Markov Chain Monte Carlo Methods

Chinmay Talegaonkar | 15D070046 | *IIT Bombay*

***Abstract*—Markov Chain Monte Carlo methods are an effective class of algorithms for modelling complicated distributions statistically. It is specially useful for estimating posterior distributions of Bayesian inferences. We start with a brief overview of Monte Carlo Integration and Markov Chains. This is followed by a description of the Metropolis Hastings algorithm, with some of its technicalities. We conclude with an overview of the future work planned for this project**

***Keywords*—Markov Chain Monte Carlo, Bayesian Inference, Metropolis Hasting**

I. Introduction

A. Bayesian Inference

The fundamental principle of bayesian statistics is that both the observables and parameters of a statistical model are random quantities. Let U be the observed data, and θ be the parameter(s) associated with the model. Assuming we have a prior distribution of θ given by $P(\theta)$. Since we are treating both θ and U as random quantities. Bayesian inference deals with the problem of estimating a posterior distribution of θ, $P(\theta|U)$, i.e updating parameters based on observations. This can be done using Bayes' rule

$$P(\theta|U) = \frac{P(U|\theta)P(\theta)}{\int P(U|\theta)P(\theta)d\theta}$$

B. Monte Carlo Integration

It is used to evaluate definite integrals non-deterministic-ally by sampling random points from a distribution, and computing an approximate value of the integral.

$$\int_\Omega f(x)dx \approx V\frac{1}{N}\sum_{i=1}^{N} f(x_i) = V\langle f\rangle$$

It is specifically useful for finding higher dimensional integrals, which are very common for multi parameter distributions. N is the number of samples taken in Ω, which is the sample space over which we integrate the function $f(x)$. V is the volume of Ω, i.e. $\int_\Omega dx$.

C. Markov Chains

Let $\{X_1, X_2, ...X_t\}$ be a sequence of random variables generated such that for $t \geq 0$, X_{t+1} is sampled from the distribution $P(X_{t+1}|X_t)$. This effectively means that X_{t+1} depends only on the last generated variable X_t. This is called a markov chain. Due to regularity conditions (large sequences), for some markov chains the distribution $P_t(.|X_0)$ converges to a stationary distribution.The next section explains the idea of Markov Chain Monte Carlo methods, for **Bayesian Inference** of posterior distribution parameters. Converging to this stationary distribution depends on the starting point of the Markov Chain. The next section explains the idea of Markov Chain Monte Carlo methods, for **Bayesian Inference** of posterior distribution parameters.

II. Markov Chain Monte Carlo

The denominator of the expression used in Bayes rule, is quite complicated to compute, hence Monte Carlo Integration is useful in evaluating the integral. Since we update parametres sequentially, i.e. using θ_t in the Bayes rule, to find θ_{t+1}, we need to draw samples from a Markov Chain (the stationary distribution of the chain), and then perform Monte Carlo integration, for estimating the posterior Bayesian distribution. This is simply the monte carlo integration of a Markov Chain, hence the name **Markov Chain Monte Carlo**. We now look at one of the MCMC algorithms, known as the Metropolis-Hastings algorithm.

A. The Metropolis Hastings Algorithm

It is used to draw samples from a distribution where direct sampling is difficult. To sample from the target distribution $f(x)$, a distribution $p(x)$ which is proportional to $f(x)$ also suffices. The algorithm is given on the top right of the page. A sample case for this algorithm will be demonstrated in the presentation.

Algorithm 1 Metropolis-Hasting

```
1) Choose an initial value $x_0$
2) Draw $x^*$ from a known proposal distribution $q(x^*|x_0)$
3) Sample $u$ from a uniform $[0,1]$ distribution
4) if $u \leq min\left\{1, \frac{p(x^*)q(x|x^*)}{p(x)q(x^*|x)}\right\}$, $x_1 = x^*$,
   else $x_1 = x_0$
5) Repeat from step 2, with $x_1$ as the initial value.
6) Iterate $N$ times to generate a sequence of samples
   $\{x_0, x_1...x_N\}$ from the target distribution $f(x)$
```

$q(x)$ is called the proposal distribution, and essentially decides the next point we sample. Since, x^* is drawn from the conditional distribution depending on the previous sample, it's a markov chain. Also, since we see that the probability of selecting x^* depends on the ratio, $\frac{p(x^*)}{p(x)}$, the exact target distribution $f(x)$ is not necessary for this algorithm to work.

B. Technicalities of Metropolis Hastings

- **Initial Value:** Starting points closer to the mode of the target distribution generally give better results, due to faster convergence. The initial samples obtained from the simulation are dependent heavily on x_0 and won't give the correct estimate of $f(x)$, hence need to eliminated (*burn in*), for accurate results and convergence

- **Burn in:** Due to the randomness of the initial point, the initial samples obtained from the simulation concentrated in the low density areas of $p(x)$, and wont give the correct estimate of $f(x)$, hence need to eliminated (burnt in), for a better estimate of $f(x)$.
- **Proposal Distribution:** In theory, any proposal distribution will lead to a convergence to the target distribution. But, for good convergence rates (specially in higher dimensions) it is necessary to determine the shape and orientation of the proposal distribution. The width of the proposal distribution, (e.g. variance of a normal distribution) also affects the performance of the algorithm. When $q(x)$ is symmetric, i.e. $q(x|y) = q(y|x)$, we get a special case of Metropolis Hasting algorithm, known as the **Metropolis algorithm**. Likewise, the **independent sampler** uses an independent proposal distribution, i.e. $q(x|y) = q(x)$. For independent sampler to work well, $q(x)$ should be a very close approximation of $f(x)$.

III. Gibbs sampling

Gibbs sampling is a Markov chain Monte Carlo (MCMC) technique for getting a sequence of observations of different random variables, which are sampled from a specified multivariate probability distribution, for which direct sampling is not very feasible. The idea behind Gibbs sampling is that given a multivariate distribution it is simpler to sample from a conditional distribution than to marginalize by integrating over a joint distribution. consider a system, which requires sampling k samples of $\mathbf{X} = (x_1, x_2, \dots x_n)$ from a joint distribution $\mathbf{p} = (p_1, p_2, \dots p_n)$. We denote the i$^{\text{th}}$ sample by $X^{(i)} = (x_1^{(i)}, x_2^{(i)}, x_3^{(i)} \dots x_n^{(i)})$ The algorithm for Gibbs sampling is as follows -:

Algorithm 2 Gibbs sampling

```
1) Choose an initial value $\mathbf{X^{(i)}}$
2) To draw the next sample which is denoted as: $\mathbf{X^{(i+1)}} = (x_1^{(i+1)}, x_2^{(i+1)}, \dots x_n^{(i+1)})$, sample each component of the above vector, $x_j^{(i+1)}$ from the distribution of that component conditioned on all the components sampled before it so far. Sampling of the components is always done in order to maintain uniformity. Hence, $x_j^{(i+1)}$ is sampled from the distribution $p(x_j^{(i+1)} \mid x_1^{(i+1)}, \dots x_{j-1}^{(i+1)}, x_{j+1}^{(i)}, \dots x_n^{(i)})$
3) Repeat this for all components of the vector.
4) Repeat the entire procedure $k$ times.
```

If Gibbs sampling is performed, the resulting samples approximate the joint distribution of all variables. The marginal distribution of any subset of variables can be approximated by simply considering the samples for that subset of variables and ignoring the rest. The expected value of any variable can be approximated by averaging over all the samples. Also, like the Metropolis-Hastings algorithm, the first few samples are not taken into account (burn in period). There is a tendency to move slowly around the sample space, with a high amount of auto correlation between samples, rather than moving around quickly, as is desired. **Blocked Gibbs Sampling** is one technique used, to reduce this auto correlation in the samples. In a blocked Gibbs sampler, two or more variables are grouped together and samples from their joint distribution are obtained, conditioned on all other variables. Blocked Gibbs sampling is used extensively in learning parameters of statistical models, an example of which we shall see ahead.

A necessary requirement for Gibbs sampling to work, is the availability of full conditional distributions. If those are not available, some other MCMC algorithm should be used instead for sampling.

IV. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo method is a variation of the Markov chain monte carlo method, which uses the **Boltzmann energy distribution** function

$$p(\theta) = \frac{1}{Z} e^{-E(\theta)}$$

as a proposal distribution. The Energy $E(\theta)$ is defined over a set of variables, and Z is the partition function, in other words a normalizing constant. Before delving into the algorithm, here is a small insight on Hamiltonian dynamics, which describe the evolution of an object or a variable in a system.

A. Hamiltonian dynamics

In classical physics, the total energy of a system is defined as a sum of Potential energy and Kinetic energy, which is defined as the Hamiltonian of the system, given as follows:

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

The potential energy $U(\mathbf{x})$ is a function of distance, and the kinetic energy $K(\mathbf{p})$ is a function of the momentum. The evolution of the kinetic and potential energies in a system as functions of $\mathbf{x}$ and $\mathbf{p}$ is given by the equations -:

$$\frac{\partial x_i}{\partial t} = \frac{\partial H}{\partial p_i} = \frac{\partial K(\mathbf{p})}{\partial p_i}$$
$$\frac{\partial p_i}{\partial t} = -\frac{\partial H}{\partial x_i} = -\frac{\partial U(\mathbf{x})}{\partial x_i}$$

Using a set of initial conditions along with these equations, it is possible to predict the location and momentum of an object at any point in time by simulating these dynamics for a duration δ . For simulating the Hamiltonian dynamics in a discrete way, we use the **leap frog method**, which divides the total time T into small quanta of duration δ.

The Leap Frog method updates the momentum and position variables sequentially, starting by simulating the momentum dynamics over a small interval of time $\frac{\delta}{2}$, then simulating the position dynamics over a slightly longer interval in time δ, then completing the momentum simulation over another small interval of time $\frac{\delta}{2}$ so that x and p now exist at the same point in time. The differential equations relating $U(\mathbf{x})$ and $K(\mathbf{p})$ with x and p are used for updates in the leap frog method.

Algorithm 3 Leap Frog Method

```
1) Take a half step in time to update momentum
2) Take a full step in time to update $x$
3) Take the remaining half step in time to finish updating
   the momentum variable, $p$
4) Repeat the entire procedure for $L$ steps to simulate
   dynamics over $L \times \delta$ time units.
```

B. *Hamiltonian Monte Carlo*

The main idea behind Hamiltonian/Hybrid Monte Carlo is to develop a Hamiltonian function such that the resulting Hamiltonian dynamics allow us to efficiently explore some target distribution . It turns out it is pretty simple to relate a to using a basic concept adopted from statistical mechanics known as the canonical/Boltzmann distribution, defined in the introduction of this section. The Energy function is same as the Hamiltonian function. The canonical distribution function, can be expressed in terms of the Hamiltonian as follows -:

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

Since the joint distribution for x and p factorizes, this means that the two variables x and p are independent. Hence, p is an auxiliary variable that facilitates the Markov chain path. Introducing the auxiliary variable p allows us to use Hamiltonian dynamics, which is not possible otherwise. Since the canonical distribution for x is independent of the canonical distribution for p, we can choose any distribution from which to sample the momentum variables. A common choice is to use a zero-mean Normal distribution with unit variance. The kinetic energy function can be defined as $K(\mathbf{p}) = \frac{\mathbf{p}^T\mathbf{p}}{2}$ This is a convenient choice, since partial derivatives are easy to compute. To estimate $p(\mathbf{x})$ using potential energy, we define the potential energy as $U(\mathbf{x}) = -\log p(\mathbf{x})$

Using Hamiltonian dynamics as a proposal function for a Markov Chain helps in exploring the target (canonical) density defined by more efficiently than using a Gaussian or likewise proposal probability distribution. The precise algorithm is shown on top of the next half of the page

We shall see the simulation results in the presentation, that convergence in HMC is much quicker than metropolis when the initial point is taken far away from the mode of the distribution. This implies that mixing occurs faster in HMC than in the metropolis algorithm. The next section talks about an interesting application of MCMC in a state of the art deep learning model known as the **Restricted Boltzmann Machine**. It is used extensively for classification problems in machine learning.

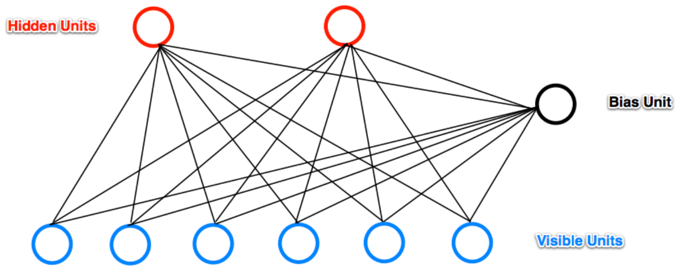
Algorithm 4 Hamiltonian Monte Carlo

```
1) Set $t = 0$
2) Generate an initial position state $\mathbf{x}^{(0)}$
3) repeat until $t = M$ where $M$ is the number of samples
   required.
   a) set $t = t + 1$
   b) Sample a new initial momentum variable from
      the canonical distribution for the momentum
      $\mathbf{p}_0\tilde{p}(\mathbf{p})$
   c) Set $\mathbf{x}_0 = \mathbf{x}^{(t-1)}$
   d) Run Leap Frog algorithm starting at $[\mathbf{x}_0, \mathbf{p}_0]$ for
      $L$ steps and stepsize $\delta$ to obtain proposed states
      $\mathbf{x}^*$ and $\mathbf{p}^*$
   e) Calculate the Metropolis acceptance probability:
      $\alpha = min(1, exp(-U(\mathbf{x}^*) + U(\mathbf{x}_0) - K(\mathbf{p}^*) + K(\mathbf{p}_0)))$
   f) Set the next state in the Markov chain $\mathbf{x}^{(t)} = \mathbf{x}^*$
      with a probability $\alpha$, else set it to the current
      value, i.e. $\mathbf{x}^{(t-1)}$
```

V. Restricted Boltzmann machines

Restricted Boltzmann Machines are stochastic neural network consisting of:

- One layer of visible units (users movie preferences whose states we know and set);
- One layer of hidden units (the latent factors we try to learn);
- A bias unit (whose state is always on, and is a way of adjusting for the different inherent popularity of each visible unit)



Furthermore, each visible unit is connected to all the hidden units (this connection is undirected, so each hidden unit is also connected to all the visible units), and the bias unit is connected to all the visible units and all the hidden units. To make learning easier, we restrict the network so that no visible unit is connected to any other visible unit and no hidden unit is connected to any other hidden unit. This helps in minimizing the energy (or cost function of the network). The aim of the RBM is to learn the hidden units, based on the information about visible units known to us, and then reconstruct the visible units based on the inferred behaviour of the hidden units. The difference in the original data for the visible units, and that between the reconstructed data using the hidden units is the error RBM minimizes. The RBM learns by changing the weights of the edges between the visible and hidden units.

A. Learning Weights

Assume we have binary vectors of length K as inputs for the visible units (number of visible units is K). Let m be the number of inputs given. We first estimate the probabilities of the hidden layer units, using an activation function. This is called the forward phase. The RBM then tries to reconstruct back the units of the visible layer, this is called the negative phase. The learning is done by updating the edge weights connecting the hidden and visible units based on the difference in the positive and negative phase. We consider a conditional Bernoulli distribution for modelling the hidden features $\mathbf{h}$. The conditional probability of $\mathbf{h}$ given the observed visible units is given by:

$$p(h_j = 1|\mathbf{V}) \quad = \quad \sigma(b_j + \sum_{i=1}^{m}\sum_{k=1}^{K} v_i^k W_{ij}^k)$$

Here, v_i^k is 1 if the movie i is rated k by the user. Similarly, while in the negative phase, the conditional probability of reconstructing the visible units is given by

$$p(v_i^k = 1|\mathbf{h}) \quad = \quad \frac{\exp\left(b_i^k + \sum_{j=1}^{F} h_j W_{ij}^k\right)}{\sum_{l=1}^{K} \exp\left(b_i^l + \sum_{j=1}^{F} h_j W_{ij}^l\right)}$$

σ is the sigmoid activation function.

Given this model of the RBM, we shall now see the algorithm for updating the edge weights of the RBM -:

Algorithm 5 Learning Weights in an RBM

```
1) Based on the input vector, set the states of the visible unit layer.
2) Update the states of the hidden units using the sigmoid activation function described before. Set the hidden unit $h_j$ to 1 according to $p(h_j = 1|\mathbf{V})$
3) For each edge $e_{ij}$ compute $Positive(e_{ij}) = x_i \times x_j$, i.e. it is 1 if both the units $i$ and $j$ are 1 (on).
4) Now reconstruct the visible units in a similar manner using the equation for $p(v_i^k = 1|\mathbf{h})$. Then update the hidden units again, and compute $Negative(e_{ij}) = x_i \times x_j$ for each edge.
5) Update the weight $w_{ij}$ for each edge $e_{ij}$ by the rule: $w_{ij} = w_{ij} + L \times (Positive(e_{ij}) - Negative(e_{ij}))$, where $L$ is the learning rate
6) Repeat for all inputs
```

B. Use of MCMC in RBMs

The hidden units, are not observed from the data and are generally marginalized over to obtain the likelihood of the observed data, i.e.

$$p(v;\theta) = \sum_h p(v;\mathbf{h},\theta)$$

where $\sum_h p(v;\mathbf{h},\theta)$ is the joint probability distribution over the visible and hidden units, based on the current model parameters θ of the RBM. The RBM models the joint probability distribution as follows -:

$$p(v,h;\theta) \quad = \quad \frac{\mathrm{e}^{-E(v,h;\theta)}}{Z(\theta)}$$

Here E is the energy function of RBM, and $Z(\theta)$ is the partition function which the sum of the numerator term over all possible values of v and h. The common way to train the Boltzmann machine is to determine the parameters that maximize the likelihood of the observed data. To determine the parameters, we perform gradient descent on the log of the likelihood function $l(v;\theta) = \log p(v)$. The gradient after a some rigorous calculations can be approximated as follows -:

$$\frac{\partial l(v;\theta)}{\partial \theta} \approx \left\langle \frac{\partial E(v,h)}{\partial \theta} \right\rangle_{p(h_{data}|v_{data})} + \left\langle \frac{\partial E(v,h)}{\partial \theta} \right\rangle_{p(h_{model}|v_{model})}$$

Hence, the derivative has the sum of expectations of 2 terms. The first one can be simply found out by averaging the value of the energy function gradient when the visible and hidden units are being driven by observed data samples. Calculating the second term is generally more complicated and involves blocked Gibbs sampling to estimate the expectation. The second term arises due to the negative phase while training the RBM. Without MCMC, it is almost impossible to compute this term, specially for a large number of units. Gibbs sampling proves to be effective, since we want to estimate the expectation of a multivariate distribution. Metropolis sampling is used in several other areas, but in context of deep learning, Gibbs sampling is more widely used. Intractable Partition functions Building up on the conclusion we arrived the previous section, we now examine the problem of the negative phase of RBMs. For most undirected models like the RBMs, the negative phase is dicult. Models with no latent variables or with few interactions between latent variables typically have a tractable positive phase. and an intractable negative phase. The Monte Carlo approach to learning undirected models thinks of both the positive phase and the negative phase as follows. In the positive phase, we increase $\log p(\mathbf{V})$ for V drawn from the data (input for visible units). In the negative phase, we decrease the partition function by decreasing $\log p(\mathbf{h})$ drawn from the model distribution. Mostly, whenever a gradient of the log likelihood function is required, Markov chains are randomly initialized every time. This means that, every time, initial samples have to be burnt in, when we need a gradient. The burn-in time makes this approach computationally infeasible. Several approximations to the minimizing $\log Z$ exist, but there is a trade off between computational speed and accuracy.

A very simple way to counter this problem, is to initialize the initial samples very close to the model distribution, so that the burn in does not take much time. One approximation through which this can be done is the **contrastive divergence** algorithm. The contrastive divergence (CD, or CD-k to indicate CD with k-Gibbs steps) algorithm initializes the Markov chain at each step with samples from the data distribution. Obtaining samples from the data distribution is free, because they are already available in the data set. Initially, the data distribution is not close to the model distribution, so the negative phase is not very accurate. Fortunately, the positive phase can still accurately increase the models probability of the data. After the positive phase has had some time to act, the model distribution is closer to the data distribution, and the negative phase starts to become accurate.

VI. Conclusion

In today's data driven world, sampling techniques play a very crucial role in how effectively we can use the data, coupling it with the power of statistics. This report briefly explained the basics of Markov Chain Monte Carlo algorithms, Metropolis Hastings and its technicalities. We also explored Gibbs Sampling, Hamiltonian Monte Carlo and its advantages over conventional metropolis sampling. We also look at the use of MCMC in the learning algorithm of Restricted Boltzmann Machines. This, was an interesting application of the MCMC idea of sampling. In a nutshell, we explored the idea of MCMC simulations in detail, right from the first Metropolis algorithms, to the recently devised Hamiltonian Monte Carlo Simulation. Also, looked at an interesting use of MCMC sampling algorithms in a deep learning setting of RBMs. One can use HMC in the learning step of RBMs, to improve the convergence rates. More work can be done in the area of applying HMC in such frameworks.

References

[1] https://blogs.utas.edu.au/my-imas/files/2013/09/MCMC1.pdf

[2] http://www.mit.edu/ĩlkery/papers/MetropolisHastingsSampling.pdf http://ecovision.mit.edu/ sai/12S990/mcmctutorial.pdf

[3] http://www.cs.cornell.edu/selman/cs475/lectures/intro-mcmc-lukas.pdf

[4] https://arxiv.org/abs/0709.0538

[5] https://arxiv.org/pdf/0808.2902.pdf

[6] https://en.wikipedia.org/wiki/Markov_chain

[7] http://www.stat.columbia.edu/ gelman/research/published/kass5.pdf

[8] https://link.springer.com/content/pdf/10.3758

[9] https://www.youtube.com/watch?v=Z3gstWVGnSM

[10] https://arxiv.org/pdf/1705.02891.pdf

[11] http://image.diku.dk/igel/paper/AItRBM-proof.pdf

[12] http://www.deeplearningbook.org/contents/monte_carlo.html

[13] https://arxiv.org/pdf/1410.0123.pdf

[14] http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/

[15] http://www.machinelearning.org/proceedings/icml2007/papers/407.pdf

[16] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.435.9521rep=rep1type=pdf

[17] http://www-scf.usc.edu/ mohammab/sampling.pdf

[18] http://biostat.jhsph.edu/ mmccall/articles/casella_1992.pdf

[19] https://en.wikipedia.org/wiki/Gibbs_samplingMathematical_background

[20] http://www4.stat.ncsu.edu/ reich/st740/Computing2.pdf