PARTICLE-BASED ALGORITHMS FOR BAYESIAN NEURAL NETWORK - HAMILTONIAN MONTE CARLO AND STEIN VARIATIONAL GRADIENT DESCENT

SENIOR HONOR THESIS

Duc Nguyen
Department of Computer Science
Tufts University

Michael Hughes
Department of Computer Science
Tufts University

Shuchin Aeron
Department of Computer Science
Tufts University

ABSTRACT

Bayesian inference is a powerful framework to do prediction under uncertainty. Bayesian Neural Networks combine the flexibility of neural networks with Bayesian machine learning's ability to incorporate uncertainty into prediction. While approximate Bayesian makes restricting assumption on the class of posterior distribution, particle-based algorithms can learn more flexible posterior distributions. The following thesis outlines, compares and contrast two particle-based algorithms for Bayesian Neural Network: Hamiltonian Monte Carlo and Stein Variational Gradient Descent. It then proposes a third algorithm that is a hybrid of the two aforementioned algorithms.

***K*eywords** Machine learning, Hamiltonian Monte Carlo · Stein Variational Gradient Descent

1 Introduction

1.1 Bayesian statistics

Machine learning is a growing sub-field of artificial intelligence. The goal in machine learning is to create algorithms that make sense of and find patterns in data. An important class of machine learning algorithms is probabilistic

algorithms. A probabilistic algorithm performs learning by first building a probabilistic model that attempts to explain the data. The algorithm then learns the model's parameters, the tuning knobs that control this probabilistic model. The algorithms of interest in this thesis are all probabilistic algorithms.

On the other hand, machine learning problems can be roughly divided into 3 groups: supervised learning, unsupervised learning and reinforcement learning. In supervised machine learning, the machine learning algorithm is given a set of observations and labels $\{X, Y\} = \{x_i, y_i\}_{i=1}^{N}$, where y_i could be numerical for regression tasks or categorical for classification tasks. The goal is to predict the label Y^* for unseen data X^*. To solve the supervised learning problem, many probabilistic machine learning algorithms first place a probabilistic model over X and Y, that could explain how the data was generated. This model is in turn controlled by hidden parameters denoted as θ. In fact, the task of learning a distribution over the hidden parameters θ is also known as inference. All of the algorithms that this senior thesis focus on are inference algorithms.

1.1.1 The frequentist approach

The frequentist approach is to find, given unlabeled data X^*, a single prediction value Y^*. Maximum likelihood, and maximum aposterior priori are both examples of the frequentist approach. Maximum likelihood is to learn hidden parameters such that $\theta^* = \arg\max p(Y|X, \theta)$ and use θ^* for prediction on unlabeled data. The maximum a posteriori (MAP) approach is to learn parameters that maximizes the posterior distribution ie $\theta^* = \arg\max p(Y|\theta, X)p(\theta)$ where $p(\theta)$ is the prior distribution over the parameters. Except in the case of ensemble learning [9], the frequentist approach does not naturally produce a notion of uncertainty associated with its prediction.

As a visualization, the following figure shows the difference between the frequentist approach and the Bayesian approach (elaborated on later). The upper half of the figure shows the frequentist approach. The dotted line shows the "prior" distribution or the sampling distribution while the solid line shows the likelihood function, which is computed given data X, Y. The frequentist approach will focus on finding point estimate of the parameters, such as the maximum likelihood estimate of the parameter $\hat{\theta}$.

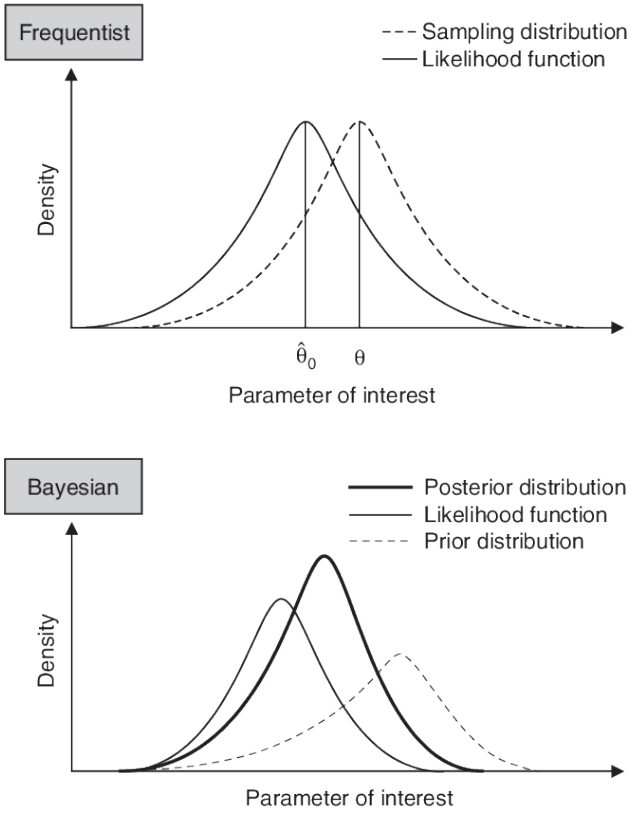

Figure 1: Frequentist vs Bayesian approach

1.1.2 The Bayesian approach

On the other hand, the Bayesian approach is not only to learn the posterior distribution $p(\theta|X, Y)$. With the posterior distribution, one could compute the predictive distribution $p(y*|y) = \int_\theta p(y*|X, \theta)p(\theta|X, y)d\theta$. As opposed to the frequentist approach, the Bayesian approach readily incorporates uncertainty into its prediction. The ability to estimate uncertainty in prediction is highly valuable in settings where wrong predictions have high costs. An example is in medical diagnosis applications such as cancer prediction where a false negative prediction, where the algorithm predicts that the patient does not have cancer when he in fact does, has devastating consequences. In these settings, if the prediction algorithm has some notion of uncertainty, the doctor could decide whether to conduct further tests to verify the presence of cancer.

In order to follow the Bayesian approach, we need to define a prior distribution over the hidden parameters θ and a likelihood distribution over observed data. In formal notations, the prior is $p(\theta)$ and the likelihood term is $p(Y|X, \theta)$. The posterior in its exact form is $\frac{p(Y|X,\theta)p(\theta)}{P(Y)}$.

In practice, however, computing the exact posterior distribution is very difficult because it requires computing $p(y)$, which is also called the normalizing constant. There have been many algorithms invented to overcome such limitation, by avoiding computing the posterior explicitly. Hamiltonian Monte Carlo, a major focus of this thesis, is an example of a sampling algorithm. A sampling algorithm constructs instances of θ so that with many samples, we can substantially "cover" the posterior. Stein variational gradient descent, another important foundation of this thesis, is an example of particle based algorithms. At a high level description, these algorithms place a fixed set of particles so that they cover the posterior distribution.

The following figure shows the result that a generic particle or sampling based algorithm tries to achieve. With a posterior distribution with two modes, a sampling based algorithm tries to sample such that there are more samples in the high density regions of the distribution (around the two modes). On the other hand, a particle based algorithm tries to place the particles such that there are more particles in the high density regions of the distribution.

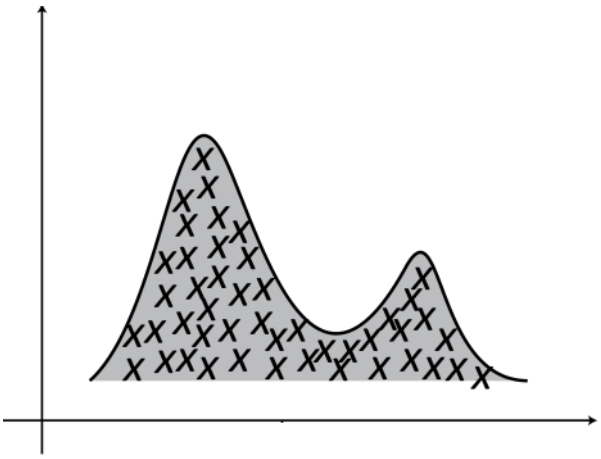

Figure 2: Both sampling and particle based algorithms aim to "cover" the posterior distribution

A Bayesian approach that is not covered in details in this thesis is approximate inference or variational inference [2]. Approximate algorithms to restrict the search for the posterior parameters θ within a well defined class of distributions. On one hand, this makes computing the posterior tractable but on the other hand, it also limits the flexibility of the learned posterior distribution. For example, the statistician might assume that the approximate posterior distribution is Gaussian, $q(z) = N(m, S)$ and optimizes the parameters m, S by minimizing the KL divergence, $KL(q(z)|p(z))$ [27], or maximizes the evidence lower bound [2].

An approximate distribution, that is often used, is the mean field approximation ([10]). However, the mean field approximation is not a very flexible assumption because it assumes independence among all hidden variables. [7] and [8] extend beyond this limited assumption of mean field approximation by learning latent variables where there is some form of dependency among the latent variables. In these works, stochastic optimization is used to scale to large data sets.

A common limitation that approximate inference algorithms suffer from is when the true posterior distribution is very different from the class of approximate posterior. For example, many approximate inference algorithms such as [11] assumes a Gaussian distribution over the hidden variables. If the true posterior is multimodal or not Gaussian, the learned posterior might not give reasonable prediction on test data.

1.2 Neural networks

Neural networks are a class of machine learning algorithms that were invented in the 1960s. Neural networks are inspired by the human brain by as they are organized in multiple layers with connecting weights between layers. This structure mirror the synaptic connections among neurons in the human brain.

Neural networks and particularly deep learning, have seen a resurgence of interest in a recent decade following their incredible success in many perception tasks such as computer vision ([26], [13]) and natural language processing ([22]). Neural networks excel in settings where there is an abundance of data and accompanying computational power. Therefore, neural networks have become a major powerhouse in modern data science.

However, neural networks are still a very active area of research because there are still many limitations and things not well understood about neural networks. Over-fitting, the problem where a learning algorithm performs very well on training data but much more poorly on unseen test data, is an important practical issue [15]. In addition, neural networks can sometimes give wrong predictions with very high confidence [25]. This again highlights the potential limitation of applying neural networks to settings where wrong predictions have high cost such as predictive health-care. In such situations, we want our algorithm to give not only a point prediction, but a range or a distribution of possible predictions and some notion of uncertainty.

There have been works which address this problem with neural networks such as [14], which uses standard neural networks and produces predictive distributions by combining multiple neural networks. Although straightforward and

easy to implement, naive ensemble methods might not be efficient at exploring the parameter space. This senior thesis will also compare between an ensemble of neural networks and the Bayesian methods.

1.3 Bayesian neural networks

Bayesian neural networks gained strong interest as a research field after it was found that there is a strong connection between dropout [28], a popular technique to prevent over-fitting, and a Bayesian treatment of neural networks [6]. The weights of the neural networks are analogous to the hidden parameters θ in probabilistic models. And in the spirit of bayesian approach, Bayesian treatment of neural networks learns the posterior distribution over weights of the networks.

The various approaches such as sampling-based algorithms, particle based algorithms and variational inference can be readily extended to apply to neural networks [23]. For example, we might assume that the approximate posterior over the weights of the neural network is Gaussian or $q(w_i) = N(m_i, S_i)$ for each weight w_i in the neural network. We can then optimize the parameters m_i and S_i through maximizing the evidence lower bound.

An advantage that BNNs bring over traditional variational inference is evident in datasets where we have reasons to believe that there is a non-linear relationship between the observed features and the latent variable. The neural network, with an appropriate choice of activation function, can approximate the non-linear mapping between the observation space and the latent variable space. The universal approximation theorem [4] strongly corroborates the representation power of neural networks.

Though neural networks with variational inference are a powerful class of approximate inference algorithms, they still share a similar issue with standard approximate inference. The assumption about the approximate posterior distribution can limit the flexibility to capture more complicated and multimodal distributions. In the next sections, we outline two methods, Hamiltonian Monte Carlo (sampling based) and Stein Variational Gradient Descent (particle based). Though seemingly distinct at first, we will see that both algorithms have strong theoretical connection. This deep connection would allow us to conceive a third algorithm which is the hybrid of HMC and SVGD.

2 Hamiltonian Monte Carlo

2.1 Motivation from Monte Carlo Markov Chain sampling

Sampling algorithms are a very important class of inference algorithms. Recall that ultimately, we want to be able to evaluate the predictive distribution $p(y*|y) = \int_\theta p(y*|X,\theta)p(\theta|X,y)d\theta$. Although we cannot directly evaluate the integral, we can sample directly from the posterior distribution. With "enough" samples, we could estimate the above integral as an average with respect to those samples.

It is noteworthy that it is not feasible to sample from the entire distribution space. Especially in higher dimension, the region of high probability becomes vanishing compared to the rest of the entire distribution space. The figure below demonstrates this phenomenon. With a distribution that concentrates in the center, when the dimension increases, the volume of the high density region becomes smaller and smaller.

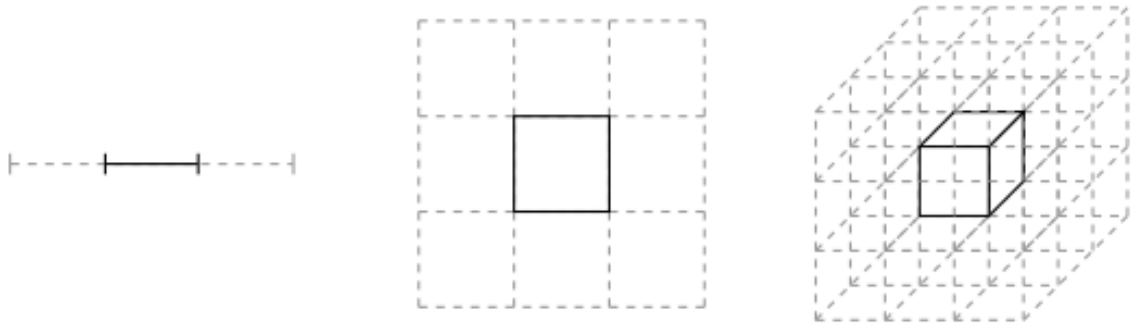

Figure 3: Problem facing sampling algorithms in high dimensions

Therefore, effective sampling algorithms try to sample from the large regions of significant mass. Regions of 'significant' probability mass is also known as the typical set. The figure below helps to show an intuitive description of the typical set. The light pink line shows the pdf function starting from the mode. The light red line shows the volume of the distribution against distance from the mode. The dark red line shows the expected "volume", which is used to compute expectation with respect to the reference distribution. We can see that most of the contribution to this expected volume lies within a region where the probability is significant and so is the volume. This is the typical set.

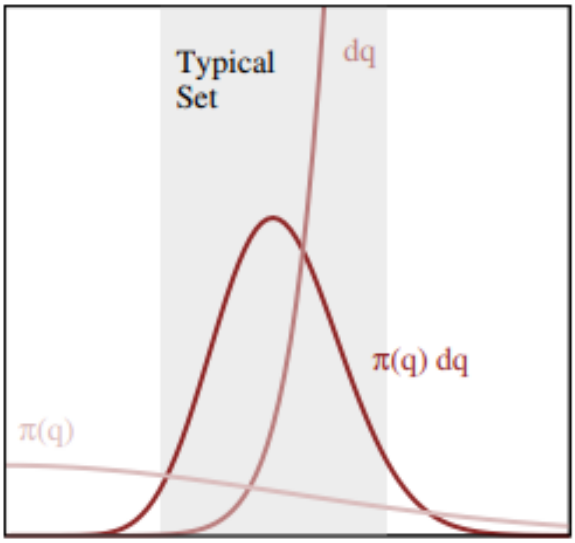

Figure 4: Visualization of the typical set

One such algorithm to sample efficient from the typical set is Monte Carlo Markov Chain (MCMC) with Metropolis Hastings [21]. However, Metropolis Hastings can exhibit random walk behaviours in high dimensions and a chain of MCMC can extend very long without resulting in an accepted sample.

Hamiltonian Monte Carlo [24] aims to avoid the random walk behaviour and stays within the typical set. The foundation of HMC is based on Hamiltonian physics with a very sophisticated wealth of mathematics behind. We highly recommend interested readers to refer to the original paper by Radford Neal. For more application-focused readers, a less mathematically involved introduction to HMC can be found at [1]. In the following subsections, we present the key ideas behind Hamiltonian Monte Carlo.

2.2 Derivations from Hamiltonian physics

In this subsection, we will look at the derivations of Hamiltonian Monte Carlo, first it its most general form, then in a much more concise form.

Hamiltonian Monte Carlo is founded on ideas from Hamiltonian Physics. In a physical system, the law of energy conservation dictates that the sum potential and kinetic energy is the total energy. And total energy of the isolated system always remains constant. The law of conservation of energies, when built into the algorithm of HMC, results

in sampling paths that often stay within the typical set instead of exhibiting random walk behaviours seen in other sampling methods such as MCMC with Metropolis Hastings.

In order to accomplish this task, HMC augments the parameter space with "fake" momentum variables. As an analogy, imagine a satellite orbiting earth. The system can be described in terms of the satellite's position and its momentum. The position variable is of primary interest and is in fact the random variable we want to sample for.

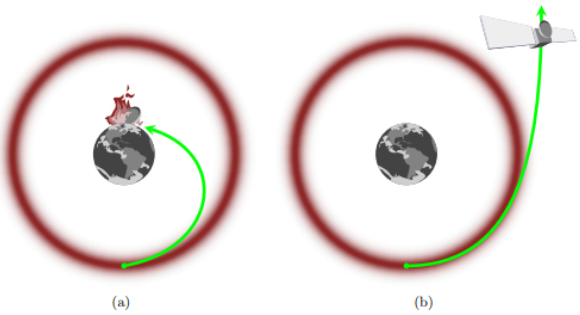

Figure 5: Satellite system with position and momentum variable. Ideally, we want the satellite to travel and explore the "typical set" denoted by the shaded circle. Too little momentum (a), the satellite comes crashing down to earth. Too much momentum (b), it flies off onto space.

We wish to sample for θ from a distribution in $\mathcal{R}^d$. In HMC, θ plays the role of the **position** variable while the momentum variable has the same dimension as the position variable and is denoted by p. The augmented system is $(p, \theta) \in \mathcal{R}^{2d}$. Define the canonical distribution as the joint distribution over position and momentum, written as $\pi(p, \theta) = \pi(p|\theta)\pi(\theta)$.

Define the Hamiltonian (energy) function of the whole system $H(p, \theta)$ as

$$H(p, \theta) = -\log \pi(p, \theta) = -\log \pi(p|\theta) - \log \pi(\theta) = K(p, \theta) + U(\theta)$$

Where the first term is the kinetic energy and the second term, which corresponds to the density of the target distribution, is the potential energy. Note again that the kinetic term might depend on both position and momentum. However, the potential term depends only on the position. The key idea in [24] incorporates the Hamiltonian equation, which dictates the conservation of total energy. The Hamiltonian equations are as follows:

$$\frac{d\theta}{dt} = \frac{dH}{dp} = \frac{dK}{dp}$$
$$\frac{dp}{dt} = -\frac{dH}{d\theta} = -\frac{dK}{d\theta} - \frac{dU}{d\theta}$$

The Hamiltonian equations are a system of partial differential equations. The stationary distribution over θ satisfies $\theta|X, Y \sim \exp(-U(\theta))$. Obviously, this is a very general form of the Hamiltonian equations. For our purpose, since we wish to sample from the posterior distribution, we can define $U(\theta)$ to be the negative log posterior and solve for the above Hamiltonian equations.

However, we can't solve the above system PDE in a closed form. One solution is to discretize it. There are a host of algorithms that are designed to solve discretized PDE. In the next section, we outline the leap-frog algorithm to solve this PDE. Note that $\frac{dU}{d\theta}$ is the derivative of the log posterior distribution and therefore we only need to compute the derivative of the unnormalized posterior distribution. An important algorithmic consideration in HMC is the choice of the kinetic energy term $K(p, \theta)$ [20].

Now, we have looked at HMC in its most general form. In practice, a common choice for the kinetic term K, is $K(p, \theta) = -\frac{1}{2}p^\top M^{-1}p$ and the potential term, U is defined as $U(\theta) = -\log p(Y|X, \theta) - \log p(\theta)$ is the negative log of the unnormalized posterior distribution. In this formulation, the above system of Hamiltonian equations can be rewritten as:

$$d\theta = M^{-1}rdt$$
$$dp = -\nabla U(\theta)dt$$

2.3 Leap frog algorithm

The leap-frog algorithm is an iterative algorithm that discretizes the partial differential equation specified by the Hamiltonian equations as outlined at the end of the previous subsection. It is an instance of symplectic integrators [5], a class of integration algorithms often used in modeling physical Hamiltonian dynamics because the trajectories that these integrators produce conserve volume. As a result, the trajectories stay on the same energy level. What the leap frog algorithm returns is a proposed new state (p, θ). As noted before, the position variable θ is the only variable of interest. The leap frog algorithm has such name because the update step in the algorithm alternate between the momentum update and

position update and there is always a half-step difference between the two updates (as shown in the algorithm box below)

Algorithm 2.1: Leap-frog algorithm

Input: p_0 : starting momentum variable, θ_0 : starting position variable, N : number of leap-frog steps, ϵ : step size

Output: The final position variable θ and momentum variable p

$\#$ Take a first half step

$p = p - \epsilon \times \frac{dU}{d\theta}/2$

$\#$ Alternate full step updates between position and momentum

for *t = 1 to N* **do**

  $\#$ Make a full step for the position

  $\theta = \theta + \epsilon \times p$

  $\#$ Make a full step for the momentum, except at end of trajectory

  if $(i \neq L) p = p - \epsilon \times \frac{dU}{d\theta}$

$\#$ Make a half step update for momentum at the end.

$p = p - \epsilon \times \frac{dU}{d\theta}/2$

$\#$ Negate momentum at end of trajectory to make the proposal symmetric

$p = -p$

2.4 Hamiltonian Monte Carlo Algorithm

Note that the leap frog algorithm outlined in the previous subsection returned a proposed state (θ, p). The complete HMC algorithm uses Metropolis Hastings to decide whether to accept or reject this proposed state. We have the algorithm outline as below:

Algorithm 2.2: Hamiltonian Monte Carlo

Input: $p(z)$: target distribution that we wish to sample from

Output: Set of final points that approximate the target distribution

Initialize position variable θ **for** *t = 1, 2, ...* **do**

  Sample a random momentum variable p_t

  Obtain proposal state (p^*, θ^*) via the leap-frog algorithm

  Use Metropolis-Hastings acceptance rule. Accept sample q^* if rand() $\leq H(p^*, \theta^*) - H(p_t, \theta_t)$

  Update $\theta_{t+1} = \theta^*$

2.5 Stochastic gradient Hamiltonian Monte Carlo

One of the major problems facing modern machine learning is scalability. The abundance of data allows machine learning algorithms to have bigger impact. However, it also poses a lot of practical challenges. A common framework in big data machine learning is online algorithms. These algorithms are often modifications of existing machine learning algorithms. However, instead of taking in "full batch" data, ie all of observation and label sets (X, Y), they use random subsets of data, known as mini batches.

Hamiltonian Monte Carlo is one algorithm that permits a stochastic version. However, the catch is that a naive implementation of stochastic hamiltonian monte carlo is not theoretially sound [3]. Recall that the system of Hamiltonian equations that characterize the HMC dynamic is:

$$d\theta = M^{-1}pdt$$

$$dp = -\nabla U(\theta)dt$$

On the other hand, the system of Hamiltonian equations that characterize the naive stochastic HMC dynamic is:

$$d\theta = M^{-1}pdt$$

$$dp = -\nabla U(\theta)dt + \mathcal{N}(0, 2B(\theta))dt$$

where $B(\theta)$ is called the diffusion matrix that reflects the noisy gradient computed using mini-batching. This system of equations, however, does not result in the desired stationary distribution.

To develop proper stochastic gradient hamiltonian monte carlo, [3] proposed adding a correction term to counter act the effect of the diffusion matrix. This correction term has an intuitive interpretation as a "friction" term. The corrected system of Hamiltonian equations is:

$$d\theta = M^{-1}pdt$$

$$dp = -\nabla U(\theta)dt - CM^{-1}pdt + \mathcal{N}(0, 2(C - \hat{B}(\theta))dt + \mathcal{N}(0, 2B(\theta))dt$$

Where C is a user-specified friction term while $\hat{B}$ is the mini-batch estimated diffusion matrix such that $C \succeq \hat{B}$. It was shown in [3] that the above system of Hamiltonian equations have the desirable stationary distribution if $\hat{B} = B$.

More importantly, there is a strong connection between stochastic gradient HMC and stochastic gradient descent with momentum. Ie, we could replace the leap frog algorithm in standard HMC with a series of stochastic gradient descent step with momentum. The appropriate choice of momentum term will implicitly inform the appropriate quantity of $\hat{B}$. With these points established, we have the following outline of the stochastic gradient HMC algorithm implemented via SGD.

Algorithm 2.3: Stochastic Gradient Hamiltonian Monte Carlo

Input: $p(z)$: target distribution that we wish to sample from, N : number of leap-frog steps, ϵ : step size, ρ : momentum constant

Output: Set of points that approximate the target distribution

Initialize position variable θ_0

for *t = 1, 2, ...* **do**

  Select a mini-batch $(\hat{X}, \hat{Y})$

  Starting from θ_t, do N steps of stochastic gradient optimization with momentum ρ and step size ϵ to obtain θ^*

  for *n = 1 to N* **do**

    $\theta_{\frac{1}{2}} = \theta_n - \epsilon \nabla_\theta \hat{U}(\theta_n)$

    $\theta_{n+1} = \rho\theta_n + (1-\rho)\theta_{\frac{1}{2}}$

  Update $\theta_{t+1} = \theta^*$

---

3 Stein Variational Gradient Descent

3.1 An overview

The HMC algorithm explained in the previous section can be broadly classified as a sampling-based algorithm. In this section, we will focus on Stein Variational Gradient Descent, which can be broadly classified as a particle based algorithm. However, these are rather surface level distinctions as there is very well founded mathematical connections between HMC and SVGD. As the reader follows this section, it will become clearer how this connection arises.

Stein variational gradient descent (SVGD) is a general Bayesian inference algorithm [19]. Intuitively, we can think of it as an iterative algorithm that transform a set of particles from some initial distribution onto a target distribution. The target distribution that we're interested in is the posterior distribution. Similar to how a sampling algorithm seeks to generate a lot of samples that cover the mass of the posterior distribution, particle-based algorithms such as SVGD aims to spread the initial set of particles to accomplish the same goal.

We first start with a higher level, intuitive description of the algorithm. As a summarizing outline, consider an example where we wish to sample from the posterior distributions $p(z|X, Y)$ with X, Y being the observation and label sets respectively. We start from a reference distribution $z \sim N(0, I)$, and $z \in \mathcal{R}^d$ and sample 100 points from this distribution:

$$S_0\{z_1^0, z_2^0, ..., z_{100}^0\}$$

From these initial particles, SVGD then iteratively applies transformations to the above sets of points. The particular type of transformations will be elaborated in later subsections.

$$\{z_i^0\}_{i=1}^{100} \rightarrow \{z_i^1\}_{i=1}^{100} \rightarrow \{z_i^2\}_{i=1}^{100} \rightarrow ... \rightarrow \{z_i^*\}_{i=1}^{100}$$

Where the final positions of the particles, $\{z_i^*\}_{i=1}^{100}$, approximate the posterior distribution. The outline of the SVGD algorithm is shown below:

Algorithm 3.1: SVGD

Input: Target distribution $p(z|X, y)$

Set of n initial points S_0 from reference distribution $q_0(z)$

Kernel function k

Output: Set of final points that approximate the target distribution

for *t = 1, 2, ...* **do**

  for *i = 1, 2, .., n* **do**

    $z_i^{t+1} \leftarrow z_i^t + \epsilon_t \hat{\phi}^*(z_i^t)$

    where

    $\hat{\phi}^*(z_i^t) = \frac{1}{n} \sum_{j=1}^{n} [k(z_j^t, z_i) \nabla_{z_i^t} \log p(z_i^t|X, y) + \nabla_{z_i^t} k(z_i^t, z)]$

SVGD is a powerful and flexible tool to learn a target distribution $p(z)$ by moving from initial distribution $q_0(z)$ iteratively with a bijective transformation in each step. If we can compute the derivative of the unnormalized posterior with respect to the particles, then we can readily apply SVGD. In the following subsections, we look more deeply at the mathematical foundations of SVGD. These subsections are meant to present the important results in the works from [19] and are by no means complete and rigorous. We highly recommend interested readers to consult the original paper for more information.

3.2 The optimization problem

The mathematical intuition behind SVGD can be drawn by comparing it to variational inference. In variational inference, we try to approximate the posterior distribution by first specifying a family of parametric distribution that we want to search in, denoted by Q. The optimal solution within this family, $q^*(z)$ can be found by minimizing the KL divergence $KL(q(z)\|p(z|X, Y))$ such that $q(z) \in Q$. In standard variational inference, we assume the approximate distribution has a well defined, parametric form.

However, in SVGD, we "restrict" the class of posterior distributions by specifying the class of iterative transformation functions that we can apply to the set of particles. The transformation function that SVGD uses is $q_{[T]}(z)$, which is any

distribution obtained by applying smooth transformations on the current particles z. Similar to variational inference, the optimization problem for SVGD is also minimizing the KL divergence between the approximate posterior and the true posterior.

$$\min_{T} KL(q_{[T]}(z) \| p(z|X, y))$$

3.3 The transformation function

Finding in closed form a transformation function T that minimizes the KL objective function is very difficult. Therefore, the authors of SVGD proposed using $T(z_0) = G_t(G_{t-1}(G_{t-2}(....(G_1(z_0))...)$. In other words, the class of transformation is a series of iterative transformations. One particular choice of transformation function that works is $G(z) = z + \epsilon\phi(z)$ where ϕ is a smooth vector-valued function that controls the direction and ϵ is the step size. Another way of interpreting this transformation function is that it is analogous to taking a gradient descent step towards minimizing the KL divergence.

3.4 Stein identity

For conciseness, in this subsection and the following, we refer to the posterior $p(z|X, Y)$ simply as $p(z)$. Now, we look at an important insight developed in [19], which is the Stein identity:

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

where the stein operator is

$$\mathcal{A}_p \boldsymbol{\phi}(z) = \boldsymbol{\phi}(z) \nabla_z \log p(z)^\top + \nabla_z \boldsymbol{\phi}(z)$$

and (recall that $z \in \mathcal{R}^d$):

$$\boldsymbol{\phi}(x) = [\phi_1(x), ...\phi_d(x)]^\top$$

The stein identity is significant because as we want to minimize the KL divergence between our approximate posterior and the true posterior. And the stein identity gives us a way of computing the derivative of the KL divergence. Optimization algorithms tell us that once we have computed the derivative of the KL divergece, we could use gradient

descent to minimize the objective function. Therefore, we want to find the transformation function T that gives the maximum KL divergence in the opposite direction. In other words, we want to find ϕ to maximize $\mathbb{E}_{z\sim q}[tr(\mathcal{A}\boldsymbol{\phi}(z))]$.

Unfortunately, for a general transformation function T, the above maximization problem doesn't have a closed form. However, if we further restrict the norm of $\phi(.)$ to be within a unit ball in the producing kernel Hilbert space, $\|\boldsymbol{\phi}\|_{\mathcal{H}} \leq 1$, then we do have a closed form solution. In fact, let:

$$\mathbb{S}(q,p) = \max_{\phi\in\mathcal{H}}\{[\mathbb{E}_{z\sim q}(tr(\mathcal{A}_p\boldsymbol{\phi}(z)))]^2, \text{ st } \|\boldsymbol{\phi}\|_{\mathcal{H}} \leq 1\}$$

Then the optimal choice (that maximizes $\mathbb{E}_{z\sim q}[tr(\mathcal{A}\boldsymbol{\phi}(z))]$ such that $\|\boldsymbol{\phi}\|_{\mathcal{H}} \leq 1$) of $\boldsymbol{\phi}$ has a closed form solution and is written as:

$$\boldsymbol{\phi}^*(.) = \mathbb{E}_{z\sim q}[\mathcal{A}_p k(z,.)]$$

in which case $\mathbb{S}^* = \|\boldsymbol{\phi}^*\|^2_{\mathcal{H}}$. Where k is a strictly positive definite kernel in some reproducing kernel hilbert space (RKHS) $\mathcal{H}$. A common choice of k is the radial basis function kernel. When we substitute this back into the Stein identity, we obtain the optimal choice of transformation that leads to the steepest descent direction on the gradient of $KL(q\|p)$ as:

$$\boldsymbol{\phi}^*(.) = \mathbb{E}_{z\sim q}[k(z,.)\nabla_z \log p(z) + \nabla_z k(z,.)]$$

Since we do not have a closed form for the above expression, it must be estimated as an average over all particles, which we can readily do since we always have the set of particles. So with all of these points established, we can obtain the algorithm outlined in the previous subsection. It is possible to interpret the above update formula in terms of attractive and repulsive forces among particles. The first term corresponds to the attractive forces that these particles exert on one another, pulling towards the mode of the posterior distribution. The second term represents the repulsive forces that the particles exert on one another. The repulsive force ensures that the particles spread out more and cover a wider region of the posterior distribution.

3.5 Bayesian Neural Networks with SVGD

It is notable that SVGD is a very flexible and general machine learning algorithm. In fact, a closer inspection at the algorithm outline tells us that if we can compute the derivative of the unnormalized posterior with respect to the particles and compute a notion of kernel discrepancy between any two particles, we can apply SVGD. In fact, in the original paper, the authors of SVGD noted that SVGD could be extended to neural networks. However, at the beginning of this thesis, there had been no work on applying SVGD to deep neural network. As of writing this thesis and to the best of our knowledge, no work has been done on applying SVGD to convolution neural networks.

In order to apply SVGD to neural networks, let us start by writing down the probabilistic model. To reuse notations, let θ denote the parameters of the neural network (weights and biases). We place a prior over the parameters of the neural networks: $p_0(\theta) = \prod_{ij} p(w_{ij}) \prod_i p(b_i)$ for each level i, where $p(w_{ij}) \sim N(0, I)$, $p(b_i) \sim N(0, I)$ for each weight and bias. The likelihood is defined as: $p(y|x, \theta) \sim N(\theta(x), \sigma^2)$ where σ is a global parameter.

Similar to Bayesian inference with neural networks, we want to estimate the distribution of the posterior $p(\theta|X, Y)$, ie the distribution over the weights and biases of the neural networks. We accomplish this via SVGD. Note again that we do not need to compute the full posterior, we only need to take the derivative of the unnormalized posterior with respect to the neural network parameters θ. This means that auto differentiation, a popular and easy to use tool in deep learning, can be used to compute the derivative of the unnormalized posterior with respect to θ.

A possible choice of the kernel function k is the RBF-kernel (for all experiments in this thesis, RBF is the kernel function of choice). We have the following BNN-SVGD algorithm.

Algorithm 3.2: BNN-SVGD

Input: Target distribution $p(\theta|X, Y)$ and set of neural networks sampled from $p_0(\theta)$

Output: Set of neural networks that approximate the posterior distribution

for *t = 1, 2, ...* **do**

  for *i = 1, 2, .., n* **do**

    $\theta_i^{t+1} \leftarrow \theta_i^t + \epsilon_t \hat{\phi}^*(\theta_i^t)$

    where

    $\hat{\phi}^*(\theta_i^t) = \frac{1}{n} \sum_{j=1}^{n} [k(\theta_j^t, \theta_i) \nabla_{\theta_i^t} \log p(\theta_i^t|X, Y) + \nabla_{\theta_i^t} k(\theta_i^t, \theta)]$

3.6 Stochastic SVGD

An advantage that SVGD has over HMC is that SVGD can be readily extended to the stochastic (ie using minibatch) setting. Whereas stochastic HMC, if implemented naively, is provably not convergent. Stochastic optimizations allow SVGD to scale to much bigger data sets where full batch training is not practical. This highlights the appropriateness of applying SVGD to deep neural networks, which are well suited for big data problems. The following outlines stochastic BNN-SVGD algorithm. Instead of computing the derivative of the full posterior, we use mini-batches to obtain stochastic estimates of the posterior gradient.

Algorithm 3.3: Stochastic BNN-SVGD

Input: Target distribution $p(\theta|X, Y)$ and set of neural networks sampled from $p_0(\theta)$

Output: Set of neural networks that approximate the posterior distribution

for *t = 1, 2, ...* **do**

Pick a minibatch $\tilde{X}, \tilde{Y} \in (X, Y)$

for *i = 1, 2, .., n* **do**

$\theta_i^{t+1} \leftarrow \theta_i^t + \epsilon_t \hat{\phi}^*(\theta_i^t)$

where

$\hat{\phi}^*(\theta_i^t) = \frac{1}{n} \sum_{j=1}^{n} [k(\theta_j^t, \theta_i) \nabla_{\theta_i^t} \log p(\theta_i^t|\tilde{X}, \tilde{Y}) + \nabla_{\theta_i^t} k(\theta_i^t, \theta)]$

4 Hybrid algorithm

In this section, we look at the theoretical foundation that connects SVGD and HMC. We then look at how the hybrid algorithm naturally arises from this connection.

4.1 Dynamics of SVGD

[18] shows that as the number of particles becomes large, the SVGD process can be interpreted as a functional gradient descent in a Riemannian metrics space defined for probability distribution. If we take $\epsilon \to 0$, we no longer have a discretized dynamic but a continuous dynamics. We recommend interested reader to consult [18] for a more detailed explanation. In short, the evolution of the approximate distribution q is governed by the following gradient flow:

$$\frac{d \log q}{dt} = -\tilde{\nabla} F(\log q)$$

Where $F(q) = KL(q|p)$ and $\tilde{\nabla} F(\log q)$ represents some form of functional gradient, with respect to $\log q$, induced in some Riemannian metric space that is defined by RKHS-related cost of transforming one distribution to another distribution. Note the $\tilde{\nabla}$ signifies that gradient is not a standard gradient but the natural gradient defined in the appropriate Riemannian metrics.

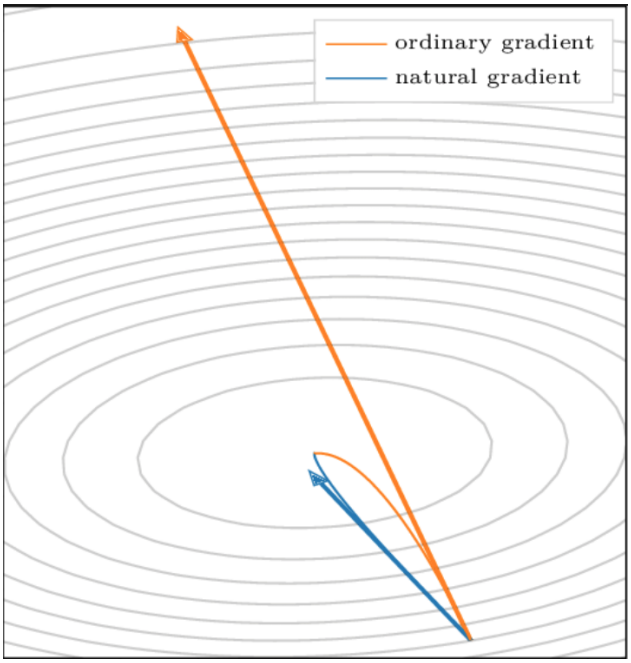

Figure 6: The difference between natural gradient and standard (or ordinary) gradient. The natural gradient takes into account the curvature of the Riemannian space. Gradient optimization method, when using the natural gradient, leads to faster convergence rate.

From a higher level description , SVGD can be treated as an Optimization-Then-Approximate approach for solving the infinite dimensional optimization problem of $\min KL(q||p)$. It is infinite dimension because the variable to optimize for is a function, the distribution q. SVGD first derives an infinite dimensional gradient descent step that is phrased the aforementioned partial differential equation (PDE). And then it uses a finite dimensional particle approximation for the gradient flow (or a numerical solution to the the above PDE). This is reflected as the stein operator and the particle update formula.

4.2 The dynamics of SG-HMC

Recall that the dynamics of stochastic gradient hamiltonian monte carlo is governed by the following system of PDEs, where the stationary distribution is the desired posterior distribution.

$$d\theta = M^{-1}pdt$$

$$dp = -\nabla U(\theta)dt - CM^{-1}pdt + \mathcal{N}(0, 2(C - \hat{B}(\theta))dt + \mathcal{N}(0, 2B(\theta))dt$$

At this point, we could see that there is a strong connection between SG-HMC and SVGD. They can both be interpreted as approximate algorithms that involve solving a system of equations of PDE.

Note that both algorithms have their own weaknesses and strengths. A limitation of HMC is that it some times fails to do substantial exploration. For example, if we would like to sample from amultimodal distribution, one chain of HMC might explore only one mode. Therefore, random restarts are required. More importantly, good initializations are needed for HMC to perform well. However, HMC can be parallelized as each chain of HMC can be run independently. Furthermore, the cost of generating samples simply scales linearly with the number of samples that we want.

SVGD also has its own weaknesses and strengths. Compared to HMC, SVGD requires more computational cost per iteration as it requires computing the derivative of log posterior with respect to each particle as well as computing the kernelized discrepancy between every pair of particles. In other words, the per iteration cost scales quadratically with the number of particles. On the other hand, SVGD is more robust than HMC to bad initialization as the algorithm itself imposes repulsion among the particles so that the particles are spread out more.

4.3 A hybrid algorithm

As SVGD and HMC have very strong mathematical connection, notably both being governed by systems of PDEs. An important ideas from differential equation is that a convex combination of two PDEs will still result in the same stationary distribution. This idea inspires the approach taken in [3] that combines the dynamic of HMC and Langevin Dynamics.

On the other hand, another approach is where the algorithm performs some M number of steps following one dynamic and K steps following the other dynamic. This is known as time sharing. As both dynamical systems have the same stationary distribution, the combined dynamic still has the same stationary distribution. This idea therefore inspires us to explore a third algorithm that combines SVGD and SG-HMC. This algorithm could inherit the advantages of both algorithms and our synthetic experiments show that. Before going into the experimental results, we first outline the

hybrid algorithm as follows:

Algorithm 4.1: HMC-SVGD-hybrid algorithm

Input: Target distribution $p(\theta|X, Y)$ and set of n neural networks sampled from reference distribution $p_0(\theta)$

Output: Set of n neural networks that approximate the posterior distribution

for *while not converged* **do**

For M iterations, perform SVGD update

For K iterations, perform HMC to sample for K neural networks with the current neural networks as initialization.

Note that the above algorithm can be readily extended to the stochastic setting as well by simply replacing HMC with stochastic gradient HMC.

5 Experiments

5.1 Synthetic data

5.1.1 Data generation

The purpose of the synthetic experiment is to focus on a multimodal distribution and to measure the ability of the algorithms of interest to recover these multiple modes. First the data is generated as follows:

```
import numpy as np
N = 100
X = np.linspace(-3, 3, N)
Y = N + np.random.randn(0, 1, size=(N,))
```

5.1.2 Probabilistic model

Note that since we want to do inference on a synthetic distribution with multiple modes, we need to create ancillary variables. We have the following probabilistic model:

Prior:

$$a \sim \mathcal{N}(a|0,1)$$

$$b \sim \mathcal{N}(a|0,1)$$

Likelihood:

$$y|x,a,b \sim \mathcal{N}(y|abx,1)$$

We then train our algorithms to do inference on a and b. The resulting posterior distribution will have two modes. It is notable that the above synthetic experiment can be extended to create posterior distribution with as many modes as we wish.

5.1.3 Measuring performance

An ideal inference algorithm should be able to recover two modes of this distribution. One mode is around $a = 1, b = 1$ and the other is around $a = -1, b = -1$. In fact, the following figure shows the data distribution for 100 points, the

numerical estimation of the log prior, log likelihood and log posterior distribution. Note the two distinct arcs and that an ideal inference distribution should be able to recover both of these arcs. For reference, the last panel in the figure below shows the results from running MCMC with many iterations.

To measure how well our inference algorithms recover the posterior, we used the Jensen-Shannon divergence [17]. The Jensen-Shannon divergence is a symmetrized version of the Kullback-Leibler divergence. We compute the Jensen-Shannon divergence between the kernel density estimator (KDE) of the final set of particles and the numerical estimation of the posterior. The kernel function of the KDE is the radial basis kernel where the length scale is set to $h = 0.02$.

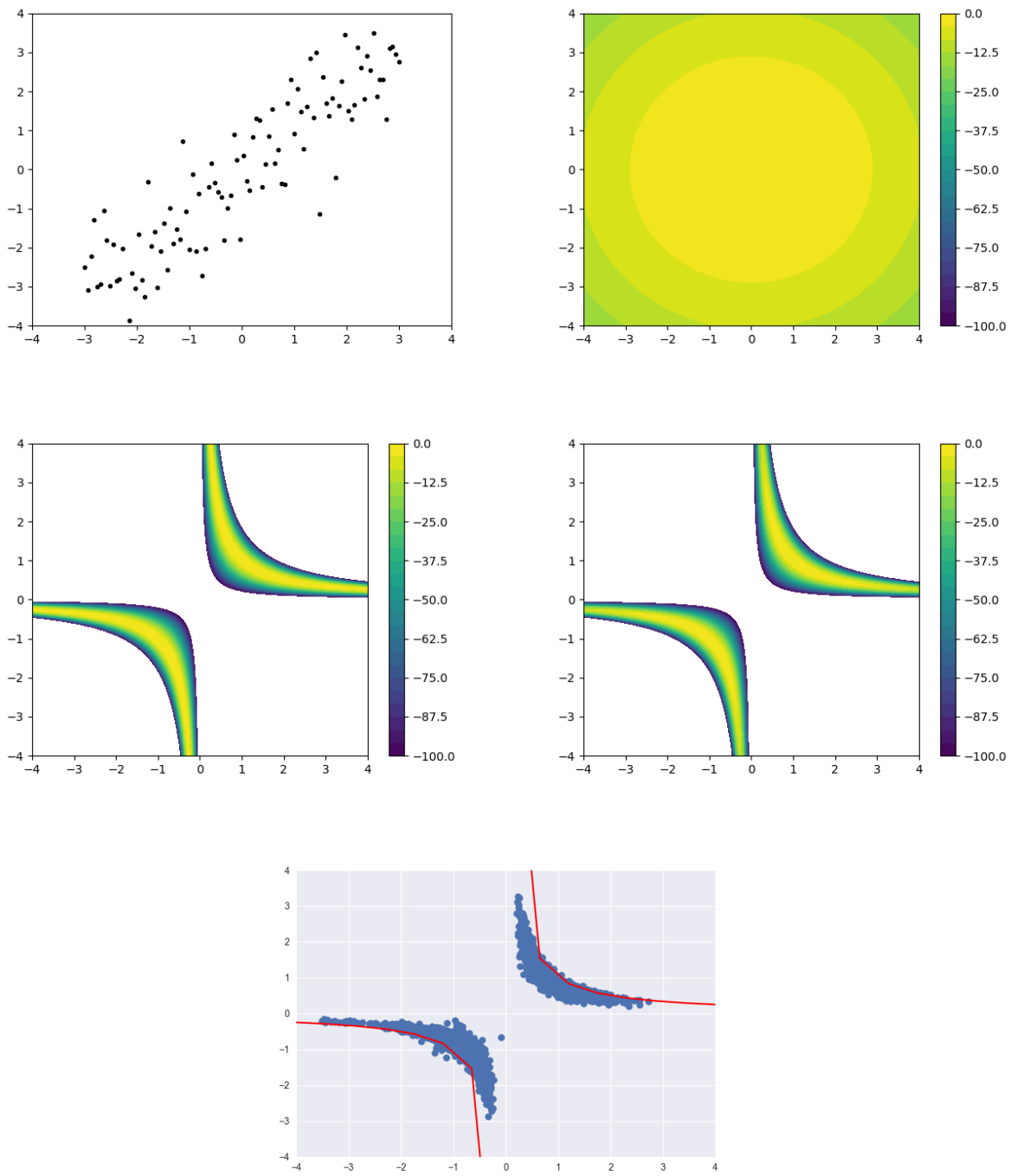

Figure 7: Data distribution, numerical estimation of log prior, log likelihood, log posterior distribution, and MCMC samples

5.1.4 Results

The figure below shows the results of running SVGD with increasing number of particles. From top to bottom and from left to right, we have 1 particle (jsd = 2.9592), 10 particles (jsd = 2.4275), 25 particles (jsd = 1.9161), 50 particles (jsd = 1.5295), 75 particles (jsd = 1.3404) and 200 particles (jsd = 1.0865). An observation not reflected in these static figures is that with increasing number of particles, SVGD also requires more iterations to reach "convergence". We observed the phenomenon in which some particles get stuck in the middle for most of the iterations before slowly moving to one of the two modes. This could be explained by inspecting the update formula of SVGD. The trajectory of one particle is dependent on all other particles. Therefore, some particles in the middle will be influenced by the other particles, sometimes to opposite effect. These particles will therefore remain effectively stationary until all other particles have reached their final position.

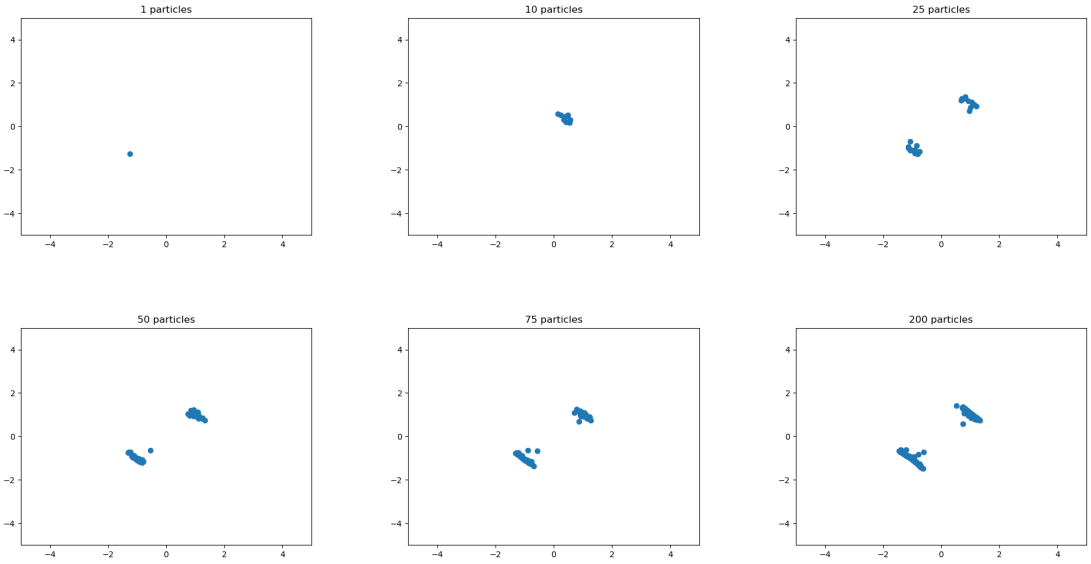

Figure 8: SVGD run with increasing number of particles

The figure below shows the results of running SVGD with increasing length scale for the RBF kernel. From top down and from left to right, we have $h = 0.001$ (jsd = 1.8281), $h = 0.01$ (jsd = 1.5166), $h = 0.1$ (jsd = 1.0225), $h = 1$ (jsd = 1.5968), $h = 10$ (jsd = 2.3245) and $h = 100$ (jsd = 2.3814). Note that the figures are representative of one run while the jsd figure is taken as average over 5 separate runs. The length scale parameter controls the extent to which the repulsive force acts on any pair of particles. Increasing length scale allows distant particles to exert repulsive force on one another but too high length scale value also diminishes this repulsive force.

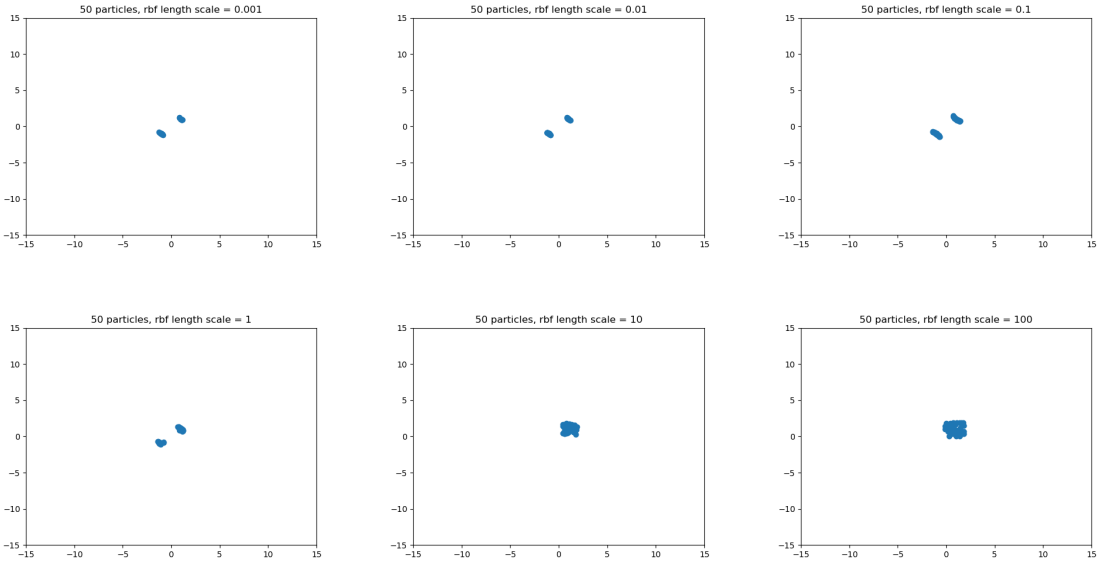

Figure 9: SVGD run with increasing RBF length scale

The following figure presents the results of Hamiltonian Monte Carlo, Stein Variational Gradient Descent and the Hybrid algorithm on the synthetic data set. Each figure below shows a representative visualization of the final distribution of particles. The Jensen-Shannon divergence figure reported is taken as the average of 5 separate runs.

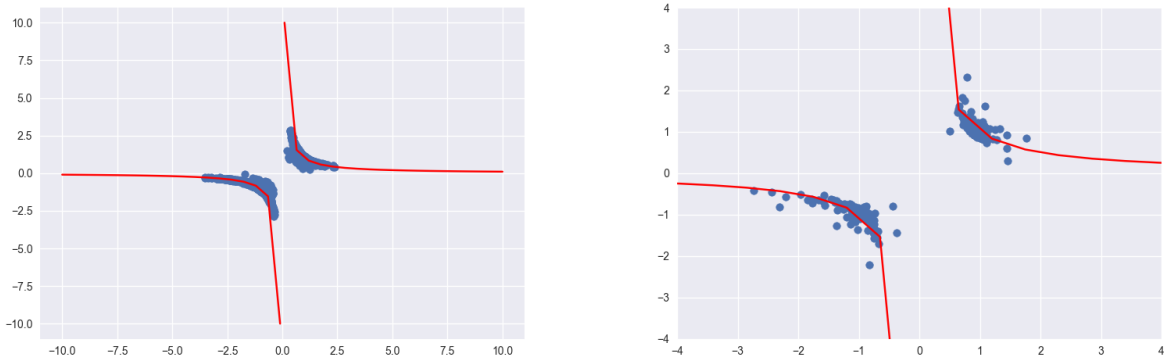

Figure 10: Sample particles from HMC under full batch training (left) and mini-batch training (right). The particles from full batch training achieved Jensen Shannon divergence of 0.253 while those from mini-batch training achieved divergence of 2.0899

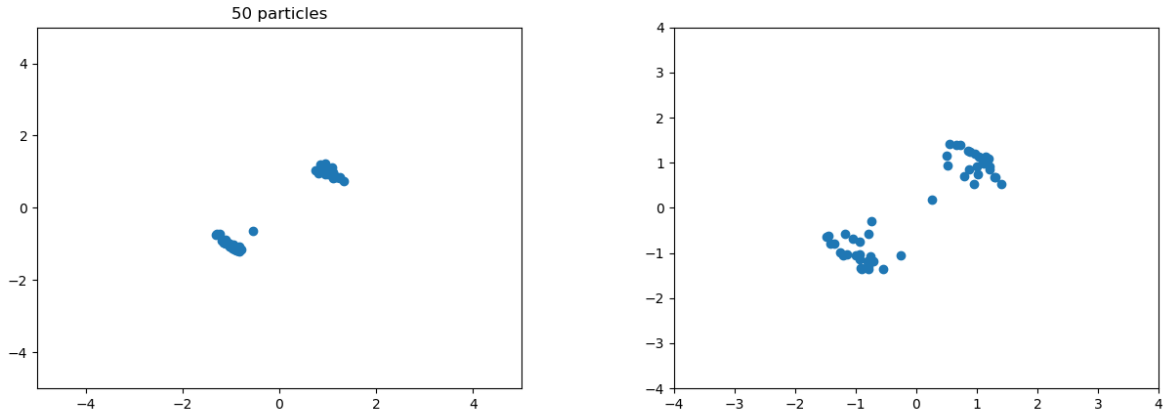

Figure 11: 50 particles from SVGD under full batch training (left) and mini-batch training (right). The particles from full batch training achieved Jensen Shannon divergence of 1.5295 while those from mini-batch training achieved divergence of 1.8182

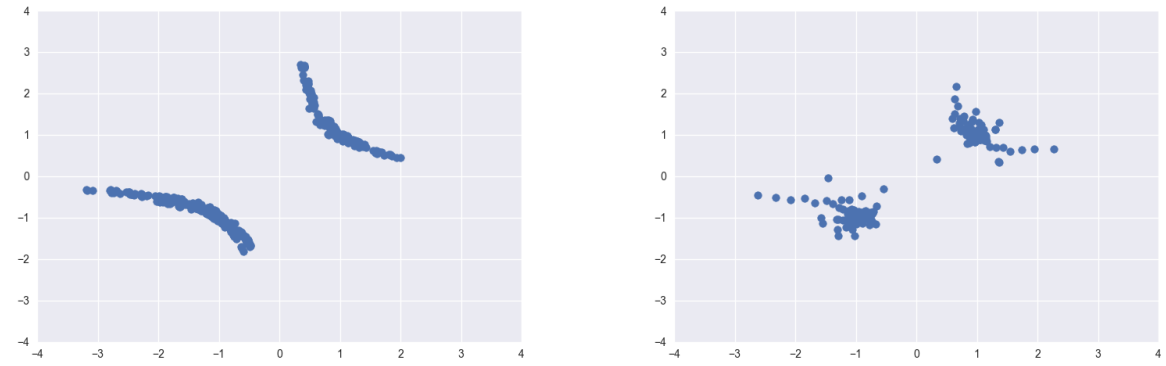

Figure 12: Sampled particles from the Hybrid algorithm under full batch training (left) and mini-batch training (right). The particles from full batch training achieved Jensen Shannon divergence of 0.575 while those from mini-batch training achieved divergence of 2.031

5.1.5 Analysis

The results for our synthetic experiment highlight several key ideas related to the hybrid algorithm:

- The hybrid algorithm achieves results comparable to Hamiltonian Monte Carlo while inheriting the robustness to initialization from SVGD.
- The hyrbrid algorithm also converges faster than standard SVGD (whose convergence is slower the more particles we have).

These results highlight the potential improvements that the hybrid algorithm can bring over both standard HMC and standard SVGD. Further more, it also highlights the possibility of applying the framework of "time sharing" to future particle-based algorithms. Time sharing, the approach taken in the hybrid algorithm can potentially lead to a speed up in convergence, ie obtaining the stationary distribution faster. Intuitively, this could be explained using an imperfect analogy and real life example of mixing honey in a cup of tea. If we stir the honey in a circular fashion, after some time, there will still be droplets of honey remaining. However, if we combine circular stirring with occasionally moving the spoon towards or away from the center, complete mixing is achieved faster. Circular stirring is analogous to following either SVGD or HMC dynamics while the vertical mixing is switching between the two dynamics.

5.2 Real-life data sets

5.2.1 Description

The CIFAR-10 dataset [12] is a dataset of natural images. There are ten mutually disjointed classes: 'plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'. CIFAR-10 consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

5.2.2 Experiment set up

We compared between an ensemble of 10 convolution neural networks trained with SGD against SVGD with 10 particles (neural networks). The neural network architecture that we used is Le Cun convolution neural network [16]. The network has 2 convolution layer and 3 fully connected layers.

5.2.3 Results

The following figure shows the training and test set accuracy when running SVGD with 10 LeCun convolution neural networks. The left figure shows the plot against epochs while the right shows the plot against time in seconds. The final test accuracy is about 66.3%

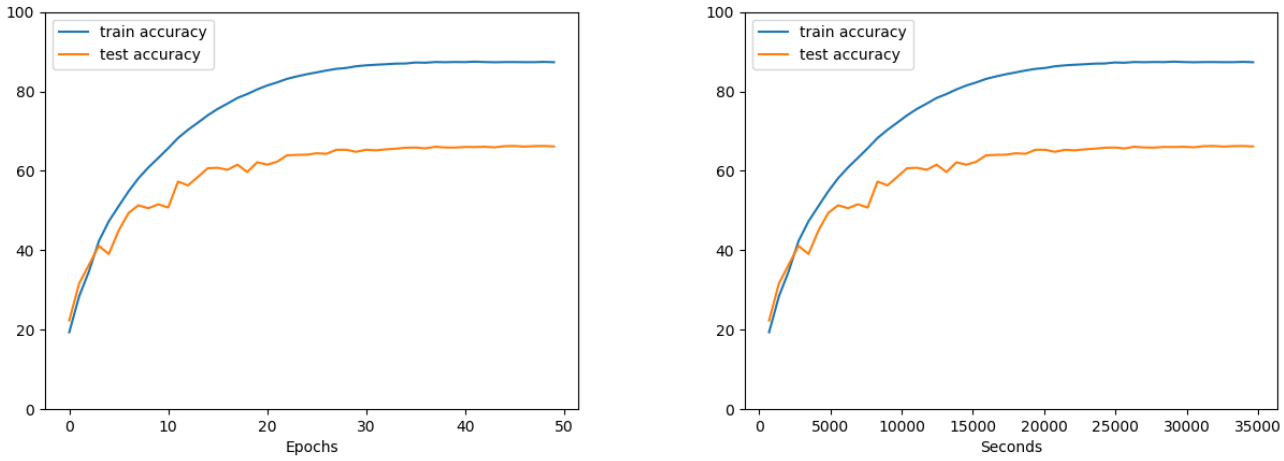

Figure 13: Training and testing accuracies when running SVGD with 10 Convolution neural networks

The following figure shows the training and test set accuracy when running SGD with LeCun convolution neural networks. The left figure shows the plot against epochs while the right shows the plot against time in seconds. The best test accuracy is about 47.8%

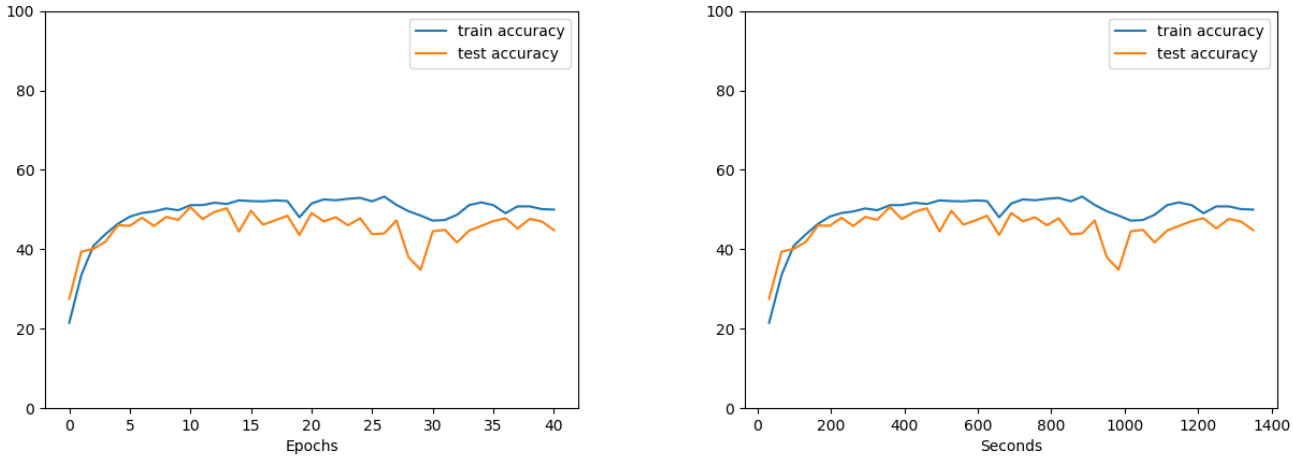

Figure 14: Training and testing accuracies when running SGD on Le Cun's CNN

We then looked at the test set accuracy between an ensemble of 10 CNNs trained with SGD and a system of 10 neural networks trained using SVGD. The ensemble achieved around 62.3% accuracy on test set while SVGD achieved 66.3% accuracy.

6 Conclusion

The conclusion of this thesis is two-fold. Firstly, we have verified that SVGD can be readily extended to deep neural networks, specifically convolution neural networks and achieve better results than comparable ensemble of deep neural networks. Secondly, we have shown that a hybrid algorithm that combines HMC and SVGD via time sharing, is feasible. Furthermore, this hybrid algorithm can combine the strengths of both HMC and SVGD while mitigating some of their weaknesses.

7 Future work

One additional experiment that would nicely complement this thesis is looking at applying the hybrid algorithm to convolution neural networks and test on the CIFAR-10 data set. One weakness of the SVGD algorithm is that the rate of convergence also slows down the more particles we use. On the other hand, the hybrid algorithm, by switching between SVGD and HMC, can achieve substantial speed up.

References

[1] Michael Betancourt. A Conceptual Introduction to Hamiltonian Monte Carlo. *ArXiv e-prints*, page arXiv:1701.02434, January 2017.

[2] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[3] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1683–1691, Bejing, China, 22–24 Jun 2014. PMLR.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[5] D. Donnelly and E. Rogers. Symplectic integrators: An introduction. *American Journal of Physics*, 73:938–945, October 2005.

[6] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *ArXiv e-prints*, page arXiv:1506.02142, June 2015.

[7] Matt Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. *ArXiv e-prints*, page arXiv:1206.7051, June 2012.

[8] Matthew D. Hoffman and David M. Blei. Structured Stochastic Variational Inference. *ArXiv e-prints*, page arXiv:1404.4114, April 2014.

[9] Zhi hua Zhou. Ensemble learning. 2009.

[10] Vishesh Jain, Frederic Koehler, and Elchanan Mossel. The Mean-Field Approximation: Information Inequalities, Algorithms, and Complexity. *ArXiv e-prints*, page arXiv:1802.06126, February 2018.

[11] Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. *ArXiv e-prints*, page arXiv:1312.6114, December 2013.

[12] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). 2014.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[14] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. *ArXiv e-prints*, page arXiv:1612.01474, December 2016.

[15] Steve Lawrence, C. Lee Giles, and Ah Chung Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *AAAI/IAAI*, 1997.

[16] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[17] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, January 1991.

[18] Qiang Liu. Stein variational gradient descent : Theory and applications. 2016.

[19] Qiang Liu and Dilin Wang. Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm. *ArXiv e-prints*, page arXiv:1608.04471, August 2016.

[20] Samuel Livingstone, Michael F. Faulkner, and Gareth O. Roberts. Kinetic energy choice in Hamiltonian/hybrid Monte Carlo. *ArXiv e-prints*, page arXiv:1706.02649, June 2017.

[21] Luca Martino and Victor Elvira. Metropolis Sampling. *ArXiv e-prints*, page arXiv:1704.04629, April 2017.

[22] Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1045–1048. ISCA, 2010.

[23] Vikram Mullachery, Aniruddh Khera, and Amir Husain. Bayesian Neural Networks. *ArXiv e-prints*, page arXiv:1801.07710, January 2018.

[24] Radford M. Neal. MCMC using Hamiltonian dynamics. *ArXiv e-prints*, page arXiv:1206.1901, June 2012.

[25] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *CoRR*, abs/1412.1897, 2014.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[27] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood. *CoRR*, abs/1404.2000, 2014.

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.