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

TIANLONG WANG* 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

There is real world adversail attack for traditional machine leanrning, and 'there is no free lunch', traditional neural network all face the problem of overconfidence, taking our example into account, the probability of burgarly is 0.001, however we bump it, the result of "No Burglary" will always win by 999 to 1 ratio. to avoid this happening and get an estimate probability of what will happen, we can modify traditional deep neural network and use Bayesian Neural Network to solve the problem.

we will prepare the dataset and training it on both normal ann and ann with a stochastic Bayesian network to see the difference of results. We will use D for the dataset, Dx for training data, Dy for the label, θ for the weights. First, let's express a traditional neural network with the given.

$$\begin{aligned} l_0 &= x, \\ l_i &= s_i(W_i l_{i-1} + b_i)\forall_i \in [1, n]\,, \\ y &= l_n. \end{aligned}$$

the biggest issue of traditional deep learning method is weights are always set up into one single number, which is a direct cause of overconfidence of the model. BNN will initialize parameters θ as a Gaussian Distribution

$$\theta \sim p(\theta) = \mathcal{N}(\mu, \Sigma)$$

And calculate the conditional probability as:

$$y \sim p(y \mid x, \theta) = \mathcal{N}(\Phi_\theta(x, \Sigma))$$

we can summarize the weight and bias as following and the work flow of BNN as:

$$\begin{aligned} &W \sim \mathcal{N}(\mu_W, \Sigma_W), \\ &b \sim (\mu_b, \Sigma_b), \\ &l = s(Wl_{-1} + b) \end{aligned}$$

To implement the actual math part of finding the unobserved variable z (sometime z can be out final classification or regression), we can build a latent variable model:

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

And plug in Eq. 4 we can see why the a direct calculate on the marginal distribution $p_\theta(x)$ is intractable:

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z) p_\theta(z)}{\int p_\theta(x, z')dz'}$$

the denominator part of Eq.5 is prohibitively impossible, we need to calculate:

$$p(x) = \int \int \cdots \int p(x, z')dz_1 dz_2 \cdots dz_{x-1}$$

which will grow fast as the dimensions of the data grows.

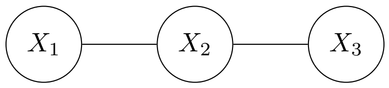
To overcome the intractable integral problem, mathematicians develop several ways to avoid the calculation of this "normalizing constant", here we will present two ways after we transform our problem into a simpler one for notation:

$$p(x) = \frac{f(x)}{NC} \tag{2}$$

2. MARKOV CHAIN MONTE CARLO SIMULATION

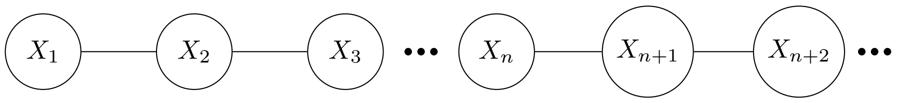
In this section, we will explore what is MCMC how it works mathematically.

2.1 what is Markov Chain MC Since we can not compute target distribution p(x) directly, so we will start randomly and simulate draws from p(x), Like shown in the graph:



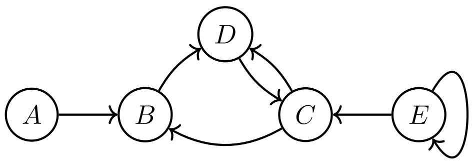
As Markov Chain stated, the new sample will depends on the last one. So in this graph, X_2 depends on X_1, X_3 depends on X_2, so on. In between each sample, there will also be transitional probabilities $T(x_2 \mid x_1)$,$T(x_3 \mid x_2)$ ... connects them and guide the growing of the Markov Chain to make sure we can arrive at the target distribution.

The target distribution is also called the steady state. Markov Chain states that, once we reach steady state, if we move one more step with our probability in the Markov Chain, we will still stay on the current probability.



As shwon in the above graph, after the "burn-in" process of X_1,X_2,X_3 ...We will have X_n, X_{n+1}, X_{n+2} that stay on our target distribution p(x).

2.2 Markov Chain Stationary Distribution we will explain what is stationary distribution through an example:



where $A \to B$ indicating move from A to B, we will also provide with a Transition table for each single node. Moving from one stage to the next stage:
Our question will be, given a probability Vector π, π_0 will be our inital distribution, to calculate the next stage π_1, we will use:

$$\pi_0 \cdot T = \pi_1$$

To generalize it, and indicate one step of move during the stationary distribution stage, we will have:

$$\pi \cdot T = \pi$$

	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: To understand the table, focus on node D, we have node B with a probability of 1, node C with a probability of 0.5 of moving into node D at the next stage.

This is exactly what steady state means in Markov Chain that we mentioned before. An intuition behind this is, after we reach the steady state, probability of each single node will not change anymore even we move one more step into the next stage of the Chain. For each single node, with the help of Transitional Matrix we will have:

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

we will also have a hidden equation for the summation of all node's probability:

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

After solving the equation, we can get the steady state of the π distribution as:

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

This indicating that the distribution probability vector π will not change anymore after we reach the steady state. Once we arrive the stationary distribution, we can apply detailed balance (or reversibility condition) on any two connected states in the Markov Chain:

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

Here $T(y \mid x)$ indicate the transitional probability given state x to y. the intuition behind maybe as we reach a stable distribution in the Markov Chain, moving forward and backward will result in the same result probability.

2.3 Metropolis-Hasting Algorithm One of the methods that determine the transition probability is Metropolis-Hasting Algorithm. we first propose a new candidate with $g(y \mid x)$ sample from any symmetric distribution like Normal distribution or asymmetric distribution like Rayleigh Distribution.

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

After we propose a new candidate, we will introduce an acceptance rate $A(x \to y)$ to decide if we will acept our candidate or not, so we can rewrite the Transitional Probability in Eq. 17 with Propose distribution and the acceptance rate:

$$\frac{f(x)}{NC} g(y \mid x) A(x \to y) = \frac{f(y)}{NC} g(x \mid y) A(y \to x)$$

after simplification, we can get:

$$\frac{A(x \to y)}{A(y \to x)} = \frac{f(y) \cdot g(x \mid y)}{f(x) \cdot g(y \mid x)} \tag{4}$$

from here, we have multiple way to interpret and keep simplify the equation. We will use the same idea as the original paper. To calculate $A(x \to y)$ Let's consider the case that (to calculate $A(y \to x)$, we can simply reverse the mark in the following inequality):

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

This means we move from x to y too often, and move from y to x to rarely. To balance that, we need to set the acceptance rate from y to x to maximum, which is:

$$A(y \to x) = 1$$

So Eq. 18 will be:

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

we 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). Also considering the fact that the probability of accepting a move can not be over one, we can define the acceptance probability as:

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

A pseudo code for Metropolis-Hasting Algorithm will be:

2.4 Intuition for MCMC and Metropolis-hasting algorithm to adopt this idea into our problem, our parameters distribution will stay unchanged after we reach the stationary distribution. For example, $p(\theta)$ which include $\{\theta_1, \theta_2, ..., \theta_n\}$ will not change a lot after we accept new sets of parameters

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
```

$\{\theta_1^*, \theta_2^*, ..., \theta_n^*\}$

let's discuss the intuition behind why Metropolis-hasting algorithm can provide us the posterior with only the joint probability.

As we said before, the true posterior is proportional to our joint probability with a normalizing constant:

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)} \tag{5}$$

where left hand side is the posterior, numerator at right hand side is the joint probability, the denominator is the Normalizing Constant that we can not calculate. Keep in mind the posterior is the probability of the parameters of the model given the dataset. So we are actually keep track of the parameters in our Markov Chain. Let's look at an naive example.

Considering our linear regression is as simple as $2x + 1$, where weight is 2 and bias is 1, and only one dimensional for x and weight. so $(2, 1)$ will be one of the state in our Markov Chain. Now, let's use the proposal function and get the new candidate as $(3, 2)$. (recall that proposal function g(x) is simply a simulation of sample from some distribution that we defined ourselve.). Then how can we make sure that the new proposal is the desired one that we want? Or in another word, how can we know that we are move into a correct direction? Now we take a second look at the acceptance rate:

$$\alpha = min(1, \frac{f(y) \cdot g(x \mid y)}{f(x) \cdot g(y \mid x)}) \tag{6}$$

if we assume our proposal is symmetric (that we will prose x if we are finding new candidates at y which is proposed from x itself), then we can simplify the acceptance rate into:

$$\alpha = min(1, \frac{f(y)}{f(x)}) \tag{7}$$

and let's consider about the case that $f(y) > f(x)$, what this mean is that in our probability curve, (which is the known joint probability but not the actual posterior), the new proposed state will result in a higher probability than the current one. Like shown in the graph, since joint probability is propotinal to our posterior, if we know the new state is lead to a higher probability in joint distribution, then we can say that the new state will also lead to a better probability in the actuall posterior, in another word, the new state is indeed the new sets of parameters that we want to explore and save.

On the contrary, if we are not resulting in a higher probability that the previous state, we will still acept it base on whether the acceptance rate is higher than a dynamic threshold. recall acceptance rate is the ratio of probability between the next state and the current state, the closer of the two state, closer the acceptance rate to 1. recall that the threhols is uniformlly drawed between $0 \sim 1$.

$$u \sim (0, 1)$$

This means if we are only decreasing a little bit, the ratio between the probability of next state and current state is still close to 1, we are still have a higher chance to accept this proposal, since we are not building a greedy algorithm and we want to explore the distribution space more comprehensively.

what we want to summarize here is (which is basically the definition of MCMC): even though we can not compute the posterior, but if we can make sure that we are keep moving into the right direction with the help of the joint probability distribution, then we will reach the target probability eventually.

3. HAMILTONIAN MONTE CARLO SIMULATION

3.1 HMC It may seems astonishing that we use physic ideas Hamilton Dynamic in Machine Learning or Math, but this is not actually the first time. Recall Gradient Descent with Momentum, there are so many different area that adopt the idea of Physics, or in another way, all different subjects cooperate together to solve our problems.

Hamiltonian Dynamic describes the following: imagine a particle moving on a 2D dimensional surface. the particle have a position vector x, and a momentum vector p. we use $U(x)$ to define the potential energy, $K(p)$ to define the kinetic energy. Hamiltonian Dynamic tell us the energy of the system will be:

$$H(x, p) = U(x) + K(p) \tag{8}$$

then we will find the current momentum and current position with the help of the partial derivative of the position, and Kinetic Energy respectively.

$$\frac{dx}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K(p)}{\partial p} \tag{9}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial x} = -\frac{\partial U(x)}{\partial x} \tag{10}$$

In hamiltonian Monte Carlo simulation, we will assign $K(p)$ as quadratic kinetic energy, and $U(x)$ as negative logarithm of the actual target distribution:

$$K(p) = \frac{p^T p}{2} \tag{11}$$

$$U(x) = -log\,\pi(x) \tag{12}$$

where $\pi(x)$ is the joint distribution of the variables.

We will use LeapFrog method to simulate the movement of the "particle" start from t to $t + T$. In real world, time T is a continuous number, but for the convience of calculation, we will discretize T into smaller intervels δ, within onestep of the leapfrog method, first we will update the momentum within $\delta/2$ interval using Eq. 21 and plug in the articulated equation for potential energy later:

$$p(t + \delta/2) = p(t) - (\delta/2)\frac{\partial U}{\partial x(t)} \tag{13}$$

and then update the new position for the entire δ time base on the updated momentum:

$$x(t + \delta) = x(t) + \delta\frac{\partial K(p)}{\partial p(t + \delta/2)} \tag{14}$$

and finally update the momentum for rest of the $\delta/2$ time interval so the time will match up again:

$$p(t + \delta) = p(t + \delta/2) - (\delta/2)\frac{\partial U}{\partial x(t + \delta)} \tag{15}$$

During the Metropolois update phase, leapfrog method will be executed L times and use δ as the length of the time. These two are important parameters that a slightly change with lead to a drastically change of the result. So we need to manually set up when we performing Hamniltonian Monte Carlo, but we will introduce a new algorithm in the next section which automatically sample those two parameters for us.

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.

So far we are still in Physics, we will build a math or canonical distribution for the energy function so that we can compare our candidates and determine if we will accept the move or not, considering some basic energy function $E(\theta)$:

$$E(\theta) = H(x,p) = U(x) + K(p) \tag{16}$$

the canonical distribution over states has PDF as:

$$q(\theta) = \frac{1}{Z} e^{-E(\theta)} \tag{17}$$

we use q substitue the p from original paper since we don't want to mix between the probability distribution and the momentum variable. And Z term is a normalizing term and we will neglect it. Substitue the energy function and explain paramter θ as x and p we can get:

$$\begin{aligned} q(x,p) &\propto e^{-H(x,p)} \\ &= e^{-[U(x)+K(p)]} \\ &= e^{-U(x)} + e^{-K(p)} \\ &\propto q(x)q(p) \end{aligned} \tag{18}$$

while the result can tell us that energy of position and energy of momentum are not dependent on each other, that's also why we can set $U(x)$ and $K(p)$ seperately and take derivative seperately. But what we need here is the Eq. 29:

$$q(x,p) \propto e^{-[U(x)+K(p)]} \tag{19}$$

and if we use x^* and p^* to denote the position and momentum of new proposed candidate respectively, new canonical distribution is going to be:

$$q(x^*,p^*) \propto e^{-[U(x^*)+K(p^*)]} \tag{20}$$

then we can build an acceptance probability like Metropolis-Hasting Algorithm by dividing $q(x^*,p^*)$ and $q(x,p)$ and then compare to 1:

$$A = min(1, e^{-U(x^*)-K(p^*)+U(x)+K(p)}) \tag{21}$$

So the general Hamiltonian Monte Carlo Algorithm will looks like Algrithm 2:

3.2 Intuition behind HMC What is x,$U(x)$, p, and $K(p)$ actually. x, the position vector will map to the current state of the system in the phase state. more specially, x is the variables of interests.

$$U(x) = -log\left[\pi(x)L(x \mid D)\right]$$

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($L$,$\epsilon$,$[x_t, p_t]$)
    **if** $u \leq A$ **then**
        $x_{t+1} = x^*$
    **else**
        $x_{t+1} = x_t$
    **end if**
**end while**
```

And if you don't mind, let's rephrase the equation that is consistent with this paper. x representing the varible of interests, in our case, it's for sure the variables of the neural network θ, $\pi(x)$ which trying to describe the prior, we will say it's $p(\theta)$, lastly,$L(x \mid D)$, which is basically the likelihood of parameters given dataset D. is exactly $p(\theta \mid x)$, where x is the dataset. so the new equation will be:

$$U(\theta) = -log(p(\theta) \cdot p(x \mid \theta))$$

If we take a closer look with the original equation of Bayes Theorem. This is exactly the numerator of the right hand side, which known as the joint distribution. So that also explain why we said $U(\theta)$ is proportional to the target distribution. Since U(x) is negative log of a distribution that is proportional to our target distribution posterior up to a normalizing constant.

now recall how we do leapfrog methods. first we will draw a momentum randomly, but we will update it base on the negative gradient of the potential energy $U(\theta)$, and then update our variable of interest base on the kinetic energy of the system or basically the momentum:

$$p(t + \delta/2) = p(t) - (\delta/2)\frac{\partial U}{\partial x(t)}$$
$$x(t + \delta) = x(t) + \delta\frac{\partial K(p)}{\partial p(t + \delta/2)}$$

Since the momentum that kick the particle is based on the potential energy equation which, as we discussed in the previous paragraph, is proportional to the target distribution, we will more likely kick the particle into a higher density area, or in another word, it will results in a higher joint distribution in the next round. This is exactly what we want in Bayesian Inference since we can not compute the target distribution directly, but we try to move our states into a higher probability area in joint distribution to get to the posterior eventually.

If we compare with Metropolist Hasting Algorithm, we can see that during the proposal phase, Metropolist-hasting algorithm will randomly draw candidate base on a distribution with mean of the previous state. So it need a round of check of acceptance rate to make sure we are moving into a correct direction. But here at HMC, we can clearly see that each proposal is more likely to be a promising choice of the next state in the Markov Chain. That's also why we say HMC can help with elimination of the random walk behavior. **here i want to include two graph of particle moving for the first part of metropolis and HMC, the result should looks like the mh is random walk, hmc is base on the position and momemtum**

Now a new question arise, 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? Let's bring back the equation of the acceptance rate again:

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

we will focus on rearranging the exponential term. But first, recall that Hamiltonian Dynamic equation and its canonical distribution:

$$\begin{aligned} H(x,p) &= U(x) + K(p) \\ P(x,p) &= e^{-H(x,p)} \\ &= e^{-U(x)-K(p)} \end{aligned}$$

we will use this to rearragne exponential term from the acceptance rate:

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

what we are compare here is actually the energy gap between current state and the proposal state. as long as the current state's energy is greater that proposal state's, than the acceptance rate is greater than 1, which means we will for sure take it. and if the total energy of the new proposal state is greater than the current state, it will result in a number that is lower than 1 and it will turn into the actual acceptance rate.

While another way to interpret this together with a graph is, the smaller the gap of the two state's total energy level, the closer acceptance rate will be one, and we are more likely to take it. Considering that the new proposal candidate has a relatively big energy level, which will lead to a small result for acceptance rate, it indicating that we are 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 **(need revise)**

4. NO-U-TURN SAMPLER

Hamiltonian Monte Carlo successfully eliminates the risk of random walking, but due to the nature of Metroplolis-Hasting's Acceptance. We still only check any two connected states. Here at a higher level, we might still face a "U" turn problem.

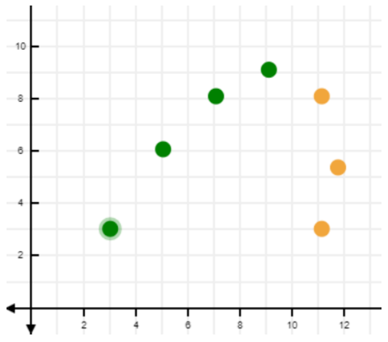


Figure 1: The dots indicate the position of each state of a certain particle in a 2D dimensional space.

If green dots indicating we are heading towards certain direction, then with orange dots, you can see we are heading back and make a U turn gradually since each time we simulate a momentum base on the derivative of the previous move, which tend to add up the steering angle.

This Paper() propose 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 5.

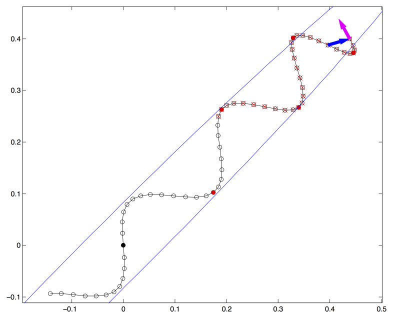


Figure 2: NUTS trajectory termination and sample selection. (Figure from Hoffman and Gelman, 2014.)

The No-U-Turn sampler define when we want to stop the Hamiltonian simulation:

$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r} < 0 \tag{22}$$

Where θ is the location, $\tilde{\theta}$ is the current location, and $\tilde{r}$ is the momentum vector. A direct interpretation of the equation is when half squared distance from initial θ to current θ is negative, we will do something else. we can also interpret it as when the angle between the position vector and momentum vector is even slightly below 90 degrees.

NUTS will check groups of nodes at a time. Before we detect the problem, we will repeatedly explore the distribution space doubling. we will have two choices of direction, either forward or backward, each time we will flip a coin to determine the direction, then we will move one step first, then two steps, then four, 2^i times and keep doubling until we find out either subtrajectory violate our defined funtion 33.

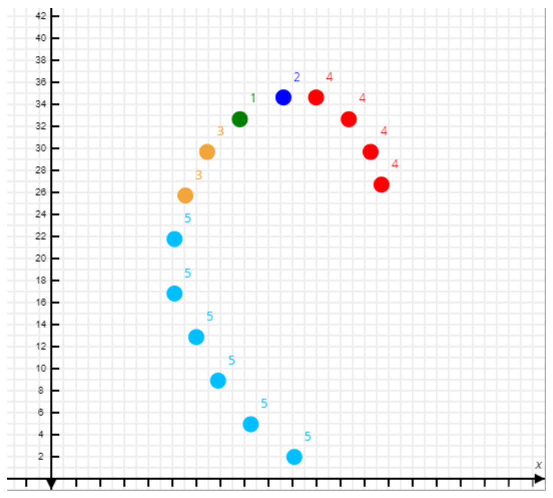
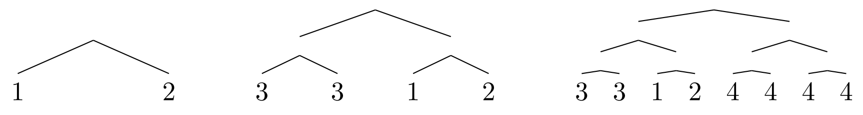


Figure 3: The number indicating how many steps there will be, we start from green 1, and then forward one step to blue 2, backward 2 step to 3, forward 4 steps into red 4, and then backward 8 steps into 5 (we didn't include all 8 of the blue in the graph)

Due to the nature of the design, we can easily build a balanced binary tree and check the equation even easier among the subtrees.



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. We define H_t for 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 our autotuning process for the step size behave – the gap between desired step size and current step size. And then we will update this parameter accordingly base on the gap.

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

where we define learning rate η_t 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, from Markov Chain Monte Carlo, to Metropolis-Hasting, Hamiltonian with NUTS extensions.

5.1 Introduction to the problem and data the data is red wine quality data (source). 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)

we aimed at finding the quality of the wine using Bayesian Neural Network. Instead of telling exact label, that one integer, 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 and edward2 packages due to it's scailibility to real-world problem. A linear regression relationship is formed to rank the quality of wine. A set of 11 distributions are used for coefficients weights to

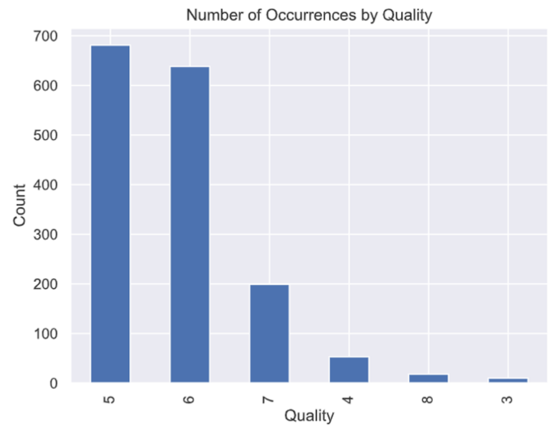


Figure 4: Distribution Plot of 11 features, bias and noise, for all 5000 states after burn-in, we can see they span around different range. x[0] is around 0.1, while x[1] is around -0.25

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 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 following 5000 states 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 8,9). The model produced a distribution of quality prediction instead of just one integer for one record in our data set(Figure 10), 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 11).

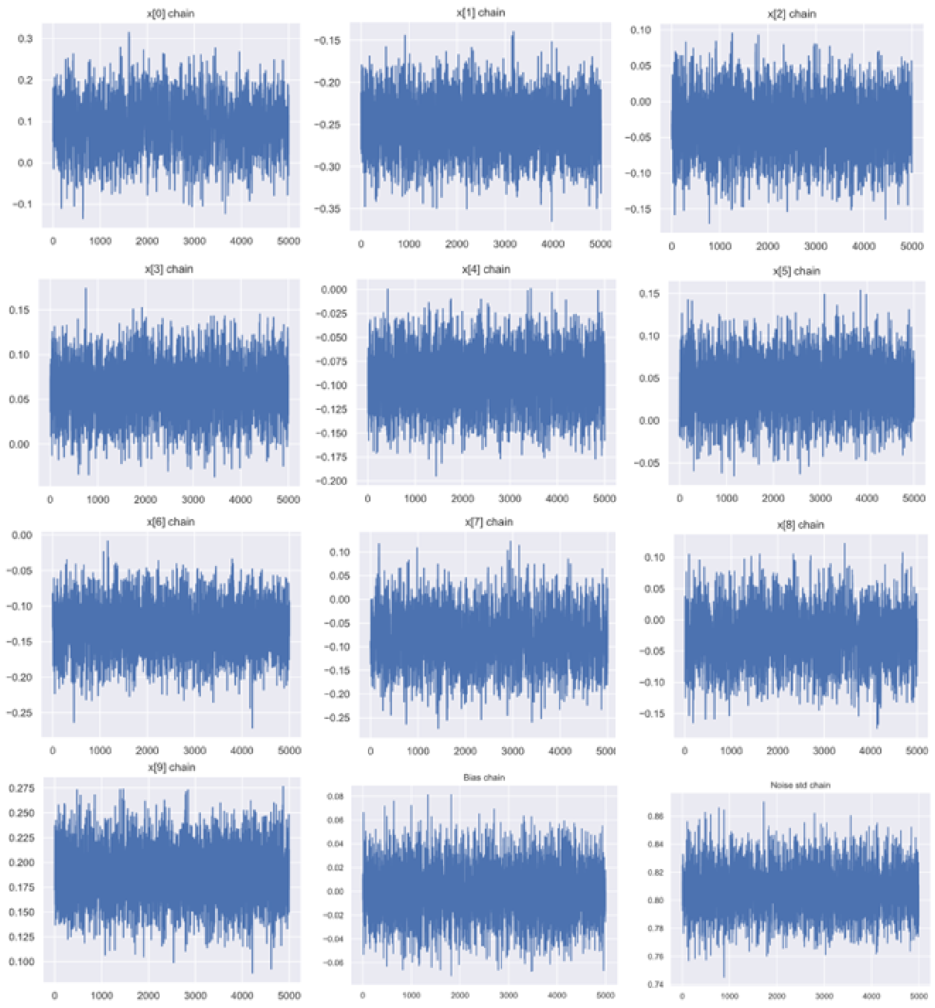


Figure 5: Distribution Plot of first 10 features, bias and noise, for all 5000 states after burn-in, we can see they span around different range. x[0] is around 0.1, while x[1] is around -0.25

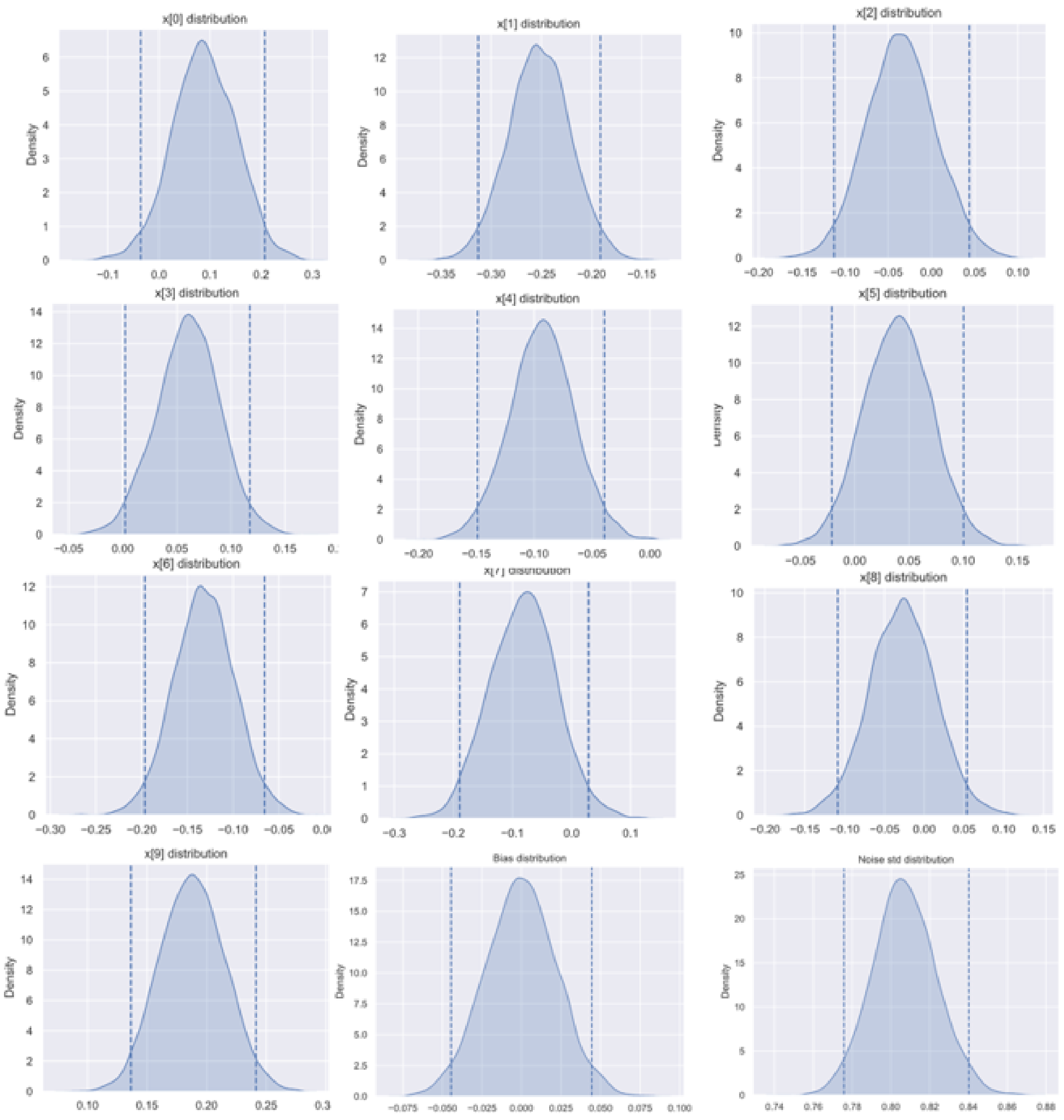


Figure 6: Density plot of first 10 features, bias and Noise, which is corresponding to figure 7, each coefficients, bias or noise have a distribution around different value after burn-in.

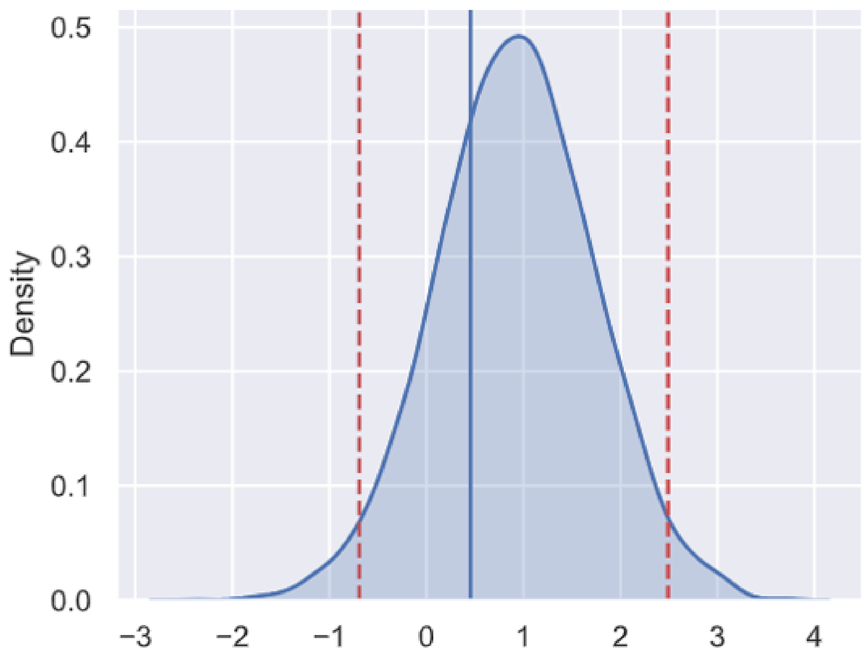


Figure 7: the graph shows the result for one record in the dataset, the blue shade area is the where the result distribution scattered in, the blue curve is the interpolation line base on the result distribution, red dot lines mark the boundary for 95% confidence of Interval, the blue vertical line is the actual label for this record, after preprocessed by normalization.

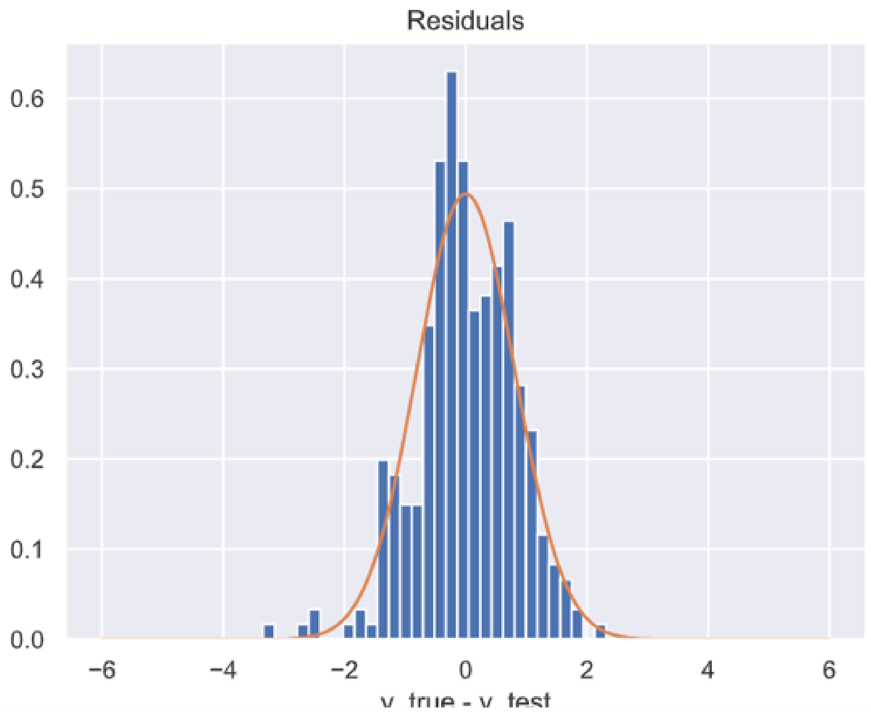


Figure 8: the x axis is the different true label and the mean of the prediction, the smaller the absolute value (closer to 0) indicating that the gap is smaller, the distribution shows that we have around 60% of the record have a correct prediction after taking mean of the prediction distribution.