Speedup Methods for Stein Variational Gradient Descent

Min Hyung Kang
Department of Computer Science
Dartmouth College
Hanover, NH 03755
`Min.Hyung.Kang.15@dartmouth.edu`

Abstract

We introduce several variations of recently introduced Stein Variational Gradient Descent algorithm that reduces the original quadratic runtime complexity to linear time. Our models approximate the update for each particle by either looking at subset of original particles or redefining kernel values using another set of reference points which we name as induced points. We utilize various variance reduction techniques and introduce novel adversarial updates which counter-intuitively increases the discrepancy between particles to gain better updates. The models are tested on various datasets, and results suggest that compared to the original algorithm our models are not only faster but also often show better performance.

1 Introduction

Bayesian inference is a core task of the field of machine learning. We seek to compute the posterior distribution given the prior, likelihood and model evidence using the Bayes' Theorem, a simple yet powerful relationship between these probabilities. Though the equation itself appears trivially simple, that task at hand is not so easy; computing the model evidence, which acts as the normalization constant, involves summing over all possible models which is very often intractable. The field has seen two major approaches to this problem.

One method which is relatively more accessible and easy to carry out is sampling. Markov chain Monte Carlo (MCMC) techniques generate samples by constructing a Markov chain whose equilibrium is the desired distribution. The generated samples can be used to approximate the posterior samples. However, the sampling methods are often slow and are difficult to reach convergence (Murphy [21], Bishop [3]). Another method is variational inference algorithms. These type of algorithms turn the problem into an optimization problem, where they seek to approximate the distribution of interest by optimizing the parameters of a family of tractable distributions to minimize the discrepancy between the approximation and the original. These algorithms also have limitations in that 1. we need to derive the model for each problem, which is often not a trivial task 2. one needs to be careful in their choice of family of distributions so that it is tractable yet expressive enough to capture the true posterior.

Recently introduced approach of Stein Variational Gradient Descent (Liu and Wang [19]) uses Stein's identity to deterministically transport a set of points, or particles, to match the distribution of interest. The fact that it is not parametric and only requires score function of the target distribution allows it to be easily applicable to variety of problems. As the complexity and dimension of the problem increases, it is natural for one to use many particles. However, the current algorithm has runtime complexity that is quadratic with respect to the number of particles. Such an algorithm may become too expensive to run as number of particles increase.

In this work, we attempt to evaluate various methods that run in linear time with respect to the number of particles. We first explore ways to approximate the gradient by using a random subset of particles, denoted in this work as **subparticles**, rather than considering every possible pair of particles. We utilize the variance reduction scheme of control functionals (Oates et al. [22]) to further equip the subparticles with importance weights. Even with such a naive approximation we see performance comparable to that of the original algorithm.

We then explore ways to approximate the kernel between every pair of particles. We introduce another set of points, called **induced points**, and define **induced kernel** which redefines kernel function between two particles in terms of the induced points. We derive the SVGD update using this formulation, which leads to a linear time approximation of the algorithm that often even outperforms the original algorithm.

We further explore enhancing this method by looking at adversarial updates. In the field of deep learning, Generative Adversarial Networks (Goodfellow et al. [8]) have recently gained a notable interest. The idea of making two networks compete in an adversarial nature have seen great progress, which have led to incredible results in various fields, especially computer vision (Salimans et al. [26], Arjovsky et al. [1]). We apply a similar concept to improve our algorithm, where we iteratively maximize and minimize a discrepancy measure by alternating between updates of induced points and particles. We observe that such method, though demanding in computation time, often outperforms the original algorithm.

Our contribution in this paper can be summarized as follows :

- We introduce fast and efficient approximations of SVGD that run in linear time rather than the original quadratic time with respect to the number of particles.
- We introduce a novel formulation of kernels and an adversarial update method that runs in linear time but outperforms the original algorithm.

Outline This paper is organized as follows. Section 2 introduces background information regarding Stein's identity and formulation of Stein Variational Gradient Descent (SVGD) using Kernelized Stein Discrepancy (KSD). We present our main results in Section 3, where we explore various methods to improve performance of SVGD. In section 4 related works are discussed. We present the experimental results in section 5 and conclude the paper in section 6.

2 Background

This paper relies heavily on the notion of function spaces, namely Reproducing Kernel Hilbert Space (RKHS). For the sake of readability we leave the detailed background and formulation in the appendix. We provide brief overview in the following section; reader is encouraged to visit the appendix for whichever part one requires more information on. Throughout this paper, we assume all the vectors are column vectors unless otherwise noted.

For further reference, we point towards several materials that offer varying levels of coverage of the material related to this paper. Poggio et al. [23] provides a gentle introduction to functional analysis and Hilbert spaces. Daume III [5] would be an appropriate reading for anyone from no background knowledge to obtain general knowledge regarding RKHS. Muandet et al. [20] covers extensive topics regarding kernel methods and RKHS, and is recommended for anyone who wants more in-depth exposure to the topic.

2.1 Bayesian Inference

Let x be a random variable of interest in space $\mathcal{X} \subset \mathbb{R}^d$. Assume that we have a dataset of n i.i.d observations : $D = \{d_i\}_{i=1}^n$. Given a prior distribution $p_0(x)$, we are interested in the posterior distribution $p(x|D) = \tilde{p}(x)/Z$ where $\tilde{p}(x) = p_0(x)\prod_{i=1}^n p(x|d_i)$ and Z is the normalization constant (also called model evidence) $\int \tilde{p}(x)dx$. The problem often arises in computing Z, for we have to sum over all possible models which is computationally intractable. Variational inference methods attempt to approximate this posterior distribution by finding the appropriate function from a family of tractable distributions. That is, the problem attempts to find $q(x) \approx p(x|D)$ where $q \in Q$ and Q is the family of tractable probability distributions over x. Stein Variational Gradient Descent

(SVGD), the algorithm this paper focuses on and attempts to improve on, takes a slightly different approach in that it moves particles to approximate the distribution rather than optimize parameters of family of functions.

2.2 Mathematical Setup

Assume we have a positive definite kernel $k(x, x') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let $\mathcal{S}$ be a set of functions $k_x : \mathcal{X} \to \mathbb{R}$ such that $k_x(x') = k(x, x')$. Then Reproducing Hilbert Kernel Space (RKHS) $\mathcal{H}$ of the kernel $k(x, x')$ is the closed set of $\{f = \sum_i^n \alpha_i k_{x_i}; k_{x_i} \in S, \alpha_i \in \mathbb{R}, n \in \mathbb{N}\}$ with inner product $\langle f, g \rangle_H = \sum_{i,j} \alpha_i \beta_j k(x_i, x_j)$ where $g = \sum_j^m \beta_j k_{x_j}$ and $k_{x_j} \in S, \beta_j \in \mathbb{R}$. Now we extend this to the space of vector functions. Define $\mathcal{H}^d$ as the space of vector functions, where each element is an element of $\mathcal{H}$. That is, $f = [f_1 \dots f_d]^T \in \mathcal{H}^d$ and $f_1 \dots f_d \in \mathcal{H}$. We define the inner product to be $\sum_{i=1}^d \langle f_i, g_i \rangle_{\mathcal{H}}$, where $g = [g_1 \dots g_d]^T \in \mathcal{H}$

2.3 Stein's Identity

Assume that $p(x)$ is a smooth (continuous and differentiable) function with support $\mathcal{X}$ as before. Then **score function** $s_p(x)$ is defined as the following :

$$s_p(x) = \nabla_x \log p(x) = \frac{\nabla_x p(x)}{p(x)} \tag{1}$$

We say that a smooth vector function f is in **Stein Class of p** if it satisfies the following property :

$$\int_{x \in \mathcal{X}} \nabla_x \left(f(x) p(x) \right) dx = 0 \tag{2}$$

Definition 2.1 (Stein's Operator). *We define* Stein's operator $\mathcal{A}_p$ *as follows :*

$$\mathcal{A}_p f(x) = s_p(x) f(x) + \nabla_x f(x) \tag{3}$$

One can note that Stein's operator is a linear operator. That is, if $f(x) = \frac{1}{m} \sum_{i=1}^m f_i(x)$. then $\mathcal{A}_p f(x) = \frac{1}{m} \sum_{i=1}^m \mathcal{A}_p f_i(x)$. The proof follows from the fact that we are simply doing multiplication with $s_p(x)$ for the first term and the property that $\nabla f(x) = \frac{1}{m} \sum_i^m \nabla_x f_i(x)$ for the second term.

Theorem 2.1 (Stein's identity). *For any smooth vector function f that is in Stein class of p,*

$$\mathbb{E}_p \left[\mathcal{A}_p f(x) \right] = \mathbb{E}_p \left[s_p(x) f(x) + \nabla_x f(x) \right] = 0 \tag{4}$$

In other words, Stein's operator, when applied to smooth vector function f in the Stein class of p, is expected to yield a vector of 0s. Stein's identity provides the main framework with which we work to derive all the following applications.

2.4 Kernelized Stein Discrepancy

Using Stein's identity, we can establish a notion of **Stein discrepancy**. Consider a different distribution q whose support is also in $\mathcal{X}$. For any smooth vector function $\phi(x) = [\phi_1 \dots \phi_d]$, if we compute the expectation of $\mathcal{A}_p^x \phi(x)$ where $x \sim q(x)$, Stein's identity no longer holds and this would be a nonzero value. Hence, we can use this value as a measurement of how much a function deviates from another function. Specifically, we find the function $\phi(x)$ out of function set $\mathcal{F}$ that violates Stein's identity the most, and use the value to represent the discrepancy between p and q.

Definition 2.2 (Stein Discrepancy).

$$\mathbb{S}(q, p) = \max_{\phi \in \mathcal{F}} \{ \mathbb{E}_{x \sim q} \left[trace(\mathcal{A}_p \phi(x)) \right] \} \tag{5}$$

Algorithm 1 Stein Variational Gradient Descent

```
Input: A target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$.
Input: A positive definite kernel $k(x, x')$
Output: A set of particles $\{x_i\}_{i=1}^n$ that approximates the target distribution.
  for $t = 1 : maxIter$ do
    $\tilde{\psi}(x) = \frac{1}{n}\sum_{j=1}^{n}\left[k_h(x_j^t, x)\nabla_{x_j^t}\log p(x_j^t) + \nabla_{x_j^t}k_h(x_j^t, x)\right]$
    $x_i^{t+1} \leftarrow x_i^t + \epsilon^t\tilde{\psi}(x_i^t)$        where $\epsilon^t$ is the step size at the $t$-th iteration.
  end for
```

Note that this is a difficult optimization problem where one has to be discreet in one's choice of $\mathcal{F}$. Liu and Lee [17] proposes bypassing this problem by maximizing $\phi(x)$ in unit ball of RKHS. They define this evaluation method as Kernelized Stein Discrepancy, and show that it is a valid discrepancy metric (see appendix). We can approximate this discrepancy using the following theorem :

Theorem 2.2. *Assume p, q are smooth densities and that $k(x, x')$ is in the Stein class of p. Define*

$$u_p(x, x') = s_p(x)^T k(x, x') s_p(x') + s_p(x)^T \nabla_{x'} k(x, x') + s_p(x')^T \nabla_x k(x, x') + trace(\nabla_{x,x'} k(x, x')) \quad (6)$$

then

$$\mathbb{S}(q, p) = \mathbb{E}_{x,x'\sim q}\left[u_p(x, x')\right] \quad (7)$$

Hence, we can approximate the value of $\mathbb{S}(q, p)$ by drawing samples from q and taking empirical average of $u_p(x, x')$.

2.5 Stein Variational Gradient Descent

Variational inference is approximating a (usually intractable) distribution $p(x)$ using a simpler distribution $q(x)$ by minimizing divergence measure between two distributions. One of the most widely used measure is KL divergence :

Definition 2.3 (KL-divergence)**.** *Kullback Leibler (KL) divergence is a discrepancy measure defined as follows :*

$$KL(q||p) = \int q(x) \log \frac{q(x)}{p(x)} dx \quad (8)$$

We choose optimal q^* among family of functions $\mathcal{F}$ that minimizes KL-divergence between q^* and p.

Liu and Wang [19] investigates Stein Variational Gradient Descent (SVGD) Algorithm, which utilizes Stein operator to deterministically transport particles to match a distribution of interest. They observe the link between Stein operator and derivative of KL divergence. Assume we perform small perturbation $T(x) = x + \epsilon\phi(x)$ to distribution q. Then we can find that derivative of KL divergence between q distribution after T and p to equal the expected value of Stein operator on $\phi(x)$.

Theorem 2.3 (Connection between Stein operator and KL divergence)**.** *Let $T(x) = x + \epsilon\phi(x)$ and $q_{[T]}(z)$ the density of $z = T(x)$ when $x \sim q(x)$ we have*

$$\nabla_\epsilon KL(q_{[T]}||p)|_{\epsilon=0} = -\mathbb{E}_{x\sim q}\left[tr(\mathcal{A}_p\phi(x))\right] \quad (9)$$

They then show that the optimal perturbation gives negative gradient of KL divergence which equals KSD.

$$\nabla_\epsilon KL(q_{[T]}||p)|_{\epsilon=0} = -\mathbb{S}(q, p) \quad (10)$$

From this intuition, the authors propose iterative algorithm which transports particles deterministically. The full algorithm is shown in Algorithm 1. For each iteration t, we update the particles $x_i^{t+1} \leftarrow$

Algorithm 2 Random Subset SVGD

```
Input: A target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$.
Input: A positive definite kernel $k(x, x')$
Output: A set of particles $\{x_i\}_{i=1}^n$ that approximates the target distribution.
  for $t = 1 : maxIter$ do
    Let $\{y_j^t\}_{j=1}^m$ be randomly selected $m$ particles from $\{x_i^t\}_{i=1}^n$.
    $\tilde{\psi}(x) = \frac{1}{m} \sum_{j=1}^{m} \left[ k_h(y_j^t, x) \nabla_{y_j^t} \log p(y_j^t) + \nabla_{y_j^t} k_h(y_j^t, x) \right]$
    $x_i^{t+1} \leftarrow x_i^t + \epsilon^t \tilde{\psi}(x_i^t)$        where $\epsilon^t$ is the step size at the $t$-th iteration.
  end for
```

$x_i^t + \epsilon^t \hat{\psi}^*(x_i^t)$ where $\hat{\psi}^*(x) = \frac{1}{n} \sum_{j=1}^n \left[k(x_j^t, x) \nabla_{x_j^t} \log p(x_j^t) + \nabla_{x_j^t} k(x_j^t, x) \right]$ and ϵ^t is the step size at the t-th iteration. This is equal to reducing the KL divergence by step size multiplied by KSD.

Upon more careful observation of the terms, we can understand the different roles of each term. The first term is a weighted sum of the gradient, and transports the particles to the direction of high probability regions in the probability distribution of interest. The second term, on the other hand, acts as a repulsive force between the particles. That is, it prevents the particles from collapsing into local modes.

Kernels The kernel that we use throughout the whole research is the radial basis function (RBF) kernel, also called the Gaussian kernel. RBF kernel between two samples x, x' takes form of $k(x, x') = \exp\left(-\frac{||x-x'||^2}{2h^2}\right)$. It is equipped with one parameter, h, which we call as the *bandwidth* of the kernel. We use this kernel for it is widely used and meets the properties that the algorithms require.

3 Speedup Methods for SVGD

In this paper we investigate two major ways to reformulate SVGD: random subset and induced points. Each method decreases the complexity of the original algorithm from quadratic to linear time with respect to number of particles. We also try to improve each method using control functional and adversarial updates. In this section, we introduce each method in order.

3.1 Random Subset

The simplest method one can think of is choosing a random subset of particles to compute the update. For each iteration ℓ, we let $\{y_1^\ell, \ldots, y_m^\ell\}$ be a random subset of our particles $\{x_1^\ell, \ldots, x_n^\ell\}$ where $m \leq n$. Note that if $m = n$ we have the original SVGD algorithm. We expect this method to perform faster than the original method, as the complexity is $O(n * m)$ to update all the particles compared to the original model's $O(n^2)$. More discussion regarding the runtime can be found in section 3.5. Performance-wise, the natural expectation is that it give worse estimate than the original method as we are using less information. The algorithm itself is very similar to the original SVGD, and is presented in Algorithm 2.

A natural question that arise when we perform random subset is how we choose the subparticles. That is, we are not limited to choosing m points only from our particles. We consider various options; first, we try performing k-means with m clusters and using cluster centers. Then we try Herding method which creates a mapping from datapoints to generate samples that approximate a probability distribution and generate m points from current particles (Welling [28], Chen et al. [4]). We do not present our investigation, but the simple method of choosing subparticles from current particles is very effective in its performance and runtime compared to above mentioned techniques. The question, then becomes, can we choose our subparticles in a manner that helps our performance? Or can we imbue chosen subparticles with weights such that our updates are more effective?

Algorithm 3 Random Subset SVGD with Control Functional

```
Input: A target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$.
Input: A positive definite kernel $k(x, x')$
Output: A set of particles $\{x_i\}_{i=1}^n$ that approximates the target distribution.
  for $t = 1 : maxIter$ do
    Let $\{y_j^t\}_{j=1}^m$ be randomly selected $m$ particles from $\{x_i^t\}_{i=1}^n$.
    Let $v^t = \frac{\mathbf{1}^T(\mathbf{K_0}+\lambda m\mathbf{I})^{-1}}{1+\mathbf{1}^T(\mathbf{K_0}+\lambda m\mathbf{I})^{-1}\mathbf{1}}$ where $(\mathbf{K_0})_{i,j} = k(y_i, y_j)$
    $\tilde{\psi}(x) = \sum_{j=1}^{m} w_j^t \left[k_h(y_j^t, x)\nabla_{y_j^t} \log p(y_j^t) + \nabla_{y_j^t} k_h(y_j^t, x)\right]$    where $w_i^t = \frac{v_i^t}{\sum_j v_j^t}$
    $x_i^{t+1} \leftarrow x_i^t + \epsilon^t \tilde{\psi}(x_i^t)$    where $\epsilon^t$ is the step size at the $t$-th iteration.
  end for
```

3.2 Control Functional

Once we randomly choose m subparticles among the particles, we can imagine giving each particle different weights. When we want to estimate a value, we often utilize the method of control variate, a function whose mean is known and whose behavior is correlated with the value of interest, to reduce the variance of an unbiased estimator. Oates et al. [22] develops the notion of *control functional*, which resembles control variates in its motivation yet follows a different formulation. A similar approach was taken by Liu and Lee [18], which proposes selecting optimal weights by minimizing the empirical Stein discrepancy. Even though this method has benefits of enforced positive weights and hence often obtains more stable results, the optimizing procedure causes extra computational overhead for the algorithm which does not suit our purpose. Hence, we follow the formulation of control functional.

Given a positive definite kernel $k(x, x')$, define a kernel k_0 as follows :

$$k_0(x, x') = \nabla_x \nabla_{x'} k(x, x') + u(x)\nabla_{x'} k(x, x') + u(x')\nabla_x k(x, x') + u(x)u(x')k(x, x') \quad (11)$$

Assume we are interested in the value of function $f(x)$, and want to approximate it with $\hat{f}(x)$. Then the control functional estimator $\mu(\hat{f}(x))$ is the following[1] :

$$\mu(\hat{f}(x)) = \frac{\mathbf{1}^T(\mathbf{K_0} + \lambda m\mathbf{I})^{-1}}{1 + \mathbf{1}^T(\mathbf{K_0} + \lambda m\mathbf{I})^{-1}\mathbf{1}} f_0 \quad (12)$$

where $(K_0)_{i,j} = k_0(x_i, x_j)$ and $f_0 = [f(x_1) \dots f(x_n)]$. We can then consider the portion multiplied to f_0 as weight for each estimated point. Hence, we compute such weights for the case of subparticles and use the weights to take weighted average of the update. Lastly, we found that normalizing the weights ($w_{new} = \frac{w_{old}}{\sum w}$) slightly improved the performance. Hence, we normalize weights in our implementation, which is presented in Algorithm 3. While this method supposedly reduces variance of estimation and improve the performance, we now require computing kernel matrix between all pairs of subparticles as well as computing the inverse of this matrix. Hence, there is sacrifice of runtime for the sake of performance.

3.3 Random Induced Points

Previously, we simply used a subset of particles to approximate the computation equal to that of regular SVGD. We now formulate our kernels in a manner so that these subparticles provide more information. Assume we have $y_1 \dots y_m$ points, which we will denote as *induced points*. Define $k_y(x, x') = \frac{1}{m}\sum_{j=1}^m k(x, y_j)k(x', y_j)$ where $k(x, x')$ is any positive definite kernel. We denote this formulation as *induced kernels*. For positive definite kernels k_i, it holds that the product $k_1^{a_1} \dots k_n^{a_n}$ is also positive definite if $a_1 \dots a_n \in \mathbb{N}$, as well as that $\sum_{i=1}^n b_i k_i$ is positive definite if $b_i \geq 0$. Therefore, we note that the kernel $k_y(x, x')$ is still positive definite.

[1]This is the simplified estimator in the original paper, which the authors recommend using as compared to the full estimator which might have high variance arising from multi-splitting.

Algorithm 4 Random Induced Points SVGD

```
Input: A target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$
Input: A positive definite kernel $k(x, x')$
Output: A set of particles $\{x_i\}_{i=1}^n$ that approximates the target distribution
  for $t = 1 : maxIter$ do
    Let $\{y_j^t\}_{j=1}^m$ be randomly selected $m$ particles from $\{x_i^t\}_{i=1}^n$
    $\tilde{\psi}(x) = \frac{1}{m}\sum_{j=1}^{m}\left[\frac{1}{n}\sum_{i=1}^{n}\left[k(x_i^t, y_j^t)\nabla_{x_i^t}\log p(x_i^t) + \nabla_{x_i^t}k(x_i^t, y_j^t)\right]k(x, y_j^t)\right]$
    $x_i^{t+1} \leftarrow x_i^t + \epsilon^t\tilde{\psi}(x_i^t)$        where $\epsilon^t$ is the step size at the $t$-th iteration
  end for
```

Algorithm 5 Adversarial Induced Points SVGD

```
Input: A target distribution with density function $p(x)$ and a set of initial particles $\{x_i^0\}_{i=1}^n$
Input: A positive definite kernel $k(x, x')$
Output: A set of particles $\{x_i^0\}_{i=1}^n$ that approximates the target distribution
  Initialize $\{y_j^0\}_{j=1}^m$ as randomly selected $m$ particles from $\{x_i^0\}_{i=1}^n$
  for $t = 1 : maxIter$ do
    $\tilde{\psi}(x) = \frac{1}{m}\sum_{j=1}^{m}\left[\frac{1}{n}\sum_{i=1}^{n}\left[k(x_i^t, y_j^t)\nabla_{x_i^t}\log p(x_i^t) + \nabla_{x_i^t}k(x_i^t, y_j^t)\right]k(x, y_j^t)\right]$
    $x_i^{t+1} \leftarrow x_i^t + \epsilon_x^t\tilde{\psi}(x_i^t)$
    for $\ell = 1 : \mathcal{L}$ do
      $y_j^{t,l+1} \leftarrow y_j^{t,l} + \epsilon_y^{t,l}\frac{1}{n^2}\sum_{x,x'}\nabla_{y_j^{t,1}}\mathcal{A}_p^x\mathcal{A}_p^{x'}k_y(x, x')$
      $h^{t,l+1} \leftarrow h^{t,l} + \epsilon_h^{t,l}\frac{1}{n^2}\sum_{x,x'}\nabla_{h^{t,1}}\mathcal{A}_p^x\mathcal{A}_p^{x'}k_y(x, x')$
    end for
    $y^{t+1,1} \leftarrow y^{t,\mathcal{L}+1}, h^{t+1,1} \leftarrow h^{t,\mathcal{L}+1}$
  end for
  where $\epsilon$ indicate step size for each variable at each iteration
```

Lemma 3.1. *Let $k_y(x, x') = \frac{1}{m}\sum_{j=1}^{m}[k(x, y_j)k(x', y_j)]$. Then SVGD update follows the following form :*

$$\phi(x')^* = \frac{1}{m}\sum_{j=1}^{m}\left[\frac{1}{n}\sum_{i=1}^{n}\left[\mathcal{A}_p k(x_i, y_j)\right]k(x', y_j)\right] \tag{13}$$

Proof. See appendix section B. □

With the new formulation of induced kernel, we can then define a modified version of SVGD that utilizes this kernel. Algorithm 4 discusses the new method to perform SVGD. For each iteration of regular SVGD, we select m induced points and use equation 13 to update the particles. Note that the definition requires selection of induced points, whose choice is essential in the formulation. Here, we follow the idea of random subset and again simply pick m points from the particles at each iteration and make them the induced points. The benefit of picking induced points in such a manner lies on its simplicity; whenever the particles move induced points also shift accordingly. In the next section, we will discuss methods to improve our selection of induced points.

3.4 Adversarial Induced Points

Equation (10) states that the decrease in KL divergence equals the KSD. That means that the higher KSD is, greater decrease in KL divergence we can gain from updating the particles. Hence, we can imagine the algorithm in an adversarial manner: 1. We fix the particles and change the environment so that KSD is maximized. 2. We fix the environment and update the particles as in the original algorithm. In the case of induced points method, the environment that we can change is the induced points and the parameters of the kernel. We can attempt to place our induced points so that the KSD of the particles is maximized.

$$\Delta y = \frac{1}{n^2} \sum_{x,x'} \nabla_y \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x') \tag{14}$$

Another parameter that we can modify to maximize KSD is the parameters of the kernel. In the case of a RBF kernel, we can tune the bandwidth, h, to increase the KSD value. The update of h takes the following form :

$$\Delta h = \frac{1}{n^2} \sum_{x,x'} \nabla_h \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x') \tag{15}$$

Algorithm 5 shows the general scheme of this update. Our explorations show updating both induced points and kernel bandwidth sequentially can substantially increase the KSD. The specific derivations for y and h updates in the case of RBF kernels as well as the relative performance of each update can be found in the appendix.

Subset update

The adversarial update at its base state is costly, since for updating any induced point we have to consider all pairs of particles. Hence, we can select a subset of particles to evaluate the update for induced points; it is performing stochastic gradient descent rather than full batch gradient descent. Depending on implementation and selection of size of subset, this significantly reduces the runtime complexity of the algorithm.

Regularization

We observed in our exploration that the induced points would often converge to same points if we keep updating according to our rule. The points likely found a local maximum that maximizes KSD. This is undesired for several reasons. First, such convergence of the particles lead to minor updates of the actual particles, hence leading to slow updates of the particles. Secondly, such convergence then loses the merit of using various induced points, for we can achieve the same effect using fewer induced points. Therefore, we introduce a regularization term that encourages the induced points to stay far away from each other.

We let y update be the following:

$$\Delta y = \frac{1}{n^2} \sum_{x,x'} \nabla_y \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x') - \alpha \frac{1}{m} \sum_{j=1}^{m} \nabla_y k(y, y_j) \tag{16}$$

where α is a regularization parameter. This formulation discourages the induced points from converging. One may note that the regularization resembles the second term of the original SVGD update, except that the gradient is with respect to the point of interest not the other points, hence explaining the difference in sign.

U-Statistics

U-statistics is an important concept in estimation theory and produces a minimum-variance unbiased estimator (Hoeffding [13], Huskova and Janssen [15]). Liu and Lee [17] have shown that utilizing U-statistics to estimate KSD improved the test performance compared to V-statistics, which is the regular average over all pairs. Hence, we utilize the idea by only focusing on pairs of (x, x') such that $x \neq x'$ to get the gradients. Note that $\mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x')$ can be understood as a mapping from a pair (x, x') to a single value. U-statistics for formulation of adversarial updates can be formulated as the following :

$$\Delta_y = \frac{1}{n(n-1)} \sum_{x \neq x'} \nabla_y \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x') \tag{17}$$

$$\Delta_h = \frac{1}{n(n-1)} \sum_{x \neq x'} \nabla_h \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x') \tag{18}$$

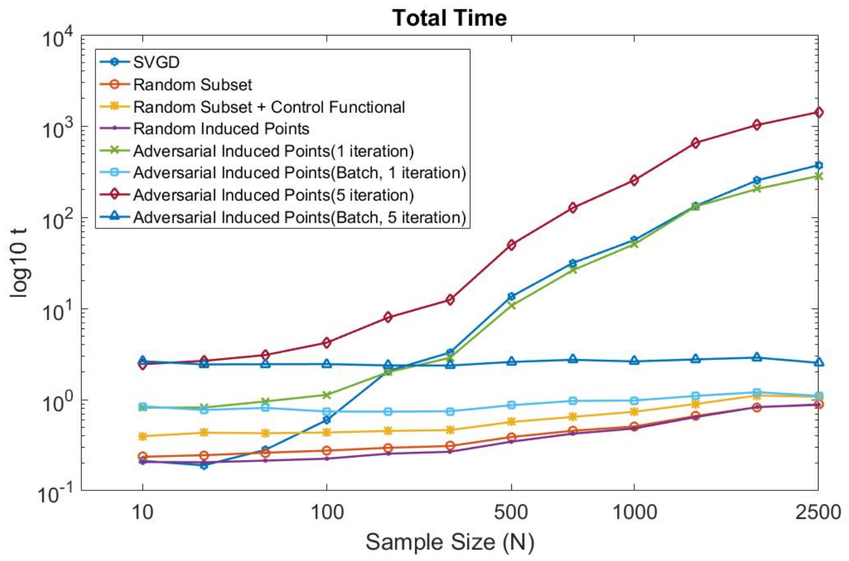


Figure 1: Comparing computation time of algorithms running 1000 iterations. The number of particles are varied, and we fix the number of subparticles or induced points to 5.

Note that in case of updates for induced points, one may combine U-statistics with the regularization. However, our exploration, which is presented in appendix, shows that even though both techniques help with the performance of adversarial update, combining them do not necessarily bring any significant benefit. When utilized separately, we observed greater performance with U-statistics. In our experiments, we try utilizing both methods. Future work may be done on improving on the regularization formulation and parameter selection.

3.5 Comparison of proposed methods

Figure 1 compares the runtime of different algorithms for 1,000 iterations. We vary the size of particles from 10 to 2,500 with the number of subparticles and induced points fixed to be 5. We observe that random subset, random subset with control functional, and induced points method all take significantly less time than regular SVGD. Note that y-axis is in logarithmic scale to show the runtime of adversarial induced points and SVGD which take several orders of magnitude longer time compared to other algorithms. We also observe that taking a batch for the update greatly reduces the computation time for adversarial updates. Note that here we only consider the respective computation speed of each method and do not explore the relative accuracy performance of each method.

Table 1 compares the complexity of different algorithms. The complexity of SVGD is $O(n^2)$ as we have to compute $\mathcal{A}_p^x k(x, x')$ between all pairs of particles. For the random subset method, rather than summing over all n particles to update a particle we sum over random subset of m subparticles, reducing the complexity to $O(nm)$. If we perform control functional, we have to compute kernel matrix K_0 where $(K_0)_{i,j} = k_0(x_i, x_j)$, k_0 being the kernel defined in equation 11. Such adds $O(m^2)$ to the complexity, but we also need to compute the inverse of this matrix, which, in naive estimation, requires $O(m^3)$. Induced points method first requires adding up $\mathcal{A}_p k(x_i, y_j)$ for each induced point, which requires $O(nm)$. Then we also need to multiply this value by $k(x, y_j)$ for all induced points to update a particle, which takes additional $O(nm)$, hence making the total complexity $O(nm)$. Lastly, for adversarial updates, for each y_j we have to compute $\nabla_{y_j}\left[(\mathcal{A}_p^x \phi_{y_j}(x))(\mathcal{A}_p^{x'} \phi_{y_j}(x'))\right]$ across all pairs of (x, x'). Hence, we see that we require $O(n^2 m)$. However, if we rather choose a subset of size m of the particles to compute the adversarial update, our runtime can be reduced to $O(m^3)$. In such a manner, our total complexity can be denoted as $O(nm + cm^3)$ where c denotes the number of adversarial updates per iteration of regular SVGD update. Note that even though theoretically random subset with control functional and adversarial induced points have same complexity, the multiplicative factor is much greater for adversarial induced points.

	SVGD	Random Subset	Random Subset (Control Functional)	Induced Points	Adversarial Induced Points
Complexity	$O(n^2)$	$O(nm)$	$O(nm+m^3)$	$O(nm)$	$O(nm+cm^3)^*$

Table 1: Complexity of different algorithms. n denotes the number of particles and m denotes the number of subparticles or induced points. * Note that for adversarial induced points we are assuming we take subset of m particles for each adversarial update, and that we make c adversarial updates for each regular SVGD update.

4 Related Works

We are observing increasing usage of Stein operator and Stein's identity in tackling various fields of problems. Pu et al. [24] explores using Stein operator to train variational autoencoders, which does not require any parametric assumption about the encoder distribution. Liu and Lee [18] computes importance weights for the samples by a convex quadratic optimization that seeks to minimize the kernelized Stein discrepancy. Han and Liu [11] uses SVGD for importance sampling; the algorithm is used to decrease the KL-divergence between importance proposal and the target distribution, allowing a standard Importance sampling framework without structural limitations or assumptions for the importance proposal. Liu [16] provides theoretical properties of SVGD including its convergence rate and asymptotic behavior.

In the original SVGD algorithm, one of the main computations is computing values of $k(x, x')$ for all pairs of x, x'. Kernel machines that largely depend on these matrices, called Gram matrix, often face computations that are too massive for efficient computation. Hence, many attempts have been made to address this problem. Williams and Seeger [29] uses Nyström method to approximate the kernel matrix as $K_{n,m}K_{m,m}^{-1}K_{m,n}$ where m is random subset of data samples. Rahimi and Recht [25] applies Fourier transform to kernels to obtain low-dimensional features where one can apply fast linear methods. Our work uses part of the gram matrix (random subset methods) or the full matrix between induced points and current particles (induced points methods); hence, further exploration may be possible to estimate these matrices to speed up the computation. This task of considering all pairs of n particles also belong to a problem set of N-body problems (Gray and Moore [9]), for which kd-tree traversal algorithms have been proposed for efficient computation. Since the initial formulation of the task, various other algorithms have been proposed, and may be considered for future work.

5 Experiments

To evaluate the performance of our models, we compare the performance on toy datasets as well as real world problems. In section 5.1 we compare models in a controlled setting of two-dimensional Gaussian. In section 5.2 we evaluate the performance of the model in Bayesian logistic regression using real world datasets. Lastly, in section 5.3 we set up Bayesian neural network and observe the results. For all the experiments, we use ADAGRAD(Duchi et al. [6]) to set our learning rate for the gradients.

5.1 2D Gaussian distribution

We test the models on toy example of 2-dimensional Gaussian distributions. We generate a random Gaussian distribution and fit the distribution using 40 particles. We approximate the mean of the true distribution and variance using maximum likelihood and compute root mean squared error. The result is shown in Figure 2. We observe that random subset and random induced points take less time than the base algorithm, while random subset with control functional requires a little more time and adversarial induced points taking substantial amount of more time. In terms of performance, we observe similar performance of random subset and random subset with control functional, tough we do see that in this toy example, control functional actually hurts the performance. Both subset methods seem to perform poorly in estimating the mean; this is understandable as the nature of the random subset makes the algorithm rather unstable. In terms of covariance, we see similar performance as the baseline SVGD. Induced points seem to underperform than yet closely match the performance of

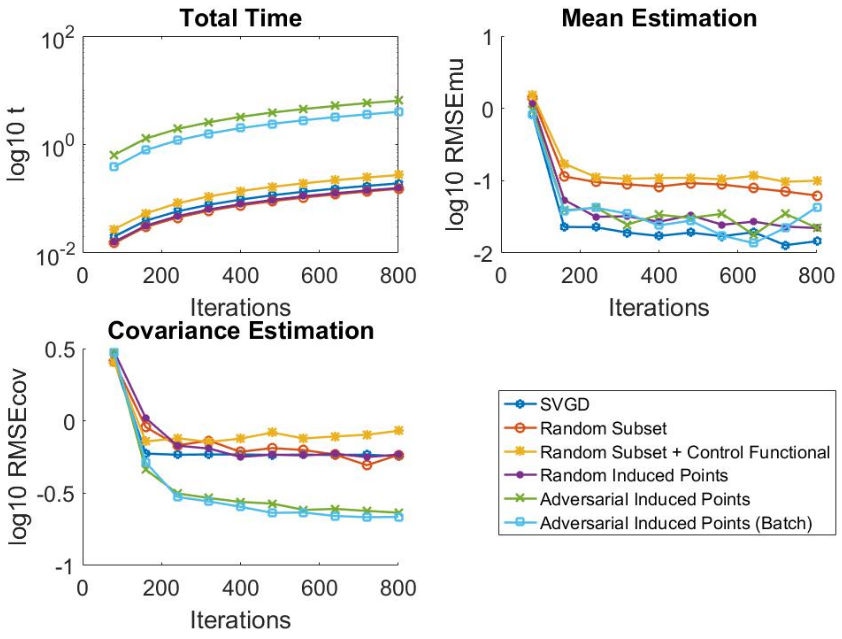


Figure 2: Comparison of main methods in estimating Gaussian distribution. We use 50 particles with 10 subparticles/induced points, and for adversarial induced points we do 10 updates per regular iteration of SVGD. We take average of 20 trials.

base SVGD but is faster. Adversarial induced points, while it requires much longer time, slightly perform poorer in mean estimation again due to random nature yet greatly outperform all models in estimation of the covariance. We also try using a batchsize of 10 for adversarial updates, which takes less time than full adversarial updates but actually performs better than full-batch updates.

5.2 Bayesian Logistic Regression

We use Bayesian logistic regression for binary classification in this experiment. The model puts a Gaussian prior on regression weights w, variance of which has gamma prior. That is, $p(w_k|\alpha) = \mathcal{N}(w_k; 0, \alpha^{-1})$ and $p(\alpha) = Gamma(\alpha; a_0, b_0)$. We then make our prediction in the following manner : $p(c_t = 1|x_t, w) = \frac{1}{1+\exp(-w^T x_t)}$. We follow the setup as in Gershman et al. [7] and let $a_0 = 1$ and $b_0 = 0.01$. We use the Covertype dataset[2] which has 581,102 datapoints with 54 attributes and 2 class labels. Given the size of the dataset, we apply stochastic gradient descent with fixed batchsize of 100 for all the tests. We randomly split the data into 80% training and 20% testing. We use different methods of SVGD to estimate the posterior distribution of weights and alpha : $p([w, \alpha]|\mathcal{D})$. To evaluate our models, we use our particles as weights to compute the accuracy as well as log-likelihood of test datapoints then take average across the particles.

Figure 3 shows the performance of subset methods compared with SVGD. We use 200 particles and 40 subparticles. We let run the algorithm for 4,000 iterations and take an average of 10 trials. We observe, as expected, random subset method take much less computation time as compared to regular SVGD. Surprisingly we see that even regular random subset method outpeforms SVGD in accuracy and log-likelihood starting from about 1000 iterations. We believe this is due to random nature of subset selection that allows more random exploration of the particles. Especially because we are performing stochastic gradient descent, deterministic shifting of the particles are not necessarily the optimal for the whole dataset. Hence, one may consider choosing random subset acting as providing random noise which allows the model to explore more options. We also see in this example how control functional helps the performance of the model. It not only reaches good results much faster

[2]https://archive.ics.uci.edu/ml/datasets/covertype

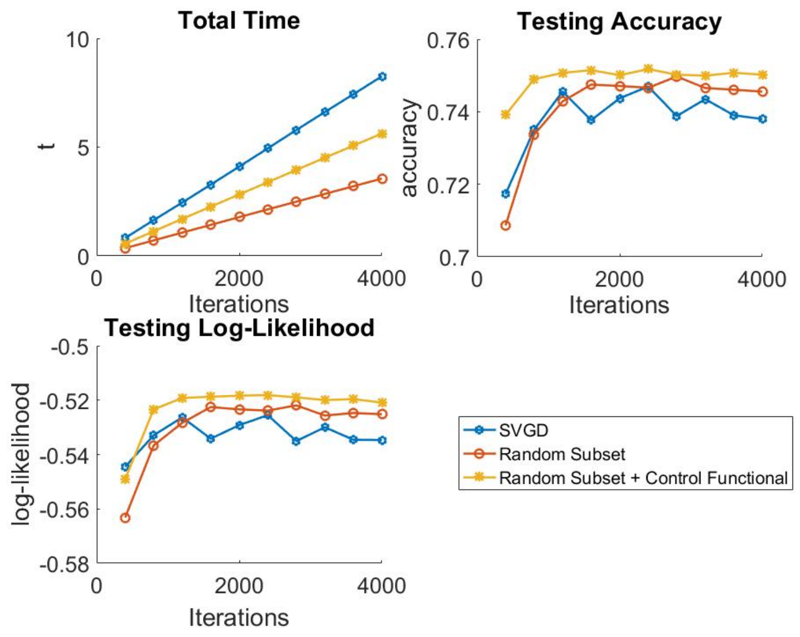


Figure 3: Comparison of random subset methods. We use 200 particles and 40 subparticles.

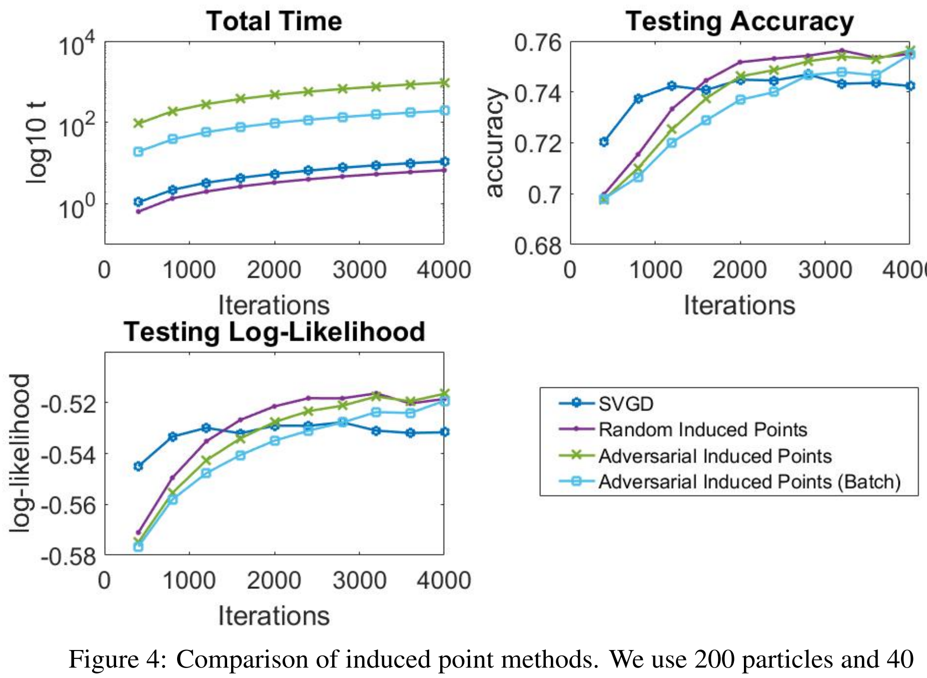


Figure 4: Comparison of induced point methods. We use 200 particles and 40 induced points.

than other algorithms, but also consistently show best performance compared with regular SVGD and random subset without control functional. Computation time-wise, it requires more time than regular random subset method due to computation of control functional, but still requires less time than regular SVGD.

Figure 4 shows the comparison between SVGD and induced points methods. Note that the plot of computation time is in logarithm scale due to adversarial induced points' computation time which is higher than other methods by an order of magnitude. We observe that random induced points are first

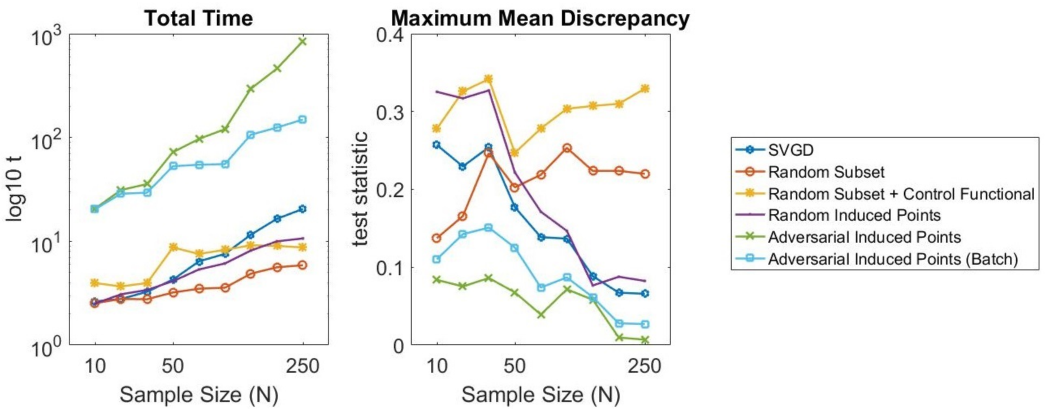


Figure 5: Maximum Mean Discrepancy on approximating posterior distribution of Bayesian Logistic Regression on Breast Cancer dataset.

slower in reaching good results, but surpasses baseline performance at around 2,000 iterations. It takes about the same time as the random subset method, though the plot is not so explicit in showing the difference as in figure 3 due to log-scaled y-axis. Adversarial Induced points, on the other hand, is not only significantly worse in runtime, but seem to actually hurt performance of random induced points in this experiment.

Maximum Mean Discrepancy

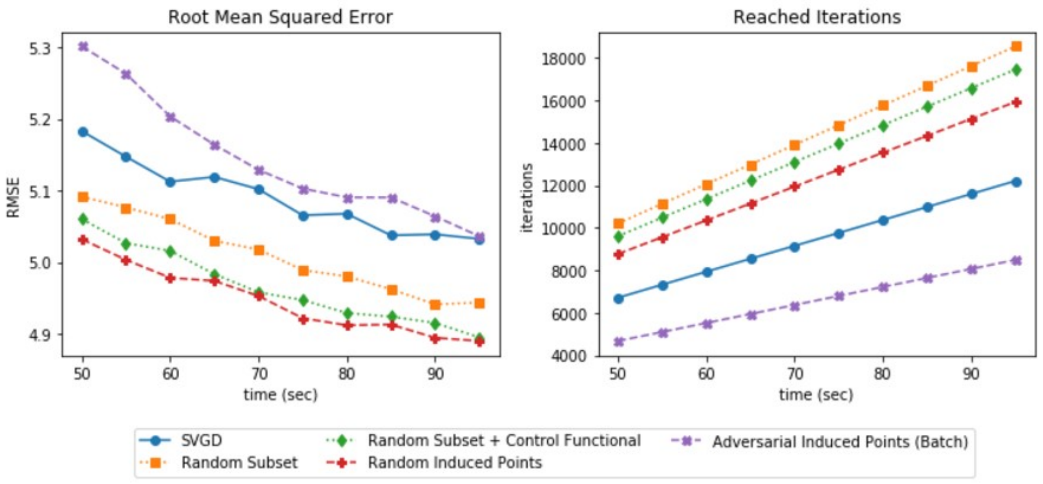
There are various methods to determine how well a set of points capture the characteristic of a probability distribution. Some basic methods are, as mentioned before, simply computing the overall mean and covariance. However, this method is rahter naive and problematic in describing distributions that are multi-modal. Hence, we use the method of Maximum Mean Discrepancy to compute how likely two sets of points come from the same distribution (Gretton et al. [10]). It measures the largest difference between two sets of points in expectation over functions in the unit ball of RKHS. It has been shown to outperform the classical two-sample t-test in various cases. Refer to the appendix for more information.

NUTS Sampler

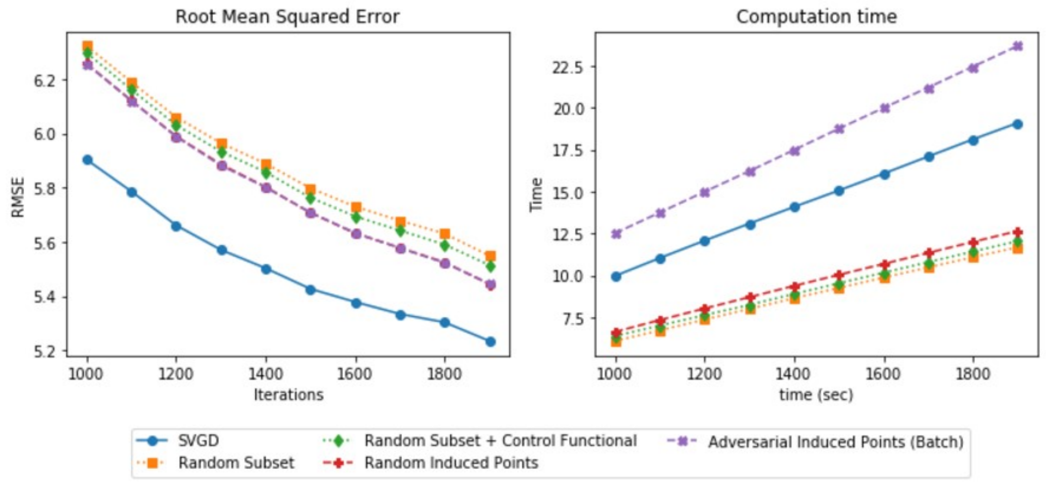
We use No-U-Turn Sampler (Hoffman and Gelman [14]) to sample points that approximate the posterior distribution. No-U-Turn Sampler is an improved technique of Markov chain Monte Carlo (MCMC) algorithm which uses first order gradient information to avoid random walk behavior and sensitivity to correlated parameters. It allows efficient sampling of points from the distribution of interest.

We evaluate the different algorithms by how well they approximate the posterior distribution. We provide the same setting as stated before. This time, we use NUTS to generate 10,000 samples after burn-in period of 10,000 steps. We compute MMD between these points and the particles after 3,000 iterations of each algorithm. We vary the number of particles from $[10, 20, 30, 50, 80, 100, 150, 200, 250]$, and the corresponding number of subparticle/induced points as $[5, 10, 10, 20, 20, 20, 35, 40, 50]$. For adversarial induced points, we perform adversarial updates 10 times per regular particle update. We show average of 10 trials.

Figure 5 shows the MMD statistics between the posterior distribution approximated by samplesof NUTS and particles of our different algorithms on the dataset of breast cancer. In terms of computation time, we observe what we expect. Random subset algorithm takes smallest amount of time, with random induced point also taking less time than regular SVGD. Random subset with control functional takes similar or slightly more time than SVGD, and induced points with adversarial updates take much longer. In terms of maximum discrepancy, we actually observe great results of induced points method. Regular induced points improve in its performance as we have more particles, while, interestingly, random subset method goes in reverse direction. This might be due to the fact that in our configuration the ratio between number of subparticles and particles decrease from $1/2$ to $1/5$. We also observe that adversarial induced points perform much better than regular SVGD across all particle numbers.



(a) We fix training time to 100 seconds and compare of different methods in Bayesian Neural Network. We take average of 10 trials. Left, root mean squared error of each model across training time. Right, training iterations reached while training.



(b) We fix training to 2,000 iterations and compare of different methods in Bayesian Neural Network. We take average of 10 trials. Left, root mean squared error of each model across iterations. Right, training time required while training.

Figure 6: Bayesian neural network test results on Concrete dataset.

Taking a batch of particles for updates rather than using all the particles indeed hurts the performance but performs faster than regular adversarial update by an order of magnitude for the case when $N = 250$. Hence, we do observe that this algorithm outperforms SVGD in its estimation of posterior distribution. More trials on other datasets are shown in the appendix. In most cases, we observe our adversarial induced points algorithm outperforming or performing at least as well as regular SVGD.

5.3 Bayesian Neural Network

Lastly we test our algorithms on the setting of Bayesian Neural Network. Bayesian Neural Network uses the framework of neural networks in a Bayesian setting (Titterington [27], Bishop [2]). Such a formulation has many benefits. Neural network provides the conventional training scheme of Bayesian inference and naturally handles the regularization, while Bayesian approach helps avoiding the overfitting problem of neural networks. We experiment using same settings as in Hernandez-Lobato and Adams [12] with parameter modification in Liu and Wang [19]. We have prediction y as Gaussian noise added to the output of neural network $y_n = f(x_n; \mathcal{W}) + \epsilon_n$ where $\epsilon_n \sim \mathcal{N}(0, \gamma^{-1})$. This gives the likelihood as normal with γ as inverse covariance : $p(y|\mathcal{W}, X, \gamma) = \prod_{i=1}^{N} \mathcal{N}(y_n|f(x_n; \mathcal{W}), \gamma^{-1})$. We define prior distribution of weights as $p(\mathcal{W}|\lambda) = \prod_{l=1}^{L} \prod_{i=1}^{V_l} \prod_{j=1}^{V_{l-1}+1} \mathcal{N}(w_{i,j,l}|0, \lambda^{-1})$ where $\lambda \sim Gamma(1, 0.1)$,$L$ is the number of layers, and V_l is number of hidden units in layer l

We use neural network with one hidden layer and 50 hidden units. Dataset was divided into 90% of training data and 10% of testing data. We use the particles to predict the results on test dataset and report the root mean squared error between the actual values and predicted values. We used 20 particles and 10 subparticles/induced points for our experiments, and we take average of 10 trials.

Figure 6a shows performance and total number of reached iterations of different models given same computation time, computed on Concrete dataset[3]. We observe that given the same time, subset method and induced points methods, with the exception of adversarial model, outperforms the original method. We also observe how the algorithms are performed for far more iterations than the original svgd algorithm due to the faster runtime. Due to high computation cost of adversarial updates, we see that adversarial induced points method reach far fewer iterations than other methods, which also explains the relative poor performance of the model in a limited time setting.

On the other hand, Figure 6b shows the performance per iteration, and we observe that the original SVGD performs better than other algorithms if they are run for the same number of iterations. Even though we need further studies to state for certain, we can state that part of performance boost of our algorithms come from its shorter runtime and hence can be run for more iterations given same resources. Adversarial updates do not show much performance boost in this example, as we do only one adversarial update using the subset of particles.

For more results, refer to the appendix.

6 Conclusion

We have investigated two major ways to reduce the complexity of the SVGD algorithm from quadratic to linear time with respect to the number of particles. First is taking a random subset of the particles to approximate the gradient update, to which applying the control functional variation reduction technique often improves the performance. Second method is using a separate set of points to approximate kernel distance between particles; we also introduce adversarial updates leading to better performance of the model. We demonstrate that despite their linear complexity, these methods often outperform the original algorithm.

There are many directions future work can take:

- Use a probability distribution to generate random points. We would learn the parameter of the distribution along with particle updates. accordingly.
- As discussed in Section 4, use random features to approximate the kernel distance between particles.
- Gain more theoretical understanding behind these algorithms.
- Conduct more experiments to find the optimal choice for number of subparticles and induced points in terms of number of particles.
- Improve the computation speed of adversarial updates.
- Conduct more experiments with much more particles ($> 1,000$).
- Parallelize the algorithm for efficient computation.

[3]https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength

References

[1] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan, `arxiv:1701.07875 [stat.ml]`. 2017.

[2] C. Bishop. Bayesian methods for neural networks. *Neural Computing Research Group*, 1995.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. 2009. ISBN 9780387310732.

[4] Y. Chen, M. Welling, and A. Smola. Super-samples from kernel herding. In *Conference on Uncertainty in Artificial Intelligence (UAI) 26*. 2010.

[5] H. Daume III. From zero to reproducing kernel hilbert spaces in twelve pages or less. 2004. URL `http://www.umiacs.umd.edu/~hal/docs/daume04rkhs.pdf`.

[6] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. volume 12, pages 2121–2159. 2011.

[7] S. Gershman, M. Hoffman, and D. Blei. Nonparametric variational inference. In *International Conference on Machine Learning 29*. 2012.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, X. Bing, W.-F. David, O. Sherjil, C. Aaron, and B. Yoshua. Generative adversarial networks. In *Advances in Neural Information Processing Systems 27*. 2014.

[9] A. Gray and A. Moore. N-body problems in statistical learning. In *Advances in Neural Information Processing Systems 13*. 2001.

[10] A. Gretton, K. Borgwardt, M. Rasch, B. Sch ölkopf, and A. Smola. A kernel method for the two-sample problem. In *Advances in Neural Information Processing Systems 20*. 2008.

[11] J. Han and Q. Liu. Stein variational adapative importance sampling, `arxiv:1704.05201 [stat.ml]`. 2017.

[12] J. Hernandez-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning 32*. 2015.

[13] W. Hoeffding. A class of statistics with asymptotically normal distribution. *Ann. Math. Statist.*, 19(3):293–325, 1948.

[14] M. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. volume 15, pages 1351–1381. 2014.

[15] M. Huskova and P. Janssen. Consistency of the generalized bootstrap for degenerate u-statistics. *Ann. Statist.*, 21(4):1811–1823, 1993.

[16] Q. Liu. Stein variational gradient descent as gradient flow, `arxiv:1704.07520 [stat.ml]`. 2017.

[17] Q. Liu and J. Lee. A kernelized stein discrepancy for goodness-of-fit tests. In *International Conference on Machine Learning 33*. 2016.

[18] Q. Liu and J. Lee. Black-box importance sampling. In *Artifical Intelligence and Statistics 20*. 2017.

[19] Q. Liu and D. Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances in Neural Information Processing Systems 29*. 2016.

[20] K. Muandet, K. Fukumizu, B. K. Sriperumbudur, and B. Schölkopf. Kernel mean embedding of distributions: A review and beyonds. *CoRR*, abs/1605.09522, 2012.

[21] K. P. Murphy. *Maching Learning A Probabilistic Perspective*. 2012. ISBN 9780262018029.

[22] C. Oates, M. Girolami, and N. Chopin. Control functionals for monte carlo integration. *Journal of the Royal Statistical Society: Series B*, 2017.

[23] T. Poggio, L. Rosasco, and C. Frogner. 9.520: Statistical learning theory and applications. 2010. URL `http://www.mit.edu/~9.520/spring10/Classes/mathcamp01.pdf`.

[24] Y. Pu, Z. Gan, R. Henao, C. Li, S. Han, and L. Carin. Stein variational autoencoder, `arxiv:1704.05155 [stat.ml]`. 2017.

[25] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems 20*. 2007.

[26] T. Salimans, I. Goodfellow, Z. Wojciech, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29*. 2016.

[27] D. Titterington. Bayesian methods for neural networks and related models. *Statistical Science*, 19:128–139, 2004.

[28] M. Welling. Herding dynamical weights to learn. In *International Conference on Interactive Mobile Communication, Technologies and Learning 4*. 2009.

[29] C. Williams and M. Seeger. Using the nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems 13*. 2000.

A Mathematical Background

Positive Definite Kernels

Definition A.1 (Positive Definite Kernel)**.** *A symmetric function, or kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{X}$, is **positive definite** if it has the following properties :*

- *Symmetric :* $k(x, x') = k(x', x)$
- *Gram matrix is positive definite. That is,*

$$\sum_{i,j=1}^{n} c_i c_j k(x_i, x_j) \geq 0 \tag{19}$$

for any $n \in \mathbb{N}$, $x_1, \ldots, x_n \in \mathcal{X}, c_1, \ldots c_n \in \mathbb{R}$

There exist various kernels that are positive definite. Ones widely used in field of machine learning include linear kernels ($k(x, x') = x^T x'$), Radial Basis Functions (RBF) kernels ($k(x, x') = \exp(-||x - x'||_2^2 / 2\sigma^2)$), Laplace kernels ($k(x, x') = \exp(-||x - x'||_1 / \sigma)$). In this work, we use RBF kernels for all our explorations.

Function Space

We start out with the basic formulation of **fields** and **vector spaces**. A field is a fundamental algrebraic structure, a set on which addition, multiplication, and identity for addition and multiplication are defined. Well known fields include the $\mathbb{Q}$, the rational numbers, as well as $\mathbb{R}$, the reals. A vector space $\mathcal{V}$ over a field $\mathcal{F}$ is a space with elements and support two operations: addition of vectors and multiplication by scalars, where scalars are from $\mathcal{F}$. Note that in some cases, this field may simply equal the vector space. Any vector field's operations should follow the associative, commutative, inverse, identity, and distributive laws. Again, the simple example of a vector space is that of $\mathbb{R}$, where each element is a element with addition and multiplication are carried out as we know them.

We say that a function $|| \cdot || : \mathcal{V} \to \mathbb{R}$ of a vector space $\mathcal{V}$ is a **norm** if $\forall u, v \in \mathcal{V}$ and $\forall a \in \mathbb{R}$ the following holds :

1. **Non-negative**: $||u|| \geq 0$
2. **Strictly positive**: $||u|| = 0 \to u = 0$
3. **Homogenuous**: $||au|| = ||a||||u||$
4. **Triangle inequality**: $||u + v|| \leq ||u|| + ||v||$

Definition A.2 (Cauchy sequence)**.** *A sequence of elements $\{x_i\}_{i=1}^{\infty}$ in a vector space $\mathcal{V}$ is a **Cauchy sequence** if $\forall \epsilon > 0, \exists N \in \mathcal{N}$ such that $\forall m, n > N$, $||v_n - v_m||_{\mathcal{V}} < \epsilon$ where $|| \cdot ||_{\mathcal{V}}$ is the defined norm of $\mathcal{V}$.*

We define a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to be a distance measure if it meets the following properties for all $x, y, z \in \mathcal{X}$.

1. **Non-negativitiy**: $d(x, y) \geq 0$
2. **Identity of indiscernibles**: $d(x, y) = 0 \leftrightarrow x = y$
3. **Symmetry**: $d(x, y) = d(y, x)$
4. **Triangle Inequality**: $d(x, z) \leq d(x, y) + d(y, z)$

With such notion of distance we can define what it means for a vector space to be complete :

Definition A.3 (Complete vector space)**.** *A vector space is complete if every Cauchy sequence is convergent. That is, there exists a point $x \in \mathcal{X}$ such that for all positive $\epsilon \in \mathbb{R}$ there exists $N \in \mathbb{N}$ such that for all $n > N$, $d(x, x_n) < \epsilon$.*

Definition A.4 (Hilbert Space)**.** ***Hilbert space*** *is a complete vector space with a defined inner product.*

In a vector space $\mathcal{V}$ over scalar field $\mathcal{F}$, the operation $\langle \cdot, \cdot \rangle_{\mathcal{V}}$ should have certain properties to be considered as a valid inner product. That is, $\forall u, v, w \in \mathcal{H}$ and $\forall a \in \mathcal{F}$,

1. **Associative**: $\langle (au), v \rangle = a\langle u, v \rangle$
2. **Commutative**: $\langle u, v \rangle = \langle v, u \rangle$
3. **Distributive**:$\langle u, (v + w) \rangle = \langle u, v \rangle + \langle u, w \rangle$

Note that a norm of a Hilbert space can easily be defined by taking inner product of the element with itself. i.e. $||u|| = \langle u, u \rangle$.

Reproducing Kernel Hilbert Space

Reproducing Kernel Hilbert Space (RKHS) is a central concept in this paper in that it allows computation of otherwise difficult optimization problems. Here we introduce its basic formulation and properties.

Definition A.5 (Functional)**.** *A* ***functional*** *$\mathcal{F}$ is a function that maps a function f to a real-value. Specifically,* ***evaluation functional*** *is a functional $\mathcal{F}_t$ that evaluates each function $f : \mathcal{X} \to \mathbb{R}$ at point $t \in \mathcal{X}$. i.e. $\mathcal{F}_t[f] = f(t)$.*

Definition A.6 (Reproducing Kernel Hilbert Space)**.** *A Hilbert space $\mathcal{H}$ is a* ***reproducing kernel Hilbert space (RKHS)*** *if the evaluation functionals are bounded. That is, $\forall x \in \mathcal{X}$, $\exists C > 0$ such that*

$$|\mathcal{F}_x[f]| = |f(x)| \leq C||f||_H \qquad \forall f \in \mathcal{H} \tag{20}$$

Each of these RKHS are equipped with the reproducing property:

Theorem A.1 (Reproducing property)**.** *For each $x \in \mathcal{X}$, there exists function $k_x \in \mathcal{H}$ such that*

$$F_x[f] = \langle k_x, f \rangle_{\mathcal{H}} = f(x) \tag{21}$$

where $< \cdot, \cdot >_{\mathcal{H}}$ is the inner product defined in $\mathcal{H}$.

Definition A.7 (Reproducing kernel)**.** *The* ***reproducing kernel*** *of $\mathcal{H}$ is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $k(x, y) = k_y(x)$.*

Then it follows from the reproducing property that $k(x, y) = k_y(x) = \langle k_x, k_y \rangle_{\mathcal{H}}$.

Theorem A.2 (Uniqueness of RKHS)**.** *For all positive definite kernel $k(\cdot, \cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ there exists a unique RKHS with k as its reproducing kernel. Conversely, a RKHS uniquely defines a positive definite reproducing kernel.*

Observation A.1 (Another definition of RKHS)**.** *Assume we have a positive definite kernel k. Consider a set of functional $S = \{k_x\}$ where $k_x(y) = k(x, y), x, y \in \mathcal{X}$. Define $\mathcal{V}$ to be set of all linear combinations of S. That, is $f \in \mathcal{V}$ can be written as $v = \sum_{i=1}^{m} \alpha_i k_{x_i}$ where $\alpha_i, m \in \mathbb{R}, x_i \in \mathcal{X}$. Let $f, g \in \mathcal{V}$ such that $f = \sum_i \alpha_i k_{x_j}, g = \sum_j \beta_j k_{x_j}$ Then we can construct a RKHS $\mathcal{H}_k$ by defining inner product as*

$$\langle f, g \rangle_{\mathcal{H}_k} = \left\langle \sum_i \alpha_i k_{x_j}, \sum_j \beta_j k_{x_j} \right\rangle_{\mathcal{X}} = \sum_i \sum_j \alpha_i \beta_j k(x_i, x_j) \tag{22}$$

where the 2nd equality follows from reproducing property.

Kernelized Stein Discrepancy

Definition A.8 (Integrally Strictly Positive). *A kernel is* ***integrally strictly positive*** *if for any function* $0 < ||g||_2^2 < \infty$

$$\int_{\mathcal{X}} g(x)k(x,x')g(x')dxdx' > 0 \tag{23}$$

Definition A.9 (Kernelized Stein Discrepancy). *The kernelized Stein discrepancy (KSD)* $\mathbb{S}(q,p)$ *between distribution* p *and* q *is defined as :*

$$\mathbb{S}(q,p) = \mathbb{E}_{x,x'\sim q}\left[\delta_{q,p}(x)^T k(x,x')\delta_{q,p}(x')\right] \tag{24}$$

where $\delta_{q,p}(x) = s_q(x) - s_p(x)$ *is the score difference between* p *and* q *and* x, x' *are i.i.d drawn from* $p(x)$*.*

Assuming that $g(x) = q(x)\left(s_q(x) - s_p(x)\right)$, if the kernel is integrally strictly positive, we have from equation 23 that the $\mathbb{S}(q,p) > 0$. We also claim that $\mathbb{S}(q,p) = 0$ iff $p = q$. To see this, if $p = q$, $\delta_{q,p}(x) = 0$ and therefore $\mathbb{S}(q,p) = 0$. On the other hand, the contrary is true since for equation 23 to be 0 $||g||_2^2$ should be 0. Therefore, we see that $\mathbb{S}(q,p)$ is a valid discrepancy measure between distributions that measures how different they are, and is 0 only when two distributions are the same.

Maximum Mean Discrepancy

Let x, y be random variables in $\mathcal{X}$, with observations $X = \{x_1, \ldots x_m\} \sim p$ and $Y = \{y_1 \ldots y_n\} \sim q$. Our question is if we can determine whether $p \neq q$ given X and Y,. Let $\mathcal{F}$ be a class of functions $\mathcal{X} \to \mathbb{R}$. Then Maximum Mean Discrepancy (MMD) can be defined as :

$$MMD\left[\mathcal{F}, p, q\right] = \sup_{f\in\mathcal{F}}\left(\mathbb{E}_{x\sim p}\left[f(x)\right] - \mathbb{E}_{y\sim q}\left[f(y)\right]\right) \tag{25}$$

or empirically,

$$MMD\left[\mathcal{F}, p, q\right] = \sup_{f\in\mathcal{F}}\left(\frac{1}{m}\sum_{i=1}^{m} f(x_i) - \frac{1}{n}\sum_{i=1}^{n} f(y_i)\right) \tag{26}$$

Note this definition is an optimization problem that largely depends on the definition of the class of functions, $\mathcal{F}$. Gretton et al. [10] bypass this problem by letting $\mathcal{F}$ be a unit ball in RKHS $\mathcal{H}$ defined for $\mathcal{X}$ with associated kernel $k(\cdot,\cdot)$. Remind ourselves that in RKHS $f(x) = \langle\phi(x), f\rangle$ where $\phi(x) = k(x,\cdot)$. Let $\mu[p] = \mathbb{E}_{x\sim p(x)}\left[\phi(x)\right]$ and note that $\mathbb{E}_p\left[f(x)\right] = \langle\mu[p], f\rangle$. Then equation 25 becomes :

$$MMD[\mathcal{F}, p, q] = \sup_{||f||_{\mathcal{X}}\leq 1}\langle\mu[p] - \mu[q], f\rangle = ||\mu[p] - \mu[q]||_{\mathcal{H}} \tag{27}$$

Since $\mu[X] = \frac{1}{m}\sum_{i=1}^{m}\phi(x_i)$ and $k(x,x') = \langle\phi(x), \phi(x')\rangle$, the empirical MMD is

$$MMD[\mathcal{F}, p, q] = \left[\frac{1}{m^2}\sum_{i,j=1}^{m} k(x_i, x_j) - \frac{2}{mn}\sum_{i,j=1}^{m,n} k(x_i, y_j) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(y_i, y)j)\right]^{1/2} \tag{28}$$

Control Functional

Assume we are interested in the value of function $f(x)$, and want to approximate it with $\hat{f}(x)$. Oates et al. [22] starts by formulating an estimator

$$f^*(x) = f(x) - \hat{f}(x) + \mu(\hat{f}(x)) \tag{29}$$

where $\mu(\hat{f}(x))$ is the expected value of $\hat{f}(x)$. We see that $E[f(x)] = E(f^*(x))$. Define

$$\hat{f}(x) = c + \psi(x) \tag{30}$$

$$\psi(x) = \nabla_x\phi(x) + \phi(x)\nabla_x \log p(x) \tag{31}$$

where $c \in \mathbb{R}$ and $\phi(x) \in C^1(\mathcal{X}, \mathbb{R}^d)$; ($C^1(\mathcal{X}, \mathbb{R}^d)$ is the space of measurable functions from $\mathcal{X}$ to $\mathbb{R}^d$ that is differentiable once.) Then, we see that $\psi \in \mathcal{H}_0$, a RKHS with k_0 defined as the following :

$$k_0(x, x') = \nabla_x \nabla_{x'} k(x, x') + u(x)\nabla_{x'} k(x, x') + u(x')\nabla_x k(x, x') + u(x)u(x')k(x, x') \quad (32)$$

where $k(x, x')$ is a positive definite kernel.

Then the control functional estimator $\mu(\hat{f}(x))$ is the following :

$$\mu(\hat{f}(x)) = \frac{\mathbf{1}^T(\mathbf{K_0} + \lambda m \mathbf{I})^{-1}}{1 + \mathbf{1}^T(\mathbf{K_0} + \lambda m \mathbf{I})^{-1}\mathbf{1}} f_0 \quad (33)$$

where $(K_0)_{i,j} = k_0(x_i, x_j)$ and $f_0 = [f(x_1) \dots f(x_n)]$.

B Proofs and Derivations

We show that the expected value of $s_p(x)f(x) + \nabla_x f(x)$ is 0 if f is in Stein class of p.

Proof of theorem 2.1

Proof.

$$\begin{aligned} s_p(x)f(x) + \nabla_x f(x) &= \nabla_x p(x)f(x) + \nabla_x f(x) = \frac{\nabla_x(p(x)f(x))}{p(x)} \\ \mathbb{E}_p\left[s_p(x)f(x) + \nabla_x f(x)\right] &= \int_x \frac{\nabla_x(p(x)f(x))}{p(x)} p(x)dx = \int_x \nabla_x(p(x)f(x))dx \\ &= 0 \qquad \text{(as } f \text{ is in Stein class of } p\text{)} \end{aligned}$$

□

Proof of lemma 3.1

We show that if $k_y(x, x') = \frac{1}{m}\sum_{j=1}^m [k(x, y_j)k(x', y_j)]$, SVGD update takes the following form :

$$\phi(x')^* = \frac{1}{m}\sum_{j=1}^m \left[\frac{1}{n}\sum_{i=1}^n [\mathcal{A}_p k(x_i, y_j)]\, k(x', y_j)\right]$$

Proof.

$$\begin{aligned} \phi(x') &= \mathcal{A}_p k_y(x, x') = \frac{1}{m}\sum_{j=1}^m \mathcal{A}_p k(x, y_j)k(x', y_j) \\ \tilde{\phi}(x') &= \frac{1}{n}\sum_{i=1}^n \frac{1}{m}\sum_{j=1}^m \mathcal{A}_p k(x_i, y_j)k(x', y_j) = \frac{1}{m}\sum_{j=1}^m \frac{1}{n}\sum_{i=1}^n \mathcal{A}_p k(x_i, y_j)k(x', y_j) \\ &= \frac{1}{m}\sum_{j=1}^m \left[\frac{1}{n}\sum_{i=1}^n [\mathcal{A}_p k(x_i, y_j)]\, k(x', y_j)\right] \end{aligned}$$

□

KSD defined with induced kernel

By definition, KSD can be defined as $\mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x')$, where $k_y(x, x')$ is our induced kernel. Since Stein operator $\mathcal{A}_p$ is a linear operator, we can make the following observation :

$$\mathcal{S}(q, p) = \mathbb{E}_{x, x' \sim q}\left[\mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x, x')\right] = \frac{1}{m}\sum_{j=1}^m \mathbb{E}_{x, x' \sim q}\left[\mathcal{A}_p^x \mathcal{A}_p^{x'} k_{y_j}(x, x')\right] \quad (34)$$

where $\mathcal{S}(q,p)$ denotes the Kernelized Stein Discrepancy between distribution p and q. We also observe the following property of Stein Operator.

$$\mathcal{A}_p^x \mathcal{A}_p^{x'} k_{y_j}(x,x') = \left(\mathcal{A}_p^x \phi_{y_j}(x)\right)^T \left(\mathcal{A}_p^{x'} \phi_{y_j}(x')\right) \tag{35}$$

where $\phi_{y_j}(x) = k(x,y_j)$.

Adversarial induced points (y) updates

We update the induced points, or $Y = \{y_j\}_{j=1}^m$, to maximized KSD. Using eq. 34, we derive a gradient of KSD with respect to h.

Due to linearity of KSD shown in equation 34, we can derive gradient of h with respect to each $\mathcal{A}_p^x \mathcal{A}_p^{x'} k_{y_j}(x,x')$ and add them up. We are interested in the following :

$$\nabla_{y_i} \mathcal{A}_p^x \mathcal{A}_p^{x'} k_{y_j}(x,x') = \mathbb{E}_{x,x'}\left[\nabla_{y_i}\left(\mathcal{A}_p^x \phi_{y_j}(x)\right)^T \left(\mathcal{A}_p^{x'} \phi_{y_j}(x')\right) + \left(\mathcal{A}_p^x \phi_{y_j}(x)\right)^T \nabla_{y_i}\left(\mathcal{A}_p^{x'} \phi_{y_j}(x')\right)\right] \tag{36}$$

We note that Equation 36 is 0 if $i \neq j$. Hence, for gradient with respect to y_i, we observe the following :

$$\begin{aligned}\nabla_{y_i} \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x,x') &= \frac{1}{m}\sum_{j=1}^m \nabla_{y_i} A_p^x A_p^{x'} k_{y_j}(x,x') \\ &= \frac{1}{m}\nabla_{y_i} A_p^x A_p^{x'} k_{y_i}(x,x')\end{aligned}$$

Assume we are interested in gradient update with respect to ith dimension of y_j.

$$\frac{\partial}{\partial y_{j_i}} \mathcal{A}_p^x \mathcal{A}_p^{x'} k_y(x,x') = \underbrace{\frac{\partial}{\partial y_{j_i}}\left(s_q(x)\phi_{y_j}(x) + \nabla_x \phi_{y_j}(x)\right)^T}_{\textbf{(a)}} \underbrace{\left(s_q(x')\phi_{y_j}(x') + \nabla_{x'}\phi_{y_j}(x')\right)}_{\textbf{(b)}}$$

$$\begin{aligned}\textbf{(a)} &= \frac{\partial}{\partial y_{j_i}}\left(s_q(x)k(x,y_j) - k(x,y_j)\frac{x-y_j}{h^2}\right) \\ &= \frac{\partial}{\partial y_{j_i}}(k(x,y_j))\left(s_q(x) - \frac{x-y_j}{h^2}\right) + (k(x,y_j))\frac{\partial}{\partial y_{j_i}}\left(s_q(x) - \frac{x-y_j}{h^2}\right)\end{aligned}$$

$$\frac{\partial}{\partial y_{j_i}}(k(x,y_j)) = \frac{\partial}{\partial y_{j_i}} \exp\left(-\frac{||x-y_j||_2^2}{2h^2}\right) = k(x,y_j)\frac{x_i - y_{j_i}}{h^2}$$

$$\frac{\partial}{\partial y_{j_i}}\left(s_q(x) - \frac{x-y_j}{h^2}\right) = \frac{1}{h^2}\mathbf{1}_i \qquad \text{where } \mathbf{1}_i \text{ is a vector of 0s with } i\text{th element} = 1$$

$$\textbf{(a)} = k(x,y_j)\frac{x_i - y_{j_i}}{h^2}\left(s_q(x) - \frac{x-y_j}{h^2}\right) + \frac{k(x,y_j)}{h^2}\mathbf{1}_i$$

$$\textbf{(b)} = k(x',y_j)\left(s_q(x') - \frac{x'-y_j}{h^2}\right)$$

In the end, we perform **(a)**^{T}**(b)** to get the following :

$$\frac{\partial}{\partial y_{j_i}}\left[(\mathcal{A}_p^x \phi_{y_j}(x))\right](\mathcal{A}_p^{x'}\phi_{y_j}(x')) =$$

$$\left[k(x,y_j)\frac{(x_i - y_{j_i})}{h^2}\left(s_q(x) - \frac{x-y_j}{h^2}\right) + \frac{k(x,y_j)}{h^2}\mathbf{1}_i\right]^T \left[k(x',y_j)\left(s_q(x') - \frac{x'-y_j}{h^2}\right)\right] \tag{37}$$

This is only first half of equation 36. But note that if we sum over all pairs of (x, x'), due to symmetry we can simply double the sum to get the total value. Lastly, we divide by n^2 to get gradient update for y_{j_i}. We perform this for all dimensions of each induced point y_j,

Adversarial bandwidth (h) updates

We can update bandwidth, or h, of RBF kernels to maximize KSD. We follow the same steps as with Y updates, but we now take gradient with h. We are interested in the following :

$$\nabla_h \mathcal{A}_p^x \mathcal{A}_p^{x'} k_{y_j}(x, x') = \mathbb{E}_{x,x'} \left[\underbrace{\nabla_h \left(\mathcal{A}_p^x \phi_{y_j}(x) \right)^T}_{\textbf{(c)}} \underbrace{\left(\mathcal{A}_p^{x'} \phi_{y_j}(x') \right)}_{\textbf{(b)}} + \left(\mathcal{A}_p^x \phi_{y_j}(x) \right)^T \nabla_h \left(\mathcal{A}_p^{x'} \phi_{y_j}(x') \right) \right] \tag{38}$$

Note that **(b)** is same as before. We take the gradient part by part.

$$\begin{aligned}
\nabla_h \left(\mathcal{A}_p^x \phi_y(x) \right) &= \nabla_h \left(s_q(x)\phi_{y_j}(x) + \nabla_x \phi_{y_j}(x) \right) = \frac{\partial}{\partial h} \left(s_q(x) k(x, y_j) + \nabla_x k(x, y_j) \right) \\
&= \frac{\partial}{\partial h} \left(s_q(x) k(x, y_j) - k(x, y_j) \frac{x - y_j}{h^2} \right) = \frac{\partial}{\partial h} \left(k(x, y_j) \left(s_q(x) - \frac{x - y_j}{h^2} \right) \right) \\
&= \frac{\partial}{\partial h} \left(k(x, y_j) \right) \left(s_q(x) - \frac{x - y_j}{h^2} \right) + k(x, y_j) \frac{\partial}{\partial h} \left(s_q(x) - \frac{x - y_j}{h^2} \right) \\
\frac{\partial}{\partial h} k(x, y_j) &= \frac{\partial}{\partial h} \exp \left(-\frac{||x - y_j'||_2^2}{2h^2} \right) = \exp \left(-\frac{||x - y_j'||_2^2}{2h^2} \right) \frac{\partial}{\partial h} \left(-\frac{||x - y_j'||_2^2}{2h^2} \right) \\
&= k(x, y_j) \left(\frac{||x - y_j'||_2^2}{h^3} \right) \\
\textbf{(c)} &= k(x, y_j) \left(\frac{||x - y_j'||_2^2}{h^3} \right) \left(s_q(x) - \frac{x - y_j}{h^2} \right) + k(x, y_j) \left(\frac{2(x - y_j)}{h^3} \right)
\end{aligned}$$

In the end, we perform $\textbf{(c)}^T\textbf{(b)}$ to get the following :

$$\begin{aligned}
&\nabla_h \left(\mathcal{A}_p^x \phi_{y_j}(x) \right)^T \left(\mathcal{A}_p^{x'} \phi_{y_j}(x') \right) = \\
&\left[k(x, y_j) \left(\frac{||x - y_j'||_2^2}{h^3} \right) \left(s_q(x) - \frac{x - y_j}{h^2} \right) + k(x, y_j) \left(\frac{2(x - y_j)}{h^3} \right) \right]^T \left[k(x', y_j) \left(s_q(x') - \frac{x' - y_j}{h^2} \right) \right]
\end{aligned} \tag{39}$$

Note that this is only first half of equation 38. But due to symmetry, we can compute equation 39 for all pairs of x, x' and multiply by 2 to get the total sum. We perform this over all y_j kernels and add all the results. We then divide the value by $m * n^2$ to get our final update Δh.

C More Results

Bayesian Logistic Regression

Figure 7 shows the comparison of the algorithms' runtime and Maximum Mean Discrepancy test statistic across various datasets, which are described in Table 2. The setup is equal to the one described in Section 5.2. We observe the adversarial induced points methods outperforming or performing comparably with the original SVGD algorithm across all datasets.

Dataset	# Datapoint	Dimension
Banana	5,300	2
Breast Cancer	263	9
Diabetis	768	8
Flare Solar	144	9
German	1,000	20
Heart	270	13

Table 2: Number of instances and dimensions of each dataset.

Bayesian Neural Network

Figures 8 and 9 demonstrate the results of running Bayesian Neural networks on the Energy[4] and Kin8nm[5] datasets. We use the same setting as before. We observe results similar to previous test in that we observe induced points and random subset methods performing better than regular SVGD in a limited time, likely due to its faster computation.

D Further Investigation

Comparison of $\mathbf{y}$ and $\mathbf{h}$ updates

We look at how updating y and h in an adversarial manner affects our performance. Fig. 10 shows increase in KSD as we update y, h, or both. We use Bayesian logistic regression setting with the Boston Housing dataset; we fix our minibatch selection and do not update the particles so that so we solely observe the effect of adversarial updates. We observe that combining y and h update indeed greatly maximize KSD as expected. In figure 11, we observe differences in performance of adversarial induced points using y and h updates.

Effect of regularization and U-statistics

We observe the effect of regularization on the performance of adversarial induced model. We test on the simple 2-d Gaussian distribution and compute the RMSE between maximum likelihood estimation of mean and variance with their true values. In figure 12, we compare the models of 1. basic adversarial updates 2. with regularization 3. using U-statistics 4. using both regularization and U-statistics. We perform 10 adversarial updates per regular particle updates. In figure (12) We observe that while the mean estimation performance does not necessarily increase, there is significant difference in the estimation of variance as the number of adverse iterations increase. Figures 13 and 14 each compare the difference between updating y and h after applying both regularization and U-statistics, and just U-statistics, respectively.

[4]https://archive.ics.uci.edu/ml/datasets/Energy+efficiency

[5]http://mldata.org/repository/data/viewslug/uci-20070111-kin8nm/1/

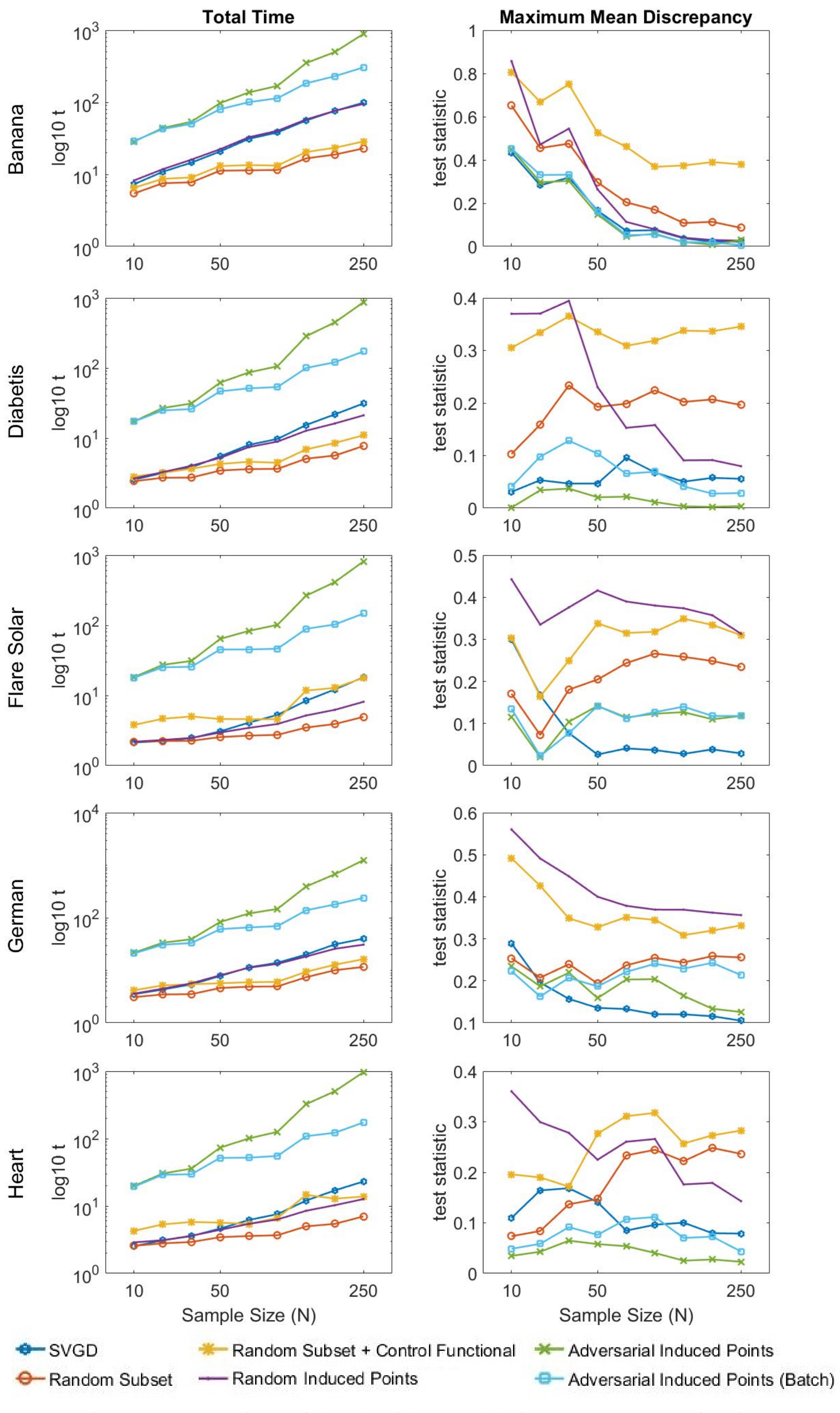
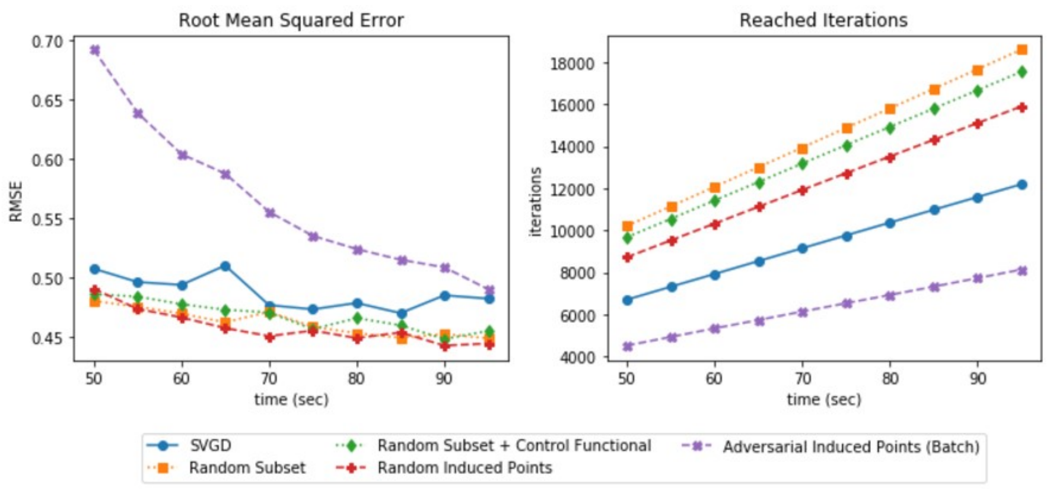


Figure 7: Comparison of the algorithms and their MMD across performing Bayesian logistic regression across various datasets.

(a) 100 second limit



(b) 2000 iteration limit

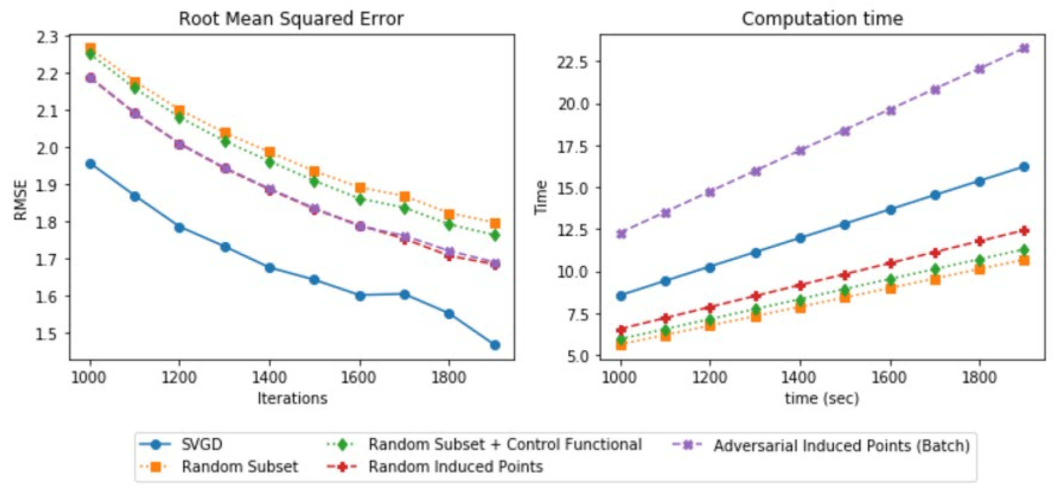
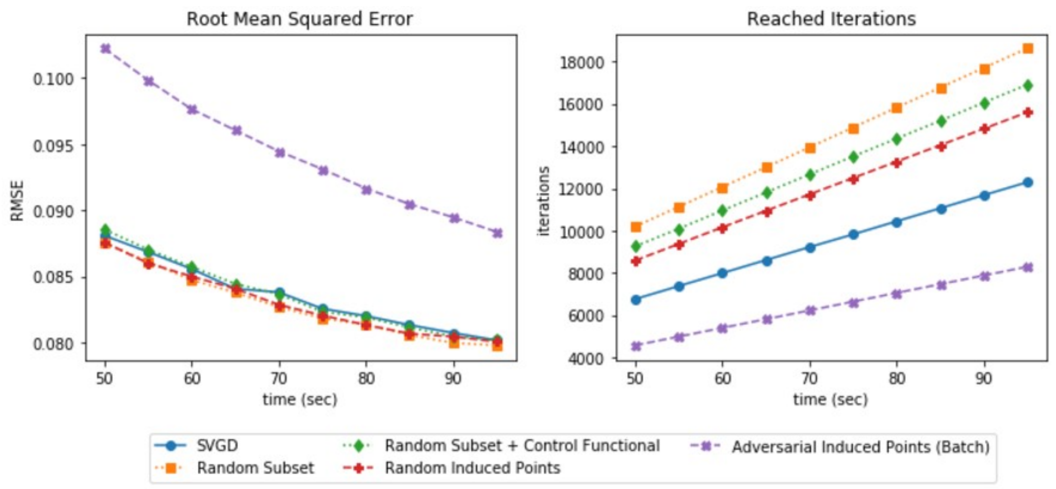


Figure 8: Example of Bayesian Neural Network (Energy dataset).

(a) 100 second limit



(b) 2000 iteration limit

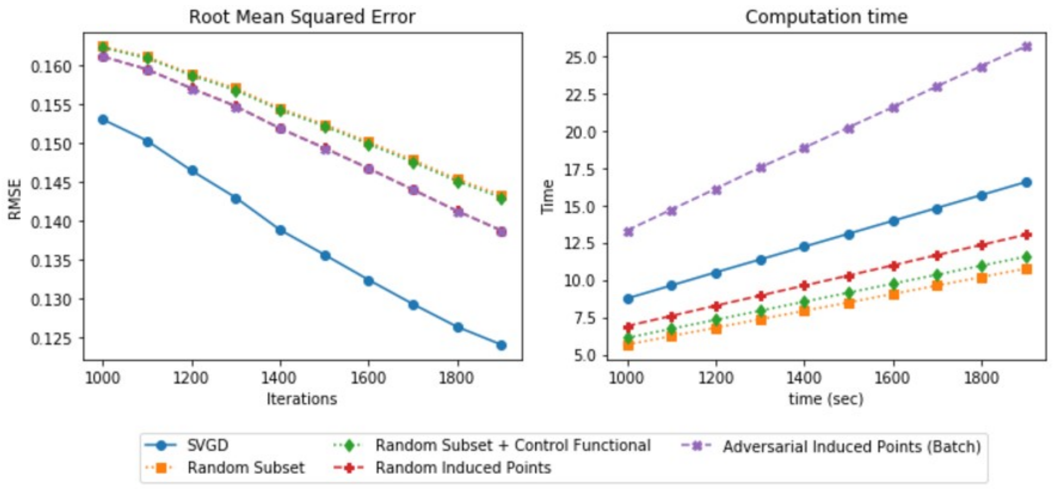


Figure 9: Example of Bayesian Neural Network (Kin8nm dataset).

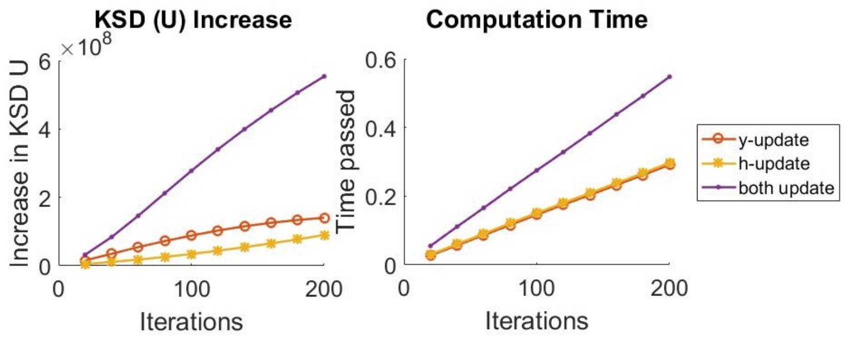


Figure 10: Comparison of effect of updating y and h both on KSD. We test using Bayeisan Logistic Regression for Boston housing dataset with fixed minibatch of size 100 and do not update the particles. We use 100 particles and 10 induced points. Average of 20 trials.

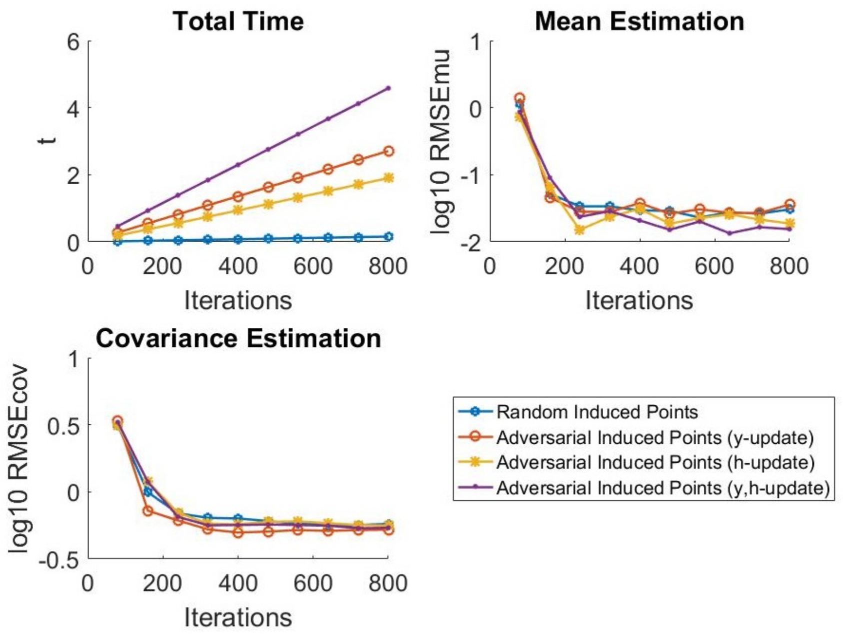


Figure 11: Comparison of performance of updating y and h in toy Gaussian. We use 100 particles and 10 induced points. 10 adversarial updates are performed after each regular particle update. Average of 20 trials.

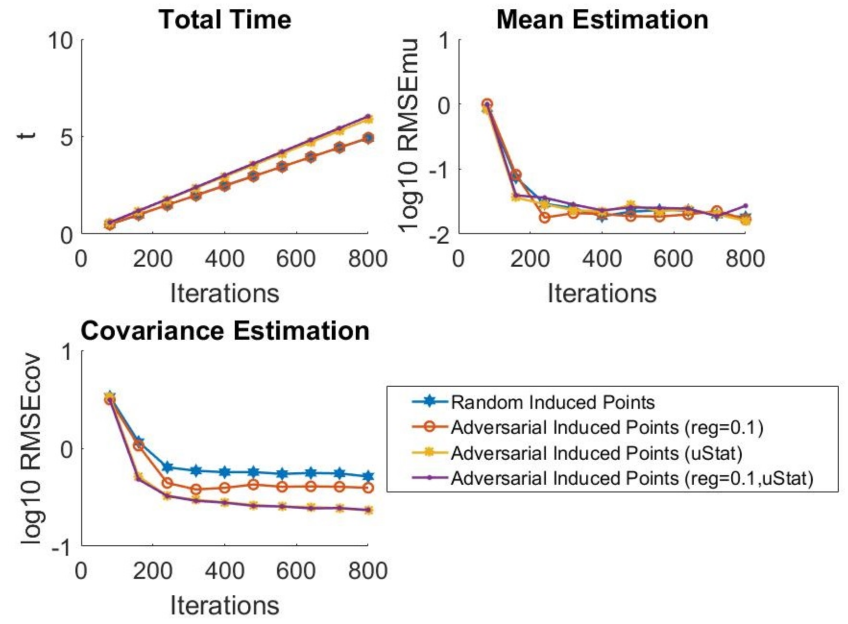


Figure 12: Comparison of effect of regularization, U-statistics, and using both methods. We used 50 particles and 10 induced points. We ran 10 adversarial updates times per regular particle update. We take average of 20 iterations. (Before updates)

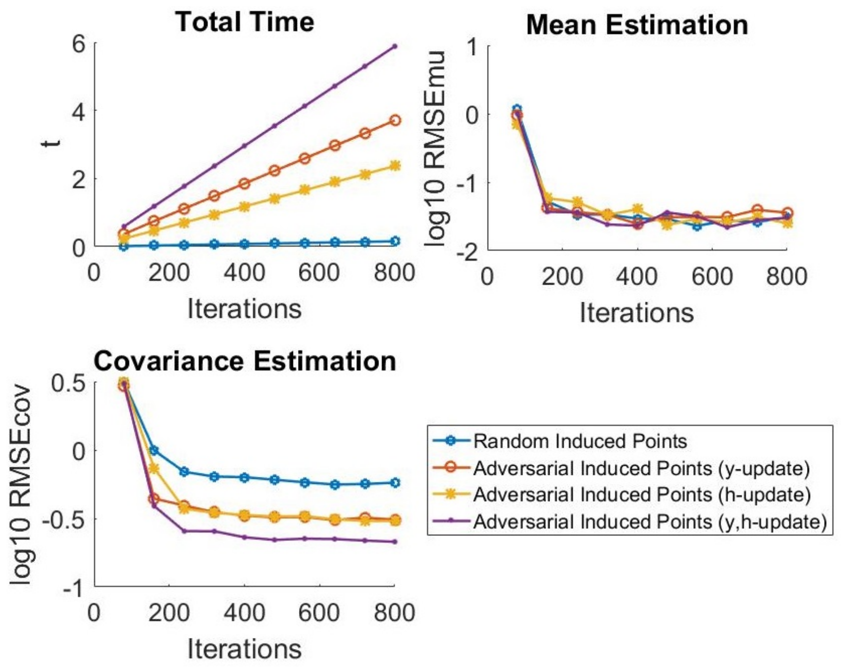


Figure 13: Comparison of performance of y,h updates after utilizing U-statistics and regularization. We used 50 particles and 10 induced points. We ran 10 adversarial updates per regular particle update. We take average of 20 iterations.

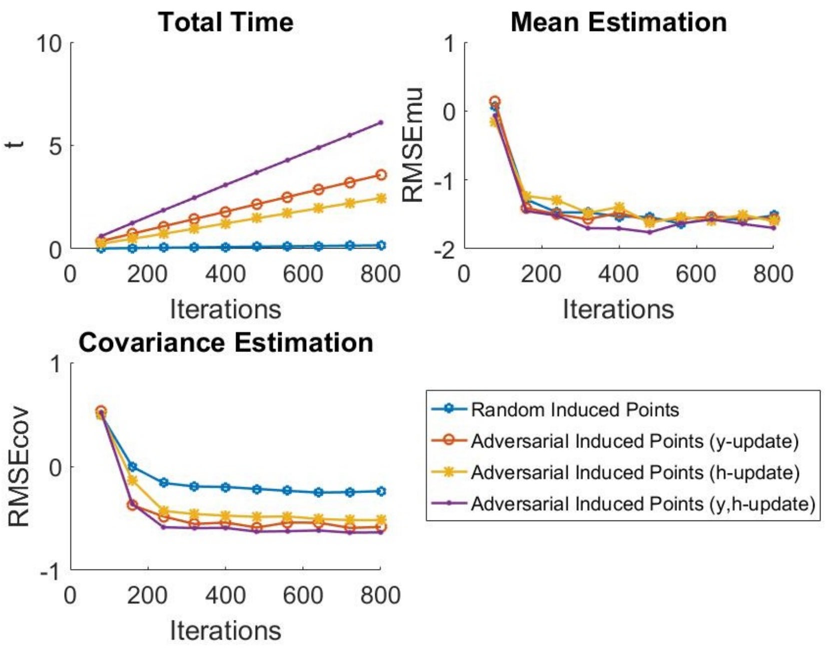


Figure 14: Comparison of performance of y,h updates after using just U-statistics. We used 50 particles and 10 induced points. We ran 10 adversarial updates per regular particle update. We take average of 20 iterations.