Gibbs PRAMpling

Henry Ehrenberg
Institute for Computational and Mathematical Engineering
Stanford University
henryre@cs.stanford.edu

Abstract

Gibbs sampling is a Markov chain Monte Carlo method used to approximate joint distributions that are difficult to sample from and intractable to compute directly. Optimizing the execution speed of Gibbs sampling is a large area of active research since it is a highly general and widely used algorithm. In this paper, we present several parallel Gibbs sampling approaches for estimating the parameters of Bayesian Gaussian mixture models. We analyze their complexity on parallel RAM architectures, and compare asymptotics with experimental wall-clock performance.

1 Introduction

Gibbs sampling is a popular Markov chain Monte Carlo (MCMC) method for approximating large probability distributions by a set of samples. Updates are performed by iteratively sampling each latent variable from its conditional distribution given the current values of the other variables. Gibbs sampling is frequently used to estimate marginal distributions of probabilistic graphical models, which have many applications in machine learning, computer vision, natural language processing, and physical sciences (Koller and Friedman, 2009). As data sets grow in these domains, so too does the value of fast inference methods.

To update a given latent variable, the Gibbs sampling routine only needs to access the values in its Markov blanket. Since graphical models used in practice tend to be sparse, these steps can be computed very quickly on modern hardware. However, the sequential scan Gibbs sampling algorithm (Algorithm 1) is well named: it is an inherently sequential algorithm for arbitrary models.

There are two prevalent methods for parallelizing Gibbs sampling. The first is *chromatic Gibbs sampling* which samples conditionally independent latent variables in parallel. This parallelizes only the inner loop of Algorithm 1, and the speedup attained is highly dependent on the dependency structure of the model. Recently, *HogWild! Gibbs sampling* algorithms have been studied to fully parallelize the execution of Gibbs sampling . Though they were initially proposed without theoretical guarantees on convergence (Smola and Narayanamurthy, 2010), asynchronous Gibbs sampling methods have recently been shown to produce accurate results under suitable conditions (De Sa et al., 2016).

In this paper, we investigate these parallelization approaches for Gibbs sampling in the context of Bayesian Gaussian mixture models (GMMs). GMMs are a ubiquitous family of models, and allow us to study the role of dependency structure in parallelization. We also study a new method for GMM Gibbs sampling, called *iteratively warmer starts Gibbs sampling*, which leverages connections between Gibbs sampling and stochastic gradient descent. We wrote specialized implementations of all of the discussed sampling methods, and evaluate their performance empirically.

Algorithm 1: Gibbs sampling (linear scan, sequential)

Data: Variables $\{z_m\}_{m=1}^M$, distribution D, iterations T
begin
  for $t = 1$ *to* T **do**
    for $m = 1$ *to* M **do**
      Re-sample z_m from $P_D(Z_m|Z_{-m})$

2 Sequential Gibbs sampling and Bayesian Gaussian mixture models

To ground this study in an inference model family, we will use GMMs, which are relevant in a wide array of real world fields. The graphical model for a GMM is given in Figure 1. The model describes the generation of N data points in d-dimensions from K Gaussian-distributed clusters. In practice, $N \gg K$, but we make no assumption here for the purposes of analysis. After observing the data points $x = \{x_i\}_{i=1}^N$ and establishing priors λ, σ, and π, we are interested in estimating the latent variables:

- $z = \{z_i\}_{i=1}^N$, the cluster assignments of the points x, and
- $\mu = \{\mu_k\}_{k=1}^d$, the d-dimensional cluster centroid of the k^{th} Gaussian cluster

There are two important dependence properties of the GMM model which make it both a good candidate for Gibbs sampling and parallelization. Note that from here forward, we omit priors from joint distribution notation and algorithm listings. However, their instantiation is used in sampling. We use $\hat{\mu}$ and $\hat{z}$ to denote the current values of the latent variables. First, the z_i are conditionally independent of each other and the irrelevant data points. That is,

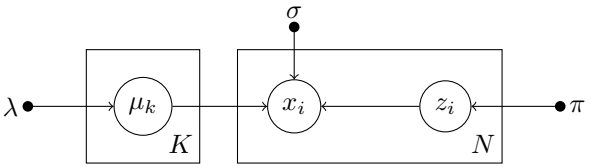
$$P(z_i|\hat{\mu}, \{z_j\}_{j\neq i}^N, x) = P(z_i|\hat{\mu}, x_i)$$

Second, the μ_k are conditionally independent of each other and the data points not assigned to them. That is,

$$P(\mu_k|\{\mu_j\}_{j\neq k}^K, z, x) = P(\mu_k|\hat{z}, x)$$

In the sequential GMM Gibbs sampler (Algorithm 2), the $K + N$ latent variables are resampled in turn. The above dependency properties are used to draw updates. We resample z_i from a categorical distribution on $\{1, 2, ..., K\}$ with weights proportional to its posterior probability of being generated from a cluster centered at each of the current $\hat{\mu}_k$ (using the prior π). To sample μ_k, we weight the mean of all points currently assigned to its cluster and sample from a Gaussian distribution centered at this location. Explicit formulas for sampling are given in Blei, 2015.

We now analyze the complexity of the sequential GMM Gibbs sampler. Updating a single z_i requires computing it likelihood of being drawn from each d-dimensional cluster, resulting in work $O(Kd)$. Resampling all assignments therefore requires $O(NKd)$ work. Updating a single μ_k is dependent on the number of points currently assigned to it, but amortized over each $k = \{1, 2, ..., K\}$, we need only scan the data set once to compute the d-dimensional means. Therefore, updating all of the cluster centers requires $O(NKd)$ work as well. Running for T iterations, this algorithm has work $O(TNKd)$. Since it is listed as fully sequential, it has the same depth. Although this Gibbs sampling routine seems inherently sequential, requiring previous updates to inform later ones, we introduce several ideas for parallelization in the next section.



$$\mu_k \sim \mathcal{N}(0, \lambda^2 I)$$
$$z_i \sim \mathbf{Cat}(\pi)$$
$$x_i \sim \mathcal{N}(\mu_{z_i}, \sigma^2 I)$$

Figure 1: Generative model for GMMs. The prior parameters are λ, σ and the class proportions π. The latent variables are the cluster centers $\{\mu_k\}_{k=1}^K$ and the cluster assignments $\{z_i\}_{i=1}^N$. The observed data points are $\{x_i\}_{i=1}^N$.

Algorithm	Work	Depth	Improvement
Sequential GS	$O(TNKd)$	$O(TNKd)$	None
Chromatic GS	$O(TNKd)$	$O(T(Kd + Nd))$	μ_k, z_i parallel
HogWild! GS	$O(TNKd)$	$O(d(N + k))$	All parallel
IWS GS	$O(TNKd)$	$O(TNKd + N + K)$	Initialization

Table 1: Work and depth time complexities for GMM Gibbs sampling algorithms.

3 Parallel methods for GMM Gibbs samplers

Using the discussed independence properties of GMMs and the clear connections between Gibbs sampling and stochastic gradient descent, we can parallelize the GMM Gibbs sampler in a number of ways. We compare the benefits of each method in this section, and analyze their complexity. In analyzing convergence, the mixing time of the chain (or the number of iterations needed to approach the steady state distribution) is often used in literature. However, many of the benefits of the discussed (such as asynchronicity and initialization states) are more closely tied to hardware and practical implementation. Therefore, we defer discussion of their performance to the experimental results in Section 3.

We now introduce and analyze the three parallel methods. When assessing depth, we consider only the parallelization of sampling tasks (i.e. updating $\hat{z}_i$ and $\hat{\mu}_k$), since more basic operations (such as sums) would not be parallelized in practice. Complexities are summarized in Table 1.

Chromatic Gibbs sampling

Chromatic Gibbs sampling (Gonzalez et al., 2011) exploits the independence properties of the graph discussed in Section 2. The name is derived from graph coloring: if we color the graphical model such that no two connected nodes share the same color, then we can sample the nodes of the same color in parallel. This approach allows us to sample the $\hat{z}_i$ in parallel, and the $\hat{\mu}_i$ in parallel while maintaining the same convergence guarantees.

By precomputing means for all clusters in a single pass over the data, the work is the same as that for the sequential algorithm: $O(TNKd)$. On an infinite number of processors, we remove the dependence on N when updating assignments, and the dependence of K when updating centers. Therefore, the chromatic GMM Gibbs sampler has depth $O(T(Kd + Nd))$. Chromatic Gibbs sampling for GMMs is listed in Algorithm 3.

HogWild! Gibbs sampling

With the advent of asynchronous methods for gradient descent, HogWild! Gibbs sampling methods (Smola and Narayanamurthy, 2010) have become a prevalent field of study. The convergence properties of these approaches were previously not well understood. However, recent work (De Sa et al., 2016) has shown under suitable conditions that compared to sequential methods, the bias of HogWild! Gibbs samplers is bounded by a constant factor and the mixing time is increased by $O(m^{-1})$, where m is the number of variables.

Algorithm 2: Sequential GMM Gibbs sampler

Data: Data $\{x_i\}_{i=1}^N$, parameters π, σ^2, λ^2
begin
  Initialize $\hat{z}, \hat{\mu}$
  for $t = 1$ *to* T **do**
    for $i = 1$ *to* N **do**
      Re-sample $\hat{z}_i$ from $P(z_i|\hat{\mu}, x_i)$
    for $k = 1$ *to* K **do**
      Re-sample $\hat{\mu}_k$ from $P(\mu_k|\hat{z}, x)$

Algorithm 3: Chromatic GMM Gibbs sampler

Data: Data $\{x_i\}_{i=1}^N$, parameters π, σ^2, λ^2
begin
  Initialize $\hat{z}, \hat{\mu}$
  for $t = 1$ *to* T **do**
    parallel for $i = 1$ *to* N **do**
      Re-sample $\hat{z}_i$ from $P(z_i|\hat{\mu}, x_i)$
    parallel for $k = 1$ *to* K **do**
      Re-sample $\hat{\mu}_k$ from $P(\mu_k|\hat{z}, x)$

Algorithm 4: HogWild! GMM Gibbs sampler

Data: Data $\{x_i\}_{i=1}^N$, parameters π, σ^2, λ^2
begin
  Initialize $\hat{z}, \hat{\mu}$
  parallel for $t = 1$ *to* T **do**
    if t *mod* $(K + N) \leq N$ **then**
      Re-sample $\hat{z}_t$ from $P(z_t|\hat{\mu}, x_t)$
    else
      $k \leftarrow t - N$
      Re-sample $\hat{\mu}_k$ from $P(\mu_k|\hat{z}, x)$

Algorithm 5: IWS GMM Gibbs sampler

Data: Data $\{x_i\}_{i=1}^N$, parameters π, σ^2, λ^2
begin
  Initialize $\Theta = \{\hat{z}^{(m)}, \hat{\mu}^{(m)}\}_{m=1}^M$
  for $t = 1$ *to* T **do**
    parallel for $m = 1$ *to* M **do**
      for $t' = 1$ *to* T' **do**
        for $i = 1$ *to* N **do**
          Re-sample $\hat{z}_i^{(m)}$ from $P(z_i^{(m)}|\hat{\mu}^{(m)}, x_i)$
        for $k = 1$ *to* K **do**
          Re-sample $\hat{\mu}_k^{(m)}$ from $P(\mu_k^{(m)}|\hat{z}^{(m)}, x)$
    $\Theta \leftarrow$ Aggregate(Θ)

Depth analysis of HogWild! style algorithms is typically not informative, since the speedup lies in the removal of locks. However for comparison, we provide one here. Sampling an assignment remains $O(Kd)$, and the entire data set needs to be scanned to update a centroid, giving $O(Nd)$. Therefore, the depth is $O(d(N + k))$. HogWild! sampling for GMMs is listed in Algorithm 4.

Iteratively warmer starts Gibbs sampling

We introduce a new method called iteratively warmer starts (IWS) Gibbs sampling, inspired by the SimuParallel approach for stochastic gradient descent (Zinkevich et al., 2010). A parallel Gibbs sampling approach that is used in practice is to start fully independent samplers on each CPU, and choose the model with the highest likelihood. This has the benefit of fault tolerance and less dependency on initialization values for $\hat{z}$ and $\hat{\mu}$, but does not fully take advantage of incremental results. The IWS approach runs independent samplers for a fixed number of iterations, aggregates the results, then restarts the independent samplers with the new aggregated values as the initialization. Majority votes/means and highest likelihood sampler are natural aggregation schemes for this method; for consistency, we use highest likelihood. The bias introduced by this method is likely negligible, due to the sparsity of the dependency graph (Barber, 2012).

For a single processor, IWS Gibbs sampling reduces to the sequential case. Therefore, it shares the same work. When $M > 1$ processors are used, the resulting likelihoods need to be compared. Assuming that $M \leq N + K$, the depth of IWS Gibbs sampling is bounded by $O(TNKd)$. IWS Gibbs sampling for GMMs is listed in Algorithm 5.

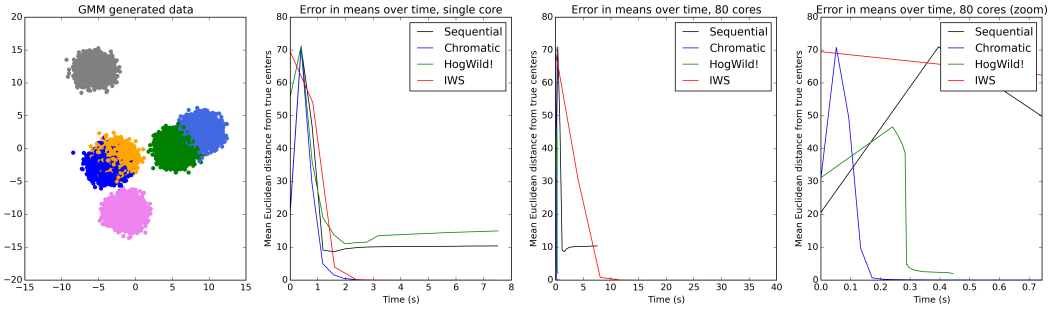


Figure 2: Experimental setup for GMM Gibbs sampling with $N = 50,000$, $K = 6$, $d = 2$, $\sigma^2 = 1$ and $\lambda^2 = 4$. The colors of the data points correspond to their true assignments z. The plot at the far right shows detail from early wall clock times of the third plot.

Experimental results

To compare the performance of these methods, we will use wall clock burn-in time. Burn-in is the practical dual to mixing time: the algorithm is run for a fixed number of iterations before steady state is assumed to be reached, at which point meaningful samples can be drawn. In particular, we compute the mean distance between the true μ and $\hat{\mu}$ (which we call error) as a function of wall clock time. All methods were hand-rolled in C++ with OpenMP[1], and are available at `http://github.com/henryre/gibbs-prampling`.

Figure 2 illustrates the experimental setup. Data was generated according to the Bayesian Gaussian mixture model with $N = 50,000$, $K = 6$, $d = 2$, $\sigma^2 = 1$ and $\lambda^2 = 4$. Each algorithm was run using a single core for $T = 20$ iterations, and the mean distance of the $\hat{\mu}$ from μ was computed at the start of each (as well as the wall clock time). The experiment was repeated using the same data set and 80 cores for the parallel methods. Note that the per-sampler iterations T' was fixed at 3 for all experiments and not tuned.

When run on a single core, the computations for each method are identical, up to randomness from initialized values and sampling. Therefore, we expect similar performance for each. The HogWild! and sequential methods appear to have converged to false centroids, due to initialized values and highly overlapping clusters. When run on 80 cores, the chromatic and HogWild! samplers both burn-in in under 0.5 seconds, but the minimal error is achieved by IWS after more than 15 seconds.

We are also interested in how the algorithms scale in response to more variables and more available cores. See Figure 3.

Figure 3(a) shows that the burn-in times scale by the same law for all algorithms with respect to the number of variables in the model. However, HogWild! and chromatic samplers displayed the fastest burn-in times overall. In particular, HogWild! performed better with fewer variables in the model, as the outer loop locks occurred more frequently in the chromatic sampler. The overhead in starting and aggregating multiple independent samplers caused IWS to have a slower execution time. Figure 3(b) assesses the accuracies of the methods. Two trends are apparent. First, as asynchronocity grows, the HogWild! algorithm loses accuracy. Second, as more independent samplers are aggregated, the IWS method improves. Therefore, the HogWild! GMM sampler and the IWS sampler issue a tradeoff between speed and accuracy, while the chromatic sampler offers a compromise.

Conclusions

Gibbs sampling is a powerful and widespread method, and continued study and improvement of its execution will have a wide reaching impact. In this paper, we presented several methods for parallelizing GMM Gibbs samplers, and analyzed their complexity and practical benefits. We showed experimentally that asynchronous and chromatic samplers burn-in very rapidly, and IWS samplers

[1] `http://openmp.org/`

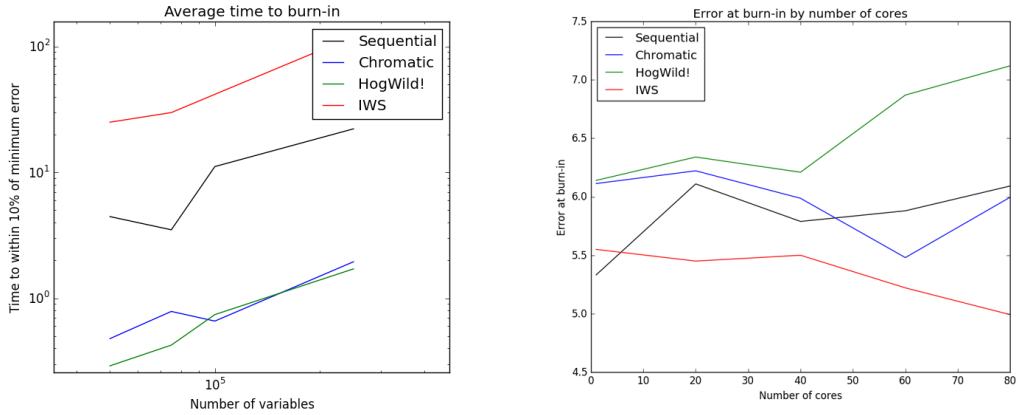


Figure 3: Left (a): A log-log plot of average time (in seconds) to reach within 10% of the minimum error for each algorithm by number of variables. Parallel methods used 80 cores. Right (b): A plot of the average error by the number of cores used. The number of variables in the model was 50,000 for all trials.

offer more accurate parameter estimates at the cost of execution speed. With inference tasks growing larger and larger, parallelism for Gibbs sampling methods is an invaluable tool for efficient computation.

Acknowledgments

I would like to thank the other members of Chris Ré's research group for their helpful chats.

References

[1] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[2] David Blei. "Bayesian Mixture Models and the Gibbs Sampler". In: *Foundations of Graphical Models* (2015).

[3] Christopher De Sa et al. "Ensuring Rapid Mixing and Low Bias for Asynchronous Gibbs Sampling". In: *arXiv preprint arXiv:1602.07415* (2016).

[4] Joseph Gonzalez et al. "Parallel gibbs sampling: From colored fields to thin junction trees". In: *International Conference on Artificial Intelligence and Statistics*. 2011, pp. 324–332.

[5] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[6] Alexander Smola and Shravan Narayanamurthy. "An architecture for parallel topic models". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 703–710.

[7] Martin Zinkevich et al. "Parallelized stochastic gradient descent". In: *Advances in neural information processing systems*. 2010, pp. 2595–2603.