10-708: Probabilistic Graphical Models 10-708, Spring 2014

27 : Distributed Monte Carlo Markov Chain

Lecturer: Eric P. Xing *Scribes: Pengtao Xie, Khoa Luu*

In this scribe, we are going to review the Parallel Monte Carlo Markov Chain (MCMC) method. First, we will recap of MCMC methods, particularly the Metropolis-Hasting and Gibbs Sampling algorithms. Then we will show the drawbacks of these classical MCMC methods as well as the Naive Parallel Gibbs Sampling approach. Finally, we will come up with the Sequential Monte Carlo and Parallel Inference for Bayesian nonparametric models, specifically, the Dirichlet Process Mixture model. Numerous kinds of inference techniques are also discussed in this paper.

1 Recap of MCMC and Naive Parallel Gibbs Sampling

Monte Carlo methods consist of a wide variety of algorithms that enable us to sample from complicated probability distribution functions without explicitly generating the probabilistic density function (pdf) [1]. These methods are particularly useful when it is computationally intractable to generate the pdf, $P(x)$, which is typical of pdfs defined in high-dimensional spaces.

Markov Chain Monte Carlo methods, on the other hand, use adaptive proposals $Q(x'|x)$ to sample from a given true distribution $P(x)$ and produce new appropriate samples x' by using a Markov chain. The distribution $P(x)$ is usually neither simple nor standard one. Therefore, it is unable to be sampled directly in practical applications. Markov Chain Monte Carlo can be applied in these situations where finding the exact solution is intractable.

Given the initial probability $P(x^{(0)})$ and the transition probabilities $T(x'|x)$, where $\sum_{x'} T(x'|x) = 1$ (it is usually called the homogeneous chain). The Markov chain can be computed by simply marginalizing the probability of x' as follows:

$$P(x') = \sum_{x} P(x)T(x'|x)$$

where x is the current sample.

The detailed balance property tells that the probability of picking x from the target density $P(x)$ and transition from x to x' is just as likely as picking x' and transition $T(x' \to x)$. Thus it implies the invariance of the distribution $P(x)$ and the probability distribution of the chain converges to $P(x)$.

MCMC algorithms use $Q(x'|x)$ for proposal distribution, instead of $Q(x)$. This process therefore generates a Markov chain from samples $x(1), x(2)$.... One of the most popular MCMC method is Metropolis-Hastings, which allows us to specify any proposal $Q(x'|x)$, although choosing a good $Q(x'|x)$ requires care. Gibbs sampling is a famous example of Metropolis-Hastings method, which sets the proposal $Q(x'|x)$ to the conditional distribution $P(x'|x)$ and the accept rate is always 1.

1.1 Metropolis-Hastings Sampling Algorithm

Metropolis-Hastings (MH) sampling algorithm is distinct in the sense that it is a Markov Chain variant of Monte Carlo sampling. It can be seen as a generalization of Metropolis algorithm. Specifically, the proposal is no longer a fixed probability distribution, the functional form changes based on the current state of the sampler. In other words, our proposal is only conditioned on the most recently drawn sample. Therefore, the proposal function can be denoted as the following conditional probability distribution $Q(x'|x^{(t)})$.

In order to perform sampling in this manner, we first assume that we can evaluate some unnormalized version of our pdf, $P^*(x)$, at all possible values of x. Secondly, we define an auxiliary proposal distribution which we denote as $Q(x)$. We define $Q(x)$ such that it is easy to evaluate and to sample from. With these two ingredients, we can sample from $P(x)$ using a variety of Monte Carlo sampling methods assuming certain mathematica properties hold for the proposal distribution.

Far apart from Metropolis algorithm where the proposal distribution must be symmetric, e.g. $Q(x'|x^{(t)}) = Q(x^{(t)}|x')$, the proposal distribution $Q(x'|x^{(t)})$ in Metropolis-Hastings algorithm doesn't require that property. Given a distribution $P(x)$, it can be sampled using the MH algorithm by performing Alg. 1.

Algorithm 1 Metropolis-Hastings Sampling Algorithm

Input: Conditional proposal $Q(x'|x)$, Target $P^*(x)$
Output: The sampled set of N correlated samples drawn from $P(x)$
Initialize state $x^{(0)}$
Initialize sampled set as Null
for $t = 1..N$ **do**
  Sample proposal $Q\left(x'|x^{(t)}\right)$ to obtain candidate x'
  Accept this state with a probability $\alpha = \frac{P^*(x')Q(x^{(t)}|x')}{P^*(x^{(t)})Q(x'|x^{(t)})}$
  if state is accepted **then**
    Set $x^{(t+1)} = x'$
  else
    Set $x^{(t+1)} = x^{(t)}$
  end if
  Add $x^{(t+1)}$ to the sampled set.
end for
return the sampled set

Selecting a right proposal distribution $Q(x'|x)$ for the given distribution $P(x)$ strongly impacts the performance of the algorithm in both convergence rate and the accuracy. When proposal distribution $Q(x'|x)$ has the form of isotropic Gaussian, smaller variance results in many accepted samples. However, the algorithm will slowly search through the continuous space and possibly ignore some of the models of $P(x)$. On the other hand, for larger variance, the algorithm rejects most of the samples generated from the proposal distribution since p(x) is low.

1.2 Gibbs Sampling Algorithm

Gibbs Sampling is also an instance of a Markov Chain Monte Carlo technique. It is a special case of MH algorithm where each random variable x_i is sampled, given the rest of variables or the Markov Blanket of x_i. The conditional probability distribution $P(x'|x_{-i})$ is selected as the proposal distribution $Q(x'|x)$ to sample $x_i^{(t+1)} = x_i'$ for the expected distribution $P(x_{1:K}) = P(x_1, x_2, ...; x_K)$, where x_i refers to the ith random variable and x_{-i} denotes the set of all random variables except x_i.

Algorithm 2 Gibbs Sampling Algorithm

Input: Conditional proposal
Output: The sampled set of N correlated samples drawn from $P(x)$
Initialize state $x^{(0)} =< x_1^{(0)}, x_2^{(0)}, .., x_K^{(0)} >$
Initialize sampled set as Null
for $t = 1..N$ **do**
  for $i = 1..K$ **do**
    Sample $x_i^{(t+1)} \sim P(Z_i|x_1^{(t+1)}, ..., x_{i-1}^{(t+1)}, x_{i+1}^{(t)}, x_K^{(t)})$
    Add $x^{(t+1)}$ to the sampled set.
  end for
end for
return the sampled set

For conditional distributions that belong to the family of standard distribution, we can sample directly from the conditionals, otherwise, we can draw samples using MH embedded within the Gibbs sampler. A simple calculation can show that the acceptance probability for each sample in Gibbs sampling is one.

$$A(x'|x) = \frac{P(x')Q(x|x')}{P(x)Q(x'|x)} = \frac{P(x'|x'_{-i})P(x'_{-i})P(x_i|x'_{-i})}{P(x_i|x_{-i})P(x_i)P(x'_i|x_{-i})} = 1$$

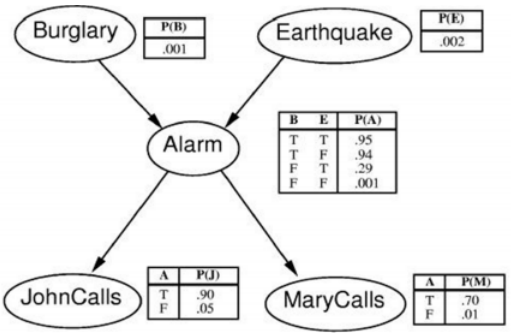
where $x'_i = x_i$. In this case, all the samples generated from a Gibbs sampler are always accepted.

1.3 MCMC in Large Scale Datasets

Datasets and models nowadays are usually very large. We may have millions to billions of data points and millions to billions of random variables. Computational time is measured in CPU-years, which means that it is impractical to run on a single thread on a single machine. In addition, we may need memory of GB or TB to store the computing information. An example is the Yahoo web graph which has 1.4 billion nodes and 6.6 billion edges. Imagining that we run a Markov Random Field on the network, it is clearly impossible to run it on a single machine. Without parallelism, large datasets and models cannot be used efficiently.

It is notice that the proper use of MCMC actually requires parallelism. To determine convergence, multiple MCMC chains have to be employed. This is the reason why we need to explore the paralleling version in Gibbs sampling algorithm. However, there is a difference between Normal Gibbs sampling approach and the Naive Parallel Gibbs Sampling one when processing multiple MCMC chains. In the Normal Gibbs sampling method, a node is updated based on the current state of all other nodes. Meanwhile, in Naive Parallel Gibbs Sampling approach, a node is updated by using a previous state of all other nodes. Fore example, recall the Alarm Network problem (as shown in Fig. 1). Let's say we initialize all variables at time t to false. The idea is to in parallel Gibbs sample all variables at step t conditioned on $t-1$. We start to sample B from $P(B|A, E)$ and the Markov blanket of B is A and E. Using Bayesian rule to figure out the conditional distribution $P(B|A, E) \propto P(A|B, E)P(B)$, a sample can be drawn from the posterior. Let's suppose the drawn B is false. Next we sample E from $P(E|A, B)$. The values of A, B are still those at $t-1$. In normal Gibbs sampling, we compute $P(E|A, B)$ based on $B_{t=1}$ and $A_{t=0}$. In a naive parallel Gibbs sampler, we compute $P(E|A, B)$ based on $B_{t=0}$ and $A_{t=0}$. This is the key difference. We always condition on the previous iteration, rather than the most recently updated values. We precede to sample A and pick false. Similarly, we precede on and sample J. Finally, we sample M and pick false. That's one iteration of the Naive Gibbs sampling. We just finished sampling variables for $t = 1$.

We run multiple Gibbs sampler on multiple machines with different seeds. They will produce different



t	B	E	A	J	M
0	F	F	F	F	F
1	F				
2					
3					
4					

Figure 1: Example of Alarm Network Problem

Markov chains. Once they are all converged to the same statistics, then we stop and say the chain has converged. If they are not converged, it is no the time to stop. However, taking multiple chains does not solve every issue. If the burn in period is long when you are doing MCMC, chains will take long time to converge. What we really want is to take each sample faster.

Using the example above, we can see that the algorithm is only conditioned on variable state at $t = 0$, which already is known in advance. We therefore can sample every node, e.g. B, E, A, J, and M, on separate processors, without having to send information between processors. In practice, this approach works very well for some graphical models, e.g. collapsed Gibbs Sampling for LDA, just assign different z_i's to different processors or machines. However, this Naive Parallel Gibbs Sampling approach may not converge to the stationary distribution since the Naive parallel GS performs poorly on near-discrete distributions.

2 Sequential Importance Sampling

When we do Sequential Importance Sampling (SIS), we draw sample x from the proposal, compute the unnormalized weights, resample given the samples and normalized weights to get a set of samples with uniform rate. Then move to the second variable, third variable, so on so forth. To summarize, in parallel Gibbs sampling, there is a naive strategy where we sample all variables at the same time and a correct strategy where we first perform graph coloring and sample same-colored node in parallel before moving to the next color. In sequential Monte Carlo, we use incremental proposal distribution in order to get an algorithm that is truly online, meaning that every new data point comes in, we only do a constant amount of work. That works in MCMC, because we have a proposal that is incremental, we have a way to compute the weights incrementally and a way to precede to the next variable incrementally. It provides a framework for designing online, parallel MCMC algorithms.

3 Parallel Inference for Bayesian Nonparametric Models

In this part, we are going to look at the parallel inference for Bayesian nonparametric models, specifically, Dirichlet process mixture model.

3.1 Recap of Dirichlet Process

First, let's take a recap of Dirichlet process. Before going to the infinite case, let's first see the finite case. Suppose there is a restaurant which has K tables in it. On each table, there is a dish. The dish here is represented by color. People enter the restaurant and pick a table. In this metaphor, table represents cluster and people are items to be clustered. Dish on the table can be thought of parameters, such as the center of each cluster. The things we want to estimate are: 1, what is the dish on each table; 2, the assignments of people to each table. We can use hard k-means to do the estimation. If we change the setting a little bit, that whether people will sit on the table is propositional to the appreciation of the dish on that table, we can use soft k-means to solve this problem. In the generative perspective, we assume each dish has a distribution, from which we can draw items. Here is the whole generative process, from some distribution H, we draw K clusters. For each item, we first draw cluster index and then draw item from the cluster corresponding to the cluster index. Now, in the clustering process, we want to incorporate the richer get richer property, which means a table with more people is more likely to be picked. In this case, the probability that a person chooses a table is not only proportional to his appreciation of the dish on the table, but also how popular the table is. This is what we called finite mixture model. If we look at the generative picture of the finite mixture model, this part is exactly the same as before. The only addition is that now the cluster index are drawn from a multinomial.

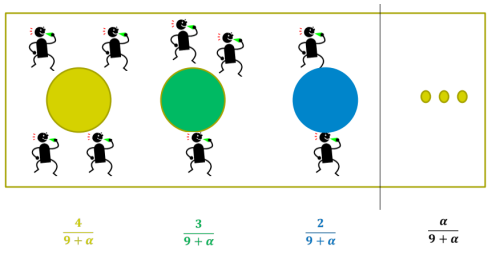


Figure 2: Restaurant Perspective of Infinite Mixture Model

Next, we consider infinite number of tables. People sit on an existing table according to their appreciation of the dish and the popularity of the table. They also pick a new table with certain amount of probability and place a new dish onto that table. This is called Dirichlet process mixture model. Here is an example (Figure 2) where three tables are occupied. The probabilities to pick an occupied table is $\frac{4}{9+\alpha}$, $\frac{3}{9+\alpha}$, $\frac{2}{9+\alpha}$. The important part is people pick a new table with probability of $\frac{\alpha}{9+\alpha}$. Note that if we want to pick a particular table, which is unoccupied, that probability is zero, because there is infinitely many of them. This process is called Dirichlet process mixture model is because, if we look at the posterior distribution, it will give we exactly the same thing as this.

The next question we are going to answer is what the sample of Dirichlet process looks like. The rectangle denotes the probability of picking a particular table. This distribution is the dish on the table. If we draw a sample from Dirichlet process, then this is what it looks like. The next question is how can we create such a distribution. One way is to create by construction. Note that there will be infinitely many such clusters. But the probability should sum to one. So what we do is we take a stick of unit length, then we break it into two parts. The first part is the probability to pick up a particular cluster. Next, we take the remain of the stick, again break it into two parts. Precede and do this infinitely often, we will get a distribution over infinite number of clusters. To the end, we have a probability to pick a table and a dish over each table.

Now it is very easy to give the generative model for Dirichlet process mixture model (Figure 3). This is the stick breaking part, using Beta distribution. This part is to sample dishes from which we can sample items later. The generative process is very similar to finite mixture model. Through the stick breaking process, we will get a multinomial distribution over infinite number of clusters. Just use this multinomial to sample

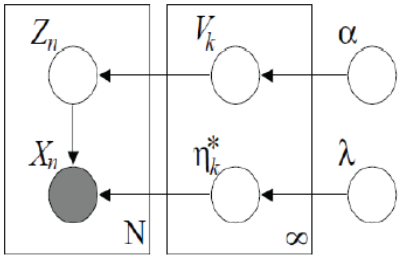


Figure 3: Graphical Representation of Dirichlet Process Model

cluster index and then generate the item from the corresponding cluster.

3.2 Parallel Inference of Dirichlet Process Mixture Model

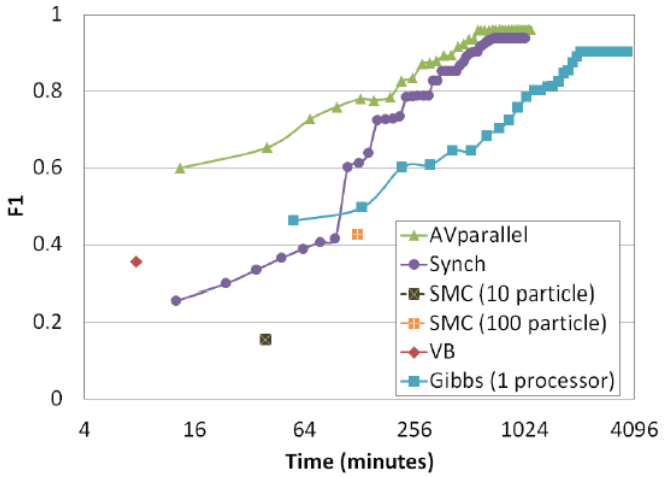


Figure 4: Performance Measure of Different Inference Methods

Now we look at different kinds of inference technique. In Gibbs sampling, we sample one random variable given the rest. That is very easy to parallelize on. If we are given V_k and η_k^*, all Z_n are independent of each other. What we can do is, we distribute the data into multiple processors. And sample Z_n independently. But we have to sample V_k and η_k^* jointly. The problem is it may show very poor mixing. Thus, it is rarely used in practice. The way to solve this problem is to use collapsed Gibbs sampling. We integrate V_k and η_k^* out, which make all Z_n inter connected. In this case, it is very difficulty to parallelize inference. There is a large cost in Gibbs sampling. Normally, in Gibbs sampling, if we fix the model size, the time complexity will increase linearly with the data size. If we look at the rate of convergence, it is $O(N)$, where N is the number of data. In Dirichlet process mixture model, since the number of parameters is also increasing, the time taken would be larger. Here is an example (Figure 4). This is synthetic data where we know which cluster each item belongs to. This enables us to plot the accuracy over time. This is the log scale plot. What we see here is that it converges around 2000 minutes. This is for 1 million points, not very large.

Another inference technique is variational inference. In variational inference, what we are trying to do is to

approximate the posterior using some manageable distribution. To make it manageable, we remove some of the dependence structure. This can lead to very simple parallel scheme. But it doesn't perform as well as MCMC techniques. You need a lot of tuning of the parameters. If we parallel the variational inference on eight processors, it converges quite early, before eight minutes or so (Figure 4). But it converges to a point that is not great in accuracy. In sequential Monte Carlo method, we keep a pool of particles. You can update the pool of particles independently, which can be trivially parallelized. These are the results of SMC on 10 particles and 100 particles. Note that to remove the high variance problem, sometime, we have to go back and run MCMC on subset of the data that we have previously seen. But we don't have a good way to distribute that particular step, so usually we just use the naive way.

Now we are done with the existing technique which can be trivially distributed, but leads to poor results. Next, we are going to discuss two methods. One is the naive method in which we distribute data on P processor and run the inference algorithm on each processor. Intermediately, we aggregate the sufficient statistics. This will give we a lot of failure. Because the color in one table is independent of the color in another table. But sometimes, for some model, if we keep doing this iteratively, it can converge and work well. There are other problems associated with this. Since this is nonparametric model, each processor can generate new clusters. How to combine the new clusters that have been found is a problem.

$$
\begin{aligned}
&D_j \sim \mathrm{DP}\left(\frac{\alpha}{P}, H\right), \quad j = 1, \ldots, P \\
&\phi \sim \mathrm{Dirichlet}\left(\frac{\alpha}{P}, \ldots, \frac{\alpha}{P}\right) \\
&\pi_i \sim \phi \\
&\theta_i \sim D_{\pi_i} \\
&x_i \sim f(\theta_i), \quad i = 1, \ldots, N.
\end{aligned}
\qquad \Longleftrightarrow \qquad
\begin{aligned}
&D \sim \mathrm{DP}(\alpha, H), \\
&\theta_i \sim D, \\
&x_i \sim f(\theta_i)
\end{aligned}
$$

Figure 5: The Equivalence between Auxiliary Variable Dirichlet Process and Ordinary Dirichlet Process

The last parallel MCMC technique for Dirichlet process mixture model is based on the following idea: the Dirichlet Mixture of Dirichlet processes are Dirichlet processes. So what does the Dirichlet mixture of Dirichlet processes mean in Chinese restaurant process mean? Each restaurant has a key and the key can lead us to a particular restaurant. The key is from a multinomial distribution which is Dirichlet distributed. Then choosing the restaurant is another Dirichlet process. How this can help us to parallel the inference of Dirichlet process? Given the keys of all items, each of the processors is independent of other processors. Then we can run the favorite inference algorithm on each of the processor independently and combine the results. In terms of generative process, it is like this: D_i the Dirichlet process for individual processor, π_i is the key of an item. The new generative model is equivalent to the original Dirichlet process model (Figure 5). The key point of this method is, we introduce an auxiliary variable, which has a conditional independence structure, which help us to distribute the model. If we look at the posterior distribution of the original model and the posterior distribution of the auxiliary variable model, they are equivalent. Once we describe the model, the inference procedure is very simple. If we get the keys, we can run inference algorithm on each processor independently. In the second step, we are going to use MCMC method to infer the keys. We can select a cluster and propose to move the cluster from one processor to another, the move will be accepted based on the likelihood ratio. It seems that this will use a lot of network bandwidth. But in practice, we can do it as less often as we want. What we found in practice is that we can do less than 10 times for this step for 10 million points.

As can be seen from Figure 4, the auxiliary variable method leads to great improvement of performance. Now let's look at the measure of perplexity. Perplexity denotes how surprise our model is when seeing new data. The lower, the better. As can be seen from the figure, the AVparallel method is much better than the

naive method. The message is that naive method will not always work. Another point is, if we can utilize the structure of the problem to explore conditional independence, the parallelization can be achieved easily.

References

[1] J. M. Hammersley et al., "Monte Carlo Methods", Vol. 1, Springer, 1964.

[2] Eric Xing, Lecture Note "Approximation Inference: Parallel MCMC", Probabilistic Graphical Models Class, CMU, 2014.

[3] J. Gonzalez et al., Parallel Gibbs Sampling: From Colored Fields to Thin Junction Trees, AISTATS, 2011.

[4] Christopher M Bishop et al. "Pattern Recognition and Machine Learning," Vol. 1, Springer, New York, 2006.