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

.

July 2, 2023

1 Introduction

Machine Learning and Deep Learning have long been explored and developed by the data science community, while this is still another aspect of machine learning - Bayesian Inference which is using probability and more intuitive to humans to make predictions and approximations. The goal of this survey paper is to reveal the power of Bayesian theory first in math, then gradually adding ideas of machine learning into it. From pure math that can be calculated by hands of Naive Bayesian classifier and Bayesian Belief Network, to Markov Chain Monte Carlo Simulation that is still a math idea but need the help of computation power of computer, to, finally, a well design architecture that combine both Bayesian Inference in math and Stochastic Gradient Descent in machine learning, and eventually a Machine Learning model archetecture Vairational AutoEncoder that using the idea of Bayesian Theorm that highly used in Computer Vision and Image Reconstruction Area.

2 Bayes' and Probability Theorem

Here we will take about some basic annotations and equations that we will use later in the paper. In statistics, we can define the sample space as Ω, and assume the sample space is divided into several mutually exclusive cases $A_1, A_2, ..., A_i$ where $A_1 \cup A_2 \cup ... \cup A_i = \Omega$ and an event B that overlap with A_i shown in the following Venn graph.

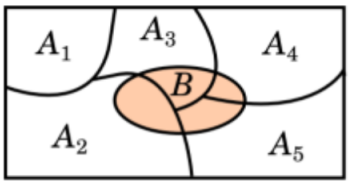


Figure 1: In this figure, the sample space is divided into 5 sets that partially overlap with set B

2.1 Joint Probability Theorem

$$P(A \mid B) = \frac{|A \cap B|}{|B|} = \frac{\frac{|A \cap B|}{|\Omega|}}{\frac{|B|}{|\Omega|}} = \frac{P(A \cap B)}{P(B)} \tag{1}$$

$$P(B \mid A) = \frac{|B \cap A|}{|B|} = \frac{\frac{|B \cap A|}{|\Omega|}}{\frac{|A|}{|\Omega|}} = \frac{P(B \cap A)}{P(A)} = \frac{P(A \cap B)}{P(A)} \tag{2}$$

after rearranging (1) and (2), we can say that:

$$P(A \cap B) = P(A \mid B)P(B) = P(B \mid A)P(A) \tag{3}$$

2.2 Bayesian Theorem

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} = \frac{P(B, A)}{\int_A P(B, A')dA'} \tag{4}$$

2.3 Sum probability (not sure about the name)

Based on the graph, we can say:

$$P(B) = \sum_A P(B, A) \tag{5}$$

3 Naive Bayesian Classifier

3.1 Equation

Naive Bayesian Classifier will assign a posterior probability on training instance $X_1, X_2, ..., X_i$ for all the labels $y_1, y_2, y_3, ..., y_j$, which is $P(Y = y_j \mid X = x_i)$, using Bayes' theorem (Eq. (4)), we can get:

$$P(Y = y_j \mid X = x_i) = \frac{P(x_i \mid y_j)P(y_j)}{P(x_i)} \tag{6}$$

note that the numerator is the join probability of x_i and y_i (Eq.3):

$$P(x_i \mid y_j)P(y_j) = P(x_i \cap y_j) = P(x_i, y_j) \tag{7}$$

if we assume each training instance x_i has feature set $x_{i1}, x_{i2}, x_{i3}, ..., x_{ip}$, and using join probablity again (Eq.3) we can further explain Eq.6, here we omit index of x for as investigating on one single instance:

$$\begin{aligned} P(x, y_j) &= P(x_1, x_2, x_3, ..., x_p, y_j) \\ &= P(x_1 \mid x_2, x_3, ..., x_p, y_j)P(x_2, x_3, x_4, ..., x_p, y_j) \\ &= P(x_1 \mid x_2, x_3, ..., x_p, y_j)P(x_2 \mid x_3, x_4, ..., x_p, y_j)P(x_3, x_4, ..., x_p, y_j) \\ &= P(x_1 \mid x_2, x_3, ..., x_p, y_j)P(x_2 \mid x_3, x_4, ..., x_p, y_j)...P(x_p \mid y_j)P(y_j) \end{aligned} \tag{8}$$

Here each multiplier imply the contribution to the final result of each attribute given the rest of the attributes and the label. Naive Bayesian Classifier will simply discard the relation among the attributes, treat attributes as mutually exclusive, so we can simplify Eq.7 further by geting rid of attributes in the equation:

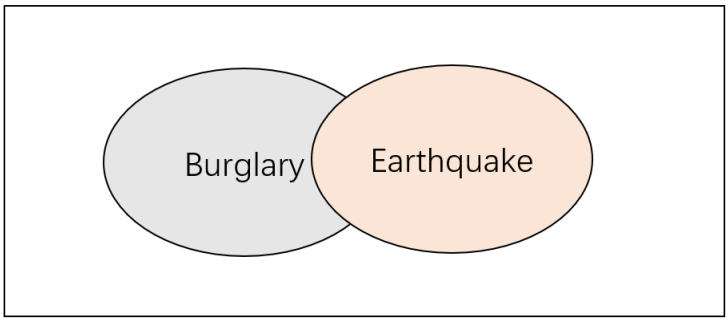
$$P(x, y_j) = P(x_1 \mid y_j)P(x_2 \mid y_j)P(x_3 \mid y_j)...P(x_p \mid y_j)P(y_j) \tag{9}$$

we can plug this back into Eq. 5 and we get a generative model equation for Naive Bayesian Classifier:

$$P(y_j \mid x) = \frac{\prod_{k=1}^{p} P(x_k \mid y_j)P(y_j)}{P(x)} \tag{10}$$

3.2 Example

Here we have one example for the utilization of a Naive Bayesian Classifier, and we will continue using this example for the later Bayesian Network for comparison. Assume we have 100000000 events happening on a house, among them, 0.1% is burglary, 0.2% is an earthquake, and the overlapping (that both burglary and earthquake happen at the same time.) is 10000 events. Like shown in the figure.



The alarm system will work in the given probability, base on the burglary and earthquake, our neighbors, John and Mary will either call us or not with the given probability base on if the alarm goes off or not.

$$P(A \mid B, E) = \begin{bmatrix} a & b & e & 0.95 \\ a & b & \neg e & 0.94 \\ a & \neg b & e & 0.29 \\ a & \neg b & \neg e & 0.001 \\ \neg a & b & e & 0.05 \\ \neg a & b & \neg e & 0.06 \\ \neg a & \neg b & e & 0.71 \\ \neg a & \neg b & \neg e & 0.999 \end{bmatrix} \quad P(J \mid A) = \begin{bmatrix} j & a & 0.9 \\ j & \neg a & 0.05 \\ \neg j & a & 0.1 \\ \neg j & \neg a & 0.95 \end{bmatrix} \quad P(M \mid A) = \begin{bmatrix} m & a & 0.7 \\ m & \neg a & 0.01 \\ \neg m & a & 0.3 \\ \neg m & \neg a & 0.99 \end{bmatrix}$$

Apply the probability into our assumption, we can get the data table as Figure 1:

Total number	Event	Alarmed?	John	Mary
100000000	Burglary: 90000	Alarmed: 84600	Called:76140	Called:59220
			No Call:8460	No Call:25380
		Unalarmed: 5400	Called:270	Called:54
			No Call:5130	No Call:5346
	Earthquake: 190000	Alarmed: 55100	Called:49590	Called:38570
			No Call:5510	No Call:16530
		Unalarmed: 134900	Called:6745	Called:1349
			No Call:128155	No Call:133551
	B+E: 10000	Alarmed: 9500	Called:8550	Called:6650
			No Call:950	No Call:2850
		Unalarmed: 500	Called:25	Called:5
			No Call:475	No Call:495
	Nothing: 100000000	Alarmed: 997100	Called:897390	Called:697970
			No Call:99710	No Call:299130
		Unalarmed: 99610290	Called:49805145.5	Called:996102.9
			No Call:94629776	No Call:98614187

Figure 2: From left to right, total number indicating that total events number, 90000 are Burglary alone, 190000 are Earthquake alone, 10000 of them are burglary and Earthquake happen at the same time, the rest are no special events. then in different events, alarm will work or not base on it's own probabilities, which is column 3, then our two neighbors, John and Mary, will call us when the alarm goes off, and won't call us when there is no alarm, but there are also chances that our neighbor doesn't call us when the alarm goes off for whatever reason, or just call us without any alarm for, again, whatever reason.

Our question is find the probability of Burglary when both John and Mary call us, using equation

of Naive Bayesian Theorem (Eq. 10), we can get:

$$P(B \mid J, M) = \frac{P(J \mid B)P(M \mid B)P(B)}{P(x)}$$
$$P(\neg B \mid J, M) = \frac{P(J \mid \neg B)P(M \mid \neg B)P(\neg B)}{P(x)}$$

Plug in numbers and we can get:

$$P(B \mid J, M) = \frac{0.0005602976065}{P(x)}$$
$$P(\neg B \mid J, M) = \frac{0.0005674753209}{P(x)}$$

Here we can see after the consideration of different factor, we almost half the probability of burglary, which use to be 0.1% and now is 0.05%. To get the percentage probability, we will normalize the results , and we can say that the probability of burglary when both our neighbor call us is:

$$P(B \mid J, M) = 0.4968177484$$

we will save the results before and after the normalization and compare them with the results from the follow architectures.

4 Bayesian Belief Network

4.1 what is Bayesian Network

In real world's situation, consider all features are mutual exclusive is not always the optimal choice, to improve the performance and take the dependency between features into account, Bayesian Network come in. Here, to calculate $P(J, M, A, B, E)$, First we using bayesian theorm (Eq. 3)

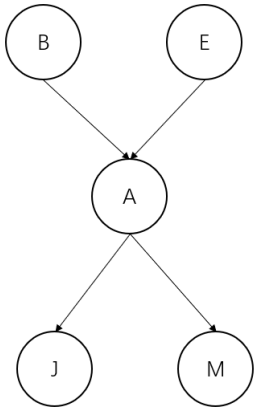


Figure 3: as shown in the graph, we can see, the alarm will depends on burglary happening and earhquake's happening, then Johhn and Mary will call us base on if the alarm goes off or not.

$$P(J, M, A, B, E) = P(J \mid M, A, B, E) * P(M \mid A, B, E) * P(A \mid B, E) * P(B \mid E) * P(E)$$

Since, J will not depends on M but on A, M will depends on A not B and E directly, B and E has no dependecy relationship, we can simplify the equation into:

$$P(J, M, A, B, E) = P(J \mid A) * P(M \mid A) * P(A \mid B, E) * P(B) * P(E)$$

so we can summarie the equation of Bayesian Network as :

$$P(x_1, ..., x_n) = \prod_{i=1}^{n} (P(x_i \mid parents(x_i) \tag{11}$$

4.2 example

Now considering the same example that we used in for naive bayesian classifer and the quesiton $P(B \mid J, M)$:

$$P(B \mid J, M) = \frac{P(B, J, M)}{P(J, M)}$$

Here we will replace the denomonator as α as a normalizer.

$$P(B \mid J, M) = \alpha * P(B, J, M)$$

the data table in figure 1 is calculated particular for Naive Bayes Classifier, usually we won't have exact number data for calculation of Bayesian Network and we will use the probability for each condition instead, for accuracy and generalization, using Eq.5, we can get:

$$P(B \mid J, M) = \alpha * \sum_A \sum_E P(B, J, M, A, E)$$

Here the order of $\sum_A$ and $\sum_E$ matters, Using join equation Eq. 3, and get rid of unnecessary dependencies, we can get:

$$P(B \mid J, M) = \alpha * \sum_A \sum_E P(B) * P(J \mid A) * P(M \mid A) * P(A \mid B, E) * P(E)$$

we can then move $P(B)$ entirely outside, since probability of B doesn't not depends on A or E, and probability of J given A, probability of M given A does not depends on E, so we can get:

$$P(B \mid J, M) = \alpha * P(B) * \Big(\sum_A P(J \mid A) * P(M \mid A) * \Big(\sum_E P(A \mid B, E) * P(E)\Big)\Big) \tag{12}$$

To calculate $P(A \mid B, E) \cdot P(E)$, we need to expand $P(E)$ to get same dimension for the element wise multiplication:

$$P(A \mid B, E)P(E) = \begin{bmatrix} a & b & e & 0.95 \\ a & b & \neg e & 0.94 \\ a & \neg b & e & 0.29 \\ a & \neg b & \neg e & 0.001 \\ \neg a & b & e & 0.05 \\ \neg a & b & \neg e & 0.06 \\ \neg a & \neg b & e & 0.71 \\ \neg a & \neg b & \neg e & 0.999 \end{bmatrix} \cdot \begin{bmatrix} a & b & e & 0.002 \\ a & b & \neg e & 0.998 \\ a & \neg b & e & 0.002 \\ a & \neg b & \neg e & 0.998 \\ \neg a & b & e & 0.002 \\ \neg a & b & \neg e & 0.998 \\ \neg a & \neg b & e & 0.002 \\ \neg a & \neg b & \neg e & 0.998 \end{bmatrix} = \begin{bmatrix} a & b & e & 0.0019 \\ a & b & \neg e & 0.93812 \\ a & \neg b & e & 0.00058 \\ a & \neg b & \neg e & 0.000998 \\ \neg a & b & e & 0.0001 \\ \neg a & b & \neg e & 0.05988 \\ \neg a & \neg b & e & 0.00142 \\ \neg a & \neg b & \neg e & 0.997002 \end{bmatrix}$$

Since we will sum on E:

$$\sum_E P(A \mid B, E)P(E) = \begin{bmatrix} a & b & 0.94002 \\ a & \neg b & 0.001578 \\ \neg a & b & 0.05998 \\ \neg a & \neg b & 0.998422 \end{bmatrix}$$

We can clearly see that the probability of a b and $\neg a$ b is alost 1, the probability of a $\neg b$ and $\neg a \neg b$ is also almost 1, so we can say $\sum_E P(A \mid B, E)P(E) = P(A \mid B)$, and simpilify Eq. 12 into:

$$P(B \mid J, M) = \alpha * P(B) * \Big(\sum_A P(J \mid A) * P(M \mid A) * P(A \mid B)\Big) \tag{13}$$

we will use the same trick to expand all three elements by adding two extra dimension for multiplication, for example:

$$P(J \mid A) = \begin{bmatrix} b & m & j & a & 0.9 \\ \neg b & m & j & a & 0.9 \\ b & \neg m & j & a & 0.9 \\ \neg b & \neg m & j & a & 0.9 \\ b & m & j & \neg a & 0.05 \\ ... & ... & ... & ... & ... \\ \neg b & \neg m & \neg j & \neg a & 0.95 \end{bmatrix}$$

since we will sum on A, the product will looks like this after the summation:

$$\sum_A P(J \mid A) * P(M \mid A) * P(A \mid B) = \begin{bmatrix} j & m & b & 0.592243 \\ j & \neg m & b & 0.25677441 \\ \neg j & m & b & 0.06637121 \\ \neg j & \neg m & b & 0.08461179 \\ j & m & \neg b & 0.001493351 \\ j & \neg m & \neg b & 0.049847949 \\ \neg j & m & \neg b & 0.009595469 \\ \neg j & \neg m & \neg b & 0.939063231 \end{bmatrix}$$

which is the first four rows summation is around 1, the last four rows summation is also one, so we can say this sum is equal to $P(J, M \mid B)$, we will simply the Eq. 13 one more step:

$$P(B \mid J, M) = \alpha * P(B) * P(J, M \mid B) \tag{14}$$

Here we will expend dimension for probabiliy of B to multipy with $P(J, M \mid B)$, which is:

$$\begin{bmatrix} j & m & b & 0.001 \\ j & \neg m & b & 0.001 \\ \neg j & m & b & 0.001 \\ \neg j & \neg m & b & 0.001 \\ j & m & \neg b & 0.999 \\ j & \neg m & \neg b & 0.999 \\ \neg j & m & \neg b & 0.999 \\ \neg j & \neg m & \neg b & 0.999 \end{bmatrix} \cdot \begin{bmatrix} j & m & b & 0.592243 \\ j & \neg m & b & 0.25677441 \\ \neg j & m & b & 0.06637121 \\ \neg j & \neg m & b & 0.08461179 \\ j & m & \neg b & 0.001493351 \\ j & \neg m & \neg b & 0.049847949 \\ \neg j & m & \neg b & 0.009595469 \\ \neg j & \neg m & \neg b & 0.939063231 \end{bmatrix} = \begin{bmatrix} j & m & b & 0.000592243 \\ j & \neg m & b & 0.000256774 \\ \neg j & m & b & 6.63712E-05 \\ \neg j & \neg m & b & 8.46118E-05 \\ j & m & \neg b & 0.001491858 \\ j & \neg m & \neg b & 0.049798101 \\ \neg j & m & \neg b & 0.009585874 \\ \neg j & \neg m & \neg b & 0.938124168 \end{bmatrix}$$

Now we can see, for each case of J and M, the summation over B is not equal to one any more, so we will do the normalization over them, for example:

$$P(j, m, b) + p(j, m, \neg b) = 0.000592243 + 0.001491828 = 0.002084 \neq 1$$

$$\alpha * P(j, m, b) = \frac{P(j, m, b)}{P(j, m, b) + p(j, m, \neg b)} = 0.284172$$

So, the final answer should be:

$$P(B \mid J, M) = \begin{bmatrix} j & m & b & 0.284171835 \\ j & \neg m & b & 0.005129858 \\ \neg j & m & b & 0.006876246 \\ \neg j & \neg m & b & 9.01844E-05 \\ j & m & \neg b & 0.715828165 \\ j & \neg m & \neg b & 0.994870142 \\ \neg j & m & \neg b & 0.993123754 \\ \neg j & \neg m & \neg b & 0.999909816 \end{bmatrix}$$

Now, given the evidence of J = True, M = True, we can confidently say that,

$$P(B \mid J, M) = 0.284171835$$

5 Bayesian Neural Network

There is real world adversail attack for traditional machine leanrning, and 'there is no free lunch', traditional neural network all face the problem of over-confidence, 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:

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

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:

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z) p_\theta(z)}{p_\theta(x)} \tag{15}$$

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.

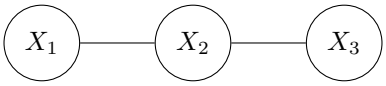
To overcome the intractable integral problem, mathematicians develop several ways to avoid the calculation of this "normalizing constant", here we will present two ways after we transform our problem into a simpler one for notation:

$$p(x) = \frac{f(x)}{NC} \tag{16}$$

5.1 Markov Chain Monte Carlo simulation

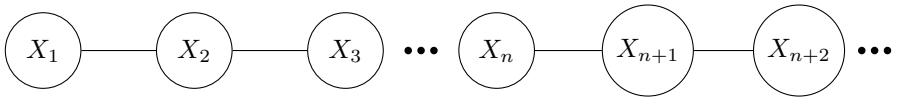
5.1.1 what is Markov Chain MC

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.

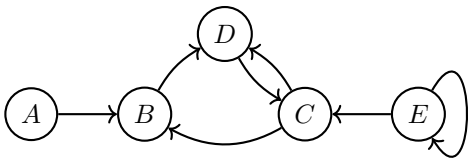


As shwon in the above graph, after the "burn-in" process of X_1,X_2,X_3 ...We will have X_n, X_{n+1}, X_{n+2} that stay on our target distribution p(x).

Why this make sense consider our case, the parameters of the model will be what we focus on, so in each state, we will have different parameters as the variables that we will update and simulate. For linear regression problem, we will have:

5.1.2 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:

	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.

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:

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

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:

$$p(x)T(y \mid x) = p(y)T(x \mid y) \tag{17}$$

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.

Why this make sense 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 $\{\theta_1^*, \theta_2^*, ..., \theta_n^*\}$

5.1.3 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:

$$\frac{A(x \to y)}{A(y \to x)} = \frac{f(y) \cdot g(x \mid y)}{f(x) \cdot g(y \mid x)} \tag{18}$$

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:

Algorithm 1 Basic Metropolis-Hasting Algorithm

```
x_0 = [0, 0, ..., 0]
while t < N do
    u ~ U(0, 1)
    x_{t+1} ~ g(x_t, σ^2)
    if u ≤ A(x_{t+1} | x_t) = min(1, rfrg) then
        x_{t+1} = x_{t+1}
    else
        x_{t+1} = x_t
    end if
end while
```

Why this make sense we will have a reflection part at the end of each section to help with understanding and connect between each techniques. Here 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:

$$p(\theta \mid x) = \frac{p(x \mid \theta)p(\theta)}{p(x)} \tag{19}$$

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:

$$\alpha = min(1, \frac{f(y) \cdot g(x \mid y)}{f(x) \cdot g(y \mid x)}) \tag{20}$$

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:

$$\alpha = min(1, \frac{f(y)}{f(x)}) \tag{21}$$

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.

5.1.4 Hamiltonian Monte Carlo

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:

$$H(x, p) = U(x) + K(p) \tag{22}$$

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

$$\frac{dx}{dt} = \frac{\partial H}{\partial p} = \frac{\partial K(p)}{\partial p} \tag{23}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial x} = -\frac{\partial U(x)}{\partial x} \tag{24}$$

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:

$$K(p) = \frac{p^T p}{2} \tag{25}$$

$$U(x) = -log\, \pi(x) \tag{26}$$

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:

$$p(t + \delta/2) = p(t) - (\delta/2)\frac{\partial U}{\partial x(t)} \tag{27}$$

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

$$x(t+\delta) = x(t) + \delta \frac{\partial K(p)}{\partial p(t+\delta/2)} \tag{28}$$

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

$$p(t+\delta) = p(t+\delta/2) - (\delta/2)\frac{\partial U}{\partial x(t+\delta)} \tag{29}$$

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 $E(\theta)$:

$$E(\theta) = H(x,p) = U(x) + K(p) \tag{30}$$

the canonical distribution over states has PDF as:

$$q(\theta) = \frac{1}{Z} e^{-E(\theta)} \tag{31}$$

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:

$$\begin{aligned} q(x,p) &\propto e^{-H(x,p)} \\ &= e^{-[U(x)+K(p)]} \\ &= e^{-U(x)} + e^{-K(p)} \\ &\propto q(x)q(p) \end{aligned} \tag{32}$$

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:

$$q(x,p) \propto e^{-[U(x)+K(p)]} \tag{33}$$

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:

$$q(x^*,p^*) \propto e^{-[U(x^*)+K(p^*)]} \tag{34}$$

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:

$$A = min(1, e^{-U(x^*)-K(p^*)+U(x)+K(p)}) \tag{35}$$

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

Algorithm 2 Basic Hamiltonian Monte Carlo Algorithm

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

Why this make sense 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.

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

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)**

5.1.5 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.

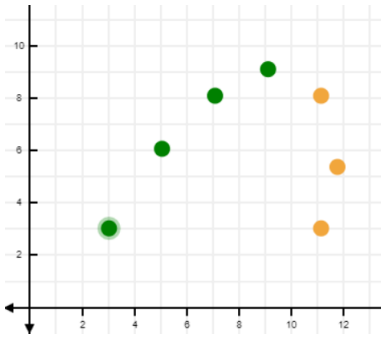


Figure 4: The dots indicate the position of each state of a certain particle in a 2D dimensional space.

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.

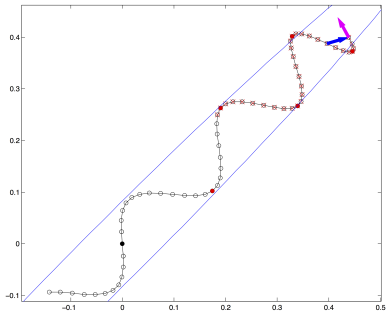


Figure 5: NUTS trajectory termination and sample selection. (Figure from Hoffman and Gelman, 2014.)

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

$$\frac{d}{dt}\frac{(\tilde{\theta}-\theta)\cdot(\tilde{\theta}-\theta)}{2} = (\tilde{\theta}-\theta)\cdot\frac{d}{dt}(\tilde{\theta}-\theta) = (\tilde{\theta}-\theta)\cdot\tilde{r} < 0 \tag{36}$$

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.

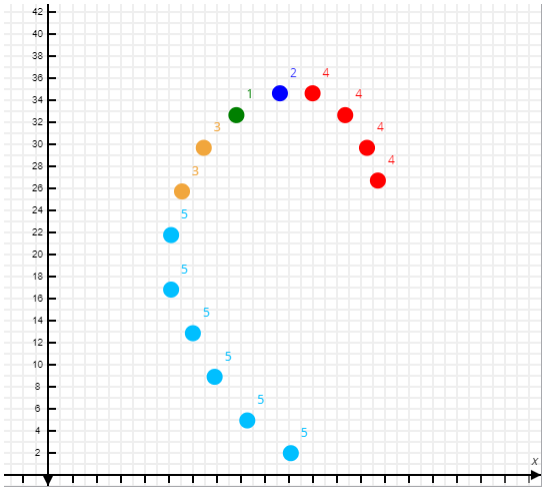
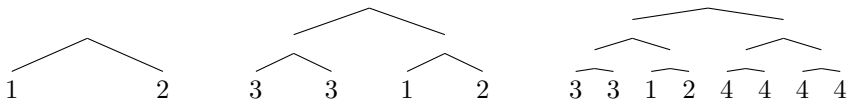


Figure 6: The number indicating how many steps there will be, we start from green 1, and then forward one step to blue 2, backward 2 step to 3, forward 4 steps into red 4, and then backward 8 steps into 5 (we didn't include all 8 of the blue in the graph)

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.

5.1.6 Example

You can find the code here:

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.

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. Since most of the exmaple or papers are using manually generated data and results in a higher accuracy and less meaning actual implication of the model. Here we will use classic Red Wine dataset.

Here is the structure of the algorithm:

Here is the result of the parameters:

Here is the result of the results:

5.2 Variational Inference

Traditional methods are expensive and will use approximate inference. So, we introduce variational inference distribution $q_\phi(x \mid z)$ with new sets of parameters ϕ that trying to remedy this issue.
base on the graph, our optimazation goal switch to:

$$q_\phi(z \mid x) \approx p_\theta(z \mid x)$$

and to get two distribution closer and closer, we will use KL-divergence as the loss, which is derived from Shannon Entropy as:

$$D_{KL}(q_\phi, p_\theta) = \mathbb{E}_{q_\phi}\left[log\frac{q_\phi(x)}{p_\theta(x)}\right] \tag{37}$$

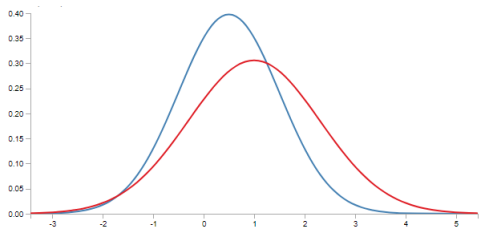


Figure 7: if the red distribution is the not calculable generative model p_θ, then we will introduce variational inference q_ϕ that try to go as close to p_θ as possible.

plug in our parameters and we can see we still need to calculate the posteior $p_\theta(z\ midx)$, so we will make some simplifications and factorization:

$$\begin{aligned} D_{KL}(q_\phi, p_\theta) &= \mathbb{E}_{q_\phi}\left[log\frac{q_\phi(z \mid x)}{p_\theta(z \mid x)}\right] \\ &= \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] - \mathbb{E}_{q_\phi}\left[log\, p_\theta(z \mid x)\right] \\ &= \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] - \mathbb{E}_{q_\phi}\left[log\frac{p_\theta(z,x)}{p_\theta(x)}\right] \\ &= \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] - \mathbb{E}_{q_\phi}\left[log\, p_\theta(z,x)\right] + \mathbb{E}_{q_\phi}\left[log\, p_\theta(x)\right] \end{aligned}$$

if we consider the third term in Eq. 24 as the expectation of z, then we can explain the expectation as integral:

$$\begin{aligned} D_{KL}(q_\phi, p_\theta) &= \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] - \mathbb{E}_{q_\phi}\left[log\, p_\theta(z,x)\right] + \int q_\phi(z \mid x)\, log\, p_\theta(x)\, dz \\ &= \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] - \mathbb{E}_{q_\phi}\left[log\, p_\theta(z,x)\right] + log\, p_\theta(x) \int q_\phi(z \mid x)\, dz \\ &= \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] - \mathbb{E}_{q_\phi}\left[log\, p_\theta(z,x)\right] + log\, p_\theta(x) \end{aligned}$$

the final step is possible because the integration of density function will equal to 1.$log\, p_\theta(x)$ is the marginal log likelihood, we will rearrange the equation for anaylsis:

$$log\, p_\theta(x) = D_{KL}(q_\phi, p_\theta) + \mathbb{E}_{q_\phi}\left[log\, p_\theta(z,x)\right] - \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] \tag{38}$$

due to the non-negative of the KL-divergence, we can say:

$$log\, p_\theta(x) \geq \mathbb{E}_{q_\phi}\left[log\, p_\theta(z,x)\right] - \mathbb{E}_{q_\phi}\left[log\, q_\phi(z \mid x)\right] \tag{39}$$

everything on the right hand side of the inequality is call the ELBO, evidence lower bound $Ł_{\theta,\phi(x)}$, since this is the lower bound of the marginal likelihood $p_\theta(x)$, which is also known as the evidence:

$$Ł_{\theta,\phi(x)} = \mathbb{E}_{q_\Phi(z|x)}\left[log\, p_\theta(x,z) - log\, q_\Phi(z \mid x)\right] \tag{40}$$

recall Eq. 17, if we want to minimize the KL-divergence, the it will be equvalent to maximize the ELBO, which is computable with the help of Stochastic Variational inference technique.
we have the loss functionl, we have the feed forward function, now let's begin to work on the back-propagation and find the gradient like normal neural network.

$$\nabla_\theta L_{\theta,\Phi}(x) = \nabla_\theta\, \mathbb{E}_{q_\Phi(z|x)}\left[logp_\theta(x,z) - logq_\Phi(z \mid x)\right] \tag{41}$$
$$\nabla_\phi L_{\theta,\Phi}(x) = \nabla_\phi\, \mathbb{E}_{q_\Phi(z|x)}\left[logp_\theta(x,z) - logq_\Phi(z \mid x)\right] \tag{42}$$

considering Leibniz integral law to push the derivative into the expectation:

$$\frac{d}{dx}\int_{a(x)}^{b(x)} f(x,t)dt = f(x,b(x))\frac{db(x)}{dx} - f(x,a(x))\frac{da(x)}{dx} + \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,t)dt$$

considering Leibniz integral law to push the derivative into the expectation:

$$\begin{aligned}\nabla_\theta L_{\theta,\Phi}(x) &= \mathbb{E}_{q_\Phi(z|x)}\left[\nabla_\theta(logp_\theta(x,z) - logq_\Phi(z \mid x))\right] \\ &= \mathbb{E}_{q_\Phi(z|x)}\left[\nabla_\theta(logp_\theta(x,z))\right] \\ &= \nabla_\theta(logp_\theta(x,z))\end{aligned}$$

using Monte Carlo estimation at the last step can help us find the gradient of generative model's gradient.
But considering Eq. 21, we are taking derivative on the same parameter sets as the expectation, which will be in the integral, $\nabla_\phi \mathbb{E}_{q_\Phi(z|x)}$ that cause intervene. so we need to use reparameterization trick to avoid the conflict. Let's restate our problem now by simplifying it first, assume we have our loss function summarized as $f(x)$, and we will do the expectation over $x \sim p_\theta(x)$, and find the gradient of it for backpropagation:

$$\nabla_\theta \mathbb{E}_{p_\theta(x)}\left[f(x)\right] = \nabla_\theta \int_\theta f(x)p_\theta(x)dx \tag{43}$$

what block us here is we are using the distribution of θ, as well as taking derivative on it, so we propose to use a new variable $\epsilon \sim N(0,1)$ to replace x, if x is continuos and $x \sim N(\mu,\sigma)$:

$$\begin{aligned}&x = p_\theta(x) \\ &\theta = \mu, \sigma \\ &x = g(\epsilon,\theta) \quad where\, \epsilon \sim p(\epsilon)\end{aligned} \tag{44}$$

Following the Location Scale Transformation Technique, we can be more specific about the x, given the distribution ϵ:

$$x = \mu + \sigma \cdot \epsilon \quad where\, \epsilon \sim p(\epsilon)$$

Now, let's reconsider Eq. 22 with the change of variable trick of Eq. 23, we will replace x with $g(\epsilon,\theta)$, and most importantly, we will switch our expectation from θ to ϵ, since we are using ϵ as our base variable with a known distribution $N(0,1)$:

$$\begin{aligned}\nabla_\theta \mathbb{E}_{p_\theta(x)}\left[f(x)\right] &= \nabla_\theta \mathbb{E}_{p(\epsilon)}\left[f(g(\epsilon,\theta)\right] \\ &= \nabla_\theta \int_\epsilon f(g(\epsilon,\theta))p_\theta(x)dx \\ &= \int_\epsilon \nabla_\theta f(g(\epsilon,\theta))p_\theta(x)dx \\ &= \mathbb{E}_{p(\epsilon)}\left[\nabla_\theta f(g(\epsilon,\theta))\right]\end{aligned} \tag{45}$$

That's all for reparameterization trick, it is widely used in different aspect of math and statistic to break down a hard problem into easy problems, for example, if we want to calculate the roots for $x^6 + 2x^3 + 1 = 0$, then we can use $u = x^3$ to change variable and solve $u^2 + 2u + 1 = 0$ instead.
Now let's implement this trick into our ELBO when finding the partial derivatives on ϕ by restate our problem:

$$\nabla_\phi L_{\theta,\Phi}(x) = \nabla_\phi \mathbb{E}_{q_\Phi(z|x)}\left[logp_\theta(x,z) - logq_\Phi(z \mid x)\right]$$

Our problem is what we are expecting z, which $z \sim N(\mu,\sigma)$ conflict with what we are differentiating on ϕ, which is $\phi = \{\mu,\sigma\}$. we will change variable on z with ϵ that $\epsilon \sim p(\epsilon)$ with known distribution $\epsilon \sim N(0,1)$:

$$z = g(\phi, x, \epsilon)$$

Our old expectation was $\frac{1}{N}\sum_{i=1}^{N}(f(z_i))$, which is also unknown due to stochasticity $N(\mu,\sigma)$, now we can switch it to expectation of ϵ, $\frac{1}{N}\sum_{i=1}^{N}(f(\epsilon_i))$ with know distribution $N(0,1)$, and use Monte Carlo

estimator to get rid of the expectation, but looking at one datapoint or minibatch SGD:

$$
\begin{aligned}
&L_{\theta,\Phi}(x) = \mathbb{E}_{p(\epsilon)}\left[log\, p_\theta(x,z) - log\, q_\Phi(z \mid x)\right] \\
&\tilde{L}_{\theta,\phi}(x^i) = \frac{1}{L}\sum_{l=1}^{L} log\, p_\theta(x^i, z^{(i,l)}) - log\, q_\Phi(z^{(i,l)} \mid x^i) \\
&where \quad z^{(i,l)} = g(\phi, \epsilon^{(i,l)}, x^i) \quad and \quad \epsilon^l \sim p(\epsilon) \quad and \quad x^i \sim x
\end{aligned}
\tag{46}
$$

This is also know as Stochastic Gradient Variational Bayes estimator, SGVB. And now we can propagate the deriviation into the integral and continue our calculation.

5.3 Bonus: Variational AutoEncoder

Variational AutoEncoder (VAE) is very similar to to variational inference model, actually VI is exactly the first part of the VAE model. According to the diagram, we can see the encoding part is exactly the structure of VI.