Self-Adversarially Learned Bayesian Sampling

Yang Zhao
State University of New York at Buffalo
yzhao63@buffalo.edu

Jianyi Zhang
Fudan University
15300180019@fudan.edu.cn

Changyou Chen
State University of New York at Buffalo
cchangyou@gmail.com

Abstract

Scalable Bayesian sampling is playing an important role in modern machine learning, especially in the fast-developed unsupervised-(deep)-learning models. While tremendous progresses have been achieved via scalable Bayesian sampling such as stochastic gradient MCMC (SG-MCMC) and Stein variational gradient descent (SVGD), the generated samples are typically highly correlated. Moreover, their sample-generation processes are often criticized to be inefficient. In this paper, we propose a novel self-adversarial learning framework that automatically learns a conditional generator to mimic the behavior of a Markov kernel (transition kernel). High-quality samples can be efficiently generated by direct forward passes though a learned generator. Most importantly, the learning process adopts a self-learning paradigm, requiring no information on existing Markov kernels, *e.g.*, knowledge of how to draw samples from them. Specifically, our framework learns to use current samples, either from the generator or pre-provided training data, to update the generator such that the generated samples progressively approach a target distribution, thus it is called self-learning. Experiments on both synthetic and real datasets verify advantages of our framework, outperforming related methods in terms of both sampling efficiency and sample quality.

1 Introduction

With the abundance of unlabeled data, Bayesian methods have been increasingly popular in modern machine learning. Various real-world applications have greatly benefited from Bayesian modeling through uncertainty modeling (Blundell et al. 2015; Zhang et al. 2018), deep generative models (Feng, Wang, and Liu 2017; Chen et al. 2017) and deep reinforcement learning (Osband and Van Roy 2017; Haarnoja et al. 2017; Liu et al. 2017). The core of Bayesian methods is efficient Bayesian inference, among which Bayesian sampling stands as one of the most effective tools.

In the setting of big data, recent research has facilitated the development of scalable Bayesian sampling methods. There are mainly two directions on developing these methods, Markov-chain (MC) based and particle-optimization (PO) based methods. Stochastic gradient Markov chain Monte Carlo (SG-MCMC) is a family of scalable MC-based Bayesian learning algorithms designed to efficiently sample from a target (posterior) distribution (Welling and Teh 2011; Chen, Fox, and Guestrin 2014; Ding et al. 2014; Chen, Ding, and Carin 2015). Specifically, SG-MCMC generates samples from a Markov chain induced by an Itó diffusion. Under a standard setting, samples from SG-MCMC can approximate a target distribution arbitrarily well given sufficient samples (Teh, Thiery, and Vollmer 2016; Chen, Ding, and Carin 2015). By contrast, PO-based sampling methods such as Stein variational gradient descent (SVGD) (Liu and Wang 2016) initiate a set of particles (or samples) from some simple distributions, and update them iteratively and interactively to approximate a target distribution. Recently, (Chen et al. 2018) proposed a unified Bayesian sampling framework by combing SG-MCMC and SVGD from a Wasserstein-gradient-flow (WGF) perspective, obtaining improved performance compared to both SG-MCMC and SVGD. Our proposed method is partly based on the WGF theory presented in (Chen et al. 2018).

Though achieving encouraging results, we note two issues in the aforementioned sampling methods: *i*) Slow sample generation: though SG-MCMC and SVGD achieve scalable sampling by adopting stochastic gradient information, sample generation is still not efficient enough under complicated models such as a very deep neural network. The problem is even more severe in SVGD as each particle needs to interact with all other particles in the sample-generation process; *ii*) Slow mixing: samples tend to be highly correlated, leading to slow mixing. Actually, it has been shown that diffusion-based methods such as SG-MCMC might need exponential time to jump out of local modes (Raginsky, Rakhlin, and Telgarsky 2017; Zhang, Liang, and Charikar 2017). Thus more sample-efficient algorithms are desperate to be designed.

In this paper, we reinterpret Bayesian sampling as learning a Markov kernel (or a transition kernel), a conditional probability sequentially mapping an old state (sample) to a new state. Leveraging advantages of scalable sampling and recent developments on deep generative models, we reformulate the sampling process based on the generative-adversarial-net (GAN) framework (Goodfellow et al. 2014). The formulation is based on the connection between density evolution in a Bayesian sampling algorithm and WGFs. Specifically, a conditional generator which solves the corresponding WGF is trained to mimic the sample-generation

process. In this way, both fast sample generation and fast sample mixing are achieved.

We consider two settings in our framework to learn the conditional generators, *i.e.*, *i*) when samples from the unknown target distribution are available, and *ii*) when only the form of the target distribution is provided. The former case can be learned by directly adopting standard GAN training techniques, whereas the later case is much more challenging. To overcome the challenge, we propose a *self-learning* paradigm that adjust samples from the generator itself to approach the target distribution in a principled way, such that the adjusted samples can be used as real samples to train the generator. We call our proposed framework self-adversarially learned Bayesian sampling. Extensive experiments are performed on both synthetic and real datasets, demonstrating the effectiveness and efficiency of the proposed framework, relative to existing methods.

2 Preliminaries

This section reviews background of related Bayesian sampling algorithms, *e.g.*, SG-MCMC, SVGD and particle-optimization Bayesian sampling (POS) (Chen et al. 2018).

2.1 Stochastic gradient MCMC

Given observations $\mathcal{D} = \{\mathbf{d}^{(i)}\}_{i=1}^N$, we aim at drawing samples from a target posterior distribution $p(\mathbf{x}|\mathcal{D})$ with model parameters $\mathbf{x} \in \mathbb{R}^d$. In Bayesian modeling, we write $p(\mathbf{x}|D) \propto \exp(-U(\mathbf{x}))$, where $U(\mathbf{x}) \triangleq -\log(p(\mathbf{x})) - \sum_{i=1}^N \log p(\mathbf{d}^{(i)} \,|\, \mathbf{x})$ is called the potential energy (negative log unnormalized posterior). SG-MCMC is a scalable Bayesian sampling method, which takes stochastic gradient information of the potential energy into consideration. Let $\nabla_{\mathbf{x}}\widetilde{U}(\mathbf{x}) \triangleq -\nabla_{\mathbf{x}}[(\log(p(\mathbf{x})) + \frac{N}{n}\sum_{i=1}^n \log p(\mathbf{d}^{\pi_i} \,|\, \mathbf{x})]$ be a stochastic version of $\nabla_{\mathbf{x}}U(\mathbf{x})$ with π_i the i-th element of a random permutation of $[1, \cdots, N]$. The stochastic gradient Langevin dynamic (SGLD) stands as the first SG-MCMC algorithm (Welling and Teh 2011), endowing the following update rule (samples are indexed by ℓ):

$$\mathbf{x}_{\ell+1} = \mathbf{x}_\ell - \epsilon_{\ell+1}\nabla_{\mathbf{x}}\widetilde{U}(\mathbf{x}_\ell) + \sqrt{2\epsilon_{\ell+1}}\,\boldsymbol{\zeta}_{\ell+1}, \qquad (1)$$

where $\{\epsilon_\ell\}$ is a stepsize sequence, and $\boldsymbol{\zeta}_\ell \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. Further development on SG-MCMC methods leads to several variants of SGLD by introducing auxiliary variables into the corresponding dynamics systems (Ding et al. 2014; Chen, Fox, and Guestrin 2014). With samples $\{\mathbf{x}_\ell\}_{i=1}^M$ from a sampler, one can approximate statistics of a function $f(\mathbf{x})$, *e.g.*, the posterior expectation is approximated as $\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x}\,|\mathcal{D})}[f(\mathbf{x})] \approx \frac{1}{M}\sum_{i=1}^M f(\mathbf{x}_\ell)$.

2.2 Stein variational gradient descent

Different from SG-MCMC, SVGD is derived from a particle-optimization perspective (Liu and Wang 2016). It iteratively and interactively updates a set of particles $\{\mathbf{x}_\ell^{(i)}\}_{i=1}^M$ drawn from some initial distribution. The updating rule follows $\mathbf{x}_{\ell+1}^{(i)} = \mathbf{x}_\ell^{(i)} + \frac{\epsilon}{M}\Delta\,\mathbf{x}_\ell^{(i)}$ with

$$\Delta\,\mathbf{x}_\ell^{(i)} \triangleq \sum_{j=1}^M [-\nabla_{\mathbf{x}_\ell^{(j)}} U(\mathbf{x}_\ell^{(j)})\kappa(\mathbf{x}_\ell^{(i)}, \mathbf{x}_\ell^{(j)}) + \nabla_{\mathbf{x}_\ell^{(j)}}\kappa(\mathbf{x}_\ell^{(i)}, \mathbf{x}_\ell^{(j)})], \qquad (2)$$

where $\kappa(\cdot,\cdot)$ is a positive definite kernel, *e.g.*, the RBF kernel $\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\| \mathbf{x}^{(i)} - \mathbf{x}^{(j)} \|_2^2/h)$ with bandwidth h, and ϵ is the step size. It is shown that (2) is equivalent to minimizing the Kullback-Leibler (KL) divergence $\mathsf{KL}(q(\mathbf{x})||p(\mathbf{x}\,|\mathcal{D}))$, where q is the underlying density of the particles. Consequently, SVGD drives the particles to asymptotically distributed as the target distribution.

2.3 Particle-optimization sampling

Compared with SVGD, an instance of the POS framework (Chen et al. 2018), samples from SG-MCMC are likely highly correlated due to the property of Markovian chains. The POS framework alleviate the issue by interpreting both SG-MCMC and SVGD as WGFs on the space of probability measures $\mathcal{P}(\Omega)$, and proposing a unified particle-optimization framework for efficient Bayesian sampling.

Specifically, the POS framework translates Bayesian sampling to solving a partial differential equation defined on $\mathcal{P}(\Omega)$ with $\Omega \subset \mathbb{R}^d$, defined as:

$$\partial_t \mu_t + \nabla \cdot (\mathbf{v}_t\, \mu_t) = 0\,. \qquad (3)$$

Here μ_t is an absolutely continuous curve on $\mathcal{P}(\Omega)$ and $\mathbf{v}_t$ is a vector field describing the direction of sample evolutions. In WGFs, $\mathbf{v}_t$ is related to what is known as energy functional $E(\mu)$, mapping a probability measure μ to a real value, *i.e.* $E : \mathbb{R}^d \to \mathbb{R}$, via the equation (Ambrosio, Gigli, and Savaré 2005): $\mathbf{v}_t = -\nabla\frac{\delta E}{\delta \mu_t}(\mu_t)$, where $\frac{\delta E}{\delta \mu_t}$ is called the *first variation* of E at μ_t, with evolved directions constrained on the tangent space of the probability manifold. Consequently, gradient flows on $\mathcal{P}(\Omega)$ can be written as

$$\partial_t \mu_t = -\nabla \cdot (\mathbf{v}_t\, \mu_t) = \nabla \cdot \left(\mu_t \nabla(\frac{\delta E}{\delta \mu_t}(\mu_t))\right)\,. \qquad (4)$$

Solving by discrete gradient flows An exact solution to the WGF formula (4) is generally infeasible. A typical solution is to approximate the continuous-time solution of (4) with discrete-time flows, called discrete gradient flows (DGFs). Denote $\mathcal{P}_s(\mathbb{R}^d)$ to be the space of probability measures with finite 2nd-order moments, and define the following optimization problem with a step size ϵ:

$$J_h(\mu) \triangleq \arg\min_{\nu\in\mathcal{P}_s(\mathbb{R}^d)} \left\{\frac{1}{2\epsilon}W_2^2(\mu,\nu) + E(\nu)\right\}\,, \qquad (5)$$

where $W_2^2(\mu,\nu)$ denotes the Wasserstein distance between μ and ν. Here $E(\nu)$ is such that $p \triangleq \arg\min_\nu E(\nu)$ corresponds to the target distribution. The idea of DGFs is to approximate the continuous-time solution μ_t from (4) via a composition of a sequence of T/ϵ discrete solutions $(\tilde{\mu}_\ell)_{\ell=1}^{T/\epsilon}$ of (5), *i.e.*,

$$\tilde{\mu}_\ell \triangleq J_h(\tilde{\mu}_{\ell-1}) = J_h(J_h(\cdots \mu_0)) \approx \mu_t \,. \tag{6}$$

The DGF method is the gradient-descent analogy on Euclidean space for ν. One can show that when $\epsilon \to 0$, the solution from DGFs (6) converges to the true flow (4) for all ℓ (Craig 2014).

3 Self-Adversarially Learned Bayesian Sampling

In this section, we develop a GAN-based framework to efficiently solve the DGF problem (6), avoiding the computational complexity of the original particle-approximation-based methods (Chen et al. 2018). Based on this, more powerful and flexible approximations with a self-adversarial learning scheme are developed.

3.1 Reformulating POS as conditional GANs

We first specify the functional energy $E(\nu)$ in (5). For popular sampling methods such as SG-MCMC and SVGD, $E(\nu)$ has been shown to be the KL-divergence between $\nu(\mathbf{x})$ and the target distribution $p(\mathbf{x}\,|\mathcal{D})$[1]. In this case, the DGF method described above becomes the well-known Jordan-Kinderlehrer-Otto scheme (JKO) (Jordan, Kinderlehrer, and Otto 1998). For convenience, we instead define the functional energy as the Jensen-Shannon divergence (JSD) between $\nu(\mathbf{x})$ and $p(\mathbf{x}\,|\mathcal{D})$. Note the JSD also endows the convexity property, rendering a unique optimal solution as for the KL-divergence. Consequently, the DGF (5) becomes

$$J_h(\mu) \approx \arg\min_{\nu \in \mathcal{P}_s(\mathbb{R}^d)} \left\{ \frac{\alpha}{2\epsilon} W_2^2(\mu, \nu) + \mathsf{JSD}(\nu \| p) \right\}, \tag{7}$$

where α is a tunable parameter.

Now solving the WGF (4) is equivalent to composing results from a sequence of optimizations defined in (7) via (6). As a result, ν is optimized sequentially, each time conditioning on its previous value. In addition, the JSD is well-known to be the objective function of GAN. Consequently, the optimization problem (7) can be reformulated as a conditional GAN, where ν is defined as an implicit distribution induced by a conditional generator G. Specifically, G is designed to take an old sample and random noise as input, and outputs the updated sample. The W_2 term in (7) regularizes the outputs such that they are not too far away from their input samples. According to the GAN theory (Goodfellow et al. 2014), (7) is equivalent to the following objective:

$$\mathcal{L} = \min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_d(\mathbf{x})} [\log D(\mathbf{x})] \tag{8}$$
$$+\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})} [\log(1 - D(G(\tilde{\mathbf{x}}, \mathbf{z})))] + \frac{\alpha}{2\epsilon} W_2^2(p_G(\tilde{\mathbf{x}}), p(\tilde{\mathbf{x}}))$$

where $p_d(\cdot)$ denotes the true data distribution, $\tilde{\mathbf{x}}$ is the previous sample from the generator G whose implicit distribution is denoted as $p(\tilde{\mathbf{x}})$; $p_G(\tilde{\mathbf{x}})$ denotes the implicit distribution of the output $G(\tilde{\mathbf{x}}, \mathbf{z})$; and $D(\cdot)$ is a discriminator network to distinguish an input to be real or fake. A conditional generator is required because the output is correlated with the input via the W_2 term. The objective (8) is illustrated in Figure 1 (left).

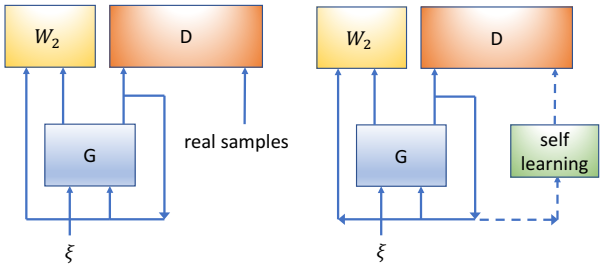

Figure 1: Graphs of the proposed framework with real samples (left) and a self-learning component (right). "G" represents generator, "D" represents discriminator, and "W_2" corresponds to the Wasserstein regularizer in (8). Dash lines indicates gradients not flowing back in backpropagation.

We note two differences between our formulation (8) and the standard GAN: *i*) Performing stochastic gradient descend (SGD) for learning is challenging due to the existence of the W_2 regularization term; and *ii*) The real sample $\mathbf{x}$ might not be available in training. In the following, we address the first problem by deriving an approximate form for W_2 in Section 3.2, by assuming the availability of real/true data $\mathbf{x}$. We then proceed to solve the second problem by proposing a self-adversarial-learning component to automatically adjust samples from the generator such that they approach real samples in Section 3.3.

3.2 Adversarially-learning Bayesian sampling

Approximating the Wasserstein regularizer Following (Chen et al. 2018), we use particle approximation to deal with the Wasserstein term in (8). Let $(\tilde{\mathbf{x}}_-^{(i)})_{i=1}^M$ be a minibatch of input samples from the generator (samples from last step), and $(\tilde{\mathbf{x}}^{(i)})_{i=1}^M$ be the output samples. Using samples/particles to approximate the W_2 in (8), $W_2^2(p_G(\tilde{\mathbf{x}}), p(\tilde{\mathbf{x}}))$ is approximated as

$$W_2^2(p_G(\tilde{\mathbf{x}}), p(\tilde{\mathbf{x}})) \approx \inf_{p_{ij}} \sum_{i,j} p_{ij} c(\tilde{\mathbf{x}}^{(i)}, \tilde{\mathbf{x}}_-^{(j)}) \tag{9}$$
$$s.t. \ \sum_j p_{ij} = \frac{1}{M}, \ \ \sum_i p_{ij} = \frac{1}{M} \,,$$

where $c(\tilde{\mathbf{x}}^{(i)}, \tilde{\mathbf{x}}_-^{(j)}) \triangleq \|\tilde{\mathbf{x}}^{(i)} - \tilde{\mathbf{x}}_-^{(j)}\|_2^2 \triangleq c_{ij}$. Now the goal is turned into solving for the optimal $\{p_{ij}\}$. Introducing Lagrangian multipliers $\{\alpha_i, \beta_i\}$ to deal with the constraints, and adding an entropy regularized term for $\{p_{ij}\}$, the dual problem can be written as

$$\mathcal{L}(\{p_{ij}\}, \{\alpha_i\}, \{\beta_i\}) = \lambda \sum_{i,j} p_{ij} \log p_{ij} + p_{ij} c_{ij}$$
$$+ \sum_i \alpha_i (\sum_j p_{ij} - \frac{1}{M}) + \sum_j \beta_j (\sum_i p_{ij} - \frac{1}{M}) \tag{10}$$

[1]Though the metric for SVGD is defined as a variant of the Wasserstein distance called $\mathcal{H}$-Wasserstein distance (Liu 2017).

Solving for (10), the optimal p_{ij} endows a forms of $p^*_{ij} = u_i e^{-c_{ij}/\lambda} v_j$, where $u_i \triangleq e^{-\frac{1}{2}-\frac{\alpha_i}{\lambda}}$, $v_j = e^{-\frac{1}{2}-\frac{\beta_j}{\lambda}}$. Now substituting the optimal p^*_{ij} back to (9), the Wasserstein distance can be approximated as:

$$W_2^2(p_G(\tilde{\mathbf{x}}), p(\tilde{\mathbf{x}})) \approx \gamma \sum_{i,j} c_{ij} e^{-c_{ij}/\lambda} , \qquad (11)$$

where the original u_i and v_j have been merged into the constant γ for simplicity.

Adversarially-learned Bayesian sampling Given the approximation (11) for W_2 and real training data $\mathbf{x}$, gradients of the generator parameters can be readily calculated by backpropagation, making generator update with SGD readily available. We call this version of our framework Adversarially-learned Markovian chain (AL-MC), with detailed algorithm given in the Supplementary Material (SM) on our homepage.

3.3 Self-adversarially learned Bayesian sampling

A more challenging setting in practice is that real samples are not readily available, *e.g.*, in posterior sampling where only an (unnormalized) posterior distribution is provided. This section addresses the problem of how to learn a generator to generate effective samples only with such information.

Our basic idea is to add a *self-learning* module that can automatically adjust the current samples from the generator to approach a target distribution. These adjusted samples are then used as real data to update both the generator and discriminator gradually. This idea is illustrated in Figure 1 (right). Based on an unnormalized target distribution, we consider two settings:

Real data generation with approximate Bayesian sampling In this case, one is assume to be able to directly draw approximate samples from a target distribution, based on the previous outputs of the generator. This procedure can be done by adopting existing effective approximate samplers such as SG-MCMC and SVGD. Specifically, let the previous outputs from the generator be $\{\tilde{\mathbf{x}}_t^{(i)}\}_{i=1}^M$, the real samples for next generator update are then approximated as

$$\{\mathbf{x}_{t+1}^{(i)}\}_{i=1}^m = \mathsf{SG\text{-}MCMC}(\{\tilde{\mathbf{x}}_t^{(i)}\}_{i=1}^m) \quad \text{or} \quad \mathsf{SVGD}(\{\tilde{\mathbf{x}}_t^{(i)}\}_{i=1}^m) ,$$

where $\mathsf{SG\text{-}MCMC}(\cdot)$ or $\mathsf{SVGD}(\cdot)$ means running one or several SG-MCMC/SVGD updates on the input samples toward the target distribution. Based on these approximate real samples, the generator update then proceeds as what AL-MC does described in the last section.

One can easily see that the effectiveness of this learning scheme highly depends on the approximate sampling algorithms, *e.g.*, SG-MCMC or SVGD. Empirically, we usually observe samples from the generator collapsed to one mode on a multi-mode target distribution. The reason is that when modes are too far away from each other, making samples jump from one mode to another with SG-MCMC or SVGD typically takes a long time, and sometimes they even fail to move. This misleads the generator to be trained to generate samples only on one mode of the target distribution. In the following, we propose a novel self-adversarial learning module to overcome this issue, which does not even need samples from an approximate sampler.

Real-data generation with self learning Developed on ideas from (Han and Liu 2018), we propose a self-learning scheme which can automatically adjust samples from a generator by only relying on gradient information from the current sample density, instead of on the true gradient information of the target distribution. Specifically, let the current induced distribution from the generator be ν (an implicit distribution without an explicit form) with corresponding samples $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(M)}\}$. The output of the self-learning module in Figure 1 (right) is defined as:

$$\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i)} + \frac{\epsilon}{Z} \sum_{j=1}^M \omega_j \Big[\nabla_{\mathbf{x}^{(j)}} \log \nu(\mathbf{x}^{(j)}) \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) + \nabla_{\mathbf{x}^{(j)}} \kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \Big] , \qquad (12)$$

where $\omega_j = \nu(\mathbf{x}^{(j)})/p(\mathbf{x}^{(j)}|\mathcal{D})$, and $Z \triangleq \sum_j \omega_j$. According to (Han and Liu 2018), the distribution of $\{\mathbf{x}^{(i)}\}$ is guaranteed to asymptotically converge to $p(\mathbf{x}|\mathcal{D})$.

Note the issue in the above update is that there are no explicit forms for both $\nu(\mathbf{x})$ and $\nabla_{\mathbf{x}} \log \nu(\mathbf{x})$ as ν is an implicit distribution. Therefore, we adopt density estimation techniques for approximation. For $\nu(\mathbf{x})$, we use the popular kernel density estimator (Guidoum 2015), *i.e.*,

$$\nu(\mathbf{x}) \approx \frac{1}{M} \sum_{i=1}^M \kappa^*(\mathbf{x}, \mathbf{x}^{(i)}), \qquad (13)$$

where κ^* is a positive kernel in our paper. For $\nabla_{\mathbf{x}} \log \nu(\mathbf{x})$, we apply the recently developed Stein gradient estimator (Li and Turner 2018), defined as

$$\nabla_{\mathbf{x}} \log \nu(\mathbf{x}) \approx -(\kappa + \eta \mathbf{I})^{-1} \sum_{i=1}^M \nabla_{\mathbf{x}^{(i)}} \kappa^*(\mathbf{x}, \mathbf{x}^{(i)}) \qquad (14)$$

where κ is a kernel matrix with $\mathbf{K}_{ij} = \kappa^*(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. The Stein gradient estimator has been shown to significantly outperform existing gradient estimator methods (Li and Turner 2018). Note we use two forms of RBF kernels $\kappa^*(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ with bandwidth h^* and $\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ with bandwidth h to allow flexibility. κ is used in SVGD, and κ^* is used to approximate the density and the log-posterior gradient. Posterior sampling with (12) is an instance of the SVGD-without-gradient framework (Han and Liu 2018). We denote the update (12) as approximate gradient SVGD (AG-SVGD), and will show impressive performances compared to standard SG-MCMC or SVGD in the experiments.

Taking AG-SVGD as the self-learning module in Figure 1 (right), we obtain what is called self-adversarially-learning Markov chain (SAL-MC) sampler, described below.

The SAL-MC sampler The training procedure is given in the first part of Algorithm 1, where AG-SVGD is used to generate approximate real samples from a target distribution. This procedure would gradually guide the generator to generate real samples as the stationary distribution of AG-SVGD is the target distribution. In addition, this self-adjusted behavior makes samples jump out of local modes easier, leading to much better performance compared to the one using SG-MCMC or SVGD to generate real samples, as will be shown in the experiments. We also find that multiple updates of the discriminator per generator update can increase effective sample size (ESS). After training, only the

Algorithm 1: SAL-MC training and sampling

Input : $\tilde{\mathbf{x}}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$, m and $S = \emptyset$
Output: Transition operator G and samples set S
1. SAL-MC training;
for t=1, 2 $\cdots$ **do**
  // Adjust particles:
  $\mathbf{x}_t \leftarrow$ AG-SVGD($\tilde{\mathbf{x}}_{t-1}$)
  // Adversarial training between $\mathbf{x}_t$ *and* $G(\tilde{\mathbf{x}}_{t-1}, \boldsymbol{\xi})$:
  for i=1, 2 $\cdots m$ **do**
    train the discriminator with the objective (8)
  end
  train the generator with the objective (8)
  // Update adversarially-learned particles:
  $\tilde{\mathbf{x}}_t \leftarrow G(\tilde{\mathbf{x}}_{t-1}, \boldsymbol{\xi})$
  for i=1, 2 $\cdots m$ **do**
    $\tilde{\mathbf{x}}_t \leftarrow G(\tilde{\mathbf{x}}_t, \boldsymbol{\xi})$
  end
end
2. SAL-MC sampling;
for l=1, 2 $\cdots$ **do**
  $\tilde{\mathbf{x}}_l \leftarrow G(\tilde{\mathbf{x}}_{l-1}, \boldsymbol{\xi})$
  no MH step and directly add $\tilde{\mathbf{x}}_l$ to S
end

conditional generator G is adopted to generate a sequence of samples via the following generative process:

$$\tilde{\mathbf{x}}_{t+1} = G(\tilde{\mathbf{x}}_t, \boldsymbol{\xi}) \,, \tag{15}$$

where $\boldsymbol{\xi}$ is a random noise drawn from a simple distribution such as an isotropy multivariate Gaussian $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. In this way, G is taken as a transition operator of a Markov chain.

Unlike SG-MCMC and SVGD, no gradient information from the target distribution is needed, leading to fast sample generation only through forward passes of a neural network. In addition, the generator allows distant jumps between consecutive samples via the complex transformation in G, making samples mix faster. In practice, we do not apply any burn-in steps or thinning methods on the samples, while it still shows a good convergence property with high ESS, as will be demonstrated in the experiments.

4 Experiments

In this section, we first examine the effectiveness of AG-SVGD with stochastic gradient estimations for direct sampling, and compare it with standard SGLD and the recently proposed Annealed-SVGD (A-SVGD) (Han and Liu 2018). We then apply the proposed SAL-MC framework on a set of multi-mode synthetic distributions, as well as on Bayesian Logistic Regression (BLR) tasks. We compare SAL-MC with the recently developed A-NICE-MC method (Song, Zhao, and Ermon 2017), which is also based on adversarial training. Finally, we apply our AL-MC algorithm for image generation trained on real samples.

4.1 Verification of AG-SVGD

We conduct experiments on two multi-mode toy examples, a 5D Gaussian Mixture Model (GMM) distributed in an aggregation state and a challenging 2D-GMM distribution with distant modes and varied variances. We use the same RBF kernel and the median trick for AG-SVGD and A-SVGD. As suggested by (Welling and Teh 2011), a polynomially-decayed step size $\epsilon_t = a/(t+1)^{0.55}$ is used in SGLD for a fair comparison. We use samples to approximate the mean and variance of the distributions, measured by mean squared errors (MSE) w.r.t. true values. The results are averaged over 20 random runs with 500 iterations in each run. To be consistent with (Han and Liu 2018), we also use RBF kernels and the median trick when calculating maximum mean discrepancy (MMD) between the sample approximation and true distribution for all the three methods.

The relatively simple 5D mixture distribution endows 10 modes. Figure 2 compares the convergence results of the three methods by varying sample sizes. Different from A-SVGD, our AG-SVGD does not need gradient information from the target distribution. Surprisingly, however, AG-SVGD is comparable to A-SVGD when the number of particles is big enough. Furthermore, AG-SVGD obtains the best estimation of variance among the three algorithms. Moreover, AG-SVGD has a slightly better convergence property when sample size is small. In contrast, SGLD performs the worse due to the noisy updates and slow mixing samples.

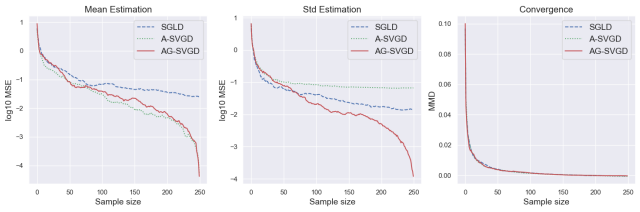

Figure 2: Simulation results on *Mog10*

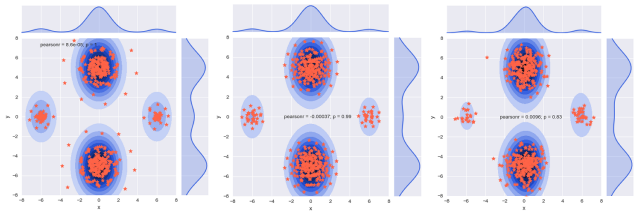

Figure 3: Samples on *Mog4*. From left to right: AG-SVGD, A-SVGD and SGLD.

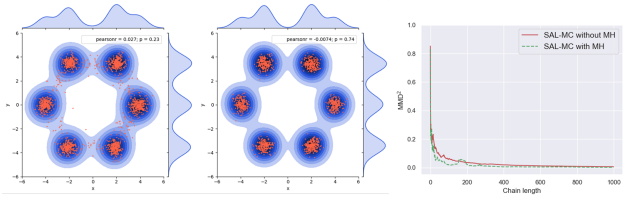

Figure 4: SAL-MC on *Mog6*. Left: without MH; Middle: with MH.

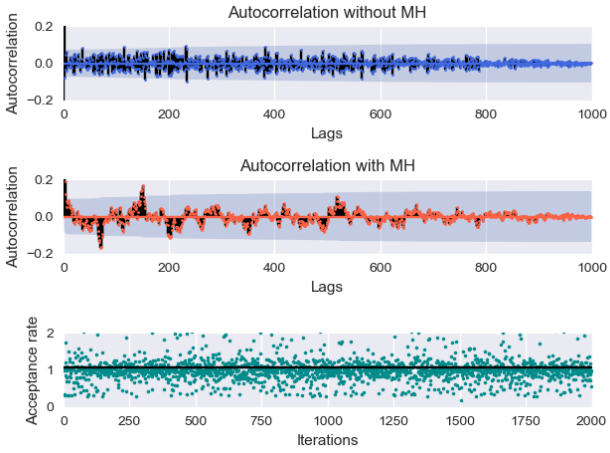

Figure 5: Convergence behavior of SAL-MC on *Mog6*. Acceptance rate in the third plot (without MH steps) denotes the ratio of the proposed-sample probability to current-sample probability, *i.e.*, $p(\tilde{\mathbf{x}}_{t+1})/p(\tilde{\mathbf{x}}_t)$)

For the challenging 2D-GMM dataset *Mog4*, the samples obtained are visualized in Figure 3. Again, we use the same bandwidth of $h = 0.01 * median$ for AG-SVGD and A-SVGD. Parameters of the gradient estimator are set among $h^* = \{0.5, 1, 2, 4, 8, 10\}$ and $\eta = \{0.05, 0.1, 1, 5, 10, 15\}$, selected by a grid search. The initial particles are empirically drawn from $\mathcal{N}(\mathbf{0}, 2.5\mathbf{I})$. As can be seen, both A-SVGD and SGLD somewhat fail to balance samples from different modes, whereas our AG-SVGD is able to travel better between low and high density regions, leading to a more accurate approximation. In the following, the well-validated AG-SVGD is applied in SAL-MC samplers as the self-learning module, which is tested on a number of datasets.

Table 1: ESS (2000 maximum) and ESS speed on synthetic distributions of both methods

ESS / (ESS/s)		*Ring*	*Mog2*	*Mog6*	*Ring5*
SAL-MC	-	**1635/121138**	**1435/72691**	**1212/65287**	-
	MH	1561/25624	1172/17531	978/12235	**414/16287**
A-NICE-MC	-	N/A	N/A	N/A	N/A
	MH	2000/43887	951/17216	889/11745	335/6022

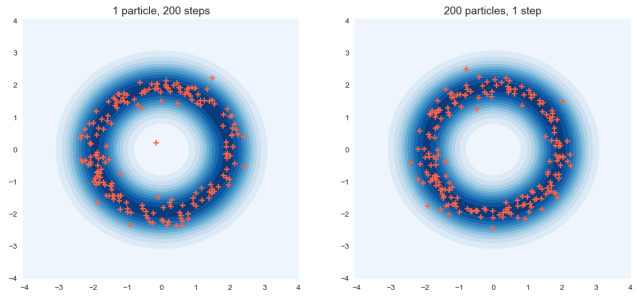

Figure 6: SAL sampler results of *Ring* under two settings

4.2 SAL-MC on synthetic datasets

Next we demonstrate the effectiveness of SAL-MC by comparing it with A-NICE-MC (Song, Zhao, and Ermon 2017). We adopt the *Ring*, *Mog2*, *Mog6* and *Ring5* datasets used in A-NICE-MC, and measure the efficiency of a MCMC method in terms of ESS (2000 maximum) and ESS per second. The smallest ESS among all dimensions is reported. More details are provided in SM. The stochastic term $\boldsymbol{\xi}$ in the algorithm is drawn from $\mathcal{N}(\mathbf{0}, 5\,\mathbf{I})$ for all experiments. Since A-NICE-MC requires a Metropolis-Hastings (MH) step to accept or reject a sample, we also test the MH step in our algorithm.

The results are shown in Table 1, where SAL-MC consistently outperforms A-NICE-MC. Interestingly, A-NICE-MC collapses without MH steps, whereas SAL-MC works similarly with or without a MH step. In addition, it is observed that both POS and SGLD achieve a very low ESS that is around 10. We also calculate the Gelman-Rubin convergence statistic $\hat{R}$ (Gelman et al. 1995; Brooks and Gelman 1998), a common convergence diagnostic using multiple chains to check for the convergence of an algorithm. Typically, a values of $\hat{R}$ close to one (*e.g.*, less than 1.1) is a good indicator of convergence. It is observed SAL-MC obtains $\hat{R} = 1.00$ using 32 chains for all tasks.

We further illustrate samples drawn from the *Mog6* example in the first two scatter plots of Figure 4. As can be seen, SAL-MC is able to learn the six modes reasonably well, no matter if it is with or without the MH step. Without MH, SAL-MC tends to be able to generate more samples in low-density regions in between different modes. Note appropriate injected noise $\boldsymbol{\xi}$ should be chosen because too small noise makes distant jumps difficult, while too large noise makes the convergence slower with fewer effective samples. The rightmost plot of Figure 4 together with Figure 5 demonstrate the proposed SAL-MC endows nice convergence properties in terms of *rapid decay of MMD* (efficiency and low bias), *low autocorrelation* (low variance), and *high acceptance rates*.

We also compare the settings of single and multiple chains in SAL-MC with equal number of samples. Specifically, for the single-chain setting, a sample is initialized from $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, followed by 200 updates from SAL-MC to form 200 samples. For the multiple-chain setting, we initialize 200 samples followed by a single-step update to form the

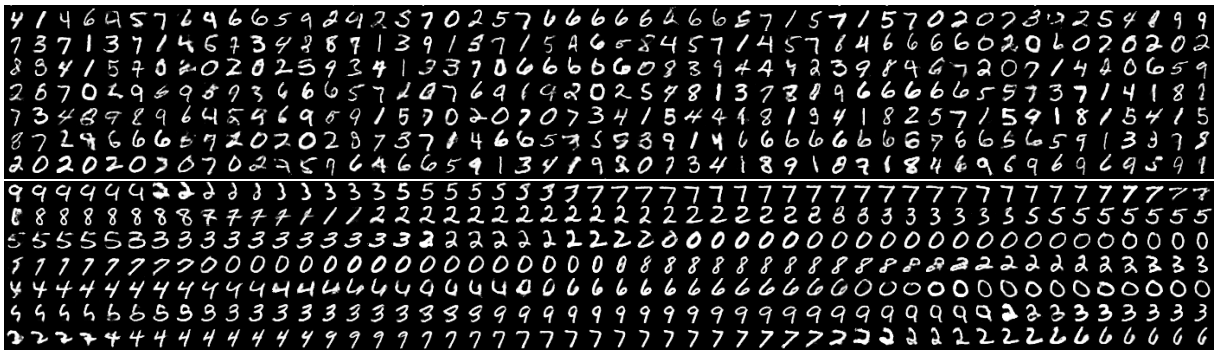

Figure 7: Sampling results of digits for $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, 0.5\mathbf{I})$ (top) and $\boldsymbol{\xi} = 0$ (bottom). Each row represents 50 consecutive samples from the same chain. The sampler with $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, 0.5\mathbf{I})$ mixes well by generating samples from different modes easily.

final 200 samples. The results on *Ring* are visualized in Figure 6, from which we can see both settings perform similarly with well approximate samples from the target distribution.

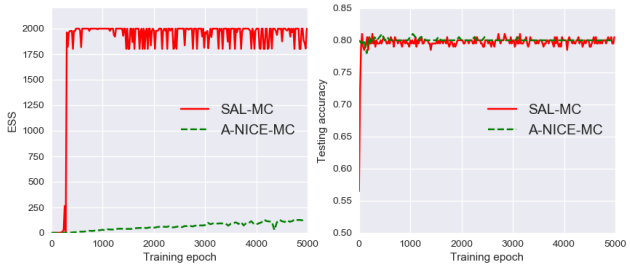

Figure 8: ESS and test accuracy with respect to training epochs averaged by 10 different runs. (*German* dataset)

Figure 9: Sampling results of faces for $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, 0.5\mathbf{I})$ (top) and $\boldsymbol{\xi} = 0$ (bottom). Each row represents 24 consecutive samples from the same chain.

4.3 SAL-MC for BLR

We further compare SAL-MC with A-NICE-MC on several BLR tasks for scalable Bayesian posterior sampling. Three datasets, *Heart* (532-13), *Australian* (690-14) and *German* (1000-24), are used, where $(a\text{-}b)$ means a dataset with a samples and feature dimensionality b. The mini-batch size for training is 64; and the injected noise $\boldsymbol{\xi}$ is drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ for all tasks.

The ESS and ESS speed are shown in Table 2, which are calculated after 20K training epochs. With a MH step, SAL-MC tends to generate relatively highly-correlated samples with high rejection rates. Nerveless, it is still better than A-NICE-MC, which, unfortunately, does not even work without MH steps. We also evaluate BLR in terms of testing accuracies, which are calculated by averaging over 10 runs with 32 chains. The models are trained on a random 80% of the datasets and tested on the remaining 20% in each run. The results are respectively 84.10%, 88.38% and 80.32% on *Heart*, *Australian* and *German* datasets for SAL-MC, which are the same as A-NICE-MC (Song, Zhao, and Ermon 2017). However, as shown in Figure 8, SAL-MC obtains much better ESS. The experiments also indicate A-NICE-MC must take 20K training iterations to get the highest ESS (which is much lower than SAL-MC as shown in Table 2).

Table 2: ESS (2000 maximum) and ESS speed for BLR.

ESS / (ESS/s)		*Heart*	*Australian*	*German*
SAL-MC	-	**1683/93140**	**1385/70655**	**1897/86512**
	MH	1/10	1/8	1/14
A-NICE-MC	-	N/A	N/A	N/A
	MH	663/10939	596/9834	483/7848

4.4 AL-MC for image synthesize

We finally test AL-MC, a variant with only real samples available in training, for image synthesis on MNIST and CelebA datasets. The balance factor of Wasserstein regularization term in (8) is set to $\alpha = 0.1$.

The generated samples are visualized in Figure 7 and Figure 9. For MNIST, when a sampler is well trained, the sample distribution over 10 classes on the generated samples should be relatively uniform in order to match that of the training-data statistics. To verify this, we classify the generated samples with a pre-trained deep-CNN MNIST classifier (with a 99.1% accuracy). We calculate the *class distribution* on two settings with different injected noise, $\boldsymbol{\xi} = 0$ and $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, 0.5\mathbf{I})$. The empirical distributions of the ten digits are indeed even. We also plot the digits generated from the learned Markov chain in Figure 7, from which we can see in the case of $\boldsymbol{\xi} \sim \mathcal{N}(\mathbf{0}, 0.5\mathbf{I})$, the sampler can make distant jumps easily; whereas when $\boldsymbol{\xi} = 0$, transitions seem to be

very smooth, thus it needs longer time to generate all digits from the ten classes. Similar results on CelebA, though not as obvious, are observed in Figure 9. More detailed results are included in the SM.

5 Conclusion

Motivated by the WGF theory, we present self-adversarially learned Bayesian sampling, a generative model learning to draw samples from a target distribution. Two settings, *i.e.* whether or not true samples are provided as training data, are considered. When learning without true samples, a self-learning mechanism is proposed to automatically adjust samples from the current generator to approach a target distribution. Our method is fully automatic, and is fast and effective in sample generation. Experiments on both synthetic and real datasets demonstrate the effectiveness of our framework, which endows good convergence property while is able to generate much less correlated samples, relative to existing methods.

References

Ambrosio, L.; Gigli, N.; and Savaré, G. 2005. *Gradient Flows in Metric Spaces and in the Space of Probability Measures*. Lectures in Mathematics ETH Zürich.

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

Brooks, S. P., and Gelman, A. 1998. General methods for monitoring convergence of iterative simulations. *Journal of computational and graphical statistics* 7(4):434–455.

Chen, C.; Li, C.; Chen, L.; Wang, W.; Pu, Y.; and Carin, L. 2017. Continuous-time flows for deep generative models. In *arXiv:1709.01179*.

Chen, C.; Zhang, R.; Wang, W.; Li, B.; and Chen, L. 2018. A unified particle-optimization framework for scalable Bayesian sampling. In *UAI*.

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*.

Craig, K., ed. 2014. *The Exponential Formula for the Wasserstein Metric*. PhD thesis, The State University of New Jersey.

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

Feng, Y.; Wang, D.; and Liu, Q. 2017. Learning to draw samples with amortized stein variational gradient descent. In *UAI*.

Gelman, A.; Carlin, J. B.; Stern, H. S.; and Rubin, D. B. 1995. *Bayesian data analysis*. Chapman and Hall/CRC.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Neural Information Processing Systems (NIPS)*.

Guidoum, A. C. 2015. Kernel estimator and bandwidth selection for density and its derivatives. Technical Report The kedd Package, Version 1.0.3.

Haarnoja, T.; Tang, H.; Abbeel, P.; and Levine, S. 2017. Reinforcement learning with deep energy-based policies. In *ICML*.

Han, J., and Liu, Q. 2018. Stein variational gradient descent without gradient. In *ICML*.

Jordan, R.; Kinderlehrer, D.; and Otto, F. 1998. The variational formulation of the Fokker-Planck equation. *SIAM Journal on Mathematical Analysis* 29(1):1–17.

Li, Y., and Turner, R. E. 2018. Gradient estimators for implicit models. In *ICLR*.

Liu, Q., and Wang, D. 2016. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *NIPS*.

Liu, Y.; Ramachandran, P.; Liu, Q.; and Peng, J. 2017. Stein variational policy gradient. In *UAI*.

Liu, Q. 2017. Stein variational gradient descent as gradient flow. In *NIPS*.

Osband, I., and Van Roy, B. 2017. Why is posterior sampling better than optimism for reinforcement learning? In *ICML*.

Raginsky, M.; Rakhlin, A.; and Telgarsky, M. 2017. Nonconvex learning via stochastic gradient Langevin dynamics: a nonasymptotic analysis. In *COLT*.

Song, J.; Zhao, S.; and Ermon, S. 2017. A-nice-mc: Adversarial training for mcmc. In *NIPS*.

Teh, Y. W.; Thiery, A. H.; and Vollmer, S. J. 2016. Consistency and fluctuations for stochastic gradient Langevin dynamics. *JMLR* 17(1):193–225.

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

Zhang, R.; Li, C.; Chen, C.; and Carin, L. 2018. Learning structural weight uncertainty for sequential decision-making. In *AISTATS*.

Zhang, Y.; Liang, P.; and Charikar, M. 2017. A hitting time analysis of stochastic gradient langevin dynamics. In *COLT*.