The Clever Machine

Topics in Computational Neuroscience & Machine Learning

Blog Archives

A Gentle Introduction to Markov Chain Monte Carlo (MCMC)

NOV 19
Posted by **dusti nstansbury**

Applying probabilistic models to data usually involves integrating a complex, multi-dimensional probability distribution. For example, calculating the expectation/mean of a model distribution involves such an integration. Many (most) times, these integrals are not calculable due to the high dimensionality of the distribution or because there is no closed-form expression for the integral available using calculus. Markov Chain Monte Carlo (MCMC) is a method that allows one to approximate complex integrals using stochastic sampling routines. As MCMC's name indicates, the method is composed of two components, the ***Markov chain*** and ***Monte Carlo integration***.

Monte Carlo integration is a powerful technique that exploits stochastic sampling of the distribution in question in order to approximate the difficult integration. However, in order to use Monte Carlo integration it is necessary to be able to sample from the probability distribution in question, which may be difficult or impossible to do directly. This is where the second component of MCMC, the Markov chain, comes in. A Markov chain is a sequential model that transitions from one state to another in a probabilistic fashion, where the next state that the chain takes is conditioned on the previous state. Markov chains are useful in that if they are constructed properly, and allowed to run for a long time, the states that a chain will take also sample from a target probability distribution. Therefore we can construct Markov chains to sample from the distribution whose integral we would like to approximate, then use Monte Carlo integration to perform the approximation.

Here I introduce a series of posts where I describe the basic concepts underlying MCMC, starting off by describing Monte Carlo Integration (https://theclevermachine.wordpress.com/2012/09/22/monte-carlo-approximations/), then giving a brief introduction of Markov chains (https://theclevermachine.wordpress.com/2012/09/24/a-brief-introduction-to-markov-chains/) and how they can be constructed to sample from a target probability distribution. Given these

foundation principles, we can then discuss MCMC techniques such as the Metropolis (https://theclevermachine.wordpress.com/2012/10/05/mcmc-the-metropolis-sampler/) and Metropolis-Hastings (https://theclevermachine.wordpress.com/2012/10/20/mcmc-the-metropolis-hastings-sampler/) algorithms, the Gibbs sampler (https://theclevermachine.wordpress.com/2012/11/05/mcmc-the-gibbs-sampler/), and the Hybrid Monte Carlo (https://theclevermachine.wordpress.com/2012/11/18/mcmc-hamiltonian-monte-carlo-a-k-a-hybrid-monte-carlo/) algorithm.

As always, each post has a somewhat formal/mathematical introduction, along with an example and simple Matlab implementations of the associated algorithms.

Posted in Algorithms, MCMC, Sampling Methods, Statistics

Tags: Gibbs Sampler, Hamiltonian Monte Carlo, Hybrid Monte Carlo, integral approximation, integration, Markov Chain, Markov Chain Monte Carlo, MCMC, Metropolis sampler, Metropolis-Hastings Sampler, Monte Carlo Integration

9 Comments

Monte Carlo Approximations

SEP 22
Posted by **dusti nstansbury**

Monte Carlo Approximation for Integration

Using statistical methods we often run into integrals that take the form:

$I = \int_a^b h(x)g(x)dx$

For instance, the expected value of a some function $f(x)$ of a random variable X

$\mathbb{E}[x] = \int_{p(x)} p(x)f(x)dx$

and many quantities essential for Bayesian methods such as the marginal likelihood a.k.a "model evidence"

$p(x) = \int_\theta p(x|\theta)p(\theta)dx$

involve integrals of this form. Sometimes (not often) such an integral can be evaluated analytically. When a closed form solution does not exist, numeric integration methods (http://en.wikipedia.org/wiki/Numerical_integration) can be applied. However numerical methods quickly become intractable for any practical application that requires more than a small number of dimensions. This is where Monte Carlo approximation comes in. Monte Carlo approximation allows us to calculate an estimate for the value of I by transforming the integration problem into a procedure of sampling values from a tractable probability distribution and calculating the average of those samples. Here's what I mean:

If the function $g(x)$ fullfills two simple criteria, namely that the function is always positive on the interval (a, b)

$$g(x) \geq 0, x \in (a, b)$$

and that the integral of the function is finite

$$\int_a^b g(x) = C < \infty$$

then we can define a corresponding probability distribution on the interval (a, b):

$$p(x) = \frac{g(x)}{C}$$

Another way to think of it is that $g(x)$ is a probability distribution scaled by a constant C.

Using this link between probability distributions $p(x)$ and $g(x)$, we can restate the original integration as

$$I = C \int_a^b h(x)p(x)dx = C\mathbb{E}_{p(x)}[h(x)]$$

This statement says that if we can sample values of x using $p(x)$, then the value of the original integral I is simply a scaled version of the expected value of the integrand function $h(x)$ calculated using those samples. Turns out that the expected value $\mathbb{E}_{p(x)}[h(x)]$ can be easily approximated by the sample mean:

$$\mathbb{E}_{p(x)}[h(x)] \approx \frac{1}{N} \sum_i^N h(x_i)$$

where samples $x_i, i = 1..N$ are drawn independently from $p(x)$. This leads to a simple 4-Step Procedure for performing Monte Carlo approximation to the integral I:

1. Identify $h(x)$
2. Identify $g(x)$ and from it determine $p(x)$ and C
3. Draw N independent samples from $p(x)$
4. Evaluate $I = C\mathbb{E}[h(x)] \approx \frac{C}{N} \sum_i^N h(x_i)$

The larger the number of samples N we draw, the better our approximation to the actual value of I. This 4-step procedure is demonstrated in a some toy examples below:

Example 1: Approximating the integral xe^x

Say we want to calculate the integral:

$$I = \int_0^1 xe^x dx$$

We can calculate the closed form solution of this integral using integration by parts:

$$u = x, dv = e^x$$

$$du = dx, v = e^x$$

and

$$I = uv - \int vdu$$

$$= xe^x - \int e^x dx$$

$$= xe^x - e^x|_0^1$$

$$= e^x(x-1)|_0^1$$

$$= 0 - (-1) = 1$$

Orr…we could calculate the Monte Carlo approximation of this integral.

Step 1 we identify

$$h(x) = xe^x$$

Step 2 we identify

$$g(x) = 1$$

and from this can also determine the probability distribution function $p(x) \in (a, b) = (0, 1)$. According to the definition expression for $p(x)$ given above we detemine $p(x)$ to be:

$$p(x) = \frac{g(x)}{\int_a^b g(x)dx} = \frac{1}{b-a}.$$

Step 3: The expression on the right is the definition for the uniform distribution $Unif(0, 1)$, which is easy to sample from using the MATLAB $\text{rand}()$ (Notice too that the constant $C = b - a = 1$).

Step 4: we calculate the Monte Carlo approximation as

$$I = C\mathbb{E}_{p(x)}h(x)$$

$$\approx \frac{1}{N} \sum_{i=1}^N x_i e^{x_i},$$

where each x_i is sampled from the standard uniform distribution. Below is some MATLAB code running the Monte Carlo Approximation for two different values of N

```
% MONTE CARLO APPROXIMATION OF INT(xexp(x))dx
% FOR TWO DIFFERENT SAMPLE SIZES
rand('seed',12345);

% THE FIRST APPROXIMATION USING N1 = 100 SAMPLES
N1 = 100;
x = rand(N1,1);
Ihat1 = sum(x.*exp(x))/N1

% A SECOND APPROXIMATION USING N2 = 5000 SAMPLES
N2 = 5000;
x = rand(N2,1);
Ihat2 = sum(x.*exp(x))/N2
```

Comparing the values of the variables Ihat1 and Ihat2 we see that the Monte Carlo approximation is better for a larger number of samples.

Example 2: Approximating the expected value of the Beta distribution

Lets look at how the 4-step Monte Carlo approximation procedure can be used to calculate expectations. In this example we will calculate

$\mathbb{E}[x] = \int_{p(x)} p(x)x dx$,

where $x \sim p(x) = Beta(\alpha_1, \alpha_2)$.

Step 1: we identify $h(x) = x$.

Step 2: the function $g(x)$ is simply the probability density function $p(x)$ due the expression for $p(x)$ above:

$p(x)^\star = \frac{p(x)}{\int p(x)dx} = p(x)$.

Step 3 we can use MATLAB to easily draw N independent samples $p(x)$ using the function $\text{betarnd}()$. And finally,

Step 4 we approximate the expectation with the expression

$\mathbb{E}[x]_{Beta(\alpha_1,\alpha_2)} \approx \frac{1}{N}\sum_i x_i$

Below is some MATLAB code that performs this approximation of the expected value.

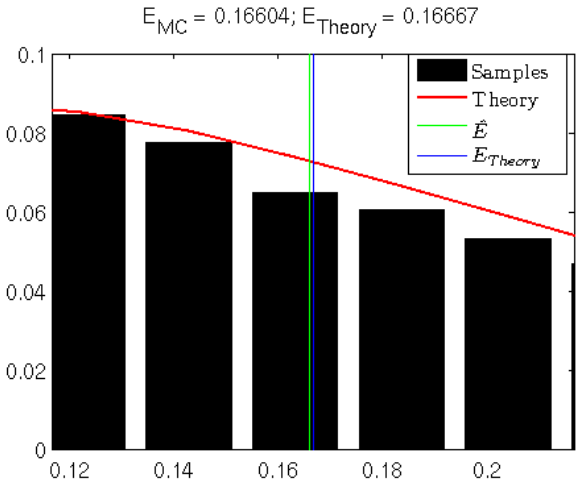
```
rand('seed',12345);
alpha1 = 2; alpha2 = 10;
N = 10000;
x = betarnd(alpha1,alpha2,1,N);

% MONTE CARLO EXPECTATION
expectMC = mean(x);

% ANALYTIC EXPRESSION FOR BETA MEAN
expectAnalytic = alpha1/(alpha1 + alpha2);

% DISPLAY
figure;
bins = linspace(0,1,50);
counts = histc(x,bins);
probSampled = counts/sum(counts);
probTheory = betapdf(bins,alpha1,alpha2);
b = bar(bins,probSampled); colormap hot; hold on;
t = plot(bins,probTheory/sum(probTheory),'r','Linewidth',2)
m = plot([expectMC,expectMC],[0 .1],'g')
e = plot([expectAnalytic,expectAnalytic],[0 .1],'b')
xlim([expectAnalytic - .05,expectAnalytic + 0.05])
legend([b,t,m,e],{'Samples','Theory','$\hat{E}$','$E_{Theory}$'},'in
title(['E_{MC} = ',num2str(expectMC),'; E_{Theory} = ',num2str(expec
hold off
```


And the output of the code:



(https://theclevermachine.files.wordpress.com/2012/09/montecarloexpectation1.png)
Monte Carlo Approximation of the the Expected value of Beta(2,10)

The analytical solution (http://en.wikipedia.org/wiki/Beta_distribution) for the expected value of this Beta distribution:

$\mathbb{E}_{Beta(2,10)}[x] = \frac{\alpha_1}{\alpha_1+\alpha_2} = \frac{2}{12} = 0.167$

is quite close to our approximation (also indicated by the small distance between the blue and green lines on the plot above).

Monte Carlo Approximation for Optimization

Monte Carlo Approximation can also be used to solve optimization problems of the form:

$\hat{x} = \underset{x\in(a,b)}{\text{argmax}}\ g(x)$

If $g(x)$ fulfills the same criteria described above (namely that it is a scaled version of a probability distribution), then (as above) we can define the probability function

$p(x) = \frac{g(x)}{C}$

This allows us to instead solve the problem

$$\hat{x} = \underset{x}{\text{argmax}}\ Cp(x)$$

If we can sample from $p(x)$, the solution $\hat{x}$ is easily found by drawing samples from $p(x)$ and determining the location of the samples that has the highest density (Note that the solution is not dependent of the value of C). The following example demonstrates Monte Carlo optimization:

Example: Monte Carlo Optimization of $g(x) = e^{-\frac{(x-4)^2}{2}}$

Say we would like to find the value of x_{opt} which optimizes the function $g(x) = e^{-((x-4)^2)/2}$. In other words we want to solve the problem

$$x_{opt} = \underset{x}{\text{argmax}}\ e^{-\frac{(x-4)^2}{2}}$$

We could solve for x_{opt} using standard calculus methods, but a clever trick is to use Monte Carlo approximation to solve the problem. First, notice that $g(x)$ is a scaled version of a Normal distribution with mean equal to 4 and unit variance:

$$g(x) = C \times \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-4)^2}{2}}$$

$$= C \times \mathcal{N}(4,1)$$

where $C = \sqrt{2\pi}$. This means we can solve for x_{opt} by drawing samples from the normal distribution and determining where those samples have the highest density. The following chunk of matlab code solves the optimization problem in this way.

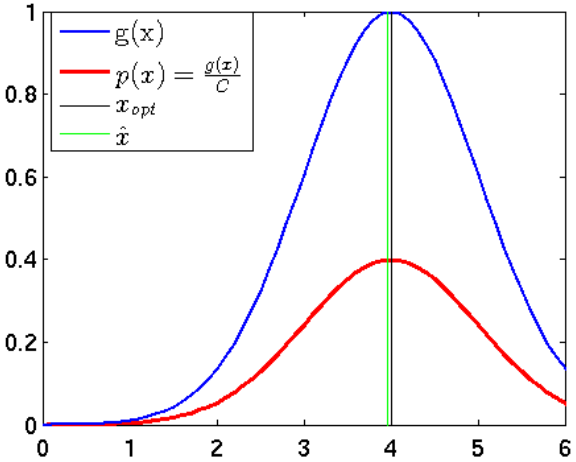
```
% MONTE CARLO OPTIMIZATION OF exp(x-4)^2
randn('seed',12345)

% INITIALZIE
N = 100000;
x = 0:.1:6;
C = sqrt(2*pi);
g = inline('exp(-.5*(x-4).^2)','x');
ph = plot(x,g(x)/C,'r','Linewidth',3); hold on
gh = plot(x,g(x),'b','Linewidth',2); hold on;
y = normpdf(x,4,1);

% CALCULATE MONTE CARLO APPROXIMATION
x = normrnd(4,1,1,N);
bins = linspace(min(x),max(x),100);
counts = histc(x,bins);
[~,optIdx] = max(counts);
xHat = bins(optIdx);

% OPTIMA AND ESTIMATED OPTIMA
oh = plot([4 4],[0,1],'k');
hh = plot([xHat,xHat],[0,1],'g');

set(gca,'fontsize',16)
legend([gh,ph,oh,hh],{'g(x)','$p(x)=\frac{g(x)}{C}$','$x_{opt}$','$\hat{x}$'})
```

(https://theclevermachine.files.wordpress.com/2012/09/montecarlooptimization.png)
Monte Carlo Optimization

In the code output above we see the function $g(x)$ we want to optimize in blue and the Normal distribution $p(x)$ from which we draw samples in red. The Monte Carlo method provides a good approximation (green) to the real solution (black).

Wrapping Up

In the toy examples above it was easy to sample from $p(x)$. However, for practical problems the distributions we want to sample from are often complex and operate in many dimensions. For these problems more clever sampling methods have to be used. Such sampling methods include Inverse Transform Sampling, Rejection Sampling, Importance Sampling, and Markov Chain Monte Carlo methods such as the Metropolis Hasting algorithm and the Gibbs sampler, each of which I plan to cover in separate posts.

Posted in Sampling Methods, Uncategorized

Tags: Monte Carlo Approximation, Monte Carlo Integration, Monte Carlo Optimization 4 Comments