ALPhA Summer Presentation 2

Braden Kronheim

June 10, 2019

Project Description

- An important theortical extension to the Standard Model is Supersymmetry.
- This class of theories predict partner partner particles for each of the known particles from the Standard Model.
- In order to study these theories it is necessary to calculate the cross section, or probability of creating, these particles.
- There are currently two algorithms named Prospino and SUSY-HIT which can do this calculation, but they are extremely slow.
  - These algorithms can generate points at a rate of about 16.6 points per CPU hour, which is way too slow.
  - The full Minimal Supersymetric Standard Model (MSSM) has 105 parameters, which on its own is far too many, as espcially so with such slow algorithm.
  - The constrained theory used for the project, the phenomenological MSSM (pMSSM), only has 19 parameters. This is slightly more tenable, but still only with a much faster alogithm.
- The goal of my project is to use Bayesian neural networks to learn the output of these algorithms.

Bayesian Neural Networks

- A Bayesian neural network is obtained by assigning a prior distribution to all of the weights and biases in the network and applying Bayes' formula, which is:

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

  where θ represents the network parameters and D is the training data.
- This formula obviously cannot be directly solved, so it is simulated via a Markov Chain, a method of generating samples of what θ should be given D.
- A Markov Chain requires a method of generating these samples. The algorithm used for this is Hamiltonian Monte Carlo, which generates new states by simulating the Hamiltonian dynamics for a state where the potential energy is the negative log probability of the distribution being sampled from.

Progress Summary

- Migrated last semester's code over to TF 2.0.
- Implemented Cauchy priors for network parameters
- Implemented Prelu activation function
- Built a leapfrog integrator in Tensorflow for non-separable Hamiltonians
- Built a Transition Kernel in Tensorflow for Riemann Manifold Hamiltonian Monte Carlo.

Cauchy Priors

- Cauchy Priors have fatter tails then Gaussian distributions.
- These fatter tails allow the network parameters to grow larger with less of a penalty. This makes it easier for hidden features to develop within the network.
- While I was implementing this, I saw that previously I had been scaling the parameter log probability in each layer by the number of parameters.
  - I took this out because I don't think it was legitimate.
- Results after 400 iterations:

Prior Configuration	Squared Error	Percent Error
Tight Cauchy	0.00264	17.405
Loose Cauchy	0.00293	18.542
Tight Gaussian	0.00271	17.740
Loose Gaussian	0.00245	16.483

Riemann Manifold Hamiltonian Monte Carlo

- This method extends normal Hamiltonian Monte Carlo by using a metric which measures the local geometry of the distribution.
- The metric becomes the mass matrix in the Hamiltonian, which is:

$$H(\theta, \boldsymbol{p}) = -\mathcal{L}(\theta) + \frac{1}{2}\log\left((2\pi)^D|\boldsymbol{G}(\theta)|\right) + \frac{1}{2}\boldsymbol{p}^\mathsf{T}\boldsymbol{G}(\theta)^{-1}\boldsymbol{p}$$

- The $\boldsymbol{G}(\theta)$ being used here is the negative Hessian matrix of the log probability.
- This Hamiltonian has the problem of not being separable, meaning the potential and kinetic energy terms are both dependent upon position of the particle.
- Normal leapfrog cannot be used in this case.

Explicit General Leapfrog Algorithm

- The general leapfrog algorithm presented in the RMHMC paper which could deal with the non separable Hamiltonian was implicit, meaning it required fixed-point iteration to solve.
- An alternative method was detailed in a paper from 2016 which presented an explicit way of solving the system. Instead of soling the Hamiltonian $H(\theta, p)$, this method solves the Hamiltonian

$$\bar{H}(\theta, p, x, y) = H(\theta, y) + H(x, p) + \frac{\omega}{2}||\theta - x||_2^2 + \frac{\omega}{2}||p - y||_2^2.$$

- This is solved via $\phi_2^\delta := \phi_{H_A}^{\delta/2} \circ \phi_{H_B}^{\delta/2} \circ \phi_{H_{\omega C}}^{\delta/2} \circ \phi_{H_B}^{\delta/2} \circ \phi_{H_A}^{\delta/2}$ where

$$\phi_{H_A}^{\delta/2} = \begin{bmatrix} \theta \\ p - \delta\partial_\theta H(\theta, y) \\ x + \delta\partial_y H(\theta, y) \\ y \end{bmatrix}, \phi_{H_B}^{\delta/2} = \begin{bmatrix} \theta + \delta\partial_p H(x, p) \\ p \\ x \\ y - \delta\partial_x H(x, y) \end{bmatrix}$$

Explicit General Leapfrog Algorithm (cont.)

$$\phi_{H_{\omega C}}^{\delta/2} = \frac{1}{2}\begin{bmatrix} \begin{pmatrix} q+x \\ p+y \end{pmatrix} + R(\delta)\begin{pmatrix} q-x \\ p-y \end{pmatrix} \\ \begin{pmatrix} q+x \\ p+y \end{pmatrix} - R(\delta)\begin{pmatrix} q-x \\ p-y \end{pmatrix} \end{bmatrix}, \text{and}$$

$$R(\delta) := \begin{bmatrix} \cos(2\omega\delta)I & \sin(w\omega\delta)I \\ -\sin(2\omega\delta)I & \cos(w\omega\delta)I \end{bmatrix}$$

- Higher order approximations can be obtained by combining different sized leapfrog steps together. The fourth order approximation is the most efficient.

Results

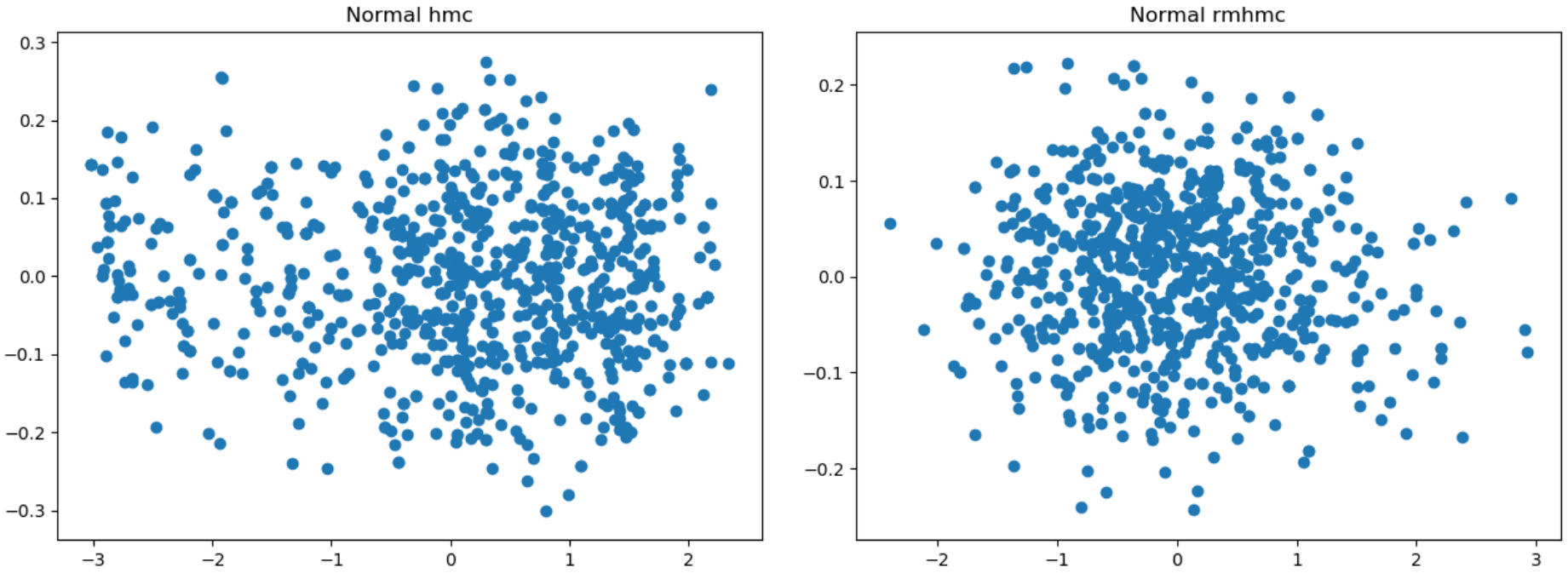

Figure: Normal HMC (left), RMHMC (right) sampling from a bivariate normal distribution with standard deviation 1.0 in the x-direction and 0.1 in the y-direction.

Goals for Next Week

- Finish the RMHMC code and try running it with the Bayesian neural network.
- Perform a longer run with the Cauchy Priors and possibly with the Prelu activation
- Perform the correct error analysis on my best run from last semester.
- Analyze the error directions of worst error more thoroughly, possibly do something like PCA.
- Try and interpret this in the context of SUSY theory.