A Survey about Bayesian Inference Power in Math and Machine Learning Perspectives

TIANLONG WANG*,TOM STOJSAVLJEVIC*, AND MEHMET DIK**
*BELOIT COLLEGE, BELOIT, WI, UNITED STATES.
**ROCKFORD UNIVERSITY, ROCKFORD, IL UNITED STATES.

ABSTRACT In the realm of Machine learning, Deep Neural Networks (DNN) alongside related architectures like Convolution Neural Networks (CNNs) and more complex models have established themselves as robust function approximators due to their extensive parameterization power. However, this flexibility is extremely vulnerable to overfitting problems. Moreover, the traditional deep neural network conceals the inner workings and decision-making mechanisms employed to produce the final result and outputs a single fixed value for point estimation without accounting for uncertainty. To address these challenges, Bayesian Neural Networks (BNNs) have emerged as a robust framework for uncertainty analysis and overfitting problems. This survey paper aims to examine the evolutional history of BNNs, focusing on the branch that starts with Markov Chain Monte Carlo (MCMC). The framework integrates MCMC, and Hamiltonian Monte Carlo (HMC) methods, with the more recent No-U-Turn Sampler (NUTS) and in-depth mathematical explanations are provided. Additionally, empirical implementations of this integration are presented with the most updated MCMC methods to bridge mathematical theory and practical application.

Key Words: Bayesian Inference, Machine learning, Bayesian Neural Network, Probabilistic Machine Learning

1. INTRODUCTION

Interdisciplinary teams work together through decades to design , build and optimize the Bayesian Neural Network , and create a rich literature for research and application (Jospin et al., 2022) . This paper aims at investigate the evolving history of Bayesian Neural Network (BNN) and it's founding parts with a focus on the approach based on the Markov Chain Monte Carlo using Metroplois-Hasting Algorithm (Hastings, 1970) and later switched to Hamiltonian Monte Carlo that was generalized in statistics by Neal (Neal, 2012) for efficiency and optimization.

In short, it aimed at solving the intractable marginal distribution $p_\theta(x)$ base on Bayesian Theorem (Eq. 1), where $p_\theta(z \mid x)$ is the posterior of the function, $p_\theta(x \mid z)$ is the likelihood, $p_\theta(z)$ is the prior, $p_\theta(x)$ is the Evidence, also known as marginal likelihood. $p_\theta(x \mid z)p_\theta(z)$ also know as joint probability, it's also where most of the simulation and inference about Bayesian Theorem comes from.

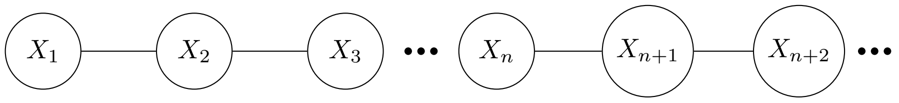
$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(x)} \tag{1}$$

The marginal likelihood $p_\theta(x)$ is mathematically intractable due to high dimensional integration as the number of parameter and classification criteria increase (Neal, 2012):

$$\begin{aligned} p_\theta(z \mid x) &= \frac{p_\theta(x \mid z)p_\theta(z)}{\int p_\theta(x, z')dz'} \\ &= \frac{p_\theta(x \mid z)p_\theta(z)}{\int \int \cdots \int p(x, z')dz_1 dz_2 \cdots dz_{x-1}} \end{aligned} \tag{2}$$

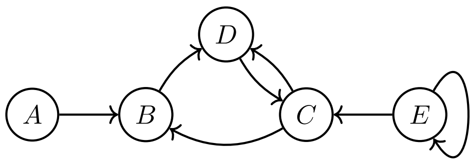
2. MARKOV CHAIN MONTE CARLO SIMULATION

2.1 Markov Chain The first time Markov Chain idea is proposed can trace all the way back to 1906, by Andrei Andreevich Markov (Markov, 2006). Markov Chain aims at drawing simulations closer and closer to the target distribution, usually the posterior in Bayesian Theorem that is not calculable due to the intractable marginal likelihood. It starts with a random state and all other states are dependent on the previous one. X_2 depends on X_1, X_3 depends on X_2, so on. In between each sample, there are transitional probabilities $T(x_2 \mid x_1)$,$T(x_3 \mid x_2)$ ... connects them and guide the growing of the Markov Chain to make sure arriving at the target distribution and stay on the distribution afterwards.



Once the Markov Chain reaches the target distribution, also known as the steady state, any states after will still stay on the current probability (Markov, 2006). The states before reaching steady state are called "burn-in" states as X_1,X_2,X_3 ... in the graph, and after "burn-in" states, states X_n, X_{n+1}, X_{n+2} stays on the target distribution.

2.2 Markov Chain Stationary Distribution



This example explain what is the steady state, or stationary distribution of a Markov Chain. In the graph above, $A \rightarrow B$ indicates a particle moves from State A to State B, etc. The Transitional table indicate where the particle will move into next step base on where the particle is at current step(Table 1), State A has a probability of 1 move into state B, state C has a probability of 0.5 to move into B, together with a probability of 0.5 to move into state D, etc.:

	To: A	To: B	To: C	To: D	To: E
From: A	0	1	0	0	0
From: B	0	0	0	1	0
From: C	0	0.5	0	0.5	0
From: D	0	0	1	0	0
From: E	0	0	0.1	0	0.9

Table 1: Transitional Table between States, columns indicating the start states, rows are the destination states. the number indicating what's the probability of moving from a state in columns to a state in the rows.

let π indicate the vector of probability for where the particle will stop after reaching the steady state, and $\pi = [\pi_A, \pi_B, \pi_C, \pi_D, \pi_E]$ indicate the probability of each state that the particle will eventually stay on. Each step of the change will be guided by the transitional table from Table 1, for each single node, a probability vector can be formed as following:

$$\begin{aligned}
\pi_A &= 0 \\
\pi_B &= \pi_A + 0.5\pi_C \\
\pi_C &= \pi_D + 0.1\pi_E \\
\pi_D &= \pi_B + 0.5\pi_C \\
\pi_E &= 0.9\pi_E
\end{aligned}$$

Base on the transitional table, there is no node goes into node A, so π_A will be 0. The probability of moving into B from A is 1, and the probability of moving from C to B is 0.5, so $\pi_B = \pi_A + 0.5\pi_C$, etc. Besides, there is also a hidden equation for the summation of all node's probability:

$$\pi_A + \pi_B + \pi_C + \pi_D + \pi_E = 1$$

solving the equations composition results in the steady state of the π distribution as:

$$[\pi_A, \pi_B, \pi_C, \pi_D, \pi_E] = \left[0, \frac{1}{5}, \frac{2}{5}, \frac{2}{5}, 0\right]$$

The result probability vector indicates that after reaching steady state of the Markov Chain, the particle has a 0 possibility stay on node A and E, 1/5 possibility stays on node B, and a 2/5 possibility stays on node C and D, and those possibility for each nodes in should not change after the chain reach the steady state.

2.3 Detailed Balance Once the chain arrived the stationary distribution, detailed balance (or reversibility condition) can be applied on any two connected states in the Markov Chain (Markov, 2006), the following detailed balance equations are guided by O'Hagan's book (O'Hagan, 2010). $p(x)$,$p(y)$ indicates the states of nodes, $T(y \mid x)$ indicates the transitional probability of nodes from state in x to state in y:

$$p(x)T(y \mid x) = p(y)T(x \mid y) \tag{3}$$

Recall the Bayesian Therom (Eq. 2), left hand side is the target posterior, numerator is the joint likelihood that MCMC simulate on, denominator serves as a normalizing constant. So, Eq. 2 can be re-write into:

$$p(x) = \frac{f(x)}{NC} \tag{4}$$

Switch p(x),p(y) from Eq.3 with Eq.4 can re-write Bayesian Therom into a function used later in Metropolis-Hasting Algorithm:

$$\frac{f(x)}{NC}T(y \mid x) = \frac{f(y)}{NC}T(x \mid y) \tag{5}$$

2.4 Metropolis-Hasting Monte Carlo Monte Carlo is proposed at 1940 by Neumann and Ulam (**empty citation**) that use Markov Chain (Markov, 2006) for statistical inference. One of the basic methods for MCMC to determine the transition probability after the Markov Chain reaching the Detailed Balance steady distribution state is Metropolis-Hasting Algorithm, which was first proposed as a symmetric proposal algorithm by Metropolis (Metropolis et al., 1953), later extended it to a more general case with asymmetric proposing approach by Hasting (Hastings, 1970) and formed Metropolis-Hasting Algorithm til now.

In this algorithm, a new candidate is proposed by sampling methods $g(y \mid x)$ from any symmetric or asymmetric distributions.

$$g(x_{t+1} \mid x_t) = N(x_t, \sigma^2)$$

After a new candidate has been proposed, an acceptance rate $A(x \rightarrow y)$ will decide if the candidate would be accepted or not, so the Transitional Probability in Eq. 5 can be re-written with the proposal distribution and the acceptance rate:

$$\frac{f(x)}{NC}g(y \mid x)A(x \rightarrow y) = \frac{f(y)}{NC}g(x \mid y)A(y \rightarrow x) \tag{6}$$

And a focus on acceptance rate after some rearrangement:

$$\frac{A(x \rightarrow y)}{A(y \rightarrow x)} = \frac{f(y) \cdot g(x \mid y)}{f(x) \cdot g(y \mid x)} \tag{7}$$

The following hypothesis case is articulated by Chib and Greenberg for better intuition about the equations manipulation (Chib & Greenberg, 1995). To calculate $A(x \to y)$, a simulated case for convenience can be constructed like the following:

$$f(x) \cdot g(y \mid x) > f(y) \cdot g(x \mid y)$$

The inequity is constructed to describe that the new state y is accepted too often, and moving from y back to x is too rarely. Thus, in Metropolis-hasting algorithm, the acceptance rate for moving from y to x should be set to maximum as 1 to balance the inequality in this simulated case. And if the inequality is reversed, $g(y \mid x)$ will be set to 1 for this case. So Eq. 7 will be:

$$A(x \to y) = \frac{f(y) \cdot g(x \mid y)}{f(x) \cdot g(y \mid x)} \tag{8}$$

For convenience, define $\frac{f(y)}{f(x)}$ as rf (ratio of f), $\frac{g(x \mid y)}{g(y \mid x)}$ as rg (ratio of g), together with the fact that the probability of accepting a move can not be over one, the acceptance probability can be described as:

$$A(x \to y) = min(1, rfrg)$$

Here is a pseudo code version for Metropolis-Hasting Algorithm:

Algorithm 1 Basic Metropolis-Hasting Algorithm

```
$x_0 = [0, 0, ..., 0]$
while $t < N$ do
    $u \sim U(0, 1)$
    $x_{t+1} \sim g(x_t, \sigma^2)$
    if $u \leq A(x_{t+1} \mid x_t) = min(1, rfrg)$ then
        $x_{t+1} = x_{t+1}$
    else
        $x_{t+1} = x_t$
    end if
end while
```

2.5 Empirical Explanation for Metropolis-Hasting algorithm In between each state in MCMC, define θ which is the collection of weights in the Neural network as $\theta = \{\theta_1, \theta_2, ..., \theta_n\}$ as the current state, $\theta^* = \{\theta_1^*, \theta_2^*, ..., \theta_n^*\}$ as the next proposed state for all the weights. Then the joint probabilty f(y) is influenced by the proposed states θ^*, and f(x) is influenced by the current states θ. Besides that, if the candidate proposal function is symmetric $g(x \mid y) = g(y \mid x)$, which means if y is proposed from x, x can also be proposed from y, then the acceptance rate from Eq. 8 can be simplified into:

$$\alpha = min(1, \frac{f(y)}{f(x)}) \tag{9}$$

Since $f(y)$ is a probability density functions influenced by θ, if θ^* leads to a better results, then $f(y)$ will increase to a denser area than $f(x)$, and decrease vice versa:

$$\begin{cases} rf > 1 & \text{if } \theta^* \text{ is better} \\ rf < 1 & \text{if } \theta \text{ is better} \end{cases}$$

Plug into Eq. 9, acceptance rate can be evaluated as:

$$\alpha = \begin{cases} 1 & \text{if } \theta^* \text{ is better} \\ rf & \text{if } \theta \text{ is better} \end{cases} \tag{10}$$

The first piece from Eq. 10 can be interpreted as if θ^* is better, the acceptance rate for the new proposed state is 100%. Recall Eq.4, $p(x)$ is proportional to $f(x)$ up to a normalizing constant. With a better θ^*, the joint probability $f(x)$ will increase. The observation as well as the proportaional relationship indicates that $p(\theta^*)$ is also increasing to a higher probability (Figure. 1). In this way, the proposal acceptance architecture is guided by the true posterior $p(x)$ even though it's unknown.

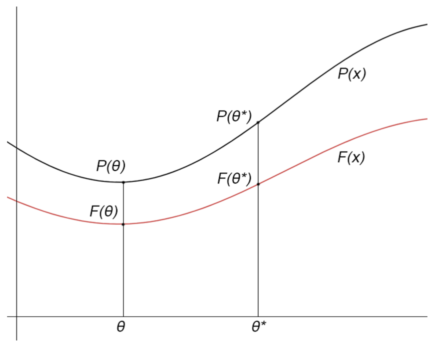


Figure 1: P(x) is proportional to F(x), θ^* leads to a higher value on F(x), so it also leads to a higher value for P(x).

The second piece from Eq. 10 means if the new proposed θ^* is actually worse than the current state, the algorithm might accept or refuse the proposal based on the noise $u \sim U(0,1)$. Nevertheless, the closer rf to 1, the higher the chance new proposal will be accepted. Thus, in both case, all new states in the MCMC chain are directed by $p(x)$ that is not calculable directly.

3. HAMILTONIAN MONTE CARLO SIMULATION

Hamiltonian Monte Carlo Simulation is orginal proposed in 1987 by Duane et al called Hybrid Monte Carlo, which is commonly called Hamiltonian Monte Carlo now, for evaluating lattice quantum chromodynamics (Duane et al., 1987). In 1996, in Neal's Ph.D. thesis, he generalized the Hybrid Monte Carlo (Neal, 2012) for statistical inference as well. It is a special and effect version of Metropolis-Hasting Algorithm that aim at improve the effiency of candidate proposal process for Markov Chain.

3.1 Hamiltonian Dynamics In Hamiltonian Dynamics (**empty citation**), Imagine a particle moving on a 2D dimensional surface,and the particle have a position vector x, and a momentum vector p. Then, the potential energy is defined as $U(x)$, and he kinetic energy is defined as $K(p)$. The total energy of the system is:

$$H(x,p) = U(x) + K(p) \tag{11}$$

Hamiltonian dynamics describe how kinetic energy is converted to potential energy (and vice versa) as an object moves throughout a system in time via a set of partial derivative equations known as Hamiltonian equation.

$$\frac{dx}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K(p)}{\partial p} \tag{12}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial x} = -\frac{\partial U(x)}{\partial x} \tag{13}$$

3.2 LeapFrog Method LeapFrog (Neal, 2012) is used to simulate the movement of the imaginary particle movement from t to $t+T$. For implementation of the calculation, time T is discreted into smaller intervals δ. Within onestep of the leapfrog method, momentum is updated for the first half of the time interval $\delta/2$ base on the partial derivative of potential energy from Eq. 12, and update the potential energy based one the updated momentum, then update the momentum for the next round of Leap Frog method:

$$\begin{aligned} p(t+\delta/2) &= p(t) - (\delta/2)\frac{\partial U}{\partial x(t)} \\ x(t+\delta) &= x(t) + \delta\frac{\partial K(p)}{\partial p(t+\delta/2)} \\ p(t+\delta) &= p(t+\delta/2) - (\delta/2)\frac{\partial U}{\partial x(t+\delta)} \end{aligned} \tag{14}$$

The implementation of leapfrog method will first draw a random momentum from certain distribution, and then approximate the momentum of the system base on the potential energy and update the position base on that kinetic energy from the momentum, then update the momentum again after the movement considering the new position and velocity of the particle, which will also benefit the next move by providing a more accurate momentum. At the end of each turn, the momentum is reversed for symmetric proposal to secure the Hasting term.

Beside the choice of elapsed time δ, another parameter L, how many round of leap frog method would be executed also need manually set up. Later section will introduce new methods that auto tuning those two parameters.

3.3 Hamiltonian Monte Carlo HMC starts with building a basic energy function $E(\theta)$ (Duane et al., 1987), the following equation manipulations are guided by Neal's Paper (Neal, 2012):

$$E(\theta) = H(x,p) = U(x) + K(p) \tag{15}$$

and the canonical distribution of this energy function has PDF as:

$$q(\theta) = \frac{1}{Z} e^{-E(\theta)} \tag{16}$$

Z term is a normalizing term and can be neglected. Substitue the energy function and explain paramter θ as x and p we can get:

$$\begin{aligned} q(x,p) &\propto e^{-H(x,p)} \\ &= e^{-[U(x)+K(p)]} \\ &= e^{-U(x)} + e^{-K(p)} \\ &\propto q(x)q(p) \end{aligned} \tag{17}$$

The result indicates that the energy distribution $q(x,p)$ is proportional to $e^{-[U(x)+K(p)]}$, let x^* and p^* to denote the position and momentum of new proposed candidate respectively, new canonical distribution is going to be:

$$q(x^*,p^*) \propto e^{-[U(x^*)+K(p^*)]} \tag{18}$$

The an acceptance probability is similar to Metropolis-Hasting Algorithm by dividing $q(x^*,p^*)$ and $q(x,p)$ and then compare to 1 :

$$A = min(1, e^{-U(x^*)-K(p^*)+U(x)+K(p)}) \tag{19}$$

A general Hamiltonian Monte Carlo Algorithm described in Algrithm 2:

3.4 Intuition behind HMC In Hamiltonian Equation, x represent the position of a imaginary particle, U(x) is the potential energy. Drawing connection to Bayesian neural Network, x refers to the variables of interests, usually the set of parameters for the model, weights, bias, etc. And U(x) is the density function for what we wish to sample, defined to be minus log probability according to canonical distribution. With the notation in Bayesian Inference:

$$p(\theta \mid D) \propto exp(-U(\theta)) \tag{20}$$

Algorithm 2 Basic Hamiltonian Monte Carlo Algorithm

```
$x_0 \sim \pi_0$
**while** $t < N$ **do**
    Define L, and $\epsilon$
    $u \sim U(0,1)$
    $p_0 \sim K(p) = p^T p/2$
    $[x^*, p^*]$ = leapFrog($L$,$\epsilon$,[$x_t, p_t$])
    $A = min(1, e^{-U(x^*)-K(p^*)+U(x)+K(p)})$
    **if** $u \leq A$ **then**
        $x_{t+1} = x^*$
    **else**
        $x_{t+1} = x_t$
    **end if**
**end while**
```

Eq. 20 draws the connection between potential energy and the joint likelihood, and help define U(x) (Neal, 2012), where q is the set of parameter in BNN Models, and D is the given data:

$$U(q) = -log\Big[\pi(q)L(D \mid q)\Big]$$

The negative log relationship transform the target from reaching a higher density area for posterior or joint likelihood function, into reaching a lower value area in the potential energy function(Figure 2).

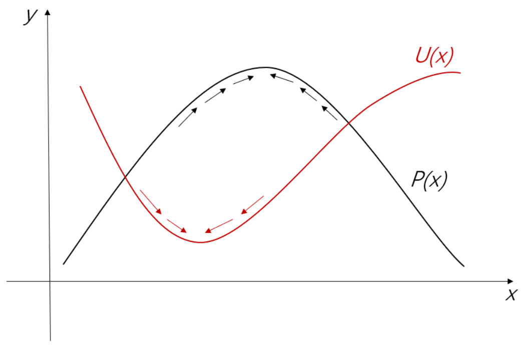


Figure 2: Probability distribution Function for U(x) and P(x). P(x) is the actual posterior that we want to simulate, U(x) is the potential energy canonical distribution function. In P(x), the best result happens at the local maximum, while in U(x), the best results happens at local minimum.

To realize this goal, in leapfrog methods, the momentum is updated based on the gradient of the potential energy (Eq. 13), and the position would be updated base on the new momentum.

$$
\begin{aligned}
p(t+\delta/2) &= p(t) - (\delta/2)\frac{\partial U}{\partial x(t)} \qquad (21)\\
x(t+\delta) &= x(t) + \delta\frac{\partial K(p)}{\partial p(t+\delta/2)}
\end{aligned}
$$

Interpreting the equation in Bayesian Statistic, the actual values for variables of interests θ are updated based on the momentum variable which is updated using the distribution U proportional to the joint probability and posterior(Eq. 20). In short, θ is updated based on a distribution proportional to the joint probability. Thus, in HMC, the value of the variables after update are more likely results in a lower density area in U(x), resulting in a higher density area for true posterior, this phenomena is one of the main advantage of deploying HMC to eliminate random walk behavior from Metropolis-Hasting algorithm.

If the leapfrog phase already propose candidates that is likely to move towards the target distribution, then what is the acceptance rate checking phase doing for HMC? Consider Acceptance rate function for HMC (Eq. 19):

$$A = min(1, e^{-U(x^*)-K(p^*)+U(x)+K(p)})$$

Rearrange exponential term from the acceptance rate can get:

$$
\begin{aligned}
e^{-U(x^*)-K(p^*)+U(x)+K(p)} &= exp\Big(-\Big[U(x^*)+K(p^*)\Big] + \Big[U(x)+K(p)\Big]\Big)\\
&= exp\Big(-H(x^*,p^*) + H(x,p)\Big)\\
&= exp(H - H^*) \qquad (22)
\end{aligned}
$$

What Eq.22 indicates is that the acceptance rate function is checking the energy gap between current state and the proposal state. The smaller the gap of the two state's total energy level, the closer acceptance rate will be one, and it's more likely to be taken. Considering that the new proposal candidate has a relatively big energy level, which will lead to a small result for acceptance rate, it indicates that the proposal candidate's potential energy is diverging too far from our target distribution and it might now be a good choice to take it. On the contrary, if the gap is too small, it will indicating that we are only explore nearby distribution and we might trapped in a local optimal instead of simulate to the target distribution.

4. NO-U-TURN SAMPLER

Proposed by Mattew Hoffman, and Andrew Gelman (Hoffman, Gelman, et al., 2014). No-U-Turn Sampler's aim is self-explanary in its name. Even though Hamiltonian Monte Carlo successfully eliminates the risk of random walking, due to the nature of Metropolis-Ha sting's Acceptance that only take first-order gradient, or, more specically, each state in the HMC only determined on the previous one state, at a higher level, the overall path might still facing a "U-turn" problem when exploring the possibility space.

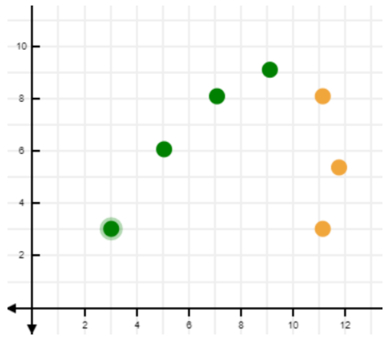


Figure 3: The dots indicate the position of each state of a certain particle in a 2D dimensional possibility space.

The green dots indicating the chain is heading towards one certain direction, then with orange dots heading back and make a U turn gradually since each time the momentum is updated base on the derivative potential energy, basically is the direction of the speed. So, it is easier for HMC to accumulate up steering angle.

4.1 No U Turn Mathew and Andrew proposed a way to restrain the direction by examine the positions of groups of particles. So the particle can move into a more informative direction and reach the target distribution (or position in this case) faster as shown in Figure 4.

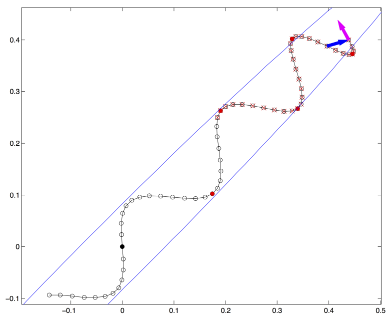


Figure 4: NUTS trajectory termination and sample selection. (Figure from Hoffman and Gelman, 2014.)

The No-U-Turn sampler define when the Hamiltonian simulation or Leapfrog method should stop:

$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r} < 0 \tag{23}$$

Where θ is the location, $\tilde{\theta}$ is the current location, and $\tilde{r}$ is the momentum vector. One way to interpret the equation is when half squared distance from initial θ to current θ is negative, the system is going backwards and the simulation should explore other tracks.

NUTS will using a method called doubling that check groups of nodes at a time. Before detecting the U-turn, NUTS will repeatedly explore the distribution space in a doubling manner. There will be two choices of direction, either forward or backward, each time there will be a fair possibility for each direction for this round, then move one step first, then two steps, then four, 2^i times and keep doubling until either sub-trajectory violate Eq. 23.

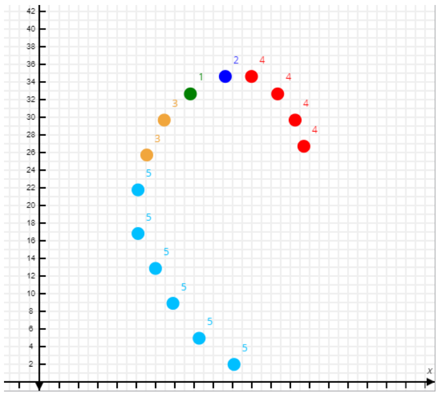
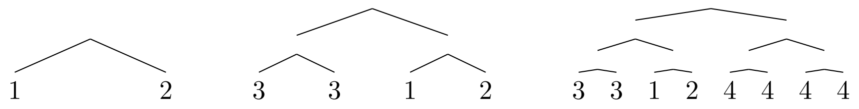


Figure 5: The number indicating how many steps there will be, Start from green 1, and then forward one step to blue 2, backward 2 step to yellow 3, forward 4 steps into red 4, and then backward 8 steps into 5

Due to the nature of the design, a balanced binary tree can be made and make checking the U tuen even easier among the subtrees.



4.2 No Manual set up Based on the NUTS sampler, the doubling will halt and the sampler will check more carefully when the subtrajectory from the leftmost to the rightmost ndoes of any balanced subtree of the overal binary tree starts to make the "U" turn. In this way, Nuts can help with tuning number of steps L automatically.

NUTS also help with tuning ϵ automatically with Dual Averaging technic, recall ϵ is the time interval parameter for leapfrog method. H_t defined as some behavior of target paramter, where t indicating iteration time:

$$H_t = \delta - \epsilon_t$$

another way to interpret is H_t describe how well the autotuning process for the step size behave – the gap between desired step size and current step size. And then parameter will be updated accordingly base on the gap.

$$\epsilon_{t+1} = \epsilon_t - \eta_t H_t$$

where learning rate η_t is defined as:

$$\eta_t \equiv t^{-k} \quad k \in (0.5, 1]$$

So as t approach infinity, the change on ϵ will be 0, that's also why NUTS will tune step size in the warm up phase and keep it during the actual simulation phase.

5. EMPIRICAL EXAMPLE

Here we will provide an empirical Evaluation of the BNN, recall that this will be an implementation example of the entire linear regression Bayes Family, primarily Markov Chain Monte Carlo, with Metropolis-Hasting, and a special version called Hamiltonian with NUTS extensions.

5.1 Introduction to the problem and data The data is Red wine quality data (Cortez et al., 2009). The data contain 1600 records, features include acidity level, sugar level, and etc. the target label is the quality of the wine, ranging from 3 to 7. the distribution of the quality is denser around 5 and 6 (figure 6)

we aimed at building a logistic regression model for predicting the quality of the wine using Bayesian Neural Network. Instead of telling an exact label all the time, we aimed at find how confident our model is about the prediction. Let our model tell us the probability of each label

5.2 Method and Materials We will use this example to connect all the algorithms that we mentioned before, and see the pipeline of probability prediction. We will use tensorflow probability (Martín Abadi et al., 2015) and edward2 (Tran et al., 2018) packages. A linear regression relationship is formed to rank the quality of wine. A set of 11 distributions are used for coefficients weights to match 11 features of the dataset in order to create the linear regression relationship, 1 for bias that shared by all features, 1 for shared noise. The distribution of coefficients, bias, and noise span initialized from normal distribution and the

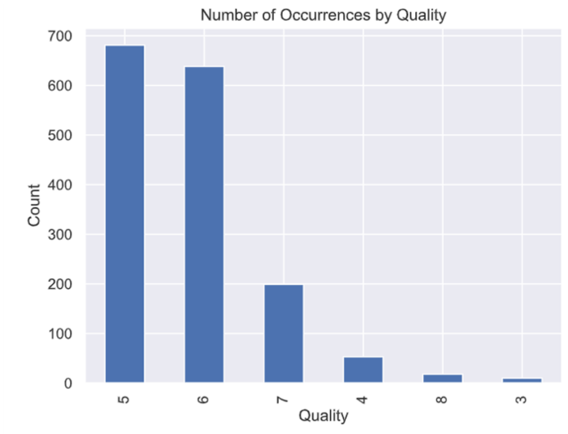


Figure 6: Distribution Plot of 11 features, bias and noise, for all 5000 states after burn-in, we can see they span around different range. x[0] is around 0.1, while x[1] is around -0.25

joint probability formed will be burnt-in for 5000 rounds in the HMC chain, with No-U-Turn Sampler as the plugin for autotuning leap time and leap step. the 5000 states after buring in phase will be saved for result and performance analysis.

5.3 Results After burn in state, we can see that coeffient, bias and noise nicely fell around different number and form new distribution as the latent parameter (Figure 7,8). The model produced a distribution of quality prediction instead of just one integer for one record in our data set(Figure 9), the target rank 6 (blue vertical line) falls within the confidence of Interval, and the intersection between actual label and interpolation curve tells us that the probability of being a quality 6 wine is around 0.42. A residual graph can be drawn to test the accuracy between the actual label and the mean of each prediction distribution, smaller the absolute value, smaller the gap. we can see around 60% of the prediction have a relative small difference (around 0) than the actual label (Figure 10).

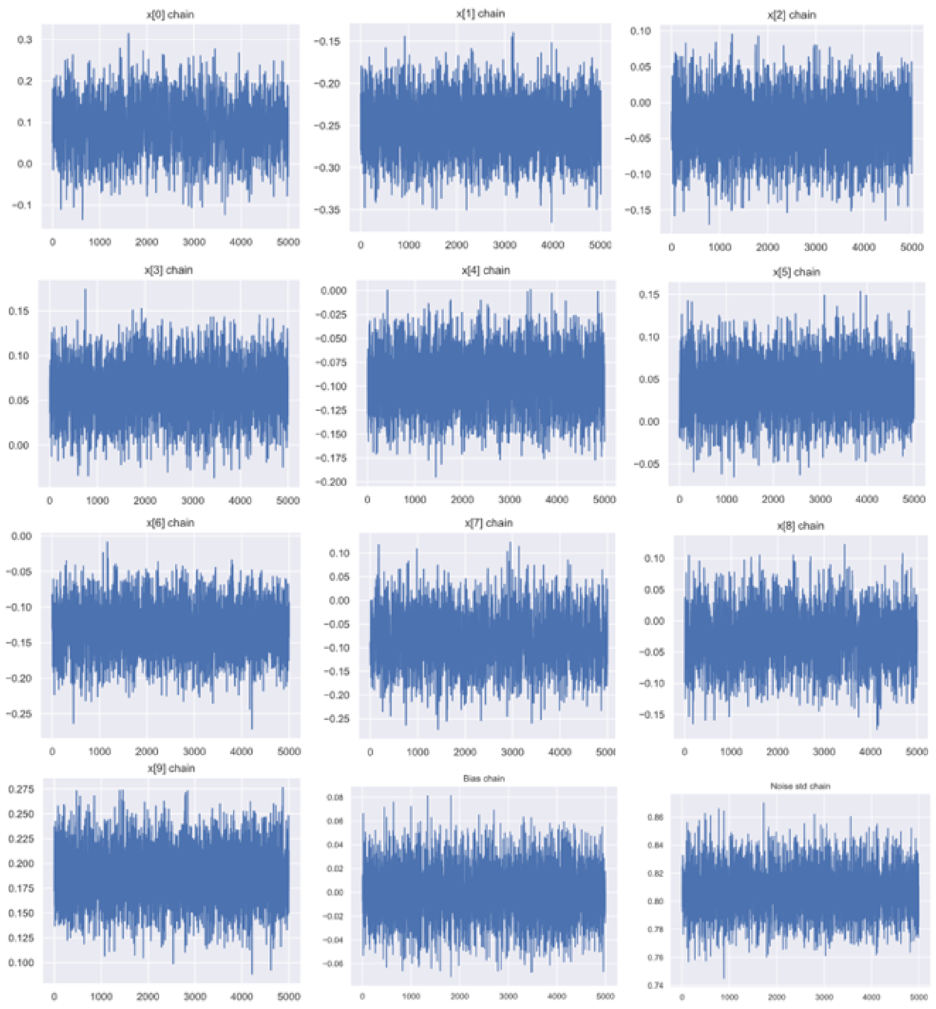


Figure 7: Distribution Plot of first 10 features, bias and noise, for all 5000 states after burn-in, we can see they span around different range. x[0] is around 0.1, while x[1] is around -0.25

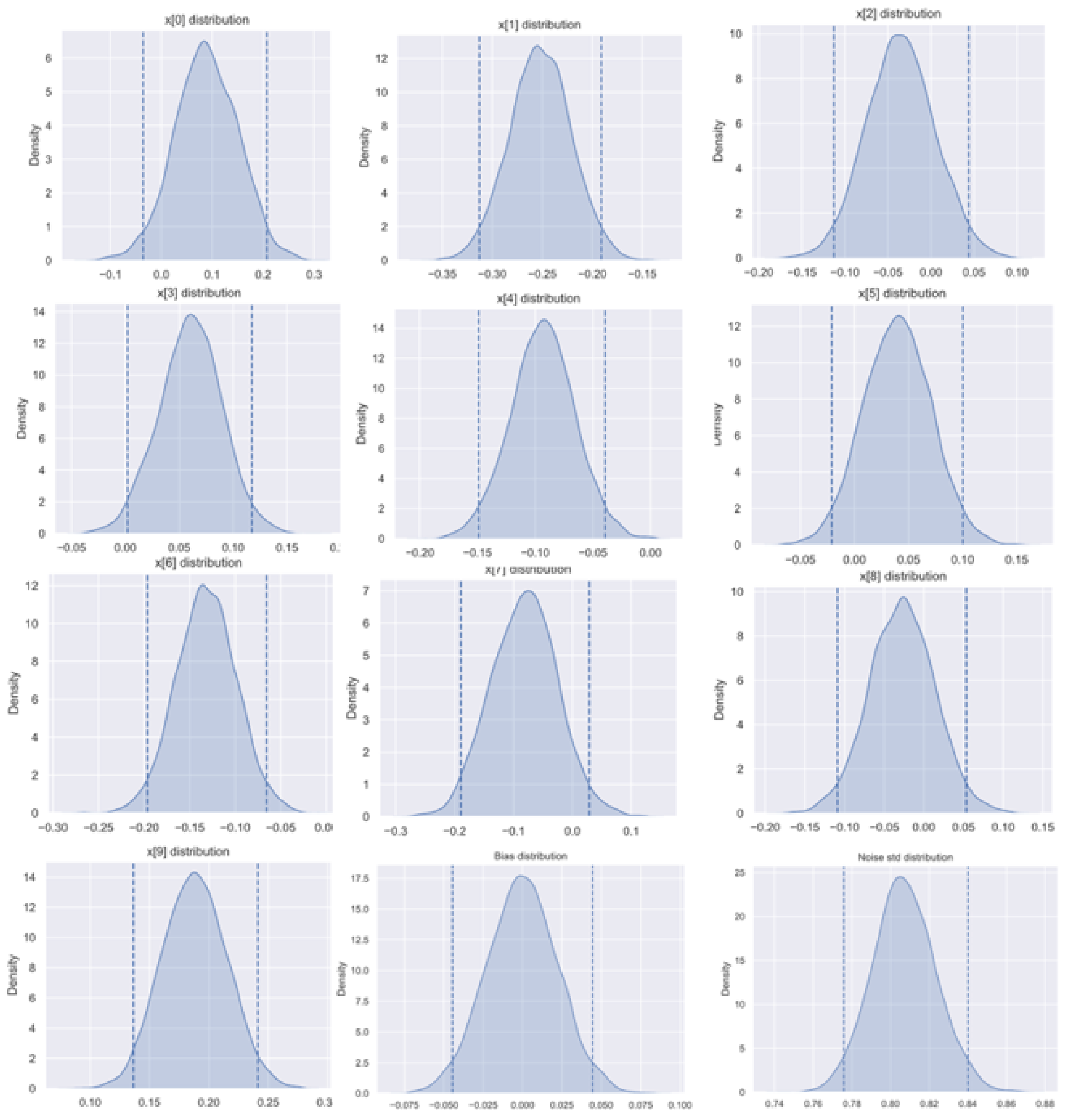


Figure 8: Density plot of first 10 features, bias and Noise, which is corresponding to figure 7, each coefficients, bias or noise have a distribution around different value after burn-in.

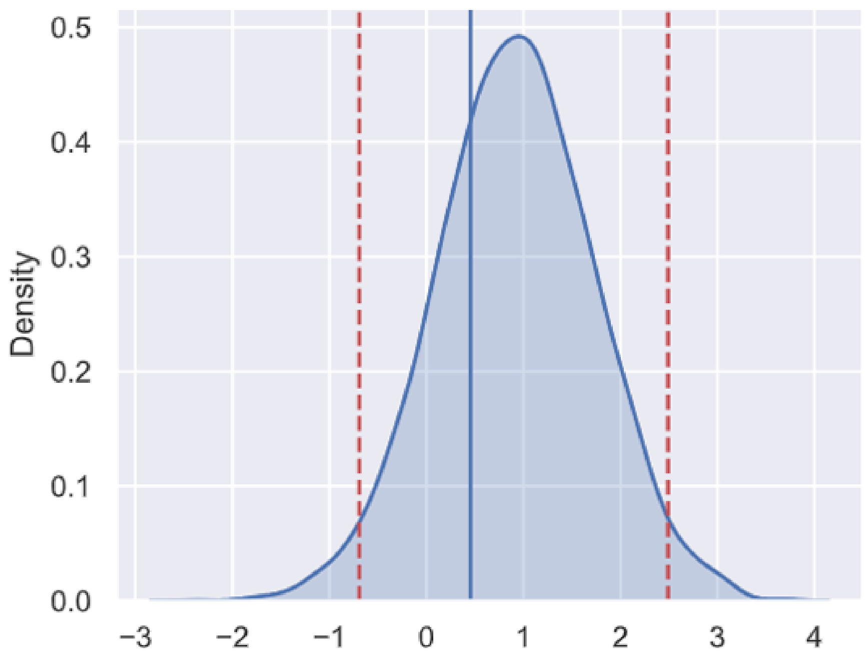


Figure 9: the graph shows the result for one record in the dataset, the blue shade area is the where the result distribution scattered in, the blue curve is the interpolation line base on the result distribution, red dot lines mark the boundary for 95% confidence of Interval, the blue vertical line is the actual label for this record, after preprocessed by normalization.

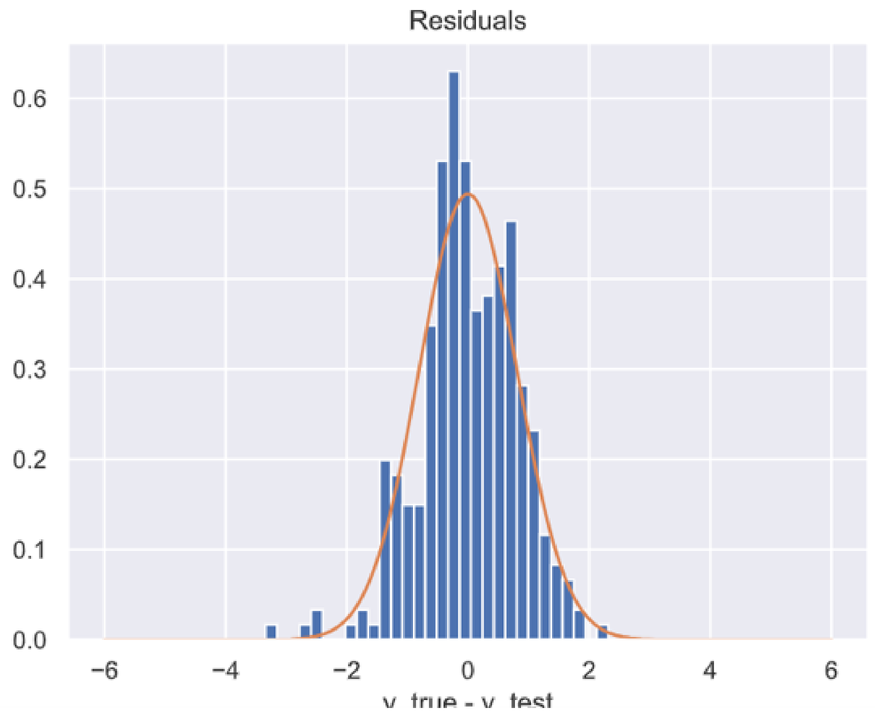


Figure 10: the x axis is the different true label and the mean of the prediction, the smaller the absolute value (closer to 0) indicating that the gap is smaller, the distribution shows that we have around 60% of the record have a correct prediction after taking mean of the prediction distribution.

REFERENCES

Chib, S., & Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. *The american statistician*, *49*(4), 327–335.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties [Smart Business Networks: Concepts and Empirical Evidence]. *Decision Support Systems*, *47*(4), 547–553. https://doi.org/https://doi.org/10.1016/j.dss.2009.05.016

Duane, S., Kennedy, A. D., Pendleton, B. J., & Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, *195*(2), 216–222.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications.

Hoffman, M. D., Gelman, A., et al. (2014). The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, *15*(1), 1593–1623.

Jospin, L. V., Laga, H., Boussaid, F., Buntine, W., & Bennamoun, M. (2022). Hands-on bayesian neural networks—a tutorial for deep learning users. *IEEE Computational Intelligence Magazine*, *17*(2), 29–48. https://doi.org/10.1109/MCI.2022.3155327

Markov, A. (2006). Extension of the law of large numbers to quantities, depending on each other (1906). reprint. *Journal Électronique d'Histoire des Probabilités et de la Statistique [electronic only]*, *2*(1b), Article 10, 12 p., electronic only–Article 10, 12 p., electronic only. http://eudml.org/doc/128778

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, . . . Xiaoqiang Zheng. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems [Software available from tensorflow.org]. https://www.tensorflow.org/

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, *21*(6), 1087–1092.

Neal, R. M. (2012). *Bayesian learning for neural networks* (Vol. 118). Springer Science & Business Media.

O'Hagan, A. (2010). *Kendall's advanced theory of statistic 2b*. Wiley. https://books.google.com/books?id=ewyYEAAAQBAJ

Tran, D., Hoffman, M. D., Moore, D., Suter, C., Vasudevan, S., Radul, A., Johnson, M., & Saurous, R. A. (2018). Simple, distributed, and accelerated probabilistic programming. *Neural Information Processing Systems*.