Intro to Monte Carlo Methods

ISTA 410 / INFO 510: Bayesian Modeling and Inference

U. of Arizona School of Information

March 1, 2021

Outline

Today: several approaches to sampling from tricky distributions

- Rejection sampling
- Gibbs sampling
- Metropolis algorithm

References:

- BDA sections 10.3 (rejection sampling), 11.1 (Gibbs sampling), 11.2 (Metropolis)
- Statistical Rethinking sections 9.1, 9.2 (Gibbs and Metropolis)

Why Monte Carlo?

History

Early history of Monte Carlo methods:

- Late 1940s, early 1950s: Manhattan project
- Stanislaw Ulam, Nicholas Metropolis, John von Neumann
- Difficulty: complex simulations of neutrons in nuclear bombs
  - Supposedly, Ulam was playing solitaire and realized that they could estimate fixed quantities by random simulation.
  - Needed a code name: Metropolis suggested *Monte Carlo*

Random simulation of deterministic quantity

- Say you need to numerically calculate an integral:

$$I = \int_a^b f(x)dx$$

Random simulation of deterministic quantity

- Say you need to numerically calculate an integral:

$$I = \int_a^b f(x)dx$$

- You think back to Calculus II, and you remember a few tricks:
    - Riemann sum (approximate the function by rectangles – constant functions)
    - Trapezoid rule (approximate the function by linear functions)
    - Simpson's rule (approximate the function by quadratic functions)

Random simulation of deterministic quantity

- Say you need to numerically calculate an integral:

$$I = \int_a^b f(x)dx$$

- You think back to Calculus II, and you remember a few tricks:
  - Riemann sum (approximate the function by rectangles – constant functions)
  - Trapezoid rule (approximate the function by linear functions)
  - Simpson's rule (approximate the function by quadratic functions)
- Idea behind all: evaluate the function on a grid
  - Rectangles: about n function evaluations, error goes like $1/n$
  - Trapezoid: about n function evaluations, error goes like $1/n^2$
  - Simpson's rule: about $2n$ evaluations, error goes like $1/n^4$

Random simulation of deterministic quantity

- Say you need to numerically calculate an integral:

$$I = \int_a^b f(x)dx$$

- You think back to Calculus II, and you remember a few tricks:
    - Riemann sum (approximate the function by rectangles – constant functions)
    - Trapezoid rule (approximate the function by linear functions)
    - Simpson's rule (approximate the function by quadratic functions)
- One more idea: sample n random x_i, evaluate the function there, and average
    - n function evaluations, error goes like $1/n^{1/2}$

Random simulation of deterministic quantity

The grid methods are uniformly better

Random simulation of deterministic quantity

The grid methods are uniformly better *in one variable*.

But what if we're calculating a d-dimensional integral?

$$I = \int_{[a_1,b_1]\times\ldots\times[a_d,b_d]} f(\mathrm{x})d\mathrm{x}$$

Then

- Rectangles: n^d function evaluations, error goes like $1/n$
- Trapezoid: n^d function evaluations, error goes like $1/n^2$
- Simpson's rule: $(2n)^d$ evaluations, error goes like $1/n^4$

Random simulation of deterministic quantity

The grid methods are uniformly better *in one variable*.

But what if we're calculating a d-dimensional integral?

$$I = \int_{[a_1,b_1]\times\ldots\times[a_d,b_d]} f(\mathrm{x})d\mathrm{x}$$

Then

- Rectangles: n^d function evaluations, error goes like $1/n$
- Trapezoid: n^d function evaluations, error goes like $1/n^2$
- Simpson's rule: $(2n)^d$ evaluations, error goes like $1/n^4$
- Monte Carlo: n function evaluations, error goes like $1/n^{1/2}$

What does this have to do with inference?

Output of the modeling process: an un-normalized posterior distribution

$$p(\boldsymbol{\theta}|y)$$

Generally interested in the statistics of various parameters θ_i, obtained from marginal posterior:

$$p(\theta_i|y) = \int p(\boldsymbol{\theta}|y) d\theta_1 d\theta_2 \dots d\theta_n$$

Problem: integration is hard.

- Approximate the distribution by something simpler, commonly a normal/normal mixture – leads to methods such as variational Bayes or expectation propagation
- Generate random samples and compute summary statistics – what we've been doing with PyMC3

Sampling probability distributions

What does a RNG give you?

Your computing environment (Python, R, whatever) is equipped with a pseudo-random number generator (PRNG) that drives all random simulation.

What does the PRNG produce?

- A deterministic sequence of numbers that is statistically difficult to distinguish from a truly random sequence of numbers
- Numbers uniformly distributed on $[0, 1)$

Much work has been put into accurately sampling from $\mathrm{Uniform}(0, 1)$.

What happens in `norm.rvs()`

What happens when you call something like `sp.stats.norm.rvs`?

- PRNG does its magic and produces $x \sim \mathrm{Uniform}(0, 1)$
- Apply a transformation to get $z \sim \mathrm{Normal}(0, 1)$

What happens in `norm.rvs()`

What happens when you call something like `sp.stats.norm.rvs`?

- PRNG does its magic and produces $x \sim \mathrm{Uniform}(0,1)$
- Apply a transformation to get $z \sim \mathrm{Normal}(0,1)$

Simplest: inverse-CDF method

$$F(z) = \Pr(Z \leq z)$$

If we have the ability to compute $F^{-1}(x)$, then $F^{-1}(x)$ has the desired distribution

Challenges

Many probability distributions of interest are not so easy to sample from!

- Computing the inverse CDF may be difficult
- If the distribution is not normalized, inverse CDF does not work at all
- Computing the normalizing constant might itself be infeasible (for reasons we considered before)

Rejection sampling

Simple problem: you have a coin with an unknown probability $p \neq 1/2$ of coming up heads. You want to use it to play a game of chance with your friend, but you need fair coin flips for the game.

Rejection sampling

Simple problem: you have a coin with an unknown probability $p \neq 1/2$ of coming up heads. You want to use it to play a game of chance with your friend, but you need fair coin flips for the game.

Solution: rejection sampling

- Flip the coin twice:
  - If the flip is HT, return H
  - If the flip is TH, return T
  - If the flip is TT or HH, discard the result and try again

Rough idea: simulate a "bigger" probability distribution and reject the samples that don't agree with your target distribution

Notice: the further p is from $1/2$, the less efficient

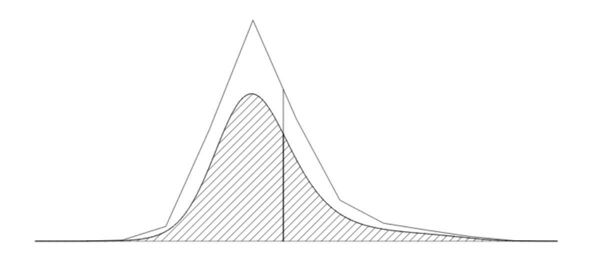
Rejection sampling, generally

More generally, if we have a target density $p(x)$, which may or may not be normalized, we can use rejection sampling with a *proposal distribution* $q(x)$, satisfying $Mq(x) \geq p(x)$, as long as we can sample from $q(x)$.

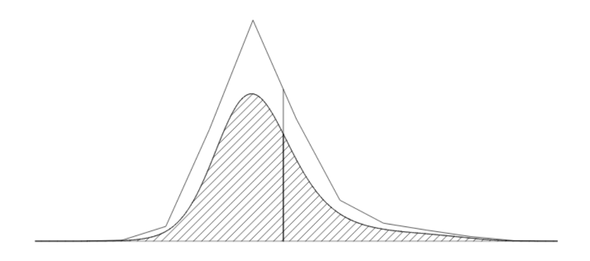
1. Draw a sample x_i from $q(x)$.
2. With probability $p(x_i)/(Mq(x_i))$, keep the sample; otherwise, reject it and try again.

Notice $M > 1$. The closer $q(x)$ is in shape to $p(x)$, the smaller we can make M and the higher our acceptance probability.

Rejection sampling, generally



Rejection sampling, generally



Problems:

- Need a good proposal distribution; otherwise we spend too much time rejecting samples.
- For sufficiently complicated $p(x)$, may be challenging to verify $Mq(x) \geq p(x)$

Markov chain Monte Carlo

What is a Markov chain?

A Markov chain is a model for a random process indexed by discrete time steps $t = 0, 1, 2 \ldots$.

- The values X_t of the chain come from a *state space* S – can be finite, infinite but discrete, or continuous
- The chain moves from state to state randomly, subject to the Markov property:

$$p(X_t | X_{t-1}, \ldots, X_0) = p(X_t | X_{t-1})$$

  i.e. distribution of X_t depends only on value of X_{t-1}, not the full history.
- This means the dynamics can be summarized in a *transition function* $p(x, y) = p(X_t = y | X_{t-1} = x)$. (For finite space chains this is represented as a matrix)

Gibbs sampler

Gibbs sampler

One of the simplest Markov chain algorithms, but still useful, is called the *Gibbs sampler* (named for, but not created by, statistical physicist Josiah Willard Gibbs).

Also called *alternating conditional sampling*:

1. Start with a parameter vector $(\theta_1, \theta_2, \ldots, \theta_d)$
2. In one iteration,
    2.1 Choose an ordering of coordinates 1-d
    2.2 In each coordinate, update θ_j according to its distribution conditional on the other θ_i (which are held fixed)
3. After these d steps, have a new parameter vector $(\theta'_1, \theta'_2, \ldots, \theta'_d)$

Gibbs sampler

The Gibbs sampler is useful when the posterior distribution of each θ_i is, conditional on all other parameters, a distribution we can sample directly from.

True for, e.g.:

- Hierarchical normal model
- Gaussian mixture models
- Many other models that use conjugate distributions

However, it doesn't have the flexibility to sample from an arbitrary joint density $p(\theta_1, \theta_2, \ldots, \theta_d)$.

Stationary distributions

A distribution $\pi(x)$ on the state space of a Markov chain is called *stationary* if it is invariant under the time steps. That is, if $X_t \sim \pi(x), X_{t+1} \sim \pi(x)$.

- Under reasonable assumptions on the chain a Markov chain will always have a unique stationary distribution
- Given any initial distribution $X_0 \sim \pi_0(x)$, the distribution of X_t approaches $\pi(x)$ as $t \to \infty$

Strategy to approximate $p(x)$: devise a Markov chain with stationary distribution $p(x)$, run it for a long time, and take the sequence $X_i, \ldots, X_{i+N}$ as a sample of N observations from $p(x)$.

Metropolis algorithm

The Metropolis algorithm was the first general purpose MCMC algorithm.

Named for Nicholas Metropolis (recall: proposed the name "Monte Carlo")

THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*
(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

Discrete example

We have a factory, and it has 7 machines. Each machine has a known rate of failures $f_i(x)$ (in, say, failures / hour).

Our maintenance robot visits each machine for half-hour shifts to run diagnostics and fix problems. Our robot should visit each machine so that it spends an amount of time proportional to that machine's failure rate.

We want to make a Markov chain of visits that has the right stationary distribution.

Discrete example

A simple algorithm:

1. Order the machines $x_1, \ldots, x_7$.
2. Start at a random machine x_i.
3. Flip a coin to decide whether to propose x_{i-1} or x_{i+1}.
4. Say we chose x_{i+1}: with probability $f(x_{i+1})/f(x_i)$, move to x_{i+1}; otherwise, stay at x_i

Let's write a little code...

Why does this work?

Detailed balance equation:

$$\pi(x)p(x,y) = \pi(y)p(y,x)$$

For any Markov chain with transition function p, if π satisfies this equation, π is stationary. (Intuition: the flow of probability mass from x to y equals that from y to x.)

Why does this work?

Detailed balance equation:

$$\pi(x)p(x,y) = \pi(y)p(y,x)$$

For any Markov chain with transition function p, if π satisfies this equation, π is stationary. (Intuition: the flow of probability mass from x to y equals that from y to x.)

Using the ratio $f(x)/f(y) = p(x)/p(y)$ to set $p(x,y)$ automatically fulfills the detailed balance equation, and importantly does not require us to know the normalized probability distribution $p(x)$, but only the un-normalized distribution $f(x)$.

Metropolis algorithm in general

The general Metropolis-Hastings algorithm:

1. Establish a proposal distribution $q(x, y)$ that is a transition probability on the state space
2. At time t, when $X_t = x$, sample a value from $q(x, y)$
3. Compute the acceptance probability

$$\alpha(x, y) = \min\left\{\frac{p(x)q(y, x)}{p(y)q(x, y)}, 1\right\}$$

   and jump from x to y with that probability, otherwise stay at x.

Notice one difference: factor of q in the acceptance probability. This is Hastings's contribution, and allows for $q(x, y) \neq q(y, x)$.

General difficulties

There are two general issues with using MCMC to sample from a target distribution:

- Burn-in: the chain takes time
- Correlation: we are not sampling independent draws from the target distribution

General difficulties

There are two general issues with using MCMC to sample from a target distribution:

- Burn-in: the chain takes time
- Correlation: we are not sampling independent draws from the target distribution
- Always use multiple chains so we can compare them to help assess convergence to the target distribution
- Compute "effective sample size"

More on this next time.

Summary

We saw two algorithms:

- Metropolis-Hastings
- Gibbs sampler

Weaknesses:

- Like rejection sampling, a lot of dependence on the proposal distribution
- Can easily get “stuck“ and have difficulty exploring the full parameter space
- Gibbs sampling requires that we can sample directly from conditional posteriors

Summary

Next time: Hamiltonian Monte Carlo. Idea: augment MCMC with a physics simulation.

Many improvements:

- Doesn't require choosing a proposal
- Avoids getting stuck; faster mixing, shorter burn-in
- Reduced autocorrelation
- Diagnostic info automatically produced
- It's really cool (personal opinion)