More Monte Carlo Methods

ISTA 410 / INFO 510: Bayesian Modeling and Inference

U. of Arizona School of Information

September 30, 2020

Hamiltonian Monte Carlo

Recap: Metropolis-Hastings

Last time, we encountered the Metropolis-Hastings algorithm for sampling from a target probability distribution $p(x)$:

1. Choose a proposal distribution $q(x, y)$ that is a transition probability on the sample 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.

Basic Metropolis version: $q(x, y) = q(y, x)$ and
$$\alpha(x, y) = \min\left\{\frac{p(x)}{p(y)}, 1\right\}$$

In code

Simple implementation with Gaussian proposal:

```
def metro_step(x, p, sigma):
    prop = x + sp.stats.norm(0,np.eye * sigma).rvs()
    alpha = min(1, p(prop) / p(x))
    if sp.stats.uniform.rvs() <= alpha:
        new_x = prop
    else:
        new_x = x
    return new_x
```

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
- Tight proposal distribution: more accepted proposals, but "random walk" behavior

This can be especially bad depending on the form of the posterior. Let's look at an example.

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:

$$\frac{\partial q_i}{\partial t} = \frac{\partial H}{\partial p_i}$$
$$\frac{\partial p_i}{\partial t} = -\frac{\partial H}{\partial q_i}$$

Intuitively: think of the potential energy as a frictionless surface, and the particle slides around it by gravity

HMC details

How HMC works:

1. Pick a starting point in parameter space, and place a particle there with fixed mass.
2. Randomly select a momentum vector, and give the particle that momentum
3. Simulate the motion of the particle for a fixed time $\Delta t \times L$
4. 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

The NUTS sampler

The leading implementation of HMC currently is called NUTS[1] (No U-Turn Sampler).

The U-turn problem:

- If the simulation runs too long, the particle can cross a local minimum, turn around, and come back
- Leads to worse mixing

Let's see a really terrible example.

[1] sorry...

The NUTS sampler

The leading implementation of HMC currently is called NUTS[1] (No U-Turn Sampler).

The U-turn problem:

- If the simulation runs too long, the particle can cross a local minimum, turn around, and come back
- Leads to worse mixing

Let's see a really terrible example.

To avoid U-turns, tune the number of physics steps. NUTS runs some simple ML on the warm-up results to automatically tune L. Δt is still a potential tuning parameter.

[1] sorry...

When things go wrong

Recap

Last week, we saw an example of a hierarchical model (hierarchical normal model for 8 schools problem) where the sampler misbehaved.

We saw:

- warnings about divergences
- trace plots
- inconsistent results from different chains

So:

- what do these warnings mean?
- what went wrong?
- how can we fix it?

What is a divergence?

The core sampling step of HMC is a physics simulation:

- The Hamiltonian $H(q, p)$ represents the total energy of the system
- Hamiltonian systems conserve energy

A “divergence” in HMC is when the energy at the start of the simulation doesn’t match the energy at the end. It means something went wrong with the physics.

Divergence: causes and effects

What causes a divergence: numerical problems.

- The physics simulation uses discrete time steps to model a continuous process
- If our time steps are too large, our simulation is too "coarse"
- Time steps need to be smaller when the potential energy (i.e. log posterior) has high curvature.

How to fix divergences

When we get divergences, the NUTS algorithm has some suggestions for us:

```
There were 282 divergences after tuning.
Increase ‘target_accept‘ or reparameterize.
```

So we have two main options: increase `target_accept` or reparameterize

Simple approach: change target acceptance

The simplest way to deal with divergences: increase the target acceptance rate, which decreases the step size in the physics simulation.

- This decreases the efficiency of the sampling, because it requires more physics steps per sample. But it's better than having many divergences.
- Default for `target_accept` is 0.8; try setting to, e.g., 0.9, 0.95

This works for random "false positive" divergences, but sometimes won't help. Divergences that won't go away are a serious problem.

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
- Set reasonable priors
- etc.

Excessive curvature in the 8 schools problem

Where is the problem in the 8 schools model?

Let's first look at some plots.

Excessive curvature in the 8 schools problem

Where is the problem in the 8 schools model?

Let's first look at some plots.

Recall the model:

$$\begin{aligned} y_j &\sim \mathrm{Normal}(\theta_j, \sigma_j) \\ \theta_j &\sim \mathrm{Normal}(\mu, \tau) \\ \mu &\sim \mathrm{Normal}(0, 5) \\ \tau &\sim \mathrm{HalfCauchy}(5) \end{aligned}$$

The source of the problem

Remember the physics “surface” is the log posterior. So let’s examine that.

The source of the problem

Remember the physics “surface” is the log posterior. So let’s examine that.

$$\begin{aligned} p(\mu,\tau,\theta|y) &\propto p(\mu,\tau,\theta)p(y|\mu,\tau,\theta) \\ &= p(\mu,\tau)p(\theta|\mu,\tau)p(y|\theta) \end{aligned}$$

Taking logs,

$$\begin{aligned} logp(\mu,\tau,\theta|y) &= \mathrm{const} + \log p(\mu,\tau) \\ &- \frac{1}{2}\sum_{j=1}^{J}\left(\frac{\theta_j-\mu}{\tau}\right)^2 \\ &- \frac{1}{2}\sum_{j=1}^{J}\left(\frac{y_j-\theta_j}{\sigma_j}\right)^2 \end{aligned}$$

Reparameterizing the 8 schools

We can reparameterize to a “non-centered” parameterization.

- If $x \sim N(0,1)$, then $\sigma x \sim N(0,\sigma)$
- So, the following is equivalent to our original model:

$$\begin{aligned} y_j &\sim \mathrm{Normal}(\theta_j, \sigma_j) \\ \eta_j &\sim \mathrm{Normal}(0,1) \\ \theta_j &= \mu + \tau\eta_j \\ \mu &\sim \mathrm{Normal}(0,5) \\ \tau &\sim \mathrm{HalfCauchy}(5) \end{aligned}$$

We add a latent variable and move all the variance of θ_j into it; τ no longer appears in the denominator in the log posterior.

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.
- Effective sample size: estimate of the equivalent number of samples *without* autocorrelation. You'll get warnings if this is low.

Both calculated by `pm.summary`; let's take a look.

Summary

Further reading:

- Divergences in the 8 schools model: `https://colcarroll.github.io/pymc3/notebooks/Diagnosing_biased_Inference_with_Divergences.html`
- General notes on using HMC in PyMC3: `https://eigenfoo.xyz/_posts/2018-06-19-bayesian-modelling-cookbook/`

Next week:

- More MCMC
- Model checking