JP Morgan Report
Hamiltonian Neural Network and Hamiltonian Monte Carlo
Name: Xingguang (Ray) Jin
Email: RayJIN@link.cuhk.edu.hk)

Question 1: Part 1

(a) Consider the paper "Efficient Bayesian Inference with Latent Hamiltonian Neural Networks in No-U-Turn Sampling" Dhulipala(23).

(b) Please write a paper explaining parts that are hard to understand in the original paper. Please point out any typos and any details that are essential for the implementation but missing in the paper.

(c) Replicate the results with Tensorflow and Tensorflow Probability. Can you get the same results as the Pytorch code of the authors?

(d) Is the TFP code on HMC useful for your implementation?

1 Literature review

Bayesian inference is a statistical method grounded in Bayes' theorem, which describes how to update the probability of a hypothesis given new evidence. Bayes' theorem is expressed as:

$$P(H|D) = \frac{P(D|H) \times P(H)}{P(D)},$$

where $P(H|D)$ is the posterior probability, $P(D|H)$ is the likelihood, $P(H)$ is the prior probability, and $P(D)$ is the marginal likelihood. From the expression of Bayes' theorem, we can see the core components of Bayes' theorem. (1) Prior Probability: Represents the initial beliefs about a parameter before observing any data. (2) Likelihood: The probability of observing the data given the parameter. (3) Posterior Probability: The updated belief about the parameter after incorporating the evidence.

Econometrics refers to the application of econometric methods to analyze economic and financial data, particularly under uncertainty. Bayesian inference offers several advantages in this field based on the three core components of Bayes' theorem.

(1) Incorporation of Prior Knowledge: Researchers can include prior beliefs about economic parameters, such as historical trends or expert opinions, which can help refine estimates. (2) Uncertainty Quantification: Bayesian methods provide a natural framework for quantifying uncertainty in model predictions, crucial for decision-making in economics and finance. (3) Flexibility in Modeling: Bayesian inference allows for complex hierarchical models that can capture relationships across different levels of economic data.

Challenges with Bayesian Inference While Bayesian inference is powerful, it often faces challenges, particularly regarding computational costs:

(1) High Computational Demand: The need to compute posterior distributions, especially in high-dimensional spaces, can be intensive. Traditional methods like Markov Chain Monte Carlo (MCMC) can be slow, particularly when the model is complex or the data set is large. (2) Convergence Issues: Ensuring convergence of MCMC samples to the true posterior distribution can require significant computational resources and careful diagnostics.

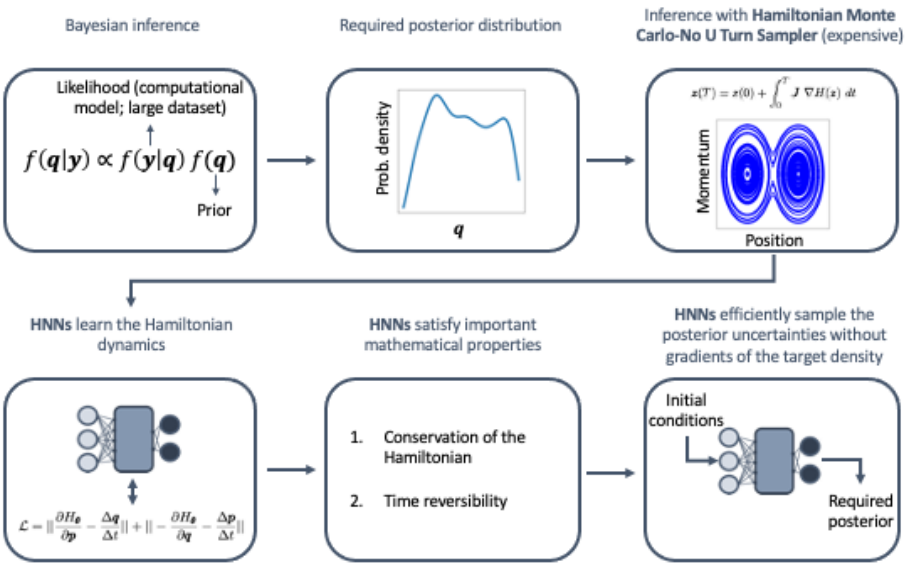


Figure 1: Schematic of the use of (L)-HNNs for efficient Bayesian inference.

Due to the high computational cost of Bayesian inference, machine learning has garnered considerable interest in accelerating this process. The following Table 1 summarizes existing methods for solving Bayesian inference problems. Based on Question 1 in paper [2], we primarily focus on Hamiltonian Monte Carlo (HMC)[1] and Hamiltonian Neural Networks (HNN) [3] for addressing Bayesian inference challenges, as both methods are efficient in reducing computational costs and improving convergence rates.

HMC and HNN share common features, as both utilize Hamiltonian dynamics to propose new states, making them particularly efficient in high-dimensional spaces. However, the HMC class of algorithms can require numerous numerical gradients of the target density, which can be computationally expensive. In contrast, HNN combines HMC with neural networks, allowing it to model complex relationships and uncertainties. This approach is efficient in high dimensions and can learn complex posterior distributions without the need for numerical gradient computation.

Table 1: Overview of different algorithms

Methods	Features
Markov chain Monte Carlo (MCMC)	Simple to implement with any given distribution
Random-walk MCMC	Weak perform in high-dimensional spaces
Hamiltonian Monte Carlo (HMC)	Require numerous numerical gradients
Hamiltonian Neural Network (HNN)	No numerical gradients
Latent Hamiltonian neural networks (L-HNNs)	Reduced integration errors

2 Computational case studies

All the code in this part can be found in my github website and the code is written in python 3 with the tools tensoflow and TensorFlow Probability (TFP).

2.1 Hamiltonian Monte Carlo (HMC)

TensorFlow Probability (TFP) and Tensorflow provide a robust implementation of Hamiltonian Monte Carlo (HMC) that can be very useful for implementing Bayesian inference problems.

- Open *tenp.py* to visualize TFP's HMC to sample from a target distribution.

  Here, we choose Normal distribution between $[-4, 4]$ as a test example.

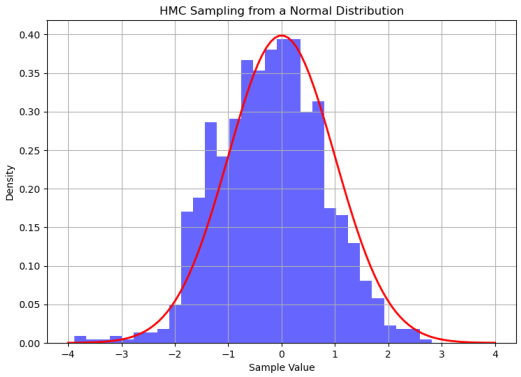


Figure 2: HMC Sampling from a Normal Distribution.

- Open *hmc.ipynb* to visualize HMC to sample from 1D Gaussian mixture density.

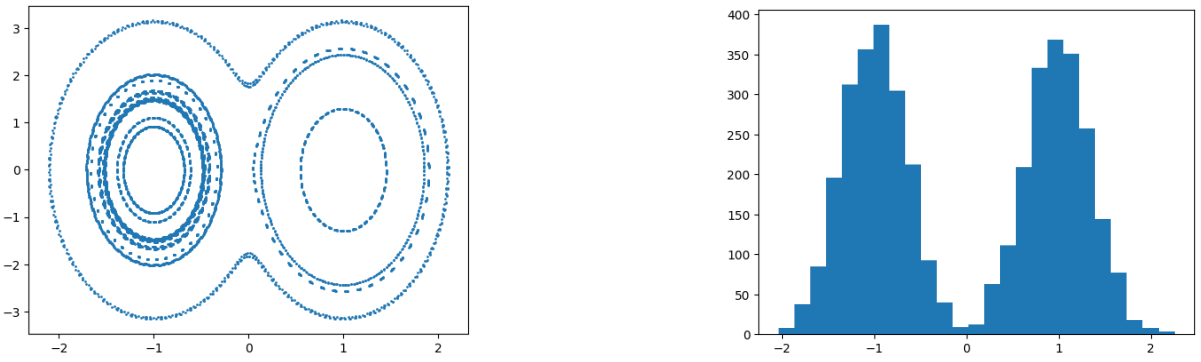
  In this section, we reproduce the results presented in Figure 2 of Soma et al [2]. The results are plotted in Figure 3 of our analysis. We also compute the required numerical gradients of the target density and find that L-HNN requires only 1.6% of the numerical gradients needed by HMC (200,000/3,200). It is important to note that the expression for the 1D Gaussian mixture density in Soma et al [2] contains an error; specifically, it omits a crucial minus sign.

  The definition of the loss function (13) in Soma et al [2] should be updated to the following formula:

$$L = \frac{1}{M}\sum_{i=1}^{M}\int(\frac{\partial H_\theta}{\partial p}(t) - \frac{\Delta q}{dt}(t))^2 dt + \int(\frac{-\partial H_\theta}{\partial p}(t) - \frac{\Delta q}{dt}(t))^2 dt.$$

  The following 1D Gaussian mixture density was considered :

$$f(q) \propto 0.5\exp(-\frac{(q-1)^2}{2\times 0.35^2}) + 0.5\exp(-\frac{(q+1)^2}{2\times 0.35^2}) \tag{1}$$



((a)) ((b))

Figure 3: (a) Phase space plot corresponding to different energy levels. (b) Histogram of samples from a $1D$ Gaussian mixture density using traditional HMC.

2.2 Hamiltonian neural nets (HNNs)

I chose the 2D Neal's funnel density for this analysis and successfully reproduced the results shown in Figure 8 of [2]. For any other figures, you only need to modify the density functions accordingly.

The following $2D$ Neal's funnel density was considered :

$$f(q_1, q_2) \propto \begin{cases} q_1 = \mathcal{N}(0, 3) \\ q_2 = \mathcal{N}(0, \exp^{q_1}). \end{cases} \tag{2}$$

- Open $2d\_neal\_funnel\_density.py$.

  Set the step to be 25000. It takes 12 hours to obtain the Figure 4 and Figure 5.

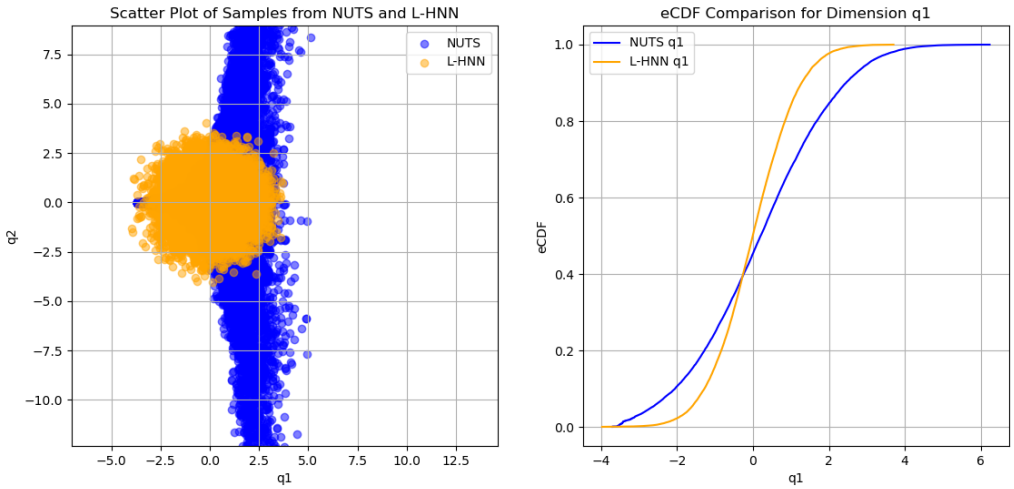


Figure 4: (Left) Scatter plot comparison of samples generated from a 2-D Neal's funnel density, using NUTS and L-HNNs; (Right) Comparison of the eCDF plots for the dimensions q_1.

- Include the Hamiltonian of the required probability distribution in the $functions.py$ and set the step to be 1000. It takes 20 mins to obatin the Figure 6 and Figure 7.

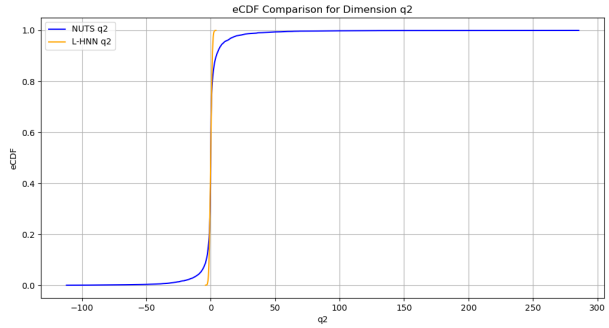


Figure 5: Comparison of the eCDF plots for the dimensions q_2.

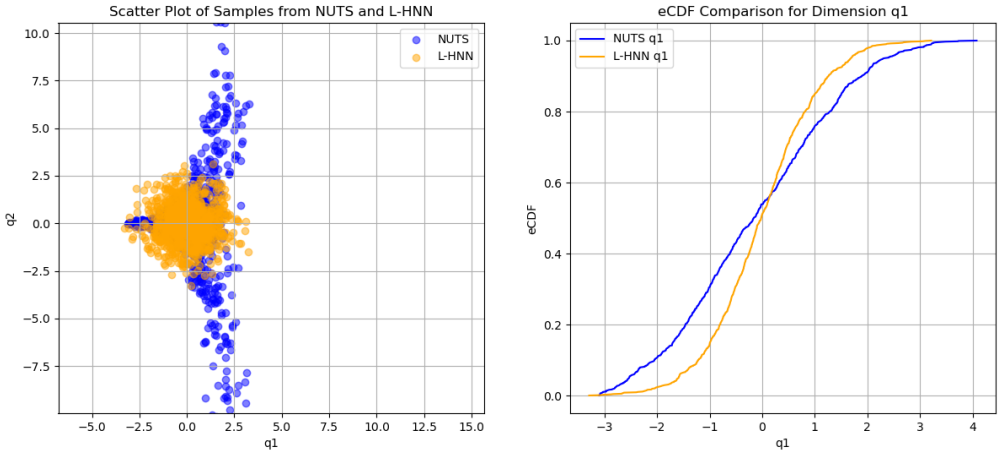


Figure 6: (Left) Scatter plot comparison of samples generated from a 2-D Neal's funnel density, using NUTS and L-HNNs; (Right) Comparison of the eCDF plots for the dimensions q_1.

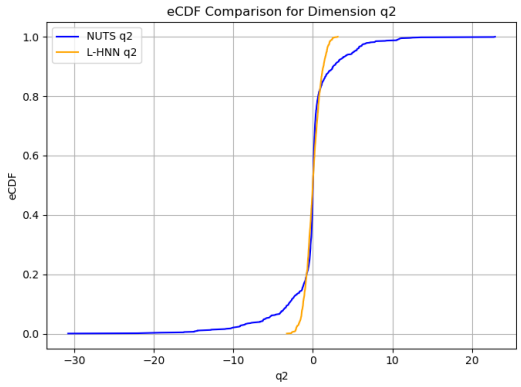


Figure 7: Comparison of the eCDF plots for the dimensions q_2.

2.3 Convergence test of the original Pytorch code

- Include the Hamiltonian of the required probability distribution in the *functions.py*. Here, I choose *nD_standard_Gaussian* as the probability distribution function.

- Adjust the parameters in *get_args.py*. Here, I choose the input dimension to be 4, which means 2 variables (q_1, q_2) as the position variables and 2 variables p_1, p_2 as the momentum variables. The total steps are set to be 5000 and the learning rate is $5e-4$.

- Run *train_hnn.py* to train the HNN model.

  In the original code, the test lose is NAN, which means the neural network is not convergent. In order to obtain the valid data for the next step, I choose the 80% data as the training data and visualize the loss in Figure 8.

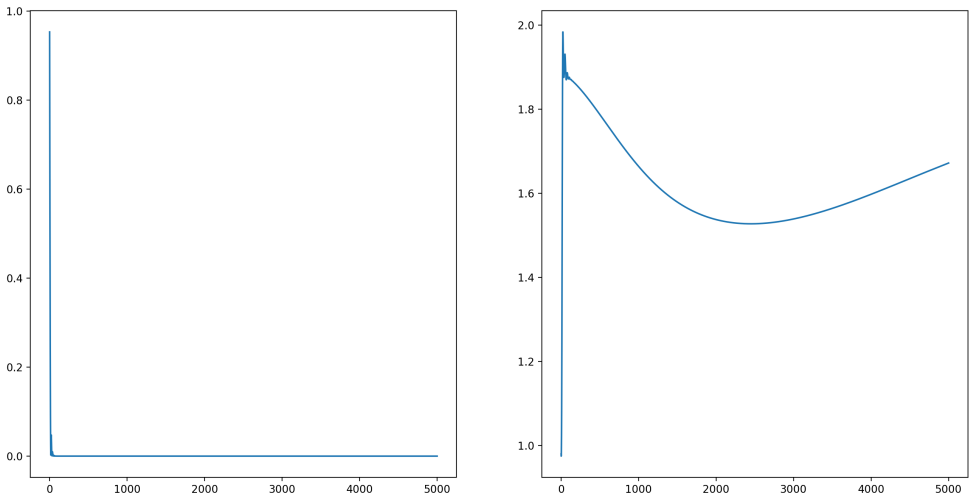


Figure 8: (Left) Training loss and (Right) is the Testing loss.

- Run *hnn_hmc.py* to perform Hamiltonian Monte Carlo with the trained HNN model. See the sampling in Figure 9.

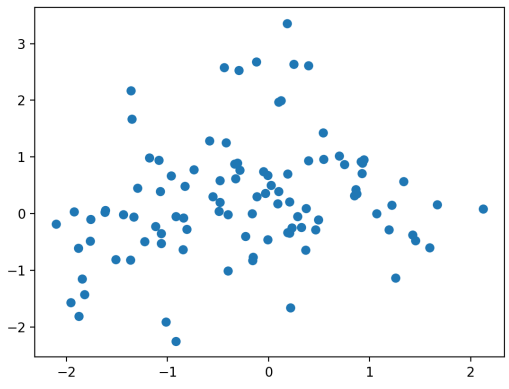


Figure 9: nD standard Gaussian Sampling

3 Next plan

Thank you for this wonderful opportunity to learn and tackle Bayesian inference problems. I'm currently fine-tuning the neural network to enhance convergence. I believe that the HNN is similar to a type of Physics-Informed Neural Network (PINN). Stay tuned for my second report! Feel free to email me for a detailed discussion.

References

[1] MJ Betancourt. The fundamental incompatibility of hamiltonian monte carlo and data subsampling. *arXiv preprint arXiv:1502.01510*, 2015.

[2] Somayajulu L.N. Dhulipala, Yifeng Che, and Michael D. Shields. Efficient bayesian inference with latent hamiltonian neural networks in no-u-turn sampling. *Journal of Computational Physics*, 492:112425, 2023.

[3] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.