Extending Surrogate Hamiltonian Monte Carlo

PSTAT 215

Abhijit Brahme

Outline

- Background
  i. Hamiltonian Dynamics + HMC
  ii. Numerical Integration
  iii. Neural ODE
- Contribution
- Results

Hamiltonian Systems

The Hamiltonian, $H(q, p)$, is a function of q and p, parameters defining an object's position and velocity.

We can decompose the Hamiltonian into the following, if the Hamiltonian is separable: $H(q, p) = U(q) + K(p)$. $U(q)$ and $K(p)$ represent potential and kinetic energy of the system respectively.

Hamiltonian Systems (cont.)

The dynamics of the system can be modeled by Hamilton's equations:

$\frac{\partial H}{\partial q} = -\frac{\partial p}{\partial t}$

$\frac{\partial H}{\partial p} = \frac{\partial q}{\partial t}$

Thus, the system can be evolved over time using numerical integration schemes that are symplectic.

Hamiltonian Systems (cont.)

A Hamiltonian system has key properties:

1. Time Reversibility
2. Conservation of Hamiltonian
3. Volume Preservation (Liouville's Theorem)
    a. Probability density doesn't change !

Hamiltonian Monte Carlo

Position given by parameters q, momentum p drawn from kinetic energy distribution, typically an Isotropic Gaussian.

$U(q) = -log(\pi(q))$

$K(p) = \frac{p^T p}{2}$

Computation of $\frac{\delta U(q)}{\delta q}$, $U(q)$ can be costly sometimes!

Leapfrog Integration

Assuming a separable Hamiltonian form commonly seen in Hamiltonian Monte Carlo, $H(q,p) = U(q) + \frac{p^T p}{2}$, one may use the following update rules for a given starting point in the phase space (q_0, p_0) to simulate dynamics with a fixed step size ϵ :

1. $q_{t+1} = q_t + \epsilon p_t + \frac{\epsilon^2}{2} \frac{\delta p_t}{\delta t}$
2. $p_{t+1} = p_t + \frac{\epsilon}{2} (\frac{\delta p_t}{\delta t} + \frac{\delta p_{t+1}}{\delta t})$

In the case where the Hamiltonian is not separable (RMHMC), the expression is more complex.

Why Leapfrog?

1. Second order method, so it is comparatively fast
2. It is a symplectic integrator, so it preserves the Hamiltonian / volume
    a. Some higher order methods like Runge-Kutta allow the system to drift
3. It is time reversible.

Motivation

The cost Ω of using Hamiltonian Monte Carlo in practice is $\propto K \cdot L \cdot C(N, D, T)$, where K is the number of samples and L is the number of leapfrog steps per sample. The function $C(N, D, T)$ is the cost associated with auto-differentiating a log probability function that requires T operations, for N observations in a parameter space of dimension D.

The costly part of HMC is $C(N, D, T)$. Some models, like GPs don't scale linearly with N, so the bottleneck of sampling becomes gradient evaluations.

The goal is to efficiently and accurately approximate gradient operations such that $\hat{C}(N, D, T) << C(N, D, T)$, while also preserving Hamiltonian dynamics during sampling.

Previous Work

1. Rasmussen
    a. Model log posterior $U(q)$ with a GP, then use its analytical derivative
2. Li, Holbrook, et.al
    a. Approximate the gradient using a Neural Network (NNgHMC)
3. Dhulipala et. al
    a. Approximate the Hamiltonian function itself using HNNs, and take its partial derivatives

All of these approaches work well, but can be improved, as they don't take into account the trajectory during training (throwing away data).

Contribution

Combine Neural ODEs using symplectic integration schemes with Hamiltonian NNs to more efficiently approximate Hamiltonian trajectories during sampling.

What does “more efficiently” mean?

1. Increase in number of effective samples produced / second
2. Use less training samples / training time / parameters than previous “surrogate” methods
3. Follow Hamiltonian dynamics more accurately (less deviation from trajectory, preservation of Hamiltonian, reversibility, etc.)

Methods

Let $f(\theta, q, p)$ be a Neural Network approximating the derivatives of the Hamiltonian of a system, $\frac{\partial H}{\partial q}$ and $\frac{\partial H}{\partial p}$, or equivalently $-\frac{\partial p}{\partial t}$ and $\frac{\partial q}{\partial t}$.

1. Collect $\{(q_i^0, p_i^0)\}_{i=1}^K$, from a burn-in period of K samples of trajectory length L
2. Collect $\{(Q_i, P_i)\}_{i=1}^K$, where Q_i and P_i are vectors of length L corresponding to an L length trajectory for burn-in sample i.
3. $(\hat{Q}, \hat{P}) = ODESolve(f(\theta, q^0, p^0), L, \epsilon)$

Neural ODE

Can be used to model continuous time dynamics. Note that if we aim to approximate a smooth function $y = f(x, \theta)$, then it is also equivalent to approximate $y = \int_{t_0}^{t_1} f^{'}(x, \theta) dt$.

1. Model $f^{'}(x, \theta)$
2. Integrate $f^{'}(x, \theta)$ to obtain $f(x, \theta)$
3. Backpropagate loss from $L(f(x, \theta), y)$ through ODE Dynamics

Methods (cont.)

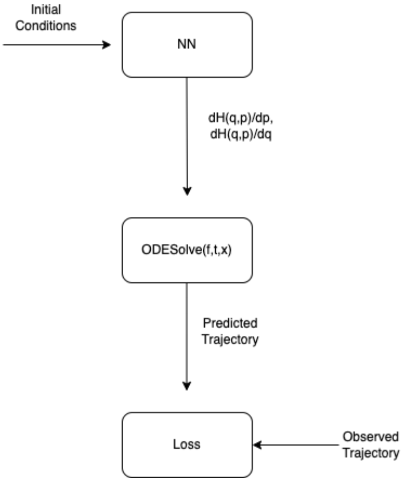

Figure 1: NNODE

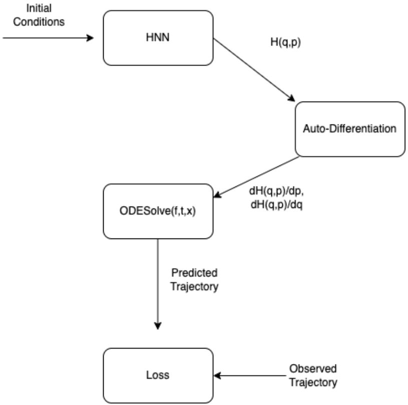

Figure 2: HNNODE

Synthetic Results (Reversibility & Posterior Samples)

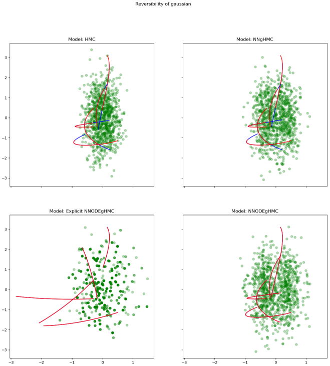

Figure 3: 2D Gaussian

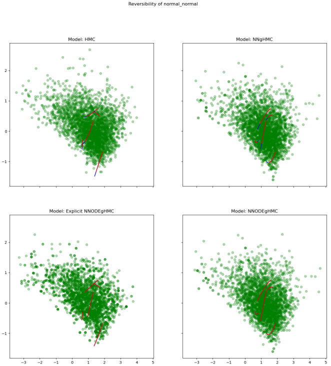

Figure 4: Normal-Gamma

Synthetic Results (Sample Efficiency)

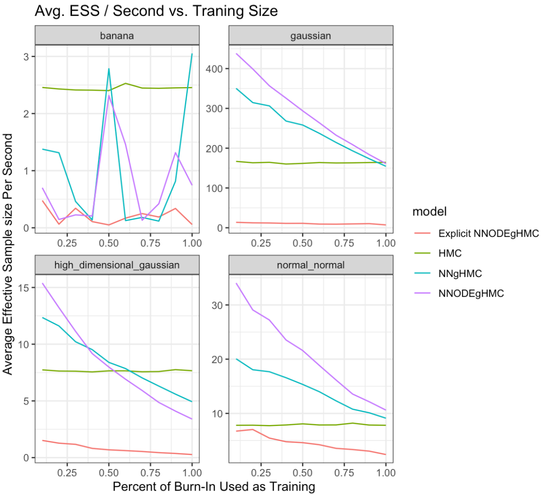

Figure 5: Varied percent of trajectories used as training, and plotted ESS as a result.

Future Work

1. Extend to Riemannian Manifold HMC
    a. Probably the most important, as this is very slow.
2. Instead of modeling the trajectory, look to find a symplectic map instead.
3. Integrate this approach with No U-Turn Sampling
4. Analysis of more challenging distributions

Notes

1. All work done in pytorch (hamiltorch package)

Questions ?