Fast Mini-Batch MCMC for Big Data Bayesian Posterior Inference

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Markov chain Monte Carlo (MCMC) methods have become one of the most popular techniques in machine learning, but they can be intractably slow when doing Bayesian posterior estimation due to the need to use information from all training data points during each sampling iteration. We propose a mini-batch approach that uses information from a small subset of samples per iteration. This, along with a new Metropolis-Hastings test, will allow us to sample faster to obtain higher-quality parameter estimates. We implement our sampler in BIDMach and benchmark it with the state of the art. Daniel: I hope we can do this. Xinlei: Compare with "cutting MH test budget" paper?

1 Introduction

MCMC has become one of the most popular techniques in machine learning...

In this paper, we resolve these challenges[1] by proposing an MCMC algorithm for Bayesian parameter estimation in which proposals are fast and mini-batch based. Our M-H test acceptance probability is about 50 percent, reasonably high, which means our samples mix well.We experimentally show that our approach is better than the state of the art.

Daniel: we can fill out this section later.

2 Related Work

In this section, we review different results on improving MCMC methods with a focus on those dealing with mini-batches of large datasets.

Cutting the Metropolis-Hastings Budget paper [1] proposed that there is a bias-variance trade-off in standard MCMC methods. Bias occurs when sampling from the wrong distribution, however, bias usually decreases fast. Variance occurs when the number of sampling is not enough since we can collect only a limited number of samples in a given amount of time. The recently proposed Stochastic Gradient Langevin Dynamics (SGLD) [2] and Stochastic Gradient Fisher Scoring methods are biased because they omit the required Metropolis-Hastings tests.

Hamiltonian Monte Carlo (HMC) methods attempt to simulate the "physics" of the system[2] in order to generate distant, high-quality sample proposals (those are the best kind of samples). Since they simulate the physical dynamics of the system, total energy is preserved and their proposals are

[1] Daniel: we can write these "challenges" later once the project becomes clearer.

[2] Intuitively, we can think of a probability distribution as being a distribution over particles in the system, where particles are the values of random variables.

always accepted and there is no need for a Metropolis-Hastings correction/test[3]. Unfortunately, HMC methods require the use of a gradient computation based on all of the data per iteration. (This means the advantage of HMC methods is that they generate high-quality proposals that mix well; their downside is that they are unsuitable for big data, which might involve Bayesian posterior inference over that data.)

The Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) method [3] tries to compromise by using a *mini-batch version* of HMC, so each iteration, only the mini-batch is used to compute the gradient. SGHMC also includes an extra friction term to counteract the effect of noise on the system (or gradient) that is added due to using only a small subset of data. (The "naive" way of replacing the batch with the mini-batch *without* friction means the system deviates from the target/desired distribution.) [3] assumes the injected noise is Gaussian (we make a similar assumption) and appeals to the Central Limit Theorem and second-order Langevin dynamics[4] to show that SGHMC maintains the desired target distribution as the stationary distribution. Our methods are similar in that we use mini-batches of data, but the difference is that they focus on improving a particular flavor of MCMC, *Hamiltonian* Monte Carlo (which uses Langevin dynamics to add noise), and they also try avoiding MH tests, while we are focused on improving *general* MCMC by means of a *more efficient* MH proposal. These are different ways of solving the same problem (i.e., the problem of using mini-batches of large data for MCMC).

The work of [1] also concerns speeding up MCMC by using mini-batches of data. Their argument is that the standard MCMC algorithm for Bayesian posterior inference (i.e., to compute $p(\theta \mid x_1, \dots, x_N)$ for i.i.d. data points) is unsuitable for large datasets because the likelihood computation requires the use of all points, $p(x_1, \dots, x_n \mid \theta) = \prod_{i=1}^{N} p(x_i \mid \theta)$. Using all of this data just to decide whether the new sample θ should be accepted or rejected is computationally inefficient. In theory, given infinite time, using all the data points is fine because it results in an unbiased estimate of the target distribution and the variance of the resulting sampled values is low given enough time.[5] In reality, we have limited time. The key argument in [1] is that given a fixed budget, it may be better to use n data points per iteration (where $n \ll N$), which will sacrifice some bias but will perhaps improve variance by allowing us to sample for θ more often, since we will get $\theta_1, \dots, \theta_T$ as opposed to the standard case using all N samples but with only $\theta_1, \dots, \theta_t$ samples ($t \ll T$).

More formally, the algorithm they develop is a sequential hypothesis testing algorithm. During each iteration, their algorithm generates a new sample θ' from the proposal distribution. They start with a small mini-batch of the data, and test the hypothesis (based on this small mini-batch) that θ' should be accepted or rejected based the likelihood ratio. (Intuitively, we can easily tell if θ' will be accepted/rejected if the mini-batch of data results in likelihood drastically different from the likelihood of the current sample θ.) If the hypothesis test is too uncertain, then we increase the mini-batch size and apply the test again, and we repeat this increment until we accept or reject θ' with high confidence (based on a Student-t distribution). The downside of this algorithm is the need to keep incrementing the mini-batch size in one iteration; in the worst case, they may not be able to tell with confidence to accept or reject until we get nearly all N of the data points, which defeats the purpose of the algorithm. We propose our own algorithm that avoids this problem. Daniel: does this make sense as a "weakness" of their approach? Ideally I would like to have a comparison between our method and theirs.

[3] I'm not sure why this follows but I haven't read the details.

[4] There is a class of MCMC method techniques called *Langevin dynamics* [4]. Those algorithms involve computing a gradient update for the parameter of interest θ and then using the updated θ as the next sample (i.e., this is like the proposal test). Moreover, by simulating the Hamiltonian dynamics of the system, there is no need for a rejection test. To converge to a *full (posterior) distribution* of θ and not a single point (i.e., the MAP estimate of θ, or the mode of the distribution) langevin dynamics adds Gaussian noise each iteration to cause the θ samples to "jump around." The work of [2] is closer to ours because it applies the normal Langevin dynamics algorithm with one difference: the gradient updates are based on a *mini-batch* of the original data.

[5] Note that [1] primarily use MCMC to estimate just the expectation of the target distribution, but their analysis should extend to other values of the target distribution for arbitrary input θ. In other words, they use MCMC to estimate $\mathbb{E}[p(\theta \mid x_1, \dots, x_N)]$, but we are mostly interested in $p(\theta \mid x_1, \dots, x_N)$.

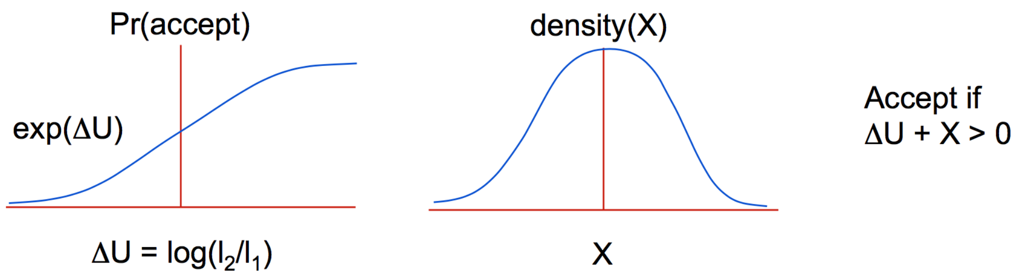

Figure 1: To the left is the logistic distribution, representing a possible acceptance test. As ΔU approaches infinity, the acceptance rate approaches 1. To the right, we have a new variable X and its density, along with an acceptance test $\Delta U + X$, which we describe in detail in Section 3 (note that we use Δ instead of ΔU for simplicity). Daniel: I think that the diagram to the left should use the logistic function, i.e., replace "$\exp(\Delta U)$" with "$(1 + \exp(\Delta U))^{-1}$."

3 Our Algorithm

3.1 The General MCMC Method

The general MCMC method proceeds as follows. For a desired random vector θ, we have a *target distribution* we wish to compute, $p(\theta \mid x_1, \ldots, x_N)$, based on N i.i.d. data points. Since exact evaluation is intractable, we generate a chain of correlated samples $\theta_1, \ldots, \theta_T$ for some large T, and approximate p by using counts of samples. For each iteration t, we start with our current θ_t. We use a *proposal distribution* $q(\theta' \mid \theta_t)$ to determine a new candidate θ'. With probability P_a, we accept it and set $\theta_{t+1} = \theta'$; otherwise, we repeat the previous value $\theta_{t+1} = \theta_t$. Traditionally, P_a is computed as follows:

$$P_a = \min\left\{1, \frac{f(\theta')q(\theta_t \mid \theta')}{f(\theta_t)q(\theta' \mid \theta_t)}\right\} = \min\left\{1, \frac{p(\theta')\prod_{i=1}^N p(x_i;\theta')q(\theta_t \mid \theta')}{p(\theta_t)\prod_{i=1}^N p(x_i;\theta_t)q(\theta' \mid \theta_t)}\right\}, \tag{1}$$

where $f(\theta_t) = p(\theta_t | x_1, \ldots, x_N) \propto p(\theta_t)\prod_{i=1}^N p(x_i \mid \theta_t)$ which roughly tells us how "likely" we are to have a θ_t sample. The normalizing constants cancel out, so the requirement for P_a is that f be proportional to the true target distribution.

One then draws a uniform random variable u from $\mathrm{Unif}[0,1]$ and accepts $\theta_{t+1} = \theta'$ if $u < P_a$, and uses $\theta_{t+1} = \theta_t$ otherwise. The P_a from Equation 1 satisfies "detailed balance" which, roughly speaking, means that if we sample long enough using the heuristic above, we will arrive at a stationary distribution.

The P_a is guaranteed to converge to the true p distribution given sufficiently many examples (though a burn-in period and/or taking every nth sample may be necessary). Unfortunately, computing f requires the use of all N training data points. In this paper, we derive a new fast, approximate Metropolis-Hastings test which does not use all N points during each test.

3.2 A Different Metropolis-Hastings Test

Our starting point is the following Lemma.

Lemma 1. *Let* $\Delta = \log\left(\frac{f(\theta')q(\theta_t|\theta')}{f(\theta_t)q(\theta'|\theta_t)}\right)$, *where f is proportional to the desired target distribution and q is our chosen proposal distribution. Any acceptance function g such that* $g(\Delta) = \exp(\Delta)g(-\Delta)$ *satisfies detailed balance. That is,* $f(\theta_t)p(\theta' \mid \theta_t) = f(\theta')p(\theta_t \mid \theta')$, *where* $p(\theta_y \mid \theta_x)$ *is the probability of jumping from* θ_x *to* θ_y *in our chain.*

Proof. We begin by deriving $p(\theta' \mid \theta_t)$. This is equivalent to the probability of proposing θ' and then accepting it, so

$$p(\theta' \mid \theta_t) = q(\theta' \mid \theta_t)g(\Delta). \tag{2}$$

Similarly, we have

$$p(\theta_t \mid \theta') = q(\theta_t \mid \theta')g(-\Delta). \tag{3}$$

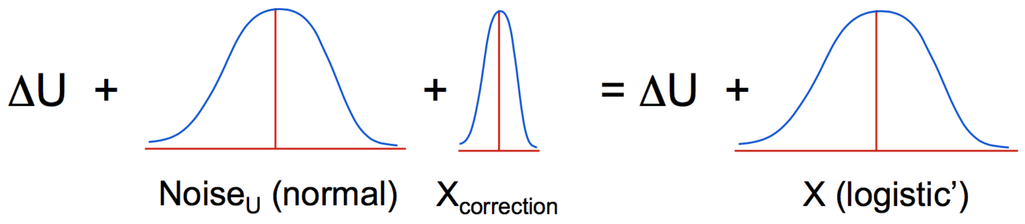

Figure 2: To the left, we have ΔU (which is simplified as Δ in Section 3), along with two noise terms that come from the new variable we define X. The details are in Section 3.

Notice that the probability of accepting a transition from θ' to θ_t is $g(-\Delta)$ because this inverts the fraction inside the logarithm term of Δ. By assumption, we can expand $g(\Delta) = \exp(\Delta)g(-\Delta)$ in Equation 2. Doing this, and combining the result of Equation 3, we get

$$g(-\Delta) = \frac{p(\theta' \mid \theta_t)}{q(\theta' \mid \theta_t)\exp(\Delta)} = \frac{p(\theta_t \mid \theta')}{q(\theta_t \mid \theta')}. \tag{4}$$

Rearranging terms and expanding $\exp(\Delta)$, we have

$$\frac{p(\theta' \mid \theta_t)f(\theta_t)q(\theta' \mid \theta_t)}{q(\theta' \mid \theta_t)f(\theta')q(\theta_t \mid \theta')} = \frac{p(\theta_t \mid \theta')}{q(\theta_t \mid \theta')}. \tag{5}$$

Cancellations result in $f(\theta')p(\theta_t \mid \theta') = f(\theta_t)p(\theta' \mid \theta_t)$. Thus, detailed balance is satisfied.

$\square$

Some straightforward arithmetic shows that the standard Metropolis-Hastings acceptance function $g(\Delta) = \min\{1, e^{\Delta}\} = \min\left\{1, \frac{f(\theta')q(\theta_t|\theta')}{f(\theta_t)q(\theta'|\theta_t)}\right\}$ satisfies the condition $g(\Delta) = \exp(\Delta)g(-\Delta)$ and consequently, results in detailed balance.

For reasons that will be clear later, we choose an alternative g, the logistic function: $g(\Delta) = (1+\exp(-\Delta))^{-1}$. Again, straightforward arithmetic shows that it satisfies the condition in Lemma 1.

We can sample from the logistic function using the following procedure. Let u be drawn from $\text{Unif}[0,1]$. At any given iteration, we can compute Δ, and we accept the new candidate θ' if $g(\Delta) > u$, and reject otherwise. Let us define the random variable $X = g^{-1}(u)$ where again, g is the logistic function. We claim that X has CDF function $F_X(x) = g(x)$, i.e., its CDF is precisely the logistic function. This is because for some $X = x$, we have

$$F_X(x) = \Pr(X \le x) = \Pr(g^{-1}(u) \le x) = \Pr(u \le g(x)) = \int_0^{g(x)} 1dx = g(x),$$

as the density of the uniform random variable here is simply one. Thus, the criteria to accept the candidate θ' is equivalent to whether $\Delta > X$, where X has CDF of logistic function. (If this isn't immediately obvious, note that we can pre-multiply g^{-1} to $g(\Delta) \ge u$, and our test is $\Delta > X$.) Since X is symmetric about zero, the acceptance criteria can also be expressed as $\Delta + X > 0$, as shown in Figure 1.

Daniel: The above makes sense, but I still don't understand why we had to show that the CDF of X is the logistic function, because I never use that when I claim that our acceptance test is now $\Delta > X$. Do we need that assumption so that, for instance, we can decompose X later the way we do it, with X_{noise} having the same variance as ϵ? That might explain it.

Unfortunately, there's a problem with the $\Delta + X > 0$ test: it requires us to compute Δ, so our test will be slow! We can get around this by defining a new scalar-valued term, Δ', which is computed the same way as Δ, but uses far fewer samples. In other words, Δ' is computed based on a *mini-batch* of data, and it is an approximation of Δ, so we can express the relationship as $\Delta' = \Delta + \epsilon$ for a noise term ϵ. *The key is that ϵ follows a normal distribution*, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, because for i.i.d. data $x_1, \dots, x_N$, the Δ is expressed as the sum of log probabilities (look at Equation 1 and apply the logarithm to the product over the N data points, to get a sum of log terms). All we do with Δ is take far fewer of the summations, and normally-distributed noise is there as a product of the Central Limit Theorem.

To sample accurately from Δ', we need to slightly change our acceptance criteria. We can decompose X as $X = X_{\text{norm}} + X_{\text{correction}}$, where the first term is a "noise term" which has the same variance as ϵ, and the second term is a "residual" term. The criteria to accept, previously expressed as $\Delta + X > 0$, can be rewritten as

$$\Delta + X = \Delta + X_{\text{norm}} + X_{\text{correction}} \approx \Delta' + X_{\text{correction}} > 0. \tag{6}$$

This explains Figure 2 (though it really should be an approximation as we can't guarantee that our X_{norm} term is exactly the correct amount of noise), where the left hand side has distributions representing normal noise (X_{norm}, which is labeled as Noise_U, but they are the same thing) and a "correction" term labeled $X_{\text{correction}}$. Be aware that all of these "additions" are to be interpreted as convolutions when computing the sum of densities.

Notice that we have several options we can control. For instance, we can adjust the mini-batch size; increasing the mini-batch size will cause the X_{noise} distribution to shrink and become narrower.

Daniel: I think it's becoming clear but I'm still trying to connect all of the steps. During each iteration of MCMC, we compute Δ' and *sample* (right?) an X (which we can do by sampling u and then doing $X = g^{-1}(u)$. This gives us $\Delta' + X$ but our MH test actually requires $X_{\text{correction}}$, so first we must estimate σ^2, the variance of $X_{\text{noise}} \sim \mathcal{N}(0, \sigma^2)$ (perhaps from the mini-batch data, somehow?) and then somehow "remove" that component from X to get $X_{\text{correction}}$? Then we have our acceptance test. Is this right? Or, of course, we don't have to sample X at all, but just figure out a way to sample $X_{\text{correction}}$ directly.

I think we talked about estimating σ^2. The variance σ^2 that we're concerned about is a scalar value based on the sum of log probabilities of different samples, so one way we could estimate the variance is by taking all samples of a mini-batch and seeing how much those individual $\log p(x_i \mid \theta)$ vary. Or we could look at k mini-batches of data which will give us $\Delta_1, \ldots, \Delta_k$ and compute the empirical σ^2 (of course that defeats the purpose of using mini-batches).

Xinlei: I agree with you that out MH test requires $X_{correction}$. My understanding is that we can actually estimate X from the whole data set, and the probability distribution of X_{norm} is what we proposed? So the determination of $X_{correction}$ is a deterministic deconvolution problem.

4 Example

Daniel: I made this to try and connect John's code with what we have earlier.

Let's do an example based on John's MATLAB code. We start with the following:

```
n = 1000;
dist = 10;
ascale = 2;
iclip = 20;
x = [(0:n), ((-n):-1)]/n*dist;
ldist = ascale./(2+exp(ascale*x)+exp(-ascale*x));
ndist = normpdf(x, 0, 1/ascale);
```

What do these mean? The easiest array is `x`; that's just an array of values bounded by -10 and 10 (with our settings above) with elements spaced out by 0.01, though oddly enough, it goes from 0 to 10, then from -10 to -0.01. I don't know why. The more interesting arrays are `ldist` and `ndist`, and Figure 3 shows plots of those arrays. Note that the `ndist` curve has the taller tails (or "center" if we were doing this from -10 to 10 in order). Intuitively, connecting this with our previous notation, we can say that `ndist` is like X_{norm} and `ldist` is Δ'. The reason is that `ldist` is "almost Gaussian" or Gaussian (but with unknown parameters, though we could estimate them) and it looks like we have a known formula for computing it. In this code, it's written as it is above, and in the "real" case, it is how we compute Δ'.

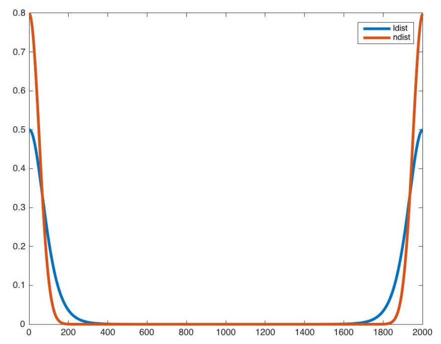

Figure 3: Plots of `ldist` and `ndist`.

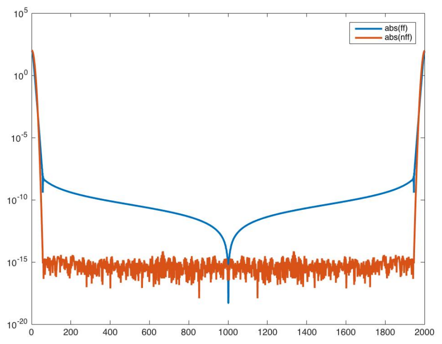

Figure 4: Plots of FFTs.

Therefore, **the goal is to recover something like the $X_{\text{correction}}$, i.e., what is the missing piece K such that** $L = N + K$, where L and N are random variables from the densities of `ldist` and `ndist`, respectively. From now on, let's just call K as $X_{\text{correction}}$ to connect it with our previous notation.

Then we do a Fast Fourier Transform on both and then we plot the absolute value of those two vectors in Figure 4 using MATLAB's `semilogy` plot, so this is on a log scale (as evident by the near-zero values we're seeing).

```
ff = real(fft(ldist));
nff = real(fft(ndist));
```

The next part is the most interesting thing, and it arises out of the following theorem. Suppose we know that random variables N and $X_{\text{correction}}$ are independent (where, again, N is distributed with a PDF represented by `ndist`. Also suppose that we have L such that it is the convolution (we made this assumption earlier, I'm just making it clear):

$$f_L(x) = f_N(x) * f_{X_{\text{correction}}}(x), \tag{7}$$

where $f_Y(x)$ is the probability density function for random variable Y.

The theorem states that we can apply the Fourier transform to each component, and amazingly, *this results in the product of the Fourier transforms!!!* Let $\hat{f}$ be the Fourier transform of a function f. We have

$$\hat{f}_L(x) = \hat{f}_N(x)\hat{f}_{X_{\text{correction}}}(x), \tag{8}$$

What this means is that we can find the Fourier transform of the density of $X_{\text{correction}}$ by simple (point-wise) division! We see this in the following code below, where `fdiv` is $\hat{f}_{X_{\text{correction}}}(x)$:

```
fdiv = ff./nff;
fdiv(iclip:(length(fdiv)-iclip+2)) = 0;
u = ifft(fdiv);
figure(4);plot(u, 'Linewidth', 2); legend('u');
v = ifft(fdiv.*nff);
figure(5);
semilogy(1:length(x), ldist, 1:length(x), v, 'Linewidth', 2);
legend('ldist', 'v');
```

Note that once we get `fdiv`, we make certain components zero, because (I assume?) we don't want negative numbers. Then, *we apply the inverse Fourier transform*! Why? Because applying that to our estimate of $\hat{f}_{X_{\text{correction}}}(x)$ will clearly get us an estimate of what we wanted all along, $f_{X_{\text{correction}}}(x)$. Our estimate of that is in Figure 5. Daniel: is this is an *un-normalized* estimate? The values seem really small ... or maybe I don't have the correct intuition on convolutions.

Later, I assume as a sanity check, we try to recover `ldist` as well, and the result is in Figure 6. Notice the log scale!

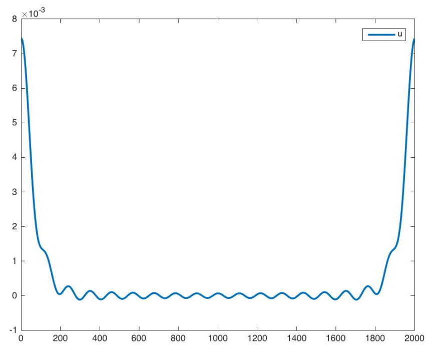

Figure 5: The recovered "u" distribution.

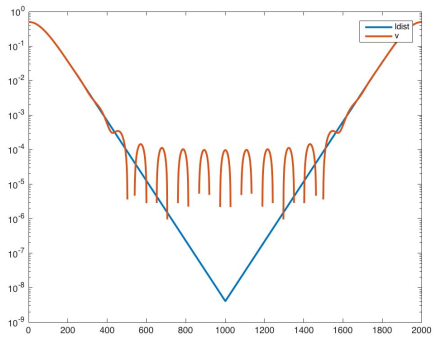

Figure 6: A sanity check for `ldist`?

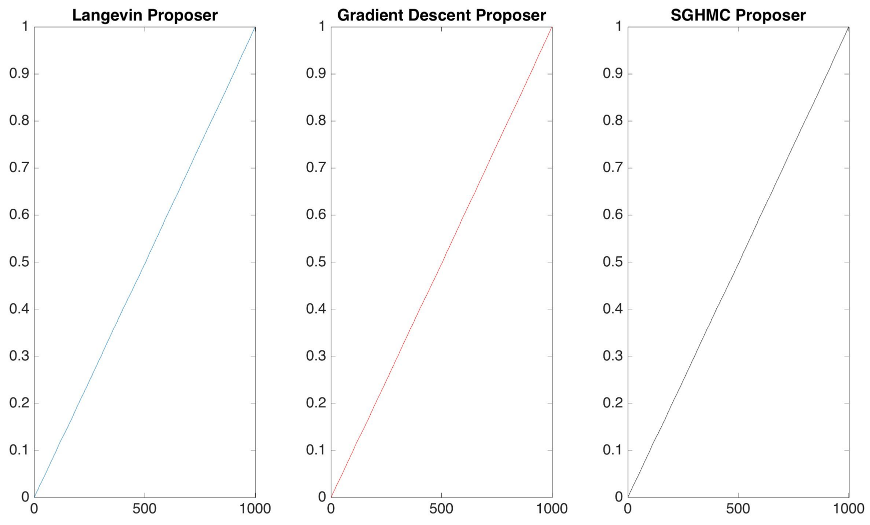

Figure 7: ROC curve with different proposers

Daniel: I hope I am understanding this right. Is Figure 6 just here as a sanity check? Also, I am just wondering why `x` goes from 0 to 10 and -10 to -0.01, instead of just -10 to 10? I tried it with -10 and 10 and got something that seemed similar except with a bunch of stuff "flipped around" so it seems like no big deal.

5 Implementation

We implement our system within the open-source BIDMach project [5]. Note to ourselves: do NOT modify the `Learner.scala` code – just make it a new "Model".

References

[1] A. K. Balan, Y. Chen, and M. Welling, "Austerity in MCMC land: Cutting the metropolis-hastings budget," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

[2] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 681–688.

[3] T. Chen, E. Fox, and C. Guestrin, "Stochastic gradient Hamiltonian Monte Carlo," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, June 2014.

[4] R. M. Neal, “MCMC using Hamiltonian dynamics,” *Handbook of Markov Chain Monte Carlo*, vol. 54, pp. 113–162, 2010.

[5] J. Canny and H. Zhao, “Bidmach: Large-scale learning with zero memory allocation,” 2013.

A Section for Appendix Information

Daniel: In case we want to use this.