Markov Chain Monte Carlo

Disclaimer: Work in progress. Portions of these written materials are incomplete.

Outline

- Estimating expectations: What are we trying to do here?
- The basic Monte Carlo method
- Rejection sampling
- Importance sampling
- Markov chain basics
- Markov chain Monte Carlo basics
- The Metropolis-Hastings algorithm

The goal: estimate averages

We've got a model p(x | θ).

We've got a prior p(θ).

This implies a posterior p(θ | x) = p(θ, x) / p(x).

We'd like to say something about the *expected value* (i.e., average) under the posterior of some function f(θ).
This can be written as a weighted sum (for discrete θ) or integral (for continuous θ):

$$\mathbb{E}_{p(\theta|x)}[f(\theta)] = \sum_{\theta} p(\theta \mid x) f(\theta) \qquad \mathbb{E}_{p(\theta|x)}[f(\theta)] = \int_{\theta} p(\theta \mid x) f(\theta) d\theta$$

Example 1: A discrete distribution

$$p(\theta = 0) = 0.25; \quad p(\theta = 1) = 0.5; \quad p(\theta = 2) = 0.25$$

$$\mathbb{E}_p[\theta] = 0.25 \cdot 0 + 0.5 \cdot 1 + 0.25 \cdot 2 = 1$$

$$\mathbb{E}_p[\theta^2] = 0.25 \cdot 0^2 + 0.5 \cdot 1^2 + 0.25 \cdot 2^2 = 1.5$$

The goal: estimate averages

We've got a model p(x | θ).

We've got a prior p(θ).

This implies a posterior p(θ | x) = p(θ, x) / p(x).

We'd like to say something about the *expected value* (i.e., average) under the posterior of some function f(θ).
This can be written as a weighted sum (for discrete θ) or integral (for continuous θ):

$$\mathbb{E}_{p(\theta|x)}[f(\theta)] = \sum_{\theta} p(\theta \mid x) f(\theta) \qquad \mathbb{E}_{p(\theta|x)}[f(\theta)] = \int_{\theta} p(\theta \mid x) f(\theta) d\theta$$

Example 2: The exponential distribution

$$p(\theta) = e^{-\theta}$$

$$\mathbb{E}_p[\theta] = \int_0^\infty e^{-\theta}\theta d\theta = \int_0^\infty e^{-\theta}\theta^{2-1} d\theta = \Gamma(2) = 1$$

$$\mathbb{E}_p[\theta^2] = \int_0^\infty e^{-\theta}\theta^2 d\theta = \int_0^\infty e^{-\theta}\theta^{3-1} d\theta = \Gamma(3) = 2$$

$$\mathbb{E}_p[\log(3+\theta^{2.3})] = \int_0^\infty e^{-\theta}\log(3+\theta^{2.3}) d\theta = ?????$$

This part of the course is really all about how to solve integrals we're not smart enough to solve analytically!

The goal: estimate averages

Some examples:

- θ = p(heads).
  x = “I saw heads three times in a row”.
  $f(\theta) = \theta$.
  $E_{p(\theta|x)}[\theta]$ = the probability that the next coin flip will be heads.

- θ = an embedding vector that summarizes our knowledge of a user.
  x = everything we’ve ever seen that user do.
  $f(\theta, v)$ = how much the model predicts the user will like video v.
  $E_{p(\theta|x)}[f(\theta, v)]$ = a guess of how much the user will like video v.
  $E_{p(\theta|x)}[(f(\theta, v) - E_{p(\theta|x)}[f(\theta, v)])^2]$ = the posterior variance of f—a measure of how much more we have to learn.

- θ = the infection rate, incubation time, death rate, etc. of an epidemiological model of COVID-19.
  x_t = number of new deaths on day t.
  $f(\theta)$ = number of predicted deaths next month.
  $E_{p(\theta|x)}[f(\theta)]$ = a guess of the number of deaths next month.
  $E_{p(\theta|x)}[I[f(\theta, v) > 1000]]$ = the probability that the number of deaths next month will exceed 1000.

Some properties of expectations

Linearity:

$$\mathbb{E}[x+y] = \mathbb{E}[x] + \mathbb{E}[y]$$

Independent products:
If x and y are *independent*, then

$$\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y]$$

This is not generally true if x and y are not independent!

Expectation of a non-random variable:
If x is not random, then

$$\mathbb{E}[x] = x$$

Variance:
The *variance* of a random variable x is defined as

$$\mathbb{E}[(x - \mathbb{E}[x])^2] = \mathbb{E}[x^2] - \mathbb{E}[x]^2$$

If we average M samples from the same distribution, the variance drops by a factor of M:

$$\mathbb{E}\left[\left(\frac{1}{M}\sum_m x_m - \mathbb{E}[x]\right)^2\right] = \frac{1}{M^2}\sum_{m,n}\mathbb{E}[(x_m - \mathbb{E}[x])(x_n - \mathbb{E}[x])] = \frac{1}{M^2}\sum_m \mathbb{E}\left[(x_m - \mathbb{E}[x])^2\right] = \frac{1}{M}\mathbb{E}\left[(x - \mathbb{E}[x])^2\right]$$

The Monte Carlo method

Typically the integral

$$\mathbb{E}_{p(\theta|x)}[f(\theta)] = \int_{\theta} p(\theta \mid x) f(\theta) d\theta$$

will be intractable.

But if we can *sample* from p(θ | x), then we can draw M samples $\theta_m \sim p(\theta \mid x)$ and approximate the expectation with

$$\hat{f} = \frac{1}{M} \sum_{m} f(\theta_m)$$

Since
$E[(1/M)\ \Sigma_m\ f(\theta_m)] = E[f(\theta_1)] = \int_\theta\ p(\theta \mid x)\ f(\theta)\ d\theta$.

That is, just look at the empirical mean of f. Simple*!

* If we can sample from p(θ | x).

Accuracy of Monte Carlo

Monte Carlo gives us *estimates* of expectations. How accurate are they?

We can decompose the expected squared error into *bias* and *variance*. Suppose we're sampling from some distribution q(θ) instead of p(θ). Then we have

$$\mathbb{E}_q\left[\left(\mathbb{E}_p[f(\theta)] - \frac{1}{M}\sum_m f(\theta_m)\right)^2\right] = \mathbb{E}_q\left[\left((\mathbb{E}_p[f(\theta)] - \mathbb{E}_q[f(\theta)]) - \left(\frac{1}{M}\sum_m f(\theta_m) - \mathbb{E}_q[f(\theta)]\right)\right)^2\right]$$

$$= (\mathbb{E}_p[f(\theta)] - \mathbb{E}_q[f(\theta)])^2 + \mathbb{E}_q\left[\left(\frac{1}{M}\sum_m f(\theta_m) - \mathbb{E}_q[f(\theta)]\right)^2\right] - 2(\mathbb{E}_p[f(\theta)] - \mathbb{E}_q[f(\theta)])\mathbb{E}_q\left[\frac{1}{M}\sum_m f(\theta_m) - \mathbb{E}_q[f(\theta)]\right]$$

$$= (\mathbb{E}_p[f(\theta)] - \mathbb{E}_q[f(\theta)])^2 + \mathbb{E}_q\left[\left(\frac{1}{M}\sum_m f(\theta_m) - \mathbb{E}_q[f(\theta)]\right)^2\right]$$

$$= (\mathbb{E}_p[f(\theta)] - \mathbb{E}_q[f(\theta)])^2 + \frac{1}{M}\mathbb{E}_q\left[(f(\theta) - \mathbb{E}_q[f(\theta)])^2\right]$$

The term on the left is the squared **bias**—how different is the expectation under q from the expectation under p?

The term on the right is the **variance**—how different is our estimate of the expectation of f from its (possibly biased) mean?

Accuracy of Monte Carlo

If we can directly sample from p(θ | x), the bias is 0, and the variance is the posterior variance divided by M:

$$\frac{1}{M}\mathbb{E}_{p(\theta|x)}\left[\left(f(\theta) - \mathbb{E}_{p(\theta|x)}[f(\theta)]\right)^2\right]$$

Since the *scale* of our error is the square root of the variance, we need to generate 100x as many samples to get one more decimal place of accuracy. Compared to quadrature methods for solving integrals (e.g., the trapezoid rule), this is awful!

So why is Monte Carlo so widely used? For one thing, the accuracy of Monte Carlo estimates does not depend on the dimensionality of θ (if we can generate good samples). Quadrature methods tend to be exponentially expensive in dimensionality.

But there's a subtler reason: Monte Carlo variance is only one source of error in our estimates of the true value of f(θ). Another is *lack of data*—if we don't have data, no amount of Monte Carlo can change that.

For Bayesian inference problems, the error of an estimate based on 50 samples will be about 1% higher than an estimate based on 5 trillion samples.

What if $p(\theta \mid x)$ is intractable?

Rejection sampling and importance sampling

Rejection sampling

Find a tractable distribution q such that
$c \cdot q(\theta) \geq p(\theta)$
for some known value of c.

While true:
  Sample $\theta \sim q$.
  Sample $u \sim \text{Uniform}(0, c \cdot q(\theta))$.
  If $u < p(\theta)$:
    break.

Intuition: We're sampling uniformly on the graph of the "envelope" distribution q, and only keeping the points on the graph of the target distribution p.

This works great in low dimensions!

But it's hopeless in high dimensions—the volume of the envelope is exponentially larger.

Aside: The Curse of Dimensionality

The volume of a D-dimensional hypercube with sides of length 1 is 1.

The volume of a D-dimensional ball with diameter 1 is

$$\frac{\pi^{D/2}}{2^D \Gamma(1 + D/2)} \approx \frac{0.886^{-D}}{(D/2)!} \text{ for even } D$$

which decays faster than exponentially with D.

So, if we try to rejection-sample from a uniform distribution on the D-ball using a uniform distribution on a hypercube as a proposal, the number of proposals we'll need to make grows faster than exponentially with D.

Intuitions from low dimensions can be misleading when applied to high-dimensional problems!

Rule of thumb: in high dimensions, assume everything is far from everything else.

Importance sampling

Choose a tractable approximating distribution q.

Observe that

$$\mathbb{E}_p[f(\theta)] = \int_\theta p(\theta \mid x) f(\theta) d\theta = \int_\theta p(\theta \mid x) \frac{q(\theta)}{q(\theta)} f(\theta) d\theta = \int_\theta q(\theta) \frac{p(\theta \mid x)}{q(\theta)} f(\theta) d\theta = \mathbb{E}_q \left[\frac{p(\theta \mid x)}{q(\theta)} f(\theta) \right]$$

So we can draw M samples $\theta_{1:M}$ from q, and weight them according to p/q:

$$w_m = \frac{p(\theta_m \mid x)}{q(\theta)}; \qquad \hat{f} = \frac{1}{M} \sum_m w_m f(\theta_m).$$

Intuition: If we're undersampling some region, pay extra attention to samples from that region; if we're oversampling, pay less attention.

Importance sampling with self-normalization

What if we can only compute p(θ, x), not p(θ | x) = p(θ, x) / p(x)?

We can use *self-normalized* importance sampling:

$$\tilde{w}_m = \frac{p(\theta_m, x)}{q(\theta)}; w_m = \frac{\tilde{w}_m}{\sum_i \tilde{w}_i}$$

Why does this work?

$$p(x) = \int_\theta p(\theta, x) d\theta = \int_\theta q(\theta) \frac{p(\theta, x)}{q(\theta)} d\theta = \mathbb{E}_q \left[\frac{p(\theta, x)}{q(\theta)} \right]$$

The unnormalized importance weights are estimators of the normalizing constant (a.k.a. marginal likelihood) p(x).

This procedure is biased, but asymptotically the squared bias decreases as $1/M^2$, and the variance decreases as $1/M$, so variance dominates pretty quickly.

Variance of importance sampling

Importance-sampling estimators can have extremely high variance if there are a lot of θs for which p(θ)/q(θ) is very large. The variance of the weights is

$$\mathbb{E}_q[(w - \mathbb{E}_q[w])^2] = \mathbb{E}_q[w^2] - \mathbb{E}_q[w]^2$$
$$= \mathbb{E}_q\left[\frac{p(\theta)^2}{q(\theta)^2}\right] - \mathbb{E}_q\left[\frac{p(\theta)}{q(\theta)}\right]^2$$
$$= \int_\theta q(\theta)\frac{p(\theta)^2}{q(\theta)^2}d\theta - \left(\int_\theta q(\theta)\frac{p(\theta)}{q(\theta)}d\theta\right)^2$$
$$= \int_\theta p(\theta)\frac{p(\theta)}{q(\theta)}d\theta - \left(\int_\theta p(\theta)d\theta\right)^2$$
$$= \mathbb{E}_p\left[\frac{p(\theta)}{q(\theta)}\right] - 1$$

If q(θ) is decaying much faster than p(θ), then the variance can actually be *infinite*!

In practice, this means that even if you draw many samples, your weights will concentrate on just a few of them.

Variance of importance sampling in high dimensions

In order to keep the variance finite, we need to keep the *tails* of the proposal distribution q above the tails of the target distribution p.

Analogy with rejection sampling:
We need q to *approximately* envelope p.

In high dimensions, this tends to mean putting almost all of q's mass too far out.

So we're very unlikely to sample a state with high density under p, and when we do, p/q will be enormous.

So in high dimensions, the variance of importance sampling procedures is

- acceptable, if we're *really* smart about choosing p, or
- enormous, if we use heavy-enough tails but q isn't close enough to p, or
- infinite, if we get the tails wrong.

One class of smart IS methods is *sequential Monte Carlo*. Some SMC methods even incorporate MCMC as proposals!

Aside: Importance sampling and variational inference

The unnormalized importance-weight formula is closely related to the ELBO used in variational inference.
Here's the single-sample importance-sampling estimator of the marginal likelihood again:

$$p(x) = \int_\theta p(\theta, x) d\theta = \int_\theta q(\theta) \frac{p(\theta, x)}{q(\theta)} d\theta = \mathbb{E}_q \left[\frac{p(\theta, x)}{q(\theta)} \right]$$

And here's the ELBO:

$$\log p(x) = \log \int_\theta p(x, \theta) d\theta = \log \int_\theta q(\theta) \frac{p(x, \theta)}{q(\theta)} d\theta = \log \mathbb{E}_q \left[\frac{p(x, \theta)}{q(\theta)} \right] \geq \mathbb{E}_q \left[\log \frac{p(x, \theta)}{q(\theta)} \right]$$

This "p over q" theme also showed up in rejection sampling. We'll see it again in the Metropolis-Hastings algorithm.

Multiplying by 1 can be powerful!

Markov chains

Markov chains: definition

A Markov chain is a random process that generates a sequence of variables $\Theta_{1,\ldots,N}$.

The “Markovian” property is that each Θ_{n+1} only depends on the previous state Θ_n.

We’ll denote the probability of Θ_{n+1} given Θ_n as $T(\Theta_{n+1} \mid \Theta_n)$. T is sometimes called the “transition kernel”.

A simple example:

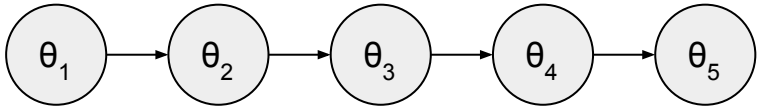
$$\theta_{n+1} = \theta_n + \xi_n; \quad \xi_n \in \{-1, 1\}$$

A slightly fancier example:

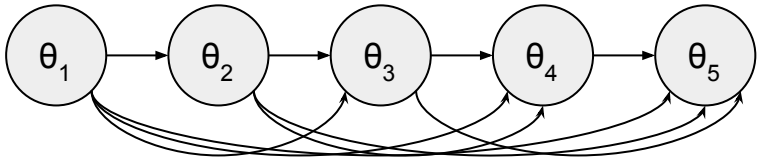
$$\theta_{n+1} = 0.9\theta_n + \sqrt{1 - 0.9^2}\xi_n; \quad \xi_n \sim \text{Normal}(0, 1)$$

Markov chains: Graphical model

The graphical model for a Markov chain looks like...a chain.



More general autoregressive processes (e.g., pixelcnn, wavenet, transformers) might look back further when deciding what to sample next:

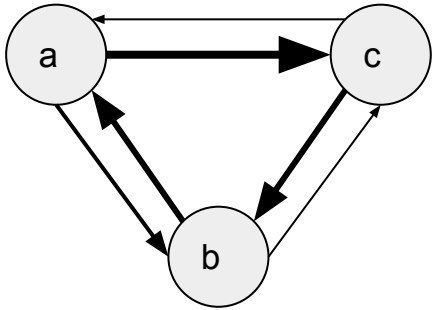


The Markovian restriction makes Markov chains much easier to analyze.

Aside: In principle, one can always make a non-Markovian process Markovian by making the state space bigger. This had better be true, since physics is Markovian!

Markov chains: Finite state-machine picture

Confusingly, people sometimes use finite-state-machine type graphs to describe Markov chains on discrete spaces:



Here the nodes are states of the Markov chain, and the size of the arrows represents the transition probability from one state to another.

(We'll usually be interested in Markov chains over state spaces with much more than three states!)

Stationary distributions

We're going to be very interested in the *stationary distributions* of Markov chains.

The stationary distribution of a Markov chain describes the distribution of its nth state θ_n as $n \to \infty$.

Not all Markov chains have stationary distributions! For example:

- The transition kernel could change with time.
- A chain with continuous states could diverge off towards infinity.
- A chain could be periodic, e.g., with θ_n being even if n is even.
- A chain might not be ergodic, i.e., it could have two disconnected sets of states that can't communicate.

The only one of these that often comes up in MCMC practice is ergodicity.

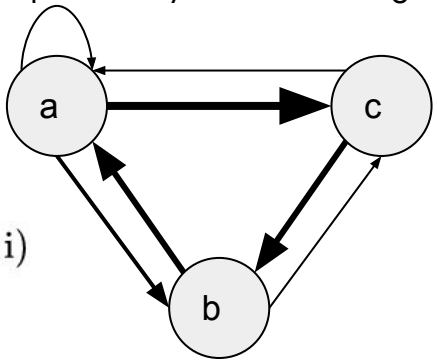
Markov chains: Matrix representation

Markov chains over discrete spaces can be represented by a *transition matrix* T, where T_{ij} is the probability of transitioning from state i to state j.

If π is a vector such that π_i is the probability of being in state i at some time n, then
$\pi' = T\pi$
gives the probability of being in any state at time n+1:

$$\pi'_i = \sum_j T_{ij}\pi_j = \sum_j (\text{probability of being in state j})(\text{probability of going from j to i})$$

If we start from some initial distribution π, then at time 3, the marginal distribution will be $T(T(T\,\pi)) = T^3\pi$.
More generally, at time n the marginal distribution will be $T^n\pi$.



	a	b	c
a	0.1	0.2	0.7
b	0.8	0	0.2
c	0.1	0.9	0

Markov chains: Matrix representation

At time n the marginal distribution will be $T^n\pi$.

Recall that raising T to a power raises its eigenvalues to that power:

$$T = \sum_k \lambda_k u_k u_k^\top; \quad T^n = \sum_k \lambda_k^n u_k u_k^\top.$$

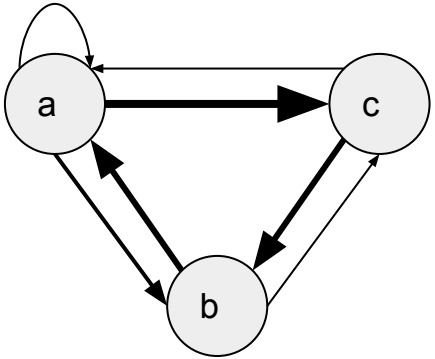
The largest-magnitude eigenvalue of T is 1:

- If it were bigger, we'd wind up with probabilities greater than 1.
- If it were smaller, we'd wind up with probabilities that don't sum to 1.

If this eigenvalue is unique (i.e., the chain is ergodic), then all other eigenvalues decay to 0 exponentially fast in n.

So as n goes to ∞, $T^n\pi = u_{max}$, the eigenvector associated with the largest eigenvalue.
This is the stationary distribution for the chain.

The magnitude of the second-largest eigenvalue λ_2 tells us how quickly the chain is "mixing".
The influence of π on the state at time n is $O(|\lambda_2|^n)$.
If $\lambda_2 = 0.5$, then after 50 iterations the bias due to π is effectively 0.
If $\lambda_2 = 0.999$, then after 500 iterations the bias is only reduced by about 40%.



	a	b	c
a	0.1	0.2	0.7
b	0.8	0	0.2
c	0.1	0.9	0

Markov chain Monte Carlo

MCMC: The big idea

We're going to set up a Markov chain whose stationary distribution is $p(\theta \mid x)$.

We're going to simulate it long enough that we're confident it's converged to that stationary distribution.
That is, we want the process to "forget" what distribution it started from.

We can do this lots of times in parallel, or we can do a smaller number of longer runs.
The samples in long runs will be correlated, but that only adds variance, not bias. (More on this later.)

MCMC: How?

We want to set up a Markov chain whose stationary distribution is p(θ | x), that is,

$$\int_\theta p(\theta \mid x) T(\theta' \mid \theta) d\theta = p(\theta' \mid x)$$

How can we do this? It turns out there's a stronger condition called *detailed balance* that will be helpful:

$$p(\theta \mid x) T(\theta' \mid \theta) = p(\theta' \mid x) T(\theta \mid \theta')$$

Intuitively, the idea is that the mass going from θ to θ' is the same as the mass going from θ' to θ.
Note that there's no impossible integral to solve here!

Detailed balance implies that p(θ | x) is the stationary distribution of T:

$$\int_\theta p(\theta \mid x) T(\theta' \mid \theta) d\theta = \int_\theta p(\theta' \mid x) T(\theta \mid \theta') d\theta = p(\theta' \mid x) \int_\theta T(\theta \mid \theta') d\theta = p(\theta' \mid x)$$

So if we can construct a kernel T that satisfies detailed balance w.r.t. p(θ | x), then we'll know it has the right stationary distribution.

Aside: On discrete state spaces, detailed balance implies a symmetric transition matrix.

The Metropolis algorithm

A top-10 algorithm of the 20th century (up there with the FFT, QR decomposition, simplex method, FORTRAN compiler, etc.).

Given current state θ_n, sample a proposed θ' from a symmetric proposal distribution $q(\theta' \mid \theta) = q(\theta \mid \theta')$.
Accept $\theta_{n+1} = \theta'$ with probability $\min\{1, p(\theta') / p(\theta)\}$.
Otherwise, $\theta_{n+1} = \theta_n$.

For example, we can propose $q(\theta' \mid \theta) = \text{Normal}(\theta'; \theta, \sigma) = \text{Normal}(\theta; \theta', \sigma)$:

The Metropolis algorithm for Bayesian inference

Typically we can't compute $p(\theta \mid x) = p(x, \theta) / p(x) = p(x \mid \theta)\, p(\theta) / p(x)$, because we don't know $p(x)$.

But we can compute the *ratio* $p(\theta' \mid x) / p(\theta \mid x) = p(x, \theta') / p(x, \theta)$!

To do MCMC, we don't need to know how likely θ' is—we just need to know how likely it is *relative* to some other θ.

This is a small miracle.

The Metropolis algorithm

Why does Metropolis work? Let's plug the transition kernel into the detailed balance formula
p(θ | x) T(θ' | θ) = p(θ' | x) T(θ | θ')
and see if it works.

First, the case where we reject, so θ' = θ:
p(θ | x) T(θ | θ) = p(θ | x) T(θ | θ).
That was easy!

Now the harder case, where we accept.
The transition probability from θ to θ' is the probability of proposing θ' times the probability of accepting it:

$$
\begin{aligned}
p(\theta \mid x) T(\theta' \mid \theta) &= p(\theta \mid x) q(\theta' \mid \theta) \min\left\{1, \frac{p(\theta' \mid x)}{p(\theta \mid x)}\right\} \\
&= q(\theta \mid \theta') \min\left\{p(\theta \mid x), p(\theta' \mid x)\right\} \\
&= p(\theta' \mid x) q(\theta \mid \theta') \min\left\{1, \frac{p(\theta \mid x)}{p(\theta' \mid x)}\right\} \\
&= p(\theta' \mid x) T(\theta \mid \theta')
\end{aligned}
$$

The symmetry of the min function is doing a lot of work here.

The Metropolis-Hastings algorithm

What if we want to use an asymmetric proposal distribution $q(\theta' \mid \theta) \neq q(\theta \mid \theta')$?

Adjust the acceptance probability to $p(\theta' \mid x)\, q(\theta \mid \theta') \,/\, p(\theta \mid x)\, q(\theta' \mid \theta)$.

The case where we reject is the same as before:
$p(\theta \mid x)\, T(\theta \mid \theta) = p(\theta \mid x)\, T(\theta \mid \theta)$.

Now, the case where we accept:

$$
\begin{aligned}
p(\theta \mid x) T(\theta' \mid \theta) &= p(\theta \mid x) q(\theta' \mid \theta) \min\left\{1, \frac{p(\theta' \mid x) q(\theta \mid \theta')}{p(\theta \mid x) q(\theta' \mid \theta)}\right\} \\
&= \min\left\{p(\theta \mid x) q(\theta' \mid \theta), p(\theta' \mid x) q(\theta \mid \theta')\right\} \\
&= p(\theta' \mid x) q(\theta \mid \theta') \min\left\{1, \frac{p(\theta \mid x) q(\theta' \mid \theta)}{p(\theta' \mid x) q(\theta \mid \theta')}\right\} \\
&= p(\theta' \mid x) T(\theta \mid \theta')
\end{aligned}
$$

Again, the symmetry of the min function does all the work.

A few Metropolis-Hastings algorithms

The ideal Metropolis-Hastings algorithm

Directly sample from the target:
$q(\theta' \mid \theta) = p(\theta' \mid x)$.

This has an acceptance probability of 1:
$p(\theta' \mid x)\, q(\theta \mid \theta') \,/\, p(\theta \mid x)\, q(\theta' \mid \theta)$
$= p(\theta' \mid x)\, p(\theta \mid x) \,/\, p(\theta \mid x)\, p(\theta' \mid x)$
$= 1$.

Nice work if you can get it.

Independence M-H

We can use other proposal distributions that don't depend on the current state, i.e., q(θ' | θ) = q(θ').

Plugging this into the M-H acceptance probability gives

$$\min\left\{1, \frac{p(\theta')q(\theta)}{p(\theta)q(\theta')}\right\} = \min\left\{1, \frac{p(\theta')}{q(\theta')}\frac{q(\theta)}{p(\theta)}\right\}$$

If we're in a state θ where p(θ)/q(θ) is large, then we'll tend to reject.

This is like importance sampling, except instead of upweighting samples using explicit weights, we're giving states with large p/q more weight in *time*.

Composing kernels: Gibbs sampling

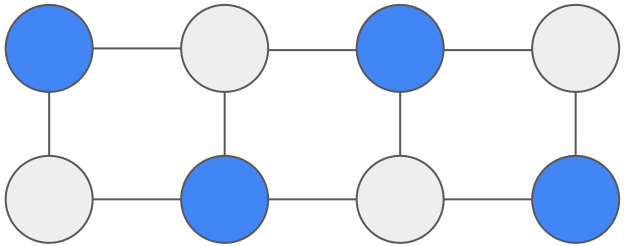
We can apply any kernels with the same stationary distribution in any order we like (including at random), as long as we don't choose which to apply based on the current state θ.

One reason to do this is if we want to use kernels that only update some elements of θ.

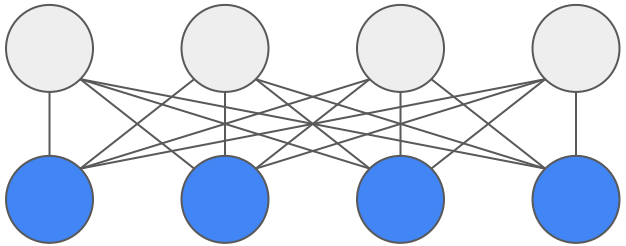
For example, we can sometimes resample each element θ_i from its conditional distribution $p(\theta_i \mid \theta_{\setminus i}, x)$.
This is called *Gibbs sampling*. It can be seen as a Metropolis-Hastings kernel with acceptance probability 1:

$$\frac{p(\theta', x)q(\theta \mid \theta')}{p(\theta, x)q(\theta' \mid \theta)} = \frac{p(\theta'_i, \theta_{\setminus i}, x)p(\theta_i \mid \theta_{\setminus i}, x)}{p(\theta_i, \theta_{\setminus i}, x)p(\theta'_i \mid \theta_{\setminus i}, x)} = \frac{p(\theta_{\setminus i}, x)}{p(\theta_{\setminus i}, x)} = 1$$

Gibbs samplers can exploit conditional independence structure in various ways, for example, *block* Gibbs samplers update conditionally independent nodes in parallel:



Ising model



Restricted Boltzmann Machine

Part II

Outline

In Part I, the focus was on MCMC basics and theory.

In Part II, the focus will be more on practical considerations, with a focus on gradient-based methods:

- The workflow: warmup, adaptation, and sampling
- Diagnostics
- Hamiltonian Monte Carlo
- MCMC and VI: better vs faster?

What do we want from an MCMC algorithm?

We want it to yield estimators with:

- Low bias,
- Low variance, and
- Low cost.

More iterations => less bias and variance, more cost.

Extreme cases:

- Run a valid (ergodic etc.) chain forever. (Unbiased! Zero variance!)
- Just run for one step. (Cheap!)
- $T(\theta' \mid \theta) = \delta(\theta' - \theta)$, i.e., $\theta_{t+1} = \theta_t$. (Simple, cheap, and reversible!)

The last is an extreme example of a chain that moves around slowly.
Intuitively, moving slowly seems bad—more on this later.

How to MCMC

A typical MCMC run consists of three phases:

- Setup:
  - build a model
  - choose an algorithm
  - choose an initial state θ_0 and initial parameters φ_0
- Warmup a.k.a. burn-in:
  Run the sampler for M_{warmup} steps, possibly adapting φ.
- Sampling:
  Run the sampler for M_{sampling} steps, holding φ fixed.

Finally, we discard the sampler output from the warmup phase,
and compute an estimate by averaging samples from the sampling phase.

$$\hat{f} = \frac{1}{M_{\text{sampling}}} \sum_{i=M_{\text{warmup}}}^{M_{\text{warmup}}+M_{\text{sampling}}} f(\theta_i)$$

Warmup: Why?

We initialize our chain with garbage.

This introduces *transient bias*:
Our estimator is influenced by the initial state, but this bias can be eliminated by running for longer.

Fortunately, the amount of information about the initial state decays exponentially fast (if we design the chain correctly).

Adaptation

Most MCMC algorithms have tunable parameters (Gibbs sampling is an exception).

For example, in random-walk Metropolis (RWM), we might propose $q(\theta' \mid \theta) = \text{Normal}(\theta'; \theta, \sigma)$.

If σ is too big, we'll reject a lot.
If σ is too small, we'll move very slowly.

Step-size adaptation

For step-size M-H parameters, we usually try to find a high-but-not-too-high average acceptance probability. (E.g., 0.234, 0.651, 0.85; details vary.)

A simple proportional-control scheme for tuning a step size to achieve a target mean acceptance probability:

```
step_size *= np.exp(adapt_speed * (accept_prob - target))
```

We have a new adaptation-speed parameter, but this is much less problem-dependent.

Why only adapt during warmup?

While we're adapting some parameters φ, φ becomes part of the state of the chain.

Just because our kernel satisfies
$p(\theta)\, T_\varphi(\theta' \mid \theta) = p(\theta')\, T_\varphi(\theta \mid \theta')$

for all φ, that doesn't imply that there exists a $p(\varphi \mid \theta)$ such that
$p(\theta)\, p(\varphi \mid \theta)\, T(\varphi' \mid \varphi, \theta)\, T_\varphi(\theta' \mid \theta) = p(\theta')\, p(\varphi' \mid \theta')\, T(\varphi \mid \varphi', \theta')\, T_{\varphi'}(\theta \mid \theta')$.
So we have no guarantee that we're leaving $p(\theta)$ invariant; in general, we aren't.

Often people freeze adaptation shortly before the end of warmup, to give the chain time to forget the bias due to adaptation.

Parallel chains

In general, Monte Carlo methods are embarrassingly parallel—we can always generate more samples to reduce variance.

In MCMC, there are several reasons to run multiple chains in parallel and average the results:

- Variance reduction. (Hooray for accelerators!)
- We can average adaptation signals across multiple chains. (Less noise, less bias.)
- If our sampler can't mix between certain regions, running multiple chains gives us better (but still biased) estimators.
- Many convergence diagnostics require multiple chains.

However, each chain usually runs its own warmup, which may waste FLOPs relative to a single long chain.

Parallel chains for adaptation

There are two main reasons to share tunable parameters φ across chains:

- The averaged adaptation signal is less noisy, so adaptation is more stable.
- The contribution of any one chain to the adaptation is smaller, so the process introduces less bias.

In the extreme case where we have infinitely many chains, φ_t is a deterministic function of t, and is therefore independent of θ, so adaptation introduces no bias.

However, sharing the same φ across chains can be dangerous:

- If there are disconnected regions, we may want different φ's for each region.
- Sometimes a chain can get "stuck", and some adaptation procedures won't notice.

Diagnostics

“R-hat” (Gelman and Rubin, 1992; Brooks and Gelman, 1998)

Intuition: If there are two or more chains that give very different estimates, we can’t be confident that they’ve converged.

R-hat compares the variance estimates each individual chain to the variance estimate obtained by pooling the chains.

If the estimated ratio R-hat of across-chain variance to average within-chain variance is much larger than 1, we should worry.

Effective sample size

How many independent samples is this chain “worth”?
What’s that even mean?

Effective sample size

We define *effective sample size* in terms of the variance of our estimator f-hat:

$$\mathrm{ESS} \triangleq \frac{\mathrm{Var}_{p(\theta)}(f)}{\mathrm{Var}(\hat{f})}$$

If we had M independent samples, then the variance of f-hat would be $\mathrm{Var}_{p(\theta)}(f) / M$,

so the “effective” sample size is how many independent samples from p we’d need to get an estimate of f that’s as good as the one our MCMC estimator gives us.

Note that ESS is defined in terms of a particular function f(θ)—it is not just a property of the chain!

Effective sample size and autocorrelation

A bit of algebra reveals that, assuming stationarity,

$$\mathrm{ESS} \triangleq \frac{\mathrm{Var}_{p(\theta)}(\theta)}{\mathrm{Var}(\hat{\theta})} = \frac{\mathrm{Var}_{p(\theta)}(\theta)}{\mathrm{Var}(\frac{1}{M}\sum_m \theta_m)} = \frac{M}{1 + 2\sum_{m=1}^{M} \frac{M-m}{M}\alpha_m} \approx \frac{M}{1 + 2\sum_{m=1}^{M} \alpha_m}$$

where α_m is the lag-m autocorrelation:

$$\alpha_m \triangleq \frac{\mathbb{E}[(\theta_t - \mathbb{E}[\theta])(\theta_{t+m} - \mathbb{E}[\theta])]}{\mathrm{Var}(\theta)}$$

This formalizes the notion that we don't want our chain to move slowly—if samples are strongly autocorrelated, the variance of our estimator goes up.

Effective sample size

How many independent samples is this chain “worth”?

~14.6 if we’re estimating $E[\theta]$.
~24.4 if we’re estimating $E[\theta^2]$.

Note that these are estimates obtained by plugging in noisy estimates of autocorrelations!

Hamiltonian Monte Carlo
(HMC, a.k.a. Hybrid Monte Carlo)

The trouble with random walks

Consider a Gaussian random walk
$\theta_{t+1} = \theta_t + \xi_t$; $\xi_t \sim \text{Normal}(0, \sigma)$.
After T steps, we've moved
$\theta_T - \theta_0 = \Sigma_t \xi_t$.
The sum of T Gaussian random variables each with variance σ^2 is a Gaussian random variable with variance $T\sigma^2$.

So after T steps, we expect to move a distance of O(sqrt(T)).

Random walks with small steps are slow!

The trouble with random walks

Reversible MCMC kernels are as likely to move forward as backward.

If they don’t move very far each iteration, then they’ll explore the space by a random walk.

If they need to use a small step size to avoid rejecting, then they’ll explore the space by a very slow random walk.

In high dimensions, naive proposals need to use a very small step size.

Random walks in high dimensions

Consider a spherical Gaussian in D dimensions: $\theta \sim \text{Normal}(0, I)$.

The norm of θ is a chi random variable with D degrees of freedom.

Even though the density is highest around the origin, as D gets large, *all of the mass is concentrated in a thin shell*. This shell is sometimes called the “typical set”.

High dimensions are weird! All of the volume is very far from 0.

Random walks in high dimensions

Now, suppose we make a Metropolis proposal $q(\theta' \mid \theta) = \text{Normal}(\theta'; \theta, \sigma^2 I)$.

If we're at stationarity, the marginal distribution of θ' will be $\text{Normal}(0, (1 + \sigma^2)I)$, and its norm will be a scaled chi distribution.

Unless we use a very small step size σ, there will be no overlap in the typical sets of the target and proposal distributions in high dimensions.

This leads to $O(D^2)$ scaling for random-walk Metropolis in the *best* case.

Hamiltonian Monte Carlo (HMC)

HMC is an MCMC method that lets us sample from high-dimensional distributions without getting hamstrung by random-walk behavior.

It uses a very clever physical analogy (and gradient information) to make far-ranging proposals that nonetheless stay within the typical set.

HMC: The big idea

Imagine that θ is a position in R^D.

Augment our model with a “momentum” vector m, also in R^D, so that $p(\theta, m) = p(\theta)\ \text{Normal}(m; 0, I)$.

Reinterpret log $p(\theta)$ as a negative potential energy function, and the normal log-density for m as a negative quadratic kinetic energy function.

Then -log $p(\theta, m)$ can be interpreted as a *Hamiltonian*—the energy of a fictitious physical system.

HMC: The big idea

-log p(θ, m) can be interpreted as a *Hamiltonian*—the energy of a fictitious physical system.

The dynamics of this system are given by Hamilton's equations:
$\partial\theta/\partial t = -\partial \log p(m) / \partial m = m$,
$\partial m/\partial t = \partial \log p(\theta) / \partial \theta$.

Hamiltonian dynamics have three very nice properties:

- They conserve energy:
  $\partial H/\partial t = \partial H/\partial \theta \, \partial\theta/\partial t + \partial H/\partial m \, \partial m/\partial t = -\partial H/\partial \theta \, \partial H/\partial m^T + \partial H/\partial m \, \partial H/\partial \theta^T = 0$.
- They conserve phase-space volume: If $z = (\theta, m)$, $|\partial z/\partial t| = 1$.
- They are reversible: negating m reverses time.

HMC: The ideal version

Hamiltonian dynamics have three very nice properties:

- They conserve energy:
  $\partial H/\partial t = \partial H/\partial \theta \, \partial \theta/\partial t + \partial H/\partial m \, \partial m/\partial t = -\partial H/\partial \theta \, \partial H/\partial m^T + \partial H/\partial m \, \partial H/\partial \theta^T = 0$.
- They conserve phase-space volume: If $z = (\theta, m)$, $|\partial z/\partial t| = 1$.
- They are reversible: negating m reverses time.

This suggests the following ideal algorithm:

1. Sample new momenta $m \sim \text{Normal}(0, I)$.
2. Simulate the Hamiltonian dynamics of θ, m for some time t to get θ', m';
   Set our new state z' to $(\theta', -m')$.

Step 1 is clearly a valid Gibbs update.

Step 2 is a valid Metropolis update, since

- Applying it twice takes us back to θ, m. ($q(z' \mid z) = q(z \mid z')$.)
- $\log p(z') = -H(\theta', -m') = -H(\theta', m') = -H(\theta, m) = \log p(z)$. ($p(z') = p(z)$.)
- There's no volume change that would invoke measure-theoretic considerations.

HMC on a computer

Computers run in discrete time, so we can't apply the ideal algorithm.

Solution: Use the "leapfrog" (Störmer-Verlet) integrator:

$$m_{1/2} = m_0 + \frac{\epsilon}{2}\nabla \log p(\theta_0)$$
$$\theta_1 = \theta_0 + \epsilon m$$
$$m_1 = m_{1/2} + \frac{\epsilon}{2}\nabla \log p(\theta_1)$$

The leapfrog integrator is reversible and volume-preserving, but does not perfectly conserve energy, so we need to apply a Metropolis correction and accept the proposed update with probability

$$\alpha = \min\left\{1, \frac{p(\theta')\mathcal{N}(m';0,I)}{p(\theta)\mathcal{N}(m;0,I)}\right\}$$

Using a small step size will make the dynamics close to the true energy-conserving Hamiltonian dynamics, and will keep α close to 1.

HMC in practice

HMC works great if you tune the step size ε and number of steps L of the leapfrog integrator (and randomize L to avoid resonances).

We can tune the step size in the usual way, by targeting a high-but-not-too-high acceptance rate.

What about L? Optimal values depend on the problem, and range
from 1 to 70,000.