Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks*

Chunyuan Li[1], **Changyou Chen**[1†], **David Carlson**[2] **and Lawrence Carin**[1]
[1]Department of Electrical and Computer Engineering, Duke University
[2]Department of Statistics and Grossman Center, Columbia University
chunyuan.li@duke.edu, cchangyou@gmail.com, david.edwin.carlson@gmail.com, lcarin@duke.edu

Abstract

Effective training of deep neural networks suffers from two main issues. The first is that the parameter spaces of these models exhibit pathological curvature. Recent methods address this problem by using adaptive preconditioning for Stochastic Gradient Descent (SGD). These methods improve convergence by adapting to the local geometry of parameter space. A second issue is overfitting, which is typically addressed by early stopping. However, recent work has demonstrated that Bayesian model averaging mitigates this problem. The posterior can be sampled by using Stochastic Gradient Langevin Dynamics (SGLD). However, the rapidly changing curvature renders default SGLD methods inefficient. Here, we propose combining adaptive preconditioners with SGLD. In support of this idea, we give theoretical properties on asymptotic convergence and predictive risk. We also provide empirical results for Logistic Regression, Feedforward Neural Nets, and Convolutional Neural Nets, demonstrating that our preconditioned SGLD method gives state-of-the-art performance on these models.

Introduction

Deep Neural Networks (DNNs) have recently generated significant interest, largely due to their state-of-the-art performance on a wide variety of tasks, such as image classification (Krizhevsky, Sutskever, and Hinton 2012) and language modeling (Sutskever, Vinyals, and Le 2014). Despite this significant empirical success, it remains a challenge to effectively train DNNs. This is due to two main problems: (i) The function under consideration is often difficult to optimize and find a good local minima. It is believed that this is in large part due to the pathological curvature and highly non-convex nature of the function to be optimized (Dauphin et al. 2014). (ii) Standard optimization techniques lead to overfitting, typically addressed through early stopping (Srivastava et al. 2014).

A Bayesian approach for learning neural networks incorporates uncertainty into model learning, and can reduce overfitting (MacKay 1992). In fact, it is possible to view the standard dropout technique (Srivastava et al. 2014) as a form of Bayesian approximation that incorporates uncertainty (Gal and Ghahramani 2015; Kingma, Salimans, and Welling 2015). Many other recent works (Blundell et al. 2015; Hernández-Lobato and Adams 2015; Korattikara et al. 2015) advocate incorporation of uncertainty estimates during model training to help improve robustness and performance.

While a Bayesian approach ameliorates the overfitting issue in these complicated models, exact Bayesian inference in DNNs is generally intractable. Recently, several approaches have been proposed to approximate a Bayesian posterior for DNNs, including a stochastic variational inference (SVI) method "Bayes by Backprop" (BBB) (Blundell et al. 2015) and an online expectation propogation method (OEP) "probabilistic backpropagation" (PBP) (Hernández-Lobato and Adams 2015). These methods assume the posterior is comprised of separable Gaussian distributions. While this is a good choice for computational reasons, it can lead to unreasonable approximation errors and underestimation of model uncertainty.

A popular alternative to SVI and OEP is to use Stochastic Gradient Markov Chain Monte Carlo (SG-MCMC) methods to generate posterior samples (Welling and Teh 2011; Chen, Fox, and Guestrin 2014; Ding et al. 2014; Li et al. 2016). One of the most common SG-MCMC methods is the Stochastic Gradient Langevin Dynamics (SGLD) algorithm (Welling and Teh 2011). One merit of this approach is that it is highly scalable; it requires only the gradient on a small mini-batch of data, as in the optimization method Stochastic Gradient Descent (SGD). It has been shown that these MCMC approaches converge to the true posterior by using a slowly-decreasing sequence of step sizes (Teh, Thiéry, and Vollmer 2014; Chen, Ding, and Carin 2015). Costly Metropolis-Hasting steps are not required.

Unfortunately, DNNs often exhibit pathological curvature and saddle points (Dauphin et al. 2014), which render existing SG-MCMC methods inefficient. In the optimization literature, numerous approaches have been proposed to overcome this problem, including methods based on adapting a preconditioning matrix in SGD to the local geometry (Duchi, Hazan, and Singer 2011; Kingma and Ba 2015; Dauphin, de Vries, and Bengio 2015). These approaches

*Appendix is at https://sites.google.com/site/chunyuan24
†Corresponding author

estimate second-order information with trivial per-iteration overhead, have improved risk bounds in convex problems compared to SGD, and demonstrate improved empirical performance in DNNs. The idea of using geometry in SG-MCMC has been explored in many contexts (Ahn, Korattikara, and Welling 2012; Girolami and Calderhead 2011; Patterson and Teh 2013) and includes second-order approximations. Often, these approaches use the expected Fisher information, adding significant computational overhead. These methods lack the scalability necessary for learning DNNs, as discussed further below.

We combine adaptive preconditioners from optimization with SGLD, to improve SGLD efficacy. To note the distinction from SGLD, we refer to this as the Preconditioned SGLD method (pSGLD). This procedure is simple and adds trivial per-iteration overhead. We first show theoretical properties of this method, including bounds on risk and asymptotic convergence properties. We demonstrate improved efficiency of pSGLD by demonstrating an enhanced bias-variance tradeoff of the estimator for small problems. We further empirically demonstrate its application to several models and large datasets, including deep neural networks. In the DNN experiments, pSGLD outperforms the results based on standard SGLD from (Korattikara et al. 2015), both in terms of convergence speed and the test-set performance. Futher, pSGLD generates state-of-the-art performance for the examples tested.

Related Work

Various regularization schemes have been developed to prevent overfitting in neural networks, such as early stopping, weight decay, dropout (Srivastava et al. 2014), and dropconnect (Wan et al. 2013). Bayesian methods are appealing due to their ability to avoid overfitting by capturing uncertainty during learning (MacKay 1992). MCMC methods work by producing Monte Carlo approximations to the posterior, with asymptotic consistency (Neal 1995). Traditional MCMC methods use the full dataset, which does not scale to large data problems. A pioneering work in combining stochastic optimization with MCMC was presented in (Welling and Teh 2011), based on Langevin dynamics (Neal 2011). This method was referred to as Stochastic Gradient Langevin Dynamics (SGLD), and required only the gradient on mini-batches of data. The per-iteration cost of SGLD is nearly identical to SGD. Unlike SGD, SGLD can generate samples from the posterior by injecting noise into the dynamics. This encourages the algorithm to explore the full posterior, instead of simply converging to a maximum *a posterior* (MAP) solution. Later, SGLD was extended by (Ahn, Korattikara, and Welling 2012), (Patterson and Teh 2013) and (Korattikara et al. 2015). Furthermore, higher-order versions of the SGLD with momentum have also been proposed, including stochastic gradient Hamiltonian Monte Carlo (SGHMC) (Chen, Fox, and Guestrin 2014) and stochastic gradient Nose-Hoover Thermostats (SGNHT) (Ding et al. 2014).

It has been shown that incoporating higher-order gradient information helps train neural networks when employing optimization methods (Ngiam et al. 2011). However, calculations of higher-order information is often cumbersome in most models of interest. Methods such as quasi-Newton, and those approximating second-order gradient information, have shown promising results (Ngiam et al. 2011). An alternative to full quasi-Newton methods is to rescale parameters so that the loss function has similar curvature along all directions. This strategy has shown improved performance in Adagrad (Duchi, Hazan, and Singer 2011), Adadelta (Zeiler 2012), Adam (Kingma and Ba 2015) and RMSprop (Tieleman and Hinton 2012) algorithms. Recently, RMSprop has been explained as a diagonal preconditioner in (Dauphin, de Vries, and Bengio 2015). While relatively mature in optimization, these techniques have not been developed in sampling methods. In this paper, we show that rescaling the parameter updates according to geometry information can also improve SG-MCMC, in terms of both training speed and predictive accuracy.

Preliminaries

Given data $\mathcal{D} = \{\boldsymbol{d}_i\}_{i=1}^N$, the posterior of model parameters $\boldsymbol{\theta}$ with prior $p(\boldsymbol{\theta})$ and likelihood $\prod_{i=1}^N p(\boldsymbol{d}_i|\boldsymbol{\theta})$ is computed as $p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\boldsymbol{\theta}) \prod_{i=1}^N p(\boldsymbol{d}_i|\boldsymbol{\theta})$. In the optimization literature, the prior plays the role of a penalty that regularizes parameters, while the likelihood constitutes the loss function to be optimized. The task in optimization is to find the MAP estimate, $\boldsymbol{\theta}_{MAP} = \operatorname{argmax} \log p(\boldsymbol{\theta}|\mathcal{D})$. Let $\Delta\boldsymbol{\theta}_t$ denote the change in the parameters at time t. Stochastic optimization methods such as Stochastic Gradient Descent (SGD)[1] update $\boldsymbol{\theta}$ using the following rule:

$$\Delta\boldsymbol{\theta}_t = \epsilon_t \Big(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t) + \frac{N}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{d}_{t_i}|\boldsymbol{\theta}_t) \Big), \quad (1)$$

where $\{\epsilon_t\}$ is a sequence of step sizes, and $\mathcal{D}^t = \{\boldsymbol{d}_{t_1}, \cdots, \boldsymbol{d}_{t_n}\}$ a subset of $n < N$ data items randomly chosen from $\mathcal{D}$ at iteration t. The convergence of SGD has been established (Bottou 2004).

For DNNs, the gradient is calculated by backpropagation (Rumelhart, Hinton, and Williams 1986). One data item $\boldsymbol{d}_i \triangleq (x_i, y_i)$ may consist of input $x_i \in \mathbb{R}^D$ and output $y_i \in \mathcal{Y}$, with $\mathcal{Y}$ being the output space (*e.g.,* a discrete label space in classification). In the testing stage, the Bayesian predictive estimate for input x, is given by $p(y|x, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[p(y|x, \boldsymbol{\theta})]$. The MAP estimate simply approximates this expectation as $p(y|x, \mathcal{D}) \approx p(y|x, \boldsymbol{\theta}_{\text{MAP}})$, ignoring parameter uncertainty.

Stochastic sampling methods such as SGLD incorporate uncertainty into predictive estimates. SGLD samples $\boldsymbol{\theta}$ from the posterior distributions via a Markov Chain with steps:

$$\Delta\boldsymbol{\theta}_t \sim \mathcal{N}\left(\frac{\epsilon_t}{2} \Big(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t) + \frac{N}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{d}_{t_i}|\boldsymbol{\theta}_t) \Big), \epsilon_t \mathbf{I} \right), \quad (2)$$

with $\mathbf{I}$ denoting the identity matrix. It also uses mini-batches to take gradient descend steps at each iteration. Rates of convergence are proven rigorously in (Teh, Thiéry, and Vollmer

[1] For maximization, this is Stochastic Gradient *Ascent*. Here, we abuse notation because SGD is a more common term.

2014). Given a set of samples from the update rule (2), posterior distributions can be approximated via Monte Carlo approximations as $p(y|x,\mathcal{D}) \approx \frac{1}{T}\sum_{t=1}^{T} p(y|x,\boldsymbol{\theta}_t)$, where T is the number of samples.

Both stochastic optimization and stochastic sampling approaches have the requirement that the step sizes satisfy the the following assumption.[2]

Assumption 1 *The step sizes $\{\epsilon_t\}$ are decreasing, i.e., $0 < \epsilon_{t+1} < \epsilon_t$, with 1) $\sum_{t=1}^{\infty}\epsilon_t = \infty$; and 2) $\sum_{t=1}^{\infty}\epsilon_t^2 < \infty$.*

If these step-sizes are not satisfied in stochastic optimization, there is no guarantee of convergence because the gradient estimation noise is not eliminated. Likewise, in stochastic sampling, decreasing step-sizes are necessary for asymptotic consistency with the true posterior, where the approximation error is dominated by the natural stochasticity of Langevin dynamics (Welling and Teh 2011).

Preconditioned Stochastic Gradient Langevin Dynamics

As noted in the previous section, standard SGLD updates all parameters with the same step size. This could lead to slow mixing when the components of $\boldsymbol{\theta}$ have different curvature. Unfortunately, this is generally true in DNNs due to the composition of nonlinear functions at multiple layers. A potential solution is to employ a user-chosen preconditioning matrix $G(\boldsymbol{\theta})$ in SGLD (Girolami and Calderhead 2011). The intuition is to consider the family of probability distributions $p(\boldsymbol{d}|\boldsymbol{\theta})$ parameterised by $\boldsymbol{\theta}$ lying on a Riemannian manifold. One can use the non-Euclidean geometry implied by this manifold to guide the random walk of a sampler. For any probability distribution, the expected Fisher information matrix $\mathcal{I}_{\boldsymbol{\theta}}$ defines a natural Riemannian metric tensor. To further scale up the method to a general online framework stochastic gradient Riemannian Langevin dynamics (SGRLD) was suggested in (Patterson and Teh 2013). At position $\boldsymbol{\theta}_t$, it gives the step[3]:

$$\Delta\boldsymbol{\theta}_t \sim \frac{\epsilon_t}{2}\Big[G(\boldsymbol{\theta}_t)\Big(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{\theta}_t) + \frac{N}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{d}_{t_i}|\boldsymbol{\theta}_t)\Big) + \Gamma(\boldsymbol{\theta}_t)\Big] + G^{\frac{1}{2}}(\boldsymbol{\theta}_t)\mathcal{N}(0,\epsilon_t\mathbf{I}), \tag{3}$$

where $\Gamma_i(\boldsymbol{\theta}) = \sum_j \frac{\partial G_{i,j}(\boldsymbol{\theta})}{\partial\theta_j}$ describes how the preconditioner changes with respect to $\boldsymbol{\theta}_t$. This term vanishes in SGLD because the preconditioner of SGLD is a constant $\mathbf{I}$. Both the direction and variance in (3) depends on the geometry of $G(\boldsymbol{\theta}_t)$. The natural gradient in the SGRLD step takes the direction of steepest descent on a manifold. Convergence to the posterior is guaranteed (Teh, Thiéry, and Vollmer 2014; Chen, Ding, and Carin 2015) as long as step sizes satisfy Assumption 1.

Unfortunately, for many models of interest, the expected Fisher information is intractable. However, we note that any positive definite matrix defines a valid Riemannian manifold metric. Hence, we are not restricted to using the exact expected Fisher information. Preconditioning aims to constitute a local transform such that the rate of curvature is equal in all directions. Following this, we propose to use the same preconditoner as in RMSprop. This preconditioner is updated sequentially using only the current gradient information, and only estimates a diagonal matrix. It is given sequentially as,

$$G(\boldsymbol{\theta}_{t+1}) = \text{diag}\left(\mathbf{1} \oslash \left(\lambda\mathbf{1} + \sqrt{V(\boldsymbol{\theta}_{t+1})}\right)\right), \tag{4}$$

$$V(\boldsymbol{\theta}_{t+1}) = \alpha V(\boldsymbol{\theta}_t) + (1-\alpha)\bar{g}(\boldsymbol{\theta}_t;\mathcal{D}^t)\odot\bar{g}(\boldsymbol{\theta}_t;\mathcal{D}^t)\,, \tag{5}$$

where for notational simplicity, $\bar{g}(\boldsymbol{\theta}_t;\mathcal{D}^t) = \frac{1}{n}\sum_{i=1}^{n}\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{d}_{t_i}|\boldsymbol{\theta}_t)$, is the sample mean of the gradient using mini-batch $\mathcal{D}^t$, and $\alpha \in [0,1]$. Operators $\odot$ and $\oslash$ represent element-wise matrix product and division, respectively.

RMSprop utilizes magnitudes of recent gradients to construct a preconditioner. Flatter landscapes have smaller gradients while curved landscapes have larger gradients. Gradient information is usually only locally consistent. Therefore, two equivalent interpretations for Eq. (3) can be reached intuitively: *i*) the preconditioner equalizes the gradient so that a constant stepsize is adequate for all dimensions. *ii*) the stepsizes are adaptive, in that flat directions have larger stepsizes while curved directions have smaller stepsizes.

In DNNs, saddle points are the most prevalent critical points, that can considerably slow down training (Dauphin, de Vries, and Bengio 2015), mostly because the parameter space tends to be flat in many directions and ill-conditioned in the neighborhood of these saddle points. Standard SGLD will slowly escape the saddle point due to the typical oscillations along the high positive curvature direction. By transforming the landscape to be more equally curved, it is possible for the sampler to move much faster.

In addition, there are two tuning parameters: λ controls the extremes of the curvature in the preconditioner (default λ=10^{-5}), and α balances the weights of historical and current gradients. We use a default value of $\alpha = 0.99$ to construct an exponentially decaying sequence. Our Preconditioned SGLD with RMSprop is outlined in Algorithm 1.

Algorithm 1 Preconditioned SGLD with RMSprop

Inputs: $\{\epsilon_t\}_{t=1:T}, \lambda, \alpha$
Outputs: $\{\boldsymbol{\theta}_t\}_{t=1:T}$
Initialize: $\mathbf{V}_0 \leftarrow \mathbf{0}$, random $\boldsymbol{\theta}_1$
for $t \leftarrow 1 : T$ **do**
  Sample a minibatch of size n, $\mathcal{D}_n^t = \{\boldsymbol{d}_{t_1},\ldots,\boldsymbol{d}_{t_n}\}$
  Estimate gradient $\bar{g}(\boldsymbol{\theta}_t;\mathbf{X}^t) = \frac{1}{n}\sum_{i=1}^{n}\nabla\log p(\boldsymbol{d}_{t_i}|\boldsymbol{\theta}_t)$
  $V(\boldsymbol{\theta}_t) \leftarrow \alpha V(\boldsymbol{\theta}_{t-1}) + (1-\alpha)\bar{g}(\boldsymbol{\theta}_t;\mathcal{D}^t)\odot\bar{g}(\boldsymbol{\theta}_t;\mathcal{D}^t)$
  $G(\boldsymbol{\theta}_t) \leftarrow \text{diag}\left(\mathbf{1}\oslash\left(\lambda\mathbf{1}+\sqrt{V(\boldsymbol{\theta}_t)}\right)\right)$
  $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \frac{\epsilon_t}{2}\Big[G(\boldsymbol{\theta}_t)\Big(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{\theta}_t) + N\bar{g}(\boldsymbol{\theta}_t;\mathcal{D}^t)\Big) + \Gamma(\boldsymbol{\theta}_t)\Big] + \mathcal{N}(0,\epsilon_t G(\boldsymbol{\theta}_t))$
end for

[2]The requirement for SGLD can be relaxed, see (Teh, Thiéry, and Vollmer 2014; Chen, Ding, and Carin 2015) for more details.

[3]The update form in (Patterson and Teh 2013) is more complicated and seemingly different from (3); however, they can be shown to be equivalent.

Preconditioned SGLD Algorithms in Practice

This section first analyzes the finite-time convergence properties of pSGLD, then proposes a more efficient variant for practical use. We note that prior work gave similar theoretical results (Chen, Ding, and Carin 2015), and we extend the theory to consider the use of preconditioners.

Finite-time Error Analysis

For a bounded function $\phi(\boldsymbol{\theta})$, we are often interested in its true posterior expectation $\bar{\phi} = \int_{\mathcal{X}} \phi(\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathcal{D}) d\boldsymbol{\theta}$. For example, the class distribution of a data point in DNNs. In our SG-MCMC based algorithm, this intractable integration is approximated by a weighted sample average $\hat{\phi} = \frac{1}{S_T}\sum_{t=1}^{T} \epsilon_t \phi(\boldsymbol{\theta}_t)$ at time $S_T = \sum_{t=1}^{T} \epsilon_t$, with stepsizes $\{\epsilon_t\}$. These samples are generated from an MCMC algorithm with a numerical integrator (*e.g.*, our pSGLD algorithm) that discretizes the continuous-time Langevin dynamics. The precision of the true posterior average and its MCMC approximation is characterized by the expected difference between $\bar{\phi}$ and $\hat{\phi}$. We analyze the pSGLD algorithm by extending the work of (Teh, Thiéry, and Vollmer 2014; Chen, Ding, and Carin 2015) to include adaptive preconditioners. We first show the asymptotic convergence properties of our algorithm in Theorem 1 by the mean of the mean squared error (MSE)[4]. To get the convergence result, some mild assumptions on the smoothness and boundness of ψ, the solution functional of $\mathcal{L}\psi = \phi(\boldsymbol{\theta}_t) - \bar{\phi}$, is needed, where $\mathcal{L}$ is the generator of corresponding stochastic differential equation for pSGLD. We discuss these conditions and prove the Theorem in Appendix A.

Theorem 1 *Define the operator* $\Delta V_t = (N\bar{g}(\boldsymbol{\theta}_t; \mathcal{D}^t) - g(\boldsymbol{\theta}_t; \mathcal{D}^t))^\top G(\boldsymbol{\theta}_t)\nabla_{\boldsymbol{\theta}}$. *Under Assumption 1, for a test function* ϕ, *the MSE of the pSGLD at finite time* S_T *is bounded, for some* $C > 0$ *independent of* $\{\epsilon_t\}$, *as:*

$$\textit{MSE}: \mathbb{E}\left[\left(\hat{\phi} - \bar{\phi}\right)^2\right] \leq \mathcal{B}_{\textit{mse}} \triangleq C\left(\sum_t \frac{\epsilon_t^2}{S_T^2}\mathbb{E}\left\|\Delta V_t\right\|^2 + \frac{1}{S_T} + \frac{(\sum_{t=1}^{T}\epsilon_t^2)^2}{S_T^2}\right). \tag{6}$$

MSE is a common measure of quality of an estimator, reflecting the precision of an approximate algorithm. It can be seen from Theorem 1 that the finite-time approximation error of pSGLD is bounded by $\mathcal{B}_{\text{mse}}$, consisting of two factors: (i) estimation error from stochastic gradients, $\sum_t \frac{\epsilon_t^2}{S_T^2}\mathbb{E}\left\|\Delta V_t\right\|^2$, and (ii) discretization error inherited from numerical integrators, $\frac{1}{S_T} + \frac{(\sum_{t=1}^{T}\epsilon_t^2)^2}{S_T^2}$. These terms asymptotically approach 0 under Assumption 1, meaning that the decreasing-step-size pSGLD is asymptotically consistent with true posterior expectation.

[4]This is different from the optimization literature where the *regret* is studied, which is not straightforward in the MCMC framework.

Practical Techniques

Of interest when considering the practical issue of limited computation time, we now interpret the above finite-time error using the framework of *risk of an estimator*, which provides practical guidance in implementation. From (Korattikara, Chen, and Welling 2014), the predictive risk, R, of an algorithm is defined as the MSE above, and can be decomposed as $R = \mathbb{E}[(\bar{\phi} - \hat{\phi})^2] = B^2 + V$, where B is the bias and V is the variance. Denote $\bar{\phi}_\eta = \int_{\mathcal{X}} \phi(\boldsymbol{\theta})\rho_\eta(\boldsymbol{\theta})d\boldsymbol{\theta}$ as the ergodic average under the invariant measure, $\rho_\eta(\boldsymbol{\theta})$, of the pSGLD. After burnin, it can be shown that

$$\text{Bias}: \; B = \bar{\phi}_\eta - \bar{\phi} \tag{7}$$

$$\text{Variance}: \; V = \mathbb{E}[(\bar{\phi}_\eta - \hat{\phi})^2] \approx \frac{A(0)}{M_\eta} \tag{8}$$

where $A(0)$ is the variance of ϕ with respect to $\rho_\eta(\boldsymbol{\theta})$ (*i.e.,* $\mathbb{E}_{\rho_\eta(\boldsymbol{\theta})}[(\phi - \hat{\phi})^2]$) , which is a constant (further details are given in Appendix D). M_η is the *effective sample size* (ESS), defined as

$$\text{ESS}: \; M_\eta = \frac{T}{1 + 2\sum_{t=1}^{\infty}\frac{A(t)}{A(0)}} = \frac{T}{2\tau} \tag{9}$$

where $A(t) = \mathbb{E}[(\bar{\phi}_\eta - \phi(\boldsymbol{\theta}_0))(\bar{\phi}_\eta - \phi(\boldsymbol{\theta}_t))]$ is the *autocovariance function*, manifesting how strong two samples with a time lag t are correlated. The term $\tau = \frac{1}{2} + \sum_{t=1}^{\infty}\frac{A(t)}{A(0)}$ is the integrated *autocorrelation time* (ACT), which measures the interval between independent samples.

In practice, there is always a tradeoff between bias and variance. In the case of infinite computation time, the traditional MCMC setting can reduce the bias and variance to zero. However, in practice, time is limited. Obtaining more effective samples can reduce the total risk (Eq. (6)), even if bias is introduced. In the following, we provide two model-independent practical techniques to further speed up the proposed pSGLD.

Excluding $\Gamma(\boldsymbol{\theta}_t)$ term Though the evaluation of $\Gamma(\boldsymbol{\theta}_t)$ in our case is manageable due to its diagonal nature, we propose to remove it during sampling to reduce the computation. It is interesting that in our case ignoring $\Gamma(\boldsymbol{\theta}_t)$ produces a bias controlled by α on the MSE.

Corollary 2 *Assume the 1st-order and 2nd-order gradients are bounded. With the same assumptions as Theorem 1, the MSE when ignoring the* $\Gamma(\boldsymbol{\theta}_t)$ *term in the algorithm can be bounded as* $\mathbb{E}\left[\left(\hat{\phi} - \bar{\phi}\right)^2\right] \leq \mathcal{B}_{\textit{mse}} + O\left(\frac{(1-\alpha)^2}{\alpha^3}\right)$, *where* $\mathcal{B}_{\textit{mse}}$ *is the bound defined in Theorem 1.*

Omitting $\Gamma(\boldsymbol{\theta}_t)$ introduces an extra term in the bound that is controlled by the parameter α. The proof is in Appendix B. Since α is always set to a value that is very close to 1, the term $(1-\alpha)^2/\alpha^3 \approx 0$, the effect of $\Gamma(\boldsymbol{\theta}_t)$ negligible. In addition, more samples per unit time are generated when $\Gamma(\boldsymbol{\theta}_t)$ is ignored, resulting in a smaller variance on the prediction. Note that the term $\Gamma(\boldsymbol{\theta}_t)$ is heuristically ignored in (Ahn, Korattikara, and Welling 2012), but is only able to approximate the true posterior in the case of infinite data, which is not required in our algorithm.

Thinning samples Making predictions using a whole ensemble of models is cumbersome and may be too computationally expensive to allow deployment for a large number of users, especially when models are large neural nets. One practical technique is to average models using a thinned version of samples. By thinning the samples in pSGLD, the total number of samples is reduced. However, these thinned samples have a lower autocorrelation time and a similar ESS. We can guarantee the MSE remains the same form under the thinning schema. The proof is in Appendix C.

Corollary 3 *By thinning samples from our pSGLD algorithm, the MSE remains the same form as in Theorem 1, and asymptotically approaches 0.*

Experiments

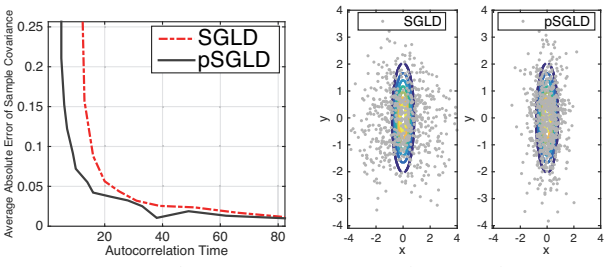
We present results in four parts: a simple simulation, Bayesian Logistic Regression (BLR), and two widely used DNN models, Feedforward Neural Networks (FNN) and Convolutional Neural Networks (CNN).

The proposed algorithm that uses the discussed practical techniques is denoted as *pSGLD*. The prior on the parameters is set to $p(\boldsymbol{\theta}) = \mathcal{N}(0, \sigma^2\mathbf{I})$. If not specifically mentioned, the default setting for DNN experiments is shared as follows. $\sigma^2 = 1$, minibatch size is 100, thinning interval is 100, burn-in is 300. We employ a block decay strategy for stepsize; it decreases by half after every L epochs.

Simulation

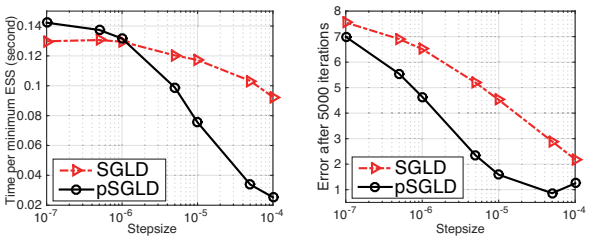
We first demonstrate pSGLD on a simple 2D Gaussian example, $\mathcal{N}(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.16 & 0 \\ 0 & 1 \end{bmatrix})$. Given posterior samples, the goal is to estimate the covariance matrix. A diagonal covariance matrix is used to show the algorithm can adjust the stepsize at different dimension.

A large range of stepsizes are used. 2×10^5 samples are collected. Reconstruction errors and autocorrelation time are shown in Fig. 1 (a). We see that pSGLD dominates the "vanilla" SGLD in that it consistently shows a lower error and autocorrelation time, particularly with larger stepsize. When the stepsize is small enough, the sampler does not move much, and the performances of the two algorithms become similar. The first 600 samples of both methods for $\epsilon = 0.3$ are shown in Fig. 1 (b). Because step sizes in pSGLD can be adaptive, it implies that even if the covariance



(a) Error and ACT (b) Samples

Figure 1: Simulation results on a 2D Gaussian.



(a) Variance (b) Parameter estimation

Figure 2: BLR on Australian dataset.

matrix of a target distribution is mildly rescaled, a new stepsize is unnecessary for pSGLD. Meanwhile, the stepsize of the vanilla SGLD needs to be fine-tuned in order to obtain decent samples. See Appendix E for further details.

Bayesian Logstic Regression

We demonstrate pSGLD on BLR. A small Australian dataset (Girolami and Calderhead 2011) is first used with $N = 690$ and dimension $D = 14$. We choose a minibatch size of 5, $\sigma^2 = 100$, $T = 5 \times 10^3$. We test stepsize ϵ ranging from 1×10^{-7} to 1×10^{-4}, with 50 runs for each algorithm. Following (Girolami and Calderhead 2011), we report the time per minimum Effective Sample ($\propto 1/\text{EES}$) in Fig. 2 (a), which is proportional to the variance. pSGLD generates much larger ESS compared to SGLD, especially when the stepsize is large. Meanwhile, Fig. 2 (b) shows that pSGLD provides smaller error in estimating weights, where the "ground truth" is obtained by 10^6 samples from HMC with Metroplis-Hastings. Therefore, the overall risk is reduced.

We then test BLR on a large-scale Adult dataset, `a9a` (Lin, Weng, and Keerthi 2008), with $N_{\text{train}} = 32561, N_{\text{test}} = 16281$, and $D = 123$. Minibatch size is set to 50, $\sigma^2 = 10$. The thinning interval is 50, burn-in is 500, and $T = 1.5 \times 10^4$. Stepsize $\epsilon = 5 \times 10^{-2}$ for pSGLD and SGLD. The test errors are compared in Table 1, and learning curves are shown in Fig. 3. Both SG-MCMC methods outperform the recently proposed doubly stochastic variational Bayes (SDVI) (Titsias and Lázaro-Gredilla 2014), and higher-order variational autoencoder methods (L-BFGS-SGVI, HFSGVI) (Fan et al. 2015). Furthermore, pSGLD converges in less than 4×10^3 iterations, while SGLD at least needs double the time to reach this accuracy.

Method	Test error
pSGLD	**14.85%**
SGLD	**14.85%**
DSVI	15.20%
L-BFGS-SGVI	14.91%
HFSGVI	15.16%

Table 1: BLR on `a9a`.

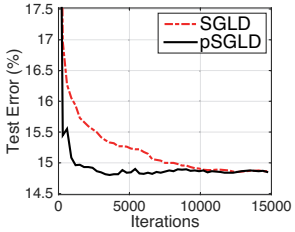


Figure 3: Learning curves.

Feedforward Neural Networks

The first DNN model we study is the FNN. The activation function is rectified linear unit (ReLU). A two-layer model is

Table 2: Classification error of FNN on MNIST. [$\diamond$] indicates results taken from (Blundell et al. 2015)

Method	Test Error		
	400-400	800-800	1200-1200
pSGLD ($\sigma^2 = 100$)	**1.40%**	**1.26%**	**1.14%**
pSGLD ($\sigma^2 = 1$)	1.45%	1.32%	1.24%
distilled pSGLD	1.44%	1.40%	1.41%
SGLD	1.64%	1.41%	1.40%
RMSprop	1.59%	1.43%	1.39%
RMSspectral	1.65%	1.56%	1.46%
SGD	1.72%	1.47%	1.47%
BPB, Gaussian$^\diamond$	1.82%	1.99%	2.04%
BPB, Scale mixture$^\diamond$	1.32%	1.34%	1.32%
SGD, dropout$^\diamond$	1.51%	1.33%	1.36%

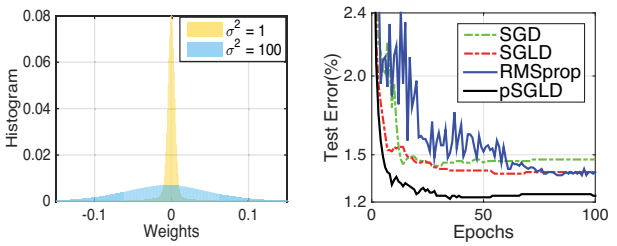
employed. 100 epochs are used, with $L = 20$. We compare our propose method, pSGLD, with representative stochastic optimization methods: SGD, RMSprop and RMSspectral (Carlson et al. 2015). After tuning, we set the optimal stepsize for each algorithm as: for pSGLD and RSMprop as $\epsilon = 5 \times 10^{-4}$, while for SGLD and SGD as $\epsilon = 5 \times 10^{-1}$.

We test on MNIST dataset, consisting of 28×28 images from 10 classes with $60,000$ training and $10,000$ test samples. The test classification errors for network size 400-400, 800-800 and 1200-1200 are shown in Table 2. The results of stochastic sampling methods are better than their corresponding optimization counterparts. This indicates that incorporating weight uncertainty can improve performances. By increasing the variance σ^2 of pSGLD from 1 to 100, more uncertainty is introduced into the model from the prior, and higher performance is obtained. Figure 4 (a) displays the histograms of weights in the last training iteration of the 1200-1200 model. We observe that smaller variance in the prior imposes lower uncertainty, by making the weights concentrate to 0; while larger variance in the prior leads to a wider range of weight choices, thus higher uncertainty.

We also compare to other techniques developed to prevent overfitting (dropout) and weight uncertainty (BPB, Gaussian and scale mixtures). pSGLD provides state-of-the-art performance for FNN on test accuracy. We further note that pSGLD is able to give increasing performance with increasing network size, whereas BPB and SGD dropout do not.

Finally, learning curves of network configuration 1200-1200 are plot in Fig. 4 (b)[5]. We empirically find that pSGLD and SGLD take fewer iterations to converge. Moreover, pSGLD consistently converges faster and to a better point than SGLD. Learning curves for other network sizes are provided in Appendix F. While the ensemble of samples requires more computation than a single FNN in testing, it shows significantly improved test set performance. As well, (Korattikara et al. 2015) showed that learning a single FNN that approximates the model average result gave nearly the same performance. We employ this idea, and suggest a fast version, *distilled pSGLD*. Its results for $\sigma^2 = 1$ show it maintains good performances.

[5]RMSspectral is not shown because it uses larger batch sizes and so is difficult to compare on this scale.



(a) Weights distribution (b) Learning curves

Figure 4: FNN of size 1200-1200 on MNIST.

Convolutional Neural Networks

Our next DNN is the popular CNN model. We use a standard network configuration with 2 convolutional layers followed by 2 fully-connected layers (Jarrett et al. 2009). Both convolutional layers use 5×5 filter size with 32 and 64 channels, respectively; 2×2 max pooling is used after each convolutional layer. The fully-connected layers have 200-200 hidden nodes with ReLU, 20 epochs are used, and L is set to 10. Additional results with CNNs are in Appendix G.

The same MNIST dataset is used. A comparison of test errors is shown in Table 3, with the corresponding learning curves in Fig. 5. We emphasize that the purpose of this experiment is to compare methods on the same model architecture, not to achieve overall state-of-the-art results. The CNN trained with traditional SGD gives an error of 0.82%. pSGLD shows significant improvement, with an error of 0.45%. This result is also comparable with some recent state-of-the-art CNN based systems, which have much more complex architectures. These include the stochastic pooling (Zeiler and Fergus 2013), Network in Network (NIN) (Lin, Chen, and Yan 2014) and Maxout Network(MN) (Goodfellow et al. 2013).

Method	Test error
pSGLD	**0.45%**
SGLD	0.71%
RMSprop	0.65%
RMSspectral	0.78%
SGD	0.82%
Stochastic Pooling	0.47%
NIN + Dropout	0.47%
MN + Dropout	0.45%

Table 3: Test error.

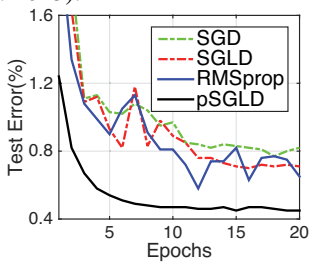


Figure 5: Learning curves.

Conclusion

A preconditioned SGLD is developed based on the RMSprop algorithm, with controllable finite-time approximation error. We apply the algorithm to DNNs to overcome their notorious problems of overfitting and pathological curvature. Extensive experiments show that our pSGLD can adaptive to the local geometry, allowing improved effective sampling rates and performance. It provides sample-based uncertainty in DNNs, and achieves state-of-the-arts performances on FNN and CNN models. Interesting future directions include exploring applications to latent variable models or recurrent neural networks (Gan et al. 2015).

Acknowledgements This research was supported in part by ARO, DARPA, DOE, NGA, ONR and NSF.

References

Ahn, S.; Korattikara, A.; and Welling, M. 2012. Bayesian posterior sampling via stochastic gradient fisher scoring. In *ICML*.

Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; and Wierstra, D. 2015. Weight uncertainty in neural networks. In *ICML*.

Bottou, L. 2004. Stochastic learning. *Advanced Lectures on Machine Learning* 146–168.

Carlson, D.; Collins, E.; Hsieh, Y. P.; Carin, L.; and Cevher, V. 2015. Preconditioned spectral descent for deep learning. In *NIPS*.

Chen, C.; Ding, N.; and Carin, L. 2015. On the convergence of stochastic gradient MCMC algorithms with high-order integrators. In *NIPS*.

Chen, T.; Fox, E. B.; and Guestrin, C. 2014. Stochastic gradient Hamiltonian Monte Carlo. In *ICML*.

Dauphin, Y. N.; Pascanu, R.; Gulcehre, C.; Cho, K.; Ganguli, S.; and Bengio, Y. 2014. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NIPS*.

Dauphin, Y. N.; de Vries, H.; and Bengio, Y. 2015. Equilibrated adaptive learning rates for non-convex optimization. In *NIPS*.

Ding, N.; Fang, Y.; Babbush, R.; Chen, C.; Skeel, R. D.; and Neven, H. 2014. Bayesian sampling using stochastic gradient thermostats. In *NIPS*.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*.

Fan, K.; Wang, Z.; Beck, J.; Kwok, J.; and Heller, J. 2015. Fast second-order stochastic backpropagation for variational inference. In *NIPS*.

Gal, Y., and Ghahramani, Z. 2015. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. *arXiv:1506.02142*.

Gan, Z.; Li, C.; Henao, R.; Carlson, D.; and Carin, L. 2015. Deep temporal sigmoid belief networks for sequence modeling. *NIPS*.

Girolami, M., and Calderhead, B. 2011. Riemann manifold langevin and hamiltonian monte carlo methods. In *JRSS: Series B*.

Goodfellow, I.; Warde-farley, D.; Mirza, M.; Courville, A.; and Bengio, Y. 2013. Maxout networks. In *ICML*.

Hernández-Lobato, J. M., and Adams, R. P. 2015. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *ICML*.

Jarrett, K.; Kavukcuoglu, K.; Ranzato, M.; and LeCun, Y. 2009. What is the best multi-stage architecture for object recognition? In *ICCV*.

Kingma, D., and Ba, J. 2015. Adam: A method for stochastic optimization. *ICLR*.

Kingma, D. P.; Salimans, T.; and Welling, M. 2015. Variational dropout and the local reparameterization trick. *NIPS*.

Korattikara, A.; Rathod, V.; Murphy, K.; and Welling, M. 2015. Bayesian dark knowledge. *NIPS*.

Korattikara, A.; Chen, Y.; and Welling, M. 2014. Austerity in MCMC land: Cutting the Metropolis-Hastings budget. *ICML*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Li, C.; Chen, C.; Fan, K.; and Carin, L. 2016. High-order stochastic gradient thermostats for Bayesian learning of deep models. In *AAAI*.

Lin, M.; Chen, Q.; and Yan, S. 2014. Network in network. *ICLR*.

Lin, C.-J.; Weng, R. C.; and Keerthi, S. S. 2008. Trust region newton method for logistic regression. *JMLR*.

MacKay, D. J. C. 1992. A practical bayesian framework for backpropagation networks. *Neural computation*.

Neal, R. M. 1995. *Bayesian learning for neural networks*. PhD thesis, University of Toronto.

Neal, R. M. 2011. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*.

Ngiam, J.; Coates, A.; Lahiri, A.; Prochnow, B.; Le, Q. V.; and Ng, A. Y. 2011. On optimization methods for deep learning. In *ICML*.

Patterson, S., and Teh, Y. W. 2013. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. In *NIPS*.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. 1986. Learning representations by back-propagating errors. *Nature*.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

Teh, Y. W.; Thiéry, A. H.; and Vollmer, S. J. 2014. Consistency and fluctuations for stochastic gradient Langevin dynamics.

Tieleman, T., and Hinton, G. E. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural Networks for Machine Learning*.

Titsias, M., and Lázaro-Gredilla, M. 2014. Doubly stochastic variational bayes for non-conjugate inference. In *ICML*.

Wan, L.; Zeiler, M.; Zhang, S.; LeCun, Y.; and Fergus, R. 2013. Regularization of neural networks using dropconnect. In *ICML*.

Welling, M., and Teh, Y. W. 2011. Bayesian learning via stochastic gradient Langevin dynamics. In *ICML*.

Zeiler, M., and Fergus, R. 2013. Stochastic pooling for regularization of deep convolutional neural networks. *ICLR*.

Zeiler, M. D. 2012. Adadelta: An adaptive learning rate method. *arXiv:1212.5701*.