Distributed Stochastic Gradient MCMC

Ahn, S.; Shahbaba, B.; Welling, M.

Publication date
2014
Document Version
Final published version
Published in
JMLR Workshop and Conference Proceedings

Distributed Stochastic Gradient MCMC

Sungjin Ahn SUNGJIA@ICS.UCI.EDU
Department of Computer Science, University of California, Irvine

Babak Shahbaba BABAKS@UCI.EDU
Department of Statistics, University of California, Irvine

Max Welling M.WELLING@UVA.NL
Machine Learning Group, University of Amsterdam

Abstract

Probabilistic inference on a big data scale is becoming increasingly relevant to both the machine learning and statistics communities. Here we introduce the first fully distributed MCMC algorithm based on stochastic gradients. We argue that stochastic gradient MCMC algorithms are particularly suited for distributed inference because individual chains can draw mini-batches from their local pool of data for a flexible amount of time before jumping to or syncing with other chains. This greatly reduces communication overhead and allows adaptive load balancing. Our experiments for LDA on Wikipedia and Pubmed show that relative to the state of the art in distributed MCMC we reduce compute time from 27 hours to half an hour in order to reach the same perplexity level.

1. Introduction

Probabilistic inference methods that can operate on a very large data scale are becoming increasingly relevant in an era that data volume grows exponentially. Two promising directions in this respect are stochastic gradient variational inference (Hoffman et al., 2010) and stochastic gradient MCMC algorithms (Welling & Teh, 2011; Ahn et al., 2012; Patterson & Teh, 2013). The main innovation for both classes of algorithms is that only a small mini-batch of data is necessary for every update, allowing many more updates per time interval. In the context of MCMC this leads to much shorter burn-in times and faster mixing speeds.

In this paper we are concerned with parallelizing stochastic gradient MCMC algorithms. The most straightforward, embarrassingly parallel implementation would be to copy the full dataset to each worker, run separate Markov chains and use their results as independent samples (see e.g. (Wilkinson, 2006; Laskey & Myers, 2003; Ahn et al., 2013)). However, the size of modern day datasets can be so large that a single machine cannot store the full dataset. In this case, one can still parallelize most MCMC algorithms by performing data-specific computations (e.g. the gradient of the log-probability for one data-case) locally on each relevant worker and combining these computations in a master server. The disadvantage of these methods is however that they lead to very high communication costs.

We argue that MCMC algorithms based on stochastic mini-batches have a key property that make them ideally suited for parallelization, *namely that each Markov chain can independently generate samples for a variable amount of time*, which can later be combined. The reason is that each chain can draw mini-batches from its local pool of data in order to generate samples. Chains must jump to other machines (synchronously or asynchronously) in order to generate unbiased estimates of the posterior in the limit, but the time spend on each worker is flexible provided that the chain's hyper-parameters are properly adjusted to remove potential bias. This flexibility leads to less communication (because chains can run longer on individual workers) and entirely removes the problem that fast workers are blocked by slower workers because they depend on their results in order to proceed.

We present distributed stochastic gradient Langevin dynamics (D-SGLD) and apply it to topic modeling. In this setting, we show that relative to the current fastest sequential MCMC sampler (Patterson & Teh, 2013), and the fastest approximate distributed MCMC samplers (Newman et al., 2007; Ahmed et al., 2012; Smola & Narayanamurthy, 2010), D-SGLD achieves equivalent perplexities at least an order of magnitude faster.

Proceedings of the 31^{st} International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP volume 32.

2. Preliminaries

Let $X = \{x_1, \ldots, x_N\}$ be a dataset of N i.i.d. data points assumed to be sampled from a parameterized distribution $p(x|\theta)$ where $\theta \in \mathbb{R}^d$ has a prior distribution $p(\theta)$. We are interested in collecting samples from the posterior distribution $p(\theta|X) \propto p(X|\theta)p(\theta)$. As discussed above, we assume that the dataset X is too large to reside in a single machine. Therefore, it is partitioned into S subsets, called *shards*: $X_1, \ldots, X_S$ such that $X = \cup_s X_s$ and $N = \sum_s N_s$. We assign shard $X_s = \{x_1^s, \ldots, x_{N_s}^s\}$ to worker s, where $s = 1, \ldots, S$. We refer to the posterior distribution based on a specific shard as *local posterior*: $p(\theta|X_s) \propto p(X_s|\theta)p(\theta)$.

The score function or the gradient of the log likelihood given a data point x is denoted by $g(\theta; x) = \nabla_\theta \log p(\theta; x)$. We also denote a mini-batch of n data points by X^n when sampled from X and by X_s^n when sampled from shard X_s. Additional time index t is used sometimes to distinguish mini-batches sampled over iterations: $X_{s,t}^n$. The sum and mean of scores over all elements of a set, X, are denoted by $G(\theta; X) = \sum_{x \in X} g(\theta; x)$ and $\bar{g}(\theta; X) = \frac{1}{|X|} G(\theta; X)$ respectively. We now review two approaches to scale up MCMC algorithms; one by using mini-batches and the other by using distributed computational resources.

2.1. Mini-batch-based MCMC

The stochastic gradient Langevin dynamics (SGLD) proposed by Welling and Teh (2011) is the first sequential mini-batch-based MCMC algorithm. In SGLD, the parameters are updated as follows:

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon_t}{2}\{\nabla \log p(\theta_t) + N\bar{g}(\theta_t; X_t^n)\} + \nu_t. \quad (1)$$

Here ϵ_t is the step size and $\nu_t \sim \mathcal{N}(0, \epsilon_t)$ is the injected Gaussian noise. The gradient of the log-likelihood, $G(\theta_t; X)$, over the whole dataset is approximated by scaling the mean score, $\bar{g}(\theta_t; X_t^n) = \frac{1}{n} \sum_{x \in X_t^n} g(\theta_t; x)$, computed based on a mini-batch X_t^n of size $n \ll N$. SGLD does not use accept-reject tests because as the step size goes to zero the acceptance rate tends to one. Therefore, SGLD requires only $\mathcal{O}(n)$ computations to generate each sample, unlike traditional MCMC algorithms that require $\mathcal{O}(N)$ computations per iteration. Because computing $\bar{g}(\theta_t, X_t^n)$ in parallel within a multi-core worker is straightforward, throughout the paper we assume that each worker is single-core.

We can generalize the SGLD update rule in Eqn. (1) by replacing the mean score $\bar{g}(\theta_t; X_t^n)$ to a general form of score estimator $f(\theta_t, Z; X)$, where Z is a set of auxiliary random variables associated with the estimator. According to Welling & Teh (2011), an estimator $f(\theta_t, Z; X)$ is guaranteed to converge to the correct posterior if (i) $f(\theta_t, Z; X)$ is an unbiased estimator of $\bar{g}(\theta_t; X) = \frac{1}{N} \sum_{x \in X} g(\theta_t; x)$ (assuming the variance of f is finite) and (ii) the step size is annealed to zero by a schedule satisfying $\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$.

Definition 1. We define an estimator $f(\theta, Z; X)$ as a *valid SGLD estimator* if it is an unbiased estimator of $\bar{g}(\theta; X)$, i.e., $\mathbb{E}_Z[f(\theta, Z; X)] = \bar{g}(\theta; X)$, where $\mathbb{E}_Z$ denotes expectation w.r.t. the distribution $p(Z; X)$, and it has finite variance $\mathbb{V}_Z[f(\theta, Z; X)] < \infty$.

For an alternative way to speed-up MCMC by using a mini-batch-based Metropolis-Hastings (MH) test, refer to Korattikara et al. (2014); Bardenet et al. (2014).

2.2. Distributed Inference in LDA

Distributing the workload using a cluster of workers is another way of speeding up MCMC. In this paper we are interested in topic models for which a number of distributed MCMC algorithms have already been developed (our method is more generally applicable however). In approximate distributed LDA (AD-LDA) by Newman et al. (2007) the computation cost per sample is reduced to $\mathcal{O}(\frac{N}{S})$ by allowing each worker to perform collapsed Gibbs sampling only on its local shard. AD-LDA also corrects (approximately) the biases in the local copies of the global states by allowing for regular global synchronization.

However, AD-LDA suffers from some shortcomings. First, it becomes slower as the dataset size increases, unless additional workers are provided. Second, due to the global synchronization, it suffers from the "block-by-the-slowest" problem, meaning that some workers are blocked until the slowest worker finishes its task. Lastly, running parallel chains usually adds large overhead. Yahoo-LDA (Y-LDA) (Ahmed et al., 2012) performs asynchronous updates to resolve the block-by-the-slowest problem. However, the unbounded asynchronous updates could deteriorate performance (Ho et al., 2013).

3. Distributed Stochastic Gradient Langevin Dynamics (D-SGLD)

3.1. SGLD on Partitioned Datasets

We begin the exposition of our algorithm with the following question: *"Is an SGLD algorithm that samples mini-batches from randomly chosen local shards valid?"* The number of possible combinations of mini-batches that can be generated by this procedure is significantly smaller set than that of the standard SGLD. The answer will clearly depend on quantities like the shard sizes and shard selection probabilities. We now introduce an estimator $\bar{g}_d$ in the proposition below as an answer to the above question (the proof is provided in the supplementary material).

Proposition 3.1. *For each shard $s = 1, \dots, S$, given the shard size, N_s, and the normalized shard selection frequency, q_s, such that $N_s > 0$, $\sum_{s=1}^{S} N_s = N$, $q_s \in (0, 1)$, and $\sum_{s=1}^{S} q_s = 1$, the following estimator is a valid SGLD estimator,*

$$\bar{g}_d(\theta; X_s^n) \stackrel{def}{=} \frac{N_s}{N q_s} \bar{g}(\theta; X_s^n) \tag{2}$$

where shard s is sampled by a scheduler $h(\mathcal{Q})$ with frequencies $\mathcal{Q} = \{q_1, \dots, q_S\}$.

For example, we can (1) choose a shard by sampling $s \sim h(\mathcal{Q}) = \text{Category}\,(q_1, \dots, q_S)$, (2) sample a mini-batch X_s^n from the selected shard, (3) compute mean score $\bar{g}(\theta; X_s^n)$ using that mini-batch, and then (4) multiply the mean score by $\frac{N_s}{N q_s}$ to correct the bias. Then, the resulting SGLD update rule becomes

$$\theta_{t+1} \leftarrow \theta_t + \frac{\epsilon_t}{2} \left\{ \nabla \log p(\theta_t) + \frac{N_{s_t}}{q_{s_t}} \bar{g}(\theta_t; X_{s_t}^n) \right\} + \nu_t. \tag{3}$$

We can interpret this as a correction to the step sizes for the $\bar{g}(\theta_t; X_{s_t}^n)$ term. That is, the algorithm takes larger steps for shards that are relatively larger in size and/or used less frequently than others. This implies that every data-case contributes equally to the mixing of the chain. Note that $\mathcal{Q}$ represent free parameters that we can choose depending on the system properties.

3.2. Traveling Worker Parallel Chains

Now assume that the shards are distributed between the workers, so from now on selecting shard s is equivalent to choosing worker s. We note that running the above algorithm occupies only a single worker at a time. Therefore, assuming single-core workers, it is possible to run $C\ (\leq S)$ independent and valid SGLD chains *in parallel*, i.e., one chain per worker.

This approach, however, has some shortcomings. First, the communication cycle is still short $\mathcal{O}(n)$ because each chain is required to jump to a new worker at every iteration. Second, it can suffer from the block-by-the-slowest problem if its next scheduled worker is still occupied by another chain due to workers' imbalanced response delays. The *response delay*, denoted by d_s, is defined as the elapsed time that worker s spends to process a $\mathcal{O}(n)$ workload. In the following sections, we present our method to address these issues.

3.2.1. Distributed Trajectory Sampling

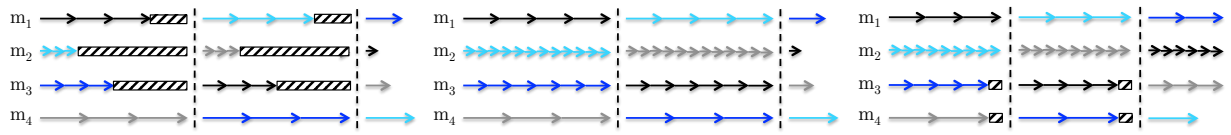
To deal with the "short-communication-cycle" problem, we propose to use *trajectory sampling*: instead of jumping to another worker at every iteration, each chain c takes τ consecutive updates in each visit to a worker. Then, after τ updates, only the last (τth) state is passed to the next worker of the chain. Trajectory sampling reduces communication overhead by increasing the communication cycle from $\mathcal{O}(n)$ to $\mathcal{O}(\tau n)$. Furthermore, instead of transferring all samples collected over a trajectory to the master, we can store them in a distributed way by caching each trajectory at its corresponding worker. This keeps the packet size at $\mathcal{O}(1)$ regardless of the trajectory length, and mitigates the memory problem caused by storing many high-dimension samples at a single machine.

In trajectory sampling for parallel chains, we employ a scheduler $h_c(\mathcal{Q})$ for each chain c to choose the *next worker* from which the next trajectory is sampled. Note here that the scheduler is now called with an interval τ. Because there are a total of C such schedulers (one per chain), the schedulers should avoid two situations in order to be efficient: (1) *collision* (i.e., multiple chains visit a worker at the same time), and (2) *jump-in-place* (i.e., jumping to the current worker) can negatively affect mixing across shards. One way to avoid these issues is to set $\mathcal{Q}$ uniform, and simply use a random permutation (or, cyclic rotation) to assign chains to workers. That is, we can sample the chain-to-worker assignments by $(s^1, \dots, s^C) \sim (h_1(\mathcal{Q}), \dots, h_C(\mathcal{Q})) = \text{RANDPERM}(S)$. Here, s^c denotes a worker that chain c is scheduled to visit; we assume $C = S$ for simplicity.

Similar to the effect of step sizes in standard SGLD, trajectory lengths can also be used to control the level of approximation by trading off computation time with asymptotic accuracy. As both the trajectory length and the annealed step sizes $\{\epsilon_t\}$ can affect the equilibrium distribution of the chain, we consider first that ϵ is fixed. Then, with a long trajectory, we can reduce the communication overhead at the cost of some loss in asymptotic accuracy. In fact, it is not difficult to see that in this case our method samples from a mixture of local posteriors, $\frac{1}{S} \sum_{s=1}^{S} p_\epsilon(\theta | X_s)$ at one end of the spectrum where long trajectory lengths are used, and it approaches the true posterior at the other end of the spectrum with short trajectory lengths (ϵ is small enough).

Note that this is indeed the desired behavior when dealing with massive datasets. That is, as $N \to \infty$, the local posteriors become close to the true posterior and thus the error decreases by the central limit theorem (provided X_s is a uniform random partition of X): $\bar{g}(\theta; X_s) \sim \mathcal{N}(\mathbb{E}[g(\theta; x)], \text{Cov}[g(\theta; x)]/N_s)$. Therefore, as the dataset increases, we can increase the trajectory length accordingly without a significant loss in asymptotic accuracy. The following Corollary 3.2 states that for any finite τ, trajectory sampling is a valid SGLD (assuming the step sizes decrease to zero over time).

Corollary 3.2. *A trajectory sampler with a finite $\tau \geq 1$, obtained by redefining the worker (shard) selection process $h(\mathcal{Q})$ in Proposition 3.1 by the process $h(\mathcal{Q}, \tau)$ below, is a*



(a) Without load balancing, $\tau = 3$ (b) With load balancing, $\bar{\tau} = 25/4$ (c) With load balancing, $\bar{\tau} = 75/16$

Figure 1. Illustration of adaptive load balancing. Each row represents a worker and the chains are represented by different colors. The box filled by diagonal lines are block-time, and at the vertical dotted lines represent chains jumping to other workers. A sample is collected at each arrow whose length represents the time required to collect the sample. In the present example, four workers have different response delays, 3, 1, 2, and 4, respectively. In (a) τ is set to a constant $\tau = 3$ for all workers, and in (b) with $\bar{\tau} = \frac{25}{4}$, the trajectory plan becomes $\mathcal{T} = (4, 12, 6, 3)$, and in (c), $\mathcal{T} = (3, 9, 4.5, 2.25)$ with $\bar{\tau} = \frac{75}{16}$.

valid SGLD sampler. $h(\mathcal{Q}, \tau)$: for chain c at iteration t, choose the next worker s^c_{t+1} by

$$s^c_{t+1} = \begin{cases} \tilde{h}(\mathcal{Q}), & \text{if } t = k\tau \text{ for } k = 0, 1, 2, \dots \\ s^c_t, & \text{otherwise,} \end{cases} \tag{4}$$

where $\tilde{h}(\mathcal{Q})$ is an arbitrary scheduler with selection probabilities $\mathcal{Q}$.

3.2.2. Adaptive Load Balancing

Using trajectory sampling, we can mitigate the short-communication-cycle problem. Moreover, if response delays are balanced, we can set $\mathcal{Q}$ to be uniform and use a random permutation scheduler to keep the block-by-the-slowest delay small. However, for imbalanced response delays, using uniform $\mathcal{Q}$ would lead to long block-by-the-slowest delays (See, Fig. 1 (a)). In this section, we propose a solution to balance the workloads by adapting $\mathcal{Q}$ to the worker response delays.

The basic idea is to make the faster workers work longer until the slower workers finish their tasks so that the overall response times of the workers become as balanced as possible. For instance, twice longer trajectories can be used for a worker that is twice as fast. More specifically, we achieve this by (1) having uniform worker selection and (2) setting the trajectory length τ_s of worker s to $\tau_s = q_s \bar{\tau} S$; here, q_s is set to $d_s^{-1} / \sum_{z=1}^{S} d_z^{-1}$ (i.e., the relative speed of worker s), and $\bar{\tau}$ is a user-defined mean trajectory length: $\mathbb{E}[\tau_s] = \sum_s \frac{1}{S} q_s \bar{\tau} S = \bar{\tau}$ (the expectation is w.r.t. the worker selection probability $1/S$).

In other words, we select a worker uniformly and perform trajectory sampling of length τ_s, which is proportional to the relative speed of the worker, q_s. (If τ_s is not an integer, we can either adjust $\bar{\tau}$ to make it integer or take simply the closest integer.) Note that using unequal trajectory lengths across the workers remains a valid SGLD because the step sizes are properly corrected by Eqn. (2) where $q_s \propto \tau_s$.

This is illustrated in Figure 1 and stated in Corollary 3.3.

Corollary 3.3. *Given τ_s, where $1 \leq \tau_s < \infty$ for $s = 1, \dots, S$, the adaptive trajectory sampler, obtained by redefining the worker (shard) selection process $h(\mathcal{Q})$ in Proposition 3.1 by the process $h(\mathcal{Q}, \{\tau_s\})$ below, is a valid SGLD sampler. $h(\mathcal{Q}, \{\tau_s\})$: for chain c at iteration t, choose the next worker s^c_{t+1} by*

$$s^c_{t+1} = \begin{cases} \tilde{h}(1/S), & \text{if } t = k\tau_{s^c_t} \text{ for } k = 0, 1, 2, \dots \\ s^c_t, & \text{otherwise,} \end{cases} \tag{5}$$

where $\tilde{h}(1/S)$ is a scheduler with uniform selection probabilities.

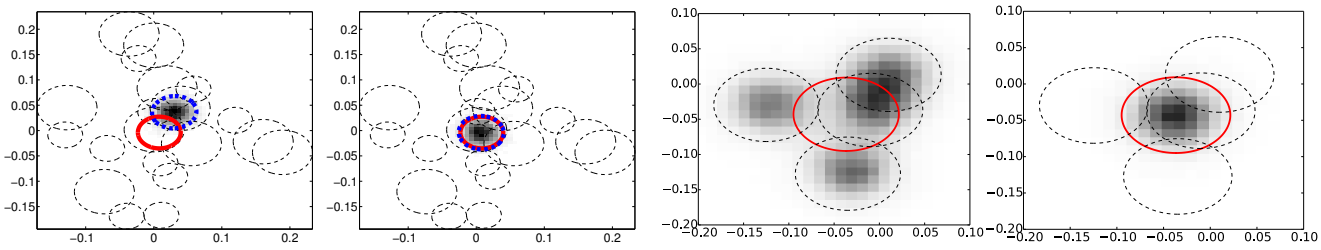
Our method can deal with temporal imbalances as well. To this end, the master needs to monitor the changes in response delays; when a substantial change is detected, a new trajectory plan can replace the old one. Note that although this online adaptation affects the Markov property, it can still converge to correct target distribution assuming that the adaptation satisfies the Corollary 3.3 and the response delays converge fast enough. Refer to Andrieu & Thoms (2008) for the details of the "fast enough" condition. Pseudo code for the proposed D-SGLD method is presented in Algorithm 1.

3.2.3. Variance Reduction by Chain Coupling

Here, we introduce one approach that could reduce the variance of the gradient estimator in Eqn (2) by having some interactions among the chains. The basic idea is to "tie" a *group* of chains by averaging their corresponding samples. More specifically, consider $R \leq S$ chains forming a group and staying at a state θ_t at time t, i.e., $\theta^r_t = \theta_t$ for $r = 1, \dots, R$. After an update using the standard SGLD update rule in Eqn. (1), we have R different states $\theta^1_{t+1}, \dots, \theta^R_{t+1}$. By averaging the new states, we have $\theta_{t+1} = \frac{1}{R} \sum_{r=1}^{R} \theta^r_{t+1}$, which is

$$\theta_t + \frac{\epsilon_t}{2} \left\{ \nabla \log p(\theta_t) + \frac{N}{nR} \sum_{x \in \cup_r X^n_{t,r}} g(\theta_t; x) \right\} + \bar{\nu}_t \tag{6}$$

Here we used $\frac{1}{R} \sum_{r=1}^{R} \nabla \log p(\theta^r_t) = \nabla \log p(\theta_t)$ and $\frac{1}{R} \sum_{r=1}^{R} \theta^r_t = \theta_t$ noting that $\theta^r_t = \theta_t$ for all r. Note that



(a) Without correction (b) With correction (c) τ = 10,000 (d) τ = 200

Figure 2. Bias correction and trajectory length effects.

Algorithm 1 D-SGLD Pseudo Code

function MASTER($S, C, \bar{\tau}$)
  while sampling **do**
    Monitor response delays $\{d_s\}$
    if $\{d_s\}$ are changed enough **then**
      Adapt $\tau_s \leftarrow \bar{\tau} S d_s^{-1} / \sum_{z=1}^{S} d_z^{-1}, \forall s$
    end if
    Assign workers $(s^1, \ldots, s^C) \sim$ RANDPERM(S)
    for each chain c ***parallel*** **do**
      $\theta_c \leftarrow$ SAMPLE_TRAJ(s^c, c, θ_c, τ_s) (line 13)
    end for
  end while
end function
function SAMPLE_TRAJ(c, θ, τ_s)
  Initialize $\theta_1 \leftarrow \theta$
  for $t = 1 : \tau_s$ **do**
    Sample a mini-batch X_s^n and noise $\nu \sim \mathcal{N}(0, \epsilon)$
    Obtain θ_{t+1} by Eqn. (3) (set $q_s = \frac{\tau_s}{\bar{\tau} S}$)
  end for
  Set trajectory, $T_c \leftarrow (\theta_1, \ldots, \theta_{\tau_s - 1})$
  Store (append) trajectory, $\Theta_s^c \leftarrow [\Theta_s^c, T_c]$
  Send the last state θ_{τ_s} to the master
end function

although the averaged noise $\bar{\nu}_t = \frac{1}{R} \sum_{r=1}^{R} \nu_t^r$ has smaller variance $\mathcal{N}(0, \frac{\epsilon_t}{R})$ than the standard SGLD, we can recover a valid SGLD with additional noise[1] $\eta_t \sim \mathcal{N}(0, \frac{R-1}{R}\epsilon_t)$ so that $\bar{\nu}_t + \eta_t \sim \mathcal{N}(0, \epsilon_t)$. By the central limit theorem, the variance of the estimator of $\bar{g}(\theta_t; X)$ reduces from $\frac{\text{Cov}[g(\theta;x)]}{n}$ to $\frac{\text{Cov}[g(\theta;x)]}{Rn}$.

Although this approach leads to a valid SGLD with reduced variance in the gradient estimation, unfortunately it is difficult to perform the trajectory sampling in this case since it requires communication among chains at each update. One way to bypass this issue is to employ the averaging strategy only at the end of each trajectory during the burn-in period. Alternatively, we can also gradually reduce the number of chains being coupled. Note that although coupling chains imposes algorithmic dependency, it is weaker than other algorithms (e.g., AD-LDA) that require synchronizations for *all* workers since (i) the number of dependent chains, R, in our method is relatively small ($R < S$), and (ii) the response delays are already balanced by the adaptive trajectory sampling.

[1]The correction cannot be used for the LDA experiments because for SGRLD the noise term depends on current state θ_t^r.

4. Experiments

4.1. Simple Demonstration

We first illustrate our proposed method based on sampling from a multivariate normal posterior distribution obtained by assuming a normal prior $\mathcal{N}(\mu_x; \mu_0, \Sigma_0)$ on the d-dimension mean μ_x of a normal distribution $\mathcal{N}(x; \mu_x, \Sigma_x)$ from which we have observed N samples. Because this is a conjugate prior, the posterior distribution is also a normal distribution.

To examine the bias correction effect, we allocated a total of 20,000 data points to a cluster of 20 workers. Furthermore, we made the shard sizes $\{N_s\}$ highly imbalanced by setting $N_s = 500$ for 10 workers and setting $N_s = 1500$ for the remaining 10 workers. Then, to impose a higher level of imbalance, we also used the small shards 7 times more often than the large shards by setting the trajectory lengths for the small shards to 70 and those for the large shards to 10. We set the step size ϵ to 10^{-7} and the mini-batch size to $n = 300$.

In Fig. 2 (a) and (b), the black dotted circles represent the 2-d marginal covariance centered at the mean of the 20 local posteriors. Note that these are rescaled such that small circles represent the local posteriors based on small shards, whereas the large circles represent the local posteriors based on large shards. Also, the red circle represents the true posterior, and the dotted blue circle represents the empirical distribution based on our samples. As we can see, our algorithm corrects the bias. We have evaluated our method for various dimensions (up to $d = 100$) and found similar results.

The effect of trajectory lengths is also tested in Fig. 2 (c) and (d) using two different trajectory lengths, $\tau = 10,000$

and $\tau = 200$, for a cluster of 4 workers. Here, the shard size was set to 2,000 for each worker, the trajectory lengths were kept the same for all workers, and the step size, ϵ, was set to 2×10^{-6}. As described in section 3.2.1, we can see that D-SGLD samples from a mixture of the local posteriors with long trajectory lengths and becomes close to the standard SGLD posterior as the length decreases.

4.2. Distributed Latent Dirichlet Allocation

Next, we evaluate our method based on an important distributed inference problem, namely, large-scale LDA, by comparing the following algorithms:

(a) *AD-LDA*: In AD-LDA, to obtain a single sample, each worker s performs collapsed Gibbs iterations only on the full local shard (and is thus approximate), and then synchronizes its local topic assignments n^s_{kw} (for topic k and word w) at the master to obtain the global state n_{kw} based on the update $n_{kw} \leftarrow n_{kw} + \sum_{s=1}^{S}(n_{kw} - n^s_{kw})$. Then, the local state is updated by the new global state, $n^s_{kw} \leftarrow n_{kw}$. It is shown that in practice AD-LDA shows comparable perplexities to the standard collapsed Gibbs sampling.

(b) *Async-LDA* (Y-LDA): Unlike AD-LDA, Y-LDA (Ahmed et al., 2012; Smola & Narayanamurthy, 2010) performs asynchronous updates for the global state by the update, $n_{k,w} \leftarrow n_{k,w} + \sum_{s=1}^{S}(\tilde{n}^s_{k,w} - n^s_{k,w})$. Here, $\tilde{n}^s_{k,w}$ is a copy of the old local state at the time of previous synchronization. Because Y-LDA is a specific implementation optimized along with many other dimensions, we implemented an algorithm called Async-LDA which replaces the update of AD-LDA with the asynchronous update of Y-LDA. Async-LDA was used to compare with the load balancing ability of D-SGLD.

(c) *SGRLD*: Stochastic gradient Riemannian Langevin dynamics (SGRLD) (Patterson & Teh, 2013) is a specific SGLD sampler designed to sample from the probability simplex using Riemannian manifold. For LDA, SGRLD achieved fast mixing rate and resulted in the state-of-the-art performance. Note that SGRLD runs on a single machine without communication overhead. Specifically, SGRLD samples from a W-dimension topic probability simplex θ_k, and the mean score $\bar{g}(\theta_{kw}; X^n_t)$ in the update rule is obtained by

$$\bar{g}(\theta_{kw}; X^n_t) = \frac{1}{n} \sum_{d \in X^n_t} \mathbb{E}_{z_d | w_d, \theta, \alpha} \left[\frac{n_{dkw}}{\theta_{kw}} - \frac{n_{dk\cdot}}{\theta_{k\cdot}} \right] \quad (7)$$

where the expectation is computed by running a collapsed Gibbs sampler on the topic assignments z_d in each document d separately. Refer to Patterson & Teh (2013) for the full update equation.

Following Patterson & Teh (2013), we set the mini-batch size to 50 documents, and for each update of Eqn. (7) we ran 100 Gibbs iterations for each document in the mini-batch. The step-sizes were annealed by a schedule $\epsilon_t = a(1+t/b)^{-c}$. As we fixed $b = 1000$ and $c = 0.6$, the entire schedule was set by a which we choose by running parallel chains with different a's and then choosing the best.

(d) *D-SGLD*: Our algorithm, D-SGLD for LDA, is built upon SGRLD. To use SGRLD as our base sampler, we only need to multiply the bias correction factor $\frac{N_s}{N q_s}$ to Eqn. (7). We used cyclic rotation as the chain-to-worker scheduler and set the trajectory length $\tau = 10$ for all workers while we kept other parameters the same as for SGRLD by default.

In particular, to see the effect of the variance reduction (i.e., sample averaging), we implemented three different versions of D-SGLD, (i) *Complete Coupling* (D-CC), (ii) *Complete Independent* (D-CI), and (iii) *Hybrid* (D-Hybrid). D-CC couples *all* chains; whereas, D-CI runs independent chains without any interaction among them. D-Hybrid partitions the chains into groups and the averaging is performed only for the chains in the same group. When the variance reduction is used, it was performed at the end of each trajectory; we did not inject any additional noise for correction.

Additionally, we used the following settings for all algorithms. The predictive perplexities were computed on 1000 separate holdout set, with a 90/10 (training/test) split, and LDA's hyper-parameters were set to $\alpha = 0.01$ and $\beta = 0.0001$ following Patterson & Teh (2013). The number of topics K was set to 100. Parallelism within a worker is not considered, although D-SGLD can be easily parallelized within a worker.

We evaluate these methods based on the following datasets: (i) *Wikipedia* corpus, which contains 4.6M articles of approximately 811M tokens in total. We used the same vocabulary of 7702 words as used by Hoffman et al. (2010). (ii) *PubMed Abstract* corpus contains 8.2M articles of approximately 730M tokens in total. After removing stopwords and low occurrence (less than 300) words, we obtained a vocabulary of 39,987 words. For our Python implementation, each of the datasets has 47GB memory footprint.

Perplexity. We first compare the above algorithms in terms of the convergence in perplexity over wall-clock time on 20 homogeneous workers dedicated to the given task only. For D-Hybrid, we set the number of groups, G, to 5 and 3 for Wikipedia and Pubmed respectively. For Wikipedia, we set the group size to $R = 4$. For Pubmed, we set the sizes of the three groups to 7,7, and 6. To examine the effect of the variance reduction strategy, it was continued until the end of the experiment, as opposed to stopping at some point. The step size parameter a was set to 0.0001 for Wikipedia

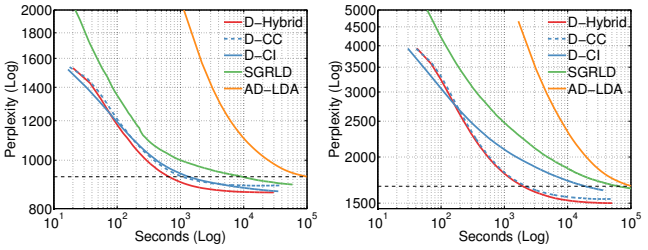


Figure 3. Perplexity. Left: Wikipedia, Right: Pubmed.

	D-SGLD	SGRLD	AD-LDA
Wikipedia	10 min.	2.6 hr.	27.7 hr.
Pubmed	33 min.	16.7 hr.	27.7 hr.

Table 1. Required time to reach the perplexity that AD-LDA obtains after running 10^5 seconds (27.7 hours).

and to 0.0005 for Pubmed.

In Fig. 3 (a) and (b), we first see that all the variants of D-SGLD significantly outperform both AD-LDA and SGRLD. Note that AD-LDA ran in an ideal setting where each worker has equal workloads (in terms of shard size) resulting in negligible block-by-the-slowest delays. As shown in Table 1, D-SGLD required substantially shorter times than AD-LDA and SGRLD to reach the same perplexity level that AD-LDA achieves after running 10^5 seconds (27.7 hours) indicated by the black horizontal dotted line. Throughout the experiments, Async-LDA always performed worse than AD-LDA given balanced workloads.

For the three different versions of D-SGLD, we see that D-CC and D-Hybrid (which use the sample averaging) converge faster than D-CI (which uses independent chains). However, when we couple too many chains as shown in D-CC, it could lead to some lose of accuracy (possibly, due to the bias by the coupling). Hence, in the following experiments, we only use hybrid D-SGLD; a proper group configuration is chosen by cross-validation. Fig. 4 shows other effects of the group configuration by increasing group size (R) and number of groups (G).

Dataset size. In D-SGLD the computation cost per sample $\mathcal{O}(n)$ is independent of N. AD-LDA, on the other hand, becomes slower as N increases. To see the effect of N, we examined the algorithms on random subsets of the full dataset with different sizes, 100K, 1000K, and full, using 20 homogeneous workers. For N=[100K, 1000K, full], the initial step sizes a were set to respectively a=[0.005, 0.0005, 0.0001] for Wikipedia and a=[0.01, 0.005, 0.0005] for Pubmed.

As shown in Fig. 5, for Wikipedia, D-SGLD showed similar convergence in perplexity (they increase slightly as the size of datasets decreases) while providing better results than AD-LDA in all settings. However, for Pubmed, which has a larger vocabulary and is expected to have a larger number of topics, D-SGLD was not better than AD-LDA for the small (100K) dataset while still had better performance for larger datasets. In fact, SGRLD seemed to work less efficiently (and so does D-SGLD) for rather small datasets as shown by Patterson & Teh (2013) based on the NIPS corpus. Nevertheless, we found that (results not shown here) D-SGLD outperforms a single SGRLD based on a 100K dataset.

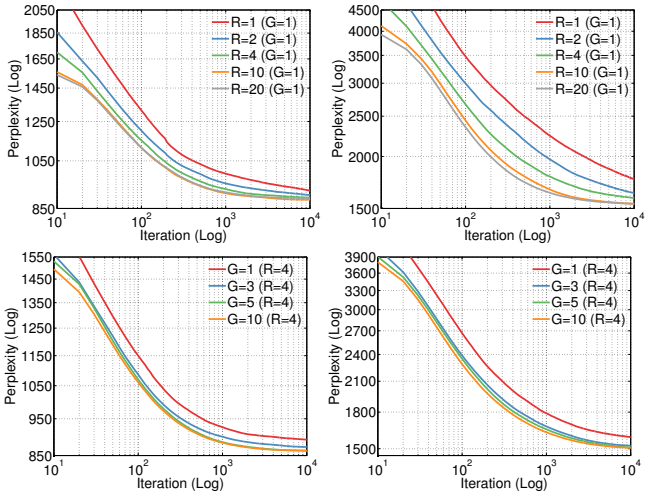


Figure 4. Group size and number of groups. Top: group size, Bottom: number of groups, Left: Wikipedia, Right: Pubmed.

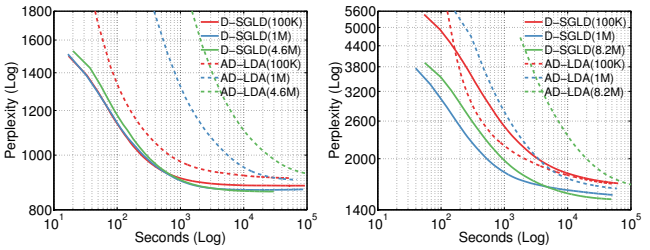


Figure 5. Dataset size. Left: Wikipedia, Right: Pubmed

Number of workers. We also varied the number of workers while fixing the dataset size to the full. In Fig. 6, we show the results for three cluster sizes, $S = [20, 40, 60]$. As expected, AD-LDA improves linearly by increasing the number of workers (i.e., by reducing local shard sizes). For D-SGLD, we fixed G to 5 and increased only the group size R to $4, 8, 12$. Although more workers imposed more communication overhead during sample averaging, D-SGLD showed its scalability by keeping the performance at a similar level (for Pubmed, it is improved). From this result, we calculated the number of workers required by AD-LDA to show a similar speed as D-SGLD with 20 workers. As shown in the Fig. 6, AD-LDA needs 2000 workers for Wikipedia and 800 workers for Pubmed to obtain a sim-

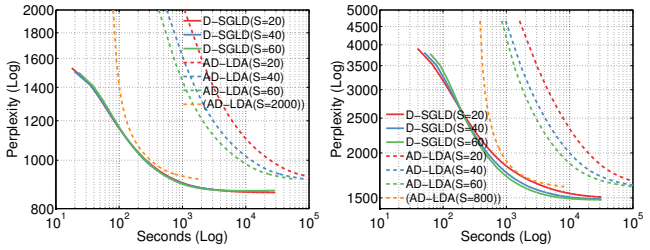


Figure 6. Number of workers. Left: Wikipedia, Right: Pubmed

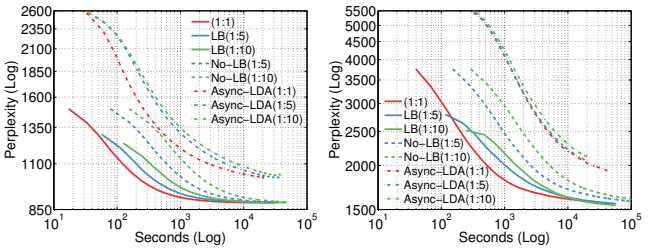


Figure 7. Load balancing. Left: Wikipedia, Right: Pubmed.

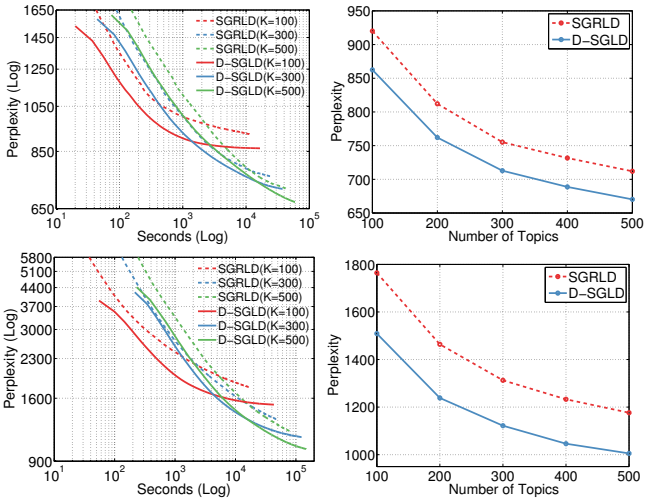


Figure 8. Number of topics. Top: Wikipedia, Bottom: Pubmed. Right: Perplexity after 10^4 updates (that is, the end points of each line in the left plots).

ilar speed as D-SGLD. (This simple calculation does not include the communication overhead.)

Load balancing. We also examined D-SGLD's ability to balance the workloads and thus mitigate the block-by-the-slowest problem on 20 workers. To do this, we added dummy delays to half of the workers to make them D times slower. We denote this setting by $(1{:}D)$ and used three settings: $D = [1, 5, 10]$. The actual response delays then became equal, for example, by setting the trajectory length to 10 for slow workers and to $D \times 10$ for fast ones. The initial step size a was set to 0.005 for all settings of Wikipedia and to 0.001 for all settings of Pubmed. Here, we used 100K Wikipedia and 1000K Pubmed corpus because the Async-LDAs (as well as AD-LDA) were too slow for the full datasets. As shown in Fig. 7, D-SGLD with load-balancing through adaptive trajectory sampling converges much faster than those without load-balancing; it also converges faster than Async-LDA.

Number of topics. We tested the effect of the number of topics K by examining K=[100,200,300,400,500] on 20 homogeneous workers. As shown in Fig.8, although the packet size increases for large K, D-SGLD consistently outperforms SGRLD for all K.

5. Conclusion

We have introduced a novel algorithm, "distributed stochastic gradient Langevin dynamics (D-SGLD)". Using D-SGLD, the advantages of the sequential mini-batch-based MCMC are extended to distributed computing environments. We showed that (i) by adding a proper correction term, our algorithm prevents the local-subset-bias while (ii) reducing communication overhead through trajectory sampling and adaptive load balancing. Furthermore, (iii) it improved convergence speed using a variance reduction strategy. Finally, in several experiments for LDA, we have shown at least an order of magnitude faster convergence speed of D-SGLD over the state of the art both in sequential mini-batch-based MCMC and distributed MCMC. We believe that D-SGLD is just one example of a much larger class of powerful MCMC algorithms that combine sampling updates based on mini-batches with distributed computation.

Acknowledgments

We thank A. Korattikara, S. Patterson, Y. W. Teh, J. Foulds and the reviewers for their valuable comments and suggestions. This work is supported by NSF grant IIS-1216045 and Amazon AWS in Education Grant award.

References

Ahmed, A., Aly, A., Gonzalez, J., Narayanamurthy, S., and Smola, A. Scalable inference in latent variable models. In *International conference on Web search and data mining*, 2012.

Ahn, S., Korattikara, A., and Welling, M. Bayesian posterior sampling via stochastic gradient fisher scoring. In *International Conference on Machine Learning*, 2012.

Ahn, S., Chen, Y., and Welling, M. Distributed and adaptive darting monte carlo through regenerations. *International Conference on Artificial Intelligence and Statistics*, 2013.

Andrieu, C. and Thoms, J. A tutorial on adaptive mcmc. *Statistics and Computing*, 18:343–373, 2008.

Bardenet, R., Doucet, A., and Holmes, C. Towards scaling up markov chain monte carlo: an adaptive subsampling approach. In *International Conference on Machine Learning*, 2014.

Ho, Q., Cipar, J., Cui, H., Kim, J. K., Lee, S., Gibbons, P. B., Gibson, G., Ganger, G. R., and Xing, E. P. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*, 2013.

Hoffman, M., Bach, F., and Blei, D. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pp. 856–864, 2010.

Korattikara, A., Chen, Y., and Welling, M. Austerity in mcmc land: Cutting the metropolis-hastings budget. In *International Conference on Machine Learning*, 2014.

Laskey, K. B. and Myers, J. W. Population markov chain monte carlo. *Machine Learning*, 50:175–196, 2003.

Newman, D., Smyth, P., Welling, M., and Asuncion, A. Distributed inference for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pp. 1081–1088, 2007.

Patterson, S and Teh, Y. W. Stochastic gradient riemannian langevin dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, 2013.

Smola, A. and Narayanamurthy, S. An architecture for parallel topic models. *VLDB*, 2010.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *International Conference on Machine Learning (ICML)*, 2011.

Wilkinson, D. J. Parallel bayesian computation. *Statistics Textbooks and Monographs*, 184:477, 2006.