Sequential Tests for Large-Scale Learning

Anoop Korattikara
akoratti@uci.edu
Department of Computer Science, University of California, Irvine, Irvine, CA 92697, U.S.A.

Yutian Chen
yutian.chen@eng.cam.ac.uk
Department of Engineering, University of Cambridge, Cambridge CB2 1PZ, U.K., and University of California, Irvine, Irvine, CA 92697, U.S.A.

Max Welling
m.welling@uva.nl
Informatics Institute, University of Amsterdam, 1098 XH Amsterdam, Netherlands

We argue that when faced with big data sets, learning and inference algorithms should compute updates using only subsets of data items. We introduce algorithms that use sequential hypothesis tests to adaptively select such a subset of data points. The statistical properties of this subsampling process can be used to control the efficiency and accuracy of learning or inference. In the context of learning by optimization, we test for the probability that the update direction is no more than 90 degrees in the wrong direction. In the context of posterior inference using Markov chain Monte Carlo, we test for the probability that our decision to accept or reject a sample is wrong. We experimentally evaluate our algorithms on a number of models and data sets.

1 Introduction

At a time when data are growing at an exponential rate, computational considerations play an increasingly central role in training predictive models. Where we used to have enough time to wait until our learning algorithm had rigorously converged, we may now need to build the best possible model in some fixed (possibly unknown) amount of time. As such, in the context of big data, we care about the amount of predictive accuracy gained per unit of computational time.

Stochastic gradient descent (SGD), in contrast to batch gradient descent (BGD), is the archetypal example of an algorithm that, when well tuned,
has the ability to gain predictive accuracy quickly per unit of computation. This is easily understood when imagining a data set of infinite size: while BGD cannot perform even a single update, SGD can iterate quickly to a good predictive model without seeing all the data.

SGD exploits the fact that learning problems form a special class of optimization problems, in that its loss function often involves an average over data points. In each iteration, SGD randomly selects a mini-batch of one or more data items and uses the mean of gradients from the data points in the mini-batch as a cheap approximation to the true gradient of the loss function. This makes SGD a very efficient learning algorithm. However, SGD ignores other useful information present in the mini-batch. In particular, the variance of gradients indicates the redundancy of information conveyed by different data points in the mini-batch.

The amount of redundancy depends on the state of the model. When a model is untrained, most data points recommend a very similar update for the parameters in order to improve the model. As an example, imagine learning the mean of a gaussian distribution. Far away from convergence, the mean is well outside the hull of the data points, and every data point will recommend an update in roughly the same direction. But close to convergence, when the current estimate of the mean is inside the data hull, data points recommend updates in very different directions, and one needs to average over a larger subset of data points in order to get a clear picture. Thus, partially or untrained models have higher data redundancy than converged models (which have zero redundancy). This notion of redundancy has been defined more formally as the learning-signal-to-noise-ratio (LSNR) in Welling (2014). LSNR measures the amount of learning signal in a mini-batch of \( n \) data points relative to the uncertainty under resampling the mini-batch. This quantity indeed has the intuitive property that the LSNR increases if we increase the size of the mini-batch (\( n \)) but decreases as the model gets closer to convergence.

In the first half of the article, we exploit these ideas by introducing a sequential hypothesis test that tests for the probability that the gradient based on the current mini-batch of data points is more than 90 degrees in the wrong direction. If this probability is too high, we fail the hypothesis that we are moving within 90 degrees of the correct update direction and request more data in our mini-batch to increase the accuracy of our gradient. If we keep the confidence level in this test constant, this naturally leads to a learning algorithm that automatically increases the mini-batch size and as such autotunes the hyperparameters of its update schedule.

In the second part of the article, we show that the same statistical properties of learning can be applied to improve the efficiency of Bayesian posterior inference when faced with big data sets. Traditional versions of algorithms such as Markov chain Monte Carlo (MCMC) or variational inference require processing the entire data set in each iteration and will not update when confronted with an infinite data set. This implies that even inference
algorithms should use only subsets of data to make updates. Some recent progress along these lines has been made in the context of variational inference (Hoffman, Blei, Wang, & Paisley, 2013) and MCMC methods (Welling & Teh, 2011). But in this article, we focus only on MCMC algorithms.

MCMC algorithms based on subsets of data may not sample from the correct distribution. One can decompose the total error (or risk) of some posterior average of interest into a bias and a variance contribution. The bias results from using a sampling procedure that samples from the wrong distribution, while the variance is caused by randomness of the sampling process. Traditional MCMC algorithms are asymptotically unbiased but pay a high computational price by using all the data for every update. In the context of very large data sets and acknowledging that in reality we have only a finite amount of time to generate a number of samples, this results in a high variance contribution to the risk. We advocate algorithms that balance bias and variance contributions more intelligently by allowing procedures that generate lots of samples in a short amount of computational time but from a slightly biased distribution.

We develop a fast MCMC algorithm for posterior inference by approximating the Metropolis-Hastings (MH) test with a sequential hypothesis test. We show that the MH test is essentially a decision problem that compares the mean difference in log likelihoods of the current and proposed parameter values to a threshold independent of the data. By estimating this mean using a subset of the data and also measuring its variance, we can conduct hypothesis tests to approximate this decision. The test fails if we do not have enough confidence to make a decision using the current mini-batch, in which case we add more data to the mini-batch to increase the accuracy of the comparison. The level of confidence necessary to reach a decision acts as a knob with which we can control the bias of the algorithm. This algorithm was developed in our prior work (Korattikara, Chen, & Welling, 2014), and a similar method based on concentration bounds was concurrently proposed in Bardenet, Doucet, and Holmes (2014).

We emphasize the unified nature of the methods we employ for optimization and sampling. In both cases, we exploit information in the distribution over certain quantities necessary for learning. In the case of optimization, this quantity is the gradient, while for posterior sampling, this quantity is the accept-reject probability of a proposal. In both cases, we exploit this information to reason about the correct size of the mini-batch necessary to perform a reliable update. In the case of optimization, we require just enough data to make sure that the update direction is within 90 degrees of the true update direction, while for posterior sampling, we require just enough data to make sure that the confidence in an accept-reject decision is high enough. To make these decisions, in both cases we rely on (frequentist) sequential hypothesis tests: we make an update only if we can reject the null hypothesis that there is not enough information in the mini-batch. If not, we add more data to the mini-batch until we can reject the null hypothesis.
with high confidence. In essence, we gauge whether there is enough signal in the learning update relative to the subsampling noise. While the philosophy behind both methods is clearly the same, the details of the convergence analysis cannot be; for optimization, we test for convergence to a point estimate, while for posterior sampling, we need to establish convergence to a probability measure.

The rest of the article is organized as follows. We introduce our sequential test for speeding up optimization algorithms in section 2. Then we use similar ideas to develop a fast MCMC sampler in section 3. We evaluate our methods on a variety of models and data sets in section 4 and conclude in section 5.

2 Sequential Tests for Optimization

Many learning problems can be cast as an optimization problem that involves minimizing a loss function with respect to a set of parameters $\theta \in \mathbb{R}^D$. The loss function is often defined as an expectation over a data set $D_N$ consisting of $N$ data points, and an iterative optimization algorithm is used to find the optimal parameters. In each iteration $t$ of the optimization algorithm, a new estimate $\theta_{t+1}$ of the optimal parameters is computed using our current estimate $\theta_t$ and the data set $D_N$.

Optimization algorithms that process the entire data set to compute each update are known as batch algorithms. When there are very large data sets, batch algorithms are clearly wasteful in terms of computational resources. In early iterations, when we are far from the solution, we need only a rough direction to move in parameter space, and this can be estimated without looking at every data point in the data set. We need more precise updates only when we get closer to the solution.

In contrast, algorithms such as stochastic gradient ascent/descent (both of which we denote as SGD henceforth) use only a single data point or a mini-batch of data points to compute each update. In the initial iterations, SGD is very efficient because we can compute noisy updates cheaply. However, the noise level remains high even when we get close to the solution. Therefore, it is common to use a step size that is annealed to zero at a certain rate to reduce the effect of noise on the optimization process. Under certain conditions, it can be shown that this converges to the correct solution (Robbins & Monro, 1951). A drawback is that the annealing schedule has to be tuned separately for every optimization problem and data set.

Instead of reducing the effect of noise by using a decreasing step size, we propose computing updates using a mini-batch that grows in size over time to reduce the level of noise itself. Thus, in the early iterations, we use a small mini-batch to compute rough updates and larger mini-batches to compute more precise updates as we move closer to the solution. The advantage of this approach is that we can increase the size of the mini-batch
Sequential Tests for Large-Scale Learning

automatically using a sequential statistical test that measures the precision of updates computed from the mini-batch.

The sequential test works as follows. First, we quantify the uncertainty in our noisy update by estimating its distribution under resampling the mini-batch from the complete data set. We can then use this to estimate the probability that an update computed from a mini-batch of the current size is more than 90 degrees away from the true update direction. If this probability is higher than a threshold, we can conclude that the mini-batch size does not provide the necessary precision required at the current stage of optimization. In this case, we reject the update and increase the size of the mini-batch until we have enough confidence to pass the test.

Similar statistical tests have previously been used to speed up optimization algorithms such as nonnegative matrix factorization (Korattikara, Boyles, Welling, Kim, & Park, 2011), $L_1$ regularized problems (Boyles, Korattikara, Ramanan, & Welling, 2011), logistic regression, and Least absolute deviation regression (Korattikara, 2011). These tests model the distribution of the update under resampling the mini-batch with replacement. In contrast, we use sampling without replacement, which allows us to obtain updates of the same precision using fewer data.

Our ideas can be used to speed up any iterative optimization algorithm if we can estimate the distribution of the updates. Fortunately, in many machine learning algorithms, the update involves an average over a large number of data points. Therefore, according to the central limit theorem, the distribution of an update computed from a large enough mini-batch is approximately normal and can be easily estimated. In this article, we illustrate our ideas on updates of the form $\theta_{t+1} \leftarrow u_N = (A_N^T A_N)^{-1} (A_N^T b_N)$ where $A_N = [a_1, \ldots a_N]^T$, $b_N = [b_1, \ldots b_N]^T$, and $\{a_i, b_i\}$ are functions of the the $i$th observation $\{x_i, y_i\}$ and our current parameter estimate $\theta_t$. Algorithms with updates of this form are known as iterative reweighted least squares (IRLS) and are used for solving a variety of problems such as parameter estimation in generalized linear models (McCullagh & Nelder, 1999), $L_p$-norm regression (Gentle, 2007), principal component analysis (Roweis, 1998), and nonnegative matrix factorization (Kim & Park, 2008).

2.1 Tests for IRLS-Type Algorithms. The update vector $u_N = (A_N^T A_N)^{-1} (A_N^T b_N)$ can be interpreted as the least squares estimator, computed from $N$ observations, of the parameters $u$ in a (fictional) linear regression model $b_j = a_i^T u + \zeta_j$, where $\zeta$ is an error term. Since computing $u_N$ requires $O(N)$ time and is impractical for very large data sets, let us consider approximating $u_N$ with an estimate $u_n$ computed from a mini-batch of $n$ data points:

$$u_n = (A_n^T A_n)^{-1} (A_n^T b_n) = \left(\frac{1}{n} \sum_{i=1}^n a_i a_i^T\right)^{-1} \left(\frac{1}{n} \sum_{i=1}^n a_i b_i\right). \quad (2.1)$$
If we consider \( \{a_i, b_i\} \) as independent samples from a data-generating process (or an infinite data set) and the following assumptions hold:

1. No multicollinearity: \( Q_{aa} = E[aa^T] \) is positive definite,
2. Exogeneity: \( E[\zeta | a] = 0 \),
3. Homoskedasticity: \( \text{Var}[\zeta | a] = \omega^2 \),

then \( u_n \) can be shown to be asymptotically (as \( n \to \infty \)) normal (Verbeek, 2000) under resampling the mini-batch from its generating process. We have

\[
    u_n \sim N \left( u, \frac{Q_{aa}^{-1} \omega^2}{n} \right). \tag{2.2}
\]

However, we are interested in the distribution of \( u_n \) under resampling the mini-batch without replacement from a data set of fixed size \( N \). This can be obtained by applying a finite population correction factor (Isserlis, 1918) to equation 2.2:

\[
    u_n \sim N \left( u, \Sigma \right) \quad \text{where} \quad \Sigma = \frac{Q_{aa}^{-1} \omega^2}{n} \left( 1 - \frac{n - 1}{N - 1} \right). \tag{2.3}
\]

Now consider the probability that an update computed from the mini-batch \( (u_n - \theta_i) \) is more than 90 degrees away from the true update \( (u_N - \theta_i) \). This is, by definition,

\[
    \delta = \int 1[\langle u_n - \theta_i, u_N - \theta_i \rangle \geq 0] p(u_n) du_n, \tag{2.4}
\]

where \( 1 \) is the indicator function and \( p(u_n) \) is the distribution of \( u_n \) defined in equation 2.3. If this probability is high, the mini-batch update \( u_n \) is not very reliable. To compute this probability (see Figure 1), we first shift the origin to \( \theta_i \) and compute the marginal distribution of \( p(u_n) \) along the direction of the true update \( u_N - \theta_i \). This distribution is \( N(\mu, \sigma^2) \), where

\[
    \mu = \|u_N - \theta_i\|, \tag{2.5a}
\]

\[
    \sigma = \sqrt{\bar{u}^T \Sigma \bar{u}} \quad \text{where} \quad \bar{u} = \frac{u_N - \theta_i}{\|u_N - \theta_i\|}. \tag{2.5b}
\]

From this we can easily compute

\[
    \delta = \Phi \left( \frac{0 - \mu}{\sigma} \right), \tag{2.6}
\]

where \( \Phi \) is the cdf of the standard normal distribution. If \( \delta \) is below a threshold \( \epsilon \), we accept the update and set \( \theta_{i+1} \leftarrow u_n \). If not, we decrease \( \delta \).
Figure 1: (a) Contours show the distribution of the update vector. The arrow is the true update direction. The shaded region represents updates, which are more than 90 degrees away from the true update direction. (b) The marginal distribution of the update along the true update direction is shown. This distribution is $\mathcal{N}(\mu, \sigma^2)$, where $\mu = \| u_N - \theta_t \|$ and $\sigma^2 = \bar{u}\Sigma \bar{u}$. The probability of falling in the shaded region can be calculated as $\Phi\left(\frac{\theta - \mu}{\sigma}\right)$.

by adding more data points and test again. This procedure will terminate because when the mini-batch grows to include all data points, $\sigma = 0$ and therefore $\delta = 0 \leq \epsilon$. Since evaluating $\Phi(.)$ for every hypothesis test is expensive, we compute $\epsilon^* = \Phi^{-1}(\epsilon)$ once, and thereafter we test $\mu > -\sigma \epsilon^*$ to accept the update.

We use the following estimators for the parameters of the distribution $p(u_n)$:

\[ \hat{u}_N = u_n = \left( \sum_{i=1}^{n} a_i a_i^T \right)^{-1} \left( \sum_{i=1}^{n} a_i b_i \right), \]  
\[ \hat{Q}_{aa} = \frac{A_n}{n-1} \text{ where } A_n = \sum_{i=1}^{n} a_i a_i^T, \]  
\[ \hat{\omega}^2 = \frac{e_n}{n-1} \text{ where } e_n = \sum_{i=1}^{n} (b_i - u_n^T a_i)^2, \]  
\[ \hat{\Sigma} = \frac{\hat{Q}_{aa}^{-1} \hat{\omega}^2}{n} \left( 1 - \frac{n-1}{N-1} \right) = \frac{A_n^{-1} e_n}{n} \left( 1 - \frac{n-1}{N-1} \right). \]  

When the size of the mini-batch is increased, we need an efficient way to update these estimators without recomputing them from scratch. The recursive least squares estimator, based on the Shermann-Morrison formula,
can be used to do this if the mini-batch grows one data point at a time. The updates are:

\[ u_n \leftarrow u_{n-1} + \left( b_n - a^T_n u_{n-1} \right) \frac{A_{n-1}^{-1} a_n}{1 + a^T_n A_{n-1}^{-1} a_n}, \]  
(2.8a)

\[ A_{n-1}^{-1} \leftarrow A_{n-1}^{-1} - \frac{A_{n-1}^{-1} a_n a^T_n A_{n-1}^{-1}}{1 + a^T_n A_{n-1}^{-1} a_n}, \]  
(2.8b)

\[ e_n \leftarrow e_{n-1} + \left( b_n - a^T_n u_n \right) \left( b_n - a^T_n u_{n-1} \right). \]  
(2.8c)

The complete test is given in algorithm 1. In our implementation, we conduct the first test after seeing \( N_0 \) data points and subsequent tests every time the mini-batch grows by \( N_{inc} \) data points. We could also consider testing at increasingly larger intervals.

Our algorithm can significantly outperform batch algorithms (see section 4.1) because we use only just enough data required to confidently move in the correct direction. While our algorithm and SGD are comparable in terms of performance (see section 4.1), an important advantage over SGD is that the main parameter of our algorithm, \( \epsilon \), is interpretable: \( \epsilon \) is an upper bound on the probability of obtaining a significantly different update direction if the update were to be computed from a different random mini-batch of the current size. Thus, \( \epsilon \) can be set independent of the particular optimization problem or data set at hand. In contrast, the parameters for tuning the annealing schedule in SGD have no intuitive interpretation and have to be tuned separately for each optimization problem or data set, using many expensive trial runs.

3 Sequential Tests for Bayesian Posterior Sampling

Similar tests can be used to perform approximate Bayesian posterior inference with very large data sets. Given a data set \( D_N \) consisting of \( N \) independent observations \( \{x_1, \ldots, x_N\} \), which we model using a distribution \( p(x_i; \theta) \) parameterized by \( \theta \in \mathbb{R}^D \), and a prior distribution \( \rho(\theta) \), we are interested in the posterior distribution \( S_0(\theta) = p(\theta | D_N) \propto \rho(\theta) \prod_{i=1}^N p(x_i; \theta) \), or, more specifically, in expectations computed with respect to this posterior distribution (e.g., average prediction on test data). Usually these expectations cannot be computed analytically, and one has to resort to sampling or variational methods to estimate them. Sampling methods estimate the expectation \( I = \langle f \rangle_{S_0} \) using an empirical average \( \hat{I} = \frac{1}{T} \sum_{t=1}^T f(\theta_t) \) where \( [\theta_1, \ldots, \theta_T] \) are \( T \) samples generated from \( S_0 \).

MCMC is a popular sampling method that can be used to generate samples from any distribution \( S_0 \) by forward-simulating a Markov chain designed to have stationary distribution \( S_0 \). A Markov chain with a given
**Algorithm 1:** Sequential Test for Optimization.

**Require:** \( \theta_t, D_N, \epsilon, N_0, N_{inc} \)

**Ensure:** \( \theta_{t+1} \)

1: \( \epsilon^* \leftarrow \Phi^{-1}(\epsilon) \)

2: Initialize \( A_0 \leftarrow 0, B_0 \leftarrow 0 \)

3: for \( n = 1 \rightarrow N \) do

4: Compute \( a_n, b_n \) from \( \theta_t, x_n, y_n \) (problem specific)

5: if \( n \leq N_0 \) then

6: \( A_n \leftarrow A_{n-1} + a_n a_n^T, B_n \leftarrow B_{n-1} + a_n b_n \)

7: end if

8: if \( n = N_0 \) then

9: Compute \( A_n^{-1} \) and set \( u_n \leftarrow A_n^{-1} B_n \)

10: \( e_n = \sum_{i=1}^{n} (u_n^T a_i - b_i)^2 \)

11: end if

12: if \( n > N_0 \) then

13: Update \( u_n, A_n^{-1} \) and \( e_n \) as in eqn. 2.8

14: end if

15: if \( n \geq N_0 \) and \((n - N_0) \mod N_{inc} = 0\) then

16: \( \hat{\mu} \leftarrow \| u_n - \theta_t \|, \hat{u} \leftarrow \frac{u_n - \theta_t}{\| u_n - \theta_t \|}, \hat{\Sigma} \leftarrow A_n^{-1} e_n \left( 1 + \frac{n-1}{N_{inc}} \right), \hat{\sigma} \leftarrow \sqrt{\hat{u}^T \hat{\Sigma} \hat{u}} \)

17: if \( \hat{\mu} > -\hat{\sigma} \epsilon^* \) then

18: break

19: end if

20: end if

21: end for

22: \( \theta_{t+1} \leftarrow u_n \)
stationary distribution can be constructed using the Metropolis-Hastings algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller, & Teller, 1953), which uses the following rule for transitioning from the current state \( \theta^t \) to the next state \( \theta^{t+1} \):

1. Draw a candidate state \( \theta' \) from a proposal distribution \( q(\theta' | \theta^t) \).
2. Compute the acceptance probability:
   \[
P_a = \min \left[ 1, \frac{S_0(\theta') q(\theta^t | \theta')}{S_0(\theta^t) q(\theta' | \theta^t)} \right].
\]
   (3.1)
3. Draw \( u \sim \text{Uniform}[0, 1] \).
4. If \( u < P_a \), set \( \theta^{t+1} ← \theta' \); otherwise, set \( \theta^{t+1} ← \theta^t \).

Following this rule ensures that the stationary distribution of the Markov chain is \( S_0 \). The bias of the MCMC estimator is \( \mathbb{E}[\hat{I} - I] = 0 \), and its variance is \( \mathbb{E}[\hat{I} - \mathbb{E}[\hat{I}]]^2 \approx \sigma^2 f_{\hat{S}_0} \tau / T \), where \( \sigma^2 f_{\hat{S}_0} \) is the variance of \( f \) with respect to \( S_0 \) and \( \tau \) is the integrated autocorrelation time. Here the expectations are taken with respect to multiple runs of the Markov chain. Unfortunately, for Bayesian posterior sampling, \( S_0 \) involves a product over \( N \) likelihood terms. Therefore, for very large data sets, computing the acceptance probability required by the Metropolis-Hastings test is very expensive and limits the number of samples that can be drawn in a reasonable amount of computational time. This results in \( \hat{I} \) having too high a variance to be useful.

However, if we can allow a small bias in the stationary distribution, we do not have to compute the posterior distribution exactly in each step and can therefore speed up MCMC. Thus, we will use samples from a distribution \( S_\epsilon \) instead of \( S_0 \) to compute expectations with respect to \( S_0 \). Here \( \epsilon \) is a knob that controls the amount of bias in \( S_\epsilon \). If we increase \( \epsilon \), \( \hat{I} \) will be more biased, but its variance will be low because we can draw a large number of samples in a given amount of time. As we decrease \( \epsilon \), the bias reduces. But since we have to compute the posterior distribution more accurately in each iteration, the computational time, and therefore the variance, increases.

To study this trade-off between bias and variance in approximate MCMC algorithms, we can look at the risk of \( \hat{I} \). The risk is \( R = \mathbb{E}[(\hat{I} - I)^2] \), that is, the expected squared error, where the expectation is taken over multiple runs of the Markov chain. It is easy to show that the risk can be decomposed as \( R = B^2 + V \), where \( B \) is the bias and \( V \) is the variance. If we ignore burn-in, we can further show that \( B = \langle f \rangle_{\hat{S}_\epsilon} - \langle f \rangle_{S_0} \) and \( V = \sigma^2 f_{\hat{S}_\epsilon} \tau / T \). The optimal setting of \( \epsilon \) that minimizes the risk depends on the amount of computational time available. If we have an infinite amount of computational time, we should set \( \epsilon \) to 0. Then there is no bias, and the variance can be brought down to 0 because we can collect an infinite number of samples. This is the traditional MCMC setting. However, given a finite amount of computational time, this setting is not optimal. It is better to tolerate a small amount of bias in the
stationary distribution if it allows us to reduce variance quickly by drawing a large number of samples.

There are at least two ways to design approximate MCMC samplers. One is to use a cheap proposal distribution $q$ that has an acceptance rate close to 1 and then accept every proposed state without conducting the expensive MH test. This method was used recently to develop efficient sampling algorithms such as stochastic gradient Langevin dynamics (Welling & Teh, 2011) and its successors (Ahn, Korattikara, & Welling, 2012; Patterson & Teh, 2013; Chen, Fox, & Guestrin, 2014; Ding et al., 2014). Another idea is to approximate the MH test directly using fewer data. In section 3.1, we describe a way of doing this using a sequential hypothesis test we developed in Korattikara et al. (2014). Moreover, these two approaches can be easily combined so that we have an efficient proposal distribution as well as an efficient accept-or-reject test (see section 4.2.4).

3.1 Approximate Sampling Using Sequential Metropolis-Hastings.
To approximate the Metropolis-Hastings test, we first frame it as a statistical decision problem. First, note that steps 2 and 3 of the MH test can be interchanged. Since $u$ is less than 1, instead of comparing $u$ to $P_{a_w}$, we can compare it directly to the ratio $S_{0}(\theta')q(\theta'|\theta_t)/S_{0}(\theta_t)q(\theta'|\theta')$. In the case of Bayesian posterior sampling, we compare $u \aleq \rho(\theta_t)\prod_{i=1}^{N}p(x_i; \theta')q(\theta'|\theta')/\rho(\theta_t)\prod_{i=1}^{N}p(x_i; \theta_t)q(\theta'|\theta_t)$. (3.2)

Taking a log on both sides and rearranging, we see that the above comparison is equivalent to comparing $\mu \aleq \mu_0$, where

$$\mu = \frac{1}{N} \sum_{i=1}^{N} l_i \text{ where } l_i = \log p(x_i; \theta') - \log p(x_i; \theta_t)$$

and

$$\mu_0 = \frac{1}{N} \log \left[ u \frac{\rho(\theta_t)q(\theta'|\theta_t)}{\rho(\theta')q(\theta_t|\theta_t)} \right].$$

(3.3)

If $\mu > \mu_0$, we accept the proposal and set $\theta_{t+1} \leftarrow \theta'$. If $\mu \leq \mu_0$, we reject the proposal and set $\theta_{t+1} \leftarrow \theta_t$. This reformulation of the MH test makes it very easy to frame it as a statistical hypothesis test: Given $\mu_0$ and a random sample $\{l_1 \ldots l_N\}$ drawn without replacement from the population $\{l_1 \ldots l_N\}$, can we confidently decide whether the population mean $\mu$ is greater than or less than the threshold $\mu_0$? The answer to this depends on the precision in the random sample. If the difference between the sample mean $\bar{l}$ and $\mu_0$ is significantly greater than the standard deviation $s$ of $\bar{l}$, we can accept or reject the proposal confidently. If not, we should draw more data to
increase the precision of $\bar{I}$ (reduce $s$) until we have enough evidence to make a decision.

More formally, we test the null hypothesis $H_0: \mu = \mu_0$ versus the alternative $H_1: \mu \neq \mu_0$. To do this, we proceed as follows. We compute the sample mean $\bar{l}$ and the sample standard deviation $s_l$. Then the standard deviation of $\bar{l}$ can be calculated as

$$s = \frac{s_l}{\sqrt{n}} \sqrt{1 - \frac{n - 1}{N - 1}}, \quad (3.4)$$

where $\sqrt{1 - \frac{n - 1}{N - 1}}$, the finite population correction term, is applied because we are drawing the subsample without replacement from a finite-sized population. Then, we compute the test statistic:

$$t = \frac{\bar{l} - \mu_0}{s}. \quad (3.5)$$

If $n$ is large enough for the central limit theorem (CLT) to hold, the test statistic $t$ follows a standard Student-t distribution with $n - 1$ degrees of freedom if the null hypothesis is true. Then we compute the $p$-value $\delta = 1 - \phi_{n-1}(|t|)$, where $\phi_{n-1}(.)$ is the cdf of the standard Student-t distribution with $n - 1$ degrees of freedom. If $\delta < \epsilon$, we can reject the null hypothesis; we have enough precision in the sample to make a decision. In this case, if $\bar{l} > \mu_0$, we accept the proposal; otherwise we reject it. If we fail to reject the null hypothesis, we draw more data to reduce the uncertainty, $s$, in the sample mean $\bar{l}$. We keep drawing more data until we have enough precision to make a decision. Note that this procedure will terminate because when we have used all the available data (i.e., $n = N$), the standard deviation $s$ is 0, the sample mean $\bar{l} = \mu$, and $\delta = 0 < \epsilon$. In this case we will make the same decision as the original MH test would make. Pseudocode for our test is shown in algorithm 2 (we have assumed that the total number of data points is divisible by the size of the mini-batch for simplicity).

The advantage of our method is that we can often make confident decisions with $n \ll N$ data points and save on computation. Although we introduce a small bias in the stationary distribution, we can use the computational time we save to draw more samples and reduce variance. The bias-variance trade-off can be controlled by adjusting the knob $\epsilon$. When $\epsilon$ is high, we make decisions without sufficient evidence and introduce a high bias. As $\epsilon \to 0$, we make more accurate decisions but are forced to examine more data, which results in high variance.

Theoretical analysis of our method can be found in Korattikara et al. (2014), Chen, Mansinghka, and Ghahramani (2015), Alquier, Friel, Everitt, and Boland (2014), and Pillai and Smith (2014). In particular, Korattikara
et al. (2014) and Chen et al. (2015) showed that the error of the approximate Markov chain transition kernel is controlled by $\epsilon$ and always diminishes as $\epsilon \to 0$ with or without the condition, respectively, that the central limit theorem holds. Pillai and Smith (2014) further proved that the approximate Markov chain is uniformly ergodic for any sufficiently small $\epsilon$ under regular

---

**Algorithm 2**: Approximate MH Test.

**Require**: $\theta_i, \theta', \epsilon, m, X_N$

**Ensure**: accept

1. Initialize estimated means $\bar{\theta} \leftarrow 0$ and $\overline{\theta^2} \leftarrow 0$

2. Initialize $n \leftarrow 0$

3. Draw $u \sim \text{Uniform}[0,1]$

4. Compute $\mu_0 \leftarrow \frac{1}{N} \log \left[ \frac{\rho(\theta_i) \rho(\theta_i')}{\rho(\theta_i') \rho(\theta_i)} \right]$

5. done $\leftarrow$ false

6. **while not** done **do**

7. Draw mini-batch $X_m$ of size $m$ without replacement from $X_N$

8. Set $X_N \leftarrow X_N \setminus X_m$

9. Update $\bar{\theta}$ and $\overline{\theta^2}$ using $X_m$.

10. $n \leftarrow n + m$

11. Update estimated standard deviation $s \leftarrow \sqrt{\frac{\overline{\theta^2} - \bar{\theta}^2}{n-1} \left( 1 - \frac{n-1}{N-1} \right)}$

12. Update Student-t statistic $t \leftarrow \frac{\bar{\theta} - \mu_0}{s}$

13. Update p-value $\delta \leftarrow 1 - \phi_{n-1}(|t|)$

14. **if** $\delta < \epsilon$ **then**

15. accept $\leftarrow$ true **if** $\bar{\theta} > \mu_0$ and false **otherwise**

16. done $\leftarrow$ true

17. **end if**

18. **end while**
conditions for the exact Markov chain and the stationary distribution of the approximate chain $S_\epsilon \rightarrow S_0$ as $\epsilon \rightarrow 0$.

Bardenet et al. (2014) concurrently developed a similar approach that uses concentration bounds to decide if the mini-batch has the required precision in contrast to the CLT-based confidence intervals that we use. This makes their approach more robust in cases where the CLT assumptions are violated. But when the CLT assumptions do hold, they show that their approach uses more data to reach an accept-or-reject decision than our test. A related method was also proposed in Singh, Wick, and McCallum (2012), where the factors of a graphical model are subsampled to compute fixed-width confidence intervals for the log likelihood. These ideas can also be used to speed up slice sampling algorithm as shown in DuBois, Korattikara, Welling, and Smyth (2014).

4 Experiments

4.1 Optimization. The threshold for passing the hypothesis test, $\epsilon$ was set to 0.01 for all algorithms and data sets. The first hypothesis test is conducted after $N_0$ data points are in the mini-batch and subsequent tests are performed every time the mini-batch grows by $N_{inc}$ data points. Both $N_0$ and $N_{inc}$ were set to 1/1000th the size of the data set. We compare our hypothesis testing method to both the batch algorithm, where the whole data set is used in every iteration, and stochastic gradient ascent/descent (both of which we will denote by SGD), where only one data point is used per iteration. For SGD, the step size was annealed as $C_1(C_2 + t)^{-C_3}$. We tried different sets of values for $C_1, C_2, C_3$ and plot only results from the set that gave the best performance on test data.

4.1.1 Least Absolute Deviation Regression. We first apply our hypothesis testing method to optimizing the parameters in a least absolute deviation regression model. Given predictors $x_i$ and targets $y_i$, the goal is to minimize the absolute error in our predictions:

$$\min_{\theta} \sum_{i=1}^{N} |y_i - \theta^T x_i|.$$  (4.1)

This can be solved by an iterative reweighted least Squares (IRLS) type algorithm where the weight of the $i$th data point is $w_i = \frac{1}{\sqrt{|y_i - \theta^T x_i|}}$. Thus, we have the following update rule: $\theta_{t+1} \leftarrow (A^T A)^{-1}(A^T b)$, where $a_i = w_i x_i$ and $b_i = w_i y_i$. In Figure 2 we compare our hypothesis-testing algorithm to the batch algorithm and SGD on the Million Song data set (Bertin-Mahieux, Ellis, Whitman, & Lamere, 2011). The goal is to predict the release year of a song from 90 audio features and we use the version of the data set available.
Sequential Tests for Large-Scale Learning

Figure 2: Test error as a function of time for LAD regression on the Million Song data set.

from the UCI machine learning repository (Bache & Lichman, 2013). There are 463,715 examples in the training set and 51,630 in the test set. For SGD, we obtained the best results with $C_1 = 0.01, C_2 = 1, \text{ and } C_3 = 0.5$.

4.1.2 Logistic Regression. Next, we apply our method to learning the parameters in a logistic regression model using an IRLS algorithm. Using the definitions $w_i = \sqrt{r_i(1 - r_i)}$ and $r_i = (1 + \exp(-\theta_i^T x_i))^{-1}$, the update rule is $\theta_{i+1} \leftarrow (A^T A)^{-1} (A^T b)$ with $a_i = w_i x_i$ and $b_i = w_i x_i^T \theta_i + \frac{y_i - 1}{w_i}$. In Figure 3, we compare our hypothesis-testing algorithm to the batch algorithm and SGD on the MNIST data set. We train a classifier for detecting the digit 3 versus all others. For training, we used a version of the Infinite data set with 8.1 million images available from Loosli, Canu, and Bottou (2007). For testing, we used the standard MNIST test set with 10,000 data points. Each image has 784 pixels. We removed pixels that were constant across all images in the training set and standardize the data by subtracting the mean and dividing by the standard deviation. Finally, we used PCA to reduce the dimensionality to 50. For SGD, we obtained the best results with $C_1 = 1, C_2 = 10, \text{ and } C_3 = 0.7$.

4.1.3 Analysis. We see that on both problems, our algorithm significantly outperforms the batch algorithm by using only the number of data required
at each stage of the optimization. Although SGD slightly outperforms our method, our algorithm has an automatic annealing schedule, which enables us to use the same parameters across various algorithms and data sets, whereas we need to carefully tune the parameters of the learning schedule in SGD. Similar results were reported in Boyles et al. (2011).

### 4.2 Sampling

#### 4.2.1 Random Walk–Logistic Regression

We first test our method using a random walk proposal $q(\theta' | \theta_t) = N(\theta_t, \sigma_{RW}^2)$. Although the random walk proposal is not efficient, it is very useful for illustrating our algorithm because it does not contain any information about the target distribution, unlike Langevin or Hamiltonian methods. Therefore, the responsibility of converging to the correct distribution lies solely with the MH test. Also since $q$ is symmetric, it does not appear in the MH test, and we can use $\mu_0 = \frac{1}{T} \log \left[ u \rho(\theta_t) / \rho(\theta') \right]$.

The target distribution in this experiment was the posterior for a logistic regression model trained on the MNIST data set for classifying the digits 7 versus 9. The data set consisted of 12,214 data points, and we reduced the dimensionality from 784 to 50 using PCA. We chose a zero mean spherical gaussian prior with precision $= 10$ and set $\sigma_{RW} = 0.01$ to give an acceptance rate $\approx 0.5$.  

![Figure 3: Test log likelihood as a function of time for logistic regression on the MNIST data set.](image-url)
In Figure 4a, we show the percentage of data used (solid blue line) and the percentage of wrong decisions (dashed red line) as a function of $\epsilon$. The number of data required drops off very fast as $\epsilon$ is increased even though the error does not increase much. Note that the setting $\epsilon = 0$ corresponds to the exact MH algorithm.

In Figure 4b, we show how the logarithm of the risk in estimating the mean prediction on test data decreases as a function of wall clock time. To calculate the risk, we first estimate the true mean prediction using a long run of hybrid Monte Carlo. Then we compute multiple estimates of the mean prediction using our approximate algorithm and obtain the risk as the mean squared error in these estimates. We plot the average risk of 2037 data points in the test set. Since the risk $R = B^2 + V = B^2 + \frac{\sigma^2}{T}$, we expect it to decrease as a function of time until the bias dominates the variance. The figure shows that even after collecting a lot of samples, the risk is still dominated by the variance, and the minimum risk is obtained with $\epsilon > 0$.

In Figure 5, we show marginals of $\theta_1$ versus $\theta_2$ for $\epsilon = 0$ and $\epsilon = 0.01$ at different values of wall clock time. The red curves are marginals for three different runs of the random walk algorithm, whereas the blue curve shows the true marginal obtained from a long run of hybrid Monte Carlo. At $T = 50$ sec, the exact MH algorithm $\epsilon = 0$ has still not completed burn-in. Our algorithm with $\epsilon = 0.01$ is able to accelerate burn-in because it can collect more samples in a given amount of time. Theoretically, as more computational time becomes available, the exact MH algorithm will catch up with (and eventually outperform) our algorithm, because the exact MH
algorithm is unbiased, unlike ours. But the bias is hardly noticeable in this example, and the error is dominated by the variance even after collecting around 100,000 samples.

4.2.2 Random Walk on Stiefel Manifold–Independent Component Analysis. Next, we use our algorithm to sample from the posterior distribution of the unmixing matrix in independent component analysis (ICA) (Hyvärinen & Oja, 2000). When prewhitened data are used, the unmixing matrix $W \in \mathbb{R}^{D \times D}$ is constrained to lie on the Stiefel manifold of orthonormal matrices. We choose a prior that is uniform over the manifold. We model the data as $p(x|W) = |\det(W)| \prod_{j=1}^{D} [4 \cosh^2(\frac{1}{2} w_j^T x)]^{-1}$ where $w_j$ are the rows of $W$. Since the prior is zero outside the manifold, the same is true for the posterior. Therefore, we use a random walk on the Stiefel manifold as a proposal distribution (Ouyang, 2008) and tuned the step size to give an acceptance rate of about 0.5. Since this is a symmetric proposal distribution, it does not appear in the MH test, and we can use $\mu_0 = \frac{1}{N} \log[u]$.

To perform a large-scale experiment, we created a synthetic data set by mixing 1.95 million samples of four sources: a classical music recording, (b) street/traffic noise, and two independent gaussian sources. To measure the correctness of the sampler, we measure the risk in estimating $I = E_{p(W|X)} [d_A(W, W_0)]$ where the test function $d_A$ is the Amari distance (Amari, Cichocki, & Yang, 1996) and $W_0$ is the true unmixing matrix. We computed the ground truth using a long run ($T = 100,000$ samples) of
then we ran each algorithm 10 times, each time for about 6400 secs. We calculated the risk by averaging the squared error in the estimate from each Markov chain over the 10 chains. This is shown in Figure 6. Note that even after 6400 sec, the variance dominates the bias, as evidenced by the still decreasing risk, except for the most biased algorithm with $\epsilon = 0.2$. Also, the lowest risk at 6400 sec is obtained with $\epsilon = 0.1$ and not the exact MH algorithm ($\epsilon = 0$). But we expect the exact algorithm to outperform all the approximate algorithms if we were to run for an infinite time.

4.2.3 Reversible Jump—Variable selection in Logistic Regression. Now, we apply our MH test to variable selection in a logistic regression model using the reversible-jump MCMC algorithm of Green (1995). We use a model that is very similar to the Bayesian Lasso model for linear regression described in Chen, Jane Wang, and McKeown (2011). Specifically, given $D$ input features (covariates), our parameter $\theta = \{\beta, \gamma\}$, where $\beta$ is a $D$-dimensional vector of regression coefficients and $\gamma$ is a $D$-dimensional binary vector that indicates whether a particular feature is included in the model. The prior we choose for $\beta$ is $p(\beta_j | \gamma, \psi) = \frac{1}{2\psi} \exp\{-\frac{|\beta_j|}{\psi}\}$ if $\gamma_j = 1$. If $\gamma_j = 0$, $\beta_j$ does not appear in the model. Here $\psi$ is a shrinkage parameter that pushes $\beta_j$ toward 0, and we choose a prior $p(\psi) \propto 1/\psi$. We also place a right-truncated Poisson prior on $\gamma$ as in Chen et al. (2011) to control the size of the model, $k = \sum_{j=1}^D \gamma_j$:

$$p(\gamma | \lambda) \propto \frac{\lambda^k}{(\binom{D}{k}k!}.$$  (4.2)
The extra $\binom{D}{k}$ factor appears because there are $\binom{D}{k}$ different models of size $k$. The likelihood of the data is

$$p(y_N|x_N, \beta, \gamma) = \prod_{i=1}^{N} \left[ \text{sigm}(\beta^T x_i)^y_i (1 - \text{sigm}(\beta^T x_i))^{1-y_i} \right]. \quad (4.3)$$

In the above, $\text{sigm}(z) = \left(1 + \exp(-z)\right)^{-1}$ is the sigmoid function. We will denote this likelihood by $l_N(\beta, \gamma)$. Then the posterior distribution after analytically integrating out $\psi$ is

$$p(\beta, \gamma|x_N, y_N, \lambda) \propto l_N(\beta, \gamma) \| \beta \|_1^{-k} \lambda^k B(k, D - k + 1), \quad (4.4)$$

where $B(a, b) = \int_0^1 t^{a-1} (1-t)^{b-1} dt$ is the beta function. We do not analytically integrate out $\lambda$, but instead use it as a knob to control the size of the model. We use the same proposal distribution as in Chen et al. (2011), which is a mixture of three types of move picked randomly in each iteration: an update move, a birth move, and a death move. The update move is the usual MCMC move, which involves changing the parameter vector $\beta$ without changing the model $\gamma$. Specifically, we randomly pick an active component $j : \gamma_j = 1$ and set $\beta_j = \beta_j + \eta$ where $\eta \sim \mathcal{N}(0, \sigma_{\text{update}})$. The birth move involves (for $k < D$) randomly picking an inactive component $j : \gamma_j = 0$ and setting $\gamma_j = 1$. We also propose a new value for $\beta_j \sim \mathcal{N}(0, \sigma_{\text{birth}})$. The birth move is paired with a corresponding death move (for $k > 1$), which involves randomly picking an active component $j : \gamma_j = 1$ and setting $\gamma_j = 0$. The corresponding $\beta_j$ is discarded. The probabilities of picking these moves $p(\gamma \rightarrow \gamma')$ is the same as in Chen et al. (2011). The value of $\mu_0$ used in the MH test for different moves follows:

1. **Update move:**

   $$\mu_0 = \frac{1}{N} \log \left[ \frac{\| \beta \|_1^{-k}}{\| \beta' \|_1^{-k}} \right]. \quad (4.5)$$

2. **Birth move:**

   $$\mu_0 = \frac{1}{N} \log \left[ \frac{\| \beta \|_1^{-k} p(\gamma \rightarrow \gamma') \mathcal{N}(\beta_j|0, \sigma_{\text{birth}})(D - k)}{\| \beta' \|_1^{-(k+1)} p(\gamma' \rightarrow \gamma) \lambda k} \right]. \quad (4.6)$$

2. **Death move:**

   $$\mu_0 = \frac{1}{N} \log \left[ \frac{\| \beta \|_1^{-k} p(\gamma \rightarrow \gamma') \lambda (k - 1)}{\| \beta' \|_1^{-(k-1)} p(\gamma' \rightarrow \gamma) \mathcal{N}(\beta_j|0, \sigma_{\text{birth}})(D - k + 1)} \right]. \quad (4.7)$$

For more details, see Chen et al. (2011).
We applied this to the MiniBooNE data set from the UCI machine learning repository. Here the task is to classify electron neutrinos (signal) from muon neutrinos (background). There are 130,065 data points with 50 features to which we add a constant feature of 1s. Around 28% of the examples are in the positive class. We randomly split the data into a training (80%) and testing (20%) set. First, to compute ground truth, we collected $T = 400,000$ samples using the exact reversible-jump algorithm. Then we ran different algorithms with $\epsilon = 0$, $\epsilon = 0.01$, $\epsilon = 0.05$, and $\epsilon = 0.1$ for around 3500 seconds. We plot the risk in mean prediction on test data (estimated from 10 Markov chains) in Figure 7. Again we see that the lowest risk is obtained with $\epsilon > 0$. We also plot the marginal posterior probability of including a feature in the model, that is, $p(\gamma_j = 1|X_N, y_N, \lambda)$ in Figure 8.

The acceptance rates for the birth and death moves start off at about 20% but die down to about 2% once a good model is found. The acceptance rate for update moves is kept at about 50%. The model also suffers from local minima. For the plot in Figure 7, we started with only one variable and ended up learning models with around 12 features, giving a classification error of approximately 15%. But if we initialize the sampler with all features included and set $\beta$ to the MAP value, we learn models with around 45 features but with a lower classification error of about 10%. Both the exact reversible-jump algorithm and our approximate version suffer from this problem. We should bear this in mind when interpreting ground truth. We can, however, see that when the sample is initialized with the same values, we obtain similar results with the approximate algorithm and the exact algorithm.

4.2.4 Stochastic Gradient Langevin Dynamics. Finally, we apply our method to stochastic gradient Langevin dynamics (Welling & Teh, 2011). In
Figure 8: Marginal probability of features to be included in the model.

Each iteration, we randomly draw a mini-batch $\mathcal{X}_n$ of size $n$ and propose $\theta' \sim q(.|\theta, \mathcal{X}_n)$ where

$$q(.|\theta, \mathcal{X}_n) = \mathcal{N}\left(\theta + \frac{\alpha}{2} \nabla_{\theta} \left\{ \frac{N}{n} \sum_{x \in \mathcal{X}_n} \log p(x|\theta) + \log \rho(\theta) \right\}, \alpha \right). \quad (4.8)$$

Every proposed state $\theta'$ is accepted without conducting any MH test. Since the acceptance rate approaches 1 as $\alpha$ goes to 0, we can keep the bias under control by keeping $\alpha$ small. However, we have to use a reasonably large $\alpha$ to keep the mixing rate high. This can be problematic for some distributions, because SGLD relies solely on gradients of the log density and it can be easily thrown off track by large gradients in low-density regions unless $\alpha \approx 0$.

As an example, consider an L1-regularized linear regression model. Given a data set $\{x_i, y_i\}_{i=1}^N$ where $x_i$ are predictors and $y_i$ are targets, we use a gaussian error model $p(y_i|x_i, \theta) \propto \exp\left\{ -\frac{\lambda}{2} (y_i - \theta^T x_i)^2 \right\}$ and choose a
Figure 9: Pitfalls of using uncorrected SGLD with certain distributions.

Laplacian prior for the parameters $p(\theta) \propto \exp(-\lambda_0 \|\theta\|_1)$. For pedagogical reasons, we restrict ourselves to a toy version of the problem where $\theta$ and $x$ are one-dimensional. We use a synthetic data set with $N = 10,000$ data points generated as $y_i = 0.5x_i + \xi$ where $\xi \sim \mathcal{N}(0, 1/3)$. We choose $\lambda = 3$ and $\lambda_0 = 4950$, so that the prior is not washed out by the likelihood. The posterior density and the gradient of the log posterior are shown in Figures 9a and 9b, respectively.

The empirical histogram of 100,000 samples obtained by running SGLD with $\alpha = 5 \times 10^{-6}$ is plotted using red bars in Figure 9c. The effect of omitting the MH test is quite severe in this case. When the sampler reaches the mode of the distribution, the Langevin noise occasionally throws it into the valley to the left, where the gradient is very high. The high gradient propels the sampler far off to the right, after which it takes a long time to find its way back to the mode. However, if we had used an MH accept-or-reject test, most of these troublesome jumps into the valley would be rejected because the density in the valley is much lower than that at the mode.

To apply an MH test, note that the SGLD proposal $q(\theta'|\theta)$ can be considered a mixture of component kernels $q(\theta'|\theta, \mathcal{X}_n)$ corresponding to different mini-batches. The mixture kernel will satisfy a detailed balance with respect to the posterior distribution if each of the component kernels $q(\theta'|\theta, \mathcal{X}_n)$ satisfies detailed balance. Thus, we can use an MH test with

$$
\mu_0 = \frac{1}{N} \log \left[ \frac{\rho(\theta_t)q(\theta'|\theta_t, \mathcal{X}_n)}{\rho(\theta')q(\theta'|\theta', \mathcal{X}_n)} \right].
$$

(4.9)
The result of running SGLD (keeping $\alpha = 5 \times 10^{-6}$ as before) with the exact MH correction is shown in Figure 9d. As expected, we are now able to sample correctly as the MH test rejects most proposals from the mode to the valley. Results of running SGLD with our approximate MH test are shown in Figures 9e ($\epsilon = 0.1$) and 9f ($\epsilon = 0.5$), respectively. The stationary distribution obtained with $\epsilon = 0.1$ is almost indistinguishable from that obtained by the exact MH test, although on average, the approximate test uses only 14.2% of the data per test. The bias with $\epsilon = 0.5$ is also negligible, even though it uses only 5% of the data per test, which is no more than the size of the mini-batch we used in the SGLD proposal distribution. Note that when $\epsilon = 0.5$, a decision is always made in the first step, without querying more data sequentially.

5 Conclusion

In the context of big data, learning and inference algorithms should acknowledge the fact that there is only a finite amount of time to perform its computations. Many traditional algorithms that use the entire data set in each iteration are becoming increasingly obsolete because they produce useful results only after an often unaffordable length of computational time. We argued that learning and inference algorithms should rather strive to gain maximal predictive power per unit of computation.

In this article, we developed such algorithms for both learning and inference using sequential hypothesis tests that use only just enough data required at each stage of computation. In the context of learning by optimization, we test for the probability that the update direction is no more than 90 degrees in the wrong direction. In the context of posterior inference using MCMC, we test for the probability that our decision to accept or reject a sample is wrong. Our hypothesis tests crucially depend on the central limit theorem to hold, which can usually be ensured by using a large enough mini-batch. In a few cases, for instance with very sparse data sets or data sets with very extreme outliers, the central limit assumptions are violated, and our algorithm can behave erratically. However, these assumptions can be easily checked before running our algorithm to avoid such pathological situations. Despite this, there are many interesting problems to which we can apply these algorithms.

Acknowledgments

This material is based on work supported by the National Science Foundation under grant no. 1216045. We thank Alex Ihler for advice on sequential hypothesis tests.
References


Received February 3, 2015; accepted September 10, 2015.