ECON622 Notes: Monte Carlo

Philip Solimine

November 2023

1 Monte-Carlo methods

[1] Probabilistic Machine Learning, Advanced Topics.
Sections 11.1-11.5, (11.6.5), 12.1-12.3, 12.5

In the simplest form, Monte-Carlo methods are all based on the idea that an integral can be approximated by random sampling. That is,

$$\mathbb{E}[f(x)] = \int f(x)p(x)dx \approx \frac{1}{N_s}\sum_{n=1}^{N_s} f(x_n)$$

This is called Monte-Carlo integration. The CLT makes uncertainty quantification in Monte-Carlo easy; we know that sample averages converge at the rate $O(\frac{1}{\sqrt{N}})$, since our "estimator" of the true value of the integral is a sample average, and

$$(\hat{\mu} - \mu) \to \mathcal{N}(0, \frac{\sigma^2}{N_s}).$$

1.1 Aside: Convergence rate

What is amazing about Monte-Carlo is that the convergence rate is independent of the dimension of the integral being evaluated. However, the convergence rate of $O(\frac{1}{\sqrt{N_s}})$ is not great. This has lead to alternative methods like latin hypercubes and quasi-Monte-Carlo (QMC) methods.

These methods work in a similar way but change the way that sampling is done, in order to attempt to get better coverage of the domain. By strategically designing sequences to simulate randomness but improve coverage, QMC methods use "low discrepency sequences", quasirandom (but deterministic) sequences of numbers designed to have good coverage. Sampling from these methods (like the Hammersley sequence) have error bounded as $O(\frac{(\log N_s)^{d-1}}{N})$. In low- to medium- dimensional problems, this is a substantial improvement. However, as d gets large, it will end up being substantially worse and MC will always win out eventually.

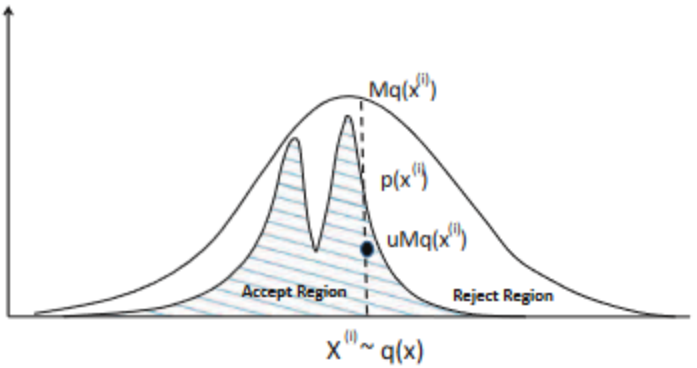

Figure 1: Rejection sampling

2 Sampling

In order to use Monte-Carlo methods at all, we need a way to sample from an arbitrary probability distribution.

2.1 Simplest case: inverse CDF sampling

The simplest case of sampling is the case where we have a univariate distribution with an invertible CDF. Let F be the CDF of our desired distribution and F^{-1} be its inverse. Then

Theorem 1. *If $U \sim U(0,1)$ is a uniform rv, then $F^{-1}(U) \sim F$.*

Proof.

$$P(F^{-1}(U) \leq x) = P(U \leq F(x)) \text{ (applying } F(x) \text{ to both sides)}$$
$$P(U \leq y) = y \Rightarrow = F(x)$$

□

This is conceptually very nice. However, it rules out a lot of distributions.

2.2 Rejection sampling

Most Monte-Carlo methods will therefore be based on another method for sampling, called *rejection sampling*. Rejection sampling is particularly powerful because it only relies on knowledge of a function that is proportional to the distribution (meaning we can often avoid computing normalizing constants)

Suppose we want to sample from a **target distribution** $p(x) = \bar{p}(x)/Z_p$. We have access to a *proposal distribution* $q(x)$, such that $Cq(x) \geq \bar{p}(x)$ for some constant C. The function $Cq(x)$ provides an upper envelope for $\bar{p}$.

First sample $x_0 \sim q(x)$ and $u_0 \sim U(0, Cq(x))$. (u_0 amounts to choosing a height under the envelope $C(q(x))$). Given that $q()$ is the proposal distribution,

$$q(accept \mid x_0) = \int_0^{\bar{p}(x_0)} \frac{1}{Cq(x_0)} du = \frac{\bar{p}(x_0)}{Cq(x_0)}$$

The probability that a value is proposed and accepted is

$$q(\text{propose and accept } x_0) = q(x_0)q(accept \mid x_0) = q(x_0)\frac{\bar{p}(x_0)}{Cq(x_0)} = \frac{\bar{p}(x_0)}{C}$$

Integrating to find the marginal distribution of acceptances

$$\int q(x_0)q(accept \mid x_0)dx_0 = q(accept) = \frac{\int \bar{p}(x_0)dx_0}{C} = \frac{Z_p}{C}$$

And therefore,

$$q(x_0 \mid accept) = \frac{q(x_0, accept)}{q(accept)} = \frac{\bar{p}(x_0)}{C}\frac{C}{Z_p} = p(x_0)$$

Notice that the probability of drawing an accepted sample is proportional to $\frac{1}{C}$. Therefore, to make this process efficient we want to choose *both* a $q(x)$ and C that makes C as small as possible while maintaining that $\bar{p}(x) \leq Cq(x)$. A good $q(x)$ can be learned by variational methods, for example by maximizing the ELBO.

2.3 Importance sampling

Rejection sampling is easily mapped into higher dimensions by using a technique known as **importance sampling**. Importance sampling is simply a [set of] weighting schemes to extend rejection sampling into higher dimensions, instead of simply accepting or rejecting.

If $\phi(\mathbf{x})$ is the target function and $\pi(\mathbf{x})$ is the target distribution, and $q(\mathbf{x})$ is a proposal distribution. Assume we can evaluate $\pi(\mathbf{x})$ (although, a similar method can be used with only a proportional function). Simply sample from $q(\mathbf{x})$ and evaluate

$$\mathbb{E}[\phi(\mathbf{x})] \approx \frac{1}{N_s}\sum_{n=1}^{N_s}\frac{\pi(\mathbf{x}_n)}{q(\mathbf{x}_n)}\phi(\mathbf{x}_n)$$

2.4 Self-normalized importance sampling

Usually we can't evaluate $\pi(\mathbf{x})$ directly. Instead, we have some function that is proportional to π up to the normalizing constant; e.g. $\gamma(\mathbf{x}) = Z\pi(x)$ where $Z = \int \gamma(\mathbf{x})dx$.

Simply define the unnormalized weights $w_n = \frac{\gamma(\mathbf{x}_n)}{q(\mathbf{x}_n)}$ and the normalized weights $W_n = \frac{w_n}{\sum_{k=1}^{n} w_k}$. Then

$$\mathbb{E}[\phi(\mathbf{x})] \approx \sum_{n=1}^{N_s} W_n\phi(\mathbf{x}_n)$$

This also gives a convenient approximation to the normalizing constant

$$Z \approx \frac{1}{N_s}\sum_{n=1}^{N_s} w_n$$

These approximations are consistent but biased.

3 Markov-Chain Monte Carlo (MCMC)

Previous methods work to produce i.i.d. samples, but have some downsides. For example

1. They require a lot of knowledge about the target distribution
2. They require calibration of the envelope or proposal distribution, otherwise can be very inefficient
3. Efficiency diminishes quickly with the dimension

Another option is to use MCMC methods. MCMC produces correlated simulations by creating a Markov chain whose stationary distribution is the target.

MCMC chains can also be slow to mix for high dimensional problems (similar to having a bad envelope), but can often be broken up into smaller local problems that speed convergence. Furthermore, it is very flexible an can be used for any distribution (if you're willing to wait long enough).

MCMC works by performing a random walk on the state space, whose stationary distribution is the target density. They require an initial state; since this state is not drawn from the stationary distribution, it may take a while to get from the starting point to the point where the chain is stationary. This time is called the **mixing** time or **burn-in** time.

3.1 Metropolis-Hastings

Metropolis-Hastings (MH) is one of the simplest forms of MCMC algorithm. MH relies on a transition kernel $q(\mathbf{x}';\mathbf{x})$, a proposal distribution that describes the distribution of transitions from one state to the next. MH is flexible because any proposal distribution can be used, as long as it covers the domain of the target.

If $q(\mathbf{x}';\mathbf{x}) = q(\mathbf{x};\mathbf{x}')$, then the proposal distribution is symmetric (and the algorithm is just called the Metropolis algorithm). In this case, the algorithm proceeds as follows: Repeat:

1. Draw a new proposal state $\mathbf{x}' \sim q(\mathbf{x}';\mathbf{x})$
2. Accept the new state with probability $A = \min\left(1, \frac{p(\mathbf{x}')}{p(\mathbf{x})}\right)$, otherwise set $\mathbf{x}' = \mathbf{x}$

Notice that because the acceptance is a ratio, this does not require the ability to compute the target distribution $p(\mathbf{x})$, only a function $\bar{p}(\mathbf{x}) \propto p(\mathbf{x})$.

Intuitively, this means that we always accept a new sample when it is more probable, but we may also accept the next sample if it is less probable, as long as we are not that much worse off. This ensures that we don't get stuck in a high-probability state, and will eventually explore the domain.

When the proposal distribution is not symmetric ($q(\mathbf{x}';\mathbf{x}) \neq q(\mathbf{x};\mathbf{x}')$), we simply adjust the acceptance probability using the Hastings correction;

$$A = \min(1, \alpha)$$
$$\alpha = \frac{p(\mathbf{x}')q(\mathbf{x};\mathbf{x}')}{p(\mathbf{x})q(\mathbf{x}';\mathbf{x})}$$

The full MH algorithm is the following

Initialize $\mathbf{x}_0$
Repeat:

1. Sample $x' \sim q(\mathbf{x}'; \mathbf{x}_t)$
2. Compute $\alpha = \frac{\bar{p}(\mathbf{x}')q(\mathbf{x};\mathbf{x}')}{\bar{p}(\mathbf{x})q(\mathbf{x}';\mathbf{x})}$
3. Compute $A = \min(1, \alpha)$
4. Sample $u \sim U(0, 1)$
5. Set $\mathbf{x}_{t+1} = \begin{cases} \mathbf{x}' & \text{if } u \leq A \\ \mathbf{x}_{t-1} & \text{otherwise} \end{cases}$

The MH algorithm defines a Markov chain with transitions

$$p(\mathbf{x}'; \mathbf{x}) = \begin{cases} q(\mathbf{x}'; \mathbf{x})A(\mathbf{x}'; \mathbf{x}) & \text{if } \mathbf{x}' \neq \mathbf{x} \\ q(\mathbf{x}; \mathbf{x}) + \sum_{\mathbf{x}' \neq \mathbf{x}} q(\mathbf{x}'; \mathbf{x})(1 - A(\mathbf{x}'; \mathbf{x})) & \text{otherwise} \end{cases}$$

A little bit of math would show that this chain has a unique limiting distribution at $p(\mathbf{x})$. However, we can say nothing about the convergence speed of this chain.

3.2 Gibbs sampling

An issue with MH is that we need to choose a proposal distribution, and also that the acceptance probabilities may be low. Gibbs sampling is a special case of MH that exploits conditional independence assumptions (a graphical model) to automatically create a good proposal.

Gibbs sampling is useful when we know the conditional distributions of all of our variables, and they are easier to sample from. It works by doing ancestral sampling, but conditioning on observations from the past sample.

For example, with 3 variables;

- $x_1^{t+1} \sim p(x_1; x_2^t, x_3^t)$
- $x_2^{t+1} \sim p(x_2; x_1^{t+1}, x_3^t)$
- $x_3^{t+1} \sim p(x_2; x_1^{t+1}, x_2^{t+1})$

Gibbs sampling is a special case of MH with $q(\mathbf{x}'; \mathbf{x}) = p(x_i'; \mathbf{x}_{-i})\mathbb{I}(\mathbf{x}'_{-i} = \mathbf{x}_{-i})$. That is, we move to a new state where x_i is sampled from its full conditional, but $\mathbf{x}_{-i}$ is left unchanged.

Notice that $\alpha = \frac{p(\mathbf{x}')q_i(\mathbf{x};\mathbf{x}')}{p(\mathbf{x})q_i(\mathbf{x}';\mathbf{x})} = \frac{p(x_i';\mathbf{x}'_{-i})p(\mathbf{x}'_{-i})p(x_i;\mathbf{x}'_{-i})}{p(x_i;\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i';\mathbf{x}_{-i})} = \frac{p(x_i';\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i;\mathbf{x}_{-i})}{p(x_i;\mathbf{x}_{-i})p(\mathbf{x}_{-i})p(x_i';\mathbf{x}_{-i})} = 1$, so every proposal is accepted. However, the Gibbs sampler may still be slow to mix, since it only changes one element of $\mathbf{x}$ at each iteration.

3.3 Hamiltonian Monte Carlo (HMC)

Many MCMC algorithms (e.g. Gibbs) can perform poorly in high-dimension, because they rely on local perturbations. HMC is a method that leverages gradient information to make the most out of local moves.

Consider a particle rolling around an energy landscape. The particle is characterized by a position $\theta \in \mathbb{R}^d$ and a momentum $v \in \mathbb{R}^d$. (θ, v) is the phase space. The Hamiltonian is a function that describes the total energy of each possible state as $\mathcal{H}(\theta, v) = \mathcal{E}(\theta) + \mathcal{K}(v)$, where $\mathcal{E}$ is the potential energy and $\mathcal{K}$ is the kinetic energy.

In a statistical case, we usually set the potential energy to $\mathcal{E} = -\log \bar{p}(\theta)$ and the kinetic energy to $\mathcal{K}(v) = \frac{1}{2} v' \Sigma^{-1} v$ where Σ is a positive definite matrix. We then introduce Hamilton's equations, which define stable orbits, or trajectories with constant energy. They are

$$\frac{d\theta}{dt} = \frac{\partial \mathcal{H}}{\partial v} = \frac{\partial \mathcal{K}}{\partial v}$$
$$\frac{dv}{dt} = -\frac{\partial \mathcal{H}}{\partial \theta} = -\frac{\partial \mathcal{E}}{\partial \theta}$$

By integrating these equations (being careful to use a reversible integrator, like the leap-frog integrator), we create a new type of sampling that incorporates momentup into the proposal distribution. That is, for a target distribution of the form

$$p(\theta; v) = \frac{1}{Z} \exp(-\mathcal{H}(\theta, v)) = \frac{1}{Z} \exp\left(-\mathcal{E}(\theta) - \frac{1}{2} v' \Sigma v\right)$$

The marginal distribution over θ (the latent variables of interest) is

$$p(\theta) = \int p(\theta, v) dv = \frac{1}{Z} e^{-\mathcal{E}(\theta)}$$

The HMC algorithm works as follows: Repeat:

1. Generate random momentum $v_{t-1} \sim \mathcal{N}(0, \Sigma)$
2. Set $(\theta_0', v_0') = (\theta_{t-1}, v_{t-1})$
3. Simulate from Hamilton's equations with momentum v_0' for L time steps to get a proposal (θ_L, v_L)
4. Compute $\alpha = \min(1, \exp(-\mathcal{H}(\theta_L, v_L) + \mathcal{H}(\theta_{t-1}, v_{t-1})))$
5. Set $\theta_t = \theta_L$ with probability α, otherwise $\theta_t = \theta_{t-1}$

Notice that there is a hyperparameter here in L, the number of integration steps. The algorithm called NUTS (No-U-Turn Sampling) will choose L automatically, by simulating from the differential equations and terminating the simulation when it starts to reverse direction. The inverse mass matrix Σ is usually taken to be I, or started as I for a warm-up period and then set to the empirical covariance of θ ($\Sigma = \mathbb{E}[(\theta - \bar{\theta})(\theta - \bar{\theta}')]$

Finally, note that HMC methods only work well for continuous random variables, since the Hamilton's equations are defined over a continuous phase space. HMC will mix much faster than Gibbs sampling for variables with high autocorrelation, since the local Gibbs sampling can easily get "stuck" if the individual elements of $\mathbf{x}$ are highly correlated. The random momentum helps HMC break out of these situations.

So it is best to use Gibbs sampling for discrete variables and HMC/NUTS for continuous ones.

3.4 Aside: Numerical integration

So far we have only talked about one way to solve (integrate) a differential equation numerically (Monte-Carlo). Interestingly enough, we derived a method (HMC) that actually relies on having access to another ODE solver.

Solving ODE's with minimal error is one of the most fundamental operations in computational science. Applications in economics primarily to macro and finance, but also in behavioral economics and game theory (e.g. learning and control theory).

The simplest way to simulate from an ODE is using Euler's method. For an initial value problem,

$$\dot{x}(t) = f(x(t)),\ x(0) = x_0,$$

Assume that $f(\cdot)$ is twice differentiable and continuous. The Taylor expansion of $f(t+h)$ about t gives a first-order approximation of the right-hand side as

$$\begin{aligned} f(t+h) &= f(t) + f'(t)(t+h-t) + \frac{f''(\chi)}{2}(t+h-t)^2 \\ &= f(t) + f'(t)h + \frac{f''(\chi)}{2}h^2 \end{aligned}$$

For some $\chi \in [t, t+h]$. This is a first-order Taylor approximation with the Lagrange form of the remainder term.

Rearranging, we get that

$$f'(t) = \frac{f(t+h) - f(t)}{h} + O(h)$$

Many attempts to bound error terms for approximations come from Taylor expansions.

Rearranging the formula to

$$\begin{aligned} x(t+h) &= x(t) + h\dot{x}(t) + O(h^2) \\ x(t+h) &= x(t) + hf(x(t)) + O(h^2) \end{aligned}$$

is the basis of the forward Euler method, which has error bounded as $O(h^2)$.

However, with a differential equation, the error also accumulates, since we are going to continue the approximation from a point that was already wrong. The error accumulates at the rate $O(h)$.

Furthermore, the error accumulations means that the process is not reversible, since going in the reverse direction will create additional error.

The leapfrog method is a second order method, so its error accumulates slower, with the local approximation bounded by $O(h^3)$ and accumulating at the rate $O(h^2)$. This method is to solve problems of the form

$$\begin{aligned}\dot{x}(t) &= v(t)\\ \dot{v}(t) &= f(x(t))\end{aligned}$$

The method works through the algorithm (written in kick-drift-kick form)

$$\begin{aligned}a_t &= f(x(t))\\ v(t+1/2) &= v_t + a_t\frac{\Delta t}{2}\\ x_{t+1} &= x_t + v_{t+1/2}\Delta t\\ v_{t+1} &= v_{t+1/2} + a_{t+1}\frac{\Delta t}{2}\end{aligned}$$

Because of the "leapfrogging", this method is reversible. This means it will conserve energy in the Hamilton's equations above.

bibliography.bib