Journal of Machine Learning **ISSN**: 2790-2048(e), 2790-203X(p)

Batch Normalization Preconditioning for Stochastic Gradient Langevin Dynamics

Susanna Lange [*] [1], Wei Deng [†] [2], Qiang Ye [‡] [3], and Guang Lin [§] [4]

[1]Data Science Institute, University of Chicago, Chicago, IL 60615, USA.
[2]Machine Learning Research, Morgan Stanley, New York City, NY 10036, USA.
[3]Department of Mathematics, University of Kentucky, Lexington, KY 40506, USA.
[4]Departments of Mathematics, & School of Mechanical Engineering, Purdue University, West Lafayette, IN 47907, USA.

Abstract. Stochastic gradient Langevin dynamics (SGLD) is a standard sampling technique for uncertainty estimation in Bayesian neural networks. Past methods have shown improved convergence by including a preconditioning of SGLD based on RMSprop. This preconditioning serves to adapt to the local geometry of the parameter space and improve the performance of deep neural networks. In this paper, we develop another preconditioning technique to accelerate training and improve convergence by incorporating a recently developed batch normalization preconditioning (BNP), into our methods. BNP uses mini-batch statistics to improve the conditioning of the Hessian of the loss function in traditional neural networks and thus improve convergence. We will show that applying BNP to SGLD will improve the conditioning of the Fisher information matrix, which improves the convergence. We present the results of this method on three experiments including a simulation example, a contextual bandit example, and a residual network which show the improved initial convergence provided by BNP, in addition to an improved condition number from this method.

Keywords:
Bayesian neural networks,
Preconditioning,
Batch normalization,
Stochastic gradient Langevin dynamics.

Article Info.:
Volume: 2
Number: 1
Pages: 65- 82
Date: March/2023
doi.org/10.4208/jml.220726a

Article History:
Received: 26/07/2022
Accepted: 10/03/2023

Handler:
Jiequn Han

1 Introduction

Markov chain Monte Carlo (MCMC) provides a principled framework for simulating the distribution of interest. During the simulation, the entire dataset is often used to compute the energy or the gradient, which, however, is not scalable enough in big data problems. To tackle this issue, stochastic gradient Langevin dynamics [23] proposes to inject additional Gaussian noise to stochastic gradient descent and smoothly transitions into an MCMC sampler as the step size goes to zero. The explorative feature of the sampler not only captures uncertainty for reliable decision-making but also facilitates non-convex optimization to alleviate over-fitting [19,25]. Since then, many interesting stochastic gradient Markov chain Monte Carlo (SG-MCMC) methods are proposed to accelerate the convergence [2,4,6,14]. However, these sampling algorithms still suffer from a slow convergence given morbid curvature information. To handle this issue, Girolami *et al.* [7] and Patterson

[*]susannalange@uchicago.edu
[†]weideng056@gmail.com
[‡]qiang.ye@uky.edu.
[§]Corresponding author. guanglin@purdue.edu.

and Teh [18] propose to adjust the Langevin algorithm on the Riemann manifold. Despite the correctness of the simulations, it is challenging to conduct the transformation in high-dimensional problems. Motivated by the adaptive preconditioner as in root mean squared propagation (RMSprop), the preconditioned SGLD algorithm (pSGLD) proposes to accelerate SGLD through a diagonal approximation of the Fisher information to resolve the scalability issue [13]. This uses gradient information to construct a preconditioner that can be interpreted to have an adaptive step size, with a smaller step size for curved directions and a larger step size for flat directions. This combats the slow training related to saddle points in neural networks. Other preconditioning methods have been investigated, including dense approximations of the inverse Hessian, as in [1,21]. There have also been approaches to use non-linear averaging methods to accelerate network convergence. He *et al.* uses a truncated generalized conjugate residual method that uses symmetry of the Hessian to improve convergence [9], and combines gradient descent ascent with Anderson mixing in generative adversarial networks, which was shown to improve adversarial training [10].

Another approach to accelerate convergence is to incorporate batch normalization (BN) layers into the network architecture [11]. BN uses mini-batch statistics to normalize hidden variables of a network and has been shown to decrease training times and improve network regularization. BN and its connection to Bayesian neural networks have been studied in [22], in particular, a network with BN can be interpreted as an approximate Bayesian model. Batch normalization has also been successfully applied to Bayesian models as studied in [16] which shows that including BN layers does not affect the probabilistic inference of variational methods. Batch normalization preconditioning is a technique that also uses mini-batch statistics but does so by transforming a network's trainable parameters using a preconditioner [12]. This is done by applying a preconditioning transformation on the parameter gradients during training. This transformation has been shown to improve the conditioning of the Hessian of the loss function which corresponds to a major advantage of the BNP transformation, that is, improvement in the convergence of the method. More importantly, BNP is a general framework that is applicable to different neural network architectures and in different settings, such as Bayesian models.

In this paper, we develop BNP for Bayesian neural networks to be used as a sampling method and examine its effects. We show that we can develop a similar preconditioning technique for SGLD that further improves initial convergence by improving the condition number of the Fisher information matrix. Additionally, we provide experimental results on three different methods, each showing improvement in convergence over our comparative baselines. We also compute the condition number of the approximate empirical Fisher information, which demonstrates the improvement in the condition number in our method.

The paper is organized as follows. In Section 2 we provide background information on stochastic gradient Langevin dynamics as well as a preconditioned version of SGLD. Section 2.2 introduces the basics of a batch normalization architecture. In Section 3 we expand upon a preconditioning method, batch normalization preconditioning, to a Bayesian setting that improves the conditioning of the Fisher information matrix and Section 4 showcases the benefits of BNP applied to SLGD in three different experiments.

2 Background

In this section, we provide preliminaries on the stochastic gradient Langevin dynamics method and preconditioned-SGLD. We also describe the batch normalization network architecture.

2.1 SGLD and pSGLD

Suppose we have a parameter θ with a prior distribution $p(\theta)$. We can then compute the posterior distribution $p(\theta|X)$ over N data points $\mathcal{X} = \{x_1, \ldots, x_N\}$ as

$$p(\theta|X) \propto p(\theta) \prod_{i=1}^{N} p(x_i|\theta),$$

where the prior serves as a regularization term, and we aim to optimize the likelihood by finding the maximum a posteriori (MAP), that is argmax $\log p(\theta|X)$. Stochastic gradient Langevin dynamics combines stochastic optimization with Markov chain Monte Carlo by incorporating uncertainty into predictive estimates by way of adding a noise component to the parameter updates. The update for SGLD is given at each time step t for a subset of n data points $X = \{x_{t1}, \ldots, x_{tn}\}$ as

$$\nabla\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla \log p(x_{ti}|\theta_t)\right) + \eta_t, \tag{2.1}$$

where $\eta \sim N(0, \epsilon_t)$. As t increases, it has been shown for SGLD that θ_t will converge in distribution to the posterior distribution [23] with the assumption that

1. The sequence of step sizes $\{\epsilon_t\}$ are decreasing with $\sum_{t=1}^{\infty} \epsilon_t = \infty$.
2. $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$.

Note that this standard SGLD algorithm updates all parameters with the same step size. However, when the different components of the parameter vector have different curvatures or different scales, it is more beneficial to use a preconditioning matrix $G(\theta)$ in SGLD to help adjust step size locally. The general framework of stochastic gradient Riemannian Langevin dynamics (SGRLD) was suggested in [18], which gives the update step

$$\begin{aligned}\nabla\theta_t = \frac{\epsilon_t}{2}\Bigg[G(\theta_t)\left(\nabla_\theta \log p(\theta_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla_\theta \log p(x_{ti}|\theta_t)\right) + \Gamma(\theta_t)\Bigg] \\ + G^{\frac{1}{2}}(\theta_t)\mathcal{N}(0, \epsilon_t I),\end{aligned} \tag{2.2}$$

where

$$\Gamma_i(\theta) = \sum_j \frac{\partial G_{i,j}(\theta)}{\partial \theta_j}$$

DOI https://doi.org/10.4208/jml.220726a | Generated on 2025-01-24 04:23:17

provides information on how the preconditioner G changes with respect to θ_t. For the convenience of implementations, $G(\theta_t)$ is replaced by the identity matrix in SGLD, in which case $\Gamma_i(\theta_t) = 0$. Particularly of interest is the preconditioner used in [13] which is the same as in RMSprop and serves to transform the rate of curvature to be equal in all directions. This preconditioning method is referred to as pSGLD. The preconditioning matrix estimates a diagonal matrix and the update at each step is given by

$$G(\theta_{t+1}) = \text{diag}\left(1 \oslash \left(\lambda 1 + \sqrt{V(\theta_{t+1})}\right)\right), \tag{2.3}$$

where

$$V(\theta_{t+1}) = \alpha V(\theta_t) + (1-\alpha)\overline{g}(\theta_t; \mathcal{X}^t) \odot \overline{g}(\theta_t; \mathcal{X}^t), \tag{2.4}$$

and

$$\overline{g}(\theta_t; \mathcal{X}^t) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \log p(x_{ti}|\theta_t)$$

is the mean of the gradient over the mini-batch $\mathcal{X}^t$ and $\alpha \in [0,1]$. Computations in Eqs. (2.3) and (2.4) are using element-wise multiplication $\odot$ and division $\oslash$. A benefit of using this RMSprop preconditioner is that it adapts to the local geometry and curvature, in particular, the step sizes can be considered as adaptive, where large steps are taken in flat directions and small steps are taken in curved directions.

2.2 Batch normalization

Batch normalization is a technique that incorporates normalization layers into a neural network architecture. It was originally developed to remedy internal covariate shift, which refers to the shifting of distributions between layers during training that can diminish the effectiveness of gradient descent [11]. Such distribution changes slow down training since parameters must adapt to the changed distribution of the different network layers. Reducing this shift causes improvement in the speed of training, network regularization, and performance.

The BN transformation normalizes the hidden variables by subtracting by the mini-batch mean and dividing by the mini-batch standard deviation while introducing trainable re-centering and re-scaling parameters. To understand the BN architecture, we consider a fully connected neural network and follow the terminology in [12] to introduce a BN network. Let the ℓ-th hidden layer of a fully connected network be defined as

$$h^{(\ell)} = g\big(W^{(\ell)}h^{(\ell-1)} + b^{(\ell)}\big) \in \mathbb{R}^{n_\ell}, \tag{2.5}$$

which takes input $h^{(\ell-1)}$ from the previous layer, a chosen activation function g, and weight and bias elements $W^{(\ell)}$ and $b^{(\ell)}$, to construct the current hidden variable $h^{(\ell)}$. Note the input to the network is given by $h^{(0)}$. Let $\{h_1^{(0)}, h_2^{(0)}, \ldots, h_N^{(0)}\}$ be a mini-batch input to the training network with N examples and $A = \{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\}$ the hidden

variables of layer $\ell - 1$. The update in Eq. (2.5) describes the standard fully connected iteration. Applying BN to this network replaces the iteration update in (2.5) by

$$h^{(\ell)} = g\big(W^{(\ell)}\mathcal{B}_{\beta,\gamma}(h^{(\ell-1)}) + b^{(\ell)}\big), \tag{2.6}$$

where

$$\mathcal{B}_{\beta,\gamma}\big(h^{(\ell-1)}\big) = \gamma\frac{h^{(\ell-1)} - \mu_A}{\sigma_A} + \beta, \tag{2.7}$$

and σ_A and μ_A are the standard deviation and mean vectors of the hidden variables in layer $\ell - 1$ and γ, β are the trainable re-scaling and re-centering parameter vectors. The BN operator is denoted $\mathcal{B}_{\beta,\gamma}(\cdot)$ in Eqs. (2.6) and (2.7).

Since BN has the mini-batch statistics embedded in the architecture, a theoretical disadvantage is that the training network depends on the mini-batch inputs, and in particular, the inference network is different from the training network.

3 Batch normalization preconditioning

We extend a preconditioning method of batch normalization preconditioning originally derived for neural networks to SGLD. BNP is also a technique to accelerate the convergence of a neural network using mini-batch statistics. Instead of changing the network architecture, as is done in BN, BNP uses a preconditioning matrix on the parameter gradients during training. This transformation improves the conditioning of the Hessian of the loss function and has been shown in [12] to outperform BN in small mini-batch settings and online learning.

We develop BNP for SGLD by considering the gradient descent for parameters in one layer. We consider the Fisher information matrix in terms of the Hessian of the log-likelihood and represent the Fisher information matrix in terms of the mini-batch activations.

Consider a Bayesian feedforward neural network with L layers as defined in Eq. (2.5). We denote $h_i^{(\ell)} = g(a_i^{(\ell)})$ as the i-th entry of $h^{(\ell)}$ where $a_i^{(\ell)} = w_i^{(\ell)^T} h^{(\ell-1)} + b_i^{(\ell)} \in \mathbb{R}$. Here $w_i^{(\ell)^T} \in \mathbb{R}^{1\times m}$ and $b_i^{(\ell)}$ are the respective i-th row and entry of $W^{(\ell)}$ and $b^{(\ell)}$, and m is the dimension of $h^{(\ell-1)}$. Let

$$\widehat{w}^T = \left[b_i^{(\ell)}, w_i^{(\ell)^T}\right] \in \mathbb{R}^{1\times(m+1)}, \tag{3.1}$$

$$\widehat{h} = \begin{bmatrix} 1 \\ h^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{(m+1)\times 1}, \quad a_i^{(\ell)} = \widehat{w}^T\widehat{h}. \tag{3.2}$$

Note that the Fisher matrix can be described as $-E(\nabla_\theta^2(\log p_\theta(x)))$, for expected value E with Hessian operator ∇_θ^2. We call

$$\mathcal{I}(\theta) = -\frac{1}{N}\sum_{j=1}^{N}\nabla_\theta^2 \log p_\theta(x_j),$$

an empirical Fisher matrix, which serves as an approximation of the Fisher matrix based on the training data and training distribution. For a fully-connected neural network, we can write the empirical Fisher matrix in terms of the mini-batch activations, as shown in the theorem below. The importance of this form is that applying the BNP preconditioner serves to improve the conditioning of this matrix.

Theorem 3.1. *Let $-\log p((x,y)|w)$ be the negative log-likelihood loss function defined from the output of a fully-connected multi-layer neural network* (2.5) *with parameter w for a single network input x. Consider the weight and bias parameters $w_i^{(\ell)}, b_i^{(\ell)}$ at the ℓ-th layer and let*

$$\widehat{w} = \begin{bmatrix} b_i^{(\ell)} \\ w_i^{(\ell)} \end{bmatrix} \quad \textit{and} \quad \widehat{h} = \begin{bmatrix} 1 \\ h^{(\ell-1)} \end{bmatrix}.$$

Write the likelihood $p((x,y)|\widehat{w})$ as a function of $\widehat{w}$ through the activation $a^{(\ell)} := \widehat{w}^T\widehat{h}$, that is $p((x,y)|\widehat{w}) = f(a^{(\ell)})$ for some function f. When training over a mini-batch of N inputs $\{x_1, x_2, \ldots, x_N\}$, let $\{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\}$ be the associated $h^{(\ell-1)}$ and let

$$\widehat{h}_j = \begin{bmatrix} 1 \\ h_j^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{m+1}.$$

Then the empirical Fisher information matrix with respect to $\widehat{w}$,

$$\mathcal{I}(\widehat{w}) := -\frac{1}{N}\sum_{j=1}^{N} \nabla^2_{\widehat{w}} \log p\big((x_j,y_j)|\widehat{w}\big),$$

can be written as

$$\mathcal{I}(\widehat{w}) = -\widehat{H}^T S \widehat{H},$$

where

$$\widehat{H} = [e,\ H], \quad H = \begin{bmatrix} h_1^{(\ell-1)^T} \\ \vdots \\ h_N^{(\ell-1)^T} \end{bmatrix},$$

and

$$S = \frac{1}{N}\operatorname{diag}\left(\frac{f'(\hat{w}^T\hat{h}_j)^2}{f(\hat{w}^T\hat{h}_j)^2} - \frac{f''(\hat{w}^T\hat{h}_j)}{f(\hat{w}^T\hat{h}_j)}\right).$$

Using this expression of the empirical Fisher information matrix, we can improve its conditioning by using a preconditioning transformation. Constructing a preconditioner $G(\theta) = PP^T$, we use the update step in (2.2) with P:=UD, and

$$U := \begin{bmatrix} 1 & -\mu_A^T \\ 0 & I \end{bmatrix}, \quad D := \begin{bmatrix} 1 & 0 \\ 0 & \operatorname{diag}(\sigma_A) \end{bmatrix}^{-1}, \tag{3.3}$$

DOI https://doi.org/10.4208/jml.220726a | Generated on 2025-01-24 04:23:17

where

$$\mu_A := \frac{1}{N} H^T e = \frac{1}{N} \sum_{j=1}^{N} h_j^{(\ell-1)}, \tag{3.4}$$

and

$$\sigma_A^2 := \frac{1}{N} \sum_{j=1}^{N} \left(h_j^{(\ell-1)} - \mu_A \right)^2 \tag{3.5}$$

are the (vector) mean and variance of $\{h_j^{(\ell-1)}\}$ respectively. Note the inverse notation in Eq. (3.3) refers to the element-wise inverse and e notation refers to a vector of ones.

As mentioned in [12], to ensure the Hessian blocks from different layers have comparable norms, we scale the preconditioner PP^T by $1/q^2$ where $q^2 = \max\{m/N, 1\}$. Thus, the BNP transformation on the gradients is outlined in Algorithm 1.

Algorithm 1 One Step of BNP Training on $W^{(\ell)}, b^{(\ell)}$ of the ℓth Dense Layer.

Given: $\epsilon_1 = 10^{-2}, \epsilon_2 = 10^{-4}$ and $\rho = 0.99$; learning rate α; initialization of vectors: $\mu = 0, \sigma = 1$;

Input: Mini-batch output of previous layer $A = \{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\} \subset \mathbb{R}^m$ and the parameter gradients: $G_w \leftarrow \frac{\partial \mathcal{L}}{\partial W^{(\ell)}} \in \mathbb{R}^{n \times m}$, $G_b \leftarrow \frac{\partial \mathcal{L}}{\partial b^{(\ell)}} \in \mathbb{R}^{1 \times n}$, and parameter noise $\eta_w \in \mathbb{R}^{n \times m}$, $\eta_b \in \mathbb{R}^{1 \times n}$.

1. Compute mini-batch mean/variance: μ_A, σ_A^2;
2. Compute running average statistics: $\mu \leftarrow \rho\mu + (1-\rho)\mu_A$, $\sigma^2 \leftarrow \rho\sigma^2 + (1-\rho)\sigma_A^2$;
3. Set $\tilde{\sigma}^2 = \sigma^2 + \epsilon_1 \max\{\sigma^2\} + \epsilon_2$ and $q^2 = \max\{m/N, 1\}$;
4. Update G_w:
$G_w(i,j) \leftarrow \frac{1}{q^2}[G_w(i,j) - \mu(j)G_b(i)]/\tilde{\sigma}^2(j)$;
5. Update G_b: $G_b(i) \leftarrow \frac{1}{q^2} G_b(i) - \sum_j G_w(i,j)\mu(j)$;
6. Update η_w: $\eta_w(i,j) \leftarrow \eta_w(i,j)/(\tilde{\sigma}(j)\sqrt{q})$;
7. Update η_b: $\eta_b(i) \leftarrow \frac{1}{\sqrt{q}} \eta_b(i) - \sum_j \eta_w(i,j)\mu(j)$.

Output: Preconditioned gradients and noise: G_w, G_b, η_w, η_b.

Note for implementation of this method as in Algorithm 1, $\max\{\sigma^2\}$ denotes the maximum entry of the vector $\sigma^2 \in \mathbb{R}^m$. Note $\tilde{\sigma}^2$ is σ^2 with a small number added to prevent division by a number smaller than $\epsilon_1 \max\{\sigma^2\}$ or ϵ_2. We use running averages for the componentwise mean and variance computed in Step 2 of Algorithm 1. We apply the preconditioner P on the noise in Steps 6 and 7, which is exactly the algorithm as in [13].

This preconditioning transformation gives the corresponding preconditioned Fisher matrix of $P^T \widehat{H}^T S \widehat{H} P$. Note that multiplying $\widehat{H}$ by U makes the first column orthogonal to the rest, as

$$\widehat{H}U = \left[e,\ H - e\mu_A^T \right],$$

and

$$\left(H - e\mu_A^T\right)^T e = 0.$$

DOI https://doi.org/10.4208/jml.220726a | Generated on 2025-01-24 04:23:17

Additionally, multiplying $H - e\mu_A^T$ by D scales all columns of $H - e\mu_A^T$ to have the same norm. Both of these results of the BNP transformation were shown in [12] to improve the condition number of the preconditioned matrix.

While Algorithm 1 focuses on a fully connected network, BNP can also be applied to convolutional neural networks (CNNs). In particular, Section 4.3 implements a residual network that has a framework of convolution layers. BNP performs well in situations where BN performs well. So, our experiments are limited to fully-connected networks or residual networks. Additionally, we expect BNP to perform well with capturing multiple modes as our algorithm is based on an MCMC method, which performs well in this situation. This is demonstrated in the experimental results of Section 4.1.

The update step in (2.2) uses $\Gamma(\theta_t)$ in the preconditioner update, we follow [13] which argues that, under given conditions, $\Gamma(\theta_t)$ contributes little to the update and can be dropped during sampling to reduce computation. This can be justified in our case by directly using [13, Theorem 1] and following [13, Corollary 2]. To summarize, this states that under the convergence assumptions (1) and (2) in Section 2.1 and a preconditioning algorithm with an update step given by Eq. (2.2), we can bound the MSE of an SG-MCMC preconditioning algorithm at a finite time. That is, given a test function ϕ that satisfies convergence assumptions, where we denote $\overline{\phi}$ the true posterior expectation and $\hat{\phi}$ the weighted sample average that approximates $\overline{\phi}$, we have that $\mathbb{E}(\hat{\phi} - \overline{\phi})^2$ is bounded.

Using the BNP preconditioner, we can follow the argument of [13] and see that the effect of Γ on the MSE is small, as it produces a controllable bias. Although we introduce bias in this way, it is controllable and much easier to implement, hence we balance efficiency with a small sacrifice in accuracy. Thus, we remove the Γ term in Eq. (2.2) during computation to speed up our BNP method.

4 Experiments

We present BNP as a sampler in three different experiments to evaluate uncertainty. First, we show a multidimensional curve-fitting example. We next present a contextual bandit problem with 4 different datasets. Additionally, we show results on a residual neural network. In all cases, we compare against other baselines, including pSGLD. All experiments show that BNP increases the speed of convergence over comparative methods. Thus, these results show that BNP can be successfully applied to SGLD. The improved early convergence is useful in the setting where it is beneficial to get an estimation of results quickly. Unless otherwise mentioned, the default setting for BNP is used with $\rho = 0.99, \epsilon_1 = 1e-2$, and $\epsilon_2 = 1e-4$.

4.1 Simulations of a multimodal distribution

We evaluate BNP in a curve-fitting example. Our data set is generated by sampling 1000 inputs (x, y) uniformly and at random from $[-7.5, -5] \times [-2.5, 15]$, capturing 4 local extrema of the target function. Note this choice of range is to take advantage of the BNP algorithm, as we choose (x, y) pairs with different scales. For each input we compute the

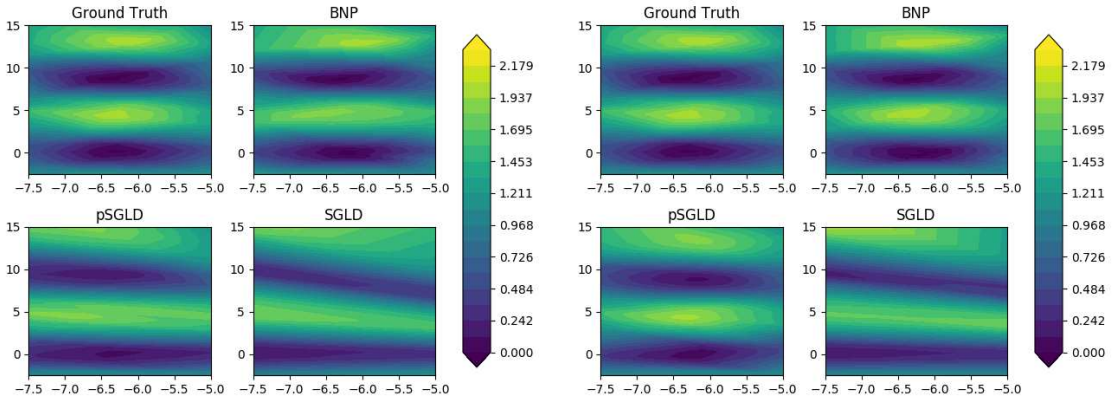


Figure 4.1: Predictions shown by BNP, SGLD, and pSGLD after 500 epochs (left) and 1000 epochs (right) given noisy data.

corresponding noisy label as

$$1 + \frac{1}{4000}(x^2 + y^2) - \cos(x)\cos\left(\frac{y\sqrt{2}}{2}\right) + \epsilon_n,$$

where $\epsilon_n \in \mathcal{N}(0, 0.1)$. Note this is a noisy Griewank function. We use a fully connected 3-layer neural network with 100 hidden units in all cases to fit this data. We compare BNP with SGLD and pSGLD after 500 epochs on 40,000 test data generated uniformly from $[-7.5, -5] \times [-2.5, 15]$. Our ground truth is generated by SGLD with 10,000 epochs. Note we use SGLD as our ground truth since it is the fundamental MCMC algorithm in big data problems and the theoretical correctness is guaranteed in the asymptotic sense [3].

For parameter settings, BNP uses $\rho = 0.985$, $\epsilon_1 = \epsilon_2 = 1e-3$. Learning rates are $1e-5$, $1e-5$, $5e-5$ for BNP, SGLD, and pSGLD respectively.

Fig. 4.1 shows the output of each method, averaging over the last 50 epochs. BNP converges much quicker to the ground truth compared with pSGLD and SGLD. However, we note that pSGLD also converges to the ground truth by increasing the number of epochs, as seen in Fig. 4.1 (right).

4.2 Contextual bandits

We experiment with Thompson sampling for contextual bandits where an optimization metric is used to evaluate the performance of different samplers as in [5, 20]. Suppose we have an agent who is given a context $x \in X$. The agent then takes some action $a \in A$ from a collection of possible actions $\{a_1, \ldots, a_n\}$. Depending on the choice, the agent receives a reward of r. The aim is to maximize the cumulative reward (or minimize regret). Since only the reward for the chosen action is revealed, there is a notion of exploration versus exploitation, that is, the desire to try new, and potentially better, actions versus exploiting the known, good actions.

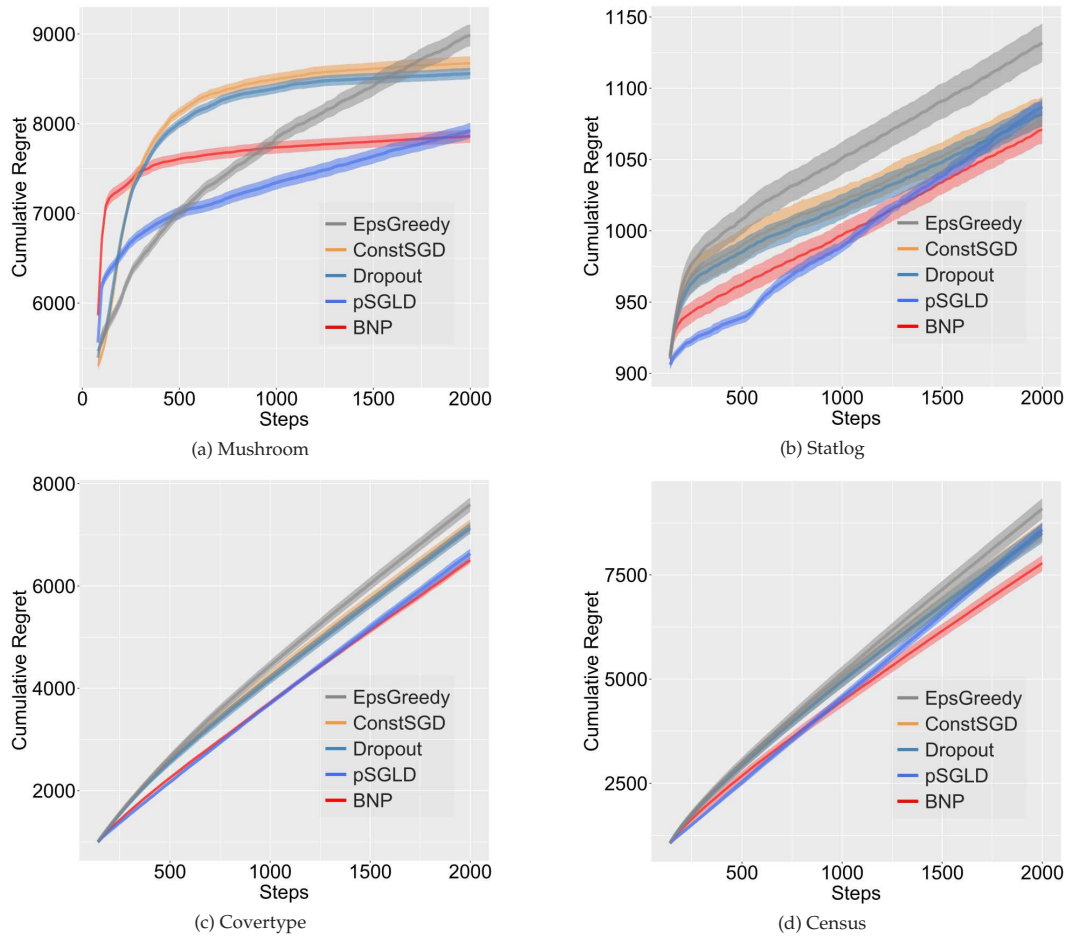
We experiment on 4 datasets: Mushroom, Statlog, Covertype, and Census.

- The mushroom dataset contains 8,124 mushrooms, with 22 attributes or contexts for each mushroom. Each mushroom is labeled as poisonous or edible and at each step, the agent is given the features of a particular mushroom and must decide whether to eat it. A reward of 5 is given for eating an edible mushroom, 0 for not eating a mushroom, and a reward is randomly chosen of 5 or -35 if a poisonous mushroom is eaten.
- The statlog dataset predicts the state of the radiator subsystem of a shuttle given 9 attributes of a space shuttle flight. This dataset contains information from 58,000 shuttle flights, and a reward of 1 is given for a correct classification of the state and 0 for an incorrect classification.
- The covertype dataset classifies forest cover type into 7 categories given a list of 54 features. This also provides a reward of 1 for a correct classification and 0 for an incorrect classification. This dataset contains 581,012 data points.
- The census dataset contains information from the US Census database and given a list of 94 attributes aims to classify them into one of 14 occupations. Again, a reward of 1 is given for correct classification and 0 otherwise. There are 48,842 data points in this dataset.

In our experiments, we compare our method of BNP applied to SGLD with preconditioned SGLD, SGD, dropout, and an epsilon greedy algorithm. The algorithms for pSGLD, SGD, and BNP are as described above. The epsilon-greedy algorithm allows for a random action to be taken a given ϵ percentage of the time and the dropout algorithm, which randomly turns off a percentage of neurons in the network. Fig. 4.2 shows the cumulative regret on the 4 datasets described above, recall a lower regret is desired. On the Mushroom dataset, BNP converges fastest. While the greedy algorithm and pSGLD have a smaller regret for the first 1000 and 1750 steps respectively, note that after 2000 steps they have not converged and surpass the cumulative regret of BNP.

For the detailed hyperparameter setups, we fix the temperature 0.03 and L_2 penalty of 1 for all the datasets, except the Mushroom dataset which uses a temperature of 0.3. The dropout rate is set to 50%, and we randomly simulate 5 samples for the Dropout approach. EpsGreedy anneals the learning rate by an annealing factor of 0.999 in each iteration and has a 0.3% chance to make random actions to avoid over-exploitation; by contrast, the rest of the algorithms adopt a constant learning rate, and the learning rates for Mushroom, Statlog, Covertype, and Census datasets are set to $1e-6$, $3e-6$, $3e-6$, and $1e-7$, respectively. pSGLD adopts a regularizer of $\lambda = 0.001$ in Eq. (2.3) to control the regularity of the preconditioner and the underlying smoothing factor is set to $\alpha = 0.99$ in Eq. (2.4). For the Mushroom dataset, $\alpha = 0.95$ performs better and is used. In particular for the BNP algorithm, the pair of hyperparameters (ϵ_1, ϵ_2) is set to $(1e-1, 1e-4)$, $(1, 0.1)$, $(3e-2, 3e-2)$, $(1e-2, 1e-2)$ for Mushroom, Statlog, Covertype, and Census datasets, respectively.

For the remaining 3 datasets - Statlog, Covertype, and Census, we find that BNP achieves lower cumulative regret at the end of 2000 steps, while convergence is not achieved with



(a) Mushroom (b) Statlog

(c) Covertype (d) Census

Figure 4.2: Cumulative regret on 4 datasets - Mushroom, Statlog, Covertype, Census using the methods of EpsGreedy, SGD, Dropout, pSGLD, and BNP.

any of the algorithms. The greedy algorithm has the highest cumulative regret after 2000 steps in each dataset. Notice that pSGLD is the most comparable with BNP, especially with the Covertype data, with BNP performing only slightly better at 2000 steps. Otherwise, we see better performance (smaller cumulative regret) earlier with pSGLD, but faster convergence with BNP.

4.3 Residual network and empirical Fisher condition number

We experiment with a residual network of depth 20 (ResNet-20) on the CIFAR100 dataset using a Bayesian neural network. The CIFAR100 dataset consists of 60,000 color images of 32 by 32 pixels with 50,000 training images and 10,000 testing images. There are 100 classes of images in this dataset. With this network, we show the faster convergence of BNP in addition to computations of the empirical Fisher condition number.

We use the SG-MCMC method of stochastic gradient Langevin dynamics and compare it with BNP and pSGLD. All networks are trained using the momentum optimizer with mini-batch size 256. The learning rate is multiplied by 0.1 at epoch 250 followed by a linear decay of 0.985. Learning rates used are $1e-6$, $1e-6$, $1e-6$, and $5e-6$ for BNP, SGLD, and the different versions of pSGLD presented, respectively. Results are shown in Fig. 4.3. We see that our method of BNP achieves faster convergence and comparable final test accuracy to SGLD. Different tuning parameters lead to versions of pSGLD that either achieve faster convergence than SGLD but do not perform well with the learning rate decay, or achieve comparable final accuracy with no increased initial convergence.

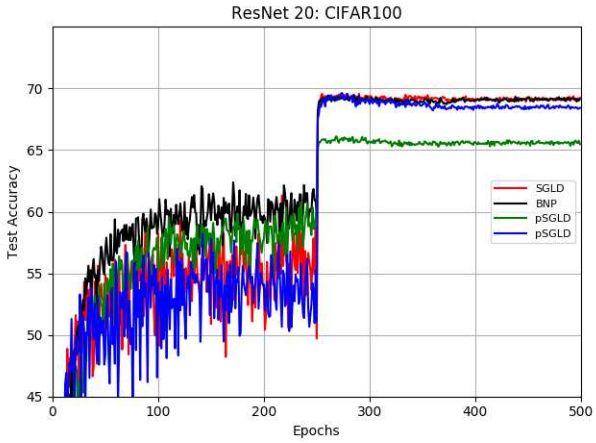


Figure 4.3: ResNet 20 on the CIFAR 100 dataset. BNP, SGLD, and 2 versions of pSGLD are shown for comparison.

We also record the Bayesian model averaging (BMA) of the accuracy and present the best results in Table 4.1. We find that BNP achieves a slightly better average by 0.04 with a BMA of 69.68, SGLD has a BMA of 69.64, and pSGLD BMA 69.59. Hence, this shows that our method of BNP achieves comparable results to SGLD and pSGLD, with a considerable improvement in initial convergence.

Table 4.1: Table of Bayesian model averaging (BMA) for ResNet 20, which shows the average accuracy of each model.

Method	**BMA**
BNP	69.68
SGLD	69.64
pSGLD	69.59

In addition to accuracy measure, we also include experiments that support our theory that BNP improves the Fisher information matrix. We calculate the condition number of $\hat{I}_{1,t}$, computed as in [1] as an online average of an approximation of the empirical Fisher

$$\hat{I}_{1,t} = (1-\kappa_t)\hat{I}_{1,t-1} - \kappa_t V\left(\theta_t, X_n^t\right),$$

DOI https://doi.org/10.4208/jml.220726a | Generated on 2025-01-24 04:23:17

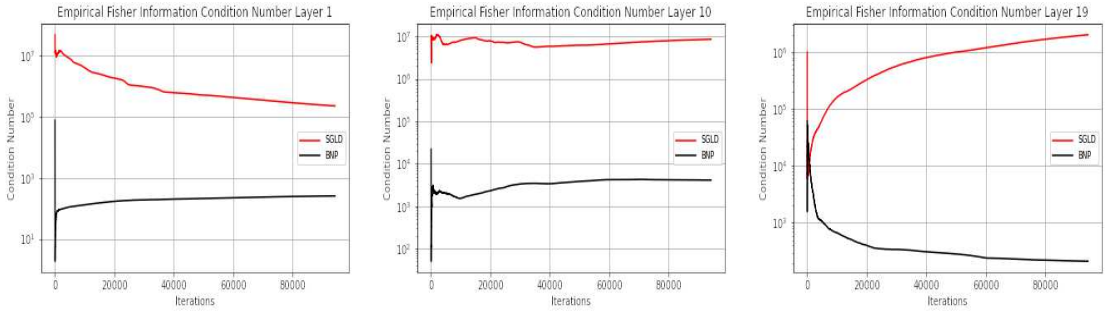


Figure 4.4: We compute the condition number of an approximation of the empirical Fisher information matrix in ResNet 20. In all layers, we found an improvement in the condition number of BNP versus SGLD.

where $\kappa = 1/t$ and t corresponds to the iteration. We use

$$V(\theta_t, X_n^t) = \bar{g}(\theta_t, X_n^t) \odot \bar{g}(\theta_t, X_n^t),$$

which serves as our approximation of the empirical Fisher information matrix for a mini-batch, where $\bar{g}(\theta_t, X_n^t)$ is the sample mean of the gradient using mini-batch X_n^t and $\odot$ represents element-wise multiplication. Note this is similar to the computation to approximate the Fisher as in [13], where we are interested in the approximation for a mini-batch. Note, since we compute this for convolutional layers, we use a fixed weight output channel and the corresponding bias term. That is, we compute the empirical Fisher information for $\theta_t = [b_t, w_t]$ where $w_t \in \mathbb{R}^{ck^2}$, where $k \times k$ corresponds to the convolution kernel and c is the input channel dimension. We compute the condition number of $\hat{I}_{1,t}$ at each of the 19 convolutional layers of ResNet 20 for θ_t. In our experiments, we find the BNP has an improved condition number over the baseline of SGLD in all 19 convolutional layers. We present layers 1, 10, and 19 in Fig. 4.4.

We also include a metric to measure the calibration of the model. Calibration is the idea that a model's confidence will match its predictions. We measure the calibration by computing the expected calibration error (ECE) as described in [17]. This metric is defined as the expected absolute difference between the model's confidence and its accuracy. The computation of ECE in practice is through an approximation. First, the interval $[0, 1]$ is partitioned into a specified number of M equally spaced bins. Let B_i be the set of samples with confidences contained in bin i. Then the accuracy and confidence of the i-th bin are computed as

$$acc_i = \frac{1}{|B_i|} \sum_{j \in B_i} \mathbb{1}_{\hat{y}_i = y_i}$$

and

$$conf_i = \frac{1}{|B_i|} \sum_{j \in B_i} \hat{p}_j,$$

respectively. Here $\mathbb{1}_{\hat{y}_i = y_i}$ is the indicator function for the predicted, $\hat{y}_i$, and actual, y_i, label. The ECE is approximated by taking the weighted average of the absolute difference

DOI https://doi.org/10.4208/jml.220726a | Generated on 2025-01-24 04:23:17

between the accuracy and confidence of each bin as

$$ECE = \sum_{i=1}^{M} \frac{|B_i|}{N} |acc_i - conf_i|.$$

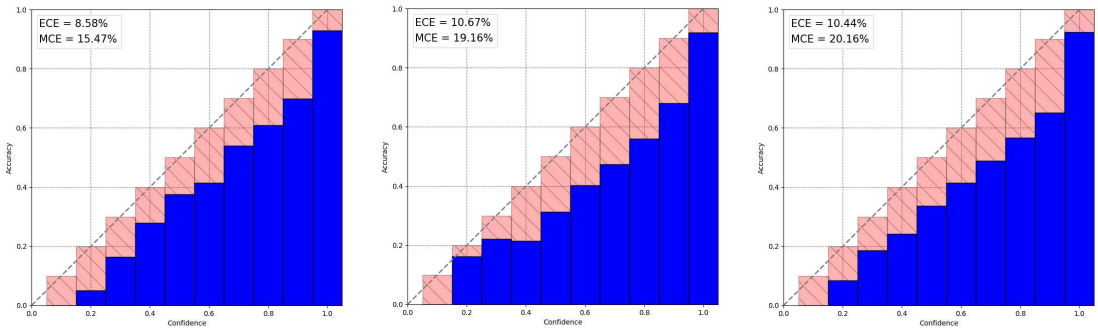
The maximum calibration error (MCE), see [17], can be computed similarly as

$$MCE = \max_{i \in \{1,2,\ldots,M\}} |acc_i - conf_i|,$$

which measures the absolute maximum difference between the accuracy and confidence of each bin. Note the lower the ECE and MCE, the better the model is calibrated. For our experiments, we choose $M = 10$ bins and compute both ECE and MCE, as seen in Fig. 4.5. We also show the corresponding reliability diagrams in Fig. 4.5, which show accuracy as a function of confidence. Note that a perfectly calibrated model will plot the identity.

We see in Fig. 4.5 that all models are overconfident in their predictions, as each accuracy bar lies under the identity plot. Comparing the ECE and MCE scores, we see that BNP has lower scores than both SGLD and pSGLD, with a lower score measuring a more calibrated model (hence a lower score is better). In Fig. 4.5 we see the accuracy of each bin in blue and the gap between a perfectly calibrated model in red. BNP has an ECE score of 8.58%, pSGLD an ECE score of 10.67% and BN a score of 10.44%. Therefore, our method of BNP achieves faster initial convergence and is more calibrated than comparative models on the residual network.

There are methods to improve the calibration of neural networks and relevant work contributing to these techniques includes temperature scaling [8], an extension of histogram binning to multi-class models [24], isotonic regression [24], and SWA-Gaussian [15], which builds on Stochastic Weight Averaging (SWA). However, we focus on computing ECE and MCE scores on these models as an additional metric of comparison without applying any calibration improvement techniques.



(a) BNP (b) pSGLD (c) SGLD

Figure 4.5: Reliability diagrams and corresponding ECE and MCE scores for BNP (left), pSGLD (center), and SGLD (right) on ResNet. The blue bars correspond to the accuracy of each bin, while the red shows the gap between a perfectly calibrated model and the given model.

4.3.1 Computation time

We expect the implementation of BNP over SGLD to be more expensive to compute. For a more thorough comparison, we test the computation time of BNP as in Algorithm 1 on ResNet 20 along with the baseline algorithms pSGLD and SGLD. These results are computed on GeForce GTX 1080 Ti. Results are summarized in Table 4.2. These computation times are gathered by averaging the training time over the first 20 epochs, hence results show the average time to train for one epoch. Both BNP and pSGLD add to the training time of the network, with BNP adding about 3.7 seconds per epoch and pSGLD adding about 1 second per epoch.

Table 4.2: Summarization of computation time comparison for ResNet 20, which shows the average time (in seconds) to run one epoch of each model.

Method	Computation Time
BNP	21.83
SGLD	18.10
pSGLD	19.04

5 Conclusion

We have introduced a batch normalization preconditioning algorithm for stochastic gradient Langevin dynamics that increases the convergence rate of training over SGLD and pSGLD. This is done by using the mini-batch statistics to construct a preconditioning matrix used to precondition the gradients during training. We apply the algorithm to a variety of Bayesian neural networks, showing faster convergence in the residual network and simulation example, as well as a more calibrated model in the ResNet, while also achieving improved results for Thompson sampling.

6 Supplemental material

We restate Theorem 3.1 and provide a proof. Note this result follows the proof idea given in [12].

Theorem 3.1. *Let* $-\log p((x,y)|w)$ *be the negative log-likelihood loss function defined from the output of a fully-connected multi-layer neural network* (2.5) *with parameter* w *for a single network input* x*. Consider the weight and bias parameters* $w_i^{(\ell)}, b_i^{(\ell)}$ *at the* ℓ*-th layer and let*

$$\widehat{w} = \begin{bmatrix} b_i^{(\ell)} \\ w_i^{(\ell)} \end{bmatrix}, \quad \widehat{h} = \begin{bmatrix} 1 \\ h^{(\ell-1)} \end{bmatrix}.$$

Write the likelihood $p((x,y)|\widehat{w})$ *as a function of* $\widehat{w}$ *through the activation* $a^{(\ell)} := \widehat{w}^T\widehat{h}$*, that is* $p((x,y)|\widehat{w}) = f(a^{(\ell)})$ *for some function* f*. When training over a mini-batch of* N *inputs*

$\{x_1, x_2, \ldots, x_N\}$, let $\{h_1^{(\ell-1)}, h_2^{(\ell-1)}, \ldots, h_N^{(\ell-1)}\}$ be the associated $h^{(\ell-1)}$ and let

$$\widehat{h}_j = \begin{bmatrix} 1 \\ h_j^{(\ell-1)} \end{bmatrix} \in \mathbb{R}^{m+1}.$$

Then the empirical Fisher information matrix with respect to $\widehat{w}$,

$$\mathcal{I}(\widehat{w}) := -\frac{1}{N} \sum_{j=1}^{N} \nabla_{\widehat{w}}^2 \log p\big((x_j, y_j)|\widehat{w}\big),$$

can be written as

$$\mathcal{I}(\widehat{w}) = -\widehat{H}^T S \widehat{H},$$

where

$$\widehat{H} = [e,\ H], \quad H = \begin{bmatrix} h_1^{(\ell-1)^T} \\ \vdots \\ h_N^{(\ell-1)^T} \end{bmatrix},$$

and

$$S = \frac{1}{N} \operatorname{diag}\left(\frac{f'(\widehat{w}^T \widehat{h}_j)^2}{f(\widehat{w}^T \widehat{h}_j)^2} - \frac{f''(\widehat{w}^T \widehat{h}_j)}{f(\widehat{w}^T \widehat{h}_j)} \right).$$

Proof. With

$$-\log\big(p((x_j, y_j)|\widehat{w})\big) = -\log\big(f(\widehat{w}^T \widehat{h}_j)\big),$$

we have the gradient with respect to $\widehat{w}$ is given by $-(f'(\widehat{w}^T \widehat{h}_j)/f(\widehat{w}^T \widehat{h}_j))\widehat{h}_j$. Computing the Hessian of the negative log-likelihood gives

$$\left(\frac{f'(\widehat{w}^T \widehat{h}_j)^2}{f(\widehat{w}^T \widehat{h}_j)^2} \widehat{h}_j - \frac{f''(\widehat{w}^T \widehat{h}_j)}{f(\widehat{w}^T \widehat{h}_j)} \widehat{h}_j \right) \widehat{h}_j^T.$$

So, the empirical Fisher information matrix can be written as

$$\mathcal{I}(\widehat{w}) = \frac{1}{N} \sum_{j=1}^{N} \left(\frac{f'(\hat{w}^T \hat{h}_j)^2}{f(\hat{w}^T \hat{h}_j)^2} - \frac{f''(\hat{w}^T \hat{h}_j)}{f(\hat{w}^T \hat{h}_j)} \right) \hat{h}_j \hat{h}_j^T.$$

Noting that $\hat{H}^T = [\hat{h_1}, \hat{h_2}, \ldots, \hat{h_N}]$, we write

$$\mathcal{I}(\hat{w}) = -\hat{H}^T S \hat{H},$$

where

$$S = \frac{1}{N} \operatorname{diag}\left(\frac{f'(\hat{w}^T \hat{h}_j)^2}{f(\hat{w}^T \hat{h}_j)^2} - \frac{f''(\hat{w}^T \hat{h}_j)}{f(\hat{w}^T \hat{h}_j)} \right).$$

The proof is complete. □

Acknowledgments

QY gratefully acknowledges the research support by NSF (Grants DMS-1821144, DMS-2208314). GL gratefully acknowledges the support of the National Science Foundation (Grants DMS-1555072, DMS-2053746, DMS-2134209), Brookhaven National Laboratory Subcontract 382247, and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program (Grants DE-SC0021142, DE-SC0023161).

References

[1] S. Ahn, A. Korattikara, and M. Welling, Bayesian posterior sampling via stochastic gradient Fisher scoring, *ICML*, 2012.
[2] T. Chen, E. B. Fox, and C. Guestrin, Stochastic gradient Hamiltonian Monte Carlo, *ICML*, 2014.
[3] A. S. Dalalyan and A. G. Karagulyan, User-friendly guarantees for the Langevin Monte Carlo with inaccurate gradient, *Stochastic Process. Appl.*, **129**(12):5278–5311, 2018.
[4] W. Deng, Q. Feng, L. Gao, F. Liang, and G. Lin, Non-convex learning via replica exchange stochastic gradient MCMC, *ICML*, 2020.
[5] W. Deng, S. Liang, B. Hao, G. Lin, and F. Liang, Interacting contour stochastic gradient Langevin dynamics, *ICLR*, 2022.
[6] W. Deng, G. Lin, and F. Liang, A contour stochastic gradient Langevin dynamics algorithm for simulations of multi-modal distributions, *NeurIPS*, 2020.
[7] M. Girolami and B. Calderhead, Riemann Manifold Langevin and Hamiltonian Monte Carlo methods, *J. R. Stat. Soc. Ser. B. Stat. Methodol.*, **73**(2):123–272, 2011.
[8] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, On calibration of modern neural networks, *ICML*, 2017.
[9] H. He, S. Zhao, Z. Tang, J. Ho, Y. Saad, and Y. Xi, An efficient nonlinear acceleration method that exploits symmetry of the Hessian, doi:10.48550/arXiv.2210.12573, 2022.
[10] H. He, S. Zhao, Y. Xi, J. C. Ho, and Y. Saad, GDA-AM: On the effectiveness of solving minimax optimization via Anderson acceleration, doi:10.48550/arxiv.2110.02457, 2021.
[11] S. Ioffe and C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *CoRR*, abs/1502.03167, 2015.
[12] S. Lange, K. Helfrich, and Q. Ye, Batch normalization preconditioning for neural network training, *J. Mach. Learn. Res.*, **23**(72):1–41, 2022.
[13] C. Li, C. Chen, D. Carlson, and L. Carin, Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks, *AAAI*, 2016.
[14] Y.-A. Ma, T. Chen, and E. B. Fox, A Complete Recipe for Stochastic Gradient MCMC, *NeurIPS*, 2015.
[15] W. Maddox, T. Garipov, P. Izmailov, D. P. Vetrov, and A. G. Wilson, A Simple Baseline for Bayesian Uncertainty in Deep Learning, *NeurIPS*, 2019.
[16] J. Mukhoti, P. K. Dokania, P. H. S. Torr, and Y. Gal, On Batch Normalisation for Approximate Bayesian Inference, In: *3rd Symposium on Advances in Approximate Bayesian Inference*, 2020.
[17] M. P. Naeini, G. F. Cooper, and M. Hauskrecht, Obtaining well calibrated probabilities using Bayesian binning, *AAAI*, 2015.
[18] S. Patterson and Y. W. Teh, Stochastic gradient Riemannian Langevin dynamics on the probability simplex, *NIPS*, 2013.
[19] M. Raginsky, A. Rakhlin, and M. Telgarsky, Non-convex Learning via Stochastic Gradient Langevin Dynamics: a Nonasymptotic Analysis, In: *Annual Conference on Learning Theory (COLT)*, 2017.
[20] C. Riquelme, G. Tucker, and J. Snoek, Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for Thompson sampling, *ICLR*, 2018.
[21] U. Şimşekli, R. Badeau, A. T. Cemgil, and G. Richard, Stochastic Quasi-Newton Langevin Monte Carlo, *ICML*, 2016.

[22] M. Teye, H. Azizpour, and K. Smith, Bayesian uncertainty estimation for batch normalized deep networks, *ICML*, 2018.
[23] M. Welling and Y. Teh, Bayesian learning via stochastic gradient Langevin dynamics, *ICML*, 2011.
[24] B. Zadrozny and C. Elkan, Transforming Classifier Scores into Accurate Multiclass Probability Estimates, *KDD*, 2002.
[25] Y. Zhang, P. Liang, and M. Charikar, A Hitting Time Analysis of Stochastic Gradient Langevin Dynamics, In: *Annual Conference on Learning Theory (COLT)*, 2017.