Inference with Automatic Gradients in Higher-Order Probabilistic Programs

{Tianlin Shi, Alexey Radul and Vikash Mansinghka} MIT

Introduction

Opportunity Higher-order probabilistic programming languages such as Venture make it easy to represent models that combine

1. discrete latent variables,
2. stochastic control flow, and
3. combinatorial stochastic processes
4. high-dimensional continuous latent spaces.

Challenge These kinds of probabilistic programs present new inference challenges that existing probabilistic programming systems have not addressed:

1. the energy function's curvature can vary unpredictably,
2. the number of continuous dimensions can grow or shrink dynamically,
3. gradients may not be derivable analytically, and
4. the energy functions may be intractable due to the presence of "likelihood-free" simulations.

Solution We describe adaptive hybrid Monte Carlo transition operators and gradient-based MAP schemes for inference in higher-order probabilistic programs that overcome these limitations.

Context

Venture induces a Markov chain on the space of possible probabilistic program execution histories [1]. The current state of the chain is stored in an explicit graph structure that tracks conditional dependence and conditional existence of variables.

Local transitions We realize adaptive hybrid Monte Carlo and gradient-based MAP as local transition operators that change part of the stored execution history. These operators can be mixed with other proposal kernels to compose higher-order inference schemes for various statistical models.

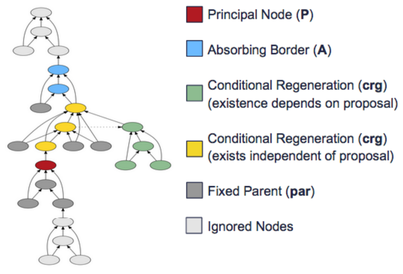


Figure 2: An Example Scaffold

In probabilistic programs, the notion of a Markov blanket needs to be generalized to handle the conditional independence and conditional existence of variables caused by stochastic control flow. In Venture, we call the generalization a **Scaffold**, which is shown in Figure 1.

Math

Local Inference on Scaffold The local inference problem is to sample from

$$\Pr(P|par, A) \propto \Pr(P|par) \sum_{crg} \Pr(crg|par, P) \Pr(A|par, P, crg)$$

Hamiltonian Monte Carlo The Hamiltonian Monte Carlo methods exploits gradient information to make better transitions, through simulating the real values V_P as a particle in the dynamic system:

$$H(V_P) = U(V_P) + K(V_P)$$

where $U(P)$ is the log density at P and $K(P) = V_P^\top M^{-1} V_P/2$ is the kinetic energy.

Conditioning on Regeneration We cannot evaluate or compute the gradient of the natural energy function

$$U(P) \quad = \quad -\log(\Pr(P|par, A))$$

at any given P. Conditional on R, the randomness of sampling crg given P and par, we can evaluate

$$\begin{aligned} U_R(P) &= -\log(\Pr(P|par, A, R)) \\ &= -\log(\Pr(P|par)) - \log(\Pr(A|par, P, R) \\ &= -\log(\Pr(P|par)) - \log(\Pr(A|par, P, crg_R(par, P)) \end{aligned}$$

We effectively treat the regeneration graph as auxiliary variables and sample them together with the principal node. The implementation accomplishes "sampling" the randomness R by controlling the state of the random number generator available to the probabilistic program.

Reverse-Mode Autodifferentiation We compute the needed gradient of the energy function U_R by reverse-mode automatic differentiation. See [3] for more details. In the terminology of automatic differentiation, the scaffold functions as the tape.

Code

```
class GradientOfRegenComputer(object):
  def __init__(self, scaffold):
    self.scaffold = scaffold
    self.rng_state = random.getstate()
  def __call__(self, values):
    # Forward phase
    with reset_rng_state(self.rng_state):
      regen(self.scaffold, values)
    # Reverse phase
    return detach(self.scaffold, compute_gradient=True)
```

```
def proposeVHMC(scaffold):
  q = currentValues(scaffold.principal)
  rhoW = detach(scaffold)
  m = sampleMomenta(q)
  start_K = kinetic(m)
  grad = GradientOfRegenComputer(scaffold)
  def grad_potential(values):
    # The potential function is - log density
    return [-dx for dx in grad(values)]
  (new_q, end_K) = evolve(grad_potential, q, m)
  xiW = regen(scaffold, new_q)
  return xiW - rhoW + start_K - end_K
```

`regen` is a Venture procedure for performing stochastic regeneration on a scaffold (constructing a new conditional regeneration graph). `detach` is a Venture procedure for traversing a scaffold in the reverse order from how regen would have built it. This contract makes `detach` a natural choice for implementing the reverse phase of automatic differentiation.

Multivariate Gaussian

```
[ASSUME x (mv_normal (array 0.0 0.0)
  (matrix (list (list 569.8 147.0)
                (list 147.0 38.9))))]
[INFER (hmc default all .04 47 1)]
```

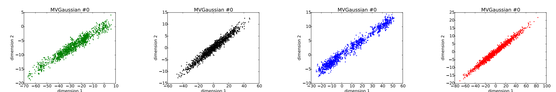


Figure 3: Visual comparison of skewed 2D Gaussian by different methods.

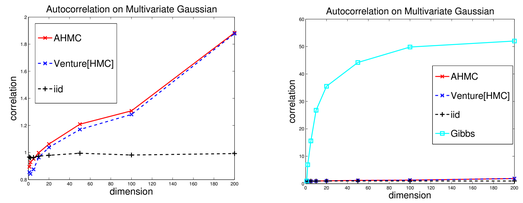


Figure 4: Autocorrelation versus dimension plot for different methods. Gibbs sampling struggles in high dimensions, but gradient based methods do not.

Bayesian Spike-and-Slab

```
[ASSUME sigma2 0.1]
[ASSUME sigmaw2 1000]
[ASSUME z (mem (lambda (j)
  (scope_include 'discrete j (flip 0.5))))]
[ASSUME w (mem (lambda (j)
  (if (z j)
      (scope_include 'weight j
       (normal 0 (* sigma2 sigmaw2)))
      0))))]
[ASSUME dotprod (lambda (dim x)
  (if (= dim 0)
      0
      (+ (* (w (- dim 1))
            (lookup x (- dim 1)))
         (dotprod (- dim 1) x))))]
[ASSUME y (lambda (dim x)
  (normal (dotprod dim x) sigma2))]
[OBSERVE (y 10 (array 37.2 70.8 ...)) 10.3]
...
;;; HMC inference program
[INFER (cycle
  ((mh discrete all 1)
   (hmc weight all 0.04 40 1)) 1)]
;;; MH inference program
[INFER (cycle
  ((mh discrete all 1)
   (mh weight all 1)) 1)]
```

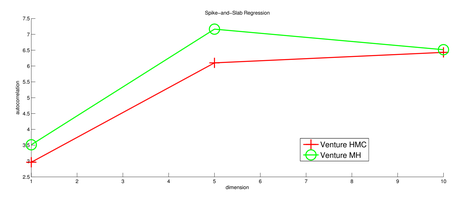
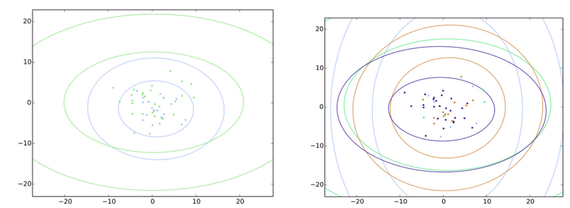


Figure 5: Autocorrelation versus dimension plot for different inference methods for spike-and-slab regression.

DP Mixture of Gaussian

```
[ASSUME prior_mu (array 0.0 0.0)]
[ASSUME prior_sigma
  (matrix (list (list 10.0 0) (list 0 10.0)))]
[ASSUME prior_dof 10]
[ASSUME alpha (scope_include 'hypers 0
  (gamma 1.0 1.0))]
[ASSUME scale (scope_include 'hypers 1
  (gamma 1.0 1.0))]
[ASSUME crp (make_crp alpha)]
[ASSUME cluster (mem (lambda (id)
  (scope_include 'clustering id (crp))))]
[ASSUME mean (mem (lambda (cluster)
  (scope_include 'parameters_mean cluster
   (mv_normal prior_mu prior_sigma))))]
[ASSUME variance (mem (lambda (cluster)
  (scope_include 'parameters_var cluster
   (wishart prior_sigma prior_dof))))]
[ASSUME component_model (lambda (cluster)
  (lambda ()
    (mv_normal (mean cluster)
               (variance cluster))))]
[ASSUME datapoint (mem (lambda (id)
  ((component_model (cluster id)))))]
[OBSERVE (datapoint 0) (array 2.71 3.14)]
...
[INFER (cycle
  ((mh hypers one 1)
   (mh clustering all 1)
   (cycle
    ((map parameters_mean one 0.05 1 1)
     (map parameters_var one 10 1 1)) 10)) 1)]
[SAMPLE (mean 1)]
[SAMPLE (variance 1)]
...
```



Clusters for one random dataset

Clusters for another random dataset

Contribution

1. Extend Hamiltonian Monte Carlo (MHC) methods to probabilitsic programming by mixing over random regenerations, which in the limiting case recovers HMC.
2. Propose gradient-based inference for high-order probablistic programs, and allow mixture of inference schemes.
3. Incorporate the automatic differentiation framework to compute gradients for local transitions.

References

[1] Vikash Mansinghka , Daniel Selsam, and Yura Perov. "Venture: a higher-order probabilistic programming platform with programmable inference." arXiv preprint arXiv:1404.0099 (2014).

[2] Stan Development Team. "Stan: A C++ Library for Probability and Sampling (Version 2.2)". http://mc-stan.org/.

[3] Andreas Griewank, and Andrea Walther. "Evaluating derivatives: principles and techniques of algorithmic differentiation". SIAM, 2008.

[4] David Wingate, et al. "Nonstandard Interpretations of Probabilistic Programs for Efficient Inference." NIPS. 2011.