Leveraging Dense Neural Networks for Improved Hamiltonian Monte Carlo Sampling

Alexander Kim

Abstract

This paper introduces a combination of neural networks and Hamiltonian Monte Carlo (HMC) sampling. HMC is a powerful method used for sampling from complex probability distributions. However, it requires hand-tuning of two parameters epsilon ϵ and number of steps L. The ability of a neural network to learn complex relations even when given high-dimensional inputs is leveraged to adjust the parameter epsilon ϵ during a warm-up period. The HMC algorithm after warm-up is not altered to preserve ergodicity, which guarantees convergence to the desired target distribution given that this distribution exists.

1 Introduction

At the heart of my proposed algorithm lies two fundamental ideas: dense neural networks and Markov chains. The following is my attempt at a concise but complete explanation to these topics.

1.1 Artificial Neural Networks. Artificial neural networks (or simply neural networks) mathematically mimic the brain's biological neural network in learning complex relations within data. In essence, neural networks consist of sequential layers that assign each input a corresponding multiplicative weight and additive bias. Through numerous training epochs, these weights and biases are adjusted to best fit the data. Ideally, a neural network should use the data to generalize itself to perform well with previously unseen test data. The simplest neural network is a dense neural network, which is composed of dense layers.

Dense layer. Given inputs X, weights W, bias β, and activation function φ, the output of a dense layer is $\varphi(X^T W + \beta)$.

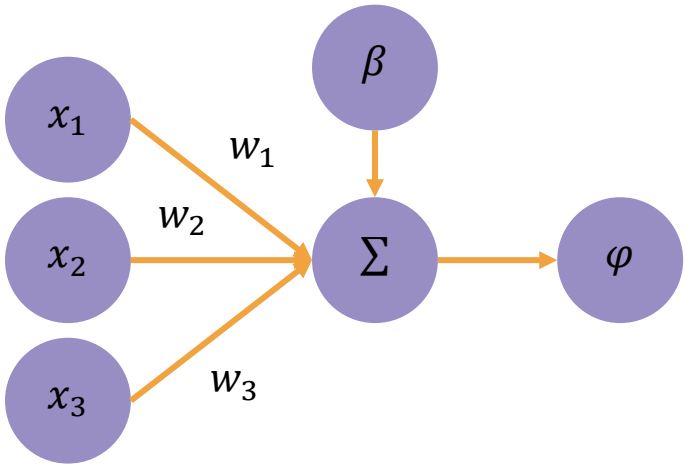


Figure 1: Visualization of a dense layer with three inputs and one neuron.

Simply put, the activation function determines the relative importance of the output. These functions are crucial to the complex performance of neural networks.

1.2 Markov Chains. Markov chains are a set of statistical models that can simulate the progression of random states through time. Markov chains satisfy the Markov property, which states that the next sampled state is only dependent on the current state. Given $X_0,\ X_1, \ldots, X_t$, the Markov chain at timestep t, the probability distribution of X_{t+1} depends only on X_t [1].

$$P(X_{t+1}|X_t) = P(X_{t+1}|X_t,\ X_{t-1}, \ldots,\ X_0) \tag{1.1}$$

The following is a brief rundown of important characteristics of Markov chains and a fundamental theorem regarding the convergence of a Markov chain.

Stationary distribution. The stationary distribution of a Markov chain is the unique distribution to which the chain converges to. Additional samples do not affect the distribution. One way to guarantee the existence of a stationary distribution is to make sure the chain satisfies a condition termed detailed balance. The specifics behind detailed balance are largely irrelevant in this paper.

Periodicity. A periodic Markov chain can only reach certain states in an additional number of time steps that are multiples of a number called the period. An aperiodic Markov chain can reach any state at any finite time step.

Reducibility. A reducible Markov chain cannot transition to certain destination states from select current states. On the other hand, an irreducible Markov chain can transition to any state regardless of the current state.

Ergodicity. Any Markov chain that is aperiodic and irreducible is considered ergodic. Ergodic Markov chains are guaranteed to converge to a unique stationary distribution given that the stationary distribution exists [1].

Basic Limit Theorem of Markov chains. The distribution π_t of states of an ergodic Markov chain $\{X_0,\ X_1, \ldots, X_t\}$ at time t should converge to a stationary distribution π_s, given that π_s exists [2].

$$\lim_{t\to\infty} \pi_t = \pi_s$$

1.3 Markov Chain Monte Carlo (MCMC). Markov chain Monte Carlo simulations are used to sample from complex probability distributions by constructing a Markov chain of samples such that $\pi_s = P$ for the target density P. Many probability distributions in practical problems cannot be sampled directly, less so integrated directly. In general, the posterior distribution $P(x)$ is only known up to a proportionality constant k. Many times, P will simply be expressed as being proportional to the known distribution p [3].

$$P(x) \propto p(x) \Leftrightarrow P(x) = kp(x)$$

$P(x)$ may also be referred to as the normalized probability density function, k as the normalization constant, and $p(x)$ as the unnormalized probability density function. One can solve for the normalization constant by integrating both sides over all possible states in the sample space Ω. By definition, the integral of the normalized probability density function $\int_\Omega P(x)dx = 1$. Therefore

$$k = \frac{1}{\int_\Omega p(x)dx} \tag{1.2}$$

In sum, the general normalized probability density function

$$P(x) = \frac{p(x)}{\int_\Omega p(x)dx}$$

Many times, calculating this normalization constant (1.2) is intractable. MCMC provides a way to approximate $P(x)$ without directly calculating the normalization constant [3].

1.4 Metropolis-Hastings Algorithm. One MCMC method is the Metropolis-Hastings algorithm. This algorithm introduces random walk behavior into exploring a probability distribution $P(x)$. At each time step, a new proposal state X^* is generated from a proposal distribution $g(x)$. The proposal is accepted or rejected based on the relative probability difference between the current state and proposed state $\frac{P(X^*)}{P(X_t)}$ [4]. Put simply, proposed values with higher values of $P(X^*)$ get accepted more frequently. By taking a ratio, the pesky denominator cancels out of $P(x)$, effectively allowing sampling without a normalization constant.

Algorithm 1: Metropolis-Hastings Algorithm

Input: Initial state X_0. Number of trials n. Proposal distribution $g(x)$.

For t from 0 to $n-1$ **do**
  Propose: $X^* \sim g(x|X_t)$
  Acceptance probability: $\alpha(X^*, X_t) \leftarrow \min\left\{\frac{P(X^*)}{P(X_t)} \cdot \frac{g(X_t|X^*)}{g(X^*|X_t)}, 1\right\}$
  If $u \sim \text{Uniform}(0,1) < \alpha$
    Accept: $X_{t+1} \leftarrow X^*$
  End if
  Else
    Reject: $X_{t+1} \leftarrow X_t$
  End else
End for

The correction factor $\frac{g(X_t|X^*)}{g(X^*|X_t)} = 1$ when the proposal distribution is symmetric, such as is the case when $g(x)$ is a normal distribution.

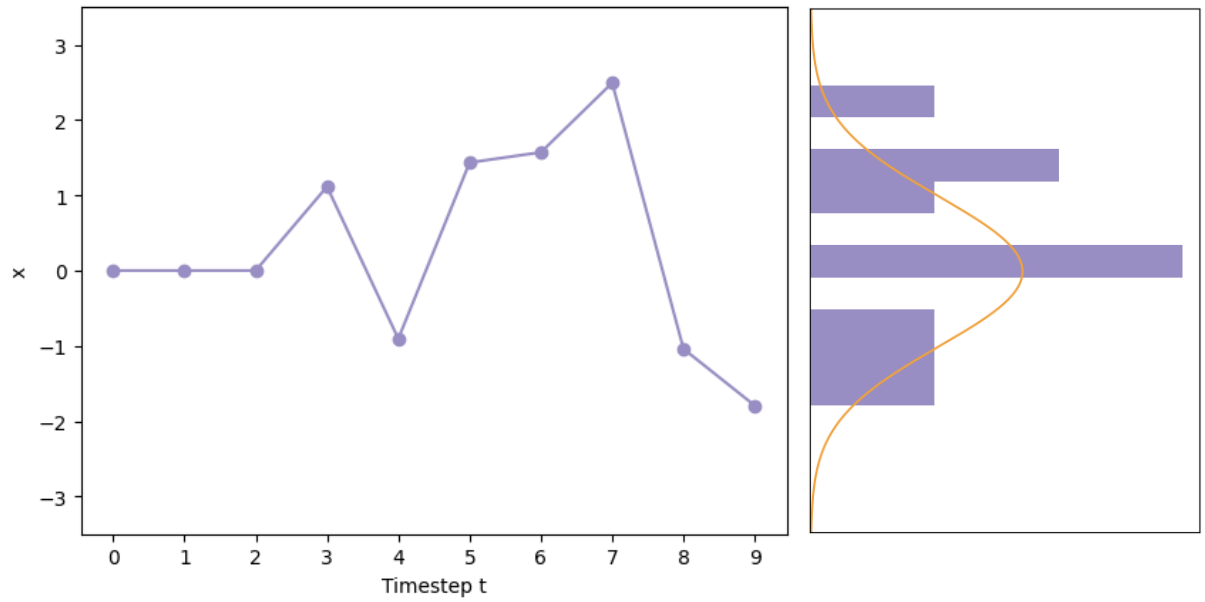


Figure 2: Visualization of 10 timesteps of the Metropolis-Hastings algorithm.

To reiterate, the Metropolis-Hastings algorithm produces inherently random proposals. Hamiltonian Monte Carlo builds on the ideas of Metropolis-Hastings using a physics inspired approach in its proposal process.

1.5 Physical Inspiration. Classical mechanics offers three main approaches to solving for the time-evolution of a system: Newtonian, Lagrangian, and Hamiltonian. While a Hamiltonian approach is not overtly helpful in classical problems, it holds importance in statistical mechanics.

At the core of Hamiltonian mechanics lies Hamilton's equations of motion, two first order differential equations which can describe the motion of an arbitrary particle in 2n-dimensional phase space for states (x, p) given position x: $(x_1, x_2, \dots, x_n)$ and momentum p: $(p_1, p_2, \dots, p_n)$ [5].

$$\frac{dx_i}{dt} = \frac{\partial H}{\partial p_i}$$
$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial x_i}$$

In a conservative or frictionless system, the Hamiltonian H, expressed as a function of position and momentum, is defined as the total energy (kinetic T plus potential U) of the system.

$$H(x, p) = T(p) + U(x) \tag{1.3}$$

$$T(p) = \frac{1}{2} p^T \mathcal{M}^{-1} p \tag{1.4}$$

where $\mathcal{M}^{-1}$ measures the covariance of the posterior distribution [5]. While the "mass matrix" $\mathcal{M}$ can be estimated numerically, this paper will largely ignore this term.

1.6 Hamiltonian Monte Carlo (HMC). Hamiltonian Monte Carlo, also known as hybrid Monte Carlo, is an elegant and powerful MCMC algorithm that applies Hamilton's equations of motion. HMC involves a particle (ball) exploring the posterior distribution (terrain). The ball gains kinetic energy as it approaches high-density areas (valleys) and loses kinetic energy as it approaches low-density areas (hills). Naturally, the valleys are explored more often and hence areas of higher density are sampled more often [5].

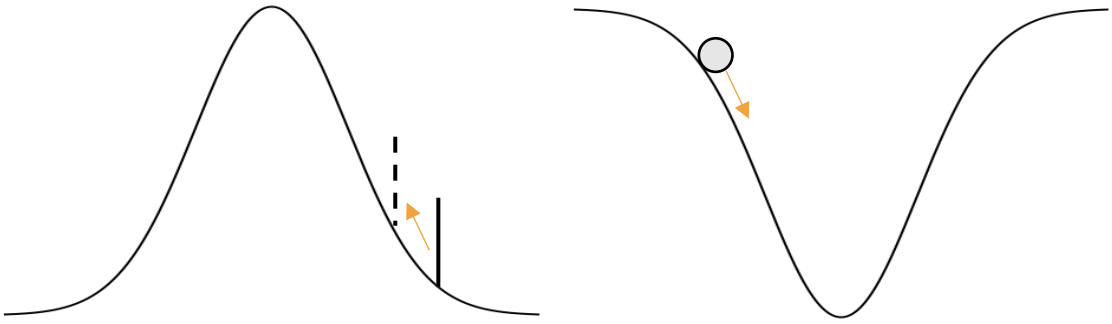


Figure 3: **Left.** Sample proposal in Metropolis-Hastings for a normal distribution visualized. **Right.** Ball gaining kinetic energy as it approaches a high-density area.

However, the Hamiltonian operator and Hamilton's equations of motion in their physical form do no good. The Hamiltonian must be defined probabilistically in relation to the posterior distribution. Assuming a frictionless environment, the Hamiltonian is simply the total energy of the system. Hence, the Hamiltonian can be plugged into the Boltzmann distribution, which describes the probability of a particle existing at a certain energy state given a temperature.

$$P(E) \propto \exp\left(-\frac{E}{k_B T}\right) \Leftrightarrow P(x,p) \propto \exp\left(-\frac{H(x,p)}{k_B T}\right)$$

for temperature T and Boltzmann constant $k_B \approx 1.38 \times 10^{-23}\ \mathrm{J \cdot K^{-1}}$. However, T and k_B can be adjusted at will since HMC runs in a make-believe environment. So, $T = 1$ K and $k_B = 1\ \mathrm{J \cdot K^{-1}}$. Notice that the units lose relevance in HMC and will be omitted for the rest of the paper. By (1.3)

$$P(x,p) \propto \exp\big(-T(p)\big)\exp\big(-U(x)\big) \tag{1.5}$$

Further, it follows from the definition of a joint density $P(x,p) = P(p|x)P(x)$ that the Hamiltonian is defined as

$$H(x,p) = -\ln P(x,p) = -\ln P(p|x) - \ln P(x)$$

$$T(p|x) = -\ln P(p|x) \tag{1.6}$$

$$U(x) = -\ln P(x) \tag{1.7}$$

It is noted that the potential energy $U(x)$ is simply the negative log probability of the target density. The definition of the kinetic energy $T(p)$ varies by implementation [5]. Some implementations may use the physical definition (1.4) while others may use the probabilistic definition (1.6).

Leapfrog integration. Solving Hamiltonian's equations of motion requires some form of integral approximation. Generally, leapfrog integration is used to produce approximate solutions to Hamilton's equations. The simplest way to understand leapfrog integration is to visualize a number line.

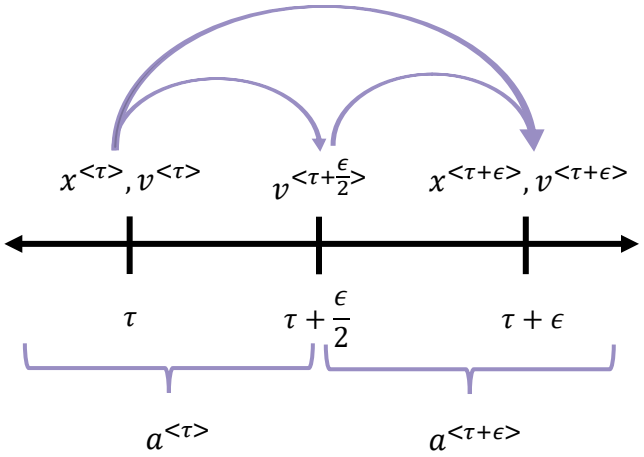


Figure 4: Visualization of leapfrog integration using familiar physics variables.

The velocity $v^{<\tau>}$ is updated half a step to $v^{<\tau+\frac{\epsilon}{2}>}$ by the current acceleration $a^{<\tau>}$. This new velocity $v^{<\tau+\frac{\epsilon}{2}>}$ is used to fully update $x^{<\tau>}$ to $x^{<\tau+\epsilon>}$. Then, the velocity $v^{<\tau+\frac{\epsilon}{2}>}$ is updated half a step to $v^{<\tau+\epsilon>}$ using $a^{<\tau+\epsilon>}$ [6]. Mathematically, one step of updates using the leapfrog integrator is

$$\begin{aligned} v^{<\tau+\frac{\epsilon}{2}>} &= v^{<\tau>} + \frac{\epsilon}{2} a^{<\tau>} \\ x^{<\tau+\epsilon>} &= x^{<\tau>} + \epsilon v^{<\tau+\frac{\epsilon}{2}>} \\ v^{<\tau+\epsilon>} &= v^{<\tau+\frac{\epsilon}{2}>} + \frac{\epsilon}{2} a^{<\tau+\epsilon>} \end{aligned}$$

Note the superscripts represent timesteps in leapfrog integration. In HMC, one step of leapfrog integration with step size ϵ is defined as

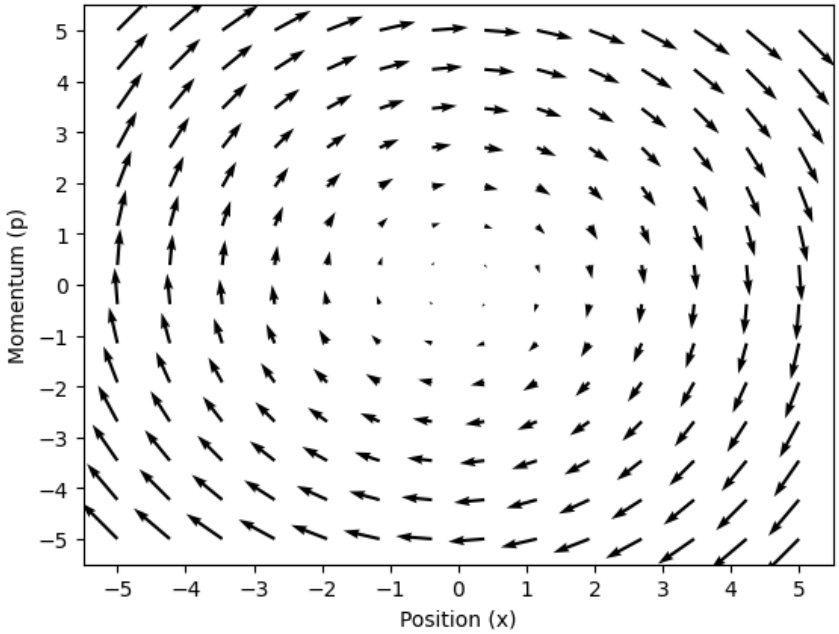
$$\begin{aligned} p^{<\tau+\frac{\epsilon}{2}>} &= p^{<\tau>} - \frac{\epsilon}{2} \nabla_{\mathrm{x}} U(x^{<\tau>}) \\ x^{<\tau+\epsilon>} &= x^{<\tau>} + \epsilon \mathcal{M}^{-1} p^{<\tau+\frac{\epsilon}{2}>} \\ p^{<\tau+\epsilon>} &= p^{<\tau+\frac{\epsilon}{2}>} - \frac{\epsilon}{2} \nabla_{\mathrm{x}} U(x^{<\tau+\epsilon>}) \end{aligned}$$

where $\nabla_{\mathrm{x}} f(k)$ refers to the gradient of $f(x)$ with respect to x evaluated at $x = k$ [5]. More precisely, $\nabla_x f(x) = \left[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \ldots \frac{\partial f}{\partial x_n}\right]^T$. Additionally, $U(x) = -\ln P(x)$ from (1.7). Running the leapfrog integrator for L steps with a step size of ϵ yields a total simulation time of ϵL. The candidate values x^* and p^* can hence be represented as

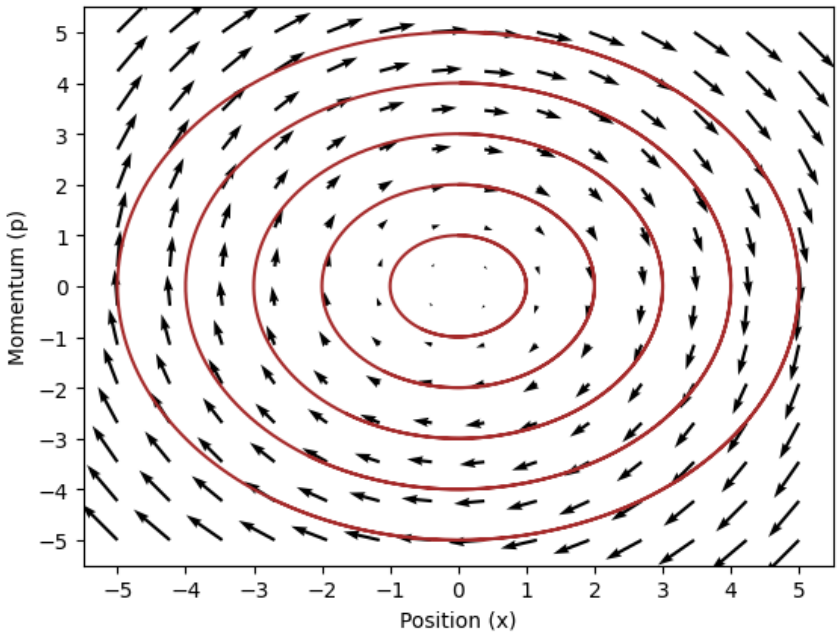
$$x^* = x^{<\epsilon L>}$$

$$p^* = -p^{<\epsilon L>}$$

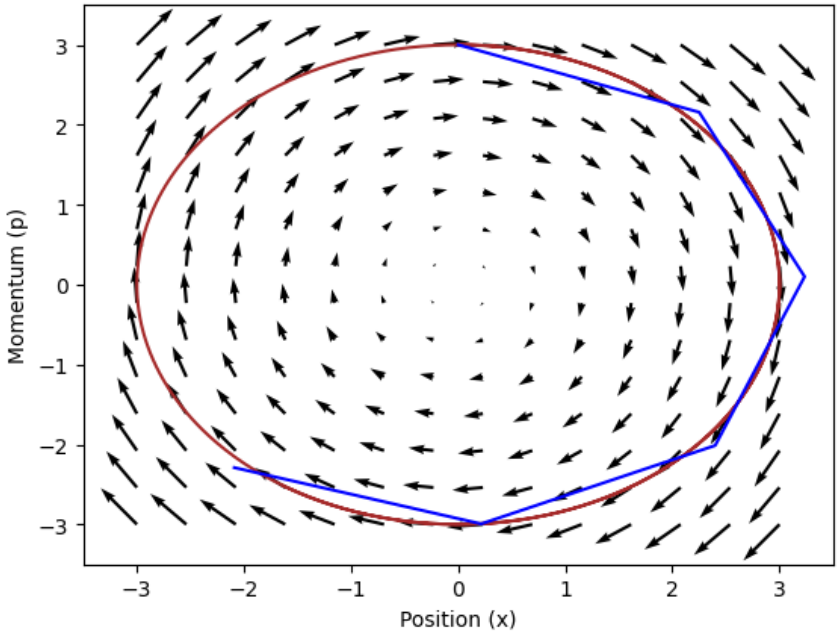
Momentum is negated to satisfy detailed balance. Solving Hamilton's equations yields the new state of the particle. Theoretically, all new states generated by solving Hamilton's equations should preserve the Hamiltonian [5] due to the law of conservation of energy. This means that given an initial position and momentum, the flow in the phase space should have a predetermined curve. In short, the nature of the Hamiltonian creates distinct energy levels that a particle must stay on.



(a) Sample Hamiltonian vector field.



(b) Each ring represents a distinct energy level. Any particle in a frictionless environment must move in a way that conserves its energy.



(c) Demonstration of error from leapfrog integration.

Figure 5: Visualization of 2-dimensional phase space.

However, since the leapfrog integrator is an approximation, error is introduced in calculation of the Hamiltonian. This means that larger ϵ values introduce greater error and consequently lead to varying acceptance probabilities, which are dependent on the Hamiltonian. Hence, the performance of HMC is particularly more sensitive to ϵ. This fact will be important in Section 2.2.

Algorithm 2: Hamiltonian Monte Carlo

Input: Initial position x_0. Number of trials n. Step size ϵ. Number of steps L.

1 **For** t from 0 to $n-1$ **do**
2 Propose momentum: $p^* \sim N(0, \mathcal{M})$
3 Propose position: $x^* \leftarrow x_t$
4 Leapfrog integration for L steps
5 $p^* \leftarrow p^* - \frac{\epsilon}{2}\nabla_x U(x^*)$
6 $x^* \leftarrow x^* + \epsilon \mathcal{M}^{-1} p^*$
7 $p^* \leftarrow p^* - \frac{\epsilon}{2}\nabla_x U(x^*)$
8 End leapfrog integration
9 Reverse momentum: $p^* \leftarrow -p^*$
10 Acceptance probability: $\alpha(x^*, p^*, x_t, p_t) \leftarrow \min\left\{\frac{\exp(-H(x^*,p^*))}{\exp(-H(x_t,p_t))}, 1\right\}$
11 **If** $u \sim \text{Uniform}(0,1) < \alpha$
12 Accept: $x_{t+1} \leftarrow x^*$
13 Accept: $p_{t+1} \leftarrow p^*$
14 **End if**
15 **Else**
16 Reject: $x_{t+1} \leftarrow x_t$
17 Reject: $p_{t+1} \leftarrow p_t$
18 **End else**
19 **End for**

Line 10 of Algorithm 2 follows directly from the Metropolis-Hastings acceptance value and (1.5).

1.7 No-U-Turn Sampler (NUTS). There have been attempts to automatically tune HMC parameters. The most prevalent of which is the No-U-Turn Sampler. Notably, Stan, a programming language for statistical inference, employs NUTS and several other autotune methods in its implementation of HMC. The NUTS algorithm modifies the vanilla HMC algorithm to adaptively set the number of steps L. The mathematical specifics behind NUTS are quite advanced. So, the following is a high-level explanation of the algorithm. First, NUTS implements two particles, a forward particle with state (x^+, p^+) and a backward particle with state (x^-, p^-). Each iteration of the leapfrog integration starts by randomly choosing one of the particles to advance or retreat in time, using positive or negative ϵ respectively. Further, the forward particle can only advance in time while the backward particle can only retreat in time. Each iteration, the number of steps taken doubles. This process continues until the U-turn condition is satisfied [7].

$$p_t^+ \cdot (x_t^+ - x_t^-) < 0 \text{ or } -p_t^- \cdot (x_t^- - x_t^+) < 0 \tag{1.8}$$

Qualitatively, the U-turn condition amounts to continuing to integrate until a momentum vector creates an acute angle with the corresponding displacement vector. Hence, the particle should explore around half of the current energy level [8].

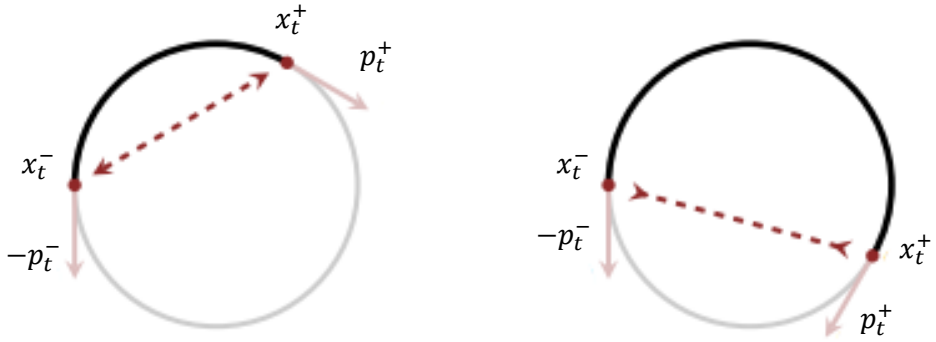


Image credit [8].

Figure 6: **Left.** U-turn condition is not satisfied. **Right.** U-turn condition is satisfied, and leapfrog integration stops.

Once the U-turn condition is met, the next position x_{t+1} of the Markov chain is chosen by randomly sampling from the set of collected leapfrog integration positions. The corresponding momentum can be stored for future reference but is not strictly necessary (as is the case in HMC).

$$x_{t+1} \sim \{x_t^-, \dots, x_t^{<-\epsilon>}, x_t^{<0>}, x_t^{<\epsilon>}, \dots, x_t^+\}$$

The exact method of sampling differs by implementation.

2 Contribution

While NUTS employs a clever idea in an attempt to autotune L, a recent thesis put the validity of the algorithm in question, "[T]he occurrence of divergences implies that the sampler has lost geometric ergodicity…" [9]. In short, NUTS may not preserve ergodicity and therefore cannot always guarantee convergence to a unique stationary distribution. Hence, I present a new neural network approach to autotune the step size ϵ instead of the number of steps L.

2.1 Dense Hamiltonian Monte Carlo (DHMC). DHMC leverages the power of neural networks in tuning ϵ. As the name suggests, the model takes the form of a dense neural network. This architecture was chosen for two main reasons.

1. *Less computational overhead.* A dense neural network is the simplest of neural networks and introduces less trainable parameters than other complex architectures.
2. *Inferential complexity.* Neural networks are known for their powerful ability to adapt to complex problems. This type of adaptability is preferred when dealing with complex distributions.

The model will need to be sufficiently complex without adding unnecessary computational baggage.

Layer	Description	Activation	Parameters
Flatten	Removes input dimensionality		
Dense	Assigns weights to inputs	Sigmoid $\sigma(x) = \frac{1}{1+\exp(-x)}$	1 neuron
Lambda	Applies function λ to input		Function $\lambda(x, s) = x/s$

The lambda layer is akin to a stability layer. The stability constant s limits the range of the neural network output from $[0, 1]$ to $[0, 1/s]$.

Given the model architecture, training requires a loss function. I chose the loss value $\mathcal{L}$ to be the reciprocal of the expected squared jumped distance (ESJD). Loosely, the ESJD is a metric considering both the change in position of the HMC particle and its corresponding acceptance probability.

$$ESJD = \mathbb{E}[\Delta x^T \mathcal{M}^{-1} \Delta x \alpha(x^*, p^*, x_t, p_t)] \tag{2.1}$$

$$\mathcal{L} = 1/ESJD$$

where $\Delta x = x_{t+1} - x_t$ [10]. Effectively, the objective is to maximize the ESJD. The ESJD is a useful metric for this scenario since a good HMC sampler should intrepidly explore the probability distribution without consequently sacrificing the acceptance value.

Algorithm 3: Dense Hamiltonian Monte Carlo (DHMC) Training

Input: Initial position x_0. Number of trials n. Number of steps L. Number of samples m. Number of training epochs T. Neural network $f(x)$. Initial training data collected by running Algorithm 2 until m samples are collected using ϵ_0 and L.

Output: List of saved ϵ^*.

Functions: accept(x, p, α) applies the updates for x and p based on the given acceptance values α. train$(\mathcal{L}, f)$ performs one training step for the given neural network f given loss $\mathcal{L}$ using gradient descent. uniform_choice$\{(x, p) \in (x^*, p^*)\}$ returns one sample from (x^*, p^*) chosen uniformly at random.

For t from $m-1$ to $T+m-2$ **do**
  Define neural network input: $S \leftarrow$ Most recent m states (x, p)
  Propose candidate epsilon: $\epsilon^* \leftarrow f(S)$
  Propose momentum: $p^* \leftarrow m$ values $\sim N(0, \mathcal{M})$
  Propose postiion: $x^* \leftarrow x_t$
  Leapfrog integration for L steps
    $p^* \leftarrow p^* - \frac{\epsilon^*}{2} \nabla_x U(x^*)$
    $x^* \leftarrow x^* + \epsilon^* \mathcal{M}^{-1} p^*$
    $p^* \leftarrow p^* - \frac{\epsilon^*}{2} \nabla_x U(x^*)$
  End leapfrog integration
  Reverse momentum: $p^* \leftarrow -p^*$
  Acceptance probability: $\alpha(x^*, p^*, x_t, p_t) \leftarrow \frac{\exp(-H(x^*, p^*))}{\exp(-H(x_t, p_t))}$
  Accept or reject proposals: $(x^*, p^*) \leftarrow \text{accept}(x^*, p^*, \alpha)$
  Calculate loss: $\mathcal{L} \leftarrow 1/ESJD$
  Model trainstep: $f \leftarrow \text{train}(\mathcal{L}, f)$
  Sample new state: $(x_{t+1}, p_{t+1}) \leftarrow$ uniform_choice$\{(x, p) \in (x^*, p^*)\}$
  Save ϵ^*
End for

The outputted list of ϵ^* can be averaged to find the tuned ϵ' value. Then Algorithm 2 can be run with ϵ' and L.

2.2 Notes. After glancing at the algorithm, the reader may have several reasonable questions regarding DHMC's usefulness and procedure. The following is a brief explanation of the parameters.

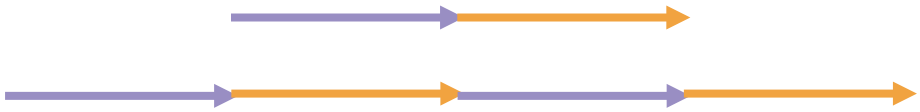
Regarding number of epochs T. As with any neural network, too high T will overtrain the model while too low T will result in frustratingly poor performance. Setting T will be addressed further in Section 4.1.

Regarding number of samples m. m defines the number of initial training samples and the number of samples fed into the neural network. m number of momentum values are drawn from a normal distribution, which represent m different simulations with the same initial position. m should be proportional to the dimensionality of the distribution. As the empirical results show, $m = 50$ is sufficient for 1 and 2-dimensional distributions.

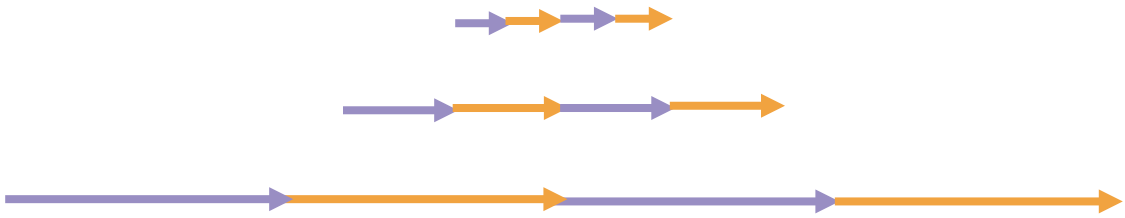
Regarding stability constant s. Some distributions are more sensitive to ϵ than others. In these cases, s should be large. Also, having too wide of a range of ϵ values may slow down the training process due to low acceptance rates. Setting s will be addressed further in Section 4.1.

Regarding ϵ_0. Collection of initial training data requires the user to set an initial ϵ value. As long as ϵ yields reasonable acceptance values, the initial choice of ϵ does not matter. After all, it is the neural network's job to tune this value. Setting ϵ_0 will be addressed further in Section 4.1.

Regarding L. The user must still set this value. As was stated in Section 1.6, the HMC sampler is not particularly affected by small adjustments in L. In fact, having large L will simply result in cyclical behavior (see Figure 5), while small L can be fine-tuned better by adjusting ϵ. To visualize this, consider a line of length ϵL of variable size.



(a) With ϵ constant, simulation length can only be multiples of ϵ.



(b) With ϵ variable, simulation length can cover a wider range of values.

Figure 7: Possible simulation times ϵL holding ϵ constant (top) or L constant (bottom).

3 The Results

DHMC was tested on two probability distributions: the standard normal distribution and the Rosenbrock distribution. DHMC's performance was compared with NUTS. For both distributions, the samplers were stopped before convergence to demonstrate relative rates of convergence. Also, all tests were run using the Python 3 TensorFlow Probability library for consistency and compatibility with the TensorFlow DHMC model. No code optimization was used.

3.1 Standard normal distribution. The probability density function for a normal distribution can be defined as

$$P(x|\mu,\sigma) \propto \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

The normalization constant is $\frac{1}{\sigma\sqrt{2\pi}}$ but will be omitted for demonstration purposes. The standard normal distribution can be defined as

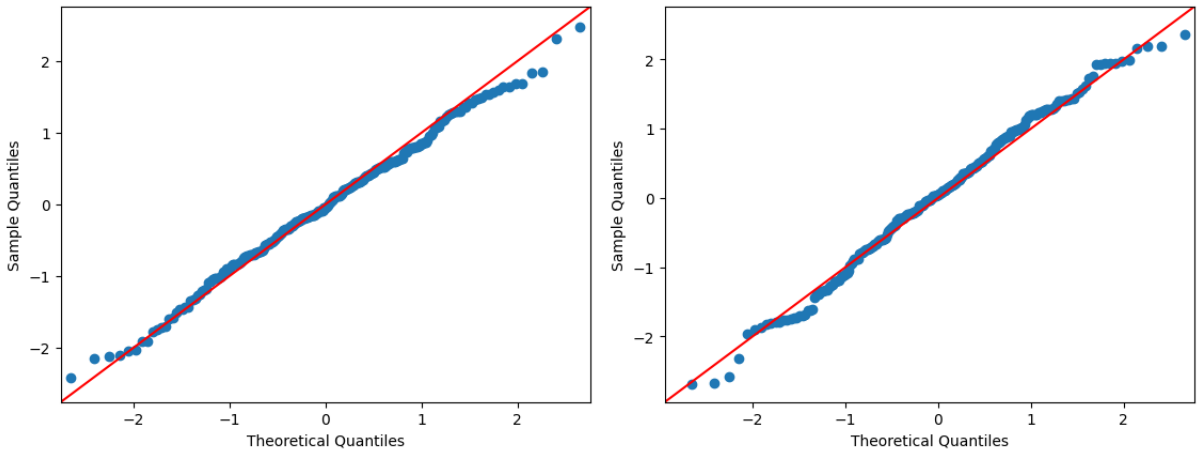
$$P(x|0,1) \propto \exp\left(-\frac{1}{2}x^2\right)$$

The execution time of DHMC and NUTS was compared to the baseline time of vanilla HMC by averaging execution time of each algorithm across 10 trials.

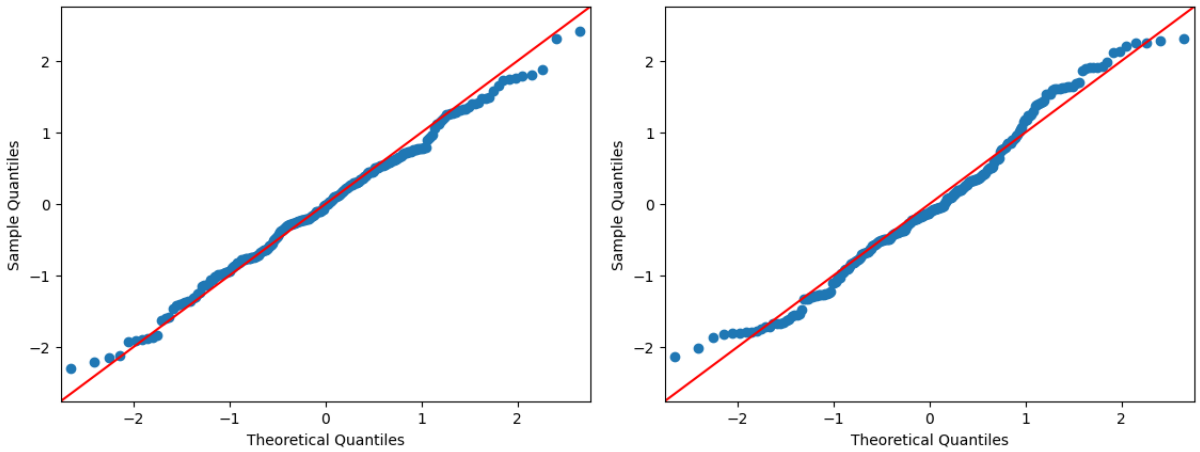
	HMC	**DHMC**	**NUTS**
Iterations	250	250 + 50 data collection + 50 warm-up steps	250
Parameters	$\epsilon = 0.25$ $L = 10$	$\epsilon_0 = 0.25$ $L = 10$ $s = 1$ $m = 50$ $T = 50$	$\epsilon = 0.25$ $L_{unr} = 1$ $TD_{max} = 10$ $\Delta E_{max} = 1000$
Execution time	22.4s (+0%)	47.4s (+112.0%)	133.3s (+495.6%)

Table 1: Comparison of execution time for 250 samples. Default TensorFlow Probability values were used for number of unrolled leapfrog steps L_{unr}, maximum tree depth TD_{max}, and maximum energy difference ΔE_{max}.

Clearly, additional execution time for DHMC stems from the collection of initial training data and training of the model. Even then, DHMC's execution time pales in comparison to NUTS. Perhaps more important than the execution time is the quality of samples. A QQ plot was drawn for samples drawn by DHMC and NUTS. Note that samples by vanilla HMC cannot be used as a baseline since the optimal epsilon is unknown.



(a) QQ plot of samples collected by DHMC.



(b) QQ plot of samples collected by NUTS.

Figure 8: Comparison of samples collected by DHMC and NUTS.

Both samplers collect values in the range [-2, 2], although NUTS tends to go slightly beyond this range. Additionally, both set of samples are reasonable. As the results from the Rosenbrock distribution will support, NUTS tends to sacrifice execution time for exploring more of the distribution. This fact follows from the U-turn condition (1.8), which forces the particle to explore around half of its initial energy level.

3.2 Rosenbrock distribution. A 2-dimensional Rosenbrock distribution [11] is defined as

$$P(x_1, x_2) \propto \exp\left(-\frac{100(x_2 - x_1^2)^2 + (1 - x_1)^2}{20}\right)$$

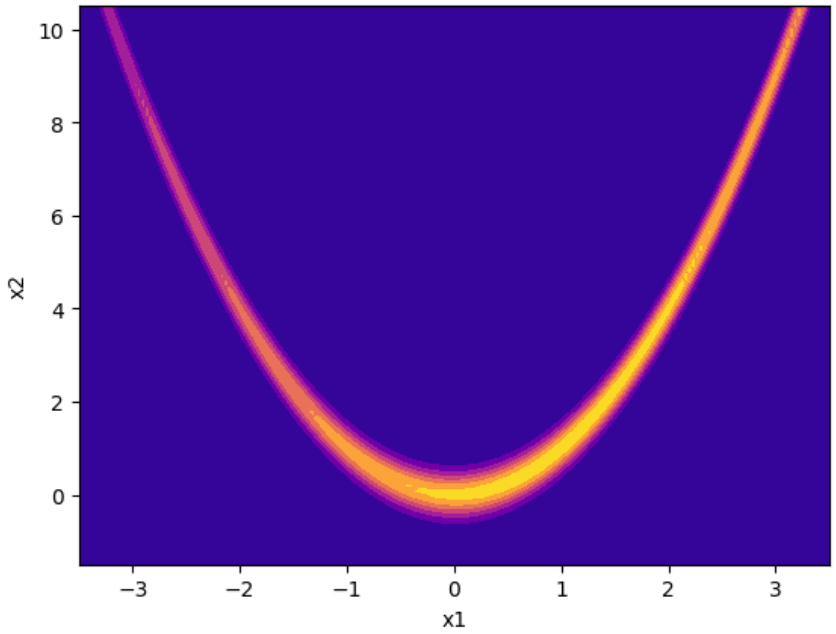


Figure 9: Contour plot of the Rosenbrock distribution.

	HMC	**DHMC**	**NUTS**
Iterations	250	250 + 50 data collection + 50 warm-up steps	250
Parameters	$\epsilon = 0.05$ $L = 10$	$\epsilon_0 = 0.05$ $L = 20$ $s = 8$ $m = 50$ $T = 50$	$\epsilon = 0.05$ $L_{unr} = 1$ $TD_{max} = 10$ $\Delta E_{max} = 1000$
Execution time	89.0s (+0%)	150.9s (+69.5%)	1,025.4s (+1,052.1%)

Table 2: Comparison of execution time for 250 samples. Default TensorFlow Probability values were used for number of unrolled leapfrog steps L_{unr}, maximum tree depth TD_{max}, and maximum energy difference ΔE_{max}.

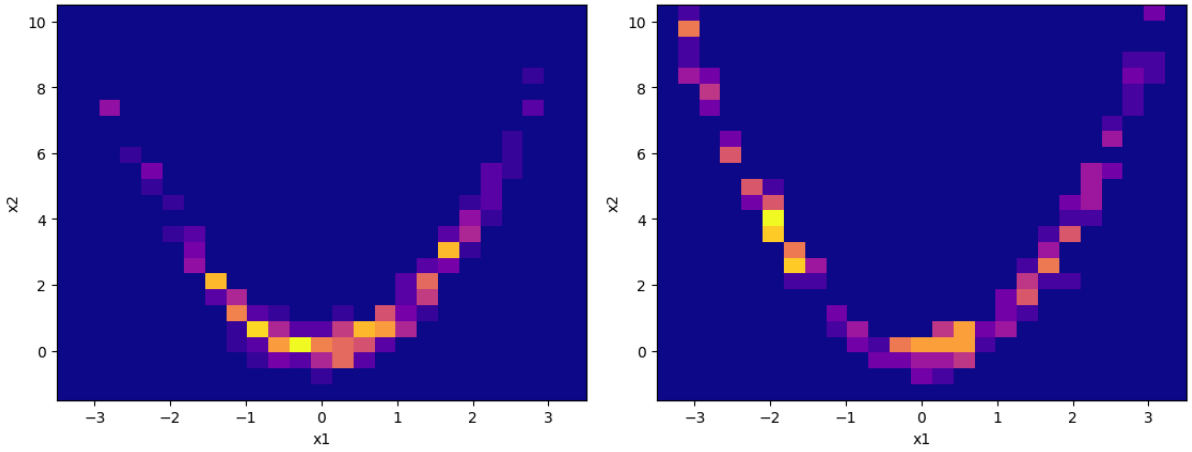


Figure 10: **Left.** 2D histogram of samples collected by DHMC. **Right.** 2D histogram of samples collected by NUTS. 44 data points omitted with $x_2 > 10.5$.

The results show that DHMC's execution time grows at a slower rate than HMC's execution time with respect to distribution dimensionality. Additionally, the execution time of NUTS increases at a faster rate compared to DHMC. To reiterate, NUTS tends to sacrifice execution time for more exploration. In contrast, DHMC seems to stay around high-density areas in exchange for faster execution time. This is not a problem since multiple DHMC chains can be run simultaneously with different starting positions.

4 Discussion

Evidently, DHMC produces competitive results compared to NUTS while showing significant time savings. Even so, DHMC is not a perfect algorithm and has several areas for potential future improvement.

4.1 Future work.

Autotuning ϵ_0, s, and T. Future versions of DHMC can autotune the initial epsilon ϵ_0, stability constant s, and the number of epochs T. During Algorithm 3, $s = 1, \epsilon_0 = 1$ to begin and s can be doubled each iteration if the acceptance value falls below a threshold. Simultaneously, ϵ_0 can be halved for each doubling of s. Additionally, the training can automatically stop when the ESJD metric (2.1) converges, possibly decreasing training time and eliminating the need to set T.

Distribution dimensionality. DHMC's effectiveness should be tested on higher-dimensional distributions. DHMC's performance with respect to higher-dimensional distributions would be valuable information. One can guess that DHMC will hold its performance thanks to the effectiveness of neural networks even when working with higher-dimensional data.

Sensitivity to ϵ. DHMC requires a range of ϵ values that it can test during training since training is inherently stochastic. Even with stability constant s, certain distributions may be severely asymmetrical and be more sensitive to ϵ in different areas of the distribution. The ideal solution would be a machine learning powered HMC variation that adapts parameters on the fly. Such an

algorithm would face the challenge of ensuring unbiased Markov chain samples. Any adjustment must not depend on previous history. Else, the chain would dissatisfy the Markov property (1.1).

4.2 Application domains. The possible applications of DHMC are bountiful. DHMC can be applied wherever Metropolis-Hastings, HMC, or NUTS can be used. Application domains include but are not limited to estimating dynamic stochastic general equilibrium (DSGE) models for economic forecasting [12], estimating parameters of solar coronal heating models [13], optimizing snowmelt runoff models in the Himalayas [14], modeling crop response to various nutrients [15], analyzing CT scans [16], and solving seismic tomography problems [17].

5 Conclusion

This paper explored the performance of DHMC, my proposed method of autotuning ϵ in Hamiltonian Monte Carlo, in comparison to the No-U-Turn Sampler. While future works can improve DHMC and test its performance on different fields, the preliminary results are promising. DHMC has potential to be a faster option than NUTS while yielding competitive results. DHMC's benefits are clear and its effects significant. DHMC can provide insight on dealing with high-dimensional distributions and allow for more powerful approaches to countless statistical problems. With the unspeakable importance of data and its corresponding statistical analysis in today's society, I am confident that DHMC can serve an important role in a variety of questions in modern academia.

6 References

[1] Manning, C. D., Raghavan, P., & Schütze, H. (2008). *An introduction to information retrieval* [PDF]. Cambridge University Press. https://nlp.stanford.edu/IR-book/pdf/irbookonlinereading.pdf

[2] Chang, J. (2007, February 2). *Stochastic processes* [Unpublished manuscript]. http://www.stat.yale.edu/~pollard/Courses/251.spring09/Handouts/Chang-notes.pdf

[3] Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian data analysis* (3rd ed.). CRC Press.

[4] Robert, C. P. (2016, January 27). *The Metropolis–Hastings algorithm*. https://doi.org/10.48550/arXiv.1504.01896

[5] Neal, R. M. (2011). MCMC using Hamiltonian dynamics. In S. Brooks, A. Gelman, G. Jones, & X.-L. Meng (Eds.), *Handbook of Markov Chain Monte Carlo* (pp. 113-162). Chapman and Hall/CRC Press. https://doi.org/10.1201/b10905

[6] Lindahl, E. (2021, February 2). *Lecture 06, concept 09: Integration using the leap-frog algorithm updates x & v* [Video]. YouTube. https://www.youtube.com/watch?v=pVudb6-_FaM

[7] Hoffman, M. D., & Gelman, A. (2014). The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo [PDF]. *Journal of Machine Learning Research*, *15*, 1351-1381. https://www.jmlr.org/papers/volume15/hoffman14a/hoffman14a.pdf

[8] Betancourt, M. (2018, July 16). *A conceptual introduction to Hamiltonian Monte Carlo*. arXiv. https://arxiv.org/pdf/1701.02434.pdf

[9] Dombowsky, A. (2020). *Assessing the quality of posterior samples from No-U-Turn Hamiltonian Monte Carlo* [Unpublished master's thesis]. McGill University.

[10] Pasarica, C., & Gelman, A. (2010). Adaptively scaling the Metropolis Algorithm using expected squared jumped distance [PDF]. *Statistica Sinica*, *20*, 343-364. https://www3.stat.sinica.edu.tw/statistica/oldpdf/A20n113.pdf

[11] Pagani, F., Wiegand, M., & Nadarajah, S. (2020, May 7). *An n-dimensional Rosenbrock distribution for MCMC testing*. https://doi.org/10.48550/arXiv.1903.09556

[12] Cai, M., Del Negro, M., Herbst, E., Matlin, E., Sarfati, R., & Schorfheide, F. (2019, August 21). Online estimation of DSGE models. *Liberty Street Economics*. https://libertystreeteconomics.newyorkfed.org/2019/08/online-estimation-of-dsge-models/

[13] Adamakis, S., Morton-Jones, T. J., & Walsh, R. W. (2006). Applications of Metropolis-Hastings algorithm to solar astrophysical data. *Statistical Challenges in Modern Astronomy*, *371*, 401-402. https://articles.adsabs.harvard.edu//full/2007ASPC..371..401A/0000401.000.html

[14] Panday, P. K., Williams, C. A., Frey, K. E., & Brown, M. E. (2014). Application and evaluation of a snowmelt runoff model in the Tamor River basin, Eastern Himalaya using a Markov chain Monte Carlo (MCMC) data assimilation approach. *Hydrological Processes*, *28*(21), 5337-5353. https://doi.org/10.1002/hyp.10005

[15] Ng'ombe, J. N., & Lambert, D. M. (2021). Using Hamiltonian Monte Carlo via Stan to estimate crop input response functions with stochastic plateaus [PDF]. *Journal of Agriculture and Food Research*, *6*. https://doi.org/10.1016/j.jafr.2021.100226

[16] Kasai, R., & Yamada, K. (2017). Computed tomography(CT)で生じる金属アーチファクト定量評価へのハミルトニアンモンテカルロ法の適用 [Application of Hamiltonian Monte Carlo method to metal artifact quantitative evaluation in computed tomography (CT)]. *Japanese Journal of Radiological Technology*, *73*(8), 654-663. https://doi.org/10.6009/jjrt.2017_JSRT_73.8.654

[17] Fichtner, A., Zunino, A., & Gebraad, L. (2019). Hamiltonian Monte Carlo solution of tomographic inverse problems. *Geophysical Journal International*, *216*(2), 1344-1363. https://doi.org/10.1093/gji/ggy496