Sequential Tests for Large Scale Learning

Korattikara, A.; Chen, Y.; Welling, M.

DOI
10.1162/NECO_a_00226
Publication date
2016
Document Version
Submitted manuscript
Published in
Neural Computation

Sequential Tests for Large Scale Learning

Anoop Korattikara[1] **Yutian Chen** **Max Welling**[2]

University of California, Irvine University of Cambridge University of Amsterdam

Keywords: Big Data, Metropolis-Hastings, MCMC, Optimization, IRLS

Abstract

We argue that when faced with big datasets, 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 MCMC, we test for the probability

[1] Work done at UC Irvine, but currently affiliated with Google.

[2] Also affiliated with the University of California, Irvine.

that our decision to accept or reject a sample is wrong. We experimentally evaluate our algorithms on a number of models and datasets.

1 Introduction

In a time when data is 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 dataset 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 will 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 discussed above 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 paper, we will 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

auto-tunes the hyper-parameters of it's update schedule.

In the second part of the paper, we show that the same statistical properties of learning can be applied to improve the efficiency of Bayesian posterior inference when faced with big datasets. Traditional versions of algorithms such as Markov Chain Monte Carlo (MCMC) or variational inference, require processing the entire dataset in each iteration and will not update when confronted with an infinite dataset. This implies that even inference algorithms should use only subsets of data to make updates. Some recent progress along these lines has been made, both in the context of variational inference (Hoffman et al., 2013) and MCMC methods (Welling and Teh, 2011). But in this paper, we will 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 datasets and acknowledging that in reality we only have 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 et al., 2014), and a similar method based on concentration bounds was concurrently proposed in Bardenet et al. (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 paper is organized as follows. We introduce our sequential test for speeding up optimization algorithms in 2. Then, we use similar ideas to develop a fast MCMC sampler in 3. We evaluate our methods on a variety of models and datasets 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 $\boldsymbol{\theta} \in \mathbb{R}^D$. The loss function is often defined as an expectation over a dataset $\mathcal{D}_N$ 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 $\boldsymbol{\theta}_{t+1}$ of the optimal parameters is computed using our current estimate $\boldsymbol{\theta}_t$ and the dataset $\mathcal{D}_N$.

Optimization algorithms which process the entire dataset to compute each update are known as 'batch' algorithms. When faced with very large datasets, batch algorithms are clearly wasteful in terms of computational resources. In early iterations, when we are far from the solution, we only need a rough direction to move in parameter space and this can be estimated without looking at every data point in the dataset. 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 will be denoted 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 very 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 and Monro, 1951). A drawback is that the annealing schedule has to be tuned separately for every optimization problem and dataset.

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, but use 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 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 dataset. 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° 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 Non-Negative Matrix Factorization (Korattikara et al., 2011), L_1 regularized problems (Boyles et al., 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 less 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 paper, we will illustrate our ideas on updates of the form $\boldsymbol{\theta}_{t+1} \leftarrow \mathbf{u}_N = (\mathbf{A}_N^T\mathbf{A}_N)^{-1}(\mathbf{A}_N^T\mathbf{b}_N)$ where $\mathbf{A}_N = [\mathbf{a}_1, ..\mathbf{a}_N]^T$, $\mathbf{b}_N = [b_1, ..b_N]^T$ and $\{\mathbf{a}_i, b_i\}$ are functions of the the i^{th} observation $\{\mathbf{x}_i, y_i\}$ and our current parameter estimate $\boldsymbol{\theta}_t$. 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 and Nelder, 1999), L_p-norm regression (Gentle, 2007), Principal Component Analysis (Roweis, 1998) and Non-Negative Matrix Factorization (Kim and Park, 2008).

2.1 Tests for IRLS type algorithms

The update vector $\mathbf{u}_N = (\mathbf{A}_N^T\mathbf{A}_N)^{-1}(\mathbf{A}_N^T\mathbf{b}_N)$ can be interpreted as the least squares estimator, computed from N observations, of the parameters $\mathbf{u}$ in a (fictional) linear

regression model $b_i = \mathbf{a}_i^T \mathbf{u} + \zeta_i$, where ζ is an error term. Since computing $\mathbf{u}_N$ requires $O(N)$ time and is impractical for very large datasets, let us consider approximating $\mathbf{u}_N$ with an estimate $\mathbf{u}_n$ computed from a mini-batch of n data points:

$$\mathbf{u}_n = (\mathbf{A}_n^T \mathbf{A}_n)^{-1} (\mathbf{A}_n^T \mathbf{b}_n) = \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_i^T \right)^{-1} \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{a}_i b_i \right) \tag{1}$$

If we consider $\{\mathbf{a}_i, b_i\}$ as independent samples from a data generating process (or an infinite dataset) and the following assumptions hold:

1. No multicollinearity: $\mathbf{Q_{aa}} = \mathbb{E}[\mathbf{a}\mathbf{a}^T]$ is positive definite
2. Exogeneity: $\mathbb{E}[\zeta | \mathbf{a}] = 0$
3. Homoscedasticity: $\text{Var}[\zeta | \mathbf{a}] = \omega^2$

then $\mathbf{u}_n$ can be shown to be asymptotically (as $n \to \infty$) normal (Verbeek, 2000) under resampling the mini-batch from its generating process. We have:

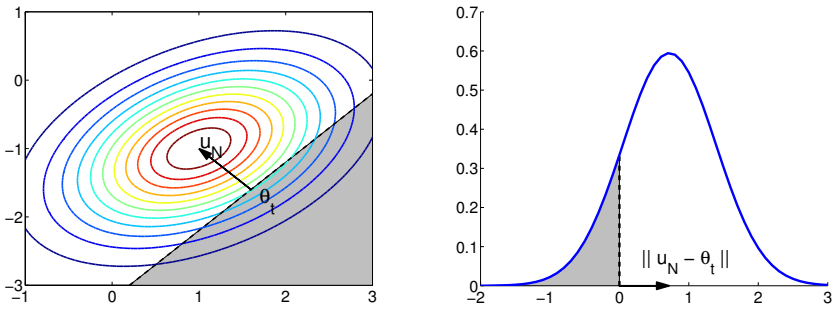
$$\mathbf{u}_n \sim \mathcal{N} \left(\mathbf{u}, \frac{\mathbf{Q_{aa}}^{-1} \omega^2}{n} \right) \tag{2}$$

However, we are interested in the distribution of $\mathbf{u}_n$ under re-sampling the mini-batch without replacement from a dataset of fixed size N. This can be obtained by applying a finite population correction factor (Isserlis, 1918) to Eqn. (2):

$$\mathbf{u}_n \sim \mathcal{N}(\mathbf{u}_N, \mathbf{\Sigma}) \text{ where } \mathbf{\Sigma} = \frac{\mathbf{Q_{aa}}^{-1} \omega^2}{n} \left(1 - \frac{n-1}{N-1} \right) \tag{3}$$

Now, consider the probability that an update computed from the mini-batch ($\mathbf{u}_n - \boldsymbol{\theta}_t$) is more than 90 degrees away from the true update ($\mathbf{u}_N - \boldsymbol{\theta}_t$). This is, by definition:

$$\delta = \int \mathbb{1} \left[\langle \mathbf{u}_n - \boldsymbol{\theta}_t, \mathbf{u}_N - \boldsymbol{\theta}_t \rangle \geq 0 \right] p(\mathbf{u}_n) \mathbf{d}\mathbf{u}_\mathbf{n} \tag{4}$$



(a) (b)

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° 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 $\mu = \|\mathbf{u}_N - \boldsymbol{\theta}_t\|$ and $\sigma^2 = \overline{\mathbf{u}}\Sigma\overline{\mathbf{u}}$. The probability of falling in the shaded region can be calculated as $\Phi\left(\frac{0-\mu}{\sigma}\right)$.

where $\mathbb{1}$ is the indicator function and $p(\mathbf{u}_n)$ is the distribution of $\mathbf{u}_n$ defined in (3). If this probability is high, the mini-batch update $\mathbf{u}_n$ is not very reliable. To compute this probability (see Figure 1), we first shift the origin to $\boldsymbol{\theta}_t$ and compute the marginal distribution of $p(\mathbf{u}_n)$ along the direction of the true update $\mathbf{u}_N - \boldsymbol{\theta}_t$. This distribution is $\mathcal{N}(\mu, \sigma^2)$ where:

$$\mu = \|\mathbf{u}_N - \boldsymbol{\theta}_t\| \tag{5a}$$

$$\sigma = \sqrt{\overline{\mathbf{u}}^T\Sigma\overline{\mathbf{u}}} \text{ where } \overline{\mathbf{u}} = \frac{\mathbf{u}_N - \boldsymbol{\theta}_t}{\|\mathbf{u}_N - \boldsymbol{\theta}_t\|} \tag{5b}$$

From this we can easily compute:

$$\delta = \Phi\left(\frac{0-\mu}{\sigma}\right) \tag{6}$$

where Φ is the cdf of the standard normal distribution. If δ is below a threshold ϵ, we accept the update and set $\boldsymbol{\theta}_{t+1} \leftarrow \mathbf{u}_n$. If not, we decrease δ 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 test $\mu > -\sigma\epsilon^*$ to accept the update.

We use the following estimators for the parameters of the distribution $p(\mathbf{u}_n)$:

$$\hat{\mathbf{u}}_N = \mathbf{u}_n = \left(\sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_i^T\right)^{-1} \left(\sum_{i=1}^{n} \mathbf{a}_i b_i\right) \tag{7a}$$

$$\hat{Q}_{aa} = \frac{\mathcal{A}_n}{n-1} \text{ where } \mathcal{A}_n = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_i^T \tag{7b}$$

$$\hat{\omega}^2 = \frac{e_n}{n-1} \text{ where } e_n = \sum_{i=1}^{n} (b_i - \mathbf{u}_n^T \mathbf{a}_i)^2 \tag{7c}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{\hat{Q}_{aa}^{-1}\hat{\omega}^2}{n}\left(1 - \frac{n-1}{N-1}\right) = \frac{\mathcal{A}_n^{-1} e_n}{n}\left(1 - \frac{n-1}{N-1}\right) \tag{7d}$$

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:

$$\mathbf{u}_n \leftarrow \mathbf{u}_{n-1} + (b_n - \mathbf{a}_n^T \mathbf{u}_{n-1}) \frac{\mathcal{A}_{n-1}^{-1} \mathbf{a}_n}{1 + \mathbf{a}_n \mathcal{A}_{n-1}^{-1} \mathbf{a}_n} \tag{8a}$$

$$\mathcal{A}_n^{-1} \leftarrow \mathcal{A}_{n-1}^{-1} - \frac{\mathcal{A}_{n-1}^{-1} \mathbf{a}_n \mathbf{a}_n^T \mathcal{A}_{n-1}^{-1}}{1 + \mathbf{a}_n \mathcal{A}_{n-1}^{-1} \mathbf{a}_n} \tag{8b}$$

$$e_n \leftarrow e_{n-1} + (b_n - \mathbf{a}_n^T \mathbf{u}_n)(b_n - \mathbf{a}_n^T \mathbf{u}_{n-1}) \tag{8c}$$

The complete test is given in Algorithm 1. In our implemetation, 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.

Algorithm 1 Sequential Test for Optimization

Require: θ_t, $\mathcal{D}_N$, ϵ, N_0, N_{inc}

Ensure: θ_{t+1}

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

Initialize $\mathcal{A}_0 \leftarrow 0$, $\mathcal{B}_0 \leftarrow 0$

for $n = 1 \rightarrow N$ **do**

  Compute $\mathbf{a}_n$, b_n from θ_t, $\mathbf{x}_n$, y_n {problem specific}

  if $n \leq N_0$ **then**

    $\mathcal{A}_n \leftarrow \mathcal{A}_{n-1} + \mathbf{a}_n\mathbf{a}_n^T$, $\mathcal{B}_n \leftarrow \mathcal{B}_{n-1} + \mathbf{a}_n b_n$

  end if

  if $n = N_0$ **then**

    Compute $\mathcal{A}_n^{-1}$ and set $\mathbf{u}_n \leftarrow \mathcal{A}_n^{-1}\mathcal{B}_n$

    $e_n = \sum_{i=1}^{n}(\mathbf{u}_n^T\mathbf{a}_i - b_i)^2$

  end if

  if $n > N_0$ **then**

    Update $\mathbf{u}_n$, $\mathcal{A}_n^{-1}$ and e_n as in Eqn. 8

  end if

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

    $\hat{\mu} \leftarrow \|\mathbf{u}_n - \boldsymbol{\theta}_t\|$, $\hat{\overline{\mathbf{u}}} \leftarrow \frac{\mathbf{u}_n - \boldsymbol{\theta}_t}{\|\mathbf{u}_n - \boldsymbol{\theta}_t\|}$, $\hat{\boldsymbol{\Sigma}} \leftarrow \frac{\mathcal{A}_n^{-1}e_n}{n}\left(1 - \frac{n-1}{N-1}\right)$, $\hat{\sigma} \leftarrow \sqrt{\hat{\overline{\mathbf{u}}}^T\hat{\boldsymbol{\Sigma}}\hat{\overline{\mathbf{u}}}}$

    if $\hat{\mu} > -\hat{\sigma}\epsilon^*$ **then**

      break

    end if

  end if

end for

$\theta_{t+1} \leftarrow \mathbf{u}_n$

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 (section 4.1), an important advantage over SGD is that the main parameter of our algorithm, ϵ, is interpretable: ϵ 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, ϵ can be set independent of the particular optimization problem or dataset 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 and/or dataset, 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 datasets. Given a dataset $\mathcal{D}_N$ consisting of N independent observations $\{\mathbf{x}_1, ...\mathbf{x}_N\}$ which we model using a distribution $p(\mathbf{x}_i; \boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta} \in \mathbb{R}^D$, and a prior distribution $\rho(\boldsymbol{\theta})$, we are interested in the posterior distribution $\mathcal{S}_0(\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\mathcal{D}_N) \propto \rho(\boldsymbol{\theta}) \prod_{i=1}^{N} p(\mathbf{x}_i; \boldsymbol{\theta})$, 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 $I = \langle f \rangle_{\mathcal{S}_0}$ using an empirical average $\hat{I} = \frac{1}{T} \sum_{t=1}^{T} f(\boldsymbol{\theta}_t)$ where $[\boldsymbol{\theta}_1 ... \boldsymbol{\theta}_T]$ are T samples generated from $\mathcal{S}_0$.

Markov Chain Monte Carlo (MCMC) is a popular sampling method that can be used

to generate samples from any distribution $\mathcal{S}_0$ by forward simulating a Markov chain designed to have stationary distribution $\mathcal{S}_0$. A Markov chain with a given stationary distribution can be constructed using the Metropolis-Hastings algorithm (Metropolis et al., 1953), which uses the following rule for transitioning from the current state $\boldsymbol{\theta}_t$ to the next state $\boldsymbol{\theta}_{t+1}$:

1. Draw a candidate state $\boldsymbol{\theta}'$ from a proposal distribution $q(\boldsymbol{\theta}'|\boldsymbol{\theta}_t)$

2. Compute the acceptance probability:

$$P_a = \min\left[1, \frac{\mathcal{S}_0(\boldsymbol{\theta}')q(\boldsymbol{\theta}_t|\boldsymbol{\theta}')}{\mathcal{S}_0(\boldsymbol{\theta}_t)q(\boldsymbol{\theta}'|\boldsymbol{\theta}_t)}\right] \tag{9}$$

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

4. If $u < P_a$ set $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}'$, otherwise set $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$.

Following this rule ensures that the stationary distribution of the Markov chain is $\mathcal{S}_0$. The bias of the MCMC estimator is $\mathbb{E}[\hat{I} - I] = 0$ and its variance is $\mathbb{E}[\hat{I} - \mathbb{E}[\hat{I}]]^2 \approx \sigma^2_{f,\mathcal{S}_0}\tau/T$, where $\sigma^2_{f,\mathcal{S}_0}$ is the variance of f with respect to $\mathcal{S}_0$ and τ is the integrated auto-correlation time. Here the expectations are taken with respect to multiple runs of the Markov chain. Unfortunately, for Bayesian posterior sampling, $\mathcal{S}_0$ involves a product over N likelihood terms. Therefore, for very large datasets, 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 $\mathcal{S}_\epsilon$ instead of $\mathcal{S}_0$ to compute expectations with respect to $\mathcal{S}_0$. Here ϵ is a knob that controls the amount of bias in $\mathcal{S}_\epsilon$. If we increase ϵ, $\hat{I}$ 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 ϵ, 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 $R = \mathbb{E}[(\hat{I} - I)^2]$, i.e. 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 $B = \langle f \rangle_{\mathcal{S}_\epsilon} - \langle f \rangle_{\mathcal{S}_0}$ and $V = \sigma^2_{f,\mathcal{S}_\epsilon} \tau / T$. The optimal setting of ϵ that minimizes the risk depends on the amount of computational time available. If we have an infinite amount of computational time, we should set ϵ 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 way 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 and Teh, 2011) and its successors (Ahn et al., 2012; Patterson and

Teh, 2013; Chen et al., 2014; Ding et al., 2014). Another idea is to approximate the MH test directly using less 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/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, we can compare it directly to the ratio $\frac{\mathcal{S}_0(\boldsymbol{\theta}')q(\boldsymbol{\theta}_t|\boldsymbol{\theta}')}{\mathcal{S}_0(\boldsymbol{\theta}_t)q(\boldsymbol{\theta}'|\boldsymbol{\theta}_t)}$. In the case of Bayesian posterior sampling, we compare:

$$u \gtrless \frac{\rho(\boldsymbol{\theta}') \prod_{i=1}^{N} p(\mathbf{x}_i;\boldsymbol{\theta}')q(\boldsymbol{\theta}_t|\boldsymbol{\theta}')}{\rho(\boldsymbol{\theta}_t) \prod_{i=1}^{N} p(\mathbf{x}_i;\boldsymbol{\theta}_t)q(\boldsymbol{\theta}'|\boldsymbol{\theta}_t)} \tag{10}$$

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

$$\begin{aligned} \mu &= \frac{1}{N}\sum_{i=1}^{N} l_i \;\text{ where }\; l_i = \log p(x_i;\boldsymbol{\theta}') - \log p(x_i;\boldsymbol{\theta}_t) \;\text{ and} \\ \mu_0 &= \frac{1}{N}\log\left[u\frac{\rho(\boldsymbol{\theta}_t)q(\boldsymbol{\theta}'|\boldsymbol{\theta}_t)}{\rho(\boldsymbol{\theta}')q(\boldsymbol{\theta}_t|\boldsymbol{\theta}')}\right] \end{aligned} \tag{11}$$

If $\mu > \mu_0$, we accept the proposal and set $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}'$. If $\mu \leq \mu_0$, we reject the proposal and set $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t$. This reformulation of the MH test makes it very easy to frame it as a statistical hypothesis test: Given μ_0 and a random sample $\{l_1 ... l_n\}$ drawn without replacement from the population $\{l_1 ... l_N\}$, can we confidently decide whether the population mean μ is greater than or less than the threshold μ_0? The answer to this

depends on the precision in the random sample. If the difference between the sample mean $\bar{l}$ and μ_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 $\bar{l}$ (reduce s) till we have enough evidence to make a decision.

More formally, we test the null hypothesis $H_0 : \mu = \mu_0$ vs the alternate $H_1 : \mu \neq \mu_0$. 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

$$s = \frac{s_l}{\sqrt{n}}\sqrt{1 - \frac{n-1}{N-1}} \tag{12}$$

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:

$$t = \frac{\bar{l} - \mu_0}{s} \tag{13}$$

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, i.e. 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 $\delta = 0 < \epsilon$. So, 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 (here 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 ϵ. When ϵ 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 et al. (2015), Alquier et al. (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 ϵ 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 ϵ under regular conditions for the exact Markov chain, and the stationary distribution of the approximate chain $\mathcal{S}_\epsilon \to \mathcal{S}_0$ as $\epsilon \to 0$.

Bardenet et al. (2014) concurrently developed a similar approach which 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/reject decision

Algorithm 2 Approximate MH test

Require: θ_t, θ', ϵ, m, X_N

Ensure: $accept$

Initialize estimated means $\bar{l} \leftarrow 0$ and $\bar{l^2} \leftarrow 0$

Initialize $n \leftarrow 0$

Draw $u \sim$ Uniform[0,1]

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

$done \leftarrow$ **false**

while not done **do**

  Draw mini-batch $\mathcal{X}_m$ of size m without replacement from X_N

  Set $X_N \leftarrow X_N \setminus \mathcal{X}_m$

  Update $\bar{l}$ and $\bar{l^2}$ using $\mathcal{X}_m$.

  $n \leftarrow n + m$

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

  Update student-t statistic $t \leftarrow \frac{\bar{l} - \mu_0}{s}$

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

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

    $accept \leftarrow$ **true** if $\bar{l} > \mu_0$ and **false** otherwise

    $done \leftarrow$ **true**

  end if

end while

than our test. A related method was also proposed in Singh et al. (2012), where the factors of a graphical model are sub-sampled 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 et al. (2014).

4 Experiments

4.1 Optimization

The threshold for passing the hypothesis test, ϵ was set to 0.01 for all algorithms and datasets. 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 $1/1000^{th}$ the size of the dataset. We compare our hypothesis testing method to both the batch algorithm where the whole dataset 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 only plot 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 $\mathbf{x}_i$ and targets y_i, the goal is to minimize the absolute error in our predictions:

$$\min_{\theta} \sum_{i=1}^{N} |y_i - \boldsymbol{\theta}^T \mathbf{x}_i| \tag{14}$$

This can be solved by an Iterative Reweighted Least Squares (IRLS) type algorithm where the weight of the i^{th} data point is $w_i = \frac{1}{\sqrt{|y_i - \boldsymbol{\theta}_t^T \mathbf{x}_i|}}$. Thus we have the following update rule: $\boldsymbol{\theta}_{t+1} \leftarrow (\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{b})$ where $\mathbf{a}_i = w_i\mathbf{x}_i$ 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 Dataset (Bertin-Mahieux et al., 2011). The goal is to predict the release year of a song from 90 audio features and we use the version of the dataset available from the UCI machine learning repository (Bache and 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 $C_1 = 0.01$, $C_2 = 1$ and $C_3 = 0.5$.

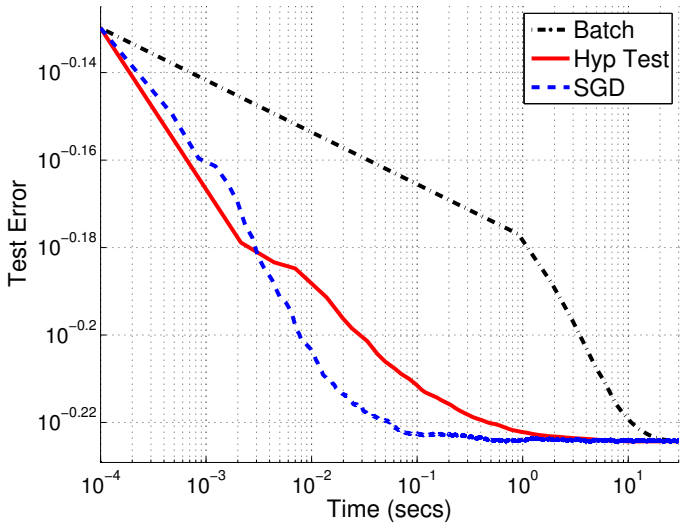


Figure 2: Test error as a function of time for LAD regression on the Million Song Dataset.

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 $r_i = (1 +$

$\exp(-\boldsymbol{\theta}_t^T \mathbf{x}_i))^{-1}$, the update rule is $\boldsymbol{\theta}_{t+1} \leftarrow (\mathbf{A}^T\mathbf{A})^{-1}(\mathbf{A}^T\mathbf{b})$ with $\mathbf{a}_i = w_i\mathbf{x}_i$ and $b_i = w_i\mathbf{x}_i^T\theta_t + \frac{y_i - r_i}{w_i}$. In Figure 3, we compare our hypothesis testing algorithm to the batch algorithm and SGD on the MNIST dataset. We train a classifier for detecting the digit 3 vs all others. For training we used a version of the Infinite dataset with 8.1 Million images available from Loosli et al. (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 we 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 $C_1 = 1$, $C_2 = 10$ and $C_3 = 0.7$.

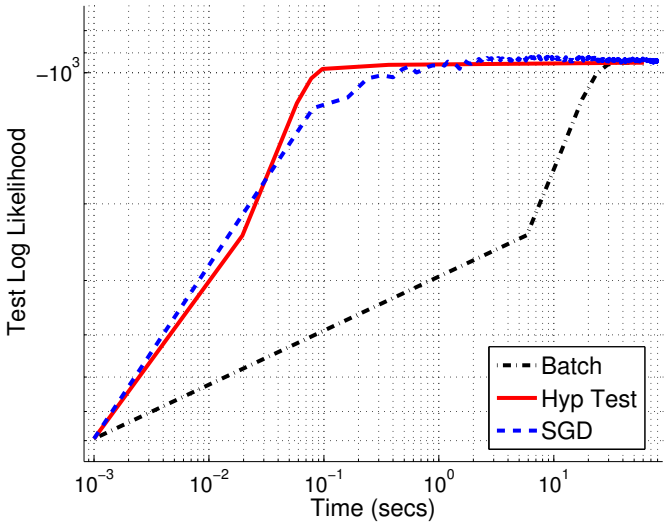


Figure 3: Test log-likelihood as a function of time for logistic regression on the MNIST dataset.

4.1.3 Analysis

We see that on both problems, our algorithm significantly outperforms the batch algorithm by using only the amount 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 datasets, whereas we need to carefully tune the parameters of the learning schedule in SGD. Similar results were reported in Boyles et al. (2011).

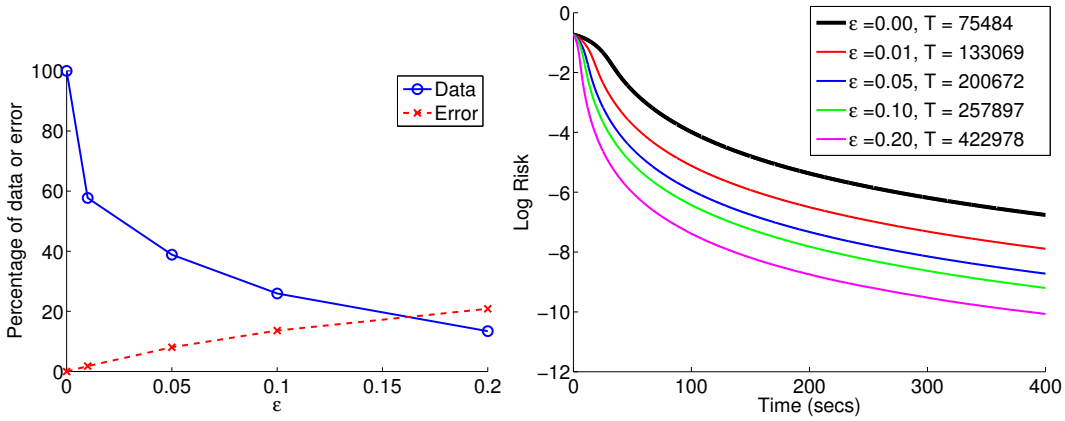
4.2 Sampling

4.2.1 Random Walk - Logistic Regression

We first test our method using a random walk proposal $q(\theta'|\theta_t) = \mathcal{N}(\theta_t, \sigma^2_{RW})$. Although the random walk proposal is not efficient, it is very useful for illustrating our algorithm because the proposal does not contain any information about the target distribution, unlike Langevin or Hamiltonian methods. So, 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 $\mu_0 = \frac{1}{N} \log [u\rho(\theta_t)/\rho(\theta')]$.

The target distribution in this experiment was the posterior for a logistic regression model trained on the MNIST dataset for classifying digits 7 vs 9. The dataset consisted of 12214 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 ≈ 0.5.

In Figure 4(a), we show the percentage of data used (solid blue line) and the per-



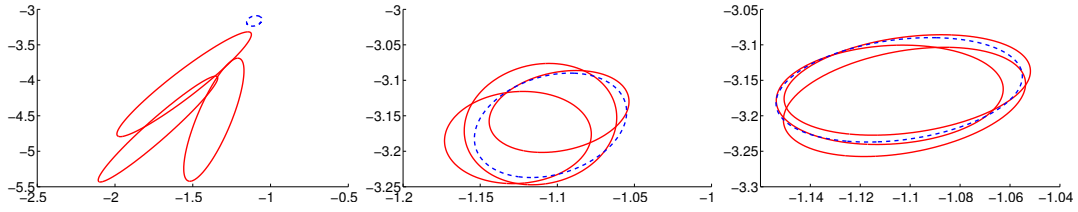
(a) (b)

Figure 4: a) Percentage of data (solid blue line) used and percentage of wrong accept-reject decisions (dashed red line) as a function of ϵ. b) Average Risk in mean prediction of 2037 test points vs wall clock time for different values of ϵ. T denotes the total number of samples collected in 400 secs. Results are for a random walk proposal on a logistic regression model trained on the MNIST dataset.

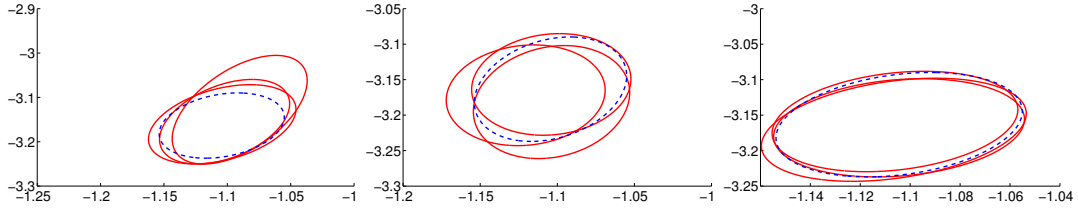
centage of wrong decisions (dashed red line) as a function of ϵ. The amount of data required drops off very fast as ϵ is increased even though the error does not increase much. Note that the setting $\epsilon = 0$ corresponds to the exact MH algorithm.

In Fig. 4(b), 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 $R = B^2 + V = B^2 + \frac{\sigma^2 f}{T}$, 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 $\epsilon > 0$.



(a) ϵ=0, WCT = 50 sec (b) ϵ=0, WCT = 100 sec (c) ϵ=0, WCT = 400 sec



(d) ϵ=0.01, WCT = 50 sec (e) ϵ=0.01, WCT = 100 sec (f) ϵ=0.01, WCT = 400 sec

Figure 5: Marginals θ_1 vs θ_t for $\epsilon = 0$ and $\epsilon = 0.01$ at different values of wall clock time. The blue curve shows the true marginal obtained using a long run of Hybrid Monte Carlo. The red curve shows marginals obtained by running the random walk algorithm 3 times. Results are for a random walk proposal on a logistic regression model trained on the MNIST dataset.

In Figure 5, we show marginals of θ_1 vs θ_2 for $\epsilon = 0$ and $\epsilon = 0.01$ at different values of wall clock time. The red curves are marginals for 3 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 100K 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 and Oja, 2000). When using prewhitened data, 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 $p(x|W) = |\det(W)| \prod_{j=1}^{D} \left[4 \cosh^2(\frac{1}{2} w_j^T x)\right]^{-1}$ 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 ≈ 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 dataset by mixing 1.95 million samples of 4 sources: (a) a Classical music recording (b) street / traffic noise (c) & (d) 2 independent Gaussian sources. To measure the correctness of the sampler, we measure the risk in estimating $I = \mathbb{E}_{p(W|X)} [d_A(W, W_0)]$ where the test function d_A is the Amari distance (Amari et al., 1996) and W_0 is the true unmixing matrix. We computed the ground truth using a long run (T = 100K samples) of the exact MH algorithm. Then we ran each algorithm 10 times, each time for $\approx$ 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 Fig. 6. Note that even after 6400 secs 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 secs 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.

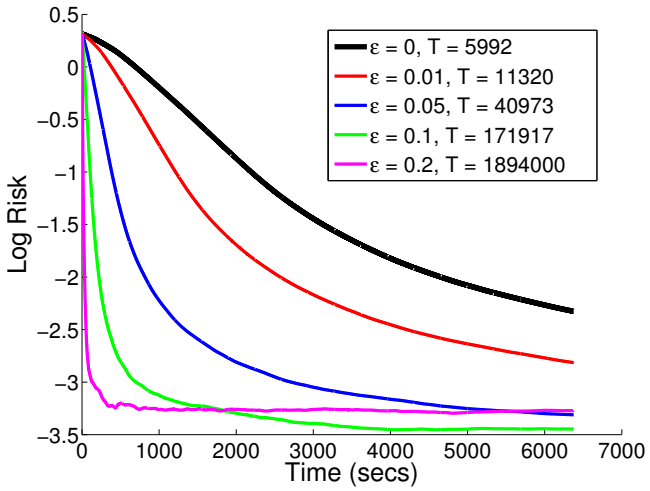


Figure 6: Risk in mean of Amari distance for the ICA model.

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 et al. (2011). Specifically, given D input features (covariates), our parameter $\theta = \{\beta, \gamma\}$ where β is a D dimensional vector of regression coefficients and γ is a D dimensional binary vector that indicates whether a particular feature is included in the model or not. The prior we choose for β is $p(\beta_j|\gamma, \psi) = \frac{1}{2\psi} \exp\left\{-\frac{|\beta_j|}{\psi}\right\}$ if $\gamma_j = 1$. If $\gamma_j = 0$, β_j does not appear in the model. Here ψ is a shrinkage parameter that pushes β_j towards 0, and we choose a prior $p(\psi) \propto 1/\psi$. We also place a right truncated Poisson prior on γ as in Chen et al.

(2011) to control the size of the model, $k = \sum_{j=1}^{D} \gamma_j$:

$$p(\gamma|\lambda) \propto \frac{\lambda^k}{\binom{D}{k} k!} \tag{15}$$

The extra $\binom{D}{k}$ factor appears because there are $\binom{D}{k}$ different models of size k. The likelihood of the data is:

$$p(y_N|X_N, \beta, \gamma) = \prod_{i=1}^{N} \left[\text{sigm}(\beta^T x_i)^{y_i}(1 - \text{sigm}(\beta^T x_i)^{(1-y_i)}\right] \tag{16}$$

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

$$p(\beta, \gamma|X_N, y_N, \lambda) \propto l_N(\beta, \gamma)\|\beta\|_1^{-k}\lambda^k \mathcal{B}(k, D - k + 1) \tag{17}$$

where $\mathcal{B}(a, b) = \int_0^1 t^{a-1}(1 - t)^{b-1}dt$ is the beta function. We do not analytically integrate out λ, 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 3 type of moves that are 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 β without changing the model γ. Specifically, we randomly pick an active component $j : \gamma_j = 1$ and set $\beta_j = \beta_j + \eta$ where $\eta \sim \mathcal{N}(0, \sigma_{update})$. 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 $\beta_j \sim \mathcal{N}(0, \sigma_{birth})$. 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 β_j is discarded. The probabilities of picking these moves $p(\gamma \to \gamma')$ is the same as in Chen et al. (2011). The value of μ_0 used in the MH test for different moves is given below.

1. Update move:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{\|\beta\|_1^{-k}}{\|\beta'\|_1^{-k}} \right] \tag{18}$$

2. Birth move:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{\|\beta\|_1^{-k} p(\gamma \to \gamma') \mathcal{N}(\beta_j | 0, \sigma_{birth})(D-k)}{\|\beta'\|_1^{-(k+1)} p(\gamma' \to \gamma) \lambda k} \right] \tag{19}$$

2. Death move:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{\|\beta\|_1^{-k} p(\gamma \to \gamma') \lambda (k-1)}{\|\beta'\|_1^{-(k-1)} p(\gamma' \to \gamma) \mathcal{N}(\beta_j | 0, \sigma_{birth})(D-k+1)} \right] \tag{20}$$

For more details see Chen et al. (2011).

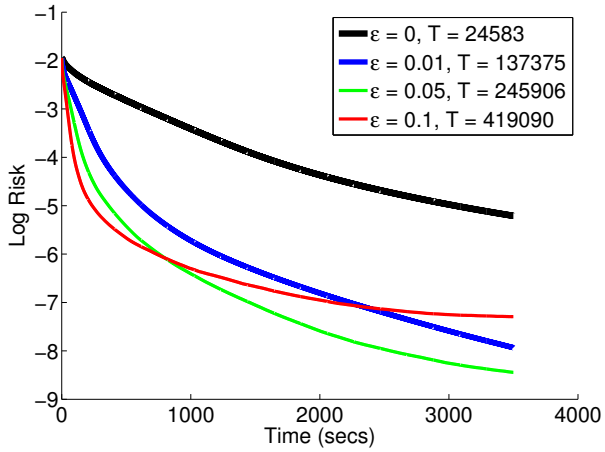


Figure 7: Variable selection using Reversible jump MCMC: Risk in mean prediction on test data for different values of ϵ.

We applied this to the MiniBooNE dataset 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 1's. 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=400K samples using the exact reversible jump algorithm. Then,

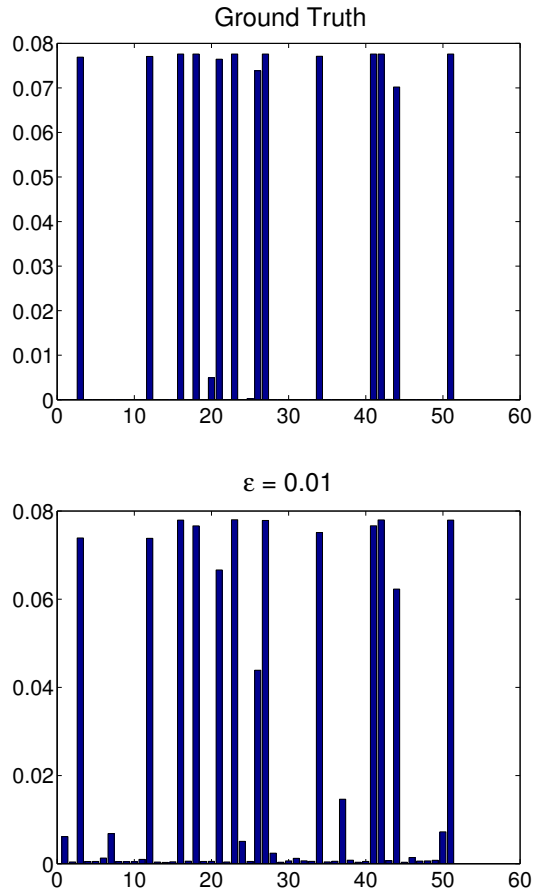


Figure 8: Marginal probability of features to be included in the model.

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, i.e. $p(\gamma_j = 1|X_N, y_N, \lambda)$ in Figure 8.

The acceptance rates for the birth/death moves starts off at $\approx 20\%$ but dies down to $\approx 2\%$ once a good model is found. The acceptance rate for update moves is kept at $\approx 50\%$. The model also suffers from local minima. For the plot in Figure 7, we started with only one variable and we ended up learning models with around 12 features,

giving a classification error $\approx 15\%$. But, if we initialize the sampler with all features included and set β to the MAP value, we learn models with around 45 features, but with a lower classification error $\approx 10\%$. 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 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 and Teh, 2011). In each iteration, we randomly draw a mini-batch $\mathcal{X}_n$ of size n, and propose $\theta' \sim q(.|\theta, \mathcal{X}_n)$ where:

$$q(.|\theta, \mathcal{X}_n) = \mathcal{N}\left(\theta + \frac{\alpha}{2}\nabla_\theta \left\{\frac{N}{n}\sum_{x\in\mathcal{X}_n} \log p(x|\theta) + \log \rho(\theta)\right\}, \alpha\right) \tag{21}$$

Every proposed state θ' is accepted without conducting any MH test. Since the acceptance rate approaches 1 as α goes to 0, we can keep the bias under control by keeping α small. However, we have to use a reasonably large α 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 dataset $\{x_i, y_i\}_{i=1}^N$ 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 Laplacian prior for the parameters $p(\theta) \propto \exp(-\lambda_0\|\theta\|_1)$. For pedagogical reasons, we will restrict ourselves to a toy

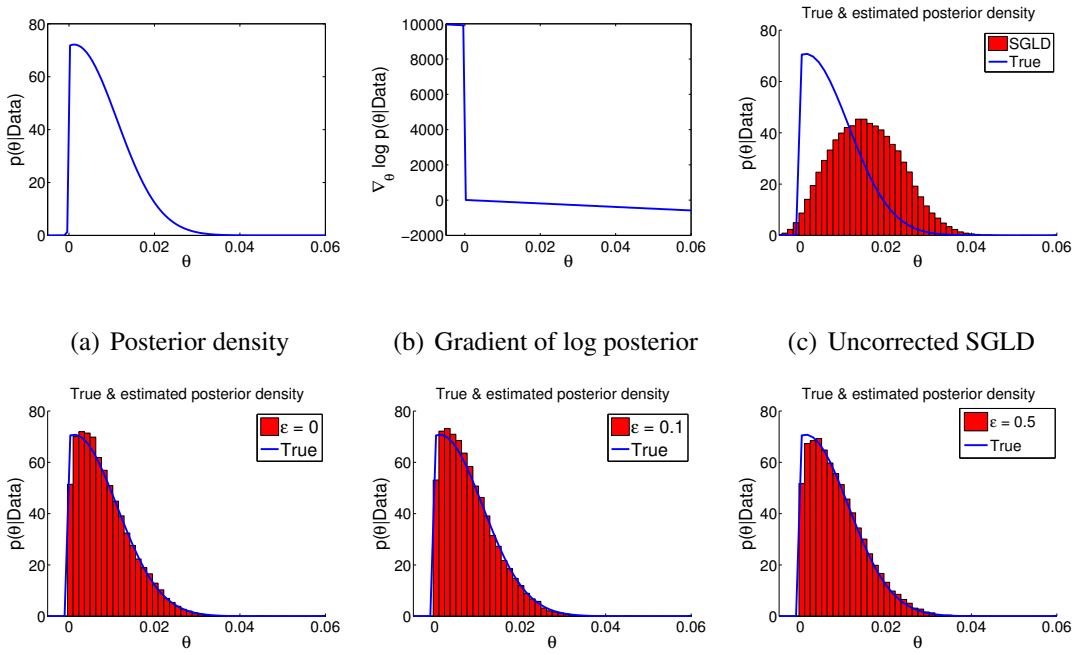
version of the problem where θ and x are one dimensional. We use a synthetic dataset with $N = 10000$ data points generated as $y_i = 0.5x_i + \xi$ where $\xi \sim \mathcal{N}(0, 1/3)$. We choose $\lambda = 3$ and $\lambda_0 = 4950$, so that the prior is not washed out by the likelihood. The posterior density and the gradient of the log posterior are shown in Figures 9(a) and 9(b) respectively.

The empirical histogram of 100000 samples obtained by running SGLD with $\alpha = 5 \times 10^{-6}$ is plotted using red bars in Figure 9(c). 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-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 detailed balance with respect to the posterior distribution if each of the component kernels $q(\theta'|\theta, \mathcal{X}_n)$ satisfy detailed balance. Thus, we can use an MH test with:

$$\mu_0 = \frac{1}{N} \log \left[u \frac{\rho(\theta_t) q(\theta'|\theta_t, \mathcal{X}_n)}{\rho(\theta') q(\theta_t|\theta', \mathcal{X}_n)} \right], \tag{22}$$

The result of running SGLD (keeping $\alpha = 5 \times 10^{-6}$ as before) with the exact MH correction is shown in Figure 9(d). 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 Figure 9(e) ($\epsilon = 0.1$) and Figure



(a) Posterior density (b) Gradient of log posterior (c) Uncorrected SGLD

(d) SGLD corrected using the exact MH test (e) SGLD corrected using our approximate MH test, with $\epsilon = 0.1$. (f) SGLD corrected using our approximate MH test, with $\epsilon = 0.5$.

Figure 9: Pitfalls of using uncorrected SGLD with certain distributions.

9(f) ($\epsilon = 0.5$) 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 which use the entire dataset in each iteration are becoming in-

creasingly 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 paper, 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 datasets or datasets 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.

Acknowledgements

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

References

Ahn, S., Korattikara, A., and Welling, M. (2012). Bayesian posterior sampling via Stochastic Gradient Fisher Scoring. In *International Conference on Machine Learning*.

Alquier, P., Friel, N., Everitt, R., and Boland, A. (2014). Noisy Monte Carlo: Convergence of Markov chains with approximate transition kernels. *Statistics and Computing*, pages 1–19.

Amari, S.-i., Cichocki, A., Yang, H. H., et al. (1996). A new learning algorithm for blind signal separation. *Advances in Neural Information Processing Systems*, pages 757–763.

Bache, K. and Lichman, M. (2013). UCI machine learning repository.

Bardenet, R., Doucet, A., and Holmes, C. (2014). Towards scaling up Markov Chain Monte Carlo: An adaptive subsampling approach. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 405–413.

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011). The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

Boyles, L., Korattikara, A., Ramanan, D., and Welling, M. (2011). Statistical tests for optimization efficiency. In *Advances in Neural Information Processing Systems*, pages 2196–2204.

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic Gradient Hamiltonian Monte Carlo. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1683–1691.

Chen, X., Jane Wang, Z., and McKeown, M. J. (2011). A Bayesian Lasso via reversible-jump MCMC. *Signal Processing*, 91(8):1920–1932.

Chen, Y., Mansinghka, V., and Ghahramani, Z. (2015). Sublinear-time approximate MCMC transitions for probabilistic programs. *arXiv preprint arXiv:1411.1690v2*.

Ding, N., Fang, Y., Babbush, R., Chen, C., Skeel, R. D., and Neven, H. (2014). Bayesian sampling using Stochastic Gradient Thermostats. In *Advances in Neural Information Processing Systems*, pages 3203–3211.

DuBois, C., Korattikara, A., Welling, M., and Smyth, P. (2014). Approximate slice sampling for Bayesian posterior inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 185–193.

Gentle, J. E. (2007). *Matrix algebra: Theory, computations, and applications in statistics*, chapter 6.8.1, pages 232–233. Springer.

Green, P. J. (1995). Reversible jump Markov Chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, 13(4):411–430.

Isserlis, L. (1918). On the value of a mean as calculated from a sample. *Journal of the Royal Statistical Society*, pages 75–81.

Kim, J. and Park, H. (2008). Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, pages 353–362.

Korattikara, A. (2011). Multivariate hypothesis tests for statistical optimization thesis. Master's thesis, University of California, Irvine.

Korattikara, A., Boyles, L., Welling, M., Kim, J., and Park, H. (2011). Statistical optimization of non-negative matrix factorization. In *International Conference on Artificial Intelligence and Statistics*, pages 128–136.

Korattikara, A., Chen, Y., and Welling, M. (2014). Austerity in MCMC land: Cutting the Metropolois-Hastings budget. In *International Conference on Machine Learning*.

Loosli, G., Canu, S., and Bottou, L. (2007). Training invariant support vector machines using selective sampling. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J., editors, *Large Scale Kernel Machines*, pages 301–320. MIT Press, Cambridge, MA.

McCullagh, P. and Nelder, J. (1999). *Generalized linear models*. Chapman & Hall, CRC.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087.

Ouyang, Z. (2008). *Bayesian Additive Regression Kernels*. PhD thesis, Duke University.

Patterson, S. and Teh, Y. W. (2013). Stochastic Gradient Riemannian Langevin Dynamics on the probability simplex. In *Advances in Neural Information Processing Systems*, pages 3102–3110.

Pillai, N. S. and Smith, A. (2014). Ergodicity of approximate MCMC chains with applications to large data sets. *arXiv preprint arXiv:1405.0182*.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407.

Roweis, S. (1998). EM algorithms for PCA and SPCA. In *Advances in Neural Information Processing Systems*, pages 626–632.

Singh, S., Wick, M., and McCallum, A. (2012). Monte Carlo MCMC: Efficient inference by approximate sampling. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1104–1113. Association for Computational Linguistics.

Verbeek, M. (2000). *A Guide to Modern Econometrics*. John Wiley and Sons.

Welling, M. (2014). Exploiting the Statistics of Learning and Inference. *arXiv preprint arXiv:1402.7025*.

Welling, M. and Teh, Y. (2011). Bayesian learning via Stochastic Gradient Langevin Dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 681–688.