Learning Stochastic Inverses for Adaptive Inference in Probabilistic Programs

Andreas Stuhlmüller
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology

Noah D. Goodman
Department of Psychology
Stanford University

Abstract

We describe an algorithm for adaptive inference in probabilistic programs. During sampling, the algorithm accumulates information about the local probability distributions that compose the program's overall distribution. We use this information to construct targeted samples: given a value for an intermediate expression, we stochastically invert each of the steps giving rise to this value, sampling *backwards* in order to assign values to random choices such that we get a likely parse of the intermediate value. We propose this algorithm for importance sampling and as a means of constructing blocked proposals for a Metropolis-Hastings sampler.

1 Introduction

Probabilistic programming is a recent merger between programming languages and Bayesian statistics that enables rapid development of complexly structured probabilistic models. Given a probabilistic program, the problem of inference is typically formulated in terms of conditioning: the program samples from a joint distribution $p(x, y)$, and our goal is to sample from a conditional distribution $p(x|y)$. Implementations of probabilistic programming languages provide generic algorithms that attempt to solve this problem.

Two considerations guide us in the design of new inference algorithms: first, we aim to provide algorithms that efficiently solve common inference problems, and, second, we aim to model human reasoning. Taken together, these goals suggest two desiderata for algorithms: We want inference to be *adaptive* in the sense that our algorithms should improve at solving a given inference problem over time, inspired by the way people learn to reason about a problem type. We further want inference to be *compositional* in the sense that the algorithms decompose a big, challenging query into smaller queries that are easier to solve.

In the following, we describe how to extend existing importance sampling and MCMC algorithms such that they more strongly exhibit these properties. We outline the dimensions along which adaptive inference algorithms vary and explain where our proposed algorithm is located in this space. In our analysis, we focus on functional probabilistic languages, using Church (Goodman et al., 2008) and its foundation, the stochastic lambda calculus, as representatives for this class.

2 Learning stochastic inverses

The goal of adaptive inference is to improve the quality and/or efficiency of sampling over time, ideally converging to an efficient source of exact samples from the posterior distribution. Adaptive algorithms vary along a number of dimensions:

What are the units we learn? There are many choices, including parameters that depend on what underlying inference algorithm we use, e.g., the step size and number of leapfrog steps for Hamiltonian Monte Carlo. We will focus on learning about the distribution of the model.

```
(define (noisify value)
  (gaussian value 2))
(define reflectance
  (gaussian 1 1))
(define illumination
  (+ (gamma 9 .5)
     (gaussian 1 .5)))
(define luminance
  (* reflectance
     illumination))

(noisify luminance)
```

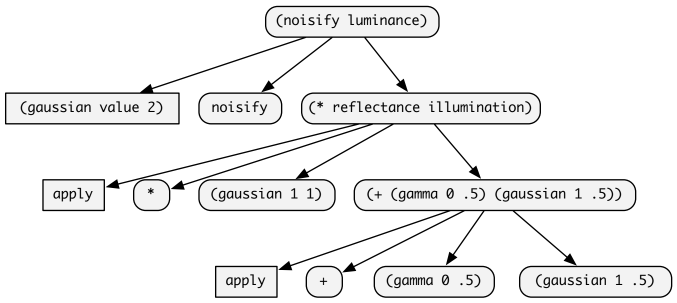

Figure 1: Left: A probabilistic program modeling brightness constancy in visual perception. Right: an execution trace corresponding to the program's evaluation.

In functional probabilistic programs, this distribution exhibits a clear independence structure. The evaluation of a functional program induces a tree of nested function applications—an execution trace. Figure 1 shows an example. For any application $(A_1\, A_2\, \ldots\, A_n)$, the probability of returning a value v is

$$p(v) = \int \cdots \int p(a_1)p(a_2)\ldots p(a_n)p(a_1(a_2,\ldots,a_n) = v)\, \mathrm{d}a_1\, \ldots\, \mathrm{d}a_n.$$

This suggests that candidates for learning include priors $p(a_1), p(a_2), \ldots, p(a_n)$, conditional distributions $p(a_1, a_2, \ldots, a_n|v)$, and joint distributions $p(a_1, a_2, \ldots, a_n, v)$. The dynamic programming algorithm presented in Stuhlmüller and Goodman (2012) can be viewed as an approach to learning about priors. Adaptive importance sampling algorithms such as AIS-BN learn about conditional distributions $p(r|c)$ where r is a primitive random choice and c the global query condition (Cheng and Druzdzel, 2000). We are interested in learning about local conditional distributions $p(a_1, a_2, \ldots, a_n|v)$, which we call *stochastic inverses*. Taken together, these inverses provide an alternative factorization of the program's joint distribution. For example, for the single application shown above, $p(a_1)p(a_2)\cdots p(a_n)p(v|a_1,\ldots,a_n)$ is replaced with the inverse factorization $p(v)p(a_1, a_2, \ldots, a_n|v)$.

How do we learn the units given program executions? In other words, how do we turn the information we gather during program evaluation into estimates for the units of interest? In general, many machine learning methods are of potential relevance, including discriminative methods.

We take a local density approximation approach.[1] Whenever we encounter an application $(A_1\, A_2\, \ldots\, A_n)$ during evaluation, we store the values $\vec{a} := a_1, a_2, \ldots, a_n$ together with application return value v. Figure 2 shows the local joint distributions for the program in Figure 1. We can view these joint distribution samples as an implicit representation of the approximate conditional distribution: In order to sample from the conditional distribution $p(\vec{a}|v)$ given a new value v^*, we find the subset of points $\{(\vec{a}^{(i)}, v^{(i)}), \ldots, (\vec{a}^{(j)}, v^{(j)})\}$ containing nearest neighbors to v^*. We construct a kernel density estimate of the preimage distribution from these neighbors, and sample a preimage $\vec{a}^*$ from this density (Algorithm 1).

For these estimates to be accurate, our sampled program executions need to provide unbiased estimates of the distributions of interest. This can be achieved by computing the estimates using samples from the unconditioned program. In the discussion, we consider the case where the program executions are generated in the process of approximate inference.

How do we use the units to change the inference process? This depends on the underlying inference algorithm. We consider two cases, importance sampling and MCMC.

In order to use stochastic inverses for importance sampling, we simply sample according to the alternative factorization described above. Unconditioned, this process starts by sampling from the

[1]Density estimation techniques such as Gaussian Processes and infinite Gaussian mixtures are alternatives. However, quality of estimation needs to be balanced against computational cost. It is also unclear whether the priors expressed by such nonparametric techniques mesh with the distributions we actually encounter.

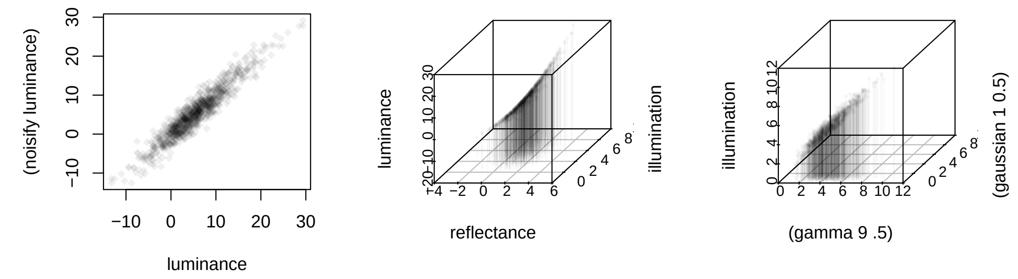

Figure 2: Local joint distributions for the brightness constancy program shown in Figure 1.

Algorithm 1: Given value v for expression e, sample immediate subexpressions.

```
function StochasticInvert($e$, $v^*$)
    find nearest neighbors for value $v^*$ in expression data $\{(\vec{a}^{(1)}, v^{(1)}), (\vec{a}^{(2)}, v^{(2)}), \ldots, (\vec{a}^{(m)}, v^{(m)})\}$
    construct density estimate $p_K$ on nearest-neighbor subexpression values $\{\vec{a}^{(i)}, \ldots, \vec{a}^{(j)}\}$
    draw $\vec{a}^*$ from density estimate
    return $\vec{a}^*$, $p_K(\vec{a}^*)$
```

marginal distribution on the return value of the program. Without loss of generality, we assume that this value represents the Boolean value of the conditioning expression. We set this value to `true` and sample *backwards* using our inverses. This process bottoms out at elementary random choices, which allows us to read off the prior probability p of the sampled execution trace and to compute the importance weight p/q of our sample (Algorithm 2).

For Metropolis-Hastings, we build on the Church MCMC algorithm that executes a random walk on program execution traces (Goodman et al., 2008). To make proposals, this algorithm selects a random choice, proposes a new value, and propagates the changes through the execution trace. We augment this algorithm by introducing a new type of proposal: select a random application node in the execution trace, then regenerate the subtree below this node using stochastic inverses. This sets all random choices in this tree, creating a blocked proposal. Once the random choices are set, we propagate the changes analogous to the original Church MCMC algorithm.

3 Examples: XOR, brightness constancy

Consider the program `(xor (flip)(flip))` conditioned to return `true`. Single-site MCMC cannot switch between the two valid assignments $[1\,0]$ and $[0\,1]$. In contrast, our algorithm can construct blocked proposals that swap both assignments simultaneously: given knowledge of the joint distribution, our algorithm will resample the arguments to `xor` in a way that leaves the condition satisfied. This could be extremely useful if the `xor` expression occurred within a larger program.

As a second example, consider the brightness constancy program shown in Figure 1. Figure 2 shows the corresponding local joint distributions. To sample from the program under the condition that we noisily observed a luminance of 3, we proceed as follows (left to right in the Figure): conditioned on `(noisify luminance)` being 3, we sample a luminance l. Next, conditioned on luminance l, we sample a reflectance r and illumination i. Finally, conditioned on illuminance i, we sample values for the gamma and Gaussian random primitives. Figure 3 shows that the empirical variance of the importance weights for a sampler that follows this procedure goes to 0 as we refine the inverses using forward samples (percentiles computed across 50 runs). If the stochastic inverses are exact, this produces a posterior sample in a way that satisfies the goals we set in the introduction: we have decomposed the overall problem into smaller problems of sampling from local conditional distributions (stochastic inverses), and adaptive learning of these inverses gradually improves inference.

Algorithm 2: Recursively importance-sample expression e conditioned on return value v.

```
procedure InverseImportance($e$, $v$)
    $\vec{a}, q_0 \leftarrow$ StochasticInvert($e$, $v$)
    for $i = 1$ to $|e|$ do
        if $e_i$ is a primitive random choice then
            store value $\vec{a}_i$ for choice $e_i$
            $p_i, q_i \leftarrow p_{e_i}(\vec{a}_i),\ 1$
        else
            $p_i, q_i \leftarrow$ InverseImportance($e_i$, $a_i$)
    return $p_1 \cdots p_n,\ q_0 q_1 \cdots q_n$
```

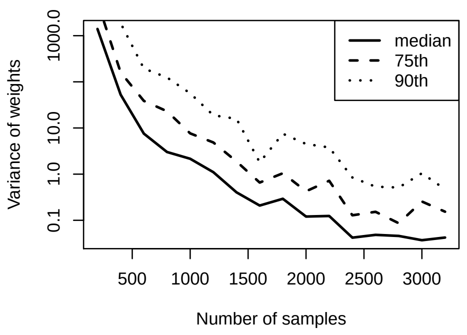

Figure 3: On the right: Results for adaptive importance sampling applied to the program in Figure 1. As the number of samples used to estimate stochastic inverses increases, the empirical variance of the importance weights approaches 0 (implying convergence to a perfect posterior sampler).

4 Related work and discussion

There exists prior work on both adaptive importance sampling and adaptive MCMC. A number of adaptive importance sampling algorithms have been proposed for Bayesian networks, e.g., Shachter and Peot (1989), Cheng and Druzdzel (2000), and Ortiz and Kaelbling (2000). These techniques typically learn importance distributions with the same independence structure as the prior, which in general excludes exact representation of the posterior. Similarly, adaptive MCMC techniques such as those presented in Roberts and Rosenthal (2009) and Haario et al. (2006) are used to tune parameters of the MCMC algorithms, but do not allow arbitrarily close adaptation to the posterior, whereas our method is designed to allow such close approximation. Future work will be required to evaluate whether our methods for importance sampling and MCMC provide better results in practice than these other approaches.

A key question for our method is whether we can use samples that are produced from a distribution that is not the prior and still guarantee eventual convergence of the estimated inverse distributions. Samples generated from the prior may be sparse in regions that have high probability under the posterior, resulting in slow convergence of our stochastic inverses for the problem of interest. We can encourage higher coverage by temporarily increasing the entropy of our random variables, or target the region of interest by using the samples generated during MCMC, however these alternatives both result in biased inverses. When can we correct for this bias, or rely on bias washing out over time?

Acknowledgments

We thank Thomas Icard, Jacob Steinhardt and Lewis Fishgold for useful discussions, and ONR grant N00014-09-0124 for funding.

References

J. Cheng and M. Druzdzel. AIS-BN: An adaptive importance sampling algorithm for evidential reasoning in large bayesian networks. *Journal of Artificial Intelligence Research*, 2000.

N. D. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, pages 220–229, 2008.

H. Haario, M. Laine, A. Mira, and E. Saksman. DRAM: efficient adaptive MCMC. *Statistics and Computing*, 16(4):339–354, 2006.

L. E. Ortiz and L. P. Kaelbling. Adaptive importance sampling for estimation in structured domains. In *Proc. of the 16th Ann. Conf. on Uncertainty in A.I. (UAI-00)*, pages 446–454. Morgan Kaufmann Publishers, 2000.

G. Roberts and J. Rosenthal. Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349–367, 2009.

R. D. Shachter and M. A. Peot. Simulation approaches to general probabilistic inference on belief networks. In *Proc. of the 5th Ann. Conf. on Uncertainty in A.I. (UAI-89)*, pages 311–318, New York, NY, 1989. Elsevier Science.

A. Stuhlmüller and N. D. Goodman. A dynamic programming algorithm for inference in recursive probabilistic programs. *Second Statistical Relational AI workshop at UAI 2012 (StaRAI-12)*, 2012.