Distributed Inference for Dirichlet Process Mixture Models

Hong Ge HG344@CAM.AC.UK
Yutian Chen YUTIAN.CHEN@ENG.CAM.AC.UK
Moquan Wan MW545@CAM.AC.UK
Zoubin Ghahramani ZOUBIN@ENG.CAM.AC.UK
Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, UK

Abstract

Bayesian nonparametric mixture models based on the Dirichlet process (DP) have been widely used for solving problems like clustering, density estimation and topic modelling. These models make weak assumptions about the underlying process that generated the observed data. Thus, when more data are collected, the complexity of these models can change accordingly. These theoretical properties often lead to superior predictive performance when compared to traditional finite mixture models. However, despite the increasing amount of data available, the application of Bayesian nonparametric mixture models is so far limited to relatively small data sets.

In this paper, we propose an efficient distributed inference algorithm for the DP and the HDP mixture model. The proposed method is based on a variant of the slice sampler for DPs. Since this sampler does not involve a pre-determined truncation, the stationary distribution of the sampling algorithm is unbiased. We provide both local thread-level and distributed machine-level parallel implementations and study the performance of this sampler through an extensive set of experiments on image and text data. When compared to existing inference algorithms, the proposed method exhibits state-of-the-art accuracy and strong scalability with up to 512 cores.

1. Introduction

Probabilistic modelling is a mainstay of modern AI, providing necessary tools for analysing the vast amount of complex data from science, scholarship and daily life. Successful applications of such models range from information retrieval (Blei et al., 2003) to large scale recommender systems (Mnih & Salakhutdinov, 2007). In particular, nonparametric mixture models based on the DP (Ferguson, 1973) and its extensions are one of the most popular family of probabilistic models for applications in clustering, density estimation, language modelling, and topic modelling (Teh, 2010). Mixture models based on DPs treat the number of represented mixture components as a latent variable, and infer it automatically from observed data. This formulation allows model complexity to grow dynamically as more data are collected. However, despite the the sharply increasing available amount of data (e.g., DNA and protein sequences, text data), the applicability of DP mixture models is so far constrained by the absence of scalable inference algorithms.

One approach to address this problem is through parallelisation. Indeed, recently it has become an active area to improve the efficiency of Bayesian inference via exploiting distributed computation. For instance, Newman et al. (2009) described two distributed approximate algorithms for the widely-used Latent Dirichlet Allocation (LDA) model; Doshi-Velez et al. (2009) proposed a parallel inference algorithm for the Indian buffet process (IBP) model; Ahn et al. (2014) proposed another distributed method for LDA based on the stochastic gradient Langevin dynamics sampler. Parallel algorithms for DP mixture models have also been proposed. An example is to re-parametrise the DP mixture model as a mixture of DPs, and then perform inference for each DP in parallel at a thread-level (Williamson et al., 2013). This method does not involve approximation, but as suggested by Gal & Ghahramani (2014) its scalability is limited due to unbalanced loading in the computing units. Chang & Fisher III (2013) studied another thread-level parallel inference algorithm for DPs based on the split-merge framework (Jain & Neal, 2004).

In this paper, we propose an unbiased distributed inference algorithm for the DP and HDP mixture models based on an improved slice sampler of DPs. We demonstrate that the proposed sampler has superb scalability in both local

Proceedings of the 32^{nd} *International Conference on Machine Learning*, Lille, France, 2015. JMLR: W&CP volume 37. Copyright 2015 by the author(s).

thread–level and distributed machine–level experiments. This strong scalability is critical when designing and evaluating distributed algorithms.

The rest of this paper is organised as follows: In Section 2 we give a brief review of the DP mixture model and HDP mixture model. In Section 3 we describe a slice sampler for the DP mixture model. In Sections 4 and 5 we derive the distributed sampler under the Map–Reduce framework and Section 6 contains our experimental results. Finally, Section 7 concludes with a brief discussion.

2. Model and notation

2.1. Dirichlet Process Mixture Models

The basic DP mixture model applies to data $y_1, y_2, \ldots, y_N$ which are drawn independently from some unknown distribution. This distribution can be modelled as a mixture of simpler distributions with the form $g(y|\theta)$, with the mixing distribution over θ given a DP prior with concentration α and a base distribution H. This gives the following model:

$$y_i|\theta_i \sim g(y \mid \theta_i), \quad \theta_i|G \overset{\text{iid}}{\sim} G, \quad G \sim \text{DP}(\alpha, H) \tag{1}$$

Here θ_i represents the parameter of the cluster associated with data point y_i, and G is a random measure.

If we integrate over G, we obtain a marginal representation of the prior distribution of θ_i in terms of successive conditional distributions of the following form:

$$\theta_{i+1}|\theta_1, \ldots, \theta_i \sim \frac{1}{\alpha + i}\left(\alpha H + \sum_{j=1}^{i} \delta_{\theta_j}\right) \tag{2}$$

Here, δ_θ is the distribution concentrated at the atomic location θ. This representation is often known as the Blackwell MacQueen urn scheme (see e.g., Pitman (1996)). An alternative way of drawing samples from a DP prior is the stick breaking procedure introduced by Sethuraman (1994). By assuming $\phi_k \sim H$, a random measure G can be drawn from a DP prior using a sequence of i.i.d. beta variables as follows:

$$G = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k}, \; \beta_k = \nu_k \prod_{i=1}^{k-1}(1 - \nu_i), \\ \nu_k \overset{\text{iid}}{\sim} \mathcal{B}e(1, \alpha). \tag{3}$$

Because G is discrete in Equation 3, θ_i (Equation 2) values will repeat and therefore data will cluster.

2.2. Hierarchical Dirichlet Process Mixture Models

The hierarchical Dirichlet process (HDP) extends the DP to model grouped data, e.g., text grouped in documents (Teh et al., 2006). In an HDP mixture model, one first samples a random measure G_0 from a DP prior with base measure γH. Then for each group of data D_j, a random measure $G_j = \sum_{k=1}^{\infty} \pi_{jk}\delta_{\phi_k}$ is drawn from a DP prior with base measure αG_0. For the i'th observation in group j, a component assignment $z_{ji} = k$ is drawn with probability π_{jk}. Observation y_{ji} is then drawn from component z_{ji}, with y_{ji} distributed as $g(\cdot \mid \phi_{z_{ji}})$. The model can be summarised as:

$$y_{ji} \sim g(y \mid \phi_{z_{ji}}), \qquad z_{ji} \sim \mathcal{C}at(\boldsymbol{\pi}_j), \\ G_j = \sum_{k=1}^{\infty} \pi_{jk}\delta_{\phi_k} \overset{\text{iid}}{\sim} \text{DP}(\alpha, G_0), \quad G_0 \sim \text{DP}(\gamma, H) \tag{4}$$

3. Posterior Inference via MCMC

Inference for DP or HDP mixture model has been made feasible by Gibbs sampling since the seminal work of Escobar & West (1998). Broadly speaking, there are two Gibbs sampling techniques for DPs, corresponding to two different representation of DPs. The marginal method exploits convenient theoretical property of DPs and integrates out analytically the random measure G. Then, a Gibbs sampler is used to sample from the posterior distribution of the component assignments and the parameters and the weights of the clusters.

In contrast, the conditional Gibbs sampler retains the the random measure G in Equation 3. It consists of the imputation of G, and subsequent Gibbs sampling of the component assignments from their posteriors. The difficulty of constructing a valid conditional sampler, is to find a way of dealing with potentially infinite number of components in DPs. A solution to this problem is the slice sampler developed by Walker (2007). In the following section, we first review the standard slice sampler for DPs, then present an improved slice sampler with a more efficient updating scheme for component weights.

3.1. A Slice Sampler

To begin, consider a random measure $G = \sum_{k=1}^{\infty} \beta_k \delta_{\phi_k}$ sampled through the stick breaking procedure as described in Equation 3. Then conditioned on this random measure G, the probability of component assignment z_i for the i'th observation can be written as

$$p(z_i = k \mid y_i, \; G) \propto \beta_k g(y_i \mid \phi_k). \tag{5}$$

Equation 5 shows that component assignment variable z_i's are independent among i given G. To be able to represent the infinite number of components in G, we have to introduce some auxiliary variables. The starting point of the solution is the data density

$$g(y_i \mid \boldsymbol{\phi},\, \boldsymbol{\beta}) = \sum_{k=1}^{\infty} \beta_k g(y_i \mid \phi_k). \tag{6}$$

Through introducing an auxiliary variable u_i, the joint density of y_i and the latent variable u_i becomes

$$\begin{aligned} g(y_i,\, u_i \mid \boldsymbol{\phi},\, \boldsymbol{\beta}) &= \sum_{k=1}^{\infty} \beta_k \mathcal{U}(u_i \mid 0,\, \beta_k) g(y_i \mid \phi_k) \\ &= \sum_{k=1}^{\infty} \mathbb{1}(u_i \leq \beta_k) g(y_i \mid \phi_k), \end{aligned} \tag{7}$$

where $\mathbb{1}(\cdot)$ is the indicator function. We can easily verify that, when u_i is integrated over, Equation 7 is equivalent to Equation 6. Thus the conditional density of y_i becomes $g(y_i \mid u_i,\, \boldsymbol{\phi},\, \boldsymbol{\beta}) \propto \sum_{k:\beta_k \geq u_i} g(y_i|\phi_k)$. The interesting fact is that given the latent variable u_i, the number of mixtures needed to be represented is *finite*: conditioned on the slice level u_i, each observation y_i can only join a clustering component k if $k \in A_i := \{k : \beta_k \geq u_i\}$. Intuitively, the latent variable u_i has an effect of "dynamically truncating" the number of components needed to be sampled. Through exploiting this property, Walker (2007); Kalli et al. (2011) proposed a slice sampler:

Step 1: Sample slice variables and find the minimum

$$\begin{aligned} u_i &\sim \mathcal{U}(0, \beta_{z_i}), \quad \forall i = 1, \ldots, N \\ u^* &:= \min_i u_i. \end{aligned} \tag{8}$$

Step 2: Create new components through stick breaking until $\beta^* < u^*$ with β^* being the remaining stick length and K^* the number of instantiated components

$$\begin{aligned} K^* &\leftarrow K^* + 1, \quad \nu_{K^*} \sim \mathcal{B}e(1, \alpha), \\ \beta_{K^*} &= \beta^* \nu_{K^*}, \quad \phi_{K^*} \sim H, \\ \beta^* &\leftarrow \beta^*(1 - \nu_{K^*}). \end{aligned} \tag{9}$$

Step 3: For each observation x_i, sample the component assignment z_i

$$p(z_i = k \mid y_i,\, u_i,\, \boldsymbol{\beta},\, \boldsymbol{\phi}) \propto \begin{cases} g(y_i|\phi_k) & \text{if } \beta_k \geq u_i \\ 0 & \text{otherwise} \end{cases}. \tag{10}$$

Step 4: For each active component k, sample component parameters ϕ_k

$$\phi_k \mid \mathbf{z},\, \mathbf{y} \sim H(\phi_k) \prod_{(i:z_i=k)} g(y_i \mid \phi_k). \tag{11}$$

Step 5: For each active component k, sample stick-breaking length and compute new component weights

$$\begin{aligned} \tilde{\nu}_k \mid \mathbf{z}, \alpha &\sim \mathcal{B}e\Big(1 + n_k, \alpha + \sum_{j=k+1}^{K} n_j\Big), \\ \beta_k &= \tilde{\nu}_k \prod_{j=1}^{k-1} (1 - \tilde{\nu}_j), \end{aligned} \tag{12}$$

where n_k is the number of observations assigned to component k and K is the number of active components.

In the above slice sampler, the latent variable u_i does not change the marginal distribution of other variables, thus the sampler target the correct posterior distribution. Another important feature of the slice sampler is that it enables direct inference of random measure G. This is important for developing efficient inference algorithm some elaborate models like HDP-HMMs (Van Gael et al., 2008). More recently, Teh et al. (2007) exploited a similar trick to develop a slice sampler for the Indian buffet process (IBP; Griffiths & Ghahramani, 2011).

3.2. An Improved Slice Sampler

Unlike a Gibbs sampler using the marginal representation, component labels in the slice sampler are no longer exchangeable. This non-exchangeability leads to a label switching problem which could have a dramatic impact on the mixing of the slice sampler (see e.g., Papaspiliopoulos & Roberts (2008)). However, through using an alternative posterior representation DPs described in Pitman (1996), the exchangeability of components in the slice sampler can be restored, which is quite helpful for deriving an improved slice sampler:

$$G \mid \boldsymbol{\beta}, \boldsymbol{\phi} = \sum_{j=1}^{K} \beta_k \delta_{\phi_k} + \beta^* H^*, \tag{13}$$

where $\boldsymbol{\beta} := (\beta_1, \beta_2, \ldots, \beta_K, \beta^*)$, and

$$\begin{aligned} \boldsymbol{\beta} \mid \mathbf{z}, \alpha &\sim \mathcal{D}ir(n_1, n_2, \ldots, n_K, \alpha), \\ H^* &\sim \mathrm{DP}(H, \alpha). \end{aligned} \tag{14}$$

Using this new formulation of posterior of DPs, we can write down an improved variant of the slice sampler for DPs:

Step 1, 2, 3, 4: Same as the slice sampler.
Step 5: For each component k, sample component weights:

$$\boldsymbol{\beta} \mid \mathbf{z}, \alpha \sim \mathcal{D}ir(n_1, n_2, \ldots, n_K, \alpha). \tag{15}$$

4. A Map–Reduce Sampler for DP

Through using the improved slice sampler described in Section 3, it is possible to derive a parallel sampler for

Algorithm 1 The M–R Sampler for DP

```
function map($y_i$, $z_i$):
  // Global:  $u^*$, $i^*$, $\boldsymbol{\beta}$, $\boldsymbol{\phi}$
  if $i == i^*$ then $u_i \leftarrow u^*$
  else sample $u_i \sim \mathcal{U}(u^*, \beta_{z_i})$
  Resample $z_i$ // Equation 10
  emit sufficient statistics $\psi(y_i, z_i)$

function reduce({$\psi(z_i, y_i)$}):
  Accumulate $\{\psi_i\}$ to get $\{n_k\}_{k=1}^K$ and
    other sufficient statistics $\{\psi'_k\}_{k=1}^K$
  For $k = 1, \ldots, K$
    Sample $\phi_k \mid n_k, \psi'_k$ \\ Equation 11
  Sample $\boldsymbol{\beta} \mid \mathbf{n}, \alpha$ \\ Equation 15
  Sample $u^*$, $i^*$ \\ Section 4.1
  Instantiate new components: $\{\phi^{new}, \beta^{new}\}$
  broadcast globals: $(u^*, i^*, \{\beta_k, \phi_k\}_{k=1}^{K^*})$
```

the DP mixture model under the Map–Reduce framework. First recall the full collection of variables to be represented in the slice sampler, for the DP these are $\{\alpha, \phi_k, \beta_k, z_i, u_i\}$, $k = 1, 2, \ldots, K^*$ and $i = 1, 2, \ldots, N$. Here K^* as defined in Step 2 denotes the number of clusters that have to be instantiated to guarantee all the A_i are available when sampling z_i. The algorithm is given in Alg. 1.

All operations in the Map function can be executed in parallel. α is sampled as in Teh et al. (2006) and $\{\phi_k\}$, $\{\beta_k\}$ are sampled as described in the previous section. We further improve the sampling efficiency of $\{u_i\}$ and $\{z_i\}$ as follows.

4.1. Efficient Sampling of Slice Variable u's

The number of clusters to be instantiated in Step 2 of Section 3 depends on the value of u^* in Step 1 that in turn depends all the slice variables. So in order to update $\boldsymbol{\beta}$ of K active clusters in Step 5 one has to sample all the local slice variables before generating new empty clusters. This leads to an awkward interleaved ordering in sampling global and local variables: $\beta_{1:K} \to \{u_i\} \to \beta_{K+1:K^*} \to \{z_i\}$, and even worse it involves two Map–Reduce operations for every MCMC iteration. It is well understood that many global sync operations could affect the scalability considerably in a distributed system due to the high communication cost and waiting time for slow workers. We propose here a more efficient way to sample the slice level of clusters K^* without sampling all the slice variables in advance.

Let $\mathbf{y}_k$ be the set of observations belonging to cluster k (i.e., $\mathbf{y}_k = \{y_i : z_i = k\}$). n_k is then the number of elements in $\mathbf{y}_k$. Denote the minimum slice variable in cluster k as $u_k^* = \min_{i:y_i \in \mathbf{y}_k} u_i$. u^* can be sampled by first sampling all u_i's as in Step 1 and taking the minimum. Equivalently, we can generate a sample of u^* as

$$u^* = \min_k u_k^*, \quad u_k^* = \beta_k b_k, \quad b_k \sim \mathcal{B}e(1, n_k) \qquad (16)$$

where the first equation follows the definition of u^* and the last two equations are due to the fact that u_k^* is the minimum statistics of n_k *i.i.d.* random variables, $\{u_i : y_i \in \mathbf{y}_k\}$, each following $\mathcal{U}(0, \beta_k)$. Equation 16 requires only the sufficient statistics n_k and therefore we can first sample all the global variables and then the local variables as $\beta_{1:K} \to u^* \to \beta_{1:K} \to \{u_i, z_i\}$ in one Map–Reduce operation as shown in the Reduce function of Alg. 1.

Following the property of minimal order statistics for uniform distributions, given u^* and the corresponding cluster index k^* we can sample the index of the slice variable that achieves u^* retrospectively as $i^* \sim \mathrm{Uniform}\{i : y_i \in \mathbf{y}_{k^*}\}$. For all the other slice variables, as anther property of the minimum statistics, they are still independent with each other and follow the truncated distribution:

$$u_i \sim \mathcal{U}(u^*, \beta_{z_i}), \quad \forall i \neq i^*, \qquad (17)$$

which remains parallelisable as shown in the Map function.

4.2. Efficient Sampling of Component Assignment Variable z's

Now we discuss how to improve the efficiency of sampling the component assignment z_i for each observation. Recall that, conditioning on the slice level u_i, each data item can only join a cluster k if $k \in A_i := \{k : \beta_k \geq u_i\}$. Since these assignment variables are conditionally independent with each other given $\{\phi_k, \beta_k\}$, they can be sampled in parallel on multiple workers. Then, the sufficient statistics are collected and fed to a single reduce worker to re-sample component parameters and weights. The conditional probability of each component assignment has been described in Equation 10.

Empirically, we observe that component weights β_k's are mostly concentrated in a small number of components while the rest exhibits a heavy tail. This fast decaying behaviour was also noted by Newman et al. (2009) and is illustrated in Figure 4. The slice sampler benefits from this fast decay of the β_k's, because in order to sample z_i one only need to consider the clusters with $\beta_k \geq u_i$. By reordering the clusters according to β_k, we can find A_i efficiently with a simple forward search. In contrast, in the Gibbs sampler, the likelihoods of y_i joining each cluster have to be computed. In our experiments, it leads to a reduction up to 50% of running time for the slice sampler.

Algorithm 2 The M–R Sampler for HDP

```
function map($y_{j\cdot}$, $z_{j\cdot}$):
  // Global: $\boldsymbol{\beta}$, $\boldsymbol{\phi}$
  For each observation $y_{ji}$
    Sample slice variables $u_{ji}$ //Eq. 23
    While $u_{ji} < \beta^*$
      Sample new components //Eq. 24, 25
    Sample component labels $z_{ji}$ // Eq. 18
  emit sufficient statistics $\psi(y_{j\cdot}, z_{j\cdot})$

function reduce($\psi(y_{j\cdot}, z_{j\cdot})$):
  Accumulate $\psi(y_{j\cdot}, z_{j\cdot})$ to get $\{n_{jk}\}$
  Sample $\boldsymbol{\beta}$ // Eq. 19
  Sample $\phi_k \mid \mathbf{z}, \mathbf{y}, k = 1, \ldots, K$ //Eq. 20
  broadcast global: $(\{\beta_k, \phi_k\}, k = 1, \ldots, K)$
```

5. A Map–Reduce Sampler for HDP

In this section, we present a Map–Reduce sampler for the HDP model: the full collection of variables to be represented in the sampler is $\{\phi_k, \beta_k, \pi_{jk}, z_{ji}, u_{ji}\}$, where $k = 1, 2, \ldots, K^*$, $j = 1, 2, \ldots, J$ and $i = 1, 2, \ldots, N_j$, where J is the number of groups and N_j is the number of observations in group j. The conditional probability of these variables are described below.

Step 1: For the i'th observation in group j, sample component assignment z_{ji}:

$$p(z_{ji} = k \mid u_{ji},\ y_{ji},\ \boldsymbol{\pi}_j,\ \boldsymbol{\phi}) \propto \begin{cases} g(y_{ji}|\phi_k) & \text{if } k \in \{c : \pi_{jc} \geq u_{ji}\} \\ 0 & \text{otherwise} \end{cases}. \tag{18}$$

Step 2: Sample component weights and parameters similarly as in Step 2 of the slice sampler of DPs:

$$\boldsymbol{\beta} \mid \mathbf{z}, \alpha \ \sim\ \mathcal{D}ir(m_{\cdot 1}, m_{\cdot 2}, \ldots, m_{\cdot K}, \gamma), \tag{19}$$

$$\phi_k \mid \mathbf{z},\ \mathbf{y} \ \sim\ H(\phi_k) \prod_{(j,i:z_{j,i}=k)} g(y_{j,i} \mid \phi_k). \tag{20}$$

where m_{jk} is a further set of auxiliary variables which are independent with conditional distributions

$$p(m_{jk} = m \mid \mathbf{z}, \boldsymbol{\beta}) \propto s(n_{jk}, m)(\alpha\beta_k)^m, \tag{21}$$

here $s(\cdot,\cdot)$ denotes Stirling numbers of the first kind. *Step 3: For each group j, sample group-level component weights $\boldsymbol{\pi}_j := (\pi_{j1}, \pi_{j2}, \ldots, \pi_{jK}, \pi_j^*)$:*

$$\boldsymbol{\pi}_j \ \sim\ \mathcal{D}ir(n_{j1} + \beta_1, n_{j2} + \beta_2, \ldots, n_{jK} + \beta_K, \alpha). \tag{22}$$

Step 4: For each group j, sample slice variables and compute the minimum:

$$u_{ji} \sim \mathcal{U}(0, \pi_{z_{ji}}), \quad u_j^* = \min_i u_{ji}. \tag{23}$$

Step 5: Create new components until $\pi_j^* < u_j^*, \forall j$:

(a) Create new components similarly as Step 2 in the slice sampler for DPs:

$$\begin{aligned} K^* \leftarrow K^* + 1, \quad \nu_{K^*} &\sim \mathcal{B}e(1, \gamma), \\ \beta_{K^*} = \beta^* \nu_{K^*}, \quad \phi_{K^*} &\sim H, \\ \beta^* \leftarrow \beta^*(1 - \nu_{K^*}). \end{aligned} \tag{24}$$

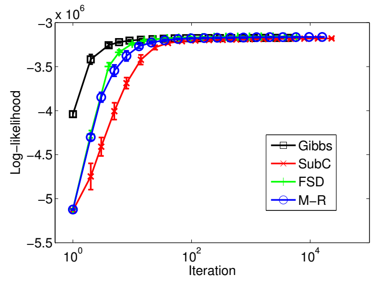
(b) *For each group j, create a new component:*

$$\begin{aligned} v_{jK^*} &\sim \mathcal{B}e\Big(\alpha\beta_{K^*}, \alpha(1 - \sum_{l=1}^{K^*} \beta_l)\Big), \\ \pi_{j,K^*} &= \pi_j^*(1 - \nu_{j,K^*}), \\ \pi_j^* &= \pi_j^*(1 - v_{j,K^*}). \end{aligned} \tag{25}$$

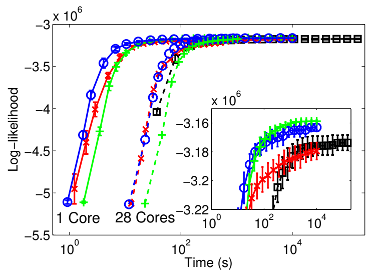
In this sampler, component labels z_{ji} are independent given component parameters ϕ_k and group-level component weights $\boldsymbol{\pi}_j$. This means we can sample all component labels in Step 1 in parallel. However, to instantiate new clusters in Step 5, the technique for efficiently sampling the minimum slice variable in Section 4.1 is still too expensive because we have to collect n_{jk} to the master from all documents. In our implementation, an alternative solution is used by noting that new components can be sampled on workers locally for each group, that is, whenever $u_{ji} < \pi_j^*$, the weight and parameter of a new component is sampled with Equation 24 and 25. The difficulty, however, is to ensure consistency of $\boldsymbol{\beta}$ and $\boldsymbol{\phi}$ across all the computing nodes. Notice that commonly used random number generators produce pseudo-random numbers that are deterministic given a random seed. Through sharing a random seed across computing nodes, and ensuring that this random seed is used for (24) only, we are guaranteed that the sequence of $(\beta_k)_{K+1}^{\infty}$ and $(\phi_k)_{K+1}^{\infty}$ are the same for all machines. This idea of replaying random numbers by sharing a random seed has been used in sequential Monte Carlo algorithms to reduce memory cost (see e.g., Jun et al. (2012)).

6. Empirical Evaluations

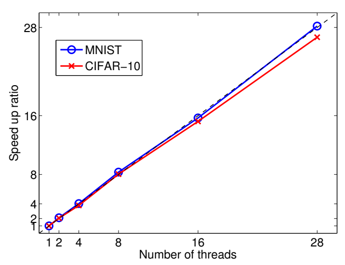
The Map–Reduce pseudo code for our sampler is given by algorithms 1 and 2. Every iteration of the sampling algorithm is implemented by one Map step followed by one Reduce step. In the Map step, the slice variable and cluster indicator of every data point are sampled in parallel on M map workers, and sufficient statistics are accumulated locally. The time complexity will be $\mathcal{O}(K^*N/M)$ because all $\mathcal{O}(N)$ computations will be performed in the Map step. Thus, the Map–Reduce sampler would speed up linearly with the number of map workers. In the Reduce step, the sufficient statistics from every mapper are accumulated by



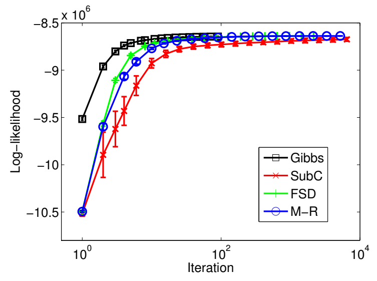
(a) Log-likelihood vs. iterations in log scale.



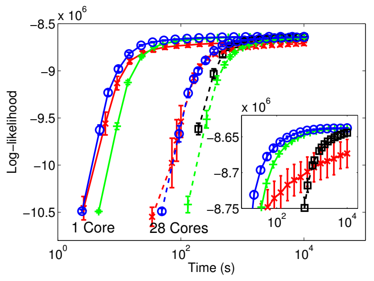
(b) Log-likelihood vs. running time in log scale.



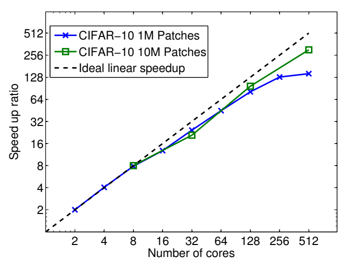
(a) Local parallelisation



(c) Log-likelihood vs. iterations in log scale.



(d) Log-likelihood vs. running time in log scale.



(b) Distributed parallelisation

Figure 1. Training log-likelihood with standard deviation for Gibbs (square), SubC (cross), FSD (+), and M–R (o) on MNIST (first row) and CIFAR-10 (second row) data sets, with 1 (dashed lines) and 28 (solid lines) threads. Insets in (b) and (d) are zoom-in of the top part.

Figure 2. Speed-up ratio of M–R on MNIST and CIFAR-10. 1 and 10 million CIFAR-10 patches are used in distributed experiments.

one reducer. It then samples all the global variables, including concentration α (and γ for HDP), component parameters $\boldsymbol{\phi}$, and the component weights $\boldsymbol{\beta}$. The time complexity is $\mathcal{O}(KM + K^*)$. The communication cost between mappers and the reducer depends on the total number of variables to be transfered, DK^*M, where D is the dimensionality of ϕ_k or that of the sufficient statistics.

In order to evaluate the performance and efficiency of the proposed slice sampling algorithm, we run experiments on a DP Gaussian mixture model with an conjugate normal-inverse-Wishart prior, and a HDP Multinomial mixture models with a Dirichlet prior. We evaluate our sampler in two different settings: local thread-level parallelisation and distributed machine-level parallelisation on Amazon EC2 clusters. Both experiments are performed using Amazon EC2 instances with up to 32 cores. For experiments with more than 32 cores, we use a cluster of c3.8xlarge instances each with 32 cores.

6.1. The DP Gaussian Mixture Model

We study the performance of the DP sampler on two data sets: the MNIST digit images and CIFAR-10 natural colour images with standard pre-processing steps. For MNIST data, we apply PCA whitening and reduce the dimensionality to 50. For CIFAR-10, we randomly draw 100K 8×8 patches with RGB channels, apply local contrast normalization, then PCA and reduce the dimensionality to 72 retaining 99% of variance. We draw 1 million patches for the distributed parallelisation experiment.

We compare our slice sampler (M–R) to the collapsed Gibbs sampling based on the marginal representation (Gibbs) (Neal, 2000), a subcluster-supercluster algorithm Chang & Fisher III (2013) based on split-merge moves (SubC), and an approximate inference algorithm with a finite symmetric Dirichlet prior (FSD) (Chang & Fisher III, 2013). The latter two algorithms are also parallelisable. For the FSD, a truncation level of 100 is used in all the experiments. We compare all the algorithms on a single machine with 32 cores with local parallelisation (no distributed implementation is available yet for competitors). For all experiments, we initialise the concentration parameter $\alpha \sim \mathcal{G}(1,1)$, and randomly assign all the observations into 50 clusters. All algorithms are run 10 times to estimate the standard deviation.

6.1.1. Mixing Rate

We first study the mixing rate of all algorithms in terms of joint log-likelihood of all the data. The first column of

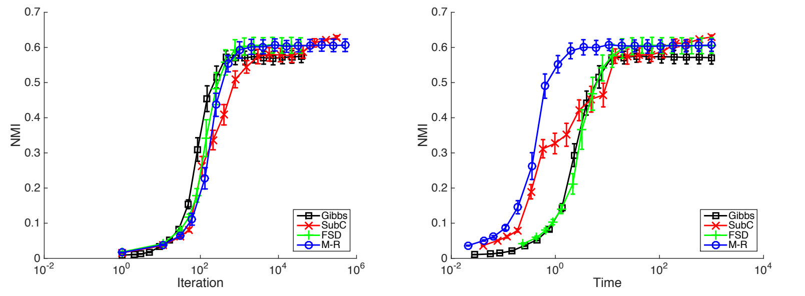


Figure 3. NMI of Gibbs (square), SubC (cross), FSD (+), and M–R (o) on a synthetic data set.

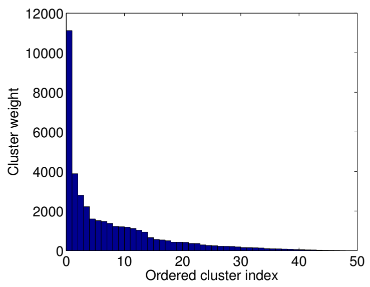


Figure 4. The fast decaying behaviour of component weights on the NYTimes dataset.

Figure 1 shows the convergence of log-likelihood against the number of iterations, and the dashed lines in the second column are plotted against the actual running time of single-threaded runs. Clearly, the collapsed Gibbs sampler achieves the best mixing rate given the same number of iterations. This is as expected because the Gibbs sampler marginalises over cluster weights and parameters. When compared to FSD, the M–R sampler converges at a similar rate per iteration but is much faster per running time because FSD relies on a large truncation level in order to guarantee a good approximation to DP. Also, M–R takes advantage of the slice variable and achieves further speed up as explained in Section 4.2. Lastly, SubC tends to mix slowest among all the algorithms per iteration in Figure 1(a) and (c). This might be due to the special restricted Gibbs sampling step that confines the new cluster assignment in a subset of the mixture components.

6.1.2. Structure Discovery

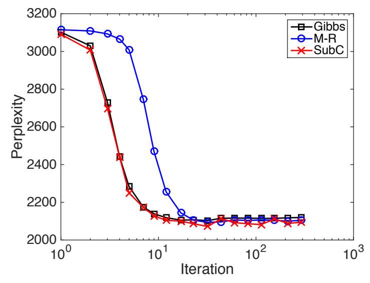
We also study the ability of recovering the true clustering structure, we apply different samplers to a synthetic data set that consists of 10,000 data points, sampled from a mixture of 50 univariate Gaussian distributions. We assess the quality of recovered structure using normalised mutual information (NMI, see e.g., Manning et al. (2008)) between the true and inferred clustering components.

Figure 3 shows the estimated NMI with 28 parallel threads over 10 runs. The comparison between the M–R and the collapsed Gibbs is consistent with those on MNIST and CIFAR-10 (Figure 1). Both algorithms converge to about the same values of both log-likelihood (not shown due to the space limit) and NMI. The slice sampler mixes slower per iteration compared to the collapsed Gibbs sampler, but significant faster per time unit. Surprisingly, we also observe that the split-merge based algorithm, SubC, mixes slower than the slice sampler. However, we also notice that given longer running time, the SubC sampler will eventually have better convergence than others. This is as expected because we use a non-informative prior on the mixture component parameters and in that situation methods based on pure Gibbs sampling are known to have difficulty in generating new clusters.

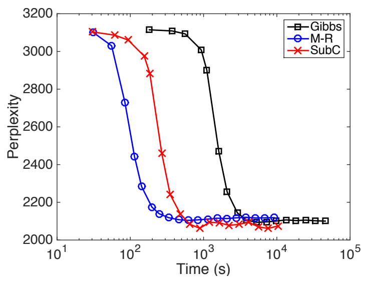
6.1.3. Speed-up Ratio

Lastly, we study the benefits of parallelisation. The solid lines in the second column of Figure 1 show the log-likelihood against running time locally parallelised with 28 threads for all the algorithms except the collapsed Gibbs sampler. The M–R sampler achieves the fastest convergence rate. On the CIFAR-10 data set, its log-likelihood reaches convergence about two orders of magnitude faster than the Gibbs sampler, even before the Gibbs sampler finishes the first iteration. We also notice that SubC is stuck at a sub-optimal state, and does not achieve the same log-likelihood within $3,000$ seconds.

Figure 2(a) shows the speed-up ratio of the running time with local parallelization. The M–R sampler achieves nearly a linear speed-up. This suggests that the computation time spent on the reducer is negligible compared to the mapper even with 28 workers. For distributed parallelization, we test the scalability on CIFAR-10 data set with 1 and 10 million patches. Figure 2(b) shows the speed-up ratio using up-to 16 machines with totally 512 cores. In the distributed environment, scheduling and communication cost play an important role, and it is harder to obtain a linear speed up. Nevertheless, we still reduces the running time per iteration from 6 min (2 cores) to 3.8 sec (512 cores) for the 1M data set, and from 12.7 min (8 cores) to 18 sec (512 cores) for the 10M data set. For the latter case, the discrepancy from a linear speed up is mainly due to hyper-threading of EC2 instances, that is, there are only 16 physical cores on each 32 core machine.



(a) Perplexity vs. iterations.



(b) Perplexity vs. running time.

Figure 5. Perplexity on the NIPS data set with 28 threads.

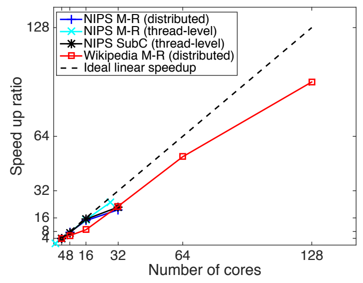


Figure 6. Speed up of M–R on the NIPS and Wikipedia corpus.

6.2. The HDP Multinomial Mixture Model

The performance of the proposed M–R sampler for the HDP mixture model is evaluated on the NIPS corpus (1.9 million words) and a subset of the Wikipedia corpus constructed by randomly selecting 10^5 documents (roughly 40 million words). The algorithms under comparison are the M–R sampler, the SubC (Chang & Fisher III, 2014), and the collapsed Gibbs sampler based on the direct assignment scheme (Teh et al., 2006). The performance is measured in terms of predictive perplexities on 10% separate hold-out test documents for both the NIPS and Wikipedia datasets. To compute perplexity of these test documents, we used a document completion approach described in Newman et al. (2009).

6.2.1. Mixing Rate

Figure 5 shows the perplexity against iterations and running time. We observe that the collapsed Gibbs and SubC samplers have similar convergence given the same number of iterations, both converge faster than the M–R sampler. This result is not surprising because the M–R sampler does not marginalise out component parameters. The performance of the M–R is better than SubC and collapsed Gibbs in terms of running time due to parallelisation. This result is consistent with the experiment for the DP mixture model.

6.2.2. Speed-up Ratio

To evaluate the speed-up for the HDP M–R sampler, we also varied the number of cores for the NIPS dataset: $\{1, 4, 16, 28\}$ with thread-level parallelisation and $\{4, 8, 16, 32\}$ with distributed parallelisation; for the Wikipedia dataset $\{4, 8, 16, 32, 64, 128\}$ with distributed parallelisation. Figure 6 shows the speed up: For the NIPS dataset we observe a near linear speed up with thread-level parallelisation. We also observe similar speed up with distributed parallelisation when the number of workers is smaller than 16 but starts to level off when multiple machines are used. This is unsurprising because the NIPS dataset is relative small, when the number of machines is sufficiently large, communication overhead starts to dominate. The running time per iteration on the NIPS data set is reduced from 82.7 seconds (1 core) to 4.1 seconds (32 cores) where the reduce step in the latter case takes 2.4 seconds. For the Wikipedia dataset, of which the dataset size is much larger, speed up with distributed parallelisation is better. The running time per iteration was reduced from 32 minutes (4 cores) to 73 seconds (128 cores).

7. Conclusion

We presented a simple yet efficient distributed inference algorithm under the Map–Reduce framework for the DP and HDP mixture model. The proposed sampler is based on an exact slice sampling representation of infinite mixtures. The performance of the sampler achieves state-of-the-art modelling performance on image and text data sets using a fraction of computing time of the standard collapsed Gibbs sampler. Because the slice sampler does not involve any approximation (unlike e.g., FSD), it will converge to the true posterior given enough running time. Moreover, we show that the Map–Reduce sampler scales nicely with respect to the number of computing units.

Acknowledgments

We would like to thank Nilesh Tripuraneni, Creighton Heaukulani, Dan Roy, and anonymous reviewers for helpful discussions and feedback on drafts. We acknowledge EPSRC grant EP/I036575/1, the DARPA PPAML programme, a Google Focused Research Award and support from Amazon and Microsoft Research. HG is supported by a Wellcome Trust PhD Fellowship.

References

Ahn, Sungjin, Shahbaba, Babak, and Welling, Max. Distributed stochastic gradient MCMC. In *Proceedings of the 31st International Conference on Machine Learning*

(ICML-14), pp. 1044–1052, 2014.

Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3: 993–1022, 2003.

Chang, J and Fisher III, J. W. Parallel Sampling of HDPs using Sub-Cluster Splits. In *Proceedings of the Neural Information Processing Systems (NIPS)*, Dec 2014.

Chang, Jason and Fisher III, John W. Parallel Sampling of DP Mixture Models using Sub-Cluster Splits. In *Advances in Neural Information Processing Systems*, pp. 620–628, 2013.

Doshi-Velez, Finale, Mohamed, Shakir, Ghahramani, Zoubin, and Knowles, David A. Large scale nonparametric Bayesian inference: Data parallelisation in the Indian buffet process. In *Advances in Neural Information Processing Systems*, pp. 1294–1302, 2009.

Escobar, Michael D and West, Mike. Computing nonparametric hierarchical models. In *Practical nonparametric and semiparametric Bayesian statistics*, pp. 1–22. Springer, 1998.

Ferguson, Thomas. Bayesian analysis of some nonparametric problems. *The annals of statistics*, pp. 209–230, 1973.

Gal, Yarin and Ghahramani, Zoubin. Pitfalls in the use of parallel inference for the Dirichlet process. In *Proceedings of the 31th International Conference on Machine Learning (ICML-14)*, 2014.

Griffiths, Thomas L and Ghahramani, Zoubin. The Indian buffet process: An introduction and review. *The Journal of Machine Learning Research*, 12:1185–1224, 2011.

Jain, Sonia and Neal, Radford M. A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13(1), 2004.

Jun, Seong-hwan, Wang, Liangliang, and Bouchard-Côté, Alexandre. Entangled Monte Carlo. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 2726–2734. Curran Associates, Inc., 2012.

Kalli, Maria, Griffin, Jim E, and Walker, Stephen G. Slice sampling mixture models. *Statistics and computing*, 21 (1):93–105, 2011.

Manning, Christopher D, Raghavan, Prabhakar, and Schütze, Hinrich. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

Mnih, Andriy and Salakhutdinov, Ruslan. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pp. 1257–1264, 2007.

Neal, Radford M. Markov chain sampling methods for Dirichlet process mixture models. *Journal of computational and graphical statistics*, 9(2):249–265, 2000.

Newman, David, Asuncion, Arthur, Smyth, Padhraic, and Welling, Max. Distributed algorithms for topic models. *The Journal of Machine Learning Research*, 10:1801–1828, 2009.

Papaspiliopoulos, Omiros and Roberts, Gareth O. Retrospective Markov chain Monte Carlo methods for Dirichlet process hierarchical models. *Biometrika*, 95(1):169–186, 2008.

Pitman, J. Some developments of the Blackwell-MacQueen urn scheme. In *Statistics, probability and game theory*, volume 30 of *IMS Lecture Notes Monogr. Ser.*, pp. 245–267. Inst. Math. Statist., Hayward, CA, 1996.

Sethuraman, J. A constructive definition of Dirichlet priors. *Statist. Sinica*, 4:639–650, 1994.

Teh, Y. W. Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer, 2010.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. Hierarchical Dirichlet processes. *J. Amer. Statist. Assoc.*, 101 (476):1566–1581, 2006.

Teh, Yee W, Görür, Dilan, and Ghahramani, Zoubin. Stick-breaking construction for the Indian buffet process. In *International Conference on Artificial Intelligence and Statistics*, pp. 556–563, 2007.

Van Gael, Jurgen, Saatci, Yunus, Teh, Yee Whye, and Ghahramani, Zoubin. Beam sampling for the infinite hidden Markov model. In *Proceedings of the 25th international conference on Machine learning*, pp. 1088–1095. ACM, 2008.

Walker, Stephen G. Sampling the Dirichlet mixture model with slices. *Communications in Statistics–Simulation and Computation*, 36(1):45–54, 2007.

Williamson, Sinead, Dubey, Avinava, and Xing, Eric. Parallel Markov chain Monte Carlo for nonparametric mixture models. In *Proceedings of the 30th International Conference on Machine Learning*, pp. 98–106, 2013.