Appendix for Generative Particle Variational Inference via Estimation of Functional Gradients

A. Proofs

The summation convention is used on all repeated indices (e.g. $a^i b^i := \sum_i a^i b^i$) in the following proofs.

A.1. Proof of Theorem 3.1

To compute the gradient of $\mathcal{J}(\boldsymbol{f}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}(\boldsymbol{z})) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)\right|}\right]$, for any $\boldsymbol{\phi} \in T_{\boldsymbol{f}}\mathcal{H}$,

$$\begin{aligned}
d\mathcal{J}_{\boldsymbol{f}}(\boldsymbol{\phi}) &= \left.\frac{d}{dt}\right|_{t=0} \mathcal{J}(\boldsymbol{f} + t\boldsymbol{\phi}) \\
&\overset{①}{=} \mathbf{E}_{\boldsymbol{z}}\left[\left.\frac{d}{dt}\right|_{t=0} (-\log p((\boldsymbol{f}+t\boldsymbol{\phi})(\boldsymbol{z})))\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left.\frac{d}{dt}\right|_{t=0} \log\left|\det\left(\frac{\partial(\boldsymbol{f}+t\boldsymbol{\phi})}{\partial \boldsymbol{z}}\right)\right|\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} \left.\frac{d}{dt}\right|_{t=0} (f^i + t\phi^i)(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\mathrm{Tr}\left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1} \left.\frac{d}{dt}\right|_{t=0} \frac{\partial(\boldsymbol{f}+t\boldsymbol{\phi})}{\partial \boldsymbol{z}}\right)\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} \phi^i(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\right)_i^j \frac{\partial \phi^i}{\partial z^j}\right] \\
&\overset{②}{=} \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} \langle k(\boldsymbol{z},\cdot),\ \phi^i(\boldsymbol{z})\rangle_{\mathcal{H}}\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left\langle \left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\right)_i^j \nabla_{z^j} k(\boldsymbol{z},\cdot),\ \phi^i(\boldsymbol{z})\right\rangle_{\mathcal{H}}\right] \\
&= \left\langle \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} k(\boldsymbol{z},\cdot) - \left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right)^{-1}\right)_i^j \nabla_{z^j} k(\boldsymbol{z},\cdot)\right], \phi^i\right\rangle_{\mathcal{H}},
\end{aligned} \tag{1}$$

the following identities are used in step ① and ②,

$$d\log|\det A| = \mathrm{Tr}(A^{-1}dA),$$

$$\phi^i(\boldsymbol{z}) = \langle k(\boldsymbol{z},\cdot), \phi^i(\boldsymbol{z})\rangle_H,$$

$$\nabla_{z^j}\phi^i(\boldsymbol{z}) = \langle \nabla_{z^j} k(\boldsymbol{z},\cdot), \phi^i(\boldsymbol{z})\rangle_H.$$

By definition of the gradient,

$$\nabla_{\boldsymbol{f}}\mathcal{J}(\boldsymbol{f})(\boldsymbol{z}) = \mathbf{E}_{\boldsymbol{z}'}\left[-\nabla_{\boldsymbol{x}}\log p(\boldsymbol{x})\Big|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z}')} k(\boldsymbol{z}',\boldsymbol{z}) - \left(\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1}\right) \nabla_{\boldsymbol{z}'} k(\boldsymbol{z}',\boldsymbol{z})\right]. \tag{2}$$

A.2. Proof of Lemma 3.2

To compute the gradient of $\mathcal{J}(\boldsymbol{\phi}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{\phi}(\boldsymbol{f}(\boldsymbol{z}))) + \log \frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial(\boldsymbol{\phi}\circ\boldsymbol{f})}{\partial \boldsymbol{z}}\right)\right|}\right]$ at $\boldsymbol{\phi} = \mathrm{id}$, for any $\boldsymbol{v} \in T_{\boldsymbol{\phi}}\mathcal{H}$,

$$
\begin{aligned}
d\mathcal{J}_{\boldsymbol{\phi}}(\boldsymbol{v}) &= \frac{d}{dt}\bigg|_{t=0} \mathcal{J}(\boldsymbol{\phi}+t\boldsymbol{v})\bigg|_{\boldsymbol{\phi}=\mathrm{id}} \\
&= \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\bigg|_{t=0} (-\log p((\boldsymbol{f}+t\boldsymbol{v}\circ\boldsymbol{f})(\boldsymbol{z})))\right] - \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\bigg|_{t=0} \log\left|\det\left(\frac{\partial(\boldsymbol{f}+t\boldsymbol{v}\circ\boldsymbol{f})}{\partial\boldsymbol{z}}\right)\right|\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} \frac{d}{dt}\bigg|_{t=0} (f^i + t(\boldsymbol{v}\circ\boldsymbol{f})^i(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\mathrm{Tr}\left(\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{z}}\right)^{-1} \frac{d}{dt}\bigg|_{t=0} \frac{\partial(\boldsymbol{f}+t\boldsymbol{v}\circ\boldsymbol{f})}{\partial\boldsymbol{z}}\right)\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} v^i(\boldsymbol{f}(\boldsymbol{z}))\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left(\left(\frac{\partial\boldsymbol{f}}{\partial\boldsymbol{z}}\right)^{-1}\right)_i^j \frac{\partial f^k}{\partial z^j}\frac{\partial v^i}{\partial x^k}\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^i}\log p(\boldsymbol{x})\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})} \langle k(\boldsymbol{f}(\boldsymbol{z}),\cdot),\ v^i(\boldsymbol{f}(\boldsymbol{z}))\rangle_{\mathcal{H}}\right] - \mathbf{E}_{\boldsymbol{z}}\left[\left\langle \nabla_{x^i} k(\boldsymbol{x},\cdot)\bigg|_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})},\ v^i(\boldsymbol{f}(\boldsymbol{z}))\right\rangle_{\mathcal{H}}\right] \\
&= \left\langle \mathbf{E}_{\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})}\left[-\nabla_{x^i}\log p(\boldsymbol{x})k(\boldsymbol{x},\cdot) - \nabla_{x^i}k(\boldsymbol{x},\cdot)\right], v^i\right\rangle_{\mathcal{H}}. \qquad (3)
\end{aligned}
$$

By definition of the gradient,

$$
\nabla_{\boldsymbol{\phi}}\mathcal{J}(\boldsymbol{\phi})(\boldsymbol{x})\Big|_{\boldsymbol{\phi}=\mathrm{id}} = \mathbf{E}_{\boldsymbol{x}'}\left[-\nabla_{\boldsymbol{x}'}\log p(\boldsymbol{x}')k(\boldsymbol{x}',\boldsymbol{x}) - \nabla_{\boldsymbol{x}'}k(\boldsymbol{x}',\boldsymbol{x})\right], \qquad (4)
$$

where $\boldsymbol{x}=\boldsymbol{f}(\boldsymbol{z})$ and $\boldsymbol{x}'=\boldsymbol{f}(\boldsymbol{z}')$.

A.3. Directly computing the gradient $\mathcal{J}(\boldsymbol{\theta})$

As a completion of the discussion, we also derive the gradient of $\mathcal{J}(\boldsymbol{\theta}) = \mathbf{E}_{\boldsymbol{z}}\left[-\log p(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})) + \log\frac{p_{\boldsymbol{z}}(\boldsymbol{z})}{\left|\det\left(\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}}}{\partial\boldsymbol{z}}\right)\right|}\right]$ w.r.t. $\boldsymbol{\theta}$, for any $\boldsymbol{v}\in T_{\boldsymbol{\theta}}W$,

$$
\begin{aligned}
d\mathcal{J}_{\boldsymbol{\theta}}(\boldsymbol{v}) &= \frac{d}{dt}\bigg|_{t=0}\mathcal{J}(\boldsymbol{\theta}+t\boldsymbol{v}) \\
&= \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\bigg|_{t=0}\big(-\log p(\boldsymbol{f}_{\boldsymbol{\theta}+t\boldsymbol{v}}(\boldsymbol{z}))\big)\right] - \mathbf{E}_{\boldsymbol{z}}\left[\frac{d}{dt}\bigg|_{t=0}\log\det\left(\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}+t\boldsymbol{v}}}{\partial\boldsymbol{z}}\right)\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^j}\log p(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}))\frac{d}{dt}\bigg|_{t=0} f^j_{\boldsymbol{\theta}+t\boldsymbol{v}}(\boldsymbol{z})\right] - \mathbf{E}_{\boldsymbol{z}}\left[\mathrm{Tr}\left(\left(\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}}}{\partial\boldsymbol{z}}\right)^{-1}\frac{d}{dt}\bigg|_{t=0}\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}+t\boldsymbol{v}}}{\partial\boldsymbol{z}}\right)\right] \\
&= \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^j}\log p(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}))\frac{\partial f^j}{\partial\theta^i}v^i i\right] - \mathbf{E}_{\boldsymbol{z}}\left[\mathrm{Tr}\left(\left(\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}}}{\partial\boldsymbol{z}}\right)^{-1}\frac{\partial^2\boldsymbol{f}_{\boldsymbol{\theta}}}{\partial\boldsymbol{z}\partial\boldsymbol{\theta}}\boldsymbol{v}\right)\right] \\
&= \left\langle \mathbf{E}_{\boldsymbol{z}}\left[-\nabla_{x^j}\log p(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z}))\frac{\partial f^j}{\partial\theta^i} - \left(\left(\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}}}{\partial\boldsymbol{z}}\right)^{-1}\right)_k^j\left(\frac{\partial^2 f^j_{\boldsymbol{\theta}}}{\partial z^k\partial\theta^i}\right)\right], v^i\right\rangle_E, \qquad (5)
\end{aligned}
$$

where $\langle\cdot,\cdot\rangle_E$ denotes the Euclidean inner product. Therefore,

$$
\nabla_{\theta^i}(\mathcal{J}(\boldsymbol{\theta})) = \mathbf{E}_{\boldsymbol{z}}\left[-\frac{\partial f^j}{\partial\theta^i}\nabla_{x^j}\log p(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{z})) - \left(\left(\frac{\partial\boldsymbol{f}_{\boldsymbol{\theta}}}{\partial\boldsymbol{z}}\right)^{-1}\right)_k^j\left(\frac{\partial^2 f^j_{\boldsymbol{\theta}}}{\partial z^k\partial\theta^i}\right)\right].
$$

Computation of the second term in the expectation involves second derivatives of $\boldsymbol{f}_{\boldsymbol{\theta}}$, which is not amenable in large-scale problems. As a result, we avoid this direct approach and choose to pullback the functional gradient (2) instead.

A.4. Proof of Injectivity of $\boldsymbol{f}$

We parameterize $\boldsymbol{f}$ as follows,

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

where $\boldsymbol{z}^{(:k)} \in \mathbb{R}^k$ denotes the vector consisting of the first $k(\ll d)$ components of $\boldsymbol{z}$, and $\boldsymbol{g} : \mathbb{R}^k \to \mathbb{R}^d$ is a much slimmer neural network. For any experimental setting, our parameterization of $\boldsymbol{g}$ uses a dimension k less than 30% the size of d. For our high dimensional open-category experiments where $d > 60000$, we use a k of less than 2% of d. The Jacobian is,

$$\left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}\right]_{d\times d} = \left[\left[\frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}^{(:k)}}\right]_{d\times k} \middle| \mathbf{0}_{d\times(d-k)}\right]_{d\times d} + \lambda \boldsymbol{I}_d. \tag{7}$$

We now show that our practical choice of $\boldsymbol{f}$ is indeed injective.

From (6), for sufficiently large λ, the Jacobian $J_{\boldsymbol{f}}$ given by (7) is positive definite. Since the domain of $\boldsymbol{z}$ is convex, it is straightforward to show $\boldsymbol{f}$ is injective,

For any two different points $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathcal{Z}$, consider the line segment $\boldsymbol{z}_1 + t(\boldsymbol{z}_2 - \boldsymbol{z}_1), t \in [0, 1]$, which lies in $\mathcal{Z}$ given the convexity of $\mathcal{Z}$. From the Fundamental Theorem of calculus,

$$\begin{aligned}(\boldsymbol{z}_2 - \boldsymbol{z}_1)^T(\boldsymbol{f}(\boldsymbol{z}_2) - \boldsymbol{f}(\boldsymbol{z}_1)) &= (\boldsymbol{z}_2 - \boldsymbol{z}_1)^T\left(\int_0^1 J_{\boldsymbol{f}}(\boldsymbol{z}_1 + t(\boldsymbol{z}_2 - \boldsymbol{z}_1))dt\right) \cdot (\boldsymbol{z}_2 - \boldsymbol{z}_1) \\ &= \int_0^1 (\boldsymbol{z}_2 - \boldsymbol{z}_1)^T J_{\boldsymbol{f}}(\boldsymbol{z}_1 + t(\boldsymbol{z}_2 - \boldsymbol{z}_1))(\boldsymbol{z}_2 - \boldsymbol{z}_1)dt > 0\end{aligned}$$

the last inequality is due to the positive definiteness of $J_{\boldsymbol{f}}$ on $\mathcal{Z}$. Therefore, $\boldsymbol{f}(\boldsymbol{z}_1) \neq \boldsymbol{f}(\boldsymbol{z}_2)$, $\boldsymbol{f}$ is injective.

B. Method Details

We provide architectures and computational details of our generator and helper networks.

B.1. Helper Network

The helper network $\boldsymbol{h}_\eta$ takes $\boldsymbol{z}'$ and $\nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})$ as inputs and outputs $\left(\frac{\partial \boldsymbol{f}(\boldsymbol{z}')}{\partial \boldsymbol{z}'}\right)^{-1} \nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})$. Given the specific parameterization of $\boldsymbol{f}$ defined by (6), the Jacobian (7) only depends on $\boldsymbol{z}'^{(:k)}$, the first k components of $\boldsymbol{z}'$. The helper network applies a fully connected layer with ReLU activation to the input $\boldsymbol{z}'^{(:k)}$ and $\nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})$ respectively, then the outputs from these two branches are concatenated and sent to a three-layer fully connected network. We optimize the helper network with the Adam optimizer Kingma & Ba (2014) and a learning rate of $1e-4$. We found that regularization techniques like batchnorm and weight decay hurt performance, presumably because we are optimizing for an exact solution and thus do not want a smoothed estimate. While our helper network makes computation of the Jacobian inverse tractable, it nonetheless adds the complexity of an extra optimization problem per training step of $\boldsymbol{f}$. To mitigate the increased training time we update $\boldsymbol{h}_\eta$ just once per training step of $\boldsymbol{f}$; taking one step also keeps the network from over-committing to a single input.

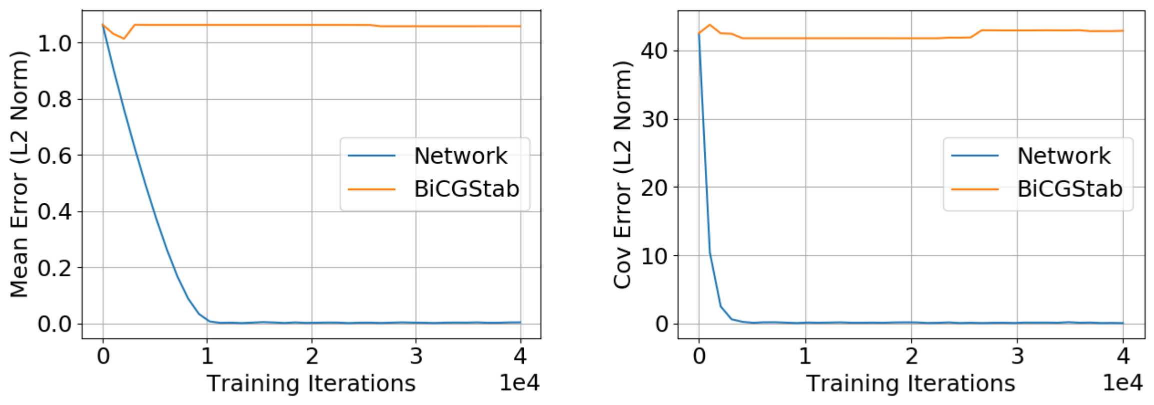
B.2. Kernel Selection

We follow Liu & Wang (2016) and use an RBF kernel $k(x, x') = \exp\left(\frac{1}{h}||x - x'||_2^2\right)$ for all kernel-based methods including GPVI, SVGD, GFSF, and their amortized variants. The bandwidth h is computed using the median method also proposed by Liu & Wang (2016): $h = med^2/\log n$, where med is the median of pairwise distances between samples $\{x_i\}_{i=1}^n$.

C. Comparing with an Alternative Solver for $\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}\right)^{-1} \nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})$

Here we justify the use of our helper network $\boldsymbol{h}_\eta$, by comparing its performance to a more traditional linear solver: the stabilized biconjugate gradient method (BiCGSTAB) (Saad, 2003). BiCGSTAB is a well-known iterative algorithm for solving systems of the form $Ax = b$. It is similar to the conjugate gradient method, but does not require A to be self-adjoint, giving BiCGSTAB wider applicability. In figure 1, we compare our helper network (”Network”) against BiCGSTAB in the Bayesian linear regression setting from section 4.2 in the main text. For BiCGSTAB, we solve $B * B$ systems of the form $JJ^{-1}\nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z}) = \nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})$ for each iteration of training our generator, where $J = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}'}$ and $B * B$ is the effective batch size of $\nabla_{\boldsymbol{z}'}k(\boldsymbol{z}', \boldsymbol{z})$. Due to the greatly increased training time from solving $B * B$ independent problems per training iteration, we only run BiCGSTAB for one step, and warm start from the previous solution at each new generator training iteration. In addition to increased training time, BiCGSTAB suffers from instability due to the constantly changing $\frac{\partial \boldsymbol{f}(\boldsymbol{z}')}{\partial \boldsymbol{z}'}$ as well as $\nabla_{\boldsymbol{z}'} k(\boldsymbol{z}', \boldsymbol{z})$, due to the resampling of $\boldsymbol{z}$ and $\boldsymbol{z}'$ at each step. As seen in figure 1, when using BiCGSTAB, GPVI is unable to fit the target distribution, while using our helper network (”Network”) we are able to efficiently minimize the mean and covariance error.

We also considered using a normalizing flow as a replacement for our generator network, as normalizing flows are invertible with lower triangular Jacobians. Unfortunately, the efficiency of normalizing flows comes from setting the input-output dimensionality to be equal to force the Jacobian to be square. When using our method for BNNs, as explained in Section B, we cannot afford to store a generator with equal input-output dimensionality, making normalizing flows an inefficient choice at best for our own method. Our helper network stands as an efficient, novel solution for explicitly estimating the Jacobian inverse vector product when needed. Nevertheless, we compare GPVI with normalizing flow method for density estimation in the next section.



(a) Mean Error (b) Cov Error

Figure 1: Comparing our helper network with BiCGSTAB in the Bayesian linear regression setting.

D. Additional Results

We show additional results on density estimation and classification.

D.1. Density Estimation

Here we provide additional results in the density estimation setting.

Comparison with Explicit Jacobian

Here we perform the experiments from section 4.1 of the main paper, and compare GPVI with a variant where the Jacobian and its inverse are explicitly computed. The inverse is computed with the PyTorch (Paszke et al., 2019) function `torch.inverse`, which uses LAPACK routines `getrf` and `getri`. We can see in table 1 that GPVI is competitive with the “Exact-Jac” variant in the 2D setting, and even performs better in the 5D setting. This is unexpected, as explicitly computing the Jacobian and its inverse should be the correct way to compute the functional gradient. We hypothesize that the Jacobian of our generator network is ill-conditioned. In this case any inversion algorithm is more prone to large numerical error. Because our helper network is updated once per training iteration, we may avoid the large gradients that come from numerical error. The increased performance of GPVI over the “Exact-Jac” variant in the 5D setting may be due to this smoother training.

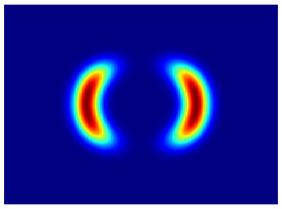
Energy Potentials

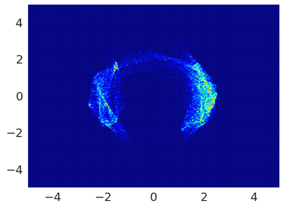
We use the four non-Gaussian energy potentials defined in Rezende & Mohamed (2015), and train GPVI as a sampler. In figures 2-5, we see (from left to right) the target density, GPVI, amortized SVGD, and normalizing flows (Rezende & Mohamed, 2015). We found that while our method has more parameters, we don’t need nearly as deep a model as with normalizing flows. We used just 2 hidden layers for GPVI, while we needed a flow of depth 32 to get comparable

Method	Σ error (2d) $\downarrow$	Σ error (5d) $\downarrow$
Amortized SVGD	$0.10 \pm .09$	$0.37 \pm .32$
GPVI	$0.14 \pm .08$	$0.14 \pm .04$
GPVI Exact Jac	$0.15 \pm .09$	$0.49 \pm .17$

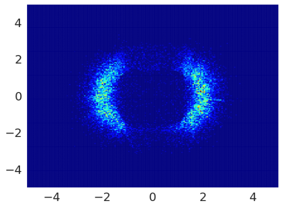
Table 1: Comparison of generative Particle VI approaches for density estimation of 2d and 5d Gaussian distributions.

performance. We trained each method for 200000 steps with a batch size of 100. To generate plots, we sampled 20000 points from each model. We detail the rest of the hyperparameters in section F. In figures 3, 4, 5 we see that GPVI is able to capture the variance of the target distribution, while amortized SVGD samples mostly from the mean. For the Sin Bisect setting, GPVI is the only method that captures some of both halves of the middle section. But in Sin Split, only normalizing flows capture both halves of the split section.

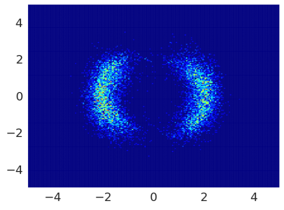

(a) Twin Moons Target



(b) GPVI

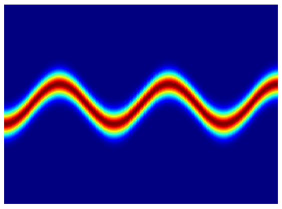


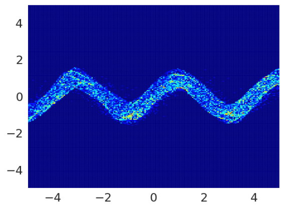
(c) Amortized SVGD



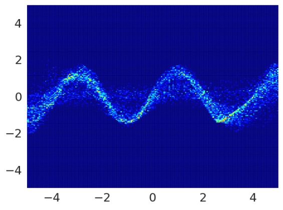
(d) Normalizing Flow

Figure 2: Twin Moons

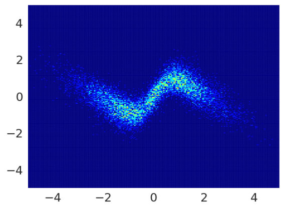

(a) Sinusoid Target



(b) GPVI

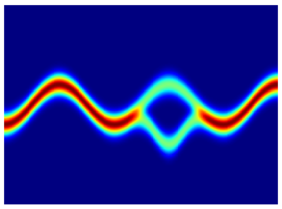


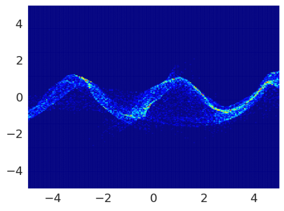
(c) Amortized SVGD



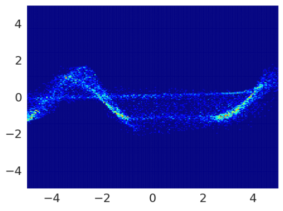
(d) Normalizing Flow

Figure 3: Sinusoid

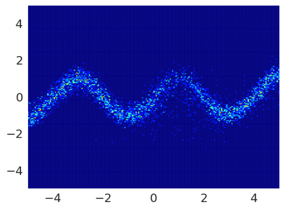

(a) Sin Bisect Target



(b) GPVI



(c) Amortized SVGD



(d) Normalizing Flow

Figure 4: Sin Bisect

D.2. Classification

In figure 6 we show the results of all evaluated methods on the four-class classification problem from section 4.3 of the main text.

In addition to the four-class classification problem we presented in the main paper, we show results on a simpler two class variant. In this setting, we generate data in the same way as in the four-class setting, but we use a mixture distribution with two components. Specifically, the mixture distribution is defined as $p(x) = \sum_{i=1}^{2} \mathcal{N}(\mu_i, 0.3)$, with means $\mu_i \in \{(-2,-2),(2,2)\}$. We assigned labels $y_i \in \{1,2\}$ according to the index of the mixture component the samples were

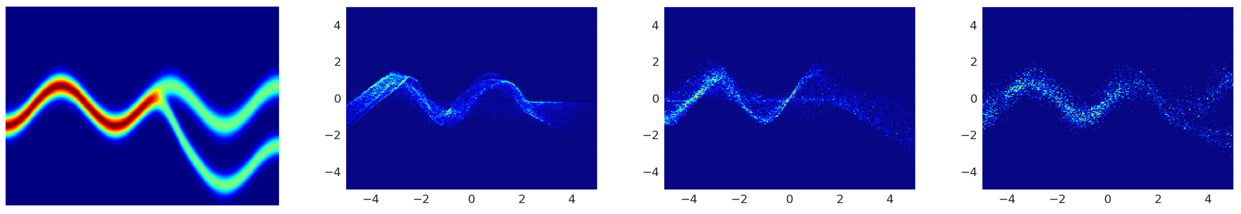

(a) Sin Split Target (b) GPVI (c) Amortized SVGD (d) Normalizing Flow

Figure 5: Sin Split

drawn from. We show the results of each method in figure 7. We can see that GPVI again gives better uncertainty estimates than other sampling based approaches. Again, we do not know what the true posterior over classifications looks like, but GPVI and RKHS-based ParVI approaches give uncertainty estimates that closely match our intuition for this problem.

E. Experimental Details

Here we provide some additional details regarding the setup and reporting of our experiments and their results.

HMC Details We use the same settings for HMC for the Bayesian linear regression and 2/4 class classification tasks. We sampled the momentum from a standard normal distribution, and used 25 leap-frog steps per sample. We ran each experiment for 25K steps in total, with a burn-in of 20K steps, and a step size of 0.0005. We thinned each chain after burn-in, and tuned the number of leapfrog steps and step-size for each experiment. To check for convergence, we checked that the means of each chain were similar, indicating mixing.

1D Regression For our 1D regression task (figure 1 of the main paper), we generate a dataset of 80 samples X with targets Y. We draw 76 samples of X uniformly from $[-6, -2] \cup [2, 6]$, and draw the 4 remaining samples from $[-2, 2]$. The targets Y are computed as $Y = -(1 + X)\sin(1.2X) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, 0.04)$. For all methods we use 100 posterior samples, and train for 50K iterations. For GPVI and amortized SVGD we use a generator with two linear layers $[32, 32]$ and Gaussian input noise of the same dimension.

Density Estimation
In the density estimation setting (section 4.1 of the main text), we randomly generate $2d$ and $5d$ target covariance matrices. Specifically, the target covariance we wish to fit is computed as $\Sigma^* = \Sigma\Sigma^T, \Sigma \sim \mathcal{N}(0, I_d), d \in \{2, 5\}$. We use an MLP with 1 hidden layer of width 2 to approximately sample from a unimodal Gaussian distribution with covariance Σ^*. Given Gaussian input noise, the variance of the output distribution of the linear generator is computed as W^TW, which should match Σ^* after training.

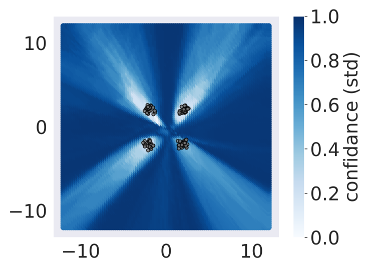
Open Category Tasks
For the open-category tasks, we computed two metrics to evaluate each method: AUROC (AUC) and ECE. In this context, AUC measures how well a binary classifier can discriminate between predictions made on inlier vs outlier test inputs. An AUC score of 1.0 indicates that the predictions made by a model are perfectly separable, meaning that we could set a threshold on predicted probabilities to detect every outlier test input. In practice we first make predictions on the inlier and outlier test sets, then compute the variance of the predictions over the sampled predictors. The AUC score is then computed against the predictive variance using the scikit-learn library.

Expected Calibration Error (ECE) measures how well calibrated a model's predictions are. Given predictions on inlier test inputs, ECE partitions predictions into $M = 15$ equally sized bins according to their confidence. For each bin b_m we compute the difference between their accuracy $1/|b_m| \sum_{x_i \in b_m} (\hat{y} - y)$ and confidence $1/b_m \sum_{x_i \in m} \hat{p}$. Where $\hat{p}, \hat{y}$ are predicted probabilities and associated label with respect to a data-point x_i and true label y. This difference in expected accuracy and confidence between each bin represents the bin's calibration error.

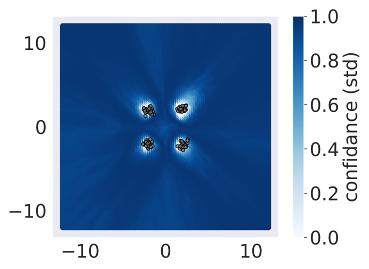
For all experiments we report an average of three runs using the hyperparameter settings given in section F.

F. Hyperparameter Settings

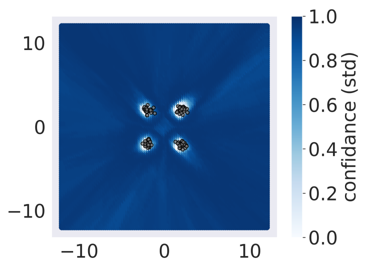
In tables 2 - 6 we detail the hyperparameters chosen for each method in each experimental setting. We refer to the sampler network as $\boldsymbol{f}$, and the predicting classifier as $\boldsymbol{g}$. We generally use the same structure of $\boldsymbol{f}$ and input noise for GPVI and



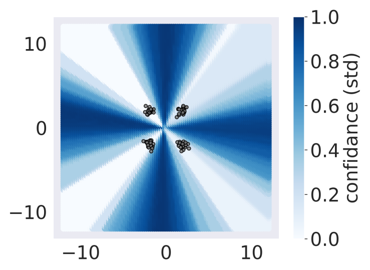
(a) GPVI



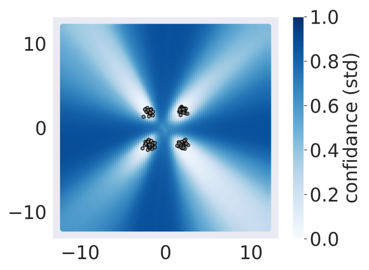
(b) SVGD



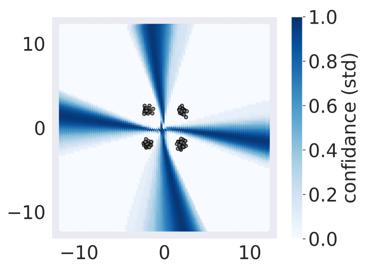
(c) GFSF



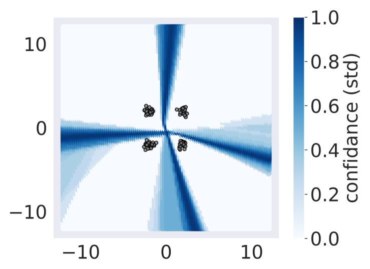
(d) KSD



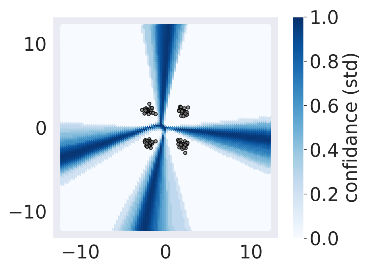
(e) HMC



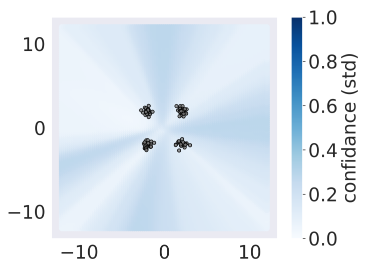
(f) Amortized SVGD



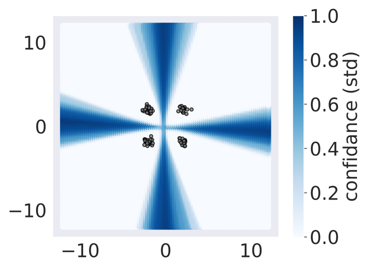
(g) Amortized GFSF



(h) Amortized KSD



(i) MF-VI



(j) Deep Ensembles

Figure 6: Predictive uncertainty of each method on the 4-class classification task, as measured by the standard deviation between predictions of sampled functions.

amortized ParVI methods.

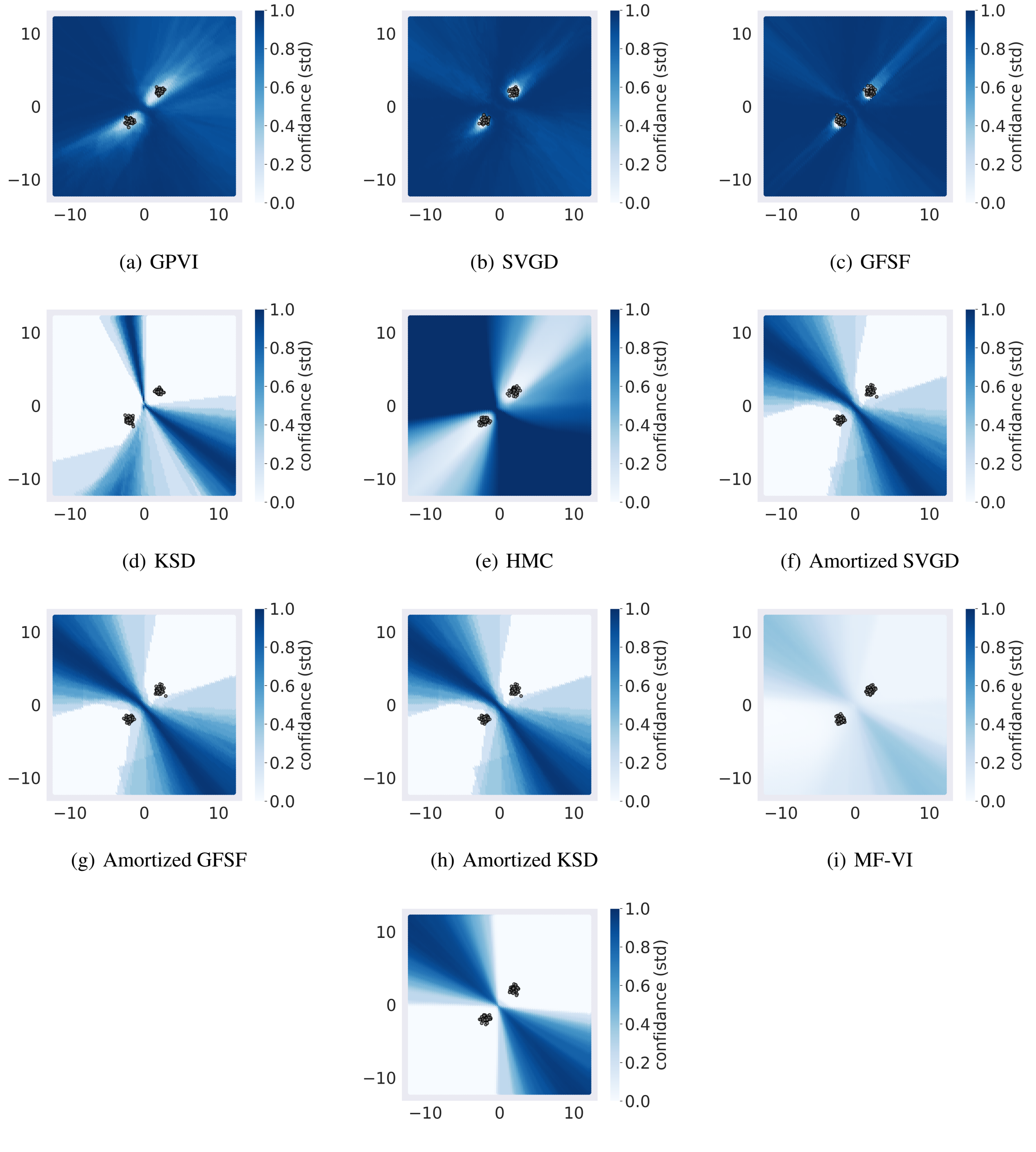
Density Estimation (energy potentials)	
Hyperparameter	Value
Common	
Posterior Samples	20000
Learning Rate	$1e-4$

$\boldsymbol{f}$ Hidden Layers	2
$\boldsymbol{f}$ Hidden Width	$[500, 500]$
$\boldsymbol{f}$ Input Noise Stdev	$\sigma \in \{1.0, 2.0, 6.0\}$
Training Steps	$200e3$
Minibatch Size	100
GPVI	
Optimizer	Adam
$\boldsymbol{h}_\eta$ Hidden Width	500
$\boldsymbol{h}_\eta$ Hidden Layers	3
$\boldsymbol{h}_\eta$ Learning Rate	$1e-4$
Normalizing Flow	
Optimizer	RMSProp
Flow Length	32
Weight Decay	$1e-3$
Base Dist Stdev	$\sigma \in \{1.0, 2.0, 6.0\}$
Flow Architecture	Planar Flow

Table 2: Hyperparameters for density estimation of energy potentials.

Bayesian Linear Regression	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	100
Learning Rate	$1e-3$
$\boldsymbol{f}$ Hidden Layers	None
$\boldsymbol{f}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_3)$
$\boldsymbol{g}$ Hidden Layers	None
Training Steps	$50e3$
Minibatch Size	10
GPVI	
$\boldsymbol{h}_\eta$ Hidden Width	10
$\boldsymbol{h}_\eta$ Hidden Layers	3
$\boldsymbol{h}_\eta$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	1
Critic Hidden Width	100
Critic Learning Rate	$1e-3$
Critic L_2 Weight	10
MF-VI (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	0

Table 3: Hyperparameters for Bayesian linear regression task



(a) GPVI (b) SVGD (c) GFSF (d) KSD (e) HMC (f) Amortized SVGD (g) Amortized GFSF (h) Amortized KSD (i) MF-VI (j) Deep Ensembles

Figure 7: Predictive uncertainty of each method on the 2-class classification task, as measured by the standard deviation between predictions of sampled functions.

4/2-class Classification	
Hyperparameter	Value
Common	
Optimizer (all)	Adam
Posterior Samples	100

Learning Rate	$1e-3$
Training Steps	500e3
Minibatch Size	100
$\boldsymbol{f}$ Hidden Layers	2
$\boldsymbol{f}$ Hidden Width	64
$\boldsymbol{f}$ Nonlinearity	ReLU
$\boldsymbol{f}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{64})$
$\boldsymbol{g}$ Hidden Layers	2
$\boldsymbol{g}$ Hidden Width	10
$\boldsymbol{g}$ Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\eta$ Hidden Width	544
$\boldsymbol{h}_\eta$ Hidden Layers	3
$\boldsymbol{h}_\eta$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	2
Critic Hidden Width	100
Critic Learning Rate	$1e-3$
Critic L_2 Weight	10
MF-VI (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

.
Table 4: Hyperparameters for 2/4 class classification task

Open-Category (MNIST)	
Common	
Optimizer (all)	Adam
Posterior Samples	10
Training Epochs	100
Minibatch Size	50
$\boldsymbol{f}$ Learning Rate	$1e-5$
$\boldsymbol{f}$ Hidden Layers	3
$\boldsymbol{f}$ Hidden Width	$[256, 512, 1024]$
$\boldsymbol{f}$ Nonlinearity	ReLU
$\boldsymbol{f}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{256})$
$\boldsymbol{g}$ Learning Rate	$1e-5$
$\boldsymbol{g}$ Architecture	LeNet-5
$\boldsymbol{g}$ Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\eta$ Hidden Width	512
$\boldsymbol{h}_\eta$ Hidden Layers	3
$\boldsymbol{h}_\eta$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	2
Critic Hidden Width	512
Critic Learning Rate	$1e-4$
Critic L_2 Weight	1.0

MF-VI (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

Table 5: Hyperparameters for MNIST open category task

Open-Category (CIFAR-10)	
Common	
Optimizer (all)	Adam
Posterior Samples	10
Training Epochs	200
Minibatch Size	50
$\boldsymbol{f}$ Learning Rate	$1e-5$
$\boldsymbol{f}$ Hidden Layers	3
$\boldsymbol{f}$ Hidden Width	$[400, 600, 1000]$
$\boldsymbol{f}$ Nonlinearity	ReLU
$\boldsymbol{f}$ Input Noise $\boldsymbol{z}$	$\mathcal{N}(0, I_{400})$
$\boldsymbol{g}$ Hidden Layers	3 conv, 2 linear
$\boldsymbol{g}$ Hidden Width	$[32, 64, 64, 128, 10]$
$\boldsymbol{g}$ Nonlinearity	ReLU
GPVI	
$\boldsymbol{h}_\eta$ Hidden Width	512
$\boldsymbol{h}_\eta$ Hidden Layers	3
$\boldsymbol{h}_\eta$ Learning Rate	$1e-4$
KSD (Amortized and ParVI)	
Critic Hidden Layers	2
Critic Hidden Width	512
Critic Learning Rate	$1e-4$
Critic L_2 Weight	1.0
MF-VI (Bayes by Backprop)	
Weight Prior	Scale Mixture
Mixture Weight π	0.5
σ_1	1.0
σ_2	$\exp(-6)$

Table 6: Hyperparameters for CIFAR-10 open category task

References

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

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.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

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

Saad, Y. *Iterative methods for sparse linear systems*. SIAM, 2003.