Firefly Monte Carlo: Exact MC with data subsets

Feynman Liang, Max Chamberlin, Kai Xu

April 30, 2016

Contents

1 Introduction

Many methods in Bayesian inference involve computing the likelihood $L(\theta)$ of parameters θ over iid observations $X = \{x_n\}_{n=1}^N$, defined as

$$L(\theta) = P(X|\theta) = \prod_{n=1}^{N} P(x_n|\theta) = \prod_{n=1}^{N} L_n(\theta)$$

For example, Monte Carlo Markov Chain (MCMC) are a class of methods often used to sample posterior distributions $P(\theta|X)$ with intractable partition functions $Z = \int P(X|\theta)P(\theta)d\theta$. When performing Metropolis-Hastings MCMC, at each iteration we need to compute the acceptance probability

$$A(\theta'|\theta) = \min\left\{1, \frac{q(\theta|\theta')L(\theta')P(\theta')}{q(\theta'|\theta)L(\theta)P(\theta)}\right\}$$

This means that each iteration requires at least one evaluation $L(\theta')$ of the likelihood over all N points ($L(\theta)$ can be cached from the prior iteration). Unfortunately, in the era of "Big Data" N may be on the order of petabytes and computing $L(\theta)$ at each iteration may be prohibitively expensive.

Many methods have been proposed to tractably estimate **??**. Firefly MC (FlyMC) belongs to a class of methods which approximate $L(\theta)$ using smaller tractable subsets of the total dataset. In FlyMC, auxilliary "brightness" variables are first introduced to decrease the number of likelihood evaluations per iteration. The Markov chain state and auxilliary variables are then successively Gibbs sampled. For certain inference problems, FlyMC can provide a significant 3-4$\times$ speedup over traditional MCMC even after accounting for differences in expected sample size.

1.1 Rationale for selecting the paper

We chose the Firefly MC paper [**?**] because of the ubiquity of MCMC sampling methods which see use not just in the fields of machine learning and Bayesian statistics, but also in the fields of computational physics, biology, and computational linguistics. Indeed, MCMC can be used a universal inference method that is agnostic to the underlying likelihood function and prior distribution. As a result, it already forms the basis of many state-of-the-art probabilistic programming frameworks [**?**].

Furthermmore, Firefly MC addresses the issue of tractable computations at scale. With increasingly larger dataset sizes across all research and industry domains, scalability has become an increasingly central concern and proven tools such as MCMC have recently begun to receive much attention [**?**].

2 Theory

Firefly Monte Carly (FlyMC) is a method which addressses the computational problem of evaluating $L(\theta)$ for a large number of data points N by using exact Monte Carlo estimates of $L(\theta)$ computed over smaller, tractable subsets. It works by augmenting the joint distribution with hidden *brightness* variables $Z = \{z_n \in \{0,1\}\}_{n=1}^N$to form a *complete data likelihood*

$$P(X, Z|\theta) = \prod_{n=1}^{N} P(x_n|\theta)P(z_n|x_n, \theta)$$

Like other auxilliary variable methods (e.g. Hamiltonian MC [?]), we can recover the desired likelihoods simply through marginalization

$$\sum_Z P(X,Z|\theta) = \sum_{z_1} \cdots \sum_{z_N} \prod_{n=1}^{N} P(|x_n|\theta)P(z_n|x_n,\theta) = \prod_{n=1}^{N} P(|x_n|\theta) \sum_{z_n} P(z_n|x_n,\theta) = L(\theta)$$

In FlyMC, the brightness variables are chosen to have Bernoulli distribution

$$P(z_n|x_n,\theta) = \left(\frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)}\right)^{z_n} \left(\frac{B_n(\theta)}{L_n(\theta)}\right)^{1-z_n} \tag{1}$$

where$B_n(\theta)$ is a lower bound on the likelihood satisfying

$$0 \leq B_n(\theta) \leq L_n(\theta) \tag{2}$$

$$\prod_{n \in A} B_n(\theta) \text{ can be efficiently computed for any data subset } A \subset X \tag{3}$$

?? is required for **??** to be a valid probability distribution. To see why **??** is necessary, consider computing the complete data likelihood

$$\begin{aligned} P(X,Z|\theta) &= \prod_{n=1}^{N} P(x_n|\theta)P(z_n|x_n,\theta) \\ &= \prod_{n=1}^{N} L_n(\theta) \left(\frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)}\right)^{z_n} \left(\frac{B_n(\theta)}{L_n(\theta)}\right)^{1-z_n} \\ &= \prod_{x_n \in X_{bright}} (L_n(\theta) - B_n(\theta)) \prod_{x_n \in X_{dim}} (B_n(\theta)) \\ &= \prod_{n=1}^{N} B_n(\theta) \prod_{x_n \in X_{bright}} \frac{L_n(\theta) - B_n(\theta)}{B_n(\theta)} \end{aligned} \tag{4}$$

where we have defined the bright points $X_{bright} = \{x_n : z_n = 1\}$ and dim points $X_{dim} = \{x_n : z_n = 0\}$. As a consequence of our choice for **??**, **??** decomposes into two products where $\frac{L_n(\theta)-B_n(\theta)}{B_n(\theta)}$ only needs to be evaluated for $x_n \in X_{bright}$. For illustration, we plot the bright/dim decomposition of $L_n(\theta)$ in **??**, as well as show an example (θ, z_n) trajectory.

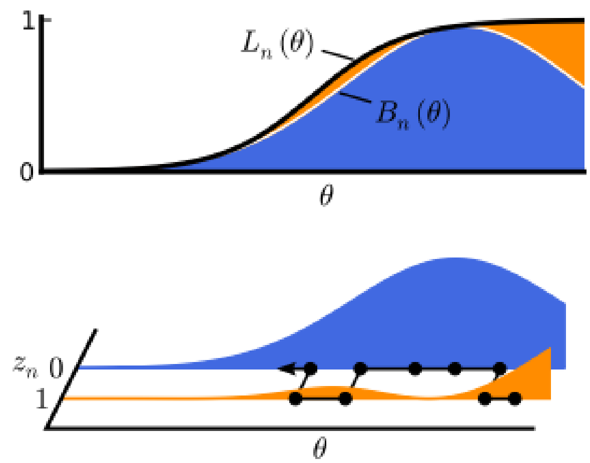


Figure 1: Graphical representation of the decomposition of L_n into a $L_n - B_n$ part associated with bright variables $z_n = 0$ and a B_n part associated with dim variables $z_n = 0$ [?]

?? also illustrates the payoff from assuming ??: by assuming $\prod_{n=1}^N B_n(\theta)$ term in ?? is efficient, the overall computational complexity of evaluating the complete likelihood $P(X,Z|\theta)$ is determined by $\#X_{bright}$.

2.1 Markov chain implementation

Samples $(\theta, \vec{z}) \sim (P(\theta|X), P(Z|X,\theta))$ can be generated using Gibbs sampling because of iid assumptions amongst the X and lack of interdependence amongst the Z. Given current $\theta^{(t)}$ and $\vec{z}^{(t)}$, we first fix the parameters θ and sample the brightness variables z_n with Bernoulli probability $\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}$. Then, the brightnesses z_n are fixed and the Markov chain is stepped using Metropolis-Hastings [?] or more complex steppers such as Metropolis-adjusted Langevin dynamics [?] when the likelihood is differentiable.

The resampling of the brightness variables at each iteration cause each data point to "blink" bright and dim across iterations, inspiring the name "Firefly" given to this method.

2.1.1 Implicit sampling

One possible criticism of the method described above is that direct sampling of z_n from N Bernoulli distributions, each with probability $\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}$, would again require N evaluations of $L_n(\theta)$. This appears to mitigate any savings obtained by FlyMC.

Fortunately, this can be mitigated through implicit sampling. Instead of recomputing Bernoulli probabilities and directly sampling the N z_ns, a second MH-MCMC is used for sampling brightnesses. The motivation is that many $z_i = 0$ ideally, hence it makes sense to only consider proposals where each dim point has a small $q_{d\to b} \in [0,1]$ probability of becoming bright. As computing the acceptance probability requires likelihoods only need to be evaluated for the brightness variables proposed to change state, this could significantly reduce the number of likelihood evaluations if few brightness variables change state across successive Gibbs iterations.

2.2 Theoretical performance analysis

FlyMC's speedup over a traditional $O(N)$ likelihood evaluation is $\frac{N}{\#X_{bright}}$ per iteration. However, since the brightness variables $z_n$are themselves, to understand the performance improvements we consider the expected number of bright points $E[\#X_{bright}]$.

Note that for any point x_n, ?? shows that the probability x_n is bright is given by $\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}$. Hence, the number of bright points is given by

$$E[\#X_{bright}] = \sum_{n=1}^N E[z_n] = \sum_{n=1}^N \int P(\theta|X) \frac{L_n(\theta) - B_n(\theta)}{L_n(\theta)} d\theta \tag{5}$$

This says that the number of bright points (and hence the speedup achieved by FlyMC) is related to the gap between the true likelihood $L_n(\theta)$ and the lower bound $B_n(\theta)$, weighted by the posterior $P(\theta|X)$.

This result makes intuitive sense. Consider the extreme case where for all n: $B_n(\theta) = 0$. Then $P(z_n = 1|x_n, \theta) = 1$ so every data point is bright (i.e. $X = X_{bright}$) and ?? reduces to evaluating the full dataset likelihood $\prod_{n=1}^N L_n(\theta)$ as usual.

In the other extreme, if $B_n(\theta) = L_n(\theta)$ so all data points are dim. At first it appears that FlyMC counterintuitively offers a $\frac{N}{\#X_{bright}} = \frac{N}{0} = \infty$speedup. However, considering ?? reveals that what is computed is $\prod_{n=1}^N B_n(\theta) = \prod_{n=1}^N L_n(\theta)$, which by assumption ??can be computed efficiently. But then computing $L(\theta) = \prod_{n=1}^N L_n(\theta)$ is efficient and hence is no longer $O(N)$, contradicting our earlier assumptions.

2.3 Discussion on the lower bound $B_n(\theta)$

The integral role played by the lower bound $B_n(\theta)$ makes it an important topic of investigation. In this section, we investigate how reasonable and justified are the assumptions required for developing FlyMC's theory.

One of our assumptions (??) was that computation of $\prod_{n=1}^N B_n(\theta)$ be efficient new θ. For example, choosing $B_n(\theta)$ to be exponential family will make computing $\prod_{n=1}^N B_n(\theta)$ for each new θ be $O(1)$ because

$$B_n(\theta) \propto \exp(\langle \eta_n, T(\theta) \rangle)$$
$$\prod_{n=1}^N B_n(\theta) \propto \exp(\langle \sum_{n=1}^N \eta_n, T(\theta) \rangle)$$

and $\sum_{n=1}^{N}\eta_n$ can be evaluated once and cached. We believe that this assumption is quite restrictive and have had some difficulty coming up with functional forms which are not exponential family that satisfy this assumption.

The other assumption (**??**) required $B_n(\theta)$ be a non-negative lower bound for $L_n(\theta)$. This is trivially satisfied by setting $B_n(\theta) = 0$, but to be practical we should try to find a lower bound with the smallest gap possible. Several forms of bounds exist for commonly used likelihoods; the original authors consider the Jaakola and Jordan bound on the log logistic function. However, this requirement is quite restrictive and unrealistic for more complex use cases. For example, if $L_n(\theta)$ is given by some complicated Bayes net or neural network, then a valid $B_n(\theta)$ for use with FlyMC is not immediately obvious.

2.3.1 MAP-tuning the lower bound

In certain cases (e.g. Jaakola and Jordan bound on log logistic), B_n is part of a parametric family of lower bounds $\{B_n(\cdot,\xi)\}_\xi$where ξ is a free parameter. Since the number of bright points determines the speedup offered by FlyMC, **??** suggests that we should choose ξ to minimize $E\left[\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}\right]$ for each datapoint n where the expectation is taken over the posterior distribution $P(\theta|X)$.

The authors of FlyMC propose a *MAP-tuned FlyMC* variant where the posterior expectation $E\left[\frac{L_n(\theta)-B_n(\theta)}{L_n(\theta)}\right]$ is replaced with the MAP point estimate $\frac{L_n(\theta_{MAP})-B_n(\theta_{MAP})}{L_n(\theta_{MAP})}$ and$\xi = \text{argmin}_\xi L_n(\theta_{MAP}) - B_n(\theta_{MAP}).\theta_{MAP} = \text{argmax}_\theta P(\theta|X)$ can be found using stochastic gradient descent to handle large data-set concerns.

Our opinions about this method are mixed. While MAP tuning should provide a tighter lower bound than no tuning, approximating the posterior with a delta distribution centered at θ_{MAP}seems suspicious. Other methods which could be considered here include first generating some posterior samples using an untuned chain or MCMC on a subset of the data, then tuning ξover the posterior samples.

3 Implementation

FlyMC augments traditional MCMC by introducing brightness variables Z which are Gibbs sampled along with the original chain's state θ. At a high level, the algorithm consists of the following steps:

1. Initialize $\theta^{(0)}$
2. For $t = 1, \cdots, N_{iter}$
   (a) For $n = 1, \cdots, N$: sample $z_n^{(t)} \sim \text{Bernoulli}\left(\frac{L_n(\theta^{(t)})-B_n(\theta^{(t)})}{L_n(\theta^{(t)})}\right)$ with $\theta = \theta^{(t)}$fixed
3. Propose $\theta^{(t+1)} \sim q(\theta^{(t+1)}|\theta^{(t)})$
4. Compute acceptance probability $A(\theta^{(t+1)}|\theta^{(t)}) = \min\left\{1, \frac{q(\theta^{(t)}|\theta^{(t+1)})P(X,Z|\theta^{(t+1)})P(\theta^{(t+1)})}{q(\theta^{(t+1)}|\theta^{(t)})P(X,Z|\theta^{(t)})P(\theta^{(t)})}\right\}$
5. Repeat from step 2 until chain has mixed, yielding a single sample $z_n^{(\infty)}$

In step 4, the complete data likelihood is computed using **??** and hence only requires evaluating the likelihood over X_{bright}.

3.1 Provided reference software

Reference code is provided by the authors[1], which provides implementations FlyMC for logistic regression using MH-MCMC, softmax using Langevin dynamics, and sparse linear regression using slice sampling. The reference implementation includes an implicit sampling scheme for sampling brightness variables z_i.

3.2 Our contributions to the project

However, the only full example provided is a logistic regression example on toy data and is insufficient for reproducing the results reported in the paper. We forked he authors' code [2] and added our own modifications, most importantly:

1. Support for MAP-tuning of the lower bounds, using scipy.optimize for MAP estimation
2. Implementation of regular full-dataset MCMC, which is achieved by fixing $\forall i : z_i = 1$

[1]https://github.com/HIPS/firefly-monte-carlo
[2]https://github.com/feynmanliang/firefly-monte-carlo

3. Improved numerical overflow handling in `LogisticModel`
4. Instrumentation for counting log-likelihood evaluations and outputting traces of chain state and log likelihood per iteration
5. Integration, parameter tuning, and reproduction of MNIST Logistic Regression, CIFAR-10 Softmax Classification, and Robust Linear Regression experimental results

We also fixed a significant bug in some caching code provided with the reference implementation and submitted our bugfix back upstream [3]. The problem was that computations of $\left\{ \frac{L_n(\theta)-B_n(\theta)}{B_n(\theta)} : n \in idx \right\}$were being cached for vectors θ and idx, but that caching was performed over the entire idx vector rather than elementwise. For example, if $idx2 \subsetneq idx$, then even though $\left\{ \frac{L_n(\theta)-B_n(\theta)}{B_n(\theta)} : n \in idx2 \right\} \subset \left\{ \frac{L_n(\theta)-B_n(\theta)}{B_n(\theta)} : n \in idx \right\}$ the reference implementation would still result in a cache miss because $idx2 \neq idx$. This bug causes the number of likelihood evaluations to be twice the number of proposed bright points $\#X_{bright}$at each iteration, which means that FlyMC will only offer speedups if $\#X_{bright} < \frac{N}{2}$.

4 Reproduction experiments

The Experiments to be replicated:

In this practical, we conducted the following three experiments:

- Logistic regression: For this particular task, we tested we tested FlyMC's preformance for the task of classifying 7s and 9s on MNIST, with prior $P(\theta) \sim \mathcal{N}(0, I)$.
- Softmax classification: For this task, we tested FlyMC's ability to fit a softmax classification model on three CIFAR-10 classes (airplane, automobile and bird) using Langevin dynamics
- Robust linear regression: For logistic regression task, we generated synthetic data and evaluated the performance of FlyMC in comparison to regular MCMC.

The first two of these three experiments were data-sets used in the paper. However, for the third data-set we decided it would be intersting to see if the same results claimed by a paper could be found in a data-set that was not hand-picked by the authors.

Results

The autocorrelation plots from the three different data-sets show the same general trends. In general, the autocorrelation curve for MCMC is below that of FlyMC and FlyMap. These results demonstrate that the different firefly Monte-Carlo methods tends to produce samples θ with a greater degree of dependence than typical MCMC methods. This can be explained by the following fact. There are fewer likelihood evaluations computed for data-points at each step of FlyMC. Since the updates for the posterior $p(\theta \mid \theta)$ depend on the sum of these updates, at any given step. TODO: don't undertsnad

A further trend to be observed is that, across all data-sets, the negative log posterior for MCMC tends to be greater than the negative log posterior for both variants of firefly Monte Carlo. This is unsurprising because at any given iteration the MCMC sampler will have seen a great many more data-points. As such, it will be more likely that a sampled θ had settled to a value that is a good model paramater for the data.

The number of log-likelihood evaluations ata given iteration is constant under the MH-MCMC sampler for both soft-max classification and logistic regression. This is to be expected since likelihood updates are computed for all the data-points under vanilla MCMC. In contrast, the parameters for the regression data are drawn from a slice-sampler [?]. Given this, we see that the number of likelihood evaluation tends to oscillate up and down depending on the weight of its 'effective' step-size.

In all cases the fireFly samplers compute likelihood evaluations that are an order of magnitude smaller than the corresponding MH and slice samplers. For MNIST logistic regression, approximately 1/12 of the likelihoods are computed at each stage under flymap, than are computed under MCMC. For softmax classification, FlyMC evaluates roughly 1/3 of the likelihoods that are evaluated under regular MCMC and FlyMap performed even better evaluating only 1/6 of these likelihoods. This is exactly what we would expect, since likelihoods are only computed for the small number of bright points, the expected number of which is given by the sum: TODO: PUT INFORMULA.

[3]https://github.com/HIPS/firefly-monte-carlo/pull/1

To summarise, we can see exactly the right sort of trends that we would expect to see for data of this sort. MCMC has better mixing (as demonstrated through autocorrelation), and better likelihoods at a given iteration (it has seen more of the data); Though FlyMC and FlyMap show worse mixing, and have lower negative log likelihoods at a given iteration, crucially they use fewer log-likelihood updates at each iteration. These are exactly the same sorts of trends that were observed in the paper by Ryan Adams [?].

Crucially, however, the effectiveness of the firefly MC interpolation depends on the effective sample size for a given number of iterations. If FlyMc is to deliver the improvements in 1000 iteration.

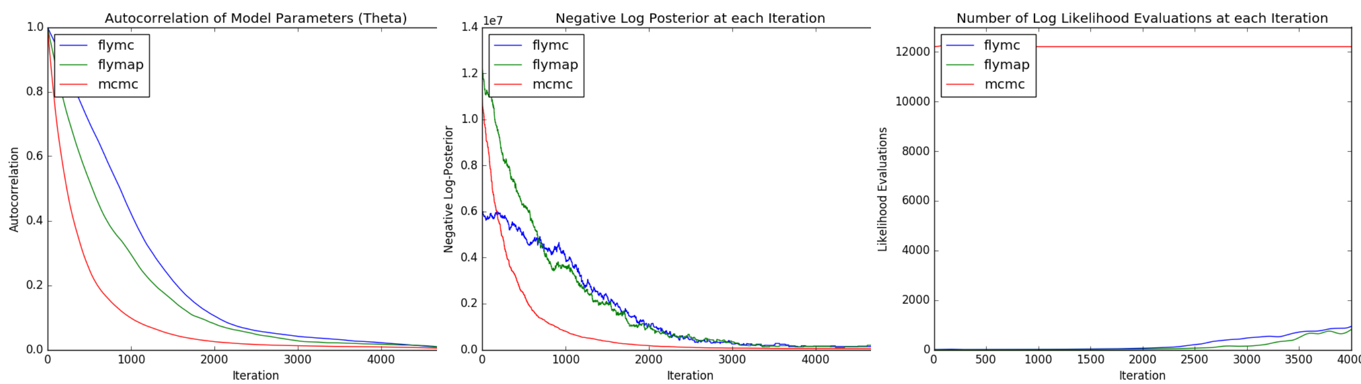
We can calculate a quantity that determines the effective sample rate by choosing a log-likelihod value, L, and computing the number of iterations MCMC, FlyMap and FlyMC took to reach that log likelihood. Suppose these numbers were M, FM and FMC respectively for each of MCMC, FlyMap and FlyMC. Then for FlyMap, the effective sample rate is given by: (FM/M)*average number of likelihood evaluations up until FM. I present our estimates of these evaluations in **??**. The results are based using, the corresponding MCMC sampler.

Following [?], we can quantify slower burn-in **??** shows the effective sample size (ESS) of regular MCMC, untuned MCMC and MAP-tuned FlyMC for the three experiments.

Table 1: ESS of regular MCMC, untuned MCMC and MAP-tuned FlyMC for the three experiments

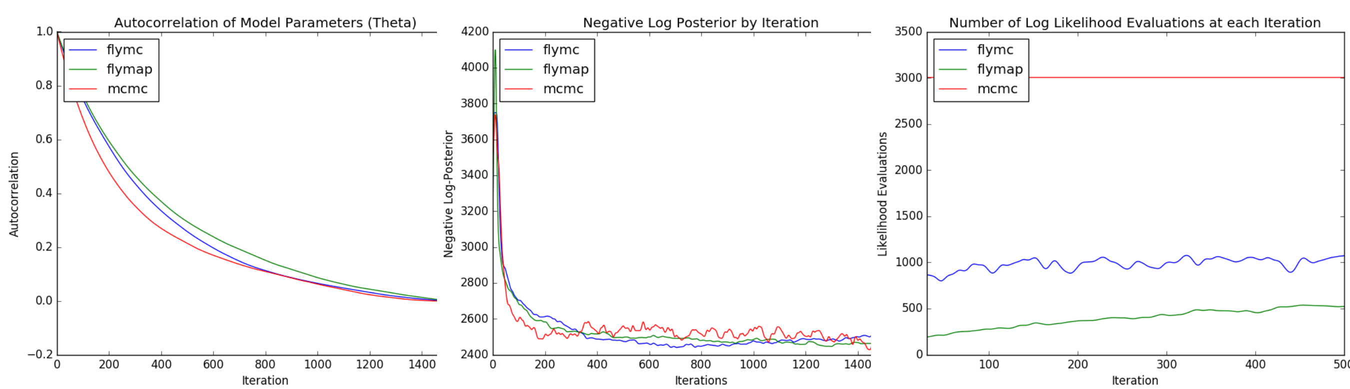
	Regular MCMC	Untuned FlyMC	MAP-tuned FlyMC
Logistic regression	3.6	0.5	0.6
Softmax clssification	7.1	4.0	3.3
Robust regression	1.1	1.0	1.1

With the ESS from the table and the speedup observed from the plots of number of log-likelihood evaluations per iteration, we can conclude that MAP-tuned FlyMC provides a reasonable speedup while the untuned one is slower than the regular one.



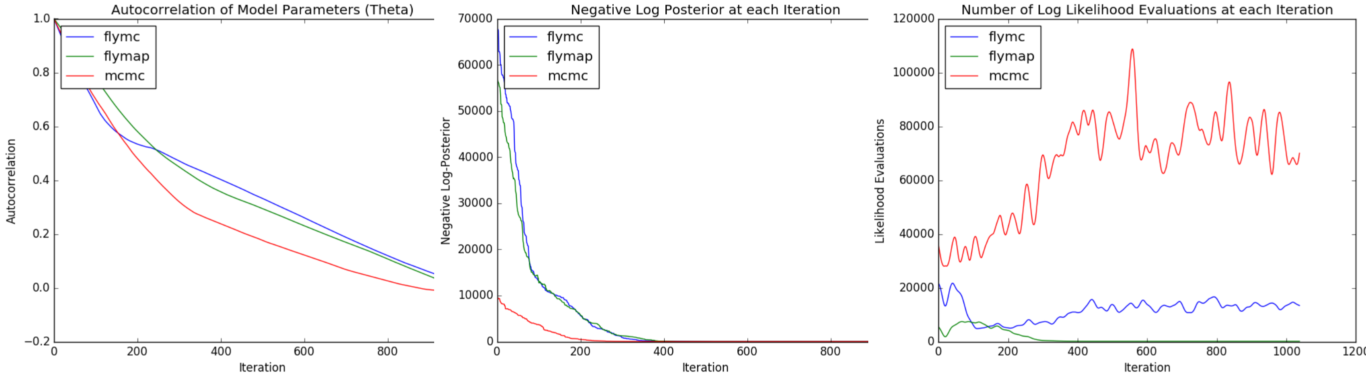
(a) Autocorrelation (b) Negative log posterior (c) Number likelihood evaluations

Figure 2: Logistic regression binary classification on MNIST with MH-MCMC



(a) Autocorrelation (b) Negative log posterior (c) Number likelihood evaluations

Figure 3: Softmax 3-way classification on CIFAR-10 with Langevin dynamics



(a) Autocorrelation (b) Negative log posterior (c) Number likelihood evaluations

Figure 4: Robust regression on synthesized data with slice sampler

Figure 5: Results reported by authors in [?]

5 Conclusion

From this practical, we learned how FlyMC can be used as an elegant framework for approximating full dataset likelihood using subsets and can lead to 2-4× speeedups over conventional MCMC even after accounting for reduced effective sample size. The FlyMC algorith can be further MAP-tuned, yielded the fastest samplers examined .

As a summary of the work undertaken, our practical implementation of Firefly MC led, which was based on the code provided by HIPs, led to several important pull requests on Github which fixed the caching of bright variables. in addition, we had to produce plots, tune parameters, and amend the sampling procedure so that they would work with the various kinds of data that was investigated in the paper.

We replicated almost almost all the graphs and figures in the paper, however, we would have liked to generate more tabulated data on effective sample rates than we could have.

Many extensions of the work we have presented are possible. One particularly important experiment would be to tune the lower bounds over multiple parameter values sampled from the posterior rather than the single MAP estimate. This would ensure a low number of expected bright points, even if we are possibly far away from the MAP estimate. Given the significant performance difference between MAP-tuned and untuned FlyMC, we expect tuning lower bounds in this manner to yield even lower numbers of bright points and hence greater speedups over traditional MCMC.

Another direction for investigation is on the lower bounds B_n. The assumptions made in assumptions 1 and 2 on B_n are required for formulating FlyMC, and the authors do a good first pass discussion about using exponential families for B_n. However, it may be interesting to consider other functional forms for B_n such that $\prod_{n=1}^{N} B_n(\theta)$ is cheap to compute. Furthermore, it is often the case that we are working with likelihoods $L(\theta)$ which do not have a well-known lower bound (e.g. $L(\theta)$ could be parameterized by a deep neural network). In these situations, one would like any form of lower bound which is non-zero. Alternatively, one could investigate what happens when approximate bounds are used for $B_n(\theta)$ where cite the 0 <= Bn <= 1 does not hold for all θ.

Finally, we would like to mention that although the motivation for this work is computing on big datasets with large N, most experiments presented here and in the paper were small and run on a single machine. For FlyMC to truly be impactful in big data applications, an implementation on a distributed computing platform (e.g. Spark, Hadoop) which demonstrates tractable MCMC on large datacenter-scale datasets would be a convincing and immediately useful line of work.