Stein Variational Gradient Descent with Variance Reduction

Nhan Dam, Trung Le, Viet Huynh, and Dinh Phung
Monash University, Australia
Email: {nhan.dam, trunglm, viet.huynh, dinh.phung}@monash.edu

***Abstract*—Probabilistic inference is a common and important task in statistical machine learning. The recently proposed Stein variational gradient descent (SVGD) is a generic Bayesian inference method that has been shown to be successfully applied in a wide range of contexts, especially in dealing with large datasets, where existing probabilistic inference methods have been known to be ineffective. In a large-scale data setting, SVGD employs the mini-batch strategy but its mini-batch estimator has large variance, hence compromising its estimation quality in practice. To this end, we propose in this paper a generic SVGD-based inference method that can significantly reduce the variance of mini-batch estimator when working with large datasets. Our experiments on 14 datasets show that the proposed method enjoys substantial and consistent improvements compared with baseline methods in binary classification task and its pseudo-online learning setting, and regression task. Furthermore, our framework is generic and applicable to a wide range of probabilistic inference problems such as in Bayesian neural networks and Markov random fields.**

***Index Terms*—Bayesian inference, variance reduction, statistical machine learning**

I. Introduction

In statistical machine learning, a very common problem is probabilistic inference with probability distributions that are factorisable. For example, in probabilistic models such as mixture models or latent Dirichlet allocation [1], it is desirable to compute the posterior of the latent variables given observed variables, and this posterior can be factorised into prior and joint likelihood. Usually our target distributions are intractable, thus an effective approximate inference is demanded.

Two popular approximate inference methods are *Markov chain Monte Carlo* (MCMC) and *variational inference* (VI). MCMC estimates the exact distribution via drawing samples. In general, this method is slow due to the fact that we can only draw a single sample at a time as the current sample depends on the past sample. Recent work has tried to scale up MCMC to work with large datasets which reduces the computational cost in each iteration by estimating noisy gradients with subsamples of data [2], [3], [4] or selecting a subset of data [5]. However, MCMC methods are challenging to evaluate the convergence which hinders their applications in practice. In contrast, VI can be efficiently accompanied by stochastic gradient descent to train on large datasets [6], which is more popular in practice. In VI, we find a member in a family of distributions $\mathcal{S}$ that best approximates our target distribution. Crucially, we need to consider the trade-off between accuracy and computational complexity when designing $\mathcal{S}$. Hence, one often determines $\mathcal{S}$ in the problem-specific manner, which may require domain expert knowledge and profound experience.

The authors of [7] partially lifted this burden by proposing a general purpose Bayesian inference algorithm called *Stein variational gradient descent* (SVGD). SVGD approximates a target distribution P via drawing particles that are first randomly sampled from an initial simple distribution Q_0 and then moved step by step towards P via simple invertible transformations. Besides only requiring an unnormalised version of the distribution P for moving particles, as stated by its authors, SVGD can be considered 'as a natural counterpart of gradient descent for full Bayesian inference' [7] which enables gradually updating particles in an economical manner. In particular, when the target distribution can be factorised into a multiplication of many component distributions, SVGD can update its particles in the stochastic manner using mini-batches. However, by nature, this traditional mini-batch estimator in the general context (i.e. not only when specifically applied in SVGD) has high variance [8], [9].

Since MCMC and VI methods for large-scale datasets use noisy gradients in their algorithm which can lead to the high variance of the gradient estimate, the following papers have been proposed to control the variance of the estimator for MCMC [10], [11], [12] as well as VI [13], [14], [15]. However, the variance reduction for SVGD is still an open problem.

In this paper, to address the high variance issue in mini-batch estimator occurred in SVGD when the target distribution is factorisable, we propose *Stein variational gradient descent with variance reduction* (SVGD-VR), which can be regarded as a substantial improvement to SVGD inspired by *stochastic variance reduced gradient* (SVRG) [8]. Our method yields an unbiased variance-reduced mini-batch estimator, which greatly reduces the variance of the mini-batch estimator in SVGD. As a consequence, we significantly improve SVGD in many aspects: faster convergence, more stable training and more robust to batch size.

In the binary classification task, besides outperforming and being more stable than SVGD and SVRG with a standard batch size (i.e. 128), we empirically observe that our proposed SVGD-VR is very stable and achieves impressive predictive performance even with tiny batch sizes (e.g. 1, 2, and 4). Thus, to demonstrate the generalisation capability, we compare our proposed SVGD-VR with SVGD and SVRG in the pseudo-online binary classification inspired by the online learning in [16], [17] (i.e. the extreme case with the batch size of 1). Additionally, the versatile applicability of SVGD-VR in

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

practice is even more persuasive by the fact that it not only outperforms but also yields more stable training than SVGD on the regression task with Bayesian neural networks on 9 over 10 chosen datasets. Our code is available on GitHub[1].

This paper is organised as follows. Section II introduces background on SVGD and variance reduction methods. Section III is dedicated to the details of our proposed method. We report experimental results in Section IV and finally draw conclusion and future developments in Section V.

II. Related Background

A. Stein Variational Gradient Descent

Stein variational gradient descent (SVGD) [7] is a general purpose method to approximate a target distribution $p(\theta)$ by particles. We denote the approximate distribution as $q(\theta)$. We want to represent the target distribution via a set of samples $\theta_1, \ldots, \theta_M$. Initially, we draw M particles $\theta_1, \ldots, \theta_M$ from an initial distribution $q_0(\theta)$ that is usually a simple and easy-to-sample one. In SVGD, the task of learning a transformation $\tilde{T}$ that transports the initial distribution q_0 to the target distribution p is decomposed into learning a sequence of simple invertible mappings $T^{(1)}, \ldots, T^{(L)}$ (i.e. $\tilde{T}(\theta) = T^{(L)}\left(T^{(L-1)}\left(\ldots T^{(1)}(\theta)\right)\right)$) wherein at each step the induced distribution $q_{[T]}$ is pushed closer to the target distribution in terms of Kullback-Leibler (KL) divergence. The authors of SVGD also proposed to use the perturbation of identity mapping as $T(\theta) = \theta + \epsilon\Phi(\theta)$, where $\Phi(\theta)$ is a smooth function in a function class $\mathcal{F}$ and $|\epsilon|$ is a small number. When we consider the function class $\mathcal{F}$ as a ball in the product space of reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ associated with a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, i.e. $\mathcal{F} = \left\{\Phi \in \mathcal{H}_k^d : \|\Phi\|_{\mathcal{H}_k^d} \leq \sqrt{\mathbb{S}(q,p)}\right\}$, where $\mathbb{S}(q,p)$ is the kernelised Stein discrepency between two distributions p and q [18], we obtain the optimal perturbation at each time step as follows: $\Phi^*(\cdot) = \frac{1}{M}\sum_{m=1}^{M}\left[k(\theta_m, \cdot)\nabla_{\theta_m}\log p(\theta_m) + \nabla_{\theta_m}k(\theta_m, \cdot)\right]$. As we iterate this transformation, the KL divergence between the approximate distribution and the target distribution gradually decreases. We can see that this procedure can work regardless of the choice of the initial approximate distribution q_0. When the number of particles is only 1, this algorithm reduces to gradient ascent for maximum a posteriori (MAP) [19].

Dual forces: In the equation of Φ^*, the first term moves the particles towards the areas with high probability density $p(\theta)$ whilst the second term exerts the repulsive force that avoids mode collapse.

SVGD has a wide range of applications such as Bayesian logistic regression and Bayesian neural networks [7], restricted Boltzmann machines and Gaussian process classification [20], image denoising [21], moment matching [22], variational autoencoder learning [23], automatic neural samplers [24], distributed inference on continuous graphical models [25], and Bayesian kernel learning for big data [19].

B. Variance Reduction Methods

Many optimisation problems can be formulated as $\min_\theta F(\theta)$, where $F(\theta) \triangleq \frac{1}{N}\sum_{i=1}^{N} f_i(\theta)$. A simple yet successful approach is *gradient descent* (GD), in which we update θ iteratively as $\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla F\left(\theta^{(t)}\right) = \theta^{(t)} - \frac{\eta_t}{N}\sum_{i=1}^{N}\nabla f_i\left(\theta^{(t)}\right)$, with $\theta^{(t)}$ and η_t denoting the values of θ and the learning rate at t^{th} update step respectively. To work with large N, we may utilise *mini-batch GD* or *stochastic GD* (SGD). In mini-batch GD, we approximate $\nabla F(\theta)$ by $\frac{1}{n}\sum_{j=1}^{n}\nabla f_{i_j}(\theta)$, where $\{i_1, \ldots, i_n\} \subset \{1, \ldots, N\}$ with $n \ll N$. When $n = 1$, mini-batch GD becomes SGD. Although mini-batch GD and SGD yield unbiased estimators, they do introduce additional variance to the estimation [8], [9]. Consequently, we often need to decay the learning rate to achieve convergence.

Some methods have been proposed to reduce the variance in the mini-batch gradient estimator such as *stochastic average gradient* (SAG) [26], *stochastic dual coordinate ascent* (SDCA) [27], *stochastic variance reduced gradient* (SVRG) [8], *semi-stochastic gradient descent* (S2GD) [28], and *SAGA* [9]. A common benefit of variance reduction methods is that the learning can still converge with relatively large learning rate (refer to Section 2 in [8]), which means we can speed up the training by using larger learning rates. As discussed in [9], we can view some variance reduction methods under a common umbrella as follows. First, mini-batch GD and SGD are an application of Monte Carlo approximation. Suppose that we have two correlated random variables X and Y with the expectation of Y being able to be computed efficiently, we want to approximate the expectation of X using Monte Carlo method. We may use the estimator ω_α for $\mathbb{E}[X]$ defined as $\omega_\alpha \triangleq \alpha(X - Y) + \mathbb{E}[Y]$, where $\alpha \in [0, 1]$. Then, we have $\mathbb{E}[\omega_\alpha] = \alpha\mathbb{E}[X] + (1-\alpha)\mathbb{E}[Y]$ and $\text{Var}[\omega_\alpha] = \alpha^2(\text{Var}[X] + \text{Var}[Y] - 2\,\text{Cov}[X, Y])$. When $\alpha = 1$, the estimator ω_α is unbiased. When X and Y are highly correlated (i.e. $\text{Cov}[X, Y]$ is large), the variance of the estimator ω_α is reduced. Using this framework, we can derive SAG, SAGA and SVRG straightforwardly. At the t^{th} update step, let X be $\nabla f_j\left(\theta^{(t)}\right)$, where j is randomly selected from $\{1, \ldots, N\}$. When Y is $\nabla f_j\left(\theta^{(t-1)}\right)$ and $\alpha = \frac{1}{N}$, we have the estimator proposed in SAG. When Y is $\nabla f_j\left(\theta^{(t-1)}\right)$ and $\alpha = 1$, we have the estimator proposed in SAGA. Finally, the estimator in SVRG is achieved when Y is $\nabla f_j\left(\tilde{\theta}\right)$ and $\alpha = 1$, where $\tilde{\theta}$ is the value of θ at some previous update step.

Apparently SAG yields a biased estimator, whilst the estimator in SVRG is unbiased and this method also has similar convergence rate as SAG and SDCA for smooth and strongly convex functions. S2GD has similar update scheme as SVRG but different way to choose the inner loop iterations. SAGA is considered as a midpoint between SAG and SVRG, having better theoretical convergence rate and yielding unbiased estimator. An advantage of SVRG is memory-efficient because we only need to store the average of all gradients evaluated at the checkpoint , whereas we have to store an entire table of all past gradients in SAG and SAGA. For an elegant optimisation framework with not much demand on memory usage, inspired by SVRG we develop our proposed Stein-based method in the next section.

[1]https://github.com/nhandam/svgd-variance-reduction

III. Proposed Framework

A. General Framework

We start with depicting the problem of interest: approximate a probability distribution that is factorisable $p(\theta) \propto \prod_{i=1}^{N} p_i(\theta)$. This kind of distribution is very popular in statistical machine learning. Now we apply SVGD to approximate $p(\theta)$ via a set of particles $\{\theta_i\}_{i=1}^{M}$. First, we sample $\left\{\theta_i^{(0)}\right\}_{i=1}^{M}$ from an initial simple distribution $q_0(\theta)$. Then, we iteratively update the particles as:

$$\Phi^*(\cdot) = \frac{1}{M}\sum_{m=1}^{M}\left[k\left(\theta_m^{(t)},\cdot\right)\nabla_{\theta_m^{(t)}}\log p\left(\theta_m^{(t)}\right) + \nabla_{\theta_m^{(t)}}k\left(\theta_m^{(t)},\cdot\right)\right], \tag{1}$$

$\theta_i^{(t+1)} = \theta_i^{(t)} + \epsilon_t\Phi^*\left(\theta_i^{(t)}\right)$, where $\theta_i^{(t)}$ is θ_i at the update step t.

Usually the number N depends on the dataset size. When working with large-scale datasets, the evaluation of $\nabla_\theta \log p(\theta)$ in (1) will be computationally expensive because $\nabla_\theta \log p(\theta) = \sum_{i=1}^{N}\nabla_\theta \log p_i(\theta)$ (the equality holds due to the fact that $\nabla_\theta \log p(\theta)$ and $\nabla_\theta \log \tilde{p}(\theta)$ are equal, where $\tilde{p}(\theta)$ is an unnormalised version of $p(\theta)$). As empirically shown in [7], SVGD worked well even with as few particles as $M = 100$ and when they increased M to 250 the performance did not noticeably improve, thus the bottleneck in SVGD when working with large datasets is indeed the gradient $\nabla_\theta \log p(\theta)$. In SVGD, the authors mitigated this issue straightforwardly by using mini-batch gradients to approximate the full batch gradient as $\nabla_\theta \log p(\theta) \approx \frac{N}{n}\sum_{j=1}^{n}\nabla_\theta \log p_{i_j}(\theta)$. Although unbiased, this estimator has high variance [8].

Inspired by SVRG, we propose *Stein variational gradient descent with variance reduction* (SVGD-VR) as a novel and substantial improvement to SVGD. In particular, our proposed framework aims to reduce the variance in the estimate of $\nabla_\theta \log p(\theta)$. It works as follows. We periodically keep a snapshot of the particles $\{\theta_i\}_{i=1}^{M}$ denoted as $\left\{\tilde{\theta}_i\right\}_{i=1}^{M}$. For example, we set the period to be T iterations, then after every T iterations we take a snapshot of the current values of particles $\tilde{\theta}$ and use this checkpoint to update θ in the next T iterations. Precisely, we estimate the full batch gradient using mini-batch gradients modified with the gradients evaluated at the checkpoint $\tilde{\theta}_i$'s as follows:

$$\nabla_\theta \log p(\theta) \approx \frac{N}{n}\sum_{j=1}^{n}\left(\nabla_\theta \log p_{i_j}(\theta) - \nabla_{\tilde{\theta}}\log p_{i_j}\left(\tilde{\theta}\right)\right) + \sum_{i=1}^{N}\nabla_{\tilde{\theta}}\log p_i\left(\tilde{\theta}\right). \tag{2}$$

The full batch gradient evaluated at the checkpoint particles $\tilde{\theta}$ (i.e. the last term in (2)) is only computed once every T iterations (which is a hyperparameter) so the computational cost is less expensive than the full batch approach.

There are two efficient ways to implement the variance reduction technique in our framework. First, we store the checkpoint particles $\tilde{\theta}$ and the full batch gradient evaluated at these particles $\sum_{i=1}^{N}\nabla_{\tilde{\theta}}\log p_i\left(\tilde{\theta}\right)$, and we only update these values every T iterations. Second, we store all stochastic gradients evaluated at the checkpoint $\nabla_{\tilde{\theta}}\log p_i\left(\tilde{\theta}\right)$ for all $i \in \{1,\ldots,N\}$ as well as their sum. The first way is more memory-efficient because M is usually very small compared to N, whereas the second way is more time-efficient because we do not need to re-compute the mini-batch gradients evaluated at the checkpoint $\tilde{\theta}$ within every T update iterations. In all of our experiments in Section IV, we follow the first way.

Algorithm 1 Learning procedure of SVGD-VR for posterior inference.

Input: Dataset $\mathcal{D}$, prior $p(\theta)$, likelihood $p(\mathcal{D} \mid \theta)$, initial particles $\theta^{(0)}$ drawn from $q_0(\theta)$, number of update iterations L, number of variance reduction updates T, batch size n, particle update rate $\epsilon \in [-\eta_0, \eta_0]$, kernel $k(\cdot,\cdot)$.
Output: Particles $\{\theta_m\}_{m=1}^{M}$ that approximate the target posterior $p(\theta \mid \mathcal{D})$.

for $l = 1,\ldots,L$ **do**
  $\theta_i \leftarrow \theta_i^{(l)}, i = 1,\ldots,M.$
  $\mu_i \leftarrow \sum_{j=1}^{N}\nabla_{\theta_i}\log p(y_j \mid x_j, \theta_i), i = 1,\ldots,M.$
  $\tilde{\theta}_i^{(0)} \leftarrow \theta_i, i = 1,\ldots,M.$
  for $t = 1,\ldots,T$ **do**
    Randomly pick $\{j_k\}_{k=1}^{n} \subset \{1,\ldots,N\}$.
    $\rho_i^{(t-1)} \leftarrow \nabla_{\tilde{\theta}_i^{(t-1)}}\log p\left(\tilde{\theta}_i^{(t-1)}\right) + \frac{N}{n}\sum_{k=1}^{n}\nabla_{\tilde{\theta}_i^{(t-1)}}\log p\left(y_{j_k} \mid x_{j_k}, \tilde{\theta}_i^{(t-1)}\right) - \frac{N}{n}\sum_{k=1}^{n}\nabla_{\theta_i}\log p(y_{j_k} \mid x_{j_k}, \theta_i) + \mu_i, i = 1,\ldots,M.$
    $\Phi^*\left(\tilde{\theta}_i^{(t-1)}\right) \leftarrow \frac{1}{M}\sum_{m=1}^{M}\left[k\left(\tilde{\theta}_m^{(t-1)}, \tilde{\theta}_i^{(t-1)}\right)\rho_m^{(t-1)} + \nabla_{\tilde{\theta}_m^{(t-1)}}k\left(\tilde{\theta}_m^{(t-1)}, \tilde{\theta}_i^{(t-1)}\right)\right], i = 1,\ldots,M.$
    $\tilde{\theta}_i^{(t)} \leftarrow \tilde{\theta}_i^{(t-1)} + \epsilon_{l,t}\Phi^*\left(\tilde{\theta}_i^{(t-1)}\right), i = 1,\ldots,M.$
  end for
  $\theta_i^{(l)} \leftarrow \tilde{\theta}_i^{(T)}, i = 1,\ldots,M.$
end for
return $\theta_i^{(L)}, i = 1,\ldots,M.$

B. Posterior Inference

To make our discussion more transparent, we consider a very particular case of the target distribution that is a posterior distribution, and our problem becomes Bayesian inference of the posterior (see Algorithm 1). Let $\mathcal{D} = \left\{(x_i, y_i)\,|_{i=1}^{N}\right\}$ be the dataset of observations and θ be the concatenation of all latent variables, then $p(\theta)$ is the prior, $p(\mathcal{D} \mid \theta)$ is the likelihood, and the posterior $p(\theta \mid \mathcal{D})$ is our target distribution. Assume that the observations $(x_1, y_1), \ldots, (x_N, y_N)$ are conditionally independent given the latent variables θ, the first gradient term in (1) is derived as: $\nabla_\theta \log p(\theta \mid \mathcal{D}) = \nabla_\theta \log p(\theta) + \sum_{i=1}^{N}\nabla_\theta \log p(y_i \mid x_i, \theta)$. Following the idea in (2), our proposed unbiased variance-reduced mini-batch estimator for the second term in the equation above is:

$$\frac{N}{n}\sum_{j=1}^{n}\left(\nabla_\theta \log p\left(y_{i_j} \mid x_{i_j}, \theta\right) - \nabla_{\tilde{\theta}}\log p\left(y_{i_j} \mid x_{i_j}, \tilde{\theta}\right)\right) + \sum_{i=1}^{N}\nabla_{\tilde{\theta}}\log p\left(y_i \mid x_i, \tilde{\theta}\right).$$

IV. Experimental Results

A. Binary Classification with Bayesian Logistic Regression

In this section, we conduct binary classification experiments on four datasets downloaded from LIBSVM repository[2]: *a9a*,

[2]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

w8a, *cod-rna*, and *covertype*. Two selected baseline methods are SVGD and SVRG. We employ two models: Bayesian logistic regression (BLR) and logistic regression (LR). Our proposed SVGD-VR and the baseline SVGD are applied to BLR, whilst the remaining baseline SVRG is applied to LR. For convenience, we might refer colloquially to this training setting as 'non-online'. To demonstrate the strong generalisation capability of SVGD-VR, we further conduct experiments in the pseudo-online learning setting. Our implementation is in Python with reference to the code of the authors of SVGD.

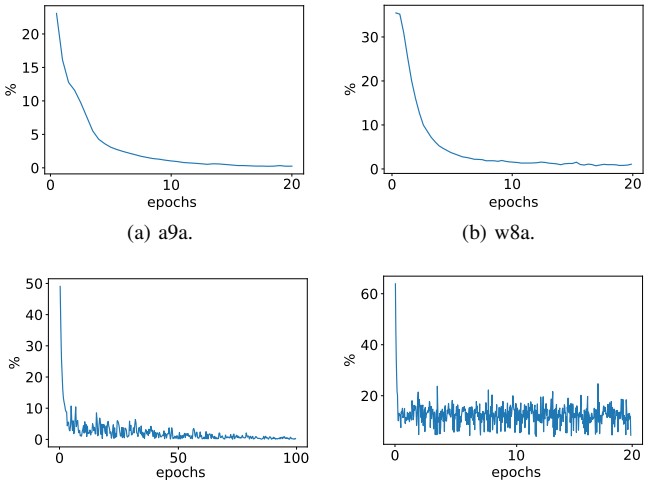
1) Model specifications: In both SVGD and SVGD-VR, we follow the generative model described in [7] to use the following BLR:

$$\alpha \sim \text{Gamma}\left(a, b\right), \qquad w \sim \mathcal{N}\left(\mathbf{0}, \text{diag}\left(\alpha^{-1}, \ldots, \alpha^{-1}\right)\right),$$
$$p\left(y = 1 \mid x, w\right) = \frac{1}{1 + e^{-w^{\intercal} x}}.$$

We apply SVGD and SVGD-VR to approximate the posterior of weights w given dataset $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$. In our experiments, we always fix the hyperparameters of the gamma distribution as $a = 1, b = 0.01$, batch size as 128, number of variance-reduced updates (i.e. variable T in Algorithm 1) as 128, number of particles w as 100. We apply RMSProp strategy with momentum factor fixed at 0.9 to compute the update terms for the particles. In LR, as in [8], we use stochastic gradient descent combined with SVRG technique and denote this model as SVRG. The batch size and number of variance-reduced updates are the same as BLR models.

2) Non-online binary classification: In both BLR and LR, we perform 10-fold cross validation to tune the step size (or learning rate) from the values $\left\{2^{-1}, 2^{-2}, \ldots, 2^{-9}\right\}$. Upon obtaining the best step size or learning rate, we train and test the models for 100 times, then compute the mean and standard deviation of the accuracy. To illustrate the training progress, we periodically evaluate the interim performance of the models on test set during training besides recording the final performance at the epoch whose accuracy in cross-validation has been found as the best. We train the models for 100 epochs on *cod-rna*, whereas we train them for 20 epochs on the remaining datasets. Furthermore, as the long training of BLR proceeded on *cod-rna*, the standard deviations tended to increase, so we apply exponential decay scheme for the step size with the decay factor of 2^{-14}. For the other datasets, step size decay is not demanded.

First, we consider the variance reduction effect, which means we will show that compared to SVGD our proposed SVGD-VR does reduce the variance of the mini-batch estimation. In each experiment, whenever we approximate a full batch gradient of $\log \tilde{p}\left(\theta \mid \mathcal{D}\right)$ by a mini-batch gradient (via either traditional approach as in SVGD or variance reduction approach as in SVGD-VR), we compute the standard deviation of the approximation. Then, we calculate the Euclidean norm of each standard deviation to represent this quantity by a single value. Finally, we compute the ratio of the corresponding norms of standard deviations in SVGD-VR and SVGD periodically during training (i.e. norm of standard deviation of estimated gradient in SVGD-VR divided by norm of standard deviation of estimated gradient in SVGD), and plot the results in Fig. 1. We can see that our proposed SVGD-VR effectively



(a) a9a. (b) w8a. (c) cod-rna. (d) covertype.

Figure 1: The ratio of Euclidean norms of standard deviation of gradient estimators used in SVGD-VR and SVGD as training progresses in binary classification.

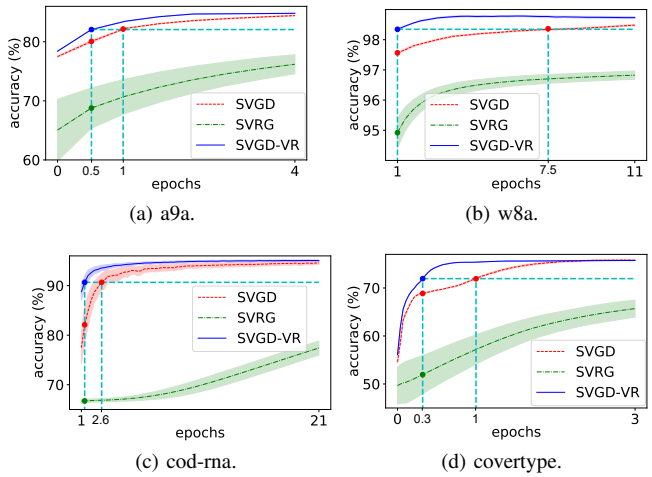
Dataset	SVGD	SVRG	SVGD-VR
a9a	84.62 ± 0.08	82.84 ± 0.45	$\mathbf{84.90 \pm 0.07}$
w8a	98.51 ± 0.03	97.02 ± 0.12	$\mathbf{98.78 \pm 0.02}$
cod-rna	94.55 ± 0.39	92.57 ± 0.15	$\mathbf{95.06 \pm 0.14}$
covertype	75.53 ± 0.15	74.07 ± 0.31	$\mathbf{75.66 \pm 0.03}$

Table I: Accuracy (%) of non-online binary classification.

reduces the variance of the gradient estimation and this effect is more apparent as the training progresses. For example, on *a9a*, at the very first iteration, the ratio is equal to 23.07%. In the first 5 epochs, it drops significantly and eventually reaches 0.25% after 20 epochs.

Second, we compute the accuracy of binary classification and report the results in Fig. 2 and Table I. Note that the results in Fig. 2 are only a part of our training that is best for visualisation purpose, whilst numerical results in Table I are reported at the optimal epochs (previously found in cross-validation). As illustrated in Fig. 2, our proposed SVGD-VR clearly converges faster than the baseline methods. The two Stein methods (i.e. SVGD and SVGD-VR) also improve the stability of training (i.e. reducing the standard deviation of accuracy) compared to SVRG, and among the two Stein methods our proposed one is more stable with smaller accuracy standard deviation in most cases. Finally, at convergence, our SVGD-VR outperforms the other methods in the sense that it yields higher accuracy mean and lower accuracy standard deviation on all chosen datasets as shown in Table I.

3) Pseudo-online binary classification: In the pseudo-online learning setting, we update each model using an incoming data point at a time (i.e. the batch size is 1). Before updating a model, we predict the label of the incoming data point using the current model and record the accumulative accuracy. However, this setting is pseudo-online because we still have access to the full dataset at the outset. Each experiment



(a) a9a. (b) w8a. (c) cod-rna. (d) covertype.

Figure 2: Results of non-online binary classification. The shaded area around each line represents the standard deviation of the accuracy. In each sub-figure, the left cyan vertical line indicates the iteration at which SVGD-VR obtains 95% of its accuracy reported in Table I, whilst the right cyan vertical line indicates the iteration at which SVGD achieves similar performance. SVRG is not able to reach this accuracy within our visualisation window in any of these sub-figures.

Dataset	SVGD	SVRG	SVGD-VR
a9a	81.44 ± 0.07	81.69 ± 0.69	$\mathbf{84.45 \pm 0.05}$
w8a	96.72 ± 0.02	96.00 ± 0.30	$\mathbf{98.56 \pm 0.02}$
cod-rna	92.61 ± 0.03	91.86 ± 0.12	$\mathbf{95.05 \pm 0.01}$
covertype	71.70 ± 0.03	73.47 ± 0.15	$\mathbf{75.53 \pm 0.02}$

Table II: Accuracy (%) of pseudo-online binary classification.

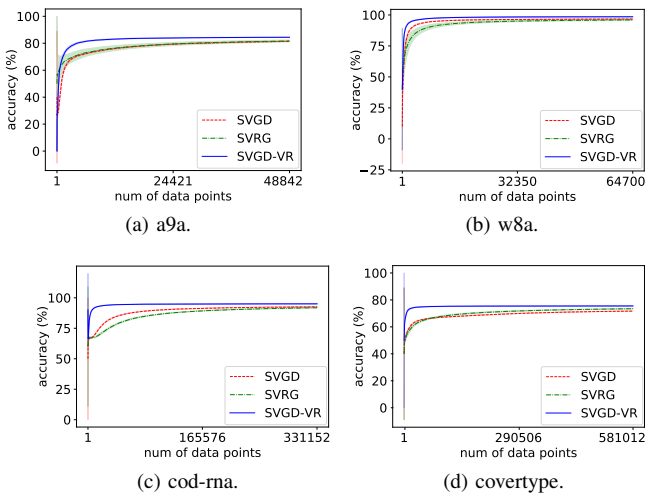
is run 10 times, then the mean and standard deviation of the accuracy are recorded.

The results are shown in Fig. 3 and Table II. In these experiments, we achieve similar behaviours to non-online learning ones above when comparing 3 methods, i.e. our proposed SVGD-VR apparently improves the learning with faster convergence, higher accuracy mean and lower accuracy standard deviation. However, the difference between SVGD-VR and the baseline methods in Table II is more significant than that in Table I, which intuitively emphasises the superior generalisation ability of SVGD-VR compared with SVGD and SVRG.

B. Regression with Bayesian Neural Networks

In this section, we conduct regression experiments on 10 datasets downloaded from OpenML and UCI Machine Learning Repository[3] (***kin8nm***, ***yacht*** *hydrodynamics*, ***housing***, ***energy*** *efficiency*, ***concrete*** *compressive strength*, ***wine*** *quality red*, *combined cycle* ***power*** *plant*, *condition based maintenance of* ***naval*** *propulsion plants*, *physicochemical properties of* ***protein*** *tertiary structure*, and ***year*** *prediction MSD*) with

[3]https://www.openml.org/d/189; https://archive.ics.uci.edu/



(a) a9a. (b) w8a. (c) cod-rna. (d) covertype.

Figure 3: Results of pseudo-online binary classification. The shaded area around each line represents the standard deviation of accuracy.

Bayesian neural network (BNN) whose posterior inference is done by SVGD-VR and SVGD. Using the code of the authors of SVGD as a reference, we utilise TensorFlow [29] in our implementation.

1) Model specifications: Our BNN architecture is based on the settings presented in [30], [7]. In particular, let the dataset be $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$, where N is the total number of data points, $x_n \in \mathbb{R}^d$ is a feature vector and $y_n \in \mathbb{R}$ is a response variable. We might also denote the dataset as $\mathcal{D} = (X, Y)$, where X is an $N \times d$ matrix consisting of all feature vectors x_n, and Y is a vector consisting of all response variables y_n. Let $f_{\mathcal{W}}(\cdot)$ be a neural network parameterised by the weights $\mathcal{W}$, we assume that each response variable is computed as $y_n = f_{\mathcal{W}}(x_n) + e_n$, where e_n is a noise variable drawn from the Gaussian distribution $\mathcal{N}(0, \gamma^{-1})$. In our experiments, the network has one hidden layer and uses ReLU as the activation function. Each element of $\mathcal{W}$ follows the Gaussian distribution $\mathcal{N}(0, \lambda^{-1})$. Finally, we assume that γ and λ both follow the same gamma distribution, i.e. $\text{Gamma}(a_0, b_0)$. In total, the set of our learnable parameters is $\theta \triangleq \{\mathcal{W}, \gamma, \lambda\}$, and SVGD and SVGD-VR will learn a set of particles $\hat{\theta}$ that best mimics the posterior $p(\theta \mid \mathcal{D}) = p(\mathcal{W}, \gamma, \lambda \mid \mathcal{D})$.
The likelihood of parameters θ given the dataset $\mathcal{D}$ is:

$$p(Y \mid X, \mathcal{W}, \gamma) = \prod_{n=1}^{N} \mathcal{N}\left(y_n \mid f_{\mathcal{W}}(x_n), \gamma^{-1}\right).$$

The priors of $\mathcal{W}$, γ and λ are respectively:

$$p(\mathcal{W} \mid \lambda) = \prod_i \mathcal{N}\left(w_i \mid 0, \lambda^{-1}\right)$$
$$p(\gamma) = \text{Gamma}(\gamma \mid a_0, b_0)$$
$$p(\lambda) = \text{Gamma}(\lambda \mid a_0, b_0).$$

Then, the posterior of parameters θ given the dataset $\mathcal{D}$ is:

$$p(\mathcal{W}, \gamma, \lambda \mid \mathcal{D}) \propto p(Y \mid X, \mathcal{W}, \gamma)\, p(\mathcal{W} \mid \lambda)\, p(\gamma)\, p(\lambda).$$

In all experiments, we fix the step size $\epsilon = 0.001$, $a_0 = 1$ and $b_0 = 0.1$ as in [7]. For other settings, we set the batch size as 128 (and 1024 for *year*), number of hidden units as 64 (and 128 for *protein* and *year*), number of SVGD updates as 2048

Dataset	Ratio of Standard Deviations		
	Minimum	Median	Maximum
yacht	4.38%	13.50%	38.54%
housing	7.86%	24.81%	33.60%
energy	4.95%	14.21%	27.63%
concrete	4.91%	9.61%	17.04%
wine	5.44%	15.62%	29.43%
kin8nm	2.99%	4.01%	13.01%
power	3.19%	4.97%	24.28%
naval	6.10%	22.49%	51.62%

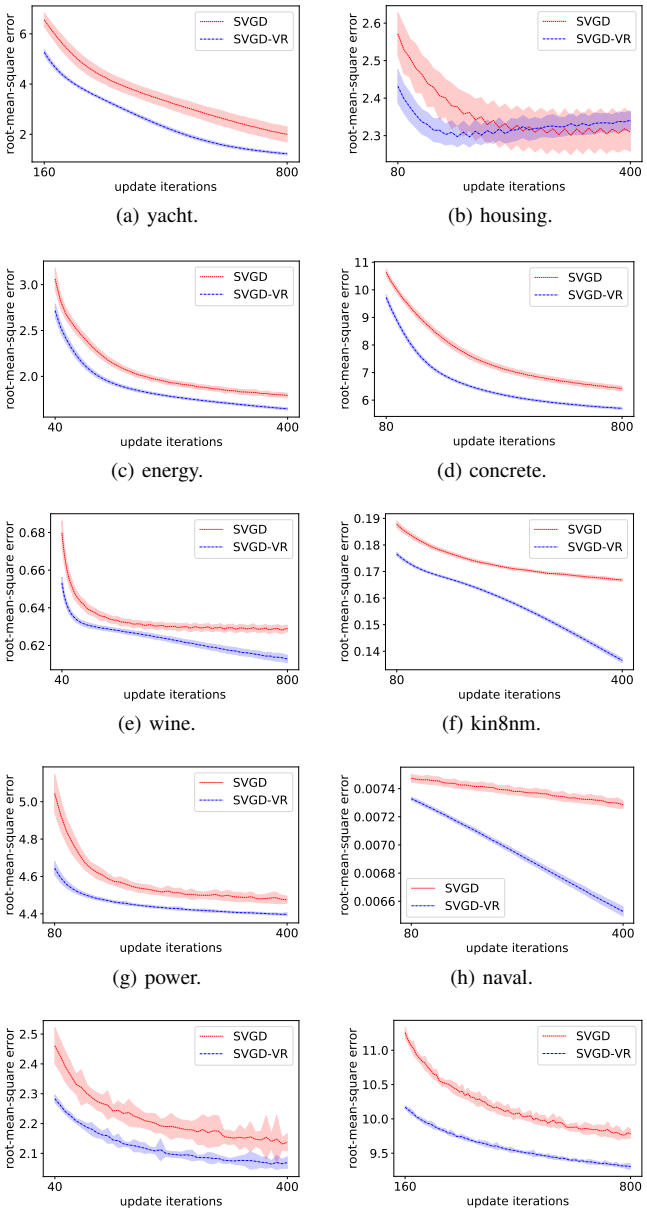
Table III: Statistics of the average ratios of norm of standard deviation of gradient estimators in SVGD-VR and SVGD in regression with Bayesian neural networks.

(and 4096 for *year*), number of variance-reduced updates as 8, number of particles as 32. Note that in our proposed SVGD-VR, the number of SVGD updates and the number of variance-reduced updates are respectively L and T in Algorithm 1.

2) Variance reduction effect: We now conduct experiments to compare the variances of the mini-batch gradient estimators in SVGD and our proposed SVGD-VR. In particular, we train both models in regression task on eight datasets. In each experiment, whenever we approximate a full gradient of $\log \tilde{p}(\theta \mid \mathcal{D})$ by a mini-batch gradient (via either traditional approach as in SVGD or variance reduction approach as in SVGD-VR) we compute the standard deviation of the approximation. As each gradient of $\log \tilde{p}(\theta \mid \mathcal{D})$ is a vector, its standard deviation is also a vector, thus we calculate the Euclidean norm of its standard deviation to represent this quantity by a single value. Finally, we compute the ratio of the corresponding norms of standard deviations in SVGD-VR and SVGD periodically during training (i.e. norm of standard deviation of estimated gradient in SVGD-VR divided by norm of standard deviation of estimated gradient in SVGD). In Table III, we report the minimum, maximum and median values of the ratios across all training iterations in each dataset. We can see that as expected our proposed SVGD-VR effectively reduces the variance of the gradient estimation on all eight chosen datasets. The variance reduction effect is most prominent on *concrete*, *kin8nm*, and *power* datasets since the norm of standard deviation of the estimated gradient in SVGD-VR is less than 10% of the norm of standard deviation of the estimated gradient in SVGD for at least half of the training time (i.e. the medians are all less than 10% on these datasets). In the entire training on all eight datasets, our proposed SVGD-VR in the worst case scenario can still reduce about half of the norm of the standard deviation of the estimated gradient (i.e. maximum value of 51.62% on *naval*).

Note that these experiments are computationally expensive because we need to compute the full batch gradient and all possible mini-batch gradients. Thus, we did not utilise the two largest datasets (i.e. *protein* and *year*) in these experiments.

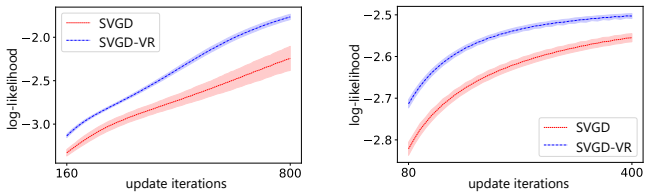
3) Regression performance comparison: In this part, we present the experiments to compare the regression performance



(a) yacht. (b) housing. (c) energy. (d) concrete. (e) wine. (f) kin8nm. (g) power. (h) naval. (i) protein. (j) year.

Figure 4: Root-mean-square error evaluated on the test set during training on 10 chosen datasets. The shaded area around each line represents the standard deviation of the root-mean-square error.

of the BNN employed with our proposed SVGD-VR and with the baseline SVGD via two metrics: average root-mean-square error (RMSE) of the test prediction and average log-likelihood given the test data. To compute the average performance, for *year* we repeat each experiment 20 times, whilst the number of trials on the remaining nine datasets is 50 (note that in [7] the authors only ran once on *year*, five times on *protein*, and 20 times on the remaining datasets). To illustrate the training progress, we periodically evaluate the interim performance of the models on the test set during training (see Fig. 4 and Fig. 5) besides recording the final performance after the training



(a) yacht. (b) housing.

(c) energy. (d) concrete.

(e) wine. (f) kin8nm.

(g) power. (h) naval.

(i) protein. (j) year.

Figure 5: Log-likelihood evaluated on the test set during training on 10 chosen datasets. The shaded area around each line represents the standard deviation of the log-likelihood.

has finished (see Tables IV and V). Note that in Fig. 4 and Fig. 5, for fair comparison, we do take into account the fact that in SVGD-VR there are T variance-reduced iterations inside each SVGD update. The x-axes in these figures represent the SVGD update iterations in the baseline SVGD method. That means, suppose that an interim performance of SVGD-VR is evaluated after l^{th} SVGD update iteration and t^{th} variance-reduced iteration, that point is plotted with the horizontal coordinate of $(l \times T + t)$. This conversion guarantees that both models are fed with the same amount of training data points at any point of comparison. It is worth pointing out that the results in Fig. 4 and Fig. 5 are only a part of the training that is best for visualisation, whilst numerical results

Dataset	Average Test RMSE		Percentage of Improvement
	SVGD	SVGD-VR	
yacht	0.871 ± 0.065	$\mathbf{0.769 \pm 0.027}$	11.71%
housing	$\mathbf{2.394 \pm 0.050}$	2.408 ± 0.032	−0.58%
energy	1.391 ± 0.027	$\mathbf{1.284 \pm 0.012}$	7.69%
concrete	5.734 ± 0.063	$\mathbf{5.304 \pm 0.044}$	7.50%
wine	0.624 ± 0.002	$\mathbf{0.599 \pm 0.003}$	4.01%
kin8nm	0.118 ± 0.001	$\mathbf{0.085 \pm 0.000}$	27.97%
power	4.391 ± 0.017	$\mathbf{4.309 \pm 0.005}$	1.87%
naval	0.006 ± 0.000	$\mathbf{0.004 \pm 0.000}$	33.33%
protein	2.054 ± 0.043	$\mathbf{2.004 \pm 0.022}$	2.43%
year	9.320 ± 0.049	$\mathbf{8.915 \pm 0.026}$	4.35%

Table IV: Root-mean-square error in regression with Bayesian neural network on 10 chosen datasets.

Dataset	Average Test Log-Likelihood		Percentage of Improvement
	SVGD	SVGD-VR	
yacht	-1.401 ± 0.076	$\mathbf{-1.259 \pm 0.027}$	10.14%
housing	$\mathbf{-2.489 \pm 0.008}$	-2.492 ± 0.007	−0.12%
energy	-1.746 ± 0.019	$\mathbf{-1.666 \pm 0.010}$	4.58%
concrete	-3.158 ± 0.011	$\mathbf{-3.072 \pm 0.009}$	2.72%
wine	-0.946 ± 0.004	$\mathbf{-0.901 \pm 0.005}$	4.76%
kin8nm	0.724 ± 0.009	$\mathbf{1.044 \pm 0.005}$	44.20%
power	-2.906 ± 0.004	$\mathbf{-2.888 \pm 0.001}$	0.62%
naval	3.714 ± 0.018	$\mathbf{4.104 \pm 0.021}$	10.50%
protein	-2.146 ± 0.016	$\mathbf{-2.121 \pm 0.011}$	1.16%
year	-3.634 ± 0.005	$\mathbf{-3.591 \pm 0.006}$	1.18%

Table V: Log-likelihood in regression with Bayesian neural network on 10 chosen datasets.

in Tables IV and V are reported after the training has finished.

From Fig. 4 and Fig. 5, we can see that on all 10 datasets our proposed SVGD-VR yields faster convergence with respect to the update iteration. Except for *housing*, SVGD-VR maintains its leading performance as training progresses. On some datasets (such as *yacht*, *energy*, *concrete*, *power*, *protein*), more stable training is also observed in SVGD-VR with the standard deviation of RMSE or log-likelihood being consistently smaller than that of SVGD.

In Tables IV and V, regarding the mean values, our proposed SVGD-VR clearly outperforms the baseline SVGD on 9 over 10 chosen datasets. When one takes into account the standard deviation (which is somehow related to the stability of the performance of a model after trained), SVGD-VR again outperforms SVGD on 8 over 10 datasets. The last columns in these tables show the percentage of improvement, which is computed as the difference of the mean scores of SVGD-VR and SVGD divided by the mean score of SVGD. This method quantifies the relative performance boost when using our proposed SVGD-VR compared with the baseline SVGD.

Except for *housing*, considering the RMSE our SVGD-VR enhances the performance up to 33.33% and not less than 1.87%, and when it comes to log-likelihood the performance boost is in the range of 0.62% to 44.20%. Particularly, on *yacht*, *kin8nm* and *naval* datasets, the proposed model enhances the performance by more than 10% in both evaluation metrics. Finally, on *housing* (the only dataset on which SVGD-VR underperforms the baseline SVGD in our experiments), our SVGD-VR drops the performance by only small fractions of 0.58% (RMSE) and 0.12% (log-likelihood).

V. Conclusion and Future Work

In this paper, we have proposed a generic Bayesian inference method grounded in SVGD to approximate a probabilistic distribution via samples. Our key contribution is the success in mitigating the limitation of SVGD when working with large datasets by introducing a method to tame the variance of the mini-batch gradient estimator. In particular, we have proposed a simple yet effective way to derive an unbiased variance-reduced mini-batch gradient estimator for the term related to the distribution evaluated on the full dataset. Our method is inspired by SVGD and SVRG, and it also enjoys their advantages. Hence, our model is an effective framework for generic probabilistic inference with large-scale datasets. Furthermore, our model provides faster convergence, more stable training, and more robust nature to the batch size (it works well even with batch size of 1).

Via experiments on 14 datasets, we have empirically demonstrated the merits of our approach in the context of binary classification problem and its pseudo-online learning setting with Bayesian logistic regression, and regression problem with Bayesian neural network. Lastly, the proposed method is a generic methodology for probabilistic inference and applicable to a much wider set of problems such as joint distribution inference in Markov random fields or Bayesian networks.

VI. Acknowledgement

This work was financially supported by the Australian Research Council (ARC) DP160109394.

References

[1] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *The Journal of Machine Learning Research (JMLR)*, vol. 3, no. 1, pp. 993–1022, 2003.

[2] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient Langevin dynamics," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2011, pp. 681–688.

[3] Y.-A. Ma, T. Chen, and E. Fox, "A complete recipe for stochastic gradient MCMC," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 2917–2925.

[4] R. Bardenet, A. Doucet, and C. Holmes, "On Markov chain Monte Carlo methods for tall data," *The Journal of Machine Learning Research (JMLR)*, vol. 18, no. 1, pp. 1515–1557, 2017.

[5] M. Quiroz, R. Kohn, M. Villani, and M.-N. Tran, "Speeding up MCMC by efficient data subsampling," *The Journal of the American Statistical Association (JASA)*, pp. 1–13, 2018.

[6] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, "Stochastic variational inference," *The Journal of Machine Learning Research (JMLR)*, vol. 14, no. 1, pp. 1303–1347, 2013.

[7] Q. Liu and D. Wang, "Stein variational gradient descent: A general purpose Bayesian inference algorithm," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 2378–2386.

[8] R. Johnson and T. Zhang, "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013, pp. 315–323.

[9] A. Defazio, F. Bach, and S. Lacoste-Julien, "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 1646–1654.

[10] K. A. Dubey, S. J. Reddi, S. A. Williamson, B. Poczos, A. J. Smola, and E. P. Xing, "Variance reduction in stochastic gradient Langevin dynamics," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016, pp. 1154–1162.

[11] J. Baker, P. Fearnhead, E. B. Fox, and C. Nemeth, "Control variates for stochastic gradient MCMC," *Statistics and Computing*, vol. 29, no. 3, pp. 599–615, 2019.

[12] N. Chatterji, N. Flammarion, Y. Ma, P. Bartlett, and M. Jordan, "On the theory of variance reduction for stochastic gradient Monte Carlo," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 763–772.

[13] M. Wu, N. Goodman, and S. Ermon, "Differentiable antithetic sampling for variance reduction in stochastic variational inference," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2019, pp. 2877–2886.

[14] A. Miller, N. Foti, A. D'Amour, and R. P. Adams, "Reducing reparameterization gradient variance," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 3708–3718.

[15] G. Roeder, Y. Wu, and D. K. Duvenaud, "Sticking the landing: Simple, lower-variance gradient estimators for variational inference," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 6925–6934.

[16] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *The Journal of Machine Learning Research (JMLR)*, vol. 7, no. 3, pp. 551–585, 2006.

[17] M. Dredze, K. Crammer, and F. Pereira, "Confidence-weighted linear classification," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2008, pp. 264–271.

[18] Q. Liu, J. Lee, and M. Jordan, "A kernelized Stein discrepancy for goodness-of-fit tests," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 276–284.

[19] K. Nguyen, T. Le, T. D. Nguyen, D. Phung, and G. I. Webb, "Robust Bayesian kernel machine via Stein variational gradient descent for big data," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD)*. ACM, 2018, pp. 2003–2011.

[20] J. Han and Q. Liu, "Stein variational gradient descent without gradient," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 1895–1903.

[21] J. Zhuo, C. Liu, J. Shi, J. Zhu, N. Chen, and B. Zhang, "Message passing Stein variational gradient descent," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 6013–6022.

[22] Q. Liu and D. Wang, "Stein variational gradient descent as moment matching," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, pp. 8868–8877.

[23] Y. Pu, Z. Gan, R. Henao, C. Li, S. Han, and L. Carin, "VAE learning via Stein variational gradient descent," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 4236–4245.

[24] Y. Feng, D. Wang, and Q. Liu, "Learning to draw samples with amortized Stein variational gradient descent," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

[25] D. Wang, Z. Zeng, and Q. Liu, "Stein variational message passing for continuous graphical models," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 5206–5214.

[26] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017.

[27] S. Shalev-Shwartz and T. Zhang, "Stochastic dual coordinate ascent methods for regularized loss minimization," *The Journal of Machine Learning Research (JMLR)*, vol. 14, no. 2, pp. 567–599, 2013.

[28] J. Konečnỳ and P. Richtárik, "Semi-stochastic gradient descent methods," *arXiv preprint arXiv:1312.1666*, 2013.

[29] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[30] J. M. Hernández-Lobato and R. Adams, "Probabilistic backpropagation for scalable learning of Bayesian neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2015, pp. 1861–1869.