. `emit` takes an optional argument that specifies what kind if any observation noise should be associated with the observation of this variable.

An example program and conditions on the output of the sum of two Gaussian variable being equal to ten might look like

```
(define (sum-of-gaussians)
    (let ( (x (gaussian 7 2))
           (y (gaussian 5 3)))
        (emit (+ x y) 10 observation-likelihood)))
```

This elucidates the general structure of a probabilistic program: we write down a regular scheme function that calls some stochastic primitives (that are either user-defined or provided as a base library). At the end of the program, we can condition on any variable or function of variables being equal to some value.

5 The Metropolis-Hastings Algorithm for Program Traces

Every time we run a program that evaluates stochastic functions, the set of random choices that are made, which we will call the *ptrace*, corresponds to a state in the domain of the distribution defined by the program. To transition to another state, we need to change those random choices in some manner. Here we propose a specific proposal distribution over ptraces and show how to calculate the acceptance probability defined in the previous section.

We define a proposal distribution $Q_{ptrace}(\bar{x}' \mid \bar{x})$ by choosing a choice point from $\bar{x}$ uniformly at random and running the program from that point forward. Suppose we choose to propose a new ith choice point. Let $\bar{x}_{before}$ specify the choices before the ith choice and let $\bar{x}_{after}$ and $\bar{x}'_{after}$ refer to the choices made after the ith choice in the original ptrace and the proposed one, respectively. Then

$$Q(\bar{x}' \mid \bar{x}) = \frac{1}{\text{length}(\bar{x})} q(x'_i \mid x_i) p(\bar{x}'_{after} \mid \bar{x}_{before}, x'_i) p(y|\bar{x}'). \quad (2)$$

The target distribution we want to sample from

$$p(\bar{x} \mid y) \propto p(y \mid \bar{x}) p(\bar(x)) \tag{3}$$

$$= p(\bar{x}_{before}) p(x_i \mid \bar{x}_{before}) p(\bar{x}_{after} \mid \bar{x}_{before}, x_i) p(y|\bar{x}). \tag{4}$$

So the acceptance ratio in Equation 1 is

$$\frac{p(\bar{x}'_{before}) p(x'_i \mid \bar{x}'_{before}) p(\bar{x}'_{after} \mid \bar{x}'_{before}, x'_i) p(y|\bar{x}')}{p(\bar{x}_{before}) p(x_i \mid \bar{x}_{before}) p(\bar{x}_{after} \mid \bar{x}_{before}, x_i) p(y|\bar{x})} \tag{5}$$

$$\frac{\frac{1}{\text{length}(\bar{x}')} q(x_i \mid x'_i) p(\bar{x}_{after} \mid \bar{x}_{before}, x_i)}{\frac{1}{\text{length}(\bar{x})} q(x'_i \mid x_i) p(\bar{x}'_{after} \mid \bar{x}'_{before}, x'_i)} \tag{6}$$

$$= \frac{p(x'_i \mid \bar{x}'_{before}) p(y|\bar{x}')}{p(x_i \mid \bar{x}_{before}) p(y|\bar{x})} \frac{\text{length}(\bar{x}) q(x_i \mid x'_i)}{\text{length}(\bar{x}') q(x'_i \mid x_i)} \tag{7}$$

$$\tag{8}$$

In words, then, every transition involves picking a random point on the trace, rolling it forward to the emit statement, and then comparing how well the new trace fits the data to how well the old data fits the data. If we follow the MH acceptance rule when choosing whether to accept or reject the new state, then the stationary distribution of the Markov chain induced by these transitions tends towards the target distribution. A schematic of this process, within a concrete example, is shown in Figure 1.

6 Conjugate pairs

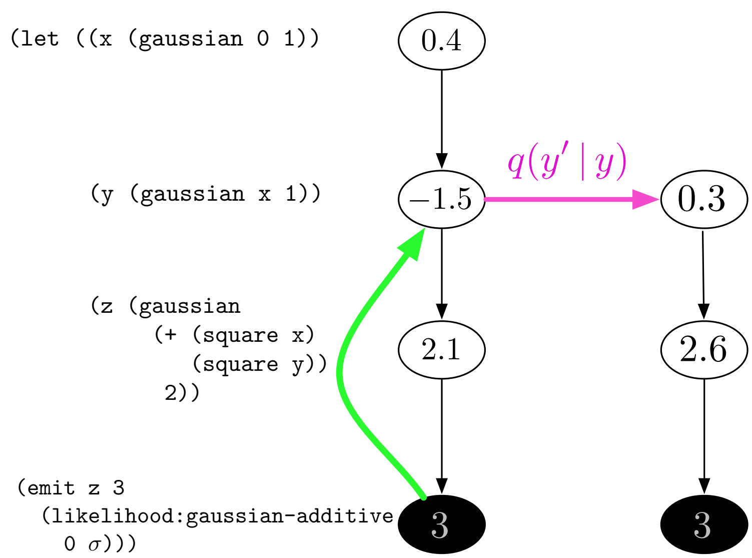
There are many pairs of probability distributions such that if the first is a prior and specific hyperparameter of the second, then many of the queries we want to ask of the distribution can be solved in closed form. One of the simplest examples of such a pair of distributions – which are called "conjugate" pairs" – is the Beta-Bernoulli pair.

The Beta distribution is a distribution over values between 0 and 1. The Bernoulli distribution is a distribution over coin flips that takes as a parameter a probability between 0 and 1 (i.e. the weight of the coin). Thus, the Beta-Bernoulli distribution places a distribution over coin-flips where the underlying coin weight is unknown.

A sampling procedure for this can easily be written as follows

```
(define (foo)
    (let ((p (beta 1 2))
          (x (bern p)))
        x)).
```

Now, suppose that we have seen many coin flips from the same coin and want to reason about the distribution of coin weights that could have given rise to



$$
\begin{aligned}
p(y' = 0.3, z' =& 2.6 | y = -1.5, z = 2.1) \\
&= q(0.3 | -1.5) \min \left\{ 1, \frac{\mathcal{N}(0.3; 0.4, 1)\mathcal{N}(3; 2.6, \sigma)}{\mathcal{N}(-1.5 | 0.4, 1)\mathcal{N}(3; 2.1, \sigma)} \right\}
\end{aligned}
$$

Figure 1: The Metropolis-Hastings algorithm over program traces records the stochastic choice points in the ptrace (i.e. the probabilistic trace). When it hits an emit declaration, it chooses random choice point, proposes a new value for that choice from $q(\cdot|\cdot)$ and rolls the program forward from that point. It accepts or rejects the new trace according the Metropolis-Hastings acceptance ratio.

this sequence of coin flips. Using the syntax we have presented, we could write a distribution of this coin weight as follows:

```
(define (test-beta-bern)
  (let* ((p (beta 1 2))
         (x1 (bern p))
         (x2 (bern p))
         (x3 (bern p)))
    (emit-bern x1 0)
    (emit-bern x2 0)
    (emit-bern x3 1)
    p))
```

In order to generate samples from the posterior distribution of coin weight `p`, we can either run the general MH sampler over program traces, or we can try to take advantage of the fact that the Beta distribution is conjugate to the Bernoulli distribution. In order to implement the latter algorithm, we create a data structure that records all uses of Beta distributions and the associated Bernoulli flips based on them.

References

[1] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[2] Noah Goodman, Vikash Mansinghka, Daniel Roy, Keith Bonawitz, and Daniel Tarlow. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

[3] David J Lunn, Andrew Thomas, Nicky Best, and David Spiegelhalter. Winbugs-a bayesian modelling framework: concepts, structure, and extensibility. *Statistics and computing*, 10(4):325–337, 2000.

[4] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. 1 blog: Probabilistic models with unknown objects. *Introduction to statistical relational learning*, page 373, 2007.

[5] Martyn Plummer. Jags: A program for analysis of bayesian graphical models using gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing (DSC 2003). March*, pages 20–22, 2003.

[6] David Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1):7–56, 1997.

[7] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.