Generative Particle Variational Inference via Estimation of Functional Gradients

Neale Ratzlaff [*1] **Qinxun Bai** [*2] **Li Fuxin** [1] **Wei Xu** [2]

Abstract

Recently, particle-based variational inference (ParVI) methods have gained interest because they can avoid arbitrary parametric assumptions that are common in variational inference. However, many ParVI approaches do not allow arbitrary sampling from the posterior, and the few that do allow such sampling suffer from suboptimality. This work proposes a new method for learning to approximately sample from the posterior distribution. We construct a neural sampler that is trained with the functional gradient of the KL-divergence between the empirical sampling distribution and the target distribution, assuming the gradient resides within a reproducing kernel Hilbert space. Our generative ParVI (GPVI) approach maintains the asymptotic performance of ParVI methods while offering the flexibility of a generative sampler. Through carefully constructed experiments, we show that GPVI outperforms previous generative ParVI methods such as amortized SVGD, and is competitive with ParVI as well as gold-standard approaches like Hamiltonian Monte Carlo for fitting both exactly known and intractable target distributions.

1. Introduction

Bayesian inference provides a powerful framework for reasoning and prediction under uncertainty. However, computing the posterior is tractable with only a few parametric distributions, making wider applications of Bayesian inference difficult. Traditionally, MCMC and variational inference methods are utilized to provide tractable approximate inference, but these approaches face their own difficulties if the dimensionality of the space is extremely high. For example, a recent case of interest is Bayesian neural networks (BNNs), which applies Bayesian inference to deep neural network training in order to provide a principled way to assess model uncertainty. The goal in this regime is to model the posterior of every parameter in all the weight tensors from every layer of a deep network. However, developing efficient computational techniques for approximating this intractable posterior with extremely high dimensionality remains challenging.

Recently, particle-based variational inference (ParVI) methods (Liu & Wang, 2016; Liu et al., 2019; Liu, 2017) have been proposed to represent the variational distribution by a set of particles and update them through a deterministic optimization process to approximate the posterior. While achieving both asymptotic accuracy and computational efficiency, ParVI methods are restricted by the fixed number of particles and lack the ability of drawing new samples beyond the initial set of particles. To address this issue, amortized ParVI methods (Wang & Liu, 2016) have been proposed to amortize the ParVI gradients in training a neural sampler. While being flexible in drawing samples, in Sec. 4 we show that amortized ParVI methods cannot match the convergence behavior of ParVI methods.

In this work, we propose a generative particle variational inference (GPVI) approach that addresses those issues. GPVI trains a neural sampler network by directly estimating the functional gradient w.r.t. the KL-divergence between the distribution of generated particles and the target distribution, and pulls it back to update the neural sampler. As such it allows the neural sampler to directly generate particles that match the posterior distribution, hence achieving the asymptotic accuracy and computational efficiency of ParVI methods. In figure 1, we show that the predictive distribution of 1D regression functions sampled from GPVI nearly matches that from the ParVI solution, while amortized ParVI fails.

The main computational challenge lies in a reliable estimate of the functional gradient that involves the inverse of the input-output Jacobian of the neural sampler. Instead of directly computing this term and paying a high computational cost, we introduce a helper network to estimate the inverse Jacobian vector product and train the helper network via gradient descent. By alternating between this gradient step and the gradient update of the sampler network, the computational cost is distributed over the whole train-

[*]Equal contribution [1]Department of Electrical Engineering and Computer Science, Oregon State University, Corvallis, Oregon [2]Horizon Robotics, Cupertino, California. Correspondence to: Neale Ratzlaff <ratzlafn@oregonstate.edu>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

ing procedure.

In experiments, our proposed approach achieves comparable convergence performance as ParVI methods. It is considerably superior than that of amortized ParVI methods, while still allowing efficient sampling from the posterior. By directly applying our approach as a hypernetwork to generate BNNs, we achieve competitive performance regarding uncertainty estimation.

In summary, our contributions are three-fold,

- We propose GPVI, a new variational inference approach that trains a neural sampler to generate particles from any posterior distribution. GPVI estimates the functional gradient and uses it to update the neural sampler. It enjoys the asymptotic accuracy and computational efficiency of ParVI methods. Comparing with existing amortized ParVI methods, our approach enjoys the same efficiency and flexibility while showing considerable advantage in convergence behavior.
- We design careful techniques for efficient gradient estimates that address the challenges in approximating the product between the inverse of the Jacobian and a vector.
- We apply our approach to BNNs and achieve competitive uncertainty estimation quality for deep neural networks.

2. Related Work

ParVI is a recent class of non-parametric variational inference methods. (Liu & Wang, 2016; Liu, 2017) proposed stein variational gradient descent (SVGD), which deterministically updates an empirical distribution of particles toward the target distribution via a series of sequentially constructed smooth maps. Each map defines a perturbation of the particles in the direction of steepest descent towards the target distribution under the KL-divergence metric. Liu et al. (2019) cast ParVI as a gradient flow on the space of probability measures $\mathcal{P}_2$ equipped with the W_2 Wasserstein metric, and proposed ParVI methods GFSD and GFSF by smoothing the density and the function respectively. A potential limitation of SVGD and GFSF is the restriction of the transport maps to an RKHS. The fisher neural sampler (Hu et al., 2018; Grathwohl et al., 2020) lifts the function space of transportation maps to the space of $\mathcal{L}_2$ functions, and performs a minimax optimization procedure to find the optimal perturbation. As a non-parametric approach, ParVI does not depend on the specific parameterization of the approximate posterior. ParVI is also more efficient than MCMC methods due to its deterministic updates and gradient-based optimization.

The limitation of ParVI is the inability to draw additional samples beyond the initial set. To address this limitation, our approach allows for arbitrary draws of new samples once the generator is trained. The most notable generative ParVI approach prior to our work was amortized SVGD (Wang & Liu, 2016), which proposed using a generative model to approximately sample from the SVGD particle distribution. Amortized SVGD performs a one-step update that backpropagates the SVGD gradient back to the generator parameters, which is often not optimal. A more detailed discussion of the differences between our method and GPVI is presented in Sec. 3.5.

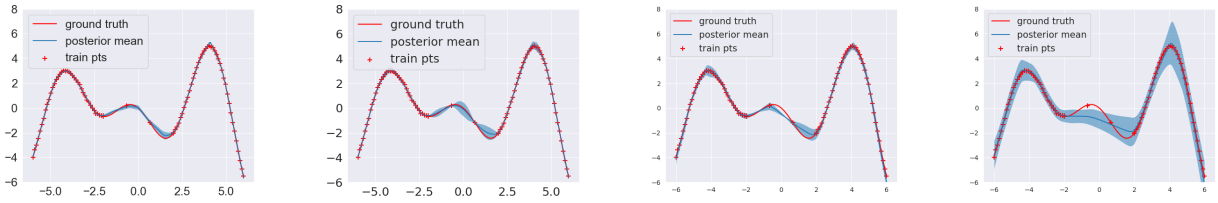
Performing direct Bayesian inference in the weight-space of neural networks (Neal, 2012; Buntine & Weigend, 1991; MacKay, 1992) is known to be intractable in the context of deep learning. Thus, approximations have been formulated using Langevin Dynamics (Welling & Teh, 2011; Korattikara Balan et al., 2015), Monte Carlo dropout (Gal & Ghahramani, 2016; Kingma et al., 2015), and variational inference with mean-field Gaussian priors (Graves, 2011). The variational approximation has been iterated upon extensively. Bayes by Backprop (Blundell et al., 2015) proposed an unbiased gradient estimator that allowed for training of deep Bayesian neural networks using the local reparameterization trick from Kingma et al. (2015). However, fully factorized Gaussian approximations are insufficient to capture the complex structure of a high dimensional posterior (Yao et al., 2019). As a result, structured approximate posteriors have been proposed using matrix Gaussians (Louizos & Welling, 2016), normalizing flows (Louizos & Welling, 2017; Krueger et al., 2017), and hypernetworks (Pawlowski et al., 2017).

Evaluating the performance of BNNs is difficult, as there is often no access to the ground truth posterior. Therefore, prior work often considers a "gold-standard" method such as HMC (Neal et al., 2011), or exact Gaussian processes (Rasmussen, 2003) as a proxy for the true posterior. While these methods set a high bar for BNNs, they are in general not scalable to large-scale problems. Nonetheless there were attempts to bring HMC (Strathmann et al., 2015; Chen et al., 2014) and GPs (Wilson et al., 2016; Cheng & Boots, 2017; Wang et al., 2019; Hensman et al., 2013; 2015) in the realm of tractability for high dimensional problems.

Variational inference via smooth transport maps is also used in normalizing flows (Rezende & Mohamed, 2015). But the transport maps used in normalizing flows and in Marzouk et al. (2016) are bijections which makes them unsuitable for learning distributions over neural network functions.

3. Generative Particle Variational Inference

We want to learn a parametric generator: $\boldsymbol{f_\theta} : \mathcal{Z} \to \mathbb{R}^d$, parameterized by $\boldsymbol{\theta}$, where $\mathcal{Z} \subset \mathbb{R}^d$ is the convex space of



(a) HMC (b) SVGD (c) GPVI (d) Amortized SVGD

Figure 1. Predictive uncertainty of methods for a 1-D regression task. (a) HMC predictive posterior matches the uncertainty in the data; (b) SVGD performs comparably to HMC; (c) Our proposed GPVI performs similarly to SVGD, with the additional capability of sampling new particles during inference; (d) Amortized SVGD overestimates the uncertainty when the data is sparse.

input noise and $\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})$ generates a sample $\boldsymbol{x}$ from an input noise $\boldsymbol{z}$. Let $q(\boldsymbol{x})$ represents the implicit distribution of samples generated by $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})$, where $\boldsymbol{z} \sim N(\boldsymbol{0}, I_d)$. Let $p(\boldsymbol{x})$ be the target distribution, we want to solve for $\boldsymbol{f}_{\boldsymbol{\theta}}$ that minimizes the objective $\mathrm{KL}\left(q(\boldsymbol{x}) \| p(\boldsymbol{x})\right)$.

Our approach treats this problem from a functional optimization perspective, by first computing the functional gradient of the objective, and then pulling it back to parameter space through the function parameterization. In the rest of this section we introduce our algorithm in detail and compare it with amortized ParVI approaches.

3.1. Functional gradient and its pullback

Let $\mathcal{J}(\boldsymbol{f}) = \mathrm{KL}\left(q(\boldsymbol{x}) \| p(\boldsymbol{x})\right)$ be the objective, where $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$. If $\boldsymbol{f}$ is injective, by change of variables for probability measure,

$$q(\boldsymbol{x}) = \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|}, \tag{1}$$

where $p_{\boldsymbol{z}}(\boldsymbol{z})$ is the distribution from which $\boldsymbol{z}$ is sampled. The minimization objective becomes,

$$\mathcal{J}(\boldsymbol{f}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|}\right]. \tag{2}$$

Consider some function approximation of $\boldsymbol{f}$, say $\boldsymbol{f} = \boldsymbol{f}_{\boldsymbol{\theta}}$, then the minimization objective becomes,

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}}{\partial \boldsymbol{z}}\right)\right|}\right]. \tag{3}$$

Directly computing the gradient of $\mathcal{J}(\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ involves not only an inverse of the Jacobian $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}$, but also second derivatives of $\boldsymbol{f}_{\boldsymbol{\theta}}$ (see Appendix A for details), which is overly expensive to compute in practice.

We propose, instead, to first compute the functional gradient of (2) w.r.t. $\boldsymbol{f}$, i.e., $\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})$, and then back-propagate it through the generator to get the gradient w.r.t. $\boldsymbol{\theta}$, i.e.,

$$\nabla_{\boldsymbol{\theta}} \mathcal{J} = \mathbf{E}_{\boldsymbol{z}}\left[\frac{\partial \boldsymbol{f}(\boldsymbol{z})}{\partial \boldsymbol{\theta}} \nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})(\boldsymbol{z})\right]. \tag{4}$$

The following theorem gives an explicit formula for computing the functional gradient $\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})$ when $\boldsymbol{f}$ is chosen from a Reproducing kernel Hilbert space (RKHS). The proof is provided in Appendix A.

Theorem 3.1. *Let $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$, where $\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})$, vector function $\boldsymbol{f} = (f^1, \ldots, f^d) \in \mathcal{H}^d$ with $f^i \in \mathcal{H}$, where $\mathcal{H}$ is the RKHS with kernel $k(\cdot, \cdot)$, $\mathcal{H}^d$ is equipped with inner product $\langle \boldsymbol{f}, \boldsymbol{g} \rangle_{\mathcal{H}^d} = \sum_{i=1}^d \langle f^i, g^i \rangle_{\mathcal{H}}$. For $\mathcal{J}(\boldsymbol{f})$ well-defined by (2), we have*

$$\begin{aligned}\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'}\Bigg[&-\nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')} k(\boldsymbol{z}', \boldsymbol{z}) \\ &- \left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})\Bigg].\end{aligned} \tag{5}$$

3.2. Reparameterization of $\boldsymbol{f}$

There are two considerations for reparameterizing $\boldsymbol{f}$. Firstly, in order for the inverse $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1}$ in (5) to be well-defined, the Jacobian $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}$ should be a square matrix, i.e., input noise $\boldsymbol{z}$ should have the same dimension as the output $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$. In practice, especially those applications involving BNNs, each $\boldsymbol{x}$ represents parameters of a sampled neural network and therefore can be extremely high dimensional. As a result, a high dimensional $\boldsymbol{f}$ can be computationally prohibitive. Secondly, in order for the change of variables density formula (1) to hold, $\boldsymbol{f}$ needs to be injective, which is in general not guaranteed for an arbitrary neural network function.

To overcome the above two concerns, we consider the following parameterization,

$$\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}) = \boldsymbol{g}_{\boldsymbol{\theta}}\left(\boldsymbol{z}^{(:k)}\right) + \lambda \boldsymbol{z}, \quad \forall \boldsymbol{z} \in \mathbb{R}^d, \tag{6}$$

where $\boldsymbol{z}^{(:k)} \in \mathbb{R}^k$ denotes the vector consisting of the first k components of $\boldsymbol{z}$, and $\boldsymbol{g}_{\boldsymbol{\theta}} : \mathbb{R}^k \to \mathbb{R}^d$ with parameters

$\boldsymbol{\theta}$ is a much slimmer neural network. In our experiments, $\boldsymbol{g_\theta}$ is designed with an input dimension k less than 30% the size of d. For high dimensional open-category experiments where $d > 60,000$, we use a k of less than 2% of d. For $\boldsymbol{f_\theta}$ defined by (6)), the Jacobian is,

$$\left[\frac{\partial \boldsymbol{f_\theta}}{\partial \boldsymbol{z}}\right]_{d\times d} = \left[\left[\frac{\partial \boldsymbol{g_\theta}}{\partial \boldsymbol{z}^{(:k)}}\right]_{d\times k} \middle| \mathbf{0}_{d\times(d-k)}\right]_{d\times d} + \lambda \boldsymbol{I}_d, \quad (7)$$

where λ is a hyper-parameter. Note that for sufficiently large λ, the Jacobian defined by (7) is positive definite. Since the domain $\mathcal{Z}$ of $\boldsymbol{z}$ is convex, it is straightforward to show that $\boldsymbol{f_\theta}$ defined by (6) is injective. We include the proof in Appendix A.

In practice we set λ to be 1.0 and find it sufficient throughout our experiments.

3.3. Estimating the Jacobian inverse

The main computational challenge of (5) lies in computing the term

$$\left(J_{\boldsymbol{f}}(\boldsymbol{z}')\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}), \quad (8)$$

where $J_{\boldsymbol{f}}(\boldsymbol{z}') = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}$, especially considering that we need an efficient implementation for batched $\boldsymbol{z}$ and $\boldsymbol{z}'$.

Directly evaluating and storing the full Jacobian $J_{\boldsymbol{f}}(\boldsymbol{z}')$ for each $\boldsymbol{z}'$ of the sampled batch is not acceptable from the standpoint of either time or memory consumption. There exist iterative methods for solving the linear equation system $\boldsymbol{y} = \left(J_{\boldsymbol{f}}(\boldsymbol{z}')\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ (Young, 1954; Fletcher, 1976), which involves computing the vector-Jacobian product $J_{\boldsymbol{f}}(\boldsymbol{z}')\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ at each iteration. By alternating between this iterative solver and the gradient update (4), it is possible to get a computationally amenable algorithm. However, as shown in the Appendix C, such an algorithm does not converge to the target distribution p even for a simple Bayesian linear regression task. This is due to the fact that batches of both $\boldsymbol{z}$ and $\boldsymbol{z}'$ for evaluating (5) need to be re-sampled for each gradient update (4) to avoid the cumulative sampling error. Therefore, the above alternating procedure of iterative solver for $\boldsymbol{y} = \left(J_{\boldsymbol{f}}(\boldsymbol{z}')\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ ends up shooting a moving target for different batches of $\boldsymbol{z}$ and $\boldsymbol{z}'$ at each iterate, which is difficult for convergence.

To overcome this computational challenge, we propose a helper network, denoted by $\boldsymbol{h_\eta}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k)$, and parameterized by $\boldsymbol{\eta}$, that consumes both $\boldsymbol{z}'$ and $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$ and predicts $\left(J_{\boldsymbol{f}}(\boldsymbol{z}')\right)^{-1} \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$. With the helper network, the functional gradient (5) can be computed by,

$$\begin{aligned}\nabla_{\boldsymbol{f}} \mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'}\Big[&- \nabla_{\boldsymbol{x}} \log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')} k(\boldsymbol{z}', \boldsymbol{z}) \\ &- \boldsymbol{h_\eta}\left(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})\right)\Big].\end{aligned} \quad (9)$$

We use the following loss to train the helper network,

$$\mathcal{L}(\boldsymbol{\eta}) = \|J_{\boldsymbol{f}}(\boldsymbol{z}')\boldsymbol{h_\eta}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})) - \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})\|^2, \quad (10)$$

where $J_{\boldsymbol{f}}(\boldsymbol{z}')\boldsymbol{h_\eta}$ can be computed by the following formula given the reparameterization (6) of $\boldsymbol{f_\theta}$,

$$\boldsymbol{h}_{\boldsymbol{\eta}}^T\left(\frac{\partial \boldsymbol{f_\theta}}{\partial \boldsymbol{z}'}\right) = \boldsymbol{h}_{\boldsymbol{\eta}}^T\left(\frac{\partial \boldsymbol{g_\theta}}{\partial \boldsymbol{z}'^{(:k)}}\right) + \lambda \boldsymbol{h}_{\boldsymbol{\eta}}^T, \quad (11)$$

where the Vector-Jacobian Product (VJP) $\boldsymbol{h}_{\boldsymbol{\eta}}^T\left(\frac{\partial \boldsymbol{g_\theta}}{\partial \boldsymbol{z}'^{(:k)}}\right)$ can be computed by one backward pass of the function $\boldsymbol{g_\theta}$.

3.4. Summary of the algorithm

Both $\boldsymbol{g_\theta}$ and $\boldsymbol{h_\eta}$ can be trained with stochastic gradient descent (SGD), and our algorithm alternates between the SGD updates of $\boldsymbol{g_\theta}$ and $\boldsymbol{h_\eta}$. Note that the helper network $\boldsymbol{h_\eta}$ only has to chase the update of $\boldsymbol{g_\theta}$, but no more extra moving targets due to re-sampling of $\boldsymbol{z}$ and $\boldsymbol{z}'$. Our experiments in Sec. 4 and the appendix show that the helper network is able to efficiently approximate (8) which enables the convergence of the gradient update (4). Our overall Generative Particle VI (GPVI) algorithm is summarized in Algorithm 1.

Algorithm 1 Generative Particle VI (GPVI)

Initialize generator $\boldsymbol{g_\theta}$, helper $\boldsymbol{h}_\eta$, and learning rate ϵ
while Not converge **do**
1. sample two batches $\{\boldsymbol{z}_i\}, \{\boldsymbol{z}'_i\} \sim N(\mathbf{0}, I_d)$
2. compute $k(\boldsymbol{z}', \boldsymbol{z})$ and $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$
3. forward $\boldsymbol{g_\theta}$ to compute $\boldsymbol{f_\theta}(\boldsymbol{z})$ and $\boldsymbol{f_\theta}(\boldsymbol{z}')$ by (6)
4. forward $\boldsymbol{h_\eta}$ to compute $\boldsymbol{h_\eta}(\boldsymbol{z}', \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z}))$
5. backward $\boldsymbol{g_\theta}$ to compute the VJP $\boldsymbol{h}_{\boldsymbol{\eta}}^T\left(\frac{\partial \boldsymbol{g_\theta}}{\partial \boldsymbol{z}'^{(:k)}}\right)$ and then construct $J_{\boldsymbol{f}}(\boldsymbol{z}')\boldsymbol{h_\eta}$ by (11),
6. update $\boldsymbol{h_\eta}$ by $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} - \epsilon\nabla_{\boldsymbol{\eta}}\mathcal{L}$, where $\nabla_{\boldsymbol{\eta}}\mathcal{L}$ is computed by back-propagating (10)
7. compute the functional gradient by (9)
8. update $\boldsymbol{\theta}$ by $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon\nabla_{\boldsymbol{\theta}}\mathcal{J}$, where $\nabla_{\boldsymbol{\theta}}\mathcal{J}$ is computed by (4)

end

3.5. Comparison with Amortized SVGD

Stein variational gradient descent (SVGD) represents $q(\boldsymbol{x})$ by a set of particles $\{\boldsymbol{x}_i\}_{i=1}^n$, which are updated iteratively by,

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \epsilon\boldsymbol{\phi}^*(\boldsymbol{x_i}), \quad (12)$$

where ϵ is a step size and $\boldsymbol{\phi}^* : \mathbb{R}^d \to \mathbb{R}^d$ is a vector field (perturbation) on the space of particles that corresponds to the optimal direction to perturb particles, i.e.,

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}\in\mathcal{F}}{\arg\min}\left\{\frac{d}{d\epsilon}\mathrm{KL}(q_{[\epsilon\boldsymbol{\phi}]}(\boldsymbol{x})\|p(\boldsymbol{x}))\Big|_{\epsilon=0}\right\},$$

where $q_{[\epsilon\boldsymbol{\phi}]}(\boldsymbol{x})$ denotes the density of particles updated by (12) using the perturbation $\boldsymbol{\phi}$, where the density of original particles is $q(x)$. When $\mathcal{F}$ is chosen to be the unit ball of some RKHS $\mathcal{H}$ with kernel function $k(\cdot,\cdot)$, SVGD gives the following closed form solution for $\boldsymbol{\phi}^*$,

$$\boldsymbol{\phi}^*(\boldsymbol{x}) = \mathbf{E}_{\boldsymbol{x}'\sim q}\left[\nabla_{\boldsymbol{x}'}\log p(\boldsymbol{x}')k(\boldsymbol{x}',\boldsymbol{x}) + \nabla_{\boldsymbol{x}'}k(\boldsymbol{x}',\boldsymbol{x})\right]. \tag{13}$$

To turn SVGD into a neural sampler, Amortized SVGD (Wang & Liu, 2016) first samples particles from a generator and then back-propagates the particle gradients (13) through the generator to update the generator parameters. Let $\boldsymbol{x} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})$, $\boldsymbol{z}\sim N(\mathbf{0}, I_d)$ be the particle generating process, where $\boldsymbol{f}_{\boldsymbol{\theta}}$ is the generator parameterized by $\boldsymbol{\theta}$, amortized SVGD updates $\boldsymbol{\theta}$ by,

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \epsilon\sum_{i=1}^{m}\frac{\partial \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_i)}{\partial\boldsymbol{\theta}}\boldsymbol{\phi}^*(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_i)), \tag{14}$$

where $\boldsymbol{\phi}^*(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}_i))$ is computed by (13).

The following Lemma gives an explicit view regarding what functional gradient amortized SVGD back-propagates through the generator. The proof is given in the Appendix A.

Lemma 3.2. *If particles are generated by $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{z})$, $\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})$, Eqn.* (13) *is the functional gradient of the KL objective w.r.t. the perturbation function $\boldsymbol{\phi}:\mathbb{R}^d\to\mathbb{R}^d$ applied on the output space of $\boldsymbol{f}$, i.e.,*

$$\nabla_{\boldsymbol{\phi}}\Big|_{\boldsymbol{\phi}=id} KL\left(q(\boldsymbol{x})\|p(\boldsymbol{x})\right) = -\boldsymbol{\phi}^*, \tag{15}$$

where $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{f}(\boldsymbol{z}))$ and $\boldsymbol{\phi} = (\phi^1,\ldots,\phi^d)\in\mathcal{H}^d$ with $\phi^i\in\mathcal{H}$, where $\mathcal{H}$ is the RKHS with kernel $k(\cdot,\cdot)$, $\mathcal{H}^d$ is equipped with inner product $\langle\boldsymbol{\phi},\boldsymbol{\xi}\rangle_{\mathcal{H}^d} = \sum_{i=1}^d\langle\phi^i,\xi^i\rangle_{\mathcal{H}}$.

To see the difference between our GPVI and amortized SVGD, both methods learn a generator $\boldsymbol{f}$ that generates particles $\boldsymbol{x}$ from input noise $\boldsymbol{z}$. To update $\boldsymbol{f}$, gradients of the KL objective $\mathcal{J}(\boldsymbol{f})$ have to be backpropagated to $\boldsymbol{f}$ in both cases. Here, the best is to use the steepest descent direction for $\boldsymbol{f}$, which GPVI does (proved in Thm 3.1). As Lemma 3.2 shows, the SVGD gradient $\boldsymbol{\phi}^*$ given by (13) is optimal for a transformation $\boldsymbol{\phi}$ applied to the set of current particles. In in both amortized SVGD and GPVI, such a set is not kept and $\boldsymbol{\phi}$ does not exist since they only maintain and update $\boldsymbol{f}$. But amortized SVGD directly uses $\boldsymbol{\phi}^*(\boldsymbol{f})$ in (14) as the direction to update $\boldsymbol{f}$, which is an unproven use – it only coincides with the steepest descent direction for $\boldsymbol{f}$ used by GPVI in special cases such as $\boldsymbol{f} = \text{id}$, where (5) and (13) are equivalent because $\boldsymbol{z} = \boldsymbol{x}$ and the Jacobian is identity. Regarding the definition of RKHS, GPVI's RKHS approximates the tangent space of $\boldsymbol{f}$, it is therefore naturally defined on $\boldsymbol{z}$. (Amortized) SVGD uses RKHS to approximate the tangent space of $\boldsymbol{\phi}$, which is naturally defined on $\boldsymbol{x}$.

The amortizing step applied in both methods, i.e., back-propagating some functional gradient to update the generator parameters, is in general not guaranteed to keep the original descent direction of the functional gradient. However, GPVI amortizes the steepest descent direction which is optimal in first order sense, while amortized SVGD amortizes a non-steepest descent direction, which is more likely to result in a non-descending direction after amortizing. In practice, as shown in the next section, our approach consistently outperforms amortized SVGD in approximating the target distribution and capturing model uncertainty.

4. Experiments

To demonstrate the effectiveness of our approach for approximate Bayesian inference, we evaluated GPVI in two different settings: density estimation and BNNs. In our density estimation experiment, we trained our generator to draw samples from a target distribution, showing that GPVI can accurately fit the posterior from the data. For our BNN experiments, we evaluated on regression, classification, and high dimensional open-category tasks. Our experiments show that among all methods compared, GPVI is the only method to excel in both sampling efficiency and asymptotic performance. We compare GPVI with ParVI methods: SVGD (Liu & Wang, 2016), GFSF (Liu et al., 2019), and KSD (Hu et al., 2018; Grathwohl et al., 2020), as well as their corresponding amortized versions. We also compared with a mean-field VI approach Bayes by Backprop (MF-VI) (Blundell et al., 2015), as well as deep ensembles (Lakshminarayanan et al., 2017), and HMC (Neal et al., 2011). We emphasize that our aim is not to only maximize the likelihood of generated samples, rather we want to closely approximate the posterior of parameterized functions given the data. Thus, predictions w.r.t data unseen during training should reflect the epistemic uncertainty of the model.

The details of our experimental setup are as follows. In our BNN experiments, we parameterized samples from the target distribution as neural networks with a fixed architecture. In the low dimensional regression and classification settings, we drew 100 samples from the approximate posterior for both training and evaluation, allowing us to evaluate the predictive mean and variance. For methods without a sampler e.g. ParVI and deep ensembles, we initialized a 100 member ensemble. In the high dimensional open-category tasks, we instead used 10 samples due to the larger computation cost. For methods that utilize a hypernetwork such as GPVI and amortized ParVI, we used Gaussian input noise $z\sim\mathcal{N}(0,I)$ and varied the hypernetwork architecture depending on the task. For ParVI and deep

Method	Σ error (2d) $\downarrow$	Σ error (5d) $\downarrow$
Amortized SVGD	$0.10 \pm .09$	$0.37 \pm .32$
Amortized GFSF	$0.18 \pm .04$	$0.21 \pm .13$
Amortized KSD	$0.28 \pm .32$	$1.68 \pm .52$
GPVI	$0.14 \pm .08$	$0.14 \pm .04$

Table 1. Comparison of generative Particle VI approaches for density estimation of 2d and 5d Gaussian distributions.

ensembles we randomly initialized each member of the ensemble. In all methods except HMC we used the Adam optimizer (Kingma & Ba, 2014). A detailed description of network architectures, chosen hyperparameters, and experimental settings is given in the appendix.

4.1. Density Estimation

We first evaluate the ability of generative approaches to fit a target distribution from data. We used 2D and 5D zero mean Gaussian distributions with non-diagonal covariances as our target distributions. We consider sampler networks with one hidden layer of width 2 and 5, respectively, and input noise with the same dimensionality as the output. The error is defined as the difference between the true variance and the variance of the sampler output distribution $WW^T - \Sigma$, where W is the weights of the sampler. As shown in Table 1, while the estimation problem becomes harder with increased dimension, our approach performs consistently well while amortized ParVI methods suffer more or less from a performance drop and all end up inferior to our approach. We have included more density estimation results in the Appendix.

4.2. Bayesian Linear Regression

We evaluated all methods on Bayesian linear regression to investigate how well each method can fit a unimodal normal distribution over linear function weights. The target function is a linear regressor with parameter vector $\boldsymbol{\beta} \in \mathbb{R}^d$, i.e., $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \epsilon$, where $\boldsymbol{X} \sim \mathcal{N}(0, I_d)$, $\epsilon \sim \mathcal{N}(0, I)$, and $\beta^i \sim U(0,1) + 5$. We chose such linear Gaussian settings where we can explicitly compute the target posterior $p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y}) \sim \mathcal{N}((\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}, \boldsymbol{X}^T\boldsymbol{X})$ given observations $(\boldsymbol{X}, \boldsymbol{y})$, allowing us to numerically evaluate how well each method fits the target distribution. Each method was trained to regress $\boldsymbol{y}$ from $\boldsymbol{X}$. For GPVI and amortized ParVI methods that make use of a hypernetwork, we used a linear generator with input noise $\boldsymbol{z} \in \mathbb{R}^d$. In this setting, the output of a generator with bias $\boldsymbol{b}$ and weights W follows the distribution $\mathcal{N}(\boldsymbol{b}, WW^T)$, which should match the posterior $p(\boldsymbol{\beta}|\boldsymbol{X}, \boldsymbol{y})$. For ParVI methods and deep ensembles, we initialized an ensemble of linear regressors with d parameters. We computed the mean and covariance of the learned parameters to measure the quality of posterior fit. For MF-VI we chose a standard normal prior on the weights, and likewise computed the mean and covariance of weight samples to measure posterior fit. We also test a variant of GPVI where we exactly compute the Jacobian inverse instead of using our helper network. We set $d = 3$ in our experiments. While simple, this quantitative sanity check is crucial before evaluating on more complicated domains where the true posterior is not available. It will be evident in later experiments that the ability to closely approximate the true posterior under this simple setup is indicative of performance on higher dimensional tasks.

Method	μ error $\downarrow$	Σ error $\downarrow$
SVGD	$0.006 \pm .0024$	$\mathbf{0.125 \pm .03}$
GFSF	$0.003 \pm .0014$	$0.139 \pm .07$
KSD	$0.009 \pm .0006$	$0.373 \pm .11$
Amortized SVGD	$\mathbf{0.002 \pm .0009}$	$0.158 \pm .04$
Amortized GFSF	$\mathbf{0.002 \pm .0003}$	$0.209 \pm .05$
Amortized KSD	$0.004 \pm .0007$	$0.430 \pm .10$
MF-VI	$0.004 \pm .0035$	$0.303 \pm .04$
Deep Ensemble	$0.004 \pm .0002$	1.0 ± 0
HMC	$0.009 \pm .0003$	$0.181 \pm .05$
GPVI	$\mathbf{0.002 \pm .0007}$	$\mathbf{0.128 \pm .04}$
GPVI Exact Jac	$\mathbf{0.002 \pm .0004}$	$\mathbf{0.106 \pm .07}$

Table 2. Bayesian linear regression. Reported error is the L_2 norm of the difference between the learned mean and covariance parameters, and the ground truth after 50000 iterations.

Results are shown in table 2. Our GPVI outperforms amortized ParVI, MF-VI, and surprisingly even HMC. GPVI is also very competitive with the best ParVI methods, SVGD and GFSF. We note that GPVI is also competitive with the "Exact-Jac" variant, where we invert the Jacobian explicitly, instead of using the helper network. This "Exact-Jac" version of GPVI is not scalable, but it serves as a benchmark to show that our helper network is able to closely approximate the exact solution. Finally, as deep ensembles lacks any mechanism for correctly estimating the covariance, it performs poorly as expected.

4.3. Multimodal Classification

The previous experiment possesses an analytically known target distribution. However, most problems of interest involve distributions without closed-form expressions. Therefore, we further tested on a 2-dimensional, 4-class classification problem, where each class consists of samples from one component of a mixture distribution. The mixture distribution is defined as $p(x) = \sum_{i=1}^{4} \mathcal{N}(\mu_i, 0.3)$, with means $\mu_i \in \{(-2,-2), (-2,2), (2,-2), (2,2)\}$. We assigned labels $y_i \in \{1,2,3,4\}$ according to the index of the mixture component the samples were drawn from. For this task, samples are weight parameters $\boldsymbol{\theta}$ of two-layer neural networks with 10 hidden units in each layer and ReLU activations, denoted by

$f_{\boldsymbol{\theta}} : \mathbb{R}^2 \to \mathbb{R}^4$.

To train each method, we drew a total of 100 training points, and 200 testing points from the target distribution. To evaluate the posterior predictive distribution, we drew points from a grid spaced from $\{-10, 10\}$, then plot the predictive distribution as measured by the standard deviation in predictions among model samples in figure 2.

In this setting, the true posterior $p(\boldsymbol{\theta}|\boldsymbol{X}, y)$ is unknown and past work often relies on "gold-standard" approaches like HMC to serve as the ground truth. In the previous Bayesian linear regression tests, however, we saw that HMC was outperformed by GPVI as well as ParVI methods SVGD and GFSF. In the current test, intuitively, a "gold-standard" approach should yield a predictive distribution with high variance (high uncertainty) in regions far from the training data, and low variance (low uncertainty) in regions near each mixture component. As shown in figure 2, GPVI and SVGD both have higher uncertainty in no-data regions than HMC, while remaining confident on the training data. On the other hand, MF-VI and amortized SVGD both underestimate the uncertainty. The performance of MF-VI is as expected, as mean-field VI approaches are known to characteristically underestimate uncertainty. Results of all methods as well as more tests on an easier 2-class variant are included in the Appendix.

4.4. Open Category

To evaluate the scalability of our approach we turn to large-scale image classification experiments. In this setting, it is not possible to exactly measure the accuracy of posterior fits. We instead utilize the open-category task to test if our uncertainty estimations can help detect outlier examples. The open category task defines a set of inlier classes that are seen during training, and a set of outlier classes only used for evaluation. While in principle, the content of the outlier classes can be arbitrary, we use semantically similar outlier classes by splitting the training dataset into inlier and outlier classes. The open-category task is in general more difficult than out-of-distribution experiments where different datasets are used as outlier classes, since distributions of categories from the same dataset may be harder to discriminate.

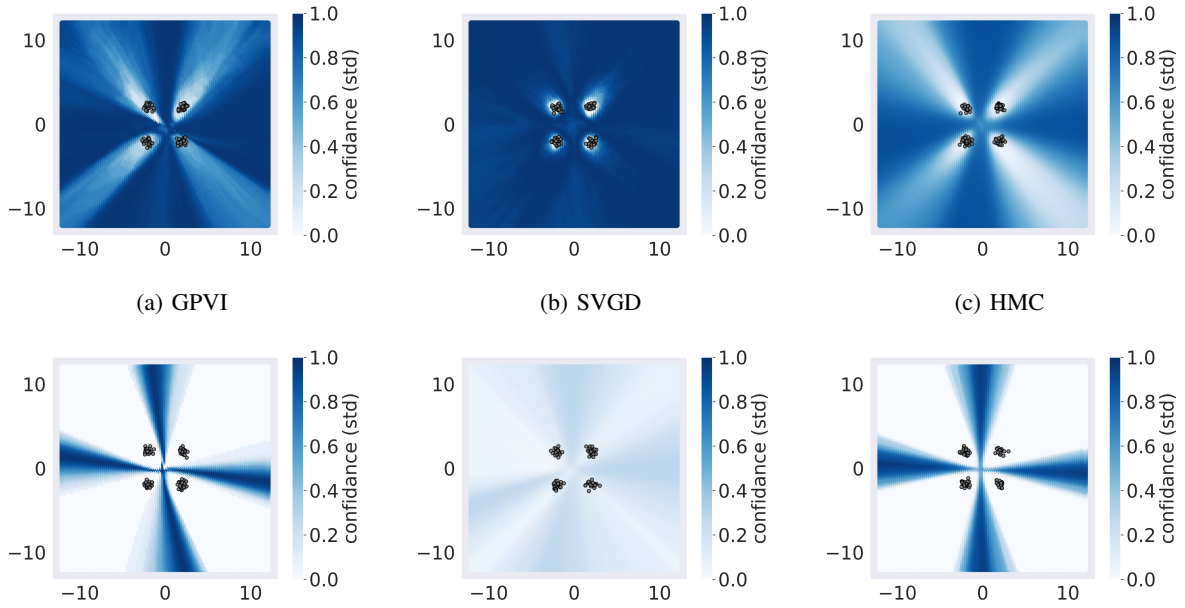
We evaluated on the MNIST and CIFAR-10 image datasets, following Neal et al. (2018) to split each dataset into 6 inlier classes and 4 outlier classes. We performed standard fully supervised training on the 6 inlier classes, and measured uncertainty in the 4 unseen outlier classes. We evaluated the uncertainty using two widely-used statistics: area under the ROC curve (AUC), and the expected calibration error (ECE). The AUC score measures how well a binary classifier discriminates between predictions made on inlier inputs, vs predictions made on outlier inputs. A perfect AUC score of 1.0 indicates a perfect discrimination, while a score of 0.5 indicates that the two sets of predictions are indistinguishable. ECE partitions predictions into equally sized bins, and computes the L_1 difference in expected accuracy and confidence between bins, which represents the calibration error of the bin. ECE computes a weighted average of the calibration error of each bin. Together AUC and ECE tell us how well a model can detect outlier inputs, as well as how well the model fits the training distribution.

MNIST consists of 70,000 grayscale images of handwritten digits at 28x28 resolution, divided into 60,000 training images and 10,000 testing images. We further split the dataset by only using the first six classes for training and testing. The remaining four classes are only used to compute the AUC and ECE statistics. We chose the LeNet-5 classifier architecture for all models, and trained for 100 epochs. Due to the larger computational burden, we only consider 10 samples from each method's approximate posterior for both training and evaluation. Note that our approach is capable of generating many more, but it would be computationally costly to train much more for ensembles and particle VI approaches. For GPVI and amortized ParVI methods we used a 3 layer MLP hypernetwork with layer widths $[256, 512, 1024]$, ReLU activations, and input noise $\boldsymbol{z} \in \mathbb{R}^{256}$. We did not test HMC in this setting, as the computational demand is too high.

Table 3 shows the results. All RKHS-based methods (GPVI, ParVI, amortized ParVI) as well as deep ensembles achieve high supervised ("clean") accuracy with the LeNet architecture. MF-VI underfits slightly, while KSD struggled to achieve competitive accuracy even after 100 epochs. All methods achieved an AUC over 0.95, but SVGD, GFSF and GPVI have the highest AUC values, respectively. In terms of calibration, GPVI and RKHS-based ParVI methods are the best calibrated. KSD is the worst calibrated model, and deep ensembles/MF-VI have middling performance.

Method	Clean	AUC↑	ECE ↓
SVGD	99.3	**.989 ± .001**	**.001 ± .0002**
GFSF	99.2	**.988 ± .003**	.002 ± .0003
KSD	97.7	.964 ± .005	.014 ± .0007
Amortized SVGD	99.1	.958 ± .015	**.002 ± .0007**
Amortized GFSF	99.2	.978 ± .005	.004 ± .0013
Amortized KSD	97.7	.951 ± .008	.017 ± .0010
MF-VI	98.6	.951 ± .008	.014 ± .0027
Deep Ensemble	99.3	.972 ± .002	.008 ± .0060
GPVI	99.3	**.988 ± .001**	**.001 ± .0005**

Table 3. Results for open-category classification on MNIST. We show the result of standard supervised training (Clean), as well as AUC and ECE statistics computed from training on a subset of classes and testing on the rest of the classes as outliers.



(a) GPVI (b) SVGD (c) HMC

(d) Amortized SVGD (e) MF-VI (f) Deep Ensemble

Figure 2. Predictive uncertainty of each method on the 4-class classification task, as measured by the standard deviation between predictions of sampled functions. Regions of high uncertainty are shown as darker, while lighter regions correspond to lower uncertainty. The training data is shown as samples from four unimodal normal distributions. It can be seen that amortized SVGD, MF-VI and deep ensembles significantly underestimate the uncertainty in regions with no training data

CIFAR-10 consists of 60,000 RGB images depicting 10 object classes at 32x32 resolution, divided into 50,000 training images and 10,000 testing images. We adopted the same 6 inlier / 4 outlier split used in (Neal et al., 2018) for the open-category setting. For this task we used a CNN with 3 convolutional layers and two linear layers, which is much smaller than SOTA classifiers for CIFAR-10. Though the classification accuracy would suffer, it allows us to clearly evaluate our method without considering interactions with architectural components such as BatchNorm or residual connections. We also used 10 samples from each method's approximate posterior for training and evaluation. For GPVI and amortized ParVI methods we used the same hypernetwork architecture as in the MNIST setup. Table 4 shows the results where it can be seen that our GPVI almost match the performance of SVGD and GFSF in terms of AUC while doing a bit better on ECE.

Method	Clean	AUC ↑	ECE ↓
SVGD	80.3	**.683 ± .008**	.055 ± .004
GFSF	80.6	**.681 ± .004**	.068 ± .012
Amortized SVGD	71.12	.636 ± .018	.073 ± .029
Amortized GFSF	71.09	.583 ± .007	.042 ± .029
MF-VI	70.0	.649 ± .006	**.016 ± .002**
Deep Ensemble	73.54	.652 ± .018	.033 ± .011
GPVI	76.2	.677 ± .008	**.018 ± .015**

Table 4. Open-category classification on CIFAR-10. We show results of standard supervised training (Clean), as well as AUC and ECE of each method trained in the open-category setting.

4.5. Discussion

Much of the focus regarding recent work on Bayesian neural networks concerns their performance on open-category and out-of-distribution tasks with high dimensional image datasets. Instead, we show that our method closely approximates the target posterior, both in tasks where the posterior is explicitly known, as well as when it is intractable. The Bayesian linear regression and density estimation tasks served as sanity checks. Because the posterior was known explicitly, we could quantitatively test how well each method fit the posterior. For density estimation, GPVI outperforms amortized approaches at matching the target covariance (Table 1). In Bayesian linear regression, while the approximation was close for all methods, there was a clear hierarchy in terms of which types of methods produced the tightest approximation (Table 2). GPVI and RKHS-based ParVI achieved the best posterior fit overall, and we see in further experiments that quality of fit here is indicative of performance in more difficult tasks.

The four-class classification problem (Fig. 2), while seemingly simple, is particularly difficult for most methods we evaluated since many methods tend to overgeneralize to the corners. MF-VI underestimates the uncertainty as expected

(Yao et al., 2019; Minka, 2001; Bishop, 2006). Notably, amortized SVGD also underestimates the uncertainty with a predictive distribution resembling that of a standard ensemble. We believe this is due to the compounding approximation error explained in Section 3.5, i.e., naively back-propagating the Stein variational gradient to update the generator is more likely to end up with a non-descent direction compared with back-propagating the exact functional gradient as GPVI. As shown in Fig. 2, the posterior approximation of GPVI is tighter, with uncertainty that better matches the data distribution. SVGD performs the best in this task, with high uncertainty everywhere except in regions near observed data. Surprisingly, GPVI and SVGD outperform HMC here, with clearly higher uncertainty near the corners of the sample space. Note that with the functional approximation by neural networks, it is hard to determine if the true posterior exactly matches the intuitively "ideal" uncertainty plot where low variance only shows up around each mixture component. On the other hand, HMC is guaranteed to converge to the true posterior over time (Durmus et al., 2017). Given a fixed computational budget, it is possible that GPVI or SVGD could achieve better performance. GPVI also has the extra benefit of the ability to draw additional samples, which is not possible with ParVI or HMC.

Our final experimental setting, open-category, reveals that GPVI consistently has a higher AUC than all other scalable sampling approaches. On MNIST, GPVI is among the overall top performers, together with two ParVI methods SVGD and GFSF. On CIFAR-10, GPVI is among the top two performers in ECE and performs only slightly behind the two best ParVI methods in AUC. Most importantly, GPVI outperforms with clear margins all amortized ParVI methods on both datasets under all metrics. This is consistent with all other qualitative and quantitative experiments we have conducted, which again shows the advantage of GPVI over existing generative ParVI methods.

We found that KSD completely failed in CIFAR-10, achieving slightly better than random accuracy. We believe this is due to the way that KSD performs the particle update. Where SVGD has a closed form expression for the particle transportation map, KSD parameterizes it with a critic network that has the input-output dimensionality of the particle parameters. Training this critic naturally becomes difficult when applying to neural network functions.

In the open-category setting, there is added time complexity for GPVI relative to amortized methods due to training our helper network. Compared to amortized SVGD, the extra time complexity of training GPVI is a constant factor less than 1. In our MNIST experiments, this additive factor is 0.28; in CIFAR-10 experiments, this factor is 0.78. The difference in MNIST and CIFAR-10 is due to that the output dimension of the helper network scales with the output dimension of the generator. We believe this is acceptable given the performance gain and the possible real (offline training) applications of GPVI.

5. Conclusion

We have presented a new method that fuses the best aspects of parametric VI with non-parametric ParVI. GPVI has asymptotic convergence on par with ParVI. Additionally, GPVI can efficiently draw samples from the posterior. We also presented a method for efficiently estimating the product between the inverse of the Jacobian of a deep network, and a gradient vector. Our experiments showed that GPVI performs on par with ParVI, and outperforms amortized ParVI and other competing methods in Bayesian linear regression, a classification task, as well as open-category tasks on MNIST and CIFAR-10. In the future we want to explore the efficacy of our method applied to large scale tasks like image generation.

Acknowledgements

Neale Ratzlaff and Li Fuxin were partially supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0011 and HR001120C0022. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

References

Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

Buntine, W. L. and Weigend, A. S. Bayesian back-propagation. *Complex systems*, 5(6):603–643, 1991.

Chen, T., Fox, E., and Guestrin, C. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pp. 1683–1691, 2014.

Cheng, C.-A. and Boots, B. Variational inference for gaussian process models with linear complexity. In *Advances in Neural Information Processing Systems*, pp. 5184–5194, 2017.

Durmus, A., Moulines, E., and Saksman, E. On the convergence of hamiltonian monte carlo. *arXiv preprint arXiv:1705.00166*, 2017.

Fletcher, R. Conjugate gradient methods for indefinite systems. In *Numerical analysis*, pp. 73–89. Springer, 1976.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059, 2016.

Grathwohl, W., Wang, K.-C., Jacobsen, J.-H., Duvenaud, D., and Zemel, R. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, pp. 3732–3747. PMLR, 2020.

Graves, A. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24:2348–2356, 2011.

Hensman, J., Fusi, N., and Lawrence, N. D. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

Hensman, J., Matthews, A., and Ghahramani, Z. Scalable variational gaussian process classification. 2015.

Hu, T., Chen, Z., Sun, H., Bai, J., Ye, M., and Cheng, G. Stein neural sampler. *arXiv preprint arXiv:1810.03545*, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28: 2575–2583, 2015.

Korattikara Balan, A., Rathod, V., Murphy, K. P., and Welling, M. Bayesian dark knowledge. *Advances in Neural Information Processing Systems*, 28:3438–3446, 2015.

Krueger, D., Huang, C.-W., Islam, R., Turner, R., Lacoste, A., and Courville, A. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in neural information processing systems*, pp. 6402–6413, 2017.

Liu, C., Zhuo, J., Cheng, P., Zhang, R., and Zhu, J. Understanding and accelerating particle-based variational inference. In *International Conference on Machine Learning*, pp. 4082–4092. PMLR, 2019.

Liu, Q. Stein variational gradient descent as gradient flow. In *Advances in neural information processing systems*, pp. 3115–3123, 2017.

Liu, Q. and Wang, D. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29: 2378–2386, 2016.

Louizos, C. and Welling, M. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pp. 1708–1716, 2016.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational bayesian neural networks. *arXiv preprint arXiv:1703.01961*, 2017.

MacKay, D. J. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Marzouk, Y., Moselhy, T., Parno, M., and Spantini, A. An introduction to sampling via measure transport. *arXiv preprint arXiv:1602.05023*, 2016.

Minka, T. P. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Neal, L., Olson, M., Fern, X., Wong, W.-K., and Li, F. Open set learning with counterfactual images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 613–628, 2018.

Neal, R. M. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

Neal, R. M. et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Pawlowski, N., Brock, A., Lee, M. C., Rajchl, M., and Glocker, B. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297*, 2017.

Rasmussen, C. E. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003.

Rezende, D. and Mohamed, S. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pp. 1530–1538. PMLR, 2015.

Strathmann, H., Sejdinovic, D., Livingstone, S., Szabo, Z., and Gretton, A. Gradient-free hamiltonian monte carlo with efficient kernel exponential families. *Advances in Neural Information Processing Systems*, 28:955–963, 2015.

Wang, D. and Liu, Q. Learning to draw samples: With application to amortized mle for generative adversarial learning. *arXiv preprint arXiv:1611.01722*, 2016.

Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. Exact gaussian processes on a million data points. In *Advances in Neural Information Processing Systems*, pp. 14648–14659, 2019.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688, 2011.

Wilson, A. G., Hu, Z., Salakhutdinov, R. R., and Xing, E. P. Stochastic variational deep kernel learning. *Advances in Neural Information Processing Systems*, 29:2586–2594, 2016.

Yao, J., Pan, W., Ghosh, S., and Doshi-Velez, F. Quality of uncertainty quantification for bayesian neural network inference. *arXiv preprint arXiv:1906.09686*, 2019.

Young, D. Iterative methods for solving partial difference equations of elliptic type. *Transactions of the American Mathematical Society*, 76(1):92–111, 1954.