Efficient Sampling for Probabilistic Programs

Vipul Venkataraman
Aditya Nori, Deepak Vijaykeerthy, Selva Samuel

Microsoft Research India
Outline

› Probabilistic Programming
› Probabilistic Programs
› Inference Techniques
› Hamiltonian Monte Carlo (HMC) Sampling
› Summary
Probabilistic Programming

› Programs in usual languages, such as C, Java, python with two added functionalities:
  • The probabilistic assignment statement \( x \sim Dist(\theta) \)
  • The observe statement \( \text{observe}(\emptyset) \)

› Goal of a probabilistic program: succinctly specify a probability distribution

› Goal of inference: infer the distribution specified by a probabilistic program
Outline

› Probabilistic Programming
› Probabilistic Programs
› Inference Techniques
› Hamiltonian Monte Carlo (HMC) Sampling
› Summary
Simple probabilistic program

```c
bool a, b;
a = Bernoulli (0.5);
b = Bernoulli (0.5);
return (a, b);
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>P(a, b)</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>1/4</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>1/4</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>1/4</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>1/4</td>
</tr>
</tbody>
</table>
bool a, b;
a = Bernoulli (0.5);
b = Bernoulli (0.5);
observe (a || b);
return (a, b);

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>P(a, b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>0</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>1/3</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>1/3</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>1/3</td>
</tr>
</tbody>
</table>
Markov Chains as probabilistic programs

```c
int next_state (int cur_state) {
    bool coin = Bernoulli (0.5);
    switch (cur_state) {
        case (0):
            if (coin) return 1 else return 2;
        case (1):
            if (coin) return 3 else return 4;
        case (2):
            if (coin) return 5 else return 6;
        case (3):
            if (coin) return 1 else return 11;
        case (4):
            if (coin) return 12 else return 13;
        case (5):
            if (coin) return 14 else return 15;
        case (6):
            if (coin) return 15 else return 16;
    }
}
main() {
    int x = 0;
    while (x < 11) { x = next_state(x); }
    return x;
}
```

Knuth-Yao’s technique to obtain fair die from fair coin tosses
True Skill

float skill_a, skill_b, skill_c;
float perf_a1, perf_b1, perf_b2, perf_c2,
    perf_a3, perf_c3;

skill_a = Gaussian(100, 10);
skill_b = Gaussian(100, 10);
skill_c = Gaussian(100, 10);

// first game: a vs b, a won
perf_a1 = Gaussian(skill_a, 15);
perf_b1 = Gaussian(skill_b, 15);
observe(perf_a1 > perf_b1);

// second game: b vs c, b won
perf_b2 = Gaussian(skill_a, 15);
perf_c2 = Gaussian(skill_b, 15);
observe(perf_b2 > perf_c2);

// third game: a vs c, a won
perf_a3 = Gaussian(skill_a, 15);
perf_c3 = Gaussian(skill_b, 15);
observe(perf_a3 > perf_c3);

return(skill_a, skill_b, skill_c);

› Sample perf_a from a noisy skill_a distribution
› Sample perf_b from a noisy skill_b distribution
› if perf_a > perf_b, then a wins, else b wins

skill_a = Gaussian (102.1, 7.8)
skill_b = Gaussian (100.0, 7.6)
skill_c = Gaussian (97.9, 7.8)
Outline

› Probabilistic Programming
› Probabilistic Programs
› Inference Techniques
› Hamiltonian Monte Carlo (HMC) Sampling
› Summary
Sampling

The aim is to solve one or both of the following problems:
› To generate samples \( \{x^{(r)}\}_{r=1}^R \) from a given probability distribution \( P(x) \)
› To estimate expectations of functions under this distribution, for example

\[
\Phi = < \varphi(x) > \equiv \int dx \, P(x) \varphi(x)
\]

› If we solve the first problem, the second can be solved by using the random samples \( \{x^{(r)}\}_{r=1}^R \) to give the estimator

\[
\Phi^* \equiv \frac{1}{R} \sum_{r} \varphi(x^{(r)})
\]
Rejection Sampling

The functions involved in rejection sampling. We desire samples from the target $P^*(x)$. We are able to draw from proposal density $Q^*(x)$, and we know a $c$ such that $cQ^*(x) > P^*(x)$ for all $x$.

A point $(x, u)$ is generated at random in the shaded area under the curve. If this point lies in the darker region then it is accepted. We essentially accept with probability $a = \frac{P^*(x)}{cQ^*(x)}$.

Figure: DJ Mackay, Information Theory, Inference and Learning Algorithms
Metropolis Hastings

- Draw sample for $x'$ from the proposal density $Q(x'; x)$
- Compute $a = \frac{P^*(x') Q(x; x')}{P^*(x) Q(x'; x)}$
- If $a > 1$ then the new state is accepted
- Otherwise, new state is accepted with probability $a$
Sampling in a program

\begin{align*}
a \sim & \text{Normal}(0, 10) \\
\text{if } (a < 0) & \quad b \sim \text{Normal}(a, 15) \\
\text{if } (b < 0) & \quad c \sim \text{Normal}(b, 25) \\
\text{else} & \quad c \sim \text{Normal}(b, 30) \\
\text{else} & \quad b \sim \text{Normal}(a, 20) \\
\text{if } (b < 0) & \quad c \sim \text{Normal}(b, 30) \\
\text{else} & \quad c \sim \text{Normal}(b, 25) \\
\text{observe } & \quad (a < b) \\
\text{return } c;
\end{align*}

\textbf{Theorem:} \ P \equiv \text{Pre}(P) \\
A. Nori, C. Hur, S. Rajamani, S. Samuel. \\
R2: An efficient MCMC Sampler for probabilistic programs, AAAI '14.
Evaluation

Number of Samples

- Model: 10000, 20000, 20000, 10000
- True Skill: 35000, 4000, 5000, 500
- Linear Regression: 50000, 50000, 50000, 500
- HIV: 50000, 50000, 50000, 1000

Legend:
- R2 - MH
- Church
- Venture
- STAN
Outline

› Probabilistic Programming
› Probabilistic Programs
› Inference Techniques
› Hamiltonian Monte Carlo (HMC) Sampling
› Summary
HMC versus MH

Hamiltonian Monte Carlo

Metropolis Hastings

- Samples generated from a bivariate Gaussian with correlation $\rho = 0.998$
- The start position in each case is pointed by the arrow
Hamiltonian Monte Carlo Sampling

The probability density $P(x)$ of a model can be written in the form

$$P(x) = \frac{e^{-E(x)}}{Z}$$

We augment the space variable $x$ by momentum variables $p$. The total energy is defined by the Hamiltonian

$$H(x, p) = E(x) + K(p)$$

We now simulate the Hamiltonian dynamics on the system

$$\dot{x} = p$$

$$\dot{p} = -\frac{\partial E(x)}{\partial x}$$
Hamiltonian model

 › We approximate the simulation through leapfrog discretization

\[
p\left(\tau + \frac{\epsilon}{2}\right) = p(\tau) - \frac{\epsilon}{2} \frac{\partial E}{\partial x}(x(\tau))
\]

\[
x(\tau + \epsilon) = x(\tau) + \epsilon p\left(\tau + \frac{\epsilon}{2}\right)
\]

\[
p(\tau + \epsilon) = p\left(\tau + \frac{\epsilon}{2}\right) - \frac{\epsilon}{2} \frac{\partial E}{\partial x}(x(\tau + \epsilon))
\]

 › Determining suitable gradient \( \frac{\partial E}{\partial x} \) key for faster convergence
Gradient Computation

› Leapfrog step

\[ p\left(\tau + \frac{\epsilon}{2}\right) = p(\tau) - \frac{\epsilon}{2} \frac{\partial E}{\partial x}(x(\tau)) \]

› Two options:
  • Local gradient
  • Global gradient

› Need to compute global gradients dynamically

› Solution? Backward computation

\[ \log(P(x_a, x_b)) = \log(P(x_a, 0,5)) + \log(P(x_b, x_a, 1)) \]
Gradient Computation

\begin{align*}
a &\sim \text{normal}(0,1); \\
\text{if } (a > 0.5) &
\quad b \sim \text{normal}(a, 2); \\
\text{else} &
\quad c \sim \text{normal}(a, 3); \\
\end{align*}

Joint log probability density

\begin{align*}
\text{if } (a > 0.5) &
\quad \log P(a, b) = \log(P(a)) + \log(P(b|a)) \\
\text{else} &
\quad \log P(a, c) = \log(P(a)) + \log(P(c|a)) \\
\end{align*}

\[ \text{grad}(a) = \frac{\partial}{\partial a} \log(P(a)) + \frac{\partial}{\partial a} \log(P(b|a)) \]
\[ \text{else} \\
\quad \text{grad}(a) = \frac{\partial}{\partial a} \log(P(a)) + \frac{\partial}{\partial a} \log(P(c|a)) \]
Gradient Computation

Linear Regression

```python
array dataX, dataY;
a = Normal (0, 1);
b = Normal (5, 1.8);
inv_noise = Gamma (1, 1);
y = array[dataY.length]
for (i = 1; i <= dataY.length; i++)
{
    y[i] = Normal(a * dataX[i] + b, 1 / inv_noise);
    observe(y[i] == dataY[i]);
}
return(a, b, inv_noise);
```

Dependency graph

```
grad(b) = \frac{\partial}{\partial b} \log(P(b)) + \sum_{i=1}^{n} \frac{\partial}{\partial b} \log(P(y[i]|b))
```
Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>True Skill</th>
<th>Linear Regression</th>
<th>HIV</th>
</tr>
</thead>
<tbody>
<tr>
<td>R2 - HMC</td>
<td>200</td>
<td>7500</td>
<td>500</td>
</tr>
<tr>
<td>R2 - MH</td>
<td>10000</td>
<td>20000</td>
<td>5000</td>
</tr>
<tr>
<td>Church</td>
<td>20000</td>
<td>35000</td>
<td>50000</td>
</tr>
<tr>
<td>Venture</td>
<td>500</td>
<td>4000</td>
<td>50000</td>
</tr>
<tr>
<td>STAN</td>
<td>150</td>
<td>5000</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1000</td>
</tr>
</tbody>
</table>
Outline

› Probabilistic Programming
› Probabilistic Programs
› Inference Techniques
› Hamiltonian Monte Carlo (HMC) Sampling
› Summary
Summary

› Improved performance from R2 – MH
› Handling observe better, through efficient sampling from truncated distributions
› Parallel tempering – run $N$ copies of the system, randomly initialized, at different start points
Thank you!