Hamiltonian Monte Carlo

ISTA 410 / INFO 510: Bayesian Modeling and Inference

U. of Arizona School of Information

September 29, 2021

Recap: Metropolis (-Hastings)

Discrete example

A simple algorithm:

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

and count this as a sample.

Metropolis algorithm in general

More generally: Metropolis algorithm.

1. Start with a (possibly un-normalized) density function f on the sample space – this is our target distribution.
2. Establish a proposal distribution $\underline{q(x, y)}$ that is a *symmetric* transition probability on the state space, meaning $q(x, y) = q(y, x)$.

   eg, Gaussian
3. At time t, when $X_t = x$, sample a value y from $q(x, y)$ – propose a transition $x \to y$
4. Compute the acceptance probability

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

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

Why does this work?

Detailed balance equation for a stationary distribution:

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

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.)

Transition probability for the Metropolis algorithm:

$$p(x,y) = \underline{q(x,y)}\frac{f(y)}{\underline{f(x)}}$$

Plug into the detailed balance equation, substituting $f(x)$ for $\underline{\pi(x)}$:

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

Assume $f(x) > f(y)$: left hand side is $\underline{q(x,y)f(y)}$, right hand side is $\underline{q(y,x)f(y)}$. If q is symmetric, then we're good!

In code

Simple option: Gaussian proposal. Proposed y is $x + N(0, \sigma)$. Depends only on distance between x and y, so definitely symmetric.

initial state density function

```python
def metro_step(x, p, sigma):
    proposal = x + sp.stats.norm(0, sigma).rvs()
    alpha = min(1, p(proposal) / p(x))
    if np.random.rand() <= alpha:
        new_x = prop
    else:
        new_x = x
```

Metropolis algorithm in general

More generally: 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

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

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 $q(x, y) \neq q(y, x)$. Exercise: demonstrate that this works.

General difficulties

Two general issues with MCMC:

- Burn-in: the chain takes time to approach the target distribution
- Autocorrelation: we are not sampling independent draws from the target distribution
- “Solutions:”
    - Always use multiple chains so we can compare them to help assess convergence to the target distribution
    - Use proposals that efficiently explore the sample space

Difficulty

The practical difficulty of Metropolis-Hastings comes in trying to choose a good proposal distribution.

Generally we have a tradeoff:

- Highly variable proposal distribution: better exploration of the sample space (less time to reach the target distribution) but many rejected proposals
- Tight proposal distribution: more accepted proposals, but ”random walk” behavior (high autocorrelation)

This can be especially bad depending on the form of the posterior. Let’s see some examples in action.

https://chi-feng.github.io/mcmc-demo

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 $(\theta'_1, \theta'_2, \ldots, \theta'_d)$

Hamiltonian Monte Carlo

A better way to get proposals

More modern update: Hamiltonian Monte Carlo (aka Hybrid Monte Carlo).

- Hybrid because it mixes a deterministic simulation with the random nature of MCMC
- Hamiltonian because the simulation uses Hamilton's equations from physics
- Essentially a way to use the properties of the target distribution to generate proposals

There are some older "adaptive" methods, but they are mostly limited to, e.g., Gaussian proposals with correlations

HMC: Borrowing an idea from physics

Hamiltonian mechanics:

- Define space coordinates and momentum coordinates q_i, p_i to describe the state of a particle
- Summarize the momentum and potential energy by a function $H(q, p)$
- System evolves according to the equations:

$$\left.\begin{aligned}\frac{\partial q_i}{\partial t} &= \frac{\partial H}{\partial p_i}\\ \frac{\partial p_i}{\partial t} &= -\frac{\partial H}{\partial q_i}\end{aligned}\right\}$$

HMC details

How HMC works:

1. Think of $-\log p(\theta|y)$, the (negative) log posterior density, as a potential energy
2. Pick a starting point in parameter space, and place a particle there with fixed mass
3. Randomly select a momentum vector, and give the particle that momentum
4. Simulate the motion of the particle for a fixed time $\Delta t \times L$
5. Use the endpoint as a proposal for a Metropolis step

HMC summary

- HMC explores parameter space efficiently because its proposals follow the geometry of the (log) posterior
- Extra variables, particularly energy, provide diagnostics
    - Hamiltonian mechanics conserves energy, so if energy changes something is wrong

HMC summary

- HMC explores parameter space efficiently because its proposals follow the geometry of the (log) posterior
- Extra variables, particularly energy, provide diagnostics
    - Hamiltonian mechanics conserves energy, so if energy changes something is wrong
- Requires computation of more stuff:
    - Need gradients of the log posterior, which are model-specific
    - Need to run the simulation
    - Extra parameters: momentum, mass
- Extra tuning: physics step size, number of steps

Tuning HMC

Two reasons you need to tune these:

1. Step size determines accuracy of the physics simulation. Small step size means more accurate physics, proposals accepted more often. But it means you need more steps to get the same amount of mixing, so more computation per sample
2. The U-turn problem
   - If the simulation runs too long, the particle can cross a local minimum, turn around, and come back
   - Leads to inefficient exploration of the sample space

This can be avoided by tuning the number of steps; the leading implementation of HMC, called NUTS (No U-Turn Sampler), begins sampling with some "tuning" samples and runs some simple ML to tune parameters

Evaluating performance and diagnosing problems

Evaluating performance from MCMC

Three targets for a good Markov chain:

- Stationarity: path of each chain should stay within the same high-probability portion of the posterior
- Mixing: each chain should rapidly explore the full region (this is what we missed with our toy Metropolis implementation)
- Convergence: the chains should be sampling from the correct posterior distribution
    - Can't compare to the true posterior – instead, sample several chains and compare to one another

Inspecting the health of your chains

- Library for visualization and summarization of MCMC results: ArviZ
- Trace plots:
  - `az.plot_trace`
  - Plot the value of the sample vs. the time step in the chain
- What to look for in a trace plot:
  - "fuzzy caterpillars" – rough, noisy plots
  - different chains show similar sample densities

Inspecting the health of your chains

§9.4, 9.5

- Trace rank plots (McElreath calls these trank plots)
- Get all samples and rank them lowest to highest
- Plot a histogram of the ranks from each chain
- If all chains are exploring the same space efficiently, the histograms should be similar and largely overlapping

An example of sampling gone wrong

Now let's see what these diagnostics look like when our sampling is bad.

Example:

- Try to estimate the mean of a normal random variable (suspected to be near 0)
- Have two observations: $\{-1, 0, 1\}$
- Model:

$$
\begin{aligned}
y &\sim \mathrm{Normal}(\mu, \sigma) \\
\mu &\sim \mathrm{Normal}(0, 1000) \\
\sigma &\sim \mathrm{Exponential}(1/1000)
\end{aligned}
$$

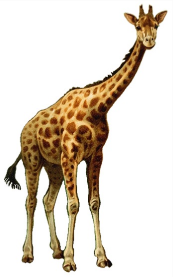
Let's try this.

Another bad model: non-identifiability

Another source of modeling problems: non- or weak identifiability

Modeling giraffe height stratified by sex

$$
\begin{aligned}
h &\sim \mathrm{Normal}(\mu, \sigma) \\
\mu &= \alpha + \beta_M M + \beta_F F \\
\alpha &\sim \mathrm{Normal}(0, 100) \\
\beta_M &\sim \mathrm{Normal}(0, 100) \\
\beta_F &\sim \mathrm{Normal}(0, 100)
\end{aligned}
$$



Can be (kind of) saved by weakly informative priors

The folk theorem

The ”folk theorem of statistical computing:”

> *[Gelman] When you have computation problems, often there’s a problem with your model.*

- Check simple things first. Things I have done that have led to computational problems:
  - Forgetting to set the shape of a parameter vector (sometimes this will throw an error, but sometimes it won’t)
  - Forgetting to link up two parameters
- Set reasonable weakly regularizing priors

Diagnostic statistics

Some other diagnostic statistics that you can use:

- $\hat{R}$ (a.k.a. the Gelman-Rubin statistic), measures the ratio of the estimated variance of the parameter, pooling several chains, to the variance within a chain. Should be nearly 1 if all chains have converged to the same distribution. You'll get warnings if $\hat{R}$ is too high for any parameter.
    - Goal: assess convergence by comparing parallel chains to one another
    - We did this visually with the trace rank plot
- Effective sample size: estimate of the equivalent number of samples *without* autocorrelation. You'll get warnings if this is really low (as either an absolute number or a fraction of samples).

$\hat{R}$ and ESS are both calculated by `az.summary`; let's take a look.

Summary

Further reading for practical advice on using MCMC methods:

- Rethinking chapter 9. 9.4, 9.5 have descriptions of diagnostics.
- General notes on using HMC in PyMC3. Lots of good small tips and tricks here. `https://eigenfoo.xyz/_posts/2018-06-19-bayesian-modelling-cookbook/`
- Notes on choosing priors. Written by Stan users, but most of it is really language-agnostic (because it's more about the modeling side). `https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations`

Next week:

- Information criteria for model evaluation