Comparison of modern sampling methods with application to Bayesian Neural Networks

I-Ping Chou[1,2] TERADA Yoshikatsu[2] UCHIDA Masayuki[2]

[1]: Department of Statistics, National Cheng Kung University; [2]: Uchida Lab, Research Group of Statistical Inference, School of Engineering Science, Osaka University

***Abstract*—Neural Networks architecture are frequently used in deep learning field. Estimation of the parameters in Neural Networks is therefore an important study. In this study, we focus on a recent Monte-Carlo method based on piece-wise deterministic Markov process, called the Bouncy Particle Sampler [1] and its variants. These BPS-based algorithms are applied to Bayesian Logistic Regression and Neural Networks. Comparing the performance by several common-used indicators.**

I. Introduction

Neural Networks (NNs) have been more and more popular with recent advances in computer technologies. In various fields, deep learning has achieved great performance. For big and complicated data, NNs have large number of parameters and thus the parameter estimation in a NN becomes a large-scale problem. Recently, to estimate the parameters in a NN, we usually consider the frequentist approach such as the maximum likelihood estimation based on the stochastic gradient approach (e.g., Adam; Kingma and Ba, 2015). On the other hand, recently, several Bayesian approaches (e.g., Saatchi and Wilson, 2017) have been developed for NNs. However, since the usual MCMC algorithms such as Metropolis-Hastings algorithm do not work in a large-scale problem, these Bayesian NNs have serious computational issues. For a large-scale Bayesian inference, new Monte-Carlo (MC) methods based on a continuous time piecewise deterministic Markov process (PDMP) have drawn considerable attention in the field of Bayesian computation.

We applied Bouncy Particle Sample [1], Stochastic Bouncy Particle Sample [2] and Discrete Bouncy Particle Sampler [3] to Bayesian Logistic Regression and shallow Neural Networks. Test out how BPS-based algorithms performed under low dimensional and high dimensional problems. Also, various sample size are tested in this study.

In Bayesian Logistic Regression test, we setup an simple logistic model with 2 parameter involved, the details of model and formula derivation for BPS-based algorithms are addressed in section II, with the hyper-parameter tuning process with each algorithm in section III. In section IV, we demonstrate the simulation result with the settings shown in above sections.

MNIST dataset is applied for Neural Networks test, we setup two different data size and pixels of MNIST data sets to test how BPS-based algorithm perform in high dimensional problems with different situation. In section V, we introduced the instruction of two data sets we used with the model of the shallow Neural Networks we applied for the test. The tuning difference between Bayesian Logistic Regression and Neural Networks of each algorithms are shown in section VI, with the hyper-parameter settings used in this study. Result demonstration of our procedure are addressed in section VII. In section VIII, we summarize our simulation result, with pros and cons of each algorithms while facing each algorithms.

II. Bayesian Logistic Regression - Model, Dataset and Derivation

In section II, III and IV, we illustrate the numerical result of applying BPS, SBPS and DBPS to Bayesian Logistic Regression with various data size. By fitting a two-parameter model Bayesian Logistic Regression to random generated sample, testing out effective sample size per second and total computational time compare with our baseline - standard Metropolis-Hastings.

A. Model setting and data generation

We set a simple Bayesian Logistic Model for all algorithm with 2 parameters, that is :

$$log\frac{\theta}{1-\theta} = \beta_0 + z\beta_1 \tag{1}$$

where z is the input data. β_0, β_1 are intercept and coefficient for regression. θ is the probability of success event. Next, for the prior of β_0 and β_1, we set standard normal distribution as [1]. Thus, the target function will be:

$$\pi(\beta_0,\beta_1) = \prod_{i=1}^{r}\frac{e^{\beta_0+x_i\beta_1}}{1+e^{\beta_0+x_i\beta_1}} \tag{2}$$

$$\prod_{j=1}^{n-r}\frac{1}{1+e^{\beta_0+y_j\beta_1}}\frac{1}{2\pi}exp\left\{-\frac{1}{2}(\beta_0^2+\beta_1^2)\right\} \tag{3}$$

Where x_i represent the success events with quantity r and y_j stands for failure event with quantity $n-r$. As for data set generation, we generate x from the standard Normal distribution, with y generated by Binomial distribution with probability:

$$p = \frac{e^{\beta_0+x_i\beta_1}}{1+e^{\beta_0+x_i\beta_1}} \tag{4}$$

B. Derivation of gradient and upper-bound for BPS algorithm

As the energy function of BPS based algorithm is defined by $U = -log\pi$ we can easily get the energy function, that is

$$U(\beta) = \sum_{i=1}^{r} \left\{ log(1 + e^{\beta_0 + x_i\beta_1}) - log(e^{\beta_0 + x_i\beta_1}) \right\} \tag{5}$$

$$+ \sum_{j=1}^{n-r} \left\{ log(1 + e^{1 + \beta_0 + y_j\beta_1}) \right\} \tag{6}$$

$$+ \frac{1}{2}\left(\beta_0^2 + \beta_1^2\right) \tag{7}$$

also, by [1] section 2.3.3, the energy function can be decomposed in multiple part, thus we can decomposed our formula

$$\frac{\partial U(\beta)^{[1]}}{\partial \beta_0} = \sum_{i=1}^{r} \left(\frac{e^{\beta_0 + x_i\beta_1}}{1 + e^{\beta_0 + x_i\beta_1}} - 1 \right) \tag{8}$$

$$\frac{\partial U(\beta)^{[1]}}{\partial \beta_1} = \sum_{i=1}^{r} \left(\frac{e^{\beta_0 + x_i\beta_1} x_i}{1 + e^{\beta_0 + x_i\beta_1}} - x_i \right) \tag{9}$$

$$\frac{\partial U(\beta)^{[2]}}{\partial \beta_0} = \sum_{j=1}^{n-r} \left(\frac{e^{\beta_0 + y_j\beta_1}}{1 + e^{\beta_0 + y_j\beta_1}} \right) \tag{10}$$

$$\frac{\partial U(\beta)^{[2]}}{\partial \beta_1} = \sum_{j=1}^{n-r} \left(\frac{e^{\beta_0 + x_i\beta_1} y_j}{1 + e^{\beta_0 + y_j\beta_1}} \right) \tag{11}$$

$$\frac{\partial U(\beta)^{[3]}}{\partial \beta_0} = \beta_0 \tag{12}$$

$$\frac{\partial U(\beta)^{[3]}}{\partial \beta_1} = \beta_1 \tag{13}$$

where $[1]$ in the formula is the first term of energy function, term (5). Next, the upper bound is defined as $\langle \nabla U(\beta + vt), v \rangle$ in [1], section 2.1, where v stands for velocity corresponds to each particle, t stands for the current action time and ∇U stands for gradient partial. Also, upper-bound can be decomposed correspond to energy function, thus the upper-bound can be decomposed as follows:

$$\sum_{i=1}^{r} \left(\frac{e^{\beta_0 + x_i\beta_1}}{1 + e^{\beta_0 + x_i\beta_1}} - 1 \right) v_1 \leq \mathbf{1}[v_1 \leq 0] \ |v_1| \ r \tag{14}$$

$$\sum_{i=1}^{r} \left(\frac{e^{\beta_0 + x_i\beta_1} x_i}{1 + e^{\beta_0 + x_i\beta_1}} - x_i \right) v_2 \leq max\Big(0, \mathbf{1}[v_2 \leq 0] \ |v_2| \ \sum_{i=1}^{r} x_i\Big) \tag{15}$$

$$\sum_{j=1}^{n-r} \left(\frac{e^{\beta_0 + y_j\beta_1}}{1 + e^{\beta_0 + y_j\beta_1}} \right) v_1 \leq \mathbf{1}[v_1 \geq 0] \ v_1 \ (n - r) \tag{16}$$

$$\sum_{j=1}^{n-r} \left(\frac{e^{\beta_0 + x_i\beta_1} y_j}{1 + e^{\beta_0 + y_j\beta_1}} \right) \leq max\Big(0, \mathbf{1}[v_2 \geq 0] \ v_2 \ \sum_{j=1}^{n-r} y_j\Big) \tag{17}$$

with our prior bound be:

$$(\beta_0 + v_1 t) v_1 \tag{18}$$

$$(\beta_1 + v_2 t) v_2 \tag{19}$$

Remind that the only random term of the upper-bound of likelihood term is the current velocity of the particle, which means we don't have to calculate the exact likelihood term once more in each iteration, this can reduce computational cost, and so as prior bound. By access the upper-bound of the intensity, BPS is able to apply thinning algorithm to propose arrival time of bounce event.

III. Bayesian Logistic Regression - Hyper-parameter tuning for Algorithm

For notations used below, Δ stands for the valid in-time upper bound for continuous BPS methods and ref stands for the refresh intensity of velocity. τ are the hyper-parameter for discretizing the continuous path generate by BPS and SBPS, by the formula

$$x_{i+1} = x_i + v_i \times \tau \tag{20}$$

δ is the time step size and κ is the perturbation on velocity for each iteration, details are addressed in section 2.1 and 3.1.2 by [3].

A. BPS

In BPS, we simply used the same settings refer to [1], section 4.6, in order to reproduce the numerical result. The settings are as follows:

- Δ : 0.5
- ref : 0.5
- τ: 2e-04

B. SBPS

In [2], section 4, SBPS propose another approach for upper-bound. In spite of calculate the actual upper-bound, SBPS estimate it by applying a Bayesian Local Regression.By setting k as the confidence band multiple, we can adjust the scale of the estimation. In tuning k, we test out various k, and observe the counts while the actual upper-bound exceeds our estimation.

While $k = 3$, SBPS holds a minimum counts, meanwhile, maintaining a reasonable fraction between estimated upper-bound and exact upper-bound, shown in figure1. This can prevent the algorithm propose a large estimation, which might caused the rejection of proposal. The settings are as follows:

- batch size : 1 % of full sample size
- Δ : 0.5
- k : 3
- τ : 2e-04

C. DBPS

There are two hyper-parameter to tune in DBPS. δ is the time step size and κ is the perturbation on velocity for each iteration. Both of them will severely affect the trajectory generate by DBPS, thus we have to tune them carefully. We plot the efficient sample size of various combination between δ and κ. The tuning details are addressed in [3], section 4.1 and 4.2.

In figure2, we fixed δ and test out κ from $10^{-3} to 10^{1}$. ESS don't vary with the change on κ, however, the ESS holds a little peak at $\kappa = 10^{-2.5}$ on β_0 (blue line). Thus we use

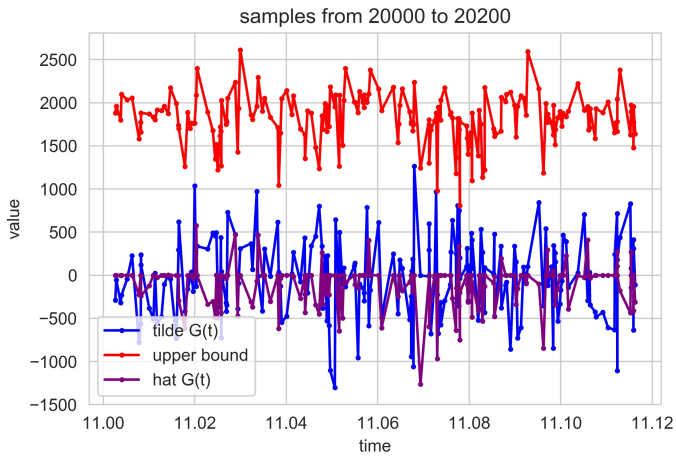

Fig. 1. Plot displays the actual upper-bound (tilde G(t)), the estimated upper-bound (hat G(t)) and estimation with k (upper bound). The proposal(red) always maintain a reasonable level higher than exact upper-bound(blue), which help SBPS keeps its mixing speed in various sample size problems.

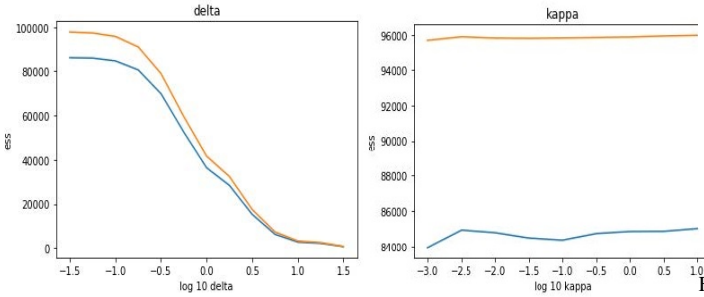

Fig. 2. Efficient sample size with various κ and δ. ESS decrease significantly as δ grows. Meanwhile, ESS don't vary with the change on κ.

$10^{-2.5}$ as our κ. Next, we fixed $\kappa = 10^{-2.5}$, test out δ from $10^{-1.5} to 10^{1.5}$. ESS decrease as δ grows, thus we use $\delta = 10^{-1.5}$ as out δ. The settings are as follows:

- $\delta : 10^{-1.5}$
- $\kappa : 10^{-2.5}$

D. Metropolis-Hastings

We used line chart to check the pattern of Metropolis-Hastings under different proposal variance. In figure3, Metropolis-Hastings holds a better pattern while proposal was sampled from normal distribution with $\mu = 0$, $\sigma = 0.1$.

IV. Bayesian Logistic Regression - Simulation Result and Performance Comparison

In this section, we test out how BPS-based algorithms perform under several indicators. By comparing trace plot, density plot, efficient sample size per second (ESS/s), execution time and acceptance rate. Examine how algorithms perform at low dimensional problems with data size from 2^{14} to 2^{19}

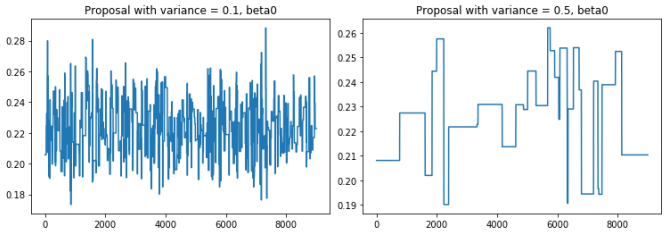

Fig. 3. Line chart of Metropolis-Hastings with proposal variance = 0.1 and 0.5. Proposal variance with 0.1 holds a more concussed pattern than 0.5 does.

A. Density plot and Trace plot

In figure4, all BPS-based algorithms converge to the same distribution as Metropolis-Hastings, this shows the invariant of BPS-based algorithms.

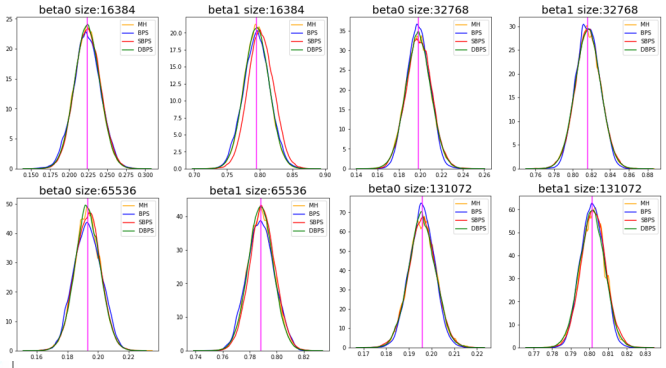

Fig. 4. Density plot of BPS-based algorithms and Metropolis-Hastings, data size from 2^{14} to 2^{17}

Next, we observed the trace plot of each algorithm. In figure5, BPS-based algorithm shows the pattern which the particle is converging to the target region, examine how particle moves from a random initial point to target.

Also, we take first 10% of samples as burn-in. In figure6, all BPS-based algorithm form well-mixed pattern in target region after burn-in.

B. Efficient Sample Size per Second (ESS/s) and Execution time

In ESS test, we calculated ESS for MCMC, that is

$$ESS = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho(k)} \tag{21}$$

In figure7, SBPS holds the best performance compare with other BPS-based algorithms, meanwhile holds a neck-to-neck result compare with our baseline. ESS/s of BPS and DBPS decrease as the data size increase, shows that BPS and DBPS has computational problem while facing larget data size problems. By checking execution time plot, figure8. BPS costs more time on each data size compare to SBPS and baseline. Meanwhile, DBPS not only costs the most time but also grows quadratically with the sample-size. This shows that BPS and DBPS might be unpractical while applying on large data size problems.

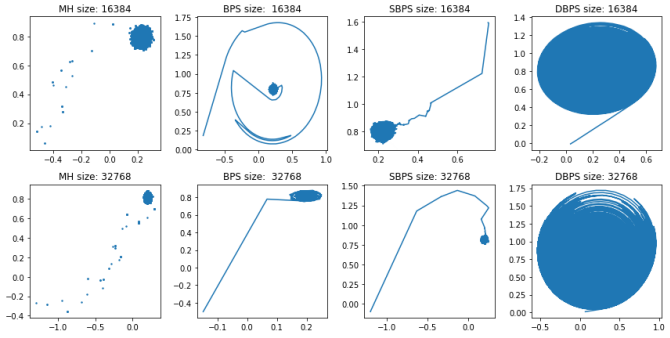

Fig. 5. Trace plot of BPS-based algorithms and Metropolis-Hastings, data size from 2^{14} to 2^{15}

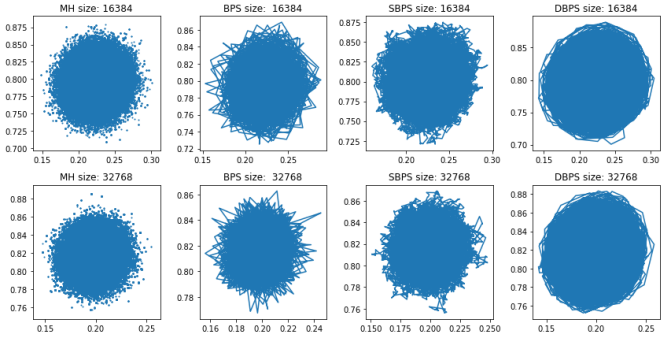

Fig. 6. Trace plot of BPS-based algorithms and Metropolis-Hastings, with 10% of samples as burn-in, data size from 2^{14} to 2^{15}

C. Acceptance Rate

Acceptance rate is observed for checking if the algorithm holds a reasonable mixing speed while applying on various data size.

In figure9, SBPS always keeps itself at around 0.1 acceptance rate with all sample size, this is because the SBPS algorithm propose another way to observe the upper bound during each iterations, by using a Bayesian local regression to estimate upper bound. This method helps SBPS propose a upper-bound proportional to exact upper-bound, which maintains the mixing speed on various situation, check section 4 in [4] and section III for more details.

DBPS holds a much higher bounce acceptance rate compare to all other methods, which means the particle of DBPS bounce more frequent to find converge point, and this may leads to a better mixing pattern and faster mixing speed. Lastly, acceptance rate for BPS and Metropolis-Hasting will decrease as sample-size grows, leads to a slower mixing speed. This problem might cause the algorithm spend too many iteration on burn in process. In figure10, BPS is still burn in while others gets a well-mixed pattern in high sample-sized simulations. This once again shows that BPS might not be a suitable algorithm while encounter large sample size problems.

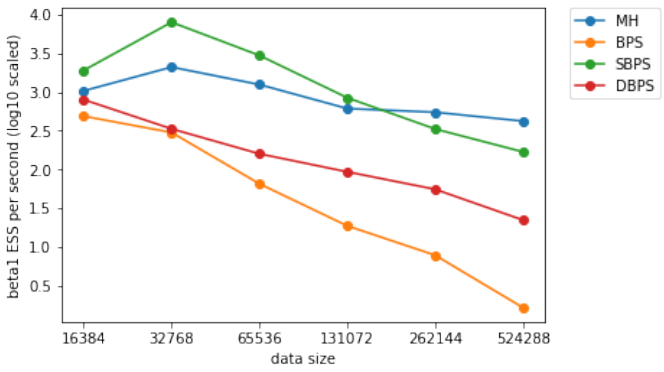

Fig. 7. ESS/s of each algorithm, data size from 2^{14} to 2^{19}

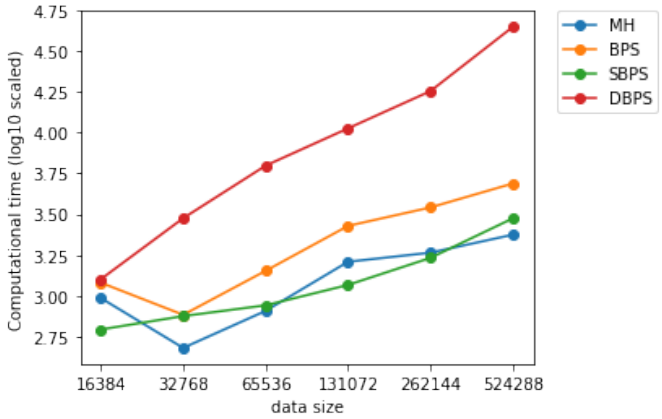

Fig. 8. Execution time of each algorithm, data size from 2^{14} to 2^{19}

V. Neural Networks - Model, Dataset and Derivation

In the following sections, we applied BPS-based algorithms to Neural Networks.

A. Dataset

Two different MNIST dataset were tested out to examine algorithms works under small and large data size. We randomly selected 2 labels, number 4 and 9 in this case, information of two dataset used in the simulation is as follows:

	Small dataset		Large dataset	
Pixels	20×20		28×28	
Label	4	9	4	9
Training data (pics)	400	400	5,842	5,949
Testing data (pics)	100	100	982	1,009
Total training data (pics)	800		11,791	
Total testing data (pics)	200		1,991	
Total parameters	10,051		19,651	

TABLE I
Form of details of two MNIST dataset for Neural Networks test

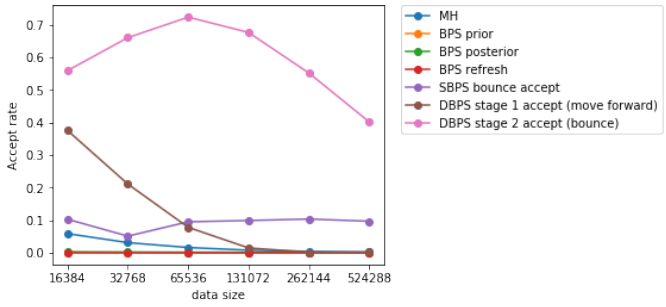

Fig. 9. Acceptance rate for each algorithm, data size from 2^{14} to 2^{19}

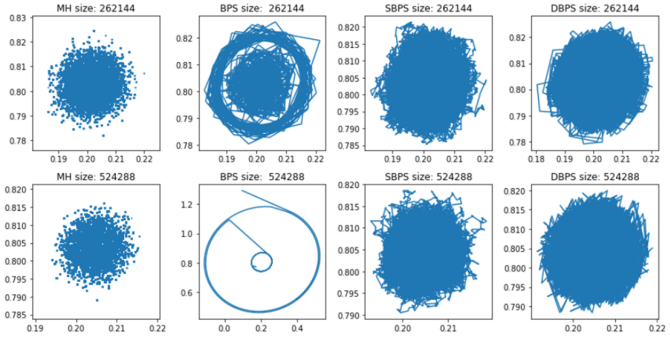

Fig. 10. Trace plot of BPS-based algorithms and Metropolis-Hastings, with 10% of samples as burn-in , data size from 2^{18} to 2^{19}

B. Model and Derivation

We applied a shallow Neural Network with a single hidden layer for the simulation, that is

$$\hat{y_i} = f_2(W_2 \cdot f_1(W_1 \cdot X_I + b_1) + b_2) \tag{22}$$

with the definition of notations

1) $\hat{y}$: the predict value of x_i.
2) W_1 : hidden layer with 25 neurons.
3) b_1 : bias for W_1.
4) f_1 : ReLU.
5) W_2 : output layer with 1 neuron.
6) b_2 : bias for W_2.
7) f_2 : Sigmoid.

Since the activation function of the output layer is sigmoid function, the derivation of gradient partial and upper bound shares the similar form with Bayesian Logistic Regression, check section II for more details.

VI. Neural Networks - Hyper-Parameter tuning for algorithm

All notations used the same definition as section III.

A. BPS and SBPS

As we mentioned in section III, SBPS applies Bayesian Local Regression to estimate upper-bound during each iteration. However, it requires the variance of all likelihood term partial ([2], section 4, equation 10), which causes the shortage of RAM usage. Thus by [2], theorem 1 in section 3.1, we simply applied Local-BPS with mini-batch per iteration.

Refresh rate is an important hyper-parameter for both BPS and SBPS, if we set the refresh rate too low, the particle will not be able to search the whole desire target region in a limited time, see figure11. The plot on the left holds an acceptable trace, however, the right one is too sparse for the algorithm. This implies the SBPS needs more iteration, or, a better refresh rate for exploring the whole target region.

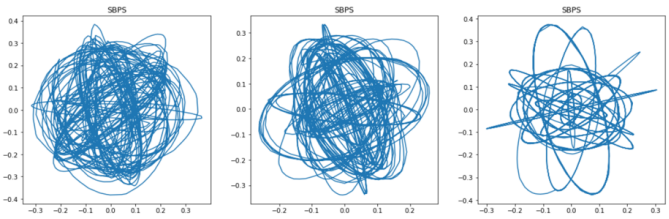

Fig. 11. Trace plot of SBPS to NNs, randomly selected two parameters for each plot, with refresh rate = 1, 10^6 iterations

Adding iteration might cost too much time for simulation, thus we decided to tune the refresh rate for better efficient sampling. Usually, we use the efficient sample size (ESS) to examine the performance of sampling, however, the refresh rate will affect the ESS so hard that cause ESS turn to be a useless indicator. In figure12, we illustrate the ESS/s with refresh rate = 10 and right with rate = 100, as we set a higher refresh rate, the algorithm is more likely to generate an independent sample, which increases the ESS.

Alternatively, we tune the refresh rate by using the trace plot of different refresh rate. In figure13, trace of SBPS with refresh rate = 100 formed a random walk since the velocity keeps refreshing during the iteration, meanwhile, rate = 10 holds a well-mixed pattern with a proper rate for SBPS to explore the target region. Thus, we take refresh rate = 10 for both BPS and SBPS. The final tunned hyper-parameters are as follows:

- Δ : 1
- ref : 10
- τ : 1e-03
- mini-batch size (for SBPS, small data set) : 200
- mini-batch size (for SBPS, large data set) : 500

B. DBPS

Next, in DBPS, we have two hyperparameter to tune, δ and κ ,and it is difficult to tune these for efficient sampling especially on high-dimensional problems.

δ holds a similar property as "learn-rate", if we set a higher δ, the particle might end up bouncing between two position, as the right plot of figure14. Also, the particle might walk too slow if we set a lower δ, and this might cause an inefficient sampling. κ holds a similar property as "refresh-rate" compare to BPS and SBPS, a high κ will leads to a random-walk and low κ will make the particle hard to explore the region in limited iterations.

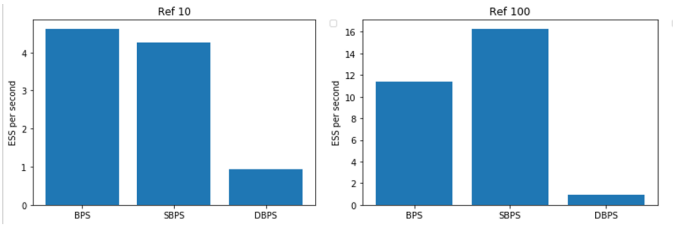

Fig. 12. ESS/s for refresh rate = 10 and 100, as we set a higher refresh rate, the algorithm is more likely to generate an independent sample, which increases the ESS.

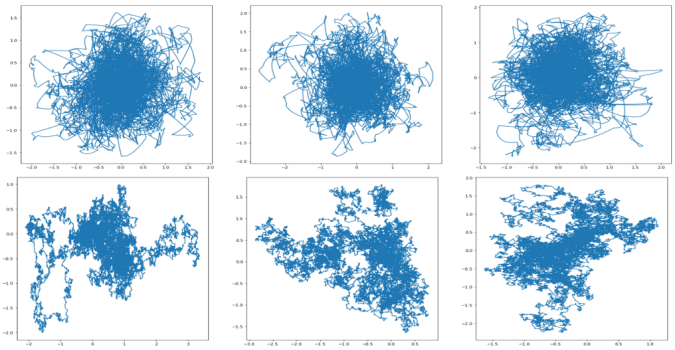

Fig. 13. Trace plot of SBPS for refresh rate = 10 and 100. **Above:** SBPS with refresh rate = 10, **Below:** SBPS with refresh rate = 100, both with 10^6 iterations.

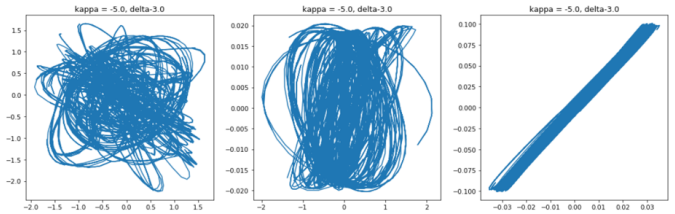

Fig. 14. Trace plot of DBPS with bad-tuned hyper-parameter, 10^6 iterations. In this figure, there are 3 different patterns with same setting. Although the left one holds a well-mixed pattern, the right one bounce between two position repeatedly, not mixing at all.

Thus, we have to try different combination of delta and kappa, and there are so many choices for them, which makes tuning DBPS becomes a difficult mission. Finally , we found δ equals to 0.01 and 0.1 holds the best pattern, and κ = 0.01 for reasonable perturbation. See figure15 for trace plot of δ = -1 and -2, κ =-1 to -3 for DBPS.

VII. Neural Networks - Simulation Result and Performance Comparison

In this section, we test out BPS, SBPS and DBPS to two data set mentioned in section V, by comparing the execution time and accuracy, examine how BPS-based algorithm performs in high dimensional problems. The following simulation tested with 10^7 iterations.

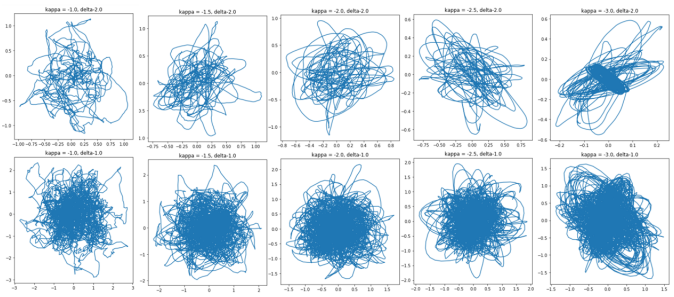

Fig. 15. Trace plot of DBPS with $\delta = -1$ and -2, $kappa = -1$ to -3.

A. Small data set simulation

In small data set, the execution time of BPS-based algorithms holds result different from Bayesian Logistic Regression, DBPS holds the minimal computational time. In contrast, BPS cost about 82 hours to complete 10^7 iterations, check figure16. This makes BPS an unpractical algorithm. Thus for accuracy test, we only compare SBPS and DBPS with other common method, SGD and Adam. SBPS gets better accuracy than DBPS, however, theres still a gap compare with SGD and Adam. Check table II for details.

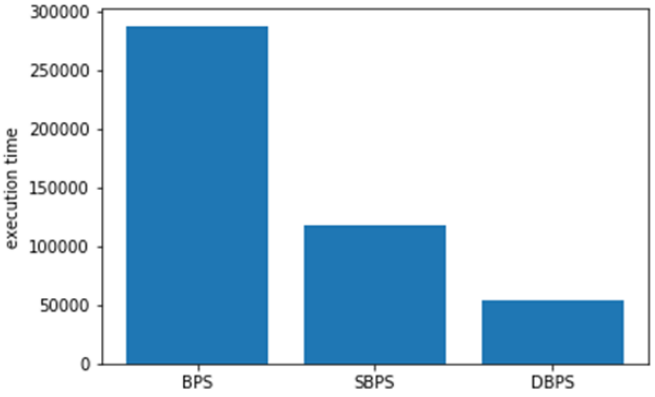

Fig. 16. Execution time cost for BPS-based algorithms.

SBPS	DBPS	SGD (5 epochs)	Adam (5 epochs)
151 / 200	96 / 200	127 / 200	195 / 200
Small data set (800 training)		SGD (100 epochs)	Adam (100 epochs)
		187 / 200	195 / 200

TABLE II
TABLE OF ACCURACY, COMPARING SBPS AND DBPS TO SGD AND ADAM. BOTH SBPS AND DBPS TAKE 30% FOR BURN-IN

B. Large data set simulation

We examine that BPS cost too much time on computation and SBPS has a better accuracy than DBPS, thus in larget data set simulation, we only compare SBPS to SGD and Adam. The accuracy of SBPS rise to 95.48%, neck to neck with Adam and SGD trained with 100 epoch. Thus the data size is also important for SBPS, check table III for details. However, the

trace plot, figure17, appears to be sparser compare with small data set, this means the iteration is not enough for SBPS to form a well-mixed pattern. Maybe we can get a better accuracy with more iteration.

SBPS	SGD (5 epochs)	Adam (5 epochs)
1901 / 1991, 95.48%	1841 / 1991, 92.47%	1957 / 1991, 98.29%
Small data set (800 training)	SGD (100 epochs)	Adam (100 epochs)
	1929 / 1991, 96.89%	1969 / 1991, 98.90%

TABLE III
TABLE OF ACCURACY, COMPARING SBPS TO SGD AND ADAM. WITH SBPS TAKE 30% FOR BURN-IN

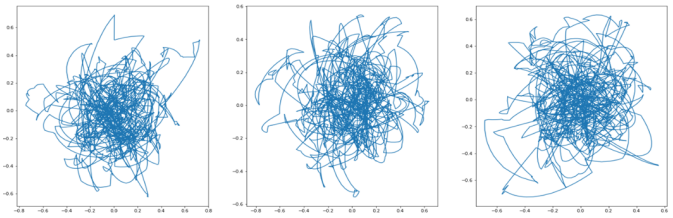

Fig. 17. Trace plot of applying SBPS to large data set.

VIII. CONCLUSION

In our result, whereas we discover that BPS with its variants still suffer computational issues for large sample-size problems and high-dimensional problems, we suggest SBPS which outperforms other algorithms. With less computational cost and minimal hyper-parameter tuning, SBPS can achieve high mixing speed. Meanwhile, DBPS holds a good mixing speed for small sample-size problems. However, the computational time of DBPS grows quadratically with the sample-size, and it is difficult to tune the hyper-parameter for the efficient sampling especially on high-dimensional problems. These problems cause that DBPS has a worse accuracy compare to SBPS in shallow NNs test. Moreover, BPS also encounter similar problems to DBPS on large sample-size problems. Besides the computational cost, acceptance rate of BPS will also decrease as sample-size grows, which leads to a slower mixing speed. In summary, SBPS holds a better performance overall, with less computational cost, minimal hyper-parameter, stable acceptance rate and a higher accuracy. Thus we suggest applying SBPS on sampling, if the upper bound are available to access.

source code at https://github.com/IPINGCHOU/FrontierLab-BPSs-code

REFERENCES

[1] A. Bouchard-Ct, S. J. Vollmer, and A. Doucet, "The bouncy particle sampler: A non-reversible rejection-free markov chain monte carlo method," 2015.

[2] A. Pakman, D. Gilboa, D. Carlson, and L. Paninski, "Stochastic bouncy particle sampler," 2016.

[3] C. Sherlock and A. H. Thiery, "A discrete bouncy particle sampler," 2017.

[4] Y. Saatchi and A. G. Wilson, "Bayesian gan," 2017.