STA4516: Topics in Probabilistic Programming Fall 2015

Lecture 2 — November 3, 2015

Prof. Daniel M. Roy *Scribe: Mufan (Bill) Li*

1 Overview

In the last lecture we saw an introduction and motivations for studying probabilistic programs.

This lecture partly taught using presentation slides and partly on chalk board. In the presentation, we briefly discussed **exchangeable** structures, **de Finetti's Theorem** and the application to achieve **conditional independence**, and the **computability** properties of these structures. On the chalk board, we saw a lightweight implementation of **probabilistic programs** using a **Metropolis-Hasting** type algorithm.

2 Presentation

Here I add several notes supplementary to the presentation, please see the original slides for full details.

2.1 de Finetti's Theorem

An exchangeable sequence $(Y_1, Y_2, \ldots, Y_N)$ has a fully connected graphical model, which has $\mathcal{O}(N^2)$ number of connections. By applying de Finetti's Theorem, we are able to reduce the number of connections to N, as well as allow **parallel computation** of random numbers since we have conditional independence.

An important remark is the transformation $f(\theta, U)$, where conditioned on θ gives independence and U is a uniform random variable, is **not unique**. Here any measure-preserving transformations of θ and U will give the same results, specifically

$$\psi(U) \overset{d}{=} U \Rightarrow \exists f^{\psi}(\theta, \psi(U)) : \big(Y_1, \ldots, Y_N\big) \overset{d}{=} \big(f^{\psi}(\theta, \psi(U_1)), \ldots, f^{\psi}(\theta, \psi(U))\big)$$

3 Chalk Board

In this section, we discuss a lightweight implementation of probabilistic programs from [1].

3.1 Introduction

Definition 1. *An* ***unconditional probabilistic program*** *is a parameterless function f with an arbitrary mix of stochastic and deterministic elements.*

Informally f is a deterministic program until it calls an external stochastic coroutine to generate randomness. For example, MATLAB calls *rand* to generate a uniform random variable. These coroutines are called **elementary random primitives** (ERPs).

Definition 2. *The families of distributions* $\big(P_t(\cdot,\theta)\big)_t$ *are* ***dominated*** *by some* σ*-finite measure* μ_t *if*

$$\exists \text{ densities } p_t(\cdot,\theta) : P_t(A|\theta) = \int_A p_t(x|\theta)\mu_t(dx)$$

For example, the family of Gaussian distributions has densities $p_t(\cdot|\theta) = \mathcal{N}(\overline{x},\sigma^2)$, where t determines the Gaussian type, and θ determines the parameters (mean and variance) $(\overline{x},\sigma^2)$. Here the Gaussian family is dominated by the Lebesgue measure i.e $\mu_t(dx) = m(dx)$.

Definition 3. $\mathcal{T}$ *is a* ***set of ERP types****, where for each* $t \in \mathcal{T}$*, we have a dominated family of distributions* $\big(P_t(\cdot,\theta)\big)_t$*, with densities* $p_t(x|\theta)$ *where* $\theta \in \Theta_t$*.*

Note μ_t can depend on the ERP type t but not parameters θ. It is also important to note the requirement that these families of distributions to be dominated is rarely mentioned, but implicit in the way that these densities are used.

Here we introduce a simple example in MATLAB.

```
for i = 1:1000
  if (rand > 0.5)
    x(i) = randn;
  else
    x(i) = gammarnd;
  end
end
```

where *rand* generates a uniform random variable, *randn* corresponds to a standard normal, and *gammarnd* is gamma.

Depending on the return values of ERPs, we may have different paths of execution. Here we introduce the **execution trace**, a minimal log of execution containing the requests for ERP and return values, which is sufficient to reconstruct the path of execution.

A possible trace may look like this:

Index	t	θ	Return
1	rand	$[a,b]$	0.7
2	randn	$(\overline{x},\sigma^2)$	-2.7
3	rand	$[a,b]$	0.2
4	gammarnd	(k,ϕ)	1.9
$\vdots$	$\vdots$	$\vdots$	$\vdots$
2000	randn	$(\overline{x},\sigma^2)$	1.5

Here a total of 2000 random choices are made. It is also possible to for it to be random. For example:

```
accept = false;
```

```
n = 0;
list = [];
while not accept
  n = n + 1;
  list = [list n]; % concatenate n to the end of list
  accept = rand < 0.5
end
```

The number of ERPs encountered here is geometrically distributed with mean 2.

3.2 Statistical Inference as Conditional Simulation

Consider the following program

```
X = gammarnd;
for j = 1:1000
  Y(j) = poissonrnd(X);
end
```

This program generates a sequence of Poisson random variables using a gamma prior, i.e. $X \sim$ Gamma, $Y_j \sim$ Poisson$(X^{-1}), Y = (Y_1, \ldots, Y_{1000})$. If Y_j model the number of emails that employee j receives in one day, how would we estimate the (common) rate at which email arrives?

Suppose we had data $y = (y_1, \ldots, y_{1000})$, we could simulate the program many times until we see data $Y = y$. The value X used for that particular simulation is then a perfect sample from the posterior $P(X|Y = y)$. This is also known as **rejection sampling**, **accept-reject**, or **guess and check**.

By **conditional simulation**, we mean a simulation of the program sampled from the distribution over simulations satisfying some predicate. In this case the predicate is "accepts" a simulation if $Y = y$, and rejects it otherwise.

3.2.1 Computing Posterior Distributions

In general sampling and taking expectations from a conditional distribution $P(X|Y = y)$ can be difficult even if we can calculate the joint density $p(x, y)$. The reason is conditional densities has the form

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(x,y)}{\int p(x,y)\tau(dx \times Y)}$$

and the integral on the bottom is generally hard to compute, even when given p and τ. Even if the conditional density is known, to sample efficiently we also need the following

$$P(x \in A|Y = y) = \int_A p(x|y)\tau(dx \times Y)$$

to be computed efficiently as well. In order to address this issue, we consider the Metropolis-Hasting algorithm.

3.2.2 Metropolis-Hasting (MH) Algorithm

MH is a way to sample from the posterior $P(X|Y = y)$ or compute conditional expectations $\mathbb{E}[f(X)|Y = y]$. MH is one of many **Markov chain Monte Carlo** (MCMC) algorithms. For an excellent survey of MCMC algorithms see [2].

We first fix a "proposal distribution", or more formally a **transition probability kernel**

$$Q : X \to M_1(X)$$

on the space X, where $M_1(X)$ is the space of distributions on X. In other words, Q is a family of distributions parameterized by $x \in X$.

We are going to assume that Q is dominated, i.e. there exist a σ-finite measure ν and density q such that

$$Q(x, A) = Q(x)(A) = \int_A q(x, y)\nu(dy)$$

For example, a choice of Q for random walk Metropolis-Hasting (RWMH) is the following

$$\epsilon \sim \mathcal{N}(0, 1),\ Q(x) = \mathcal{L}(x + \epsilon) \overset{d}{=} \mathcal{N}(x, 1)$$

Here $Q(x)$ is a measure, $q(x, y)$ is Gaussian, and $\nu(dy)$ is the Lebesgue measure. This leads to a small perturbation of current value of x. Note the choice of ν is important if the space of interest can be a null set w.r.t. ν. This is possible if the space of interest is lies on a sub-manifold of measure zero, such as a curve in $\mathbb{R}^2$.

We then construct the following Markov chain

```
Initialize $X_0, i = 1$;
repeat
    Sample $X_i' \sim Q(X_{i-1})$;
    with probability $a(X_{i-1}, X_i)$
        Set $X_i = X_i'$ (accept);
    otherwise
        Set $X_i = X_{i-1}$ (reject);
    $i = i + 1$;
until;
```

where we define the function a as

$$a(x, x') = \min\left\{1, \frac{p(x'|y)q(x', x)}{p(x|y)q(x', x)}\right\}$$

The choice of the $a(x, x')$ results in the following property: let $T(x, \cdot) = P(X_1|X_0 = x)$ be the transition probability kernel for the chain $(X_0, X_1, \ldots)$, then

$$P(X \in dx|Y = y)T(x, dx') = P(X \in dx'|Y = y)T(x', dx) \tag{1}$$

Therefore the Markov chain is **reversible**. In other words, the chain $(X_0, X_1, \ldots)$ can be extended to $(\ldots, X_{-2}, X_{-1}, X_0)$ and the backward processes are equal in distribution.

Condition (1) is called **detailed balance**. When (1) holds, it implies that $P(X|Y = y)$ is a stationary distribution: i.e. if we sample $X_0 \sim P(X|Y = y)$, then every $X_j \sim P(X|Y = y)$. Furthermore, if the Markov chain is **ergodic**, then for an arbitrary starting point X_0, we have

$$X_j \xrightarrow{d} X \sim P(X|Y = y)$$

That is, we can generate a sample from the posterior $P(X|Y = y)$ from running the Markov chain from any starting point X_0.

Fact 4. *Only requirement of Q is ergodicity. Here the choice of a will bias Q to achieve detailed balance (1).*

Observation 5. *The MH algorithm allowed us to avoid computing the integral by taking a ratio instead*

$$\frac{P(x'|y)}{P(x|y)} = \frac{P(x', y)}{P(x, y)}$$

This is a result of ergodicity, where we exchange the integral over space to integral over time.

3.2.3 MH in Probabilistic Programming

In order to apply MH, we need to answer the following questions:

1. What is the space X?
   Space of trace.

2. What is the transition probability kernel Q?
   Transition probability from one trace path to another.

3. How to compute a?
   TBA next lecture.

References

[1] David Wingate, Andreas Stuhlmüller, and Noah D. Goodman, *Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation*, AISTATS, 2011.

[2] Gareth O. Roberts, Jeffrey S. Rosenthal, *General state space Markov chains and MCMC algorithms.* Probability Surveys, 2004.