Hamiltonian Monte Carlo

Bayesian posterior sampling

Goal: sample from

$$p(x|y) = \frac{p(x,y)}{p(y)}$$

.

- $x \in \mathcal{X}$: parameters or latent variables.
- $y \in \mathcal{Y}$: observations.

MCMC

- Metropolis-Hastings: need to choose a proposal distribution q.
- We are "taught" to choose a local proposal:
  - Gaussian random walk ensures the state space is recurrent,
  - Acceptance probability should not be too high but not too low.
- Choosing the right proposal distribution requires experience + trial-error.
- The exception is Gibbs sampling, where the proposal is given by the conditional distribution. But Gibbs is not always possible.

Disadvantages of random walk

- Random walk can be inefficient in high dimension – can result in large number of rejections and waste of computation.
- Random walk can get stuck on a mode especially if the local proposal is too small and we may never sufficiently explore the posterior.
- High correlation between the samples.

Disadvantages of random walk

The number of iterations needed to reach a state almost independent of the current state is mostly determined by how long it takes to explore the less constrained direction, which for this example has standard deviation 1.41 — about ten times greater than the standard deviation in the most constrained direction. We might therefore expect that we would need around ten iterations of random-walk Metropolis in which the proposal was accepted to move to a nearly independent state. But the number needed is actually roughly the square of this — around 100 iterations with accepted proposals — because the random-walk Metropolis proposals have no tendency to move consistently in the same direction.

Gradient of posterior

- Gradient of the posterior points in the direction of steepest ascent.
- Gradient based optimization can find the *maximum aposteriori* (MAP) estimate.

Gradient of posterior

- Gradient of the posterior points in the direction of steepest ascent.
- Gradient based optimization can find the *maximum aposteriori* (MAP) estimate.
- In drawing samples from the posterior distribution, we expect more samples from regions with high posterior mass and less samples from regions with small posterior mass.

Gradient of posterior

- Gradient of the posterior points in the direction of steepest ascent.
- Gradient based optimization can find the *maximum aposteriori* (MAP) estimate.
- In drawing samples from the posterior distribution, we expect more samples from regions with high posterior mass and less samples from regions with small posterior mass.
- Rather than using a random proposal, can we explore the posterior space by utilizing the gradient and trace the surface of the posterior?

Hamiltonian dynamics

Imagine a frictionless puck sliding along a surface defined by the posterior distribution.

- x_i: the position of the ball in dimension $i = 1, ..., D$
- m_i: the momentum along dimension i.

Hamiltonian dynamics

Imagine a frictionless puck sliding along a surface defined by the posterior distribution.

- x_i: the position of the ball in dimension $i = 1, ..., D$
- m_i: the momentum along dimension i.

The puck moves at a constant velocity and when it encounters a slope, its momentum allows it to continue, with the kinetic energy decreasing as potential energy builds up.

Hamiltonian dynamics

Imagine a frictionless puck sliding along a surface defined by the posterior distribution.

- x_i: the position of the ball in dimension $i = 1, ..., D$
- m_i: the momentum along dimension i.

The puck moves at a constant velocity and when it encounters a slope, its momentum allows it to continue, with the kinetic energy decreasing as potential energy builds up.

At some point, kinetic energy reaches zero, at which point the puck slides back down with kinetic energy increasing again and the potential energy decreasing.

Hamiltonian dynamics

The Hamiltonian system is described by a function $H(x, m)$ and the change in (x, m) over time t is described by a set of differential equations:

$$\frac{dx_i}{dt} = \frac{\partial H}{dm_i} \tag{1}$$

$$\frac{dm_i}{dt} = -\frac{\partial H}{dx_i}. \tag{2}$$

- The change in position along dimension i is given by derivative of H wrt momentum along i.
- The change in momentum is given by the negative of the derivative of H wrt position (inverse to change in position).

Hamiltonian dynamics

Let $H(x, m) = U(x) + K(m)$, where

- Potential energy: $U(x)$,
- Kinectic energy: $K(m)$.

Hamiltonian dynamics

Let $H(x, m) = U(x) + K(m)$, where

- Potential energy: $U(x)$,
- Kinectic energy: $K(m)$.

$$\frac{dx_i}{dt} = \frac{\partial H}{dm_i} = \frac{\partial K(m)}{dm_i} \tag{3}$$

$$\frac{dm_i}{dt} = -\frac{\partial H}{dx_i} = -\frac{\partial U(x)}{dx_i} \tag{4}$$

Hamiltonian Monte Carlo

How can we utilize Hamiltonian dynamics to draw samples from the posterior?

Hamiltonian Monte Carlo

How can we utilize Hamiltonian dynamics to draw samples from the posterior?

- $U(x) = -\log p(x|y) = -\log p(x, y) + \log p(y)$.
- $K(m) = \frac{1}{2} m^T \Sigma^{-1} m$.

Let ϕ denote zero mean Guassian, then

$$K(m) = \frac{1}{2} m^T \Sigma^{-1} m \propto -\log \phi(m) = -\log \exp\left(-\frac{m^T \Sigma^{-1} m}{2}\right) + C.$$

Hamiltonian Monte Carlo

$$H(x, m) = U(x) + K(m).$$

$$\frac{dx_i}{dt} = \frac{\partial H}{dm_i} \tag{5}$$

$$= [\Sigma^{-1} m]_i \tag{6}$$

If $\Sigma = I$, then the change in position along i-th coordinate is given by the momentum m_i.

Hamiltonian Monte Carlo

$$H(x, m) = U(x) + K(m)$$

$$\frac{dm_i}{dt} = -\frac{\partial H}{dx_i} \tag{7}$$

$$= -\frac{\partial U(x)}{dx_i} \tag{8}$$

$$= \frac{\partial \log p(x, y)}{dx_i}. \tag{9}$$

The change in the momentum accounted by the gradient of the posterior (direction of the steepest ascent).

Hamiltonian Monte Carlo

We previously defined a probability measure on a physical system (e.g., Ising model) in terms of energy in the system; we can associate probability distribution to the physical system z:

$$p(z) = \frac{1}{Z} \exp(-E(z)).$$

For Hamiltonian system, consider $E(z) = H(x, m)$, $z = (x, m)$.

Hamiltonian Monte Carlo

With $H(x, m) = U(x) + K(m)$, the joint distribution of position and momentum is independent:

$$
\begin{aligned}
p(x, m) &\propto \exp(-H(x, m)) && (10)\\
&= \exp(-U(x) - K(m)) && (11)\\
&= \exp(-U(x)) \exp(-K(m)) && (12)\\
&= p(x)p(m) && (13)
\end{aligned}
$$

We set $U(x) = -\log p(x|y)$, yielding $p(x) = \exp(-U(x)) = p(x|y)$.

And set $K(m) = -\log \phi(m)$, yielding $p(m) = \exp(-K(m)) = \phi(m)$.

Hamiltonian Monte Carlo

We now have a joint distribution of position x and momentum m. By construction, the marginal distribution of x is our target distribution:

$$\int p(x, m)dx = p(x).$$

So all that is left is to design an algorithm to sample from $p(x, m)$.

Hamiltonian Monte Carlo

Initialize x_0.

For $t = 1, ..., T$:

- Propose a new momentum $m \sim N(0, I)$
- $(x', m') = \text{SimulateHamiltonian}(x_{t-1}, m)$
- Compute acceptance probability:

$$A = \min(1, \exp(U(x) - U(x') + K(m) - K(m')))$$

- Draw $u \sim \text{Uniform}(0, 1)$
  - if $u < A$ accept $x_t = x'$
  - otherwise $x_t = x_{t-1}$.

Leap frog

How do we simulate Hamiltonian dynamic?

Given a position x and momentum m at time t, Euler's formula provides a discretization of the continuous time dynamic that approximates the position and momentum of the system at time $t + \epsilon$ for some $\epsilon > 0$:

$$m(t + \epsilon) = m(t) + \epsilon \frac{\partial p}{dt}(t) \tag{14}$$

$$x(t + \epsilon) = x(t) + \epsilon \frac{\partial q}{dt}(t). \tag{15}$$

Leap frog

Adapted to our choice of $H(x, m) = U(x) + K(m)$ (fill in the blanks):

Leap frog

Adapted to our choice of $H(x, m) = U(x) + K(m)$ (fill in the blanks):

$$m(t + \epsilon) = m(t) - \epsilon \frac{\partial U}{dx}(x(t)) \tag{16}$$

$$x(t + \epsilon) = x(t) + \epsilon \Sigma^{-1} m(t + \epsilon). \tag{17}$$

Leap frog

Euler's discretization can be a bit crude.

Modified Euler: first take half step for the momentum followed by taking the full step on the position, and then taking another half step for the momentum.

$$m(t+\epsilon/2) = m(t) - (\epsilon/2)\frac{\partial U}{dx}(x(t)) \tag{18}$$

$$x(t+\epsilon) = x_i(t) + \epsilon\Sigma^{-1}m(t+\epsilon/2) \tag{19}$$

$$m(t+\epsilon) = m(t+\epsilon/2) - (\epsilon/2)\frac{\partial U}{dx}(x(t+\epsilon)). \tag{20}$$

We can repeat the above update for L steps to simulate the puck sliding according to the Hamiltonian dynamics, utilizing the surface of the distribution given by $p(x, m)$ to update the position and momentum.

Leap frog

Hamiltonian dynamic is reversible and volume preserving (shown in the paper).

But leap frog may introduce error due to discretization, Metropolis accept-reject is needed to correct for the error.

Why does this work?

For a Markov chain Monte Carlo algorithm to "work", we need to prove that the samples drawn from the algorithm are from the right target distribution in our case, $p(x, m) = \exp(-H(x, m))$.

To that end, we need to show

1. Stationarity,
2. Ergodicity.

For Metropolis-Hastings and Gibbs, we showed that it satisfies a stronger condition of detailed balance, which implies stationarity.

Detailed balance

Let F be the mapping defined by a Hamiltonian simulator. Let the current system configuration be $z = (x, m)$ such that $z' = F(z)$ (this mapping is deterministic given current position and momentum).

Reversible mapping means $T^{-1}(z') = z$ with the direction of the momentum reversed:

- Re-run the simulation starting at $(x', -m')$ for L steps with same ϵ.
- The final state will be $(x, -m)$, flipt the momentum and we obtain (x, m).

Detailed balance

Let K be the HMC kernel. We need to show.

$$p(z)K(z \to z') = p(z')K(z' \to z).$$

Detailed balance

Let K be the HMC kernel. We need to show.

$$p(z)K(z \to z') = p(z')K(z' \to z).$$

This is trivially true for self-transition (if $T(z)$ is rejected, then $z = z'$).

In the case that the new state $z' = T(z)$ is accepted, the kernel is $1 \wedge a(z'|z) = 1 \wedge \frac{\exp(-H(z'))}{\exp(-H(z))}$.

Detailed balance

Case 1: $\exp(-H(z)) > \exp(-H(z'))$,

$$p(z)K(z \to z') = \exp(-H(z))\frac{\exp(-H(z'))}{\exp(-H(z))} \tag{21}$$

$$= \exp(-H(z')). \tag{22}$$

On the other side,

$$p(z')K(z' \to z) = \exp(-H(z')) \wedge 1 = \exp(-H(z')). \tag{23}$$

Detailed balance

Case 2: $\exp(-H(z)) \leq \exp(-H(z'))$,

$$p(z)K(z \to z') = \exp(-H(z)) \wedge 1 \tag{24}$$
$$= \exp(-H(z)). \tag{25}$$

On the other side,

$$p(z')K(z' \to z) = \exp(-H(z'))\frac{\exp(-H(z))}{\exp(-H(z'))} \tag{26}$$
$$= \exp(-H(z)). \tag{27}$$

Ergodicity

- The momentum variable is sampled from global proposal (mean zero Gaussian with identity matrix).
- This momentum can affect the position variable in an arbitrary way.

Together, this means that all state is recurrent and aperiodic.

The only cautionary note is that the leapfrog algorithm may lead to periodicity in some cases – this can be avoided by randomizing the selection of ϵ or L.

Ergodicity

- The momentum variable is sampled from global proposal (mean zero Gaussian with identity matrix).
- This momentum can affect the position variable in an arbitrary way.

Together, this means that all state is recurrent and aperiodic.

The only cautionary note is that the leapfrog algorithm may lead to periodicity in some cases – this can be avoided by randomizing the selection of ϵ or L.

No U-Turn Sampler (NUTS): an improvement over Leap frog (also alleviates the above concern).

Autodiff

- Autodiff computes derivatives by systematically applying the chain rule to the operations performed by your code. It "traces" the computation graph and efficiently computes the derivatives.
- It outputs the **numerical values** of the derivatives at specific input values without generating an explicit symbolic expression.
- Efficiency: (in reverse mode) very efficient for functions with many inputs and a single output.

Autodiff

- Autodiff has propelled gradient based optimization, widely used for training complex neural networks.
- Using autodiff and HMC, it is now feasible to perform Bayesian analysis on complex models without manual derivative derivation/difficulties of MH.
- These techniques have also enabled HMC to be widely deployed as part of software packages where the user only needs to specify the model, also known as **probabilistic programming frameworks** (e.g., Stan, PyMC, TensorFlow probability, Pyro, NumPyro, BlackJax).
- Difference from symbolic differentiation?

Symbolic differentiation

- Symbolic differentiation manipulates the mathematical expression symbolically and derives an explicit formula for the derivative.
- Output is a **symbolic expression**. For complex models, this can lead to an overwhelming expression that is hard to simplify.
- Not very practical for large-scale computational problems due to the overhead in generating and simplifying these expressions.

Examples

- Go over the figures in the paper.

Example: Multinomial regression

$k = 1, ..., K$ categories and $p = 1, ..., P$ covariates. Posterior:

$$p(\beta|Y) \propto \prod_{n=1}^{N} p(y_n|\beta)p(\beta), \tag{28}$$

where,

$$Y_n|\beta \sim \frac{\exp(\beta_k^T x_n)}{\sum_{k'=1}^{K} \exp(\beta_{k'}^T x_n)} \tag{29}$$

$$\beta_k \sim N(0, I_{P\times P}). \tag{30}$$

There are $(K-1) \times P$ parameters in this model (the last category chosen as the "pivot" to avoid overparameterization and ensure identifiability).

Example: Multinomial regression

- Implement HMC (NumPyro).

Example: Gaussian mixture model

Implement MH in class.