Approximate Slice Sampling for Bayesian Posterior Inference

Christopher DuBois[*] GraphLab, Inc.

Anoop Korattikara[*] Dept. of Computer Science UC Irvine

Max Welling Informatics Institute University of Amsterdam

Padhraic Smyth Dept. of Computer Science UC Irvine

Abstract

In this paper, we advance the theory of large scale Bayesian posterior inference by introducing a new approximate slice sampler that uses only small mini-batches of data in every iteration. While this introduces a bias in the stationary distribution, the computational savings allow us to draw more samples in a given amount of time and reduce sampling variance. We empirically verify on three different models that the approximate slice sampling algorithm can significantly outperform a traditional slice sampler if we are allowed only a fixed amount of computing time for our simulations.

1 Introduction

In a time where the amount of data is expanding at an exponential rate, one might argue that Bayesian posterior inference is neither necessary nor a luxury that one can afford computationally. However, the lesson learned from the "deep learning" literature is that the best performing models in the context of big data are models with very many adjustable degrees of freedom (in the order of tens of billions of parameters in some cases). In line with the nonparametric Bayesian philosophy, real data is usually more complex than any finitely parameterized model can describe, leading to the conclusion that even in the context of big data the best performing model will be the most complex model that does not overfit the data. To find that boundary between under- and overfitting, it is most convenient to over-parameterize a model and regularize the model capacity back through methods such as regularization penalties, early stopping, adding noise to the input features, or as is now popular in the context of deep learning, dropout. We believe deep learning is so successful because it provides the means to train arbitrarily complex models on very large datasets that can be effectively regularized.

Methods for Bayesian posterior inference provide a principled alternative to frequentist regularization techniques. The workhorse for approximate posterior inference has long been MCMC, which unfortunately is not (yet) quite up to the task of dealing with very large datasets. The main reason is that every iteration of sampling requires $O(N)$ calculations to decide whether a proposed parameter is accepted or not. Frequentist methods based on stochastic gradients are orders of magnitude faster because they effectively exploit the fact that far away from convergence, the information in the data about how to improve the model is highly redundant. Therefore, only a few data-cases need to be queried in order to make a good guess on how to improve the model. Thus, a key question is whether it is possible to design MCMC algorithms that behave more like stochastic gradient based algorithms. Some methods have appeared recently [1, 13, 5] that use stochastic minibatches of the data at every iteration rather then the entire dataset. These methods exploit an inherent trade-off between bias (sampling from the wrong distribution asymptotically) and variance (error due to sampling noise). Given a limit on the total sampling time, it may be beneficial to use a biased procedure if it allows you to sample faster, resulting in lower error due to variance.

The work presented here is an extension of [5] where sequential hypothesis testing was used to adaptively choose the size of the mini-batch in an approximate Metropolis-Hastings step. The size of the mini-batch is just large enough so that the proposed sample is accepted or rejected with sufficient confidence. That work used a random walk which is well known to be rather slow mixing. Here we show that a similar strategy can be exploited to speed up a slice sampler [10], which is known to have much better mixing behav-

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33. Copyright 2014 by the authors.

∗. The first two authors contributed equally.

ior than an MCMC sampler based on a random walk kernel.

This paper is organized as follows. First, we review the slice sampling algorithm in Section 2. In Section 3, we discuss the bias-variance trade-off for MCMC algorithms. Then, we develop an approximate slice sampler based on stochastic minibatches in Section 4. In Section 5 we test our algorithm on a number of models, including L_1-regularized linear regression, multinomial regression and logistic regression. We conclude in Section 6 and discuss possible future work.

2 Slice Sampling

Slice sampling [10] is an MCMC method for sampling from a (potentially unnormalized) density $P_0(\theta)$, where $\theta \in \Theta \subset \mathbb{R}^D$, by sampling uniformly from the $D+1$ dimensional region under the density. In this section, we will first introduce slice sampling for univariate distributions and then describe how it can be easily extended to the multivariate case.

We start by defining an auxiliary variable h, and a joint distribution $p(\theta, h)$ that is uniform over the region under the density (i.e. the set $\{(\theta, h) : \theta \in \Theta, 0 < h < P_0(\theta)\}$) and 0 elsewhere. Since marginalizing $p(\theta, h)$ over h yields $P_0(\theta)$, sampling from $p(\theta, h)$ and discarding h is equivalent to sampling from $P_0(\theta)$. However, sampling from $p(\theta, h)$ is not straightforward and is usually implemented as an MCMC method where in every iteration t, we first sample $h_t \sim p(h|\theta_{t-1})$ and then sample $\theta_t \sim p(\theta|h_t)$.

Sampling $h_t \sim p(h|\theta_{t-1})$ is trivial since $p(h|\theta_{t-1})$ is just $\texttt{Uniform}[0, P_0(\theta_{t-1})]$. For later convenience, we will introduce the uniform random variable $u_t \sim \texttt{Uniform}[0, 1]$, so that h_t can be defined deterministically as $h_t = u_t P_0(\theta_{t-1})$. Now, $p(\theta|h_t)$ is a uniform distribution over the 'slice' $S[u_t, \theta_{t-1}] = \{\theta : P_0(\theta) > u_t P_0(\theta_{t-1})\}$. Sampling $\theta_t \sim p(\theta|h_t)$ is hard because of the difficulty in finding all segments of the slice in a finite amount of time.

The authors of [10] developed a method where instead of sampling from $p(\theta|h_t)$ directly, a 'stepping-out' procedure is used to find an interval (L, R) around θ_{t-1} that includes most or all of the slice, and θ_t is sampled uniformly from the part of (L, R) that is on $S[u_t, \theta_{t-1}]$. Pseudo code for one iteration of slice sampling is shown in Algorithm 1. A pictorial illustration is given in Figure 1.

The stepping out procedure involves placing a window (L, R) of width w around θ_{t-1} and widening the interval by stepping away from θ_{t-1} in opposite directions until L and R are no longer in the slice. The maximum number of step-outs can be limited by a constant $V_{\max}$. Note that the initial placement of the interval (L, R) around θ_{t-1} in Algorithm 1 is random, and $V_{\max}$ is randomly divided into a limit on stepping to the left, V_L, and a limit on stepping to the right, V_R. These are necessary for detailed balance to hold, as described in [10].

Once a suitable interval containing all or most of the slice has been found, θ_t is sampled by repeatedly drawing a candidate state θ_{prop} uniformly from (L, R) until it falls on the slice S. At this point, we set $\theta_t \leftarrow \theta_{\text{prop}}$ and move to the next iteration. If a proposed state θ_{prop} does not fall on the slice, the interval (L, R) can be shrunk to $(L, \theta_{\text{prop}})$ or $(\theta_{\text{prop}}, R)$ such that θ_{t-1} is still included in the new interval. A proof that Algorithm 1 defines a Markov chain with the correct stationary distribution $p(\theta, h)$ can be found in [10].

Algorithm 1 Slice Sampling

Input: θ_{t-1}, w, $V_{\max}$
Output: θ_t
  Draw $u_t \sim \texttt{Uniform}[0, 1]$
    // Pick an initial interval randomly:
  Draw $\overline{w} \sim \texttt{Uniform}[0, 1]$
  Initialize $L \leftarrow \theta_{t-1} - w\overline{w}$ and $R \leftarrow L + w$
    // Determine maximum steps in each direction:
  Draw $\overline{v} \sim \texttt{Uniform}[0, 1]$
  Set $V_L \leftarrow \texttt{Floor}(V_{\max}\overline{v})$ and $V_R \leftarrow V_{\max} - 1 - V_L$
    // Obtain boundaries of slice:
  while $V_L > 0$ **and** $\texttt{OnSlice}(L, \theta_{t-1}, u_t)$ **do**
    $V_L \leftarrow V_L - 1$
    $L \leftarrow L - w$
  end while
  while $V_R > 0$ **and** $\texttt{OnSlice}(R, \theta_{t-1}, u_t)$ **do**
    $V_R \leftarrow V_R - 1$
    $R \leftarrow R + w$
  end while
    // Sample until proposal is on slice:
  loop
    Draw $\eta \sim \texttt{Uniform}[0, 1]$
    $\theta_{\text{prop}} \leftarrow L + \eta(R - L)$
    if $\texttt{OnSlice}(\theta_{\text{prop}}, \theta_{t-1}, u_t)$ **then**
      $\theta_t \leftarrow \theta_{\text{prop}}$
      break
    end if
    if $\theta_{\text{prop}} < \theta_{t-1}$ **then**
      $L \leftarrow \theta_{\text{prop}}$
    else
      $R \leftarrow \theta_{\text{prop}}$
    end if
  end loop

The algorithm uses Procedure `OnSlice`, both during the stepping-out phase and the rejection sampling phase, to check whether a point (L, R or θ_{prop}) lies on the slice $S[u_t, \theta_{t-1}]$. In a practical implementation,

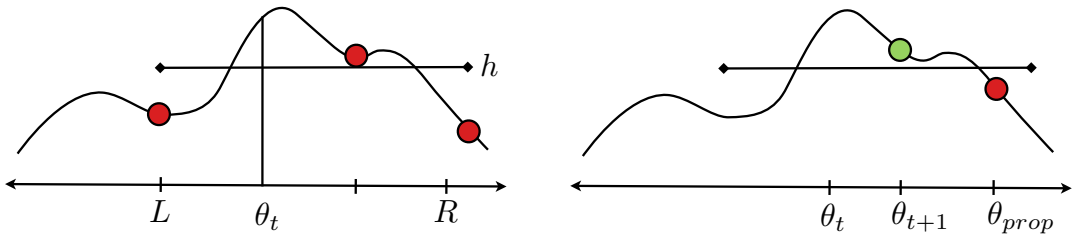

Figure 1: Illustration of the slice sampling procedure. Left: A "slice" is chosen at a height h and with endpoints (L, R) that both lie above the unnormalized density (the curved line). Right: We sample proposed locations θ_{prop} until we obtain a sample on the slice (in green), which is our θ_{t+1}. The proposed method replaces each function evaluation (represented as dots) with a sequential hypothesis test.

Procedure 2 `OnSlice`
Input: θ', θ, u
Output: Is θ' on $S[u_t, \theta_{t-1}]$?
1: **return** $P_0(\theta') > uP_0(\theta)$

$P_0(\theta)$ is calculated only once per iteration of the slice sampler, and does not have to be recomputed every time Procedure 2 is used.

One way to generalize the slice sampling algorithm to multiple dimensions is to pick a direction in parameter space in each iteration and update θ only in the chosen direction. The direction can be picked uniformly at random from one of the co-ordinate directions or from all possible directions. It is also valid to cycle through the co-ordinate directions instead of picking one at random.

Relative to other MCMC techniques, slice sampling can allow for larger moves, allowing one to reduce autocorrelation in the samples of the Markov chain and thereby explore the parameter space more efficiently. The technique has been extended to multivariate distributions [12], truncated normal distributions [7], latent Gaussian models [8, 11], and others.

3 The Bias Variance Trade-off

The samples produced by an MCMC algorithm such as slice sampling are used to estimate the expectation of a function $f(\theta)$ with respect to the distribution $P_0(\theta)$, i.e. we estimate $I = \langle f \rangle_{P_0}$ 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 from the Markov chain. Since the equilibrium distribution of the Markov chain is $P_0(\theta)$, $\hat{I}$ is an unbiased estimator of I, if we ignore burn-in. The variance of the estimator is $V \approx \frac{\sigma^2_{f,P_0}\tau}{T}$, where σ^2_{f,P_0} is the variance of f with respect to P_0 and τ is the integrated auto-correlation time (a measure of correlation between successive samples of the Markov chain).

For many problems of interest, it is quite difficult to collect a large number of samples to estimate I with sufficiently low variance. A common example is Bayesian posterior inference given a dataset with billions of data points. Evaluating the posterior distribution is very expensive because it involves computing the likelihood of every datapoint in the dataset. Since a typical MCMC method has to do this at least once for each sample that it generates, we can collect only a limited number of samples in a reasonable amount of computational time. As a result, the variance of $\hat{I}$ is often too high to be of much practical use.

However, if we are willing to allow some bias in the stationary distribution of the Markov chain, we do not have to evaluate the posterior density exactly in each step. This will allow us to speed up the Markov chain simulation and collect a large number of samples to reduce variance quickly. The higher the bias we can tolerate, the faster we can reduce variance. This bias-variance trade-off has been exploited to develop efficient approximate MCMC algorithms such as Stochastic Gradient Langevin Dynamics[13], Stochastic Gradient Fisher Scoring[1] and sequential Metropolis-Hastings[5].

Instead of the target distribution P_0, approximate MCMC algorithms converge to a slightly biased stationary distribution P_ϵ, where ϵ is a knob of the algorithm that controls the amount of bias. Thus, we will estimate $I = \langle f \rangle_{P_0}$ with $\hat{I} = \frac{1}{T}\sum_{t=1}^{T} f(\theta_t)$ where the θ_t's are samples from P_ϵ instead of P_0. As in [5], the quality of the estimator $\hat{I}$ can be assessed using its risk. The risk of $\hat{I}$ can be defined as $R = \mathbb{E}[(I - \hat{I})^2]$, where the expectation is over multiple runs of the Markov chain. There are two contributors to risk, bias (B) and variance (V), and these are related as $R = B^2 + V$. If we ignore burn-in, it can be shown that $B = \langle f \rangle_{P_0} - \langle f \rangle_{P_\epsilon}$ and $V = \frac{\sigma^2_{f,P_\epsilon}\tau}{T}$.

The optimal setting of ϵ that minimizes the risk depends on the computational time available. If we have an infinite amount of computational time, we can bring down the variance to zero by drawing an infinite number of samples and we set $\epsilon = 0$ so that the bias is zero as well. This is the setting which traditional unbiased MCMC methods use. However, given an finite amount of computational time, this setting is not optimal. It might be more beneficial to allow a small amount of bias if we can decrease variance faster. This motivates the need for approximate MCMC algorithms such as [1, 13, 5]. In the next section, we will develop a new approximate MCMC algorithm by replacing the expensive density comparisons in slice sampling with a sequential hypothesis test as in [5].

4 Approximate Slice Sampling

In Bayesian posterior inference, we are given a dataset X_N consisting of N independent observations $\{x_1, ...x_N\}$ which we model using a distribution $p(x|\theta)$ parameterized by $\theta \in \mathbb{R}^D$. We choose a prior distribution $\rho(\theta)$ and the goal is to generate samples from the posterior distribution $\mathcal{P}_0(\theta) \propto \rho(\theta) \prod_{i=1}^{N} p(x_i|\theta)$.

A typical sampling algorithm must evaluate the posterior density exactly, atleast once for each sample that it generates. In slice sampling, this evaluation is performed by the `OnSlice` procedure which tests whether a given point (L,R or θ_{prop}) lies on the slice or not. When the dataset has billions of datapoints, computing the posterior density is very expensive as it requires $O(N)$ evaluations of the likelihood. Note that, in slice sampling, this has to be done multiple times for each sample we generate. As noted in [5], performing $O(N)$ computations for 1 bit of information, i.e. whether a point is on the slice, is not a cogent use of computational resources. Therefore, in this section, we will develop an approximate slice sampling algorithm by cutting down the computational time of the `OnSlice` procedure.

The `OnSlice` procedure determines whether a point θ' is on the slice $S[u, \theta]$ by checking if $P_0(\theta') > uP_0(\theta)$. When P_0 is a Bayesian posterior distribution, we can take the logarithm of both sides of this inequality and divide by N to rephrase the test as $\mu > \mu_0$, where:

$$\mu_0 = \frac{1}{N} \log \left[u_t \frac{\rho(\theta_{t-1})}{\rho(\theta')} \right], \quad \text{and}$$
$$\mu = \frac{1}{N} \sum_{i=1}^{N} l_i \quad \text{where} \quad l_i = \log p(x_i; \theta') - \log p(x_i; \theta_t) \tag{1}$$

i.e., if $\mu > \mu_0$, we can conclude that θ' lies on the slice. In other words, we are testing if the mean of the finite population $\{l_1...l_N\}$ is greater than a constant that does not depend on the data.

A similar scenario appears in the Metropolis-Hastings algorithm, where the mean difference in log-likelihoods is compared to a constant to determine whether to accept or reject a proposed sample. A sequential hypothesis testing procedure was recently proposed in [5] to cut down the computational budget of this comparison. We can easily apply this test to efficiently compare μ and μ_0 in the slice sampling algorithm.

The sequential test is follows. We randomly draw a mini-batch $\mathcal{X} = \{x_1...x_n\}$ of size $n < N$ without replacement from X_N and compute the difference in log-likelihoods $\{l_1, \ldots, l_n\}$. The goal is to use the sample mean $\bar{l} = \frac{1}{n} \sum_{i=1}^{n} l_i$ to decide whether the population mean μ is greater than μ_0 or not. We can do this confidently if the difference between $\bar{l}$ and μ_0 is significantly larger than s, the standard deviation of $\bar{l}$. If not, we can add more data to the mini-batch until we can make a more confident decision.

More formally, we test the null hypothesis $H_0 : \mu = \mu_0$ vs the alternate $H_a : \mu \neq \mu_0$. To do this, we first compute the sample standard deviation $s_l = \sqrt{\sum_{i=1}^{n} (l_i - \bar{l})^2/(n-1)}$ and then estimate the standard deviation of $\bar{l}$ as:

$$s = \frac{s_l}{\sqrt{n}} \sqrt{1 - \frac{n}{N}} \tag{2}$$

Here $\sqrt{1 - \frac{n}{N}}$ is a finite population correction factor to account for the correlation between samples drawn without replacement from a finite population. Then, we compute the test statistic:

$$t = \frac{\bar{l} - \mu_0}{s} \tag{3}$$

If n is large enough for the central limit theorem to hold, t follows a standard Student-t distribution with $n - 1$ degrees of freedom when the null hypothesis $\mu = \mu_0$ is true. To determine whether $\bar{l}$ is significantly different from μ_0, we compute the probability $\delta = 1 - \phi_{n-1}(|t|)$ where $\phi_{n-1}(.)$ is the cdf of the Student-t distribution. If δ is less than a suitably chosen threshold ϵ, we can reject the null hypothesis and confidently say that $\bar{l}$ is different from μ_0. In this case, we can decide that θ' is on the slice if $\bar{l} > \mu_0$, or that θ' is not on the slice if $\bar{l} \leq \mu_0$.

If δ is greater than ϵ, the difference between $\bar{l}$ and μ_0 is not large compared to the standard deviation of $\bar{l}$. In this event, we add more data to the mini-batch to increase the precision of $\bar{l}$. We keep adding more data until δ falls below ϵ. At this point, we have enough precision in $\bar{l}$ to decide whether θ' is on the

slice or not. This procedure will necessarily terminate because when $n = N$, the standard deviation $s = 0$, $\bar{l} = \mu$, $t = \pm\infty$ and $\delta = 0 < \epsilon$. Also, in this case, we will make the correct decision since $\bar{l} = \mu$.

The sequential test is illustrated in Procedure `OnSliceApprox`. Here we start with a mini-batch of size $n = m$ and increase n by m datapoints until a decision is made. An efficient, but approximate, slice sampling algorithm can be obtained by replacing all calls to the `OnSlice` procedure in Algorithm 1 with `OnSliceApprox`.

Procedure 3 `OnSliceApprox`

Input: θ', θ, u, ϵ, m
Output: Is θ' on $S[u_t, \theta_{t-1}]$?

Initialize mean estimates $\bar{l} \leftarrow 0$ and $\overline{l^2} \leftarrow 0$
Initialize $n \leftarrow 0$
$\mu_0 \leftarrow \frac{1}{N}[\log u + \log \rho(\theta) - \log \rho(\theta')]$
loop
  Draw mini-batch $\mathcal{X}$ of size min $(m, N-n)$ without replacement from X_N and set $X_N \leftarrow X_N \backslash \mathcal{X}$
  Update $\bar{l}$ and $\overline{l^2}$ using $\mathcal{X}$, and $n \leftarrow n + |\mathcal{X}|$
  Estimate std $s \leftarrow \sqrt{1 - \frac{n}{N}} \sqrt{\frac{\overline{l^2} - (\bar{l})^2}{n-1}}$
  Compute $\delta \leftarrow 1 - \phi_{n-1}\left(\left|\frac{\bar{l} - \mu_0}{s}\right|\right)$
  if $\delta < \epsilon$ **then**
    break
  end if
end loop
return $\bar{l} > \mu_0$

It should be noted that our algorithm will behave erratically if the central limit theorem does not hold. This may happen, for example, when the dataset is sparse or has extreme outliers. However, the central limit assumption can be easily verified empirically at a few values of θ before running the algorithm to avoid such pathological situations.

5 Experiments

In this section we evaluate the ability of the proposed approximate slice sampler to obtain samples from several posterior distributions of interest at a reduced computational cost.

5.1 Banana dataset

We first consider posterior inference of $\theta = (\theta_1, \theta_2)|y$ under the model

$$y_i|\theta \sim N(\theta_1 + \theta_2^2, \sigma_y^2) \qquad \theta_j \sim N(0, \sigma_\theta^2)$$

We generate $N = 1000$ observations where $\theta_1 + \theta_2^2 = 1$, $\sigma_y^2 = 4$ and $\sigma_\theta^2 = 1$, similar to [6]. The variables θ_1 and θ_2 have a highly correlated posterior distribution.

Figure 2 shows a density estimate of samples using $\epsilon = 0.1, 0.01, 0.001$ and 0.0 with $V_{\max} = 100$ and $w = 0.25$. We see that the approximate slice sampler provides a good estimate of the posterior even with fairly large values of ϵ. Even with $\epsilon = 0.1$ we observe that the marginal distributions match the true posterior with $\epsilon = 0$ quite well.

In Figure 3 we show the total time required for a given number of iterations of the Markov chain for each setting of ϵ, as well as summaries of the number of data points seen and the number of tests performed per iteration. For larger values of ϵ we achieve significant time savings, evidently because fewer data points have been used. It appears that the number of tests used stays constant across the choice of ϵ. We see that the number of data points required to make the statistical decisions for those tests decreases as ϵ increases. As fewer data points are required, the time required to perform 10000 iterations in turn decreases.

5.2 L_1 Regularized Linear Regression

Next, we test our method on the posterior distribution of a linear regression model with a Laplacian prior on the parameters. We trained the model using a 100K subset of the Million Song Dataset year prediction challenge[2]. The goal is to predict the release year of a song from audio features. There are 90 features to which we added a constant feature of ones.

We use the risk in estimating the 'average prediction' as a measure to compare the relative performance of the exact and approximate slice sampling algorithms. The average prediction is defined as the expectation of the prediction under the posterior distribution, i.e. $\mathbb{E}_{p(\theta|X_N)}[p(x^*|\theta)]$, where x^* is a test point. To compute risk, we first compute the true average prediction using a long run (100K samples) of the exact slice sampling algorithm initialized at the mode of the distribution. Then, we run the exact and approximate slice samplers at different values of ϵ for 500 seconds. On average, we obtained $T = 12214$, 48122 and 104732 samples with $\epsilon = 0$, 0.2 and 0.3 respectively in 500 secs of computation. Each algorithm is run 10 times, and the risk is calculated as the squared error in the estimated mean prediction averaged over the 10 Markov chain simulations. We plot the risk in the average prediction, futher averaged over 1000 randomly chosen test points, for different values of ϵ in Figure 4. $\epsilon = 0$ denotes the exact slice sampling algorithm.

For a typical sampling algorithm, the risk is initially dominated by the burn-in bias. Burn-in usually hap-

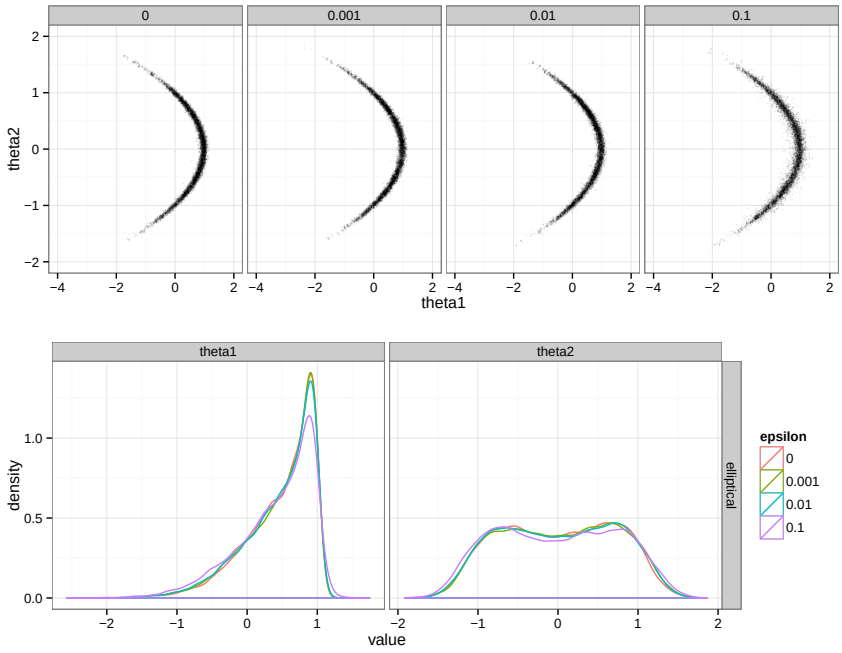

Figure 2: Effect of the tuning parameter ϵ on the quality of the approximate sampler for the banana-shaped posterior. Top: Samples using $V_{\max} = 100$ and various settings of ϵ. Bottom: Density estimates of the marginal distributions. Even with $\epsilon = 0.1$ the marginal distributions are fairly accurate.

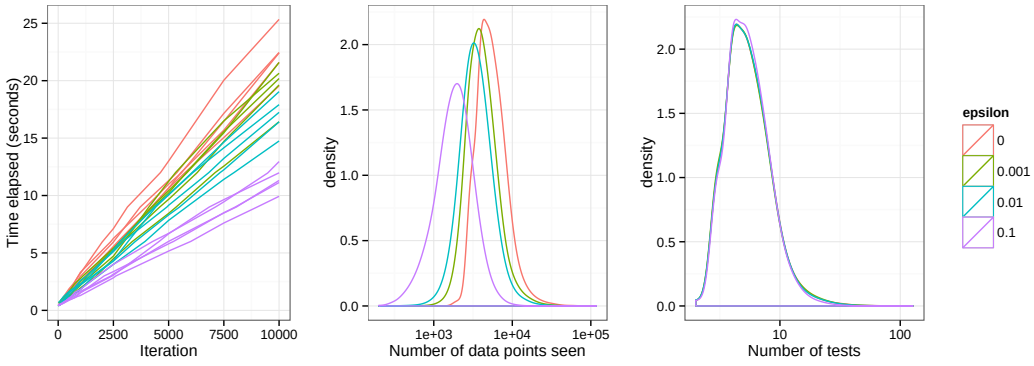

Figure 3: Effect of ϵ on computational requirements for MCMC for the banana-shaped posterior. Results from five different random initializations are shown for each setting. Left: Time taken versus iteration for sampling using the approximate slice sampler. Middle: Density estimate for the number of data points seen per iteration. Right: Density estimate for the number of tests performed per iteration.

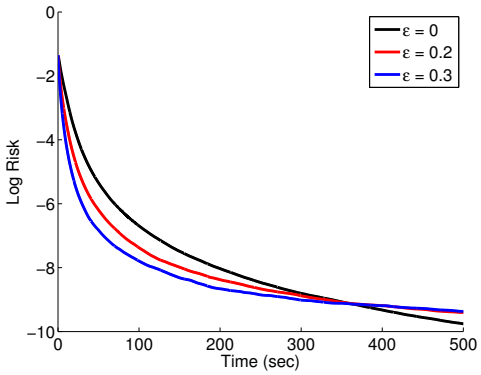

Figure 4: Risk in average prediction of test data for a linear regression model with a Laplacian prior

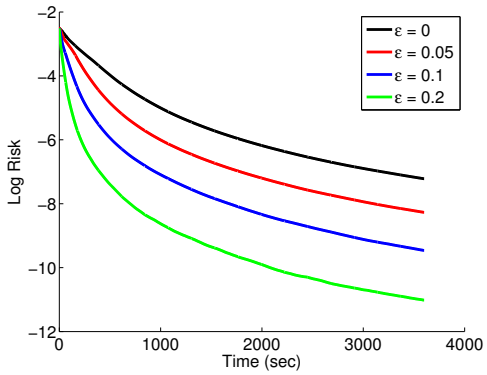

Figure 5: Risk in average prediction of test data for a Multinomial Regression model

pens very fast, after which the risk is dominated by the sampling variance and reduces as $O(1/T)$. For approximate MCMC algorithms, after collecting a large number of samples, the sampling variance will eventually be dominated by the asymptotic bias in the stationary distribution. This can be seen in Figure 4. Algorithms with a high ϵ are able to burn-in and reduce variance quickly. After collecting a large number of samples, their asymptotic bias will dominate the variance and prevent the risk from reducing further. The exact slice sampler is slower to burn-in and reduce variance, but it can achieve zero risk given an infinite amount of computational time.

5.3 Multinomial Regression

We then tested our method on a multinomial regression model trained on the MNIST dataset of handwritten digits. There are 60000 training points with 784 dimensions each. We reduced the dimensionality to 50 using PCA and added a constant feature of ones. Since there are 10 classes, this resulted in a 459 dimensional parameter vector. We chose a Gaussian prior over the parameters.

We plot the risk in estimating the average prediction in Figure 5. As in the previous experiment, ground truth was computed using 100K samples collected from the exact slice sampler and the risk was computed by averaging the squared error over 10 Markov chains for each value of ϵ. We also average the risk over 10 randomly chosen test points. We ran each algorithm for one hour and obtained $T = 13805$, 27587, 56409 and 168624 samples for $\epsilon = 0$, 0.05, 0.1 and 0.2 respectively.

The lowest risk after one hour of computation was achieved by the approximate slice sampler with the highest bias in the stationary distribution. Thus, even after one hour of computation, the risk is still dominated by sampling variance. If we were to run for a very long time, we would expect the exact slice sampler to outperform all the approximate algorithms.

5.4 Logistic Regression

Finally, we test the performance of our approximate slice sampler on the posterior distribution of a logistic regression model. The model was trained on the Covertype dataset[2], where the goal is to predict forest cover type using cartographic variables. There are 7 cover types in the dataset, but we trained a 1-vs-rest classifier for the most populous class (Lodgepole pine). There were a total of 464809 training points with 54 features (to which we add a 'constant' feature of ones). We chose a Gaussian prior over the parameters.

To compute ground truth, we used a long run (100K samples) of the exact slice sampler initialized from the mode of the distribution. Then, we ran the exact and approximate slice sampling algorithms at different values of ϵ for 3 hours each. We obtained $T = 14808$, 19182, 54156 and 289497 samples with $\epsilon = 0$, 0.05, 0.075 and 0.1 respectively. In Figure 6, we plot the risk in estimating the average prediction of 10,000 randomly chosen test points.

The risk is initially dominated by the burn-in bias, and then by sampling variance. Approximate MCMC algorithms with higher values of ϵ are able to reduce both of these very quickly as shown in Figure 6. As more computational time becomes available, risk will eventually be dominated by the bias in the stationary distribution. In Figure 6, this can be seen for $\epsilon = 0.1$ which initially reduces risk very fast but is eventually outperformed by the less biased algorithms. However, this crossover does not happen until a large amount of computational time ($\approx$ 8000 secs) has passed.

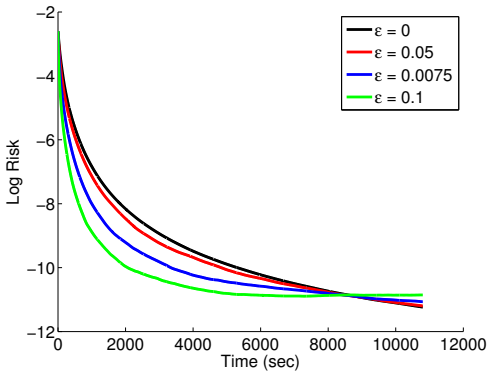

Figure 6: Risk in average prediction of test data for a Logistic Regression model

6 Conclusion

We have shown that our approximate slice sampling algorithm can significantly improve (measured in terms of the risk) the traditional slice sampler of [10] when faced with large datasets. The relative performance gain is a function of the amount of allowed computing time relative to the size of the dataset. For relatively short times (which can still be long in absolute terms for very large datasets) there is more to be gained by using stochastic minibatches because the error due to sampling noise (variance) is expected to dominate the risk. Reversely, for very long simulation times one can draw enough samples to drive the error due to sampling noise to near zero with the standard slice sampler and so it is best to avoid error due to bias.

In our experiments we found that the performance gain for the multinomial regression model was most significant (up to a factor of 7 for one hour of simulation). For L_1 regularized linear regression and logistic regression, the performance gains were more modest. The reason for this discrepancy is that likelihood calculations (which we speed up) comprise a more significant fraction of the total computing time for the multinomial model than for the other two models. We thus expect that for more complex models the gains could potentially be even larger.

Having freed MCMC algorithms from the need to be asymptotically unbiased some intriguing new possibilities arise. First, stochastic minibatch updates should be incorporated into samplers such as Hamiltonian Monte Carlo algorithms [9, 3] and their Riemannian extensions [4] which are the state of the art in the field. But to save even more computation, one could imagine storing likelihood computations which can later be used to predict likelihood values when the MCMC sampler returns to a neighboring region. Proving convergence or controlling the error will be an important challenge.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 1216045 and by the Office of Naval Research/Multidisciplinary University Research Initiative under Grant No. N00014-08-1-1015.

References

[1] S. Ahn, A. Korattikara, and M. Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. In *Proceedings of the International Conference on Machine Learning*, 2012.

[2] K. Bache and M. Lichman. UCI Machine Learning Repository, 2013.

[3] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987.

[4] Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

[5] Anoop Korattikara, Yutian Chen, and Max Welling. Austerity in MCMC land: Cutting the Metropolis-Hastings budget. In *NIPS 2013 Workshop on Probabilistic Models for Big Data*, 2013.

[6] Shiwei Lan, Vassilios Stathopoulos, Babak Shahbaba, and Mark Girolami. Langrangian dynamical Monte Carlo. *arXiv preprint arXiv:1211.3759*, 2012.

[7] Jingjing Lu. *Multivariate slice sampling*. PhD thesis, Drexel University, 2008.

[8] Iain Murray, Ryan Prescott Adams, and David JC MacKay. Elliptical slice sampling. *arXiv preprint arXiv:1001.0175*, 2009.

[9] R Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162, 2011.

[10] Radford M Neal. Slice sampling. *Annals of statistics*, pages 705–741, 2003.

[11] Robert Nishihara, Iain Murray, and Ryan P Adams. Parallel MCMC with generalized elliptical slice sampling. *arXiv preprint arXiv:1210.7477*, 2012.

[12] Madeleine B Thompson and Radford M Neal. Slice sampling with adaptive multivariate steps: The shrinking-rank method. *arXiv preprint arXiv:1011.4722*, 2010.

[13] M. Welling and Y.W. Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the International Conference on Machine Learning*, pages 681–688, 2011.