Metropolis-Hastings for Mixtures of Conditional Distributions

Oleg Kiselyov
Tohoku University, Japan
oleg@okmij.org

Abstract

Models with embedded conditioning operations – especially with conditioning within conditional branches – are a challenge for Monte-Carlo Markov Chain (MCMC) inference. They are out of scope of the popular Wingate et al. algorithm or many of its variations. Computing the MCMC acceptance ratio in this case has been an open problem. We demonstrate why we need such models. Second, we derive the acceptance ratio formula. The corresponding MH algorithm is implemented in the Hakaru10 system, which thus can handle mixtures of conditional distributions.

1. Summary

Conditioning has always been the source of challenges for probabilistic programming. Even in case of discrete distributions, where conditioning is not theoretically problematic, there is an ever-present danger that a sampling inference procedure degenerates to the wasteful rejection sampling. With continuous distributions, it is not often clear what meaning to assign to mixture models with embedded conditioning – let alone how to sample from the distribution of these models.

Recently Wingate et al. [5] proposed a general implementation method for Metropolis-Hastings (MH) MCMC, which is used, with variations, in many modern probabilistic programming systems. The method is explicitly designed for probabilistic languages with loops and conditionals and provides a formula – albeit without any justification – for computing the MH acceptance ratio. Although the paper [5] mentions conditioning, it is not included in its described algorithm and not reflected in the acceptance ratio formula. When it comes to conditioning, the implementors are on their own. In particular, neither the original algorithm [5] nor its various alternatives [3] can deal with the models in which conditioning occurs within conditional branches.

Hakaru10 [2] is a probabilistic programming language for general graphical models which supports mixing of any distributions, including conditional ones. It relies on an incremental MH algorithm: 'single-site' proposals followed by only the necessary recomputations of the trace. Hakaru10 seems to be the first MH system that permits conditioning operations anywhere within the model, hence supporting mixing of conditional distributions. The present paper distills and justifies the algorithm. The implementation of Hakaru10 along with many examples is publicly available at `http://okmij.org/ftp/kakuritu/Hakaru10/`.

2. Why Conditioning with Branching?

Typically, a modeler first draws a model specifying all relevant random quantities and their dependencies. Later on, if some quantities are observed to hold particular values, they are annotated as such and the corresponding conditional probability distribution is determined. Probabilistic programming often follows the similar pattern, e.g., [1]. First, a program expressing a model is written: for example, the following trivial model mixing two normal distributions, with the means of resp. 10 and 11 and the same standard deviation 1.

```
p1 = do x ← dist bern 0.5
        y ← if x then dist norm 10 1 else dist norm 11 1
        return (x,y)
```

The program, written essentially in Hakaru10 syntax, denotes the joint distribution of x and y. If y is observed at a particular value, say, 10, we add the conditioning statement to express that fact.

```
p1c' = do
  (x,y) ← p1
  observe (y==10)  -- Not a valid Hakaru10 statement!
  return x
```

The program `p1c'` then denotes the conditional distribution $Pr(x \mid y = 10)$. For continuous distributions, specifying just the observed value is not enough: we have to know the distribution it comes from. For example, the conditioning operation in [3] requires the user to specify not the observation per se but its likelihood function. Either we have to posit that the observation on y was noisy and we know the noise distribution function. If we insist on conditioning on exactly 10, we have to know the probability density for y, which we can obtain from `p1` by marginalizing over x. That is, to properly specify the conditional distribution on `p1` we have to first compute a marginal over `p1`, which is the problem of roughly the same difficulty as the original conditional one.

A more attractive choice is to "push" `observe` into the conditional branches, where the distribution of y is obvious. We end up with the following, this time proper Hakaru10 program:

```
p1c = do x ← dist bern 0.5
         if x then condition 10 norm 10 1
              else condition 10 norm 11 1
         return x
```

Since the result of the **if** statement is unused, there must be a side-effect. As we shall see below, conditioning is indeed a side-effectful operation. The program `p1c` is so simple that one can work out the distibution of x from the basic probability theory: it is a Bernoulli distribution with the probability of True being $\phi(0)/(\phi(0)+\phi(-1))$ where $\phi(x)$ is the PDF of the standard normal distribution. The problem thus is to perform MCMC inference on the program like `p1c` with embedded conditioning, in particular, within conditional branches. Neither Wingate method [5] nor various alternatives described in [3] can handle such models.

Programs like `p1c` arise naturally in probabilistic programming systems that truly support modularity and compositionality. Many programming languages let us combine previously written computations into more complex ones. Probabilistic programming languages should likewise let us combine programs (distributions), including conditional distributions like `p1c'`. Therefore, conditioning expressions can easily end up deeply embedded into a complex, composed computation. Paper [2, §3] shows a realistic example of such complex conditioning.

3. Models with Branching

As a warm-up, we first consider unconditional mixtures, of the following general pattern

```
p2 = do x ← dist bern a
        y ← if x then do {yt ← et; return yt}
                  else do {yf ← ef; return yf}
        return (x,y)
```

where et and ef are some models (distributions) without conditioning on external observations and a is a probability value. Such models are fully supported by the Wingate et al. algorithm: the paper [5] gives the formula for the MH acceptance ratio – although without any derivation or justification. In fact, the correct formula appears only in the revised version of the paper. In this section we *derive* the Wingate et al. formula. Our method turns out general: it can be extended to include conditioning, as shown in the next section.

In the notation and approach of [4], the acceptance ratio $\alpha(s_1, s_2)$ for transitioning from state s_1 to state s_2 is $\min(1, r(s_2, s_1))$ where

$$r(s_2, s_1) = \frac{\pi(s_2)q(s_2,s_1)}{\pi(s_1)q(s_1,s_2)}$$

and $\pi(s)$ is (proportional to the) the density of the target distribution and $q(s_1, s_2)$ is the proposal kernel.

In the spirit of Wingate et al. [5] we take a model (program) as a directed acyclic graph of elementary random primitives (ERP) like bern in p2. Contra Wingate et al., each ERP is uniquely named. We write $|p|$ for the number of (active) ERPs in program p (see below for 'active'). A sample from p (which we call a trace – as an execution trace of a program) is a set of samples of all ERPs from their respective distributions; the distribution of one ERP may depend on other ERPs: that is, the parameters of an ERP distribution may depend on sample values of other ERPs. Like in Wingate et al., we build a Markov chain over the space of traces, by proposing an update to one ERP sample. (There are exceptions noted below).

The trace of p2 contains x, y, yt and yf (plus ERPs within et and ef which we remember later). Let the current trace (state) be s_1: (x=true, y=yt) and we propose a move to s_2: (x=false, y=yf). Clearly,

$$\pi(s_1) = \pi(x{=}\mathsf{true})\ \delta(y{=}yt)\ \pi(et{=}yt)\ \pi(ef{=}yf)$$

and similarly for $\pi(s_2)$. (To ease notation we do not write conditional dependencies on x). What is $q(s_1, s_2)$? First, we have to select x from all other ERPs eligible for being updated. The uniform selection brings in the factor $1/(1 + |et|)$. Second, we chose to update x from true to false with some probability, say, b_{tf} (let b_{ft} be the probability of the reverse update). Finally, the switch from y=yt to y=yf is deterministic and the corresponding q-factor is 1.

Putting it all together gives the final expression for $r(s_2, s_1)$ as $(1 - a)/a \cdot b_{ft}/b_{tf} \cdot (1 + |et|)/(1 + |ef|)$, which matches the expression for the acceptance ratio in Wingate et al., once we account for all cancellations. Now we see where all the factors come from, which is not that obvious: the likelihoods of submodels in conditional branches turn out to cancel out (but not when there is conditioning on external observations, see below). The first version of Wingate et al. paper did not include the $|et|$-like factors. They come in naturally in our derivation.

In the state s_1 none of the ERPs in ef can possibly have any effect on the other ERPs in the program, so there is no sense of selecting them for update. Therefore, $|p|$ counts only those ERPs whose update makes sense (may influence the last, in topological order, ERP in the model – the program result). We call these ERPs active, and their count may depend on the state – as it does for p2.

If et and ef have the same number of ERPs and if the proposal to update x is taking a sample from its distribution (that is, $b_{ft} = a$) then the acceptance ratio becomes 1, which agrees with our expectations. If et has one ERP but ef has three, then the acceptance ratio is 0.5, which again makes sense if we consider the distribution of x values in the resulting MCMC chain: if x is false we have half as many chances to select it for update compared with it being true, so we make it twice as likely to stay at true than to switch to false. In the result, the frequency of x being true in the chain approaches a, as expected.

The just considered transition updates not only x from true to false but also y, from yt to yf. The need for such a complex update becomes clear if the distributions of et and ef have non-overlapping supports. Thus our proposals are not always 'single-site', strictly speaking.

4. Models with Conditioning and Branching

Now we consider the case when et or ef in p2, or both, contain conditioning on external observations. To be explicit, let z be a random variable in the model that is later observed (to say, as 0).

```
p3 = do x ← dist bern a
        (y,z) ← if x then do {(yt,zt) ← et; return (yt,zt)}
                     else do {(yf,zf) ← ef; return (yf,zf)}
        return (x,y,z)
```

and p3c be p3 conditioned on z=0. Let $\pi(x, y, z)$ be the target distribution of p3. The distribution of p3c then is proportional to $\pi(x, y, z{=}0)$.

If we again consider the MH on p3c and the transition from s_1: (x=true, y=yt) to s_2: (x=false, y=yf), we now have

$$\pi(s_1) = \pi(x{=}\mathsf{true})\ \delta(y{=}yt)\ \pi(et{=}yt)\ \pi(ef{=}yf)\ \pi(zt{=}0)$$

Hence the function $r(s_2, s_1)$ to compute the acceptance ratio is

$$r(s_2, s_1) = \frac{\pi(x{=}\mathsf{false})}{\pi(x{=}\mathsf{true})}\ \frac{\pi(zf{=}0)}{\pi(zt{=}0)}\ \frac{b_{ft}}{b_{tf}}\ \frac{1+|et|}{1+|ef|}$$

Compared to the formula in the previous section, there is a new factor that scores the observation z=0 within the distributions of et and ef. Now, submodels in conditional branches do influence the acceptance ratio for the move to switch branches.

Overall, we obtain the following algorithm. In Hakaru10, an ERP is a tuple of the current sampled value, its log likelihood (LL), the distribution and its parameters – and a procedure to recompute the sample or LL should the parameters (which depend on sample values of other ERPs) change. The overall Hakaru10 algorithm is not unlike Algorithm 2 in [5] (elided to save space), described in more detail in [2]. At each step one ERP among active is chosen and resampled; the update procedures of all transitively dependent ERPs are run. If an update procedure changes LL of an ERP, it adds the difference to the global variable LLcum, initialized to 0 at the beginning. After the updates are finished, LLcum hence contains $\log(\pi(p_{new})/\pi(p_{old}))$, which is one of the factors of in computing the acceptance ratio for the proposed update.

Besides 'ordinary' ERPs like bern, Hakaru10 has special ERPs: for Dirac distribution (the LL is always zero and the update procedure always recomputes the 'sample'), for a conditioned ERP (the 'sample' is fixed at the observed value), and for ERPs for a conditional branch statement **if** test **then** et **else** ef. The update procedure for the latter is as follows. It is run when the value of the test has changed as the result of an update. We assume the global variables Nrem and Nadd initialized to zero, tracking the changes to the number of active ERPs.

```
let (activated, passivated) =
    if test changed from true to false then (ef,et) else (et,ef)
Nrem +:= |passivated|
for each ERP x in passivated:
  if x is a Conditioned ERP, LLcum −:= ll x
Nadd +:= |activated|
for each ERP x in activated:
  if x is a Conditioned ERP, LLcum +:= ll x
```

As a spot check, for `p1c` we have $\pi(zt{=}0)$ is $\phi(10-10)$ and the acceptance ratio for the proposal to randomly switch x from `true` to `false` is $\phi(-1)/\phi(0)$, which is the expected result.

Acknowledgments

Discussion with Daniel E. Huang and Yufei Cai are gratefully acknowledged. The work on Hakaru10 was supported by DARPA grant FA8750-14-2-0007.

References

[1] N. D. Goodman, V. K. Mansinghka, D. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: A language for generative models. In D. A. McAllester and P. Myllymäki, editors, *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 220–229, Corvallis, Oregon, 9–12 July 2008. AUAI Press. ISBN 0-9749039-4-9. URL `http://web.mit.edu/droy/www/papers/church_GooManRoyBonTenUAI2008.pdfhttp://uai2008.cs.helsinki.fi/UAI_camera_ready/goodman.pdf`.

[2] O. Kiselyov. Probabilistic programming language and its incremental evaluation. In *APLAS 2016: Asian Symp. Progr. Lang. Systems*, number 10017 in Lecture Notes in Computer Science, Berlin, 21–23 Nov. 2016. Springer.

[3] A. Ścibior, Z. Ghahramani, and A. D. Gordon. Practical probabilistic programming with monads. In *Proceedings of the 8th ACM SIGPLAN Symposium on Haskell*, pages 165–176, New York, 2015. ACM Press. ISBN 978-1-4503-3808-0.

[4] L. Tierney. A note on Metropolis-Hastings kernels for general state spaces. *The Annals of Applied Probability*, 8(1):1–9, Feb. 1998.

[5] D. Wingate, A. Stuhlmüller, and N. D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *AISTATS 2011*, number 15, pages 770–778, Cambridge, 2011. MIT Press. Revision 3. February 8, 2014.