NEURAL STEIN VARIATIONAL GRADIENT DESCENT

Zeeshan Ashraf[1] and Naeem Turner-Bandele[1]

[1]Department of Electrical and Computer Engineering, Carnegie Mellon University

May 18, 2019

1 Introduction

A key challenge of modern statistics and machine learning is to approximate complex probability distributions. This concept is central to Bayesian Inference, which models the machine learning model parameters as probability distributions rather than point estimates. This involves approximating the intractable posterior distribution $p(paramters|data)$.

Several methods exist to approximate the intractable posterior. One of the most popular methods is variational inference. Key benefits of variational inference are its speed, ability to handle large datasets, and the power to use optimization techniques. Despite its strengths, variational inference is difficult to optimize in higher dimensions, computationally expensive and distribution dependent. Together, these factors make traditional variational inference impractical and difficult to use.

Another method to approximate the posterior is using Markov Chain Monte Carlo (MCMC) methods. MCMC methods are used to generate approximate samples from the posterior and then the distribution can be estimated from the samples. Generating MCMC samples is often slow and difficult to access if we have the sample from the correct distribution.

Recently, the authors in [3] proposed Kernelized Stein Variational Gradient Descent (KSVGD). In this method, they iteratively model a target distribution with a set of particles and update the particles to eventually converge to the posterior distribution. The converged particles can be considered as samples from the posterior distribution. We can estimate the posterior from these samples as in case of MCMC. In order to approximate complex posteriors in high dimensions, we require a large number of samples. KSVGD gets increasingly slow as the number of particles increases.

In this work, we introduce the NSVGD inference method. This method can be used to efficiently generate a large number of samples from a distribution with an intractable normalization constant. We validate the performance of our method by comparing the computational efficiency and Kolmogrov-Smirnov (KS) test values for NSVGD inference and KSVGD.

1.1 Outline

The rest of this paper is structured as follows. Section 2 begins with an overview of Bayesian Inference, Stein Variational Gradient Descent (SVGD) and Kernelized Stein Variational Gradient Descent(KSVGD). Using the insights of Section 2, Section 3 and 4 extend the SVGD method to develop the Neural SVGD (NSVGD) and Neural SVGD inference formulation. Section 5 presents empirical experiments conducted to analyze the NSVGD and NSVGD inference methods and compare it with KSVGD. Future work is discussed in Section 6.

2 Background

Let $\{D_k\}$ be a set of i.i.d. observations. We want to fit a known model to the data. Let $x \subset R^d$ be the parameter of the model and $p_0(x)$ be the prior distribution of the parameters. Since, we choose the model to fit to the data, we have the likelihood distribution of the data given the parameters i.e. $\prod_{k=1}^{N} p(D_k|x)$. Bayesian inference of parameters x involves reasoning and drawing samples from the posterior distribution of x, $p(x) = \left\{\prod_{k=1}^{N} p(D_k|x)\right\} p_0(x)/Z$. Calculating the normalization constant Z = $\int p_0(x) \prod_{k=1}^{N} p(D_k|x)dx$ is difficult since it is an intractable, high-dimensional integral. Note that moving forward we are writing $p(x|D_k)$ as $p(x)$ for convenience.

Liu and Wang, the authors of [3] suggest an alternate method, known as Stein Variational Gradient Descent (SVGD), to approximate the posterior distribution. The method takes some randomly initialized particles in the space R^d of the parameters. These particles are incrementally moved towards the posterior distribution so that the $\mathbf{KL}(q(x_t)||p(x))$ is decreased, where $q(x_t)$ represents the distribution followed by the particles at step t. The particles updated in this manner eventually mimic the posterior.

A chief benefit of this formulation is that it is independent of Z, the normalization constant of the distribution. SVGD utilizes $\nabla_x logp(x)$ to move the particles towards the posterior. Unlike classical Bayesian inference, Z is not required to calculate $\nabla_x logp(x)$ because it is constant.

Stein Variational Gradient Descent utilizes both Stein's identity and the Stein discrepancy at the core of its formulation. Both concepts are discussed in the next section.

2.1 Stein's Identity and Stein Discrepancy

Stein Variational Gradient Descent relies on the *Stein's Discrepancy* which originates from *Stein's identity* [2]. Stein's identity states that for for a smooth density $p(x)$ and function $\phi(x) \subset R^d$ that satisfy $\lim_{||x||\to\infty} p(x)\phi(x) = 0$, we have

$$\mathbb{E}_{x\sim p}[\phi(x)\nabla_x \log p(x)^T + \nabla_x \phi(x)] = 0 \tag{1}$$

where $\mathcal{A}_p\phi(x) = \phi(x)\nabla_x \log p(x)^T + \nabla_x\phi(x)$ is the Stein Operator, $\nabla_x\phi(x)$ is the Jacobian, while $\phi(x)$ and $\nabla_x \log p(x)$ are vectors. Equation (1) can be validated by expanding the expectation, employing integration by parts, and using the boundary conditions [2].

Stein's Identity, guarantees that if the Stein operator acts on a reasonable set of functions $\phi \in \mathcal{F}$ then we can generate scalar valued functions that are mean zero under a target distribution p. If (1) holds, we state that ϕ is a member of the *Stein class* of p.

Using *Stein's identity*, we can derive the Stein discrepancy. Let $q(x)$ be a different yet smooth distribution designed to approximate p, and $\phi(x)$ be a smooth vector that is a member of the Stein class q. Now, in contrast to the result of the $\mathbb{E}_{x\sim p}$, if we consider $\mathbb{E}_{x\sim q}$, since $p \neq q$, we have a nonzero equality relation:

$$\exists\phi, \ \ \mathbb{E}_{x\sim q}[\mathcal{A}_p\phi(x)] \neq 0 \tag{2}$$

This can be seen as below, since $\mathbb{E}_{x\sim q}[\mathcal{A}_q\phi(x)] = 0$:

$$\begin{aligned}\mathbb{E}_{x\sim q}[\mathcal{A}_p\phi(x)] &= \mathbb{E}_{x\sim q}[\mathcal{A}_p\phi(x)] - \mathbb{E}_{x\sim q}[\mathcal{A}_q\phi(x)] \\ &= \mathbb{E}_{x\sim q}[\phi(x)\left\{\nabla_x logp(x) - \nabla_x logq(x)\right\}^T]\end{aligned} \tag{3}$$

Since, $p \neq q$, $\nabla_x logp(x) - \nabla_x logq(x) \neq 0$ and hence the result follows. The above identity is zero only when the distributions p and q are the same. It decreases as the expected gradients of log density become similar. As a result, we can use the identity as a measure of difference between two distributions.

The quantity also depends on the function $\phi(x)$. In order to have a meaningful and descriptive difference we should have a ϕ such that $\mathbb{E}_{x\sim q}[\mathcal{A}_p\phi(x)]$ is large when the distributions are not similar. Using this reasoning we arrive at the Stein Discrepancy between two distributions. It has the similar properties as (3), but it is a scalar quantity and hence can be used as a loss function. It is given by:

$$\mathbb{S}(q,p) = \max_{\phi\in\mathcal{F}} \left[\mathbb{E}_{x\sim q}[\left\{\nabla_x logp(x) - \nabla_x logq(x)\right\}^T \phi(x)]\right]^2 \tag{4}$$

Using simple matrix manipulation we can show (4) is equivalent to:

$$\mathbb{S}(q,p) = \max_{\phi\in\mathcal{F}} \left[\mathbb{E}_{x\sim q}[\text{trace}(\mathcal{A}_p\phi(x))]\right]^2 \tag{5}$$

In order for the Stein Discrepancy to hold, the density functions p and q must be computationally tractable and the functions $\phi \in \mathcal{F}$ should be well behaved [3].

2.2 Stein Variational Gradient Descent

In traditional variational inference we assume a standard form of the approximate distribution and bring it closer to the true (posterior) distribution by reducing the KL divergence between the two distributions [1]. This is limited by how well our chosen distribution can approximate the true one. Liu and Wang propose an alternate form of variational inference.

Let x_k be the particles at step k and $q(x_k)$ be the probability density of these particles. We can think of variational inference as transformation of a probability density through a sequence of invertible mappings:

$$x_{k+1} = T(x_k) = x_k + \epsilon\phi(x_k) \tag{6}$$

where $\phi(x)$ is a smooth function that represents our perturbation direction, and ϵ is a scalar that denotes our perturbation magnitude. $q(x_{k+1})$ is part of the set $\mathcal{Q}$ consisting of distributions formed by smooth transforms. Under this transform, Liu and Wang demonstrate that the following relationship holds:

$$-\nabla_\epsilon KL(q_{(x_{k+1})} \mid\mid p)|_{\epsilon=0} = \mathbb{E}_{x\sim q}[\mathrm{trace}(\mathcal{A}_p\phi(x))] \tag{7}$$

This result states that the KL divergence between the distribution of the particles $q(x_{k+1})$ and $p(x)$ decreases by amount $\mathbb{E}_{x\sim q}[\mathrm{trace}(\mathcal{A}_p\phi(x))]$ by taking small step ϵ in direction $\phi(x)$. We want to maximize this decrease in KL divergence. This gives us the relation for the ideal particle perturbation direction:

$$\begin{aligned}\phi^*(x) &= -\arg\max_{\phi\in\mathcal{F}} \nabla_\epsilon KL(q_{(x_{k+1})} \mid\mid p)|_{\epsilon=0}\\ &= \arg\max_{\phi\in\mathcal{F}} \mathbb{E}_{x\sim q}[\mathrm{trace}(\mathcal{A}_p\phi(x))]\\ &= \sqrt{\mathbb{S}(q,p)}\end{aligned} \tag{8}$$

Updating the particles using the perturbation direction $\phi^*(x)$ has the effect of decreasing the KL divergence between $q(x_{k+1})$ and $p(x)$ by an amount equal to $\epsilon\sqrt{\mathcal{S}(q,p)}$. We can update the particles iteratively using the strategies discussed above. Due to this, the particles approach the true posterior distribution and eventually follow the posterior distribution.

The method outlined above is Stein Variational Gradient Descent. Stein Variational Gradient Descent reduces the KL Divergence between the density of the particles and true (posterior) distribution without the need for explicitly calculating the KL divergence between the distributions.

2.3 Kernelized Stein Variational Gradient Descent Algorithm

Liu and Wang use a unit norm ball in the reproducing kernel Hilbert Space (RKHS) defined by a symmetric, positive-definite kernel $K(x, y)$ as the space of functions $\mathcal{F}$. Since a function ϕ chosen from this space will be sufficiently well-behaved and follow the boundary conditions, we know that the selection of $\mathcal{F}$ is valid.

The optimum perturbation direction for steepest decrease in KL divergence is:

$$\phi^*(x) = \arg\max_{\phi\in\mathcal{F}} \mathbb{E}_{x\sim q}[\mathrm{trace}(\mathcal{A}_p\phi(x))], \qquad where\ \ \mathcal{F} = \{\phi : ||\phi(x)||_{H^d} \leq 1\} \tag{9}$$

Note that since ϕ is function that gives the perturbation direction of particle x in R^d, it is a vector valued function in H^d. The above problem has a closed form solution that is given by:

$$\phi^*_{(q,p)}(\cdot) = \mathbb{E}_{x\sim q}[k(x,.)\nabla_x \log p(x) + \nabla_x k(x,.)] \tag{10}$$

From (10) and (6) we get the following relation to iteratively update the particles to converge to the true (posterior) distribution.

$$x_{k+1} = x_k + \epsilon\mathbb{E}_{x\sim q}[k(x,.)\nabla_x \log p(x) + \nabla_x k(x,.)] \tag{11}$$

The $\nabla_x \log p(x)$ term pushes the particles towards high density of the true distribution $p(x)$. At the same time, $\nabla_x k(x,.)$ moves the particles in a direction away from other particles. This balances the effect of $\nabla_x \log p(x)$ and prevents particles from collapsing to the mode of the distribution [3].

The algorithm for Kernelized Stein Variational Gradient Descent (KSVGD) is mentioned below [3].

Algorithm 1: Kernelized Stein Variational Gradient Descent

```
Input 1 : Target distribution with density $p(x)$.
Input 2 : Initial particles $\{x_i^0\}_{i=1}^n$
1 for $l = 1$ to $n$ do
2     Calculate $\nabla_x \log p(x)$
3     Compute $\hat{\phi} * (x) = \frac{1}{n}\sum_{j=1}^{n}[k(x_j^l, x)\nabla_{x_j}^l \log p(x_j^l) + \nabla_{x_j}^l k(x_j^l, x)]$
4     Update $x_i^{l+1} \leftarrow x_i^l + \epsilon_l \hat{\phi} * (x_i^l)$
5 end
Output : Particles $\{x_i\}_{i=1}^n$ approximating $p$.
```

2.4 Drawback of KSVGD

In KSVGD, we use (11) to update the particles. The expectation of (11) is approximated using an empirical mean. Notice that to update one particle, we have to evaluate the Kernel function and the gradient of the Kernel with every other particle. As the number of particles increases, the number of computations explodes. Worse, if x has a high dimensionality computational complexity will only worsen. Thus, the time taken to retrieve a large number of samples from a high dimensional distribution is immense and impractical. Given the deficiencies of Kernelized Stein Variational Gradient Descent, we propose a tweaked version of the algorithm known as Neural Stein Variational Gradient Descent (NSVGD).

3 Our Contribution

Our contribution is that we introduce a neural network in the optimization problem. We show that by adding the neural network we are able to accelerate the process of retrieving new samples from the posterior, thereby speeding up the inference.

4 Neural Stein Variational Gradient Descent

Our fundamental idea is that we use a neural network to learn and predict the perturbation direction $\phi(x)$ of the particles x. From (6), (8) the update equation of particles is:

$$x_{k+1} = x_k + \epsilon \left\{ \arg\max_{\phi \in \mathcal{F}} \mathbb{E}_{x\sim q}[\text{trace}(\phi(x)\nabla_x \log p(x)^T + \nabla_x \phi(x))] \right\} \tag{12}$$

We take the space $\mathcal{F}$ to be the space of functions representable by the neural network. For the above update equation to make the particles converge, we want $\mathcal{F}$ to be a space of bounded functions i.e. for $\phi \in \mathcal{F}$, $\lim_{||x||\to\infty} p(x)\phi(x) = 0$. To achieve this, we use the hyperbolic secant function as the first layer of the neural network. This ensures that even when the input swells, the function $\phi(x)$ remains bounded. Since $p(x)$ is a probability distribution $\lim_{||x||\to\infty} p(x) = 0$. Therefore, the required boundary condition is satisfied.

Our method involves iteratively conducting two steps. The inner loop trains the neural network to maximize $\mathbb{E}_{x\sim q}[\text{trace}(\phi(x)\nabla_x \log p(x)^T + \nabla_x \phi(x))]$. The trained neural network will have learned $\phi^*(\cdot)$. The outer loop then forward propagates the particles through the neural network to get the optimal perturbation direction $\phi^*(x)$.

We update the particles using $\phi^*(x)$. The algorithm that implements the aforementioned theory is outlined below.

Algorithm 2: Neural Stein Variational Gradient Descent

```
Input 1 : Target distribution with density $p(x)$.
Input 2 : Initial particles $\{x_i^0\}_{i=1}^n$
$\phi(x)$  : Current Function Represented by the neural network
1 for $l = 1$ to $n$ do
2     Initialize neural network
3     for $1$ to $m$ do
4         Train neural network to minimize loss = $-\mathbb{E}_{x \sim q}[\text{trace}(\phi(x)\nabla_x \log p(x)^T + \nabla_x \phi(x))]$
5     end
6     Compute $\hat{\phi} * (x) : Forward\ Propagate\ x\ through\ Neural\ Network$
7     Update $x_i^{l+1} \leftarrow x_i^l + \epsilon_l \hat{\phi} * (x_i^l)$
8 end
Output : Particles $\{x_i\}_{i=1}^n$ approximating $p$.
```

Notice that in each iteration of the outer loop, we are training a separate neural network for particles in that iteration.

A key insight is that instead of taking initial particles x_0 to be randomly initialized, they can be taken from a known distribution $q_0(x_0)$ in space R^d. This enables our method to generate samples from the particles used in training and to generalize to unseen particles. Initial particles can be fresh samples from the distribution $q_0(x_0)$. Since the first neural network was trained to learn the perturbation direction for $x_0 \sim q_0$, we get the correct $\phi^*(x_0)$ and particles are updated to get x_1. Now x_1 can be forward propagated through the second neural network and updated using $\phi^*(x_1)$. Repeating this process for all the saved neural networks, we eventually get samples from the true (posterior) distribution p(x). Thus, generating unseen samples from the posterior distribution can be done in a very small amount of time. The overhead of this method is the time taken to train the neural networks.

The KSVGD update is particle specific. If the initial particle changes, then the perturbation direction for the particle also changes. There is no learning component in the problem to generalize for unseen samples. For example, if we want to generate 1000 samples from a posterior distribution twice, KSVGD requires that we run the alrorithm twice. On the other hand, with NSVGD, we run the algorithm to train the networks only once. Samples are generated using the particles by forward propagation through the pre-trained networks and performing particle updates, both of which take relatively less time. We refer to this method as NSVGD inference. Further details about NSVGD and NSVGD inference are described in the experiments section.

5 Experiments

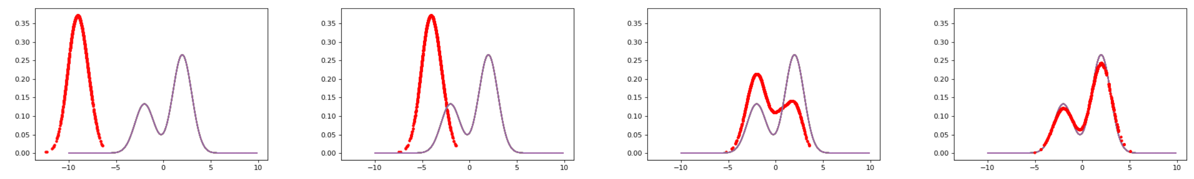
Our experiments compare the performance of KSVGD, NSVGD and NSVGD inference on a 1D Gaussian mixture toy example. The posterior distribution is assumed to be a univariate bimodal distribution $1/3 * \mathcal{N}(-2, 1) + 2/3 * \mathcal{N}(2, 1)$. Initial particles are taken to be samples from the distribution $\mathcal{N}(-10, 1)$.

Initially, we run both KSVGD and NSVGD to update the particles and ensure they converge to the posterior. Later the trained NSVGD neural networks are used for NSVGD inference by updating unseen particles to converge to the posterior. We refer to this step as NSVGD inference.

Visual results from KSVGD, NSVGD, and NSVGD inference are presented and compared. Also, we compare the time taken by the three methods. Lastly, we explore the Kolmogrov-Smirnov (KS) test as a measure of particle convergence.

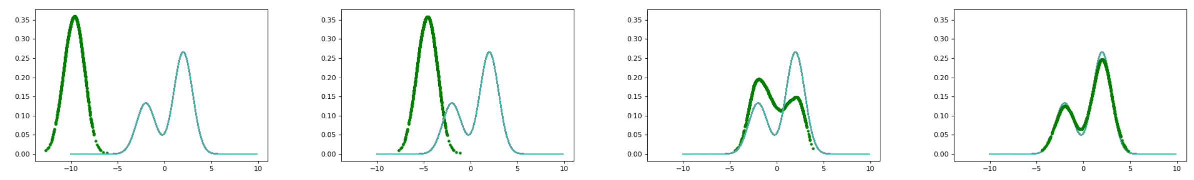
5.1 Kernelized Stein Variational Gradient Descent Training

For KSVGD, we use the Radial Basis Function (RBF) Kernel $k(x, x^{'}) = exp(-1/h * ||x - x^{'}||_2^2)$ for all the experiments. The bandwidth of the Kernel is different at every iteration. It is calculated by computing $h = med^2/logn$, where med is median of the pairwise distances between the current points $\{x_i\}_{i=1}^n$. This bandwidth setting was obtained from [3] and it ensures that for each particle x_i the gradient due to the point itself and the contribution from the other points is balanced out. We use AdaGrad to update the particles.



(a) 1st Iteration (b) 6th Iteration (c) 11th Iteration (d) 25th Iteration

Figure 1: A 1D Gaussian mixture example using KSVGD. The purple dashed lines are our target distribution and the red dotted lines represent the particle densities at the 1st, 6th, 11th, and 25th iterations. We use $n = 1000$ particles. Convergence occurs at the 25th iteration.



(a) 1st Iteration (b) 15th Iteration (c) 34th Iteration (d) 80th Iteration

Figure 2: A 1D Gaussian mixture example using NSVGD. The teal dashed lines are our target distribution and the green dotted lines represent the particle densities at the 1st, 15th, 34th, and 80th iterations. We use $n = 1000$ particles. Convergence occurs at the 80th iteration.

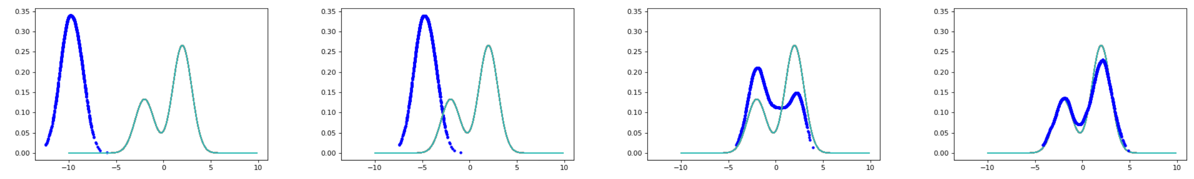
5.2 Neural Stein Variational Gradient Descent Training

As stated previously, we use the hyperbolic secant as the first layer of the neural network to ensure that boundary conditions are met. The Neural Networks have 2 additional fully connected layers, followed by an output layer consisting of a single unit.

For each particle, the inner loop is run for 100 iterations. We pass the current state of the particles to the neural network to get the output $\phi(x)$. Next we use $-\mathbb{E}_{x\sim q}[\text{trace}(\phi(x)\nabla_x \log p(x)^T + \nabla_x \phi(x))]$ as the loss and get the gradients of the loss with respect to the neural network parameters. We use the Adam optimization algorithm in the inner loop to train the neural network. With the optimal perturbation direction from the trained neural network, we use Adagrad to update the particles. An adaptive learning rate is used with the Adagrad algorithm to achieve better convergence of particles. The inner loop is run for 100 epochs. The outer loop is run for 80 epochs, meaning 80 different neural networks are trained in the process.

5.3 Qualitative Convergence of Particles

We present the convergence of 1000 particles initialized using the distribution $\mathcal{N}(-10, 1)$. Figures 1, 2, and 3 show the convergence of these particles to the true posterior distribution. The particle update causes the particles to move towards



(a) 1st Iteration (b) 15th Iteration (c) 35th Iteration (d) 80th Iteration

Figure 3: A 1D Gaussian mixture example using NSVGD inference. The blue dashed lines are our target distribution and the teal dotted lines represent the particle densities at the 1st, 15th, 35th, and 80th iterations. We use $n = 1000$ particles. Convergence occurs at the 80th iteration.

the region of the posterior density. A kernel density estimator (KDE) is used to evaluate the particle density for plotting. For our use, we selected the Epanechnikov kernel. Note that this Kernel is different from the Kernel used in KSVGD.

From the graphs, we can see that the convergence for KSVGD and NVSGD is comparable. We also observe that the convergence using NSVGD inference is reasonable, albeit not as reasonable as the other two methods. This is because with the NSVGD method we are updating particles that were used to train the neural network. We can compare this to the training phase in conventional machine learning literature. With NSVGD inference, we are moving particles not seen by the neural networks before. We can consider this the testing phase of machine learning. As we see from the plots, NSVGD inference is able to generalize quite well for unseen samples.

Note that even though particles trained using NSVGD show better convergence, NSVGD inference takes lesser time to update particles. We expound on this topic further in the next section.

5.4 Comparison of Time Taken by Methods

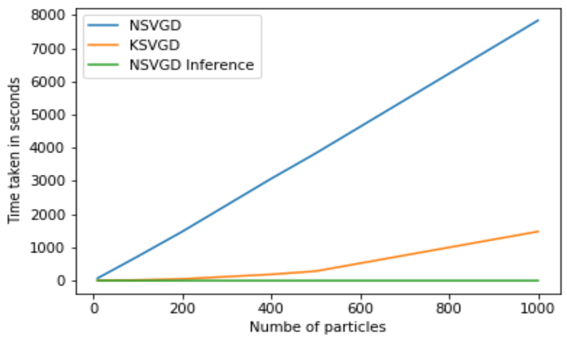


Figure 4: A computational efficiency comparison of the NSVGD, KSVGD, and NSVGD inference methods. NSVGD inference outperforms both KSVGD and NSVGD.

Figure 4 illustrates the time taken by the algorithms. All the experiments are run on the same machine using a CPU and under the same set of conditions.

From our graph, we can see that the time taken for NSVGD to complete increases as the number of particles increases. This is because of the computation involved in finding the loss $-\mathbb{E}_{x\sim q}[\text{trace}(\phi(x)\nabla_x \log p(x)^T + \nabla_x\phi(x))]$. The term $\nabla_x\phi(x)$ is a Jacobian matrix and we only want the trace of the Jacobian for the loss computation.

To implement KSVGD, NSVGD, and NSVGD inference we use Pytorch. However, Pytorch does not have an efficient method for computing the Jacobian matrix or it's trace. In order to calculate the trace of the Jacobian of a function learned by neural network in Pytorch, we have to forward propagate each particle through the neural network and take the gradient of the output $\phi(x)$ with respect to the input x. This is done for each particle in the inner loop, one particle at a time. Unquestionably, this hinders our ability to use matrix operations to speed up calculations and is responsible for the time taken in training NSVGD. The time spent might be reduced by using other frameworks that have efficient ways to calculate the Jacobian matrix.

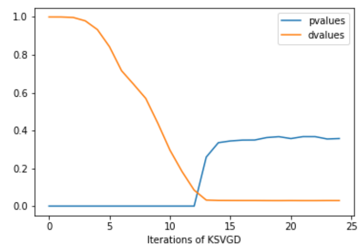
While training the neural networks in NSVGD, we save the intermediate weights of the neural network. These are used to update the particles in NSVGD inference. Since, this does not involve training the neural network in the inner loop, NSVGD inference is much faster than NSVGD and KSVGD as evident from the graph. KSVGD involves pairwise calculations of the Kernel function and it's gradient in the inner loop, hence it slows down with an increase in the number of particles.

The qualitative results from section 5.3 and the results presented in this section corroborate our claim that sample generation time is saved using NSVGD inference. If the task at hand involves continuously sampling the posterior distribution for some sufficiently large number of samples, NSVGD inference will be a better alternative to KSVGD.

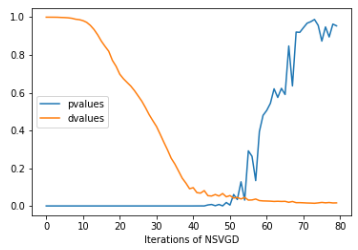
5.5 Comparison of Kolmogrov-Smirnov (KS) Test Results for Methods

In this section we look to quantify the "difference" between the distribution represented by the particles and the posterior distribution for our toy example. Note that we cannot use KL divergence as we do not have the analytical form of the distribution represented by the particles.

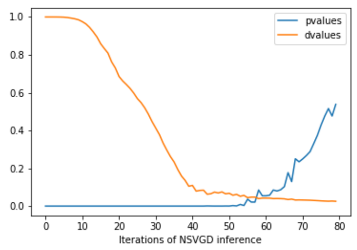
We use the Kolmogrov-Smirnov test to give an estimate of the difference between the sample population and the posterior distribution. The KS test fits find an approximate cumulative distribution function (cdf) for the samples and returns the maximum value of the difference between sample CDF and the analytical distribution CDF. This value is called the d-value. The KS test also gives a p-value which represents the probability of getting the d-value given that the samples were actually generated from the analytical distribution. We can say that the particles have converged to the posterior distribution if the d-value from the KS-test is small with a considerably high p-value.



(a) KSVGD KS test



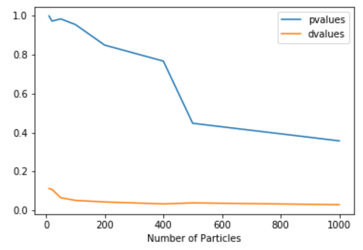
(b) NSVGD KS test



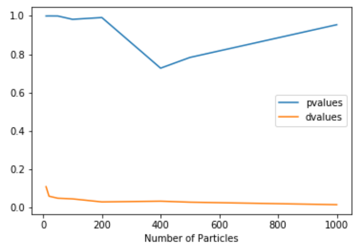
(c) NSVGD inference KS test

Figure 5: Here we show the p-values and d-values from the KS test on a 1D Gaussian mixture with $n = 1000$ particles. Figures (a), (b), and (c) use KSVGD, NSVGD, and NSVGD inference methods respectively.

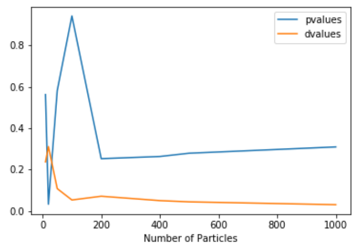
Figure 5 shows the KS Test value plots for 1000 particles. The iterations mentioned on the X-axis are the iterations of the outer loop. The d-value decreases with the number of iterations in all the three cases and achieves a low value. Surprisingly, NSVGD achieves a much better p-value on convergence than KSVGD. This tells us that particles updated using NSVGD best mimic the posterior. But due to the high computation time of this method, it is not feasible for regular use. Also notice that the p-value achieved by NSVGD inference is better than that of KSVGD. Thus, NSVGD inference provides sufficient approximate samples from the posterior.



(a) KSVGD KS test



(b) NSVGD KS test



(c) NSVGD inference KS test

Figure 6: The p-values and d-values from the KS test of 1D Gaussian mixture for $n = 10, 20, 50, 100, 200, 400, 500,$ and 1000 particles. Figures (a), (b), and (c) use KSVGD, NSVGD, and NSVGD inference methods respectively.

Figure 6 shows the KS Test value for converged particles. These values are calculated by taking the number of particles as: 10, 20, 50, 100, 200, 400, 500 and 1000. For NSVGD inference, the neural networks' weights were pre-learnt by training on 1000 particles. The plot reveals that NSVGD inference works poorly for a small number of particles as it exhibits a higher d-value.

For a small number of initial particles(20), we cannot say that the points are representative of the initial distribution. Some points might be sampled from a low density region and hence fraction of points from low density region might be high. A lower d-value could result given that we are generalizing using pre-trained neural networks with NSVGD.

Additionally, the p-value for NSVGD remains consistently high. Surprisingly, the p-value of KSVGD falls as the number of particles increase. This observation, together with the high d-value of NSVGD inference for a small number of particles and computational considerations, make KSVGD fit for generating a small number of samples and NSVGD inference fit for generating larger number of samples from the posterior.

6 Conclusion

We've established through multiple experiments, that the Neural Stein Variational Gradient Descent inference method can be used to generate samples from an intractable posterior distribution in a tractable amount of time. The experiments reveal that if a small number of samples are required, using Kernelized Stein Variational Gradient Descent is a better option. However, if generating a large number of samples from a posterior then NSVGD inference is more efficient.

Effectively generating a large number of samples from the posterior enables us to approximate high-dimensional posterior distributions quickly. These approximations should expedite Bayesian inference for models that have a high number of parameters.

7 Future Work

We demonstrated the utility of the proposed method on a 1D Gaussian Mixture. However, for the method to be practical, more experiments should be conducted. NSVGD inference should be validated by testing on a complicated multi-dimensional posterior distribution with multiple modes.

Another milestone in the same direction will be demonstrating efficient and accurate Bayesian Inference of the parameters in a complicated model using NSVGD inference.

References

[1] D. M. Blei, A. Kucukelbir, and J. D. Mcauliffe. Variational Inference: A Review for Statisticians. , 2018. URL `https://arxiv.org/pdf/1601.00670.pdf`.

[2] J. Gorham and L. Mackey. Measuring Sample Quality with Stein's Method. . URL `https://arxiv.org/pdf/1506.03039.pdf`.

[3] Q. Liu and D. Wang. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. . URL `https://arxiv.org/pdf/1608.04471.pdf`.