Neural Variational Gradient Descent

Lauro Langosco di Langosco [1] **Vincent Fortuin** [1] **Heiko Strathmann** [2]

Abstract

Particle-based approximate Bayesian inference approaches such as Stein Variational Gradient Descent (SVGD) combine the flexibility and convergence guarantees of sampling methods with the computational benefits of variational inference. In practice, SVGD heavily relies on the choice of an appropriate kernel function, which impacts its ability to model the target distribution—a challenging problem with only heuristic solutions. We propose Neural Variational Gradient Descent (NVGD), which is based on parametrizing the witness function of the Stein discrepancy by a deep neural network whose parameters are learned in parallel to the inference, mitigating the necessity to make any kernel choices. We empirically validate our method's performance on synthetic and real-world inference problems.

1. Introduction

This work is concerned with the problem of generating samples from an unnormalized Bayesian posterior. In particular, we are interested in the case where the target posterior is estimated from a large dataset, and only stochastic (mini-batch) approximations are available. In this setting, standard MCMC algorithms such as Hamiltonian Monte Carlo (Neal, 2012) or MALA (Roberts & Rosenthal, 1998) face difficulties (Betancourt, 2015; Roberts & Rosenthal, 1998). As a consequence, practitioners tend to prefer simpler methods such as variational inference (VI) and stochastic gradient Langevin dynamics (SGLD) (Welling & Teh, 2011). A more recent method developed by Liu & Wang (2016), Stein variational gradient descent (SVGD), has been gaining popularity.

These methods have in common that they minimize the KL divergence $\mathrm{KL}(q \parallel p)$ between an approximating distribution q and the target posterior p. VI does so explicitly by optimizing a parameterized density q_θ, thus necessitating the choice of a (more or less) limited variational family of densities, which can introduce bias. SGLD and SVGD do so non-parametrically by producing samples whose distribution follows a gradient-descent-like trajectory towards the target (Liu, 2017). While SGLD is a stochastic process, the Wasserstein gradient flow that it approximates can be viewed as a fixed trajectory in the space of probability distributions that depends only on the initialization q_0. This trajectory can also be approximated deterministically by evolving a set of particles using SVGD (Liu & Wang, 2016), which also allows to take the geometry of the target space into account. However, this geometry has to be encoded in a kernel function within the Stein class, which is often challenging to choose in practice.

In this work, we aim to overcome those limitations by learning an update function f_θ parameterized by a neural network that is trained in parallel to the inference. The samples are then updated via $x \leftarrow x + \varepsilon f_\theta(x)$. This approach has the following benefits:

- Unlike VI, our method does not constrain inference to a fixed family of distributions.
- Unlike SGLD, our method is deterministic and adapts to the geometry of the target posterior.
- Unlike SVGD, it is not dependent on a choice of kernel or kernel parameters, and thus can *automatically* adapt to the geometry of the target.

Empirically, we observe that our method performs at least as well as SVGD and SGLD across a range of tasks. In addition, we find that our method converges faster than Langevin dynamics and scales better to high dimensions than SVGD.

2. Inference via KL-Minimizing Flows

In the late 1990s, Jordan et al. (1998) famously showed that many variational schemes can be viewed as a gradient flow in the space of probability measures equipped with the Wasserstein metric. In this section we will give a very short overview of the results relevant for our setting.

Let $p \in \mathcal{P}(\mathbb{R}^n)$ be the target posterior we wish to sample

[1]ETH Zürich, Zürich, Switzerland [2]Deepmind, London, United Kingdom. Correspondence to: Lauro Langosco di Langosco <langosco.lauro@gmail.com>.

Presented at the ICML 2021 Workshop on Uncertainty and Robustness in Deep Learning.,

from. We start by casting the problem of sampling as an optimization problem: the goal is to generate samples distributed according to the solution q^* of the optimization problem

$$q^* = \operatorname*{argmin}_{q \in \mathcal{P}'} \mathrm{KL}(q \parallel p),$$

where is $\mathcal{P}'$ some appropriate space of candidate measures. Now consider the space $\mathcal{P}$ of probability measures on $\mathbb{R}^n$ equipped with the 2-Wasserstein metric. A Wasserstein gradient flow is a continuous trajectory $(q_t)_{t\geq 0}$ in $\mathcal{P}$ that locally minimizes the KL-divergence $\mathrm{KL}(q_t \parallel p)$. More details on Wasserstein gradient flows are available in textbooks (e.g., Villani, 2008; Ambrosio et al., 2008). In terms of samples $X_t \sim q_t$, this flow is characterized by the deterministic Markov process $(X_t)_{t\geq 0}$ given by

$$\frac{dX_t}{dt} = \nabla \log p(X_t) - \nabla \log q_t(X_t), \tag{1}$$

where $X_0 \sim q_0$ is initialized according to some initial distribution q_0, and q_t denotes the density of X_t. Under reasonable conditions this process converges in the sense that $\lim_{t\to\infty} \mathrm{KL}(q_t \parallel p) = 0$ (Vempala & Wibisono, 2019). The direct application of the Euler discretization

$$x_{k+1} = x_k + \varepsilon\Big(\nabla \log p(x_k) - \nabla \log q_k(x_k)\Big)$$

as a sampling method is intractable because it is usually not possible to efficiently compute $q_k(x)$.[1] Instead, the usual discretization is Langevin dynamics (SGLD), which Jordan et al. (1998) have shown to be equivalent (in the limit of infinitesimal step-size) to the flow (1) in the sense that the marginal distributions $(q_t)_{t\geq 0}$ are equal.

We take a different approach. By evolving many samples in parallel, we estimate the gradient using a neural network f_θ and then perform the update

$$x_{k+1} = x_k + \varepsilon f_\theta(x_k) \tag{2}$$

3. Neural Variational Gradient Descent

In this section, we will see how to train a neural network f_θ to approximate the gradient $\nabla \log p(x_k) - \nabla \log q_k(x_k)$. We build heavily on the method of Stein variational inference developed by Liu & Wang (2016). In a slightly non-standard manner, let us define the Stein discrepancy (Gorham & Mackey, 2018) given an $L^2(q)$-measureable vector field $f : \mathbb{R}^n \to \mathbb{R}^n$ as

$$\begin{aligned}\mathrm{SD}(q \parallel p; f) =& E_{x\sim q}[f(x)^T(\nabla \log p(x) - \nabla \log q(x))] \\ =& E_{x\sim q}[f(x)^T \nabla \log p(x) + \operatorname{div} f(x)],\end{aligned} \tag{3}$$

where the second equality follows via partial integration (details in Appendix A.1). Here, $\operatorname{div} f = \sum_{i=1}^d \frac{\partial f_i}{\partial x_i}$ denotes the divergence of f (the trace of f's Jacobian).

There are two properties of the Stein discrepancy that deserve special emphasis: Firstly, it can be estimated using only samples from q, without access to the explicit form of the density. Secondly, a regularized Stein discrepancy is maximized by the Wasserstein gradient from Equation 1. Indeed, following Grathwohl et al. (2020), let us define the *regularized Stein discrepancy* as

$$\mathrm{RSD}(f_\theta) = \mathrm{SD}(q \parallel p; f_\theta) - \frac{1}{2}\|f_\theta\|^2_{L^2(q)}, \tag{4}$$

where $\|f_\theta\|^2_{L^2(q)} = E_{x\sim q}[f_\theta(x)^T f_\theta(x)]$. Now it is easy to confirm that

$$f^* = \operatorname*{argmax}_{f \in L^2(q)} \mathrm{RSD}(q \parallel p; f) = \nabla \log p - \nabla \log q.$$

The regularization term is necessary because otherwise the optimization would scale f by an arbitrarily large factor.

3.1. Computation

Let $x_1, \ldots, x_n$ be a set of samples which we wish to transport towards the target posterior p. In this section, we describe how to compute a single iteration of our method; the entire procedure is summarized in Algorithm 1.

While it is intractable to compute $\mathrm{RSD}(f)$ exactly, we can approximate it with the Monte Carlo estimate

$$\begin{aligned}\widehat{\mathrm{RSD}}(f) = \frac{1}{n}\sum_{i=1}^n & f(x_i)^T \nabla \log p(x_i) + \operatorname{div} f(x_i) \\ & - \frac{1}{2} f(x_i)^T f(x_i).\end{aligned}$$

If the dimension d of the sample space is large, the $O(d^2)$ computation of the divergence $\operatorname{div} f$ might be too expensive. We follow Grathwohl et al. (2020) in using Hutchinson's estimator (Hutchinson, 1989), which gives an efficient and unbiased estimate of $\operatorname{div} f$ via

$$\operatorname{div} f(x) \approx z^T \nabla f(x) z, \quad z \sim \mathcal{N}(0, 1),$$

which is derived from the identity

$$\operatorname{div} f(x) = E_{z\sim\mathcal{N}(0,1)}[z^T \nabla f(x) z],$$

and has been used in a number of recent works (Grathwohl et al., 2020; Han et al., 2017; Grathwohl et al., 2018). One iteration of our method then consists of two steps: 1) train the model f_θ to maximize $\widehat{\mathrm{RSD}}(\theta)$ and 2) update the particles using the trained model. The full algorithm is described in Algorithm 1. To prevent the model from overfitting the particles $\{x_1, \ldots, x_n\}$, we track the value of the $\widehat{\mathrm{RSD}}$ on a validation set of particles and early stop the training if the validation $\widehat{\mathrm{RSD}}$ stops increasing.

[1] For an invertible, differentiable transformation T, the push-forward distribution of q under T can be computed via the change of variables formula $T_\# q(x) = q(T^{-1}x) \cdot \det |J_{T^{-1}}(x)|$. The computation of the determinant of a general $d \times d$ matrix needs $\mathcal{O}(d^3)$ operations; when d is large, this is too expensive.

Algorithm 1 Neural variational gradient descent

Input: Learning rates η and ε, number of gradient updates P
Initialize parameters θ and particles $x_1, \dots, x_n$
repeat
  $X_{\text{train}},\ X_{\text{validation}} \leftarrow \text{RandomSplit}(x_1, \dots, x_n)$
  for P steps **do**
    $\theta \leftarrow \theta + \eta \nabla_\theta \widehat{\text{RSD}}(\theta; X_{\text{train}})$
    if $\text{EarlyStopCondition}(X_{\text{validation}})$ **then**
      break
    end if
  end for
  $x_i \leftarrow x_i + \varepsilon f_\theta(x_i)$ for all $1 \le i \le n$
until converged

3.2. Computational Complexity

In large-data applications, the bottleneck of our method is the computation of the posterior $\nabla \log p$, which needs to be computed n times per step. The same bottleneck applies to competing methods such as SGLD, SVGD, and variational inference, which makes the methods similar as far as computational cost *per step* is concerned. In practice, we show empirically that our method converges in a similar or smaller number of steps than both SGLD and SVGD.

4. Related work

SVGD. Our method is based on the work of Liu & Wang (2016), who developed the SVGD algorithm. It has been previously noted that this algorithm often fails when the target distribibution is high-dimensional and has a complex shape. Attempted remedies include lower-dimensional projections (Zhuo et al., 2018; Gong et al., 2020; Chen & Ghattas, 2020) or modifications to the update equations (Gallego & Insua, 2018; Chewi et al., 2020; D'Angelo & Fortuin, 2021). The main difference between our method and these ones is that they only consider perturbations within an RKHS, thus requiring the explicit choice of a kernel function and restricting the flexibility of the method, while our method does not require any kernel choice whatsoever.

Learning the witness function. The approach used in this paper for learning the Stein discrepancy is modeled after the approach used by Grathwohl et al. (2020) to train and evaluate energy-based models. Another work that uses a similar approach to learn a Stein discrepancy is Hu et al. (2018). Both of these works directly minimize the Stein discrepancy between an approximating distribution (an energy-based model or a set of samples, respectively) and a target. In contrast, our method minimizes the KL-divergence by leveraging Equation 3, using an implicit gradient obtained via optimizing the Stein discrepancy.

5. Experiments

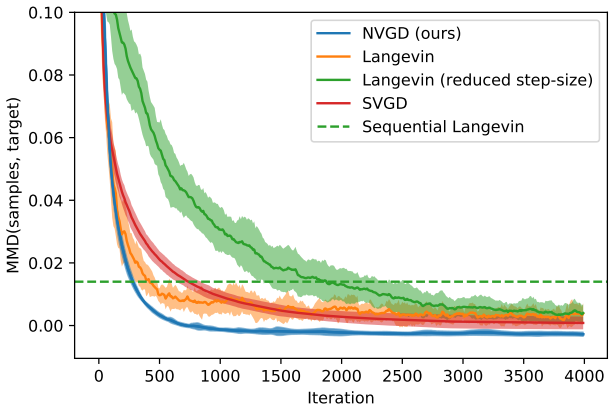

Figure 1. We compare NVGD, SVGD, and Langevin dynamics on the funnel density. If we use the same step-size for NVGD and Langevin, then the convergence is fast, but the asymptotic error is larger for Langevin (orange). When the step-size of Langevin is smaller, its asymptotic performance improves, but it converges more slowly (green). We plot the mean of ten runs and empirical standard deviation. The step-size of SVGD is tuned to be maximal while still converging to low MMD error.

In this section, we test our NVGD on various synthetic benchmarks and one large-data logistic regression task. More information on the technical setup is available in Appendix B, and ongoing efforts to scale NVGD to Bayesian neural network (BNN) inference are detailed in Appendix C.2.

For all experiments except those in Appendix C.2 we choose f_θ to be a neural network with two linear hidden layers of size 32 and an output layer of size d, where d is the dimensionality of the sample space.

We compare NVGD with two closely related sampling algorithms: SVGD (Liu & Wang, 2016) and the unadjusted Langevin algorithm (ULA). In the logistic regression and the BNN experiments, we estimate the posterior density using minibatches; in this setting, ULA is referred to as stochastic gradient Langevin dynamics (SGLD) (Welling & Teh, 2011).

For SVGD, one has to choose a kernel. The most frequent choice in the literature is the squared-exponential kernel $k(x, y) = \exp(-\|x-y\|^2/(2h^2))$ with bandwidth h chosen according to the median heuristic

$$h^2 = \frac{\text{med}^2}{2 \log n},$$

where med^2 is the median of all squared pairwise distances $\|x_i - x_j\|^2$ between samples $x_1, \dots, x_n$. We follow this choice in all experiments.

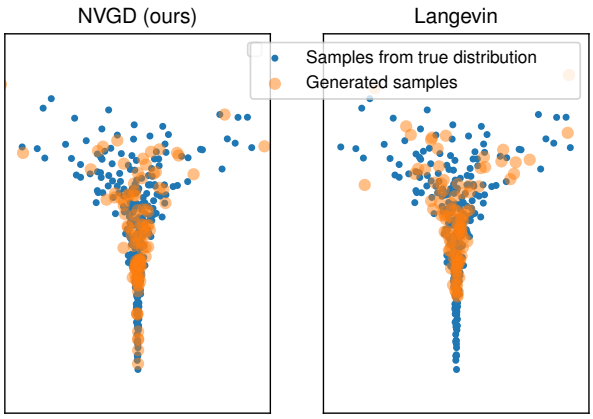

Figure 2. Samples generated for the Funnel density (same setting as Figure 1). The difficulty arises from the funnel geometry: wide at the top, and narrow at the bottom. In particular, Langevin dynamics (pULA) is not able to sample from the narrow strait if the step size is too large.

5.1. Synthetic benchmark distributions

Neal's Funnel. The Funnel density (Neal, 2003) is a common benchmark for sampling algorithms because it is easy to sample from directly (and thus easy to track convergence) but also challenging due to its geometry. The d-dimensional Funnel is defined for $x \in \mathbb{R}^d$ as the density

$$p(x) = \mathcal{N}(x_1; 0, 3) \cdot \prod_{i=2}^{d} \mathcal{N}(x_i; 0, \exp(x_1)). \qquad (5)$$

We choose $d = 2$ and initialize a set of 100 particles by sampling from a standard Gaussian and evolve them along the approximate gradient flows given by pULA and NVGD. We track convergence using the Maximum Mean Discrepancy (MMD) (Gretton et al., 2012) with a squared-exponential kernel. Figure 1 shows how the methods converge, measured by MMD, and Figure 2 is a scatterplot that visually compares how NVGD and SGLD perform, thus demonstrating why the funnel is a challenging target.

Higher-dimensional funnels. We test performance of NVGD for the Funnel distribution across 40 dimensions. Figure 3 shows how the performance of SVGD degrades with dimensionality, while that is not true for NVGD.

5.2. Bayesian logistic regression

We sample from a Bayesian logistic regression model trained on the binary Covertype dataset[2] (Dua & Graff, 2017). We compare again the three methods NVGD, SGLD, and SVGD, sampling 100 particles for each method. Test accuracy averaged over ten runs is reported in Figure 4.

[2]available at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

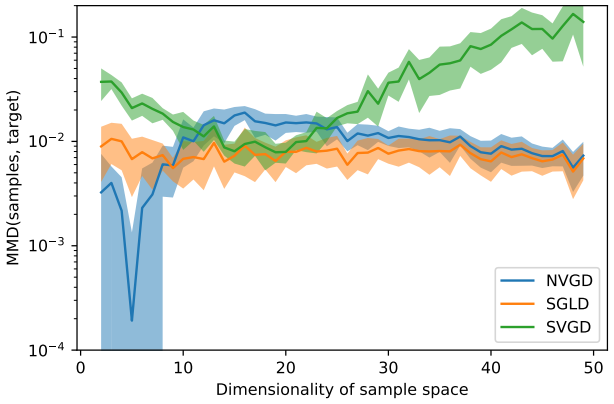

Figure 3. Results of sampling from Neal's Funnel across different numbers of dimenions. We plot the mean and standard deviation across ten runs.

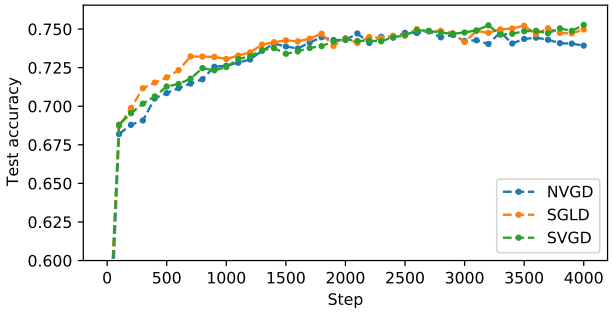

Figure 4. Bayesian logistic regression on the Covertype dataset. We use 100 particles and a mini-batch size of 128, so 4000 steps corresponds to one epoch. Results are averaged over ten runs.

6. Conclusion

We introduced NVGD, a novel particle-based variational inference method that deterministically transports particles from a reference distribution to the target posterior along a trajectory that (locally) minimizes the KL divergence. We have performed comparisons to the two most relevant competitor methods (SVGD and SGLD), and have demonstrated that our method performs as well as (or better than) them across different tasks.

References

Ambrosio, L., Gigli, N., and Savaré, G. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.

Betancourt, M. The fundamental incompatibility of scalable hamiltonian monte carlo and naive data subsampling. In *International Conference on Machine Learning*, pp. 533–540, 2015.

Chen, P. and Ghattas, O. Projected stein variational gradient descent. *arXiv preprint arXiv:2002.03469*, 2020.

Chewi, S., Le Gouic, T., Lu, C., Maunu, T., and Rigollet, P. Svgd as a kernelized wasserstein gradient flow of the chi-squared divergence. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, pp. 2098–2109, 2020.

D'Angelo, F. and Fortuin, V. Annealed stein variational gradient descent. *arXiv preprint arXiv:2101.09815*, 2021.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Gallego, V. and Insua, D. R. Stochastic gradient mcmc with repulsive forces. *arXiv preprint arXiv:1812.00071*, 2018.

Garriga-Alonso, A. and Fortuin, V. Exact langevin dynamics with stochastic gradients. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021. URL https://openreview.net/forum?id=Rprd8aVUYkE.

Gong, W., Li, Y., and Hernández-Lobato, J. M. Sliced kernelized stein discrepancy. *arXiv e-prints*, pp. arXiv–2006, 2020.

Gorham, J. and Mackey, L. Measuring Sample Quality with Stein's Method. *arXiv:1506.03039 [cs, math, stat]*, December 2018. URL http://arxiv.org/abs/1506.03039. arXiv: 1506.03039.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

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. *arXiv:2002.05616 [cs, stat]*, August 2020. URL http://arxiv.org/abs/2002.05616. arXiv: 2002.05616.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012. Publisher: JMLR. org.

Han, I., Malioutov, D., Avron, H., and Shin, J. Approximating spectral sums of large-scale matrices using stochastic chebyshev approximations. *SIAM Journal on Scientific Computing*, 39(4):A1558–A1585, 2017. Publisher: SIAM.

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

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989. Publisher: Taylor & Francis.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(Apr):695–709, 2005.

Jordan, R., Kinderlehrer, D., and Otto, F. The variational formulation of the Fokker–Planck equation. *SIAM journal on mathematical analysis*, 29(1):1–17, 1998. Publisher: SIAM.

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. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2378–2386. Curran Associates, Inc., 2016.

Neal, R. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 06 2012. doi: 10.1201/b10905-6.

Neal, R. M. Slice Sampling. *The Annals of Statistics*, 31 (3):705–741, 2003. ISSN 0090-5364. URL https://www.jstor.org/stable/3448413. Publisher: Institute of Mathematical Statistics.

Roberts, G. O. and Rosenthal, J. S. Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268, February 1998. ISSN 1369-7412, 1467-9868. doi: 10.1111/1467-9868.00123. URL http://doi.wiley.com/10.1111/1467-9868.00123.

Vempala, S. and Wibisono, A. Rapid convergence of the unadjusted langevin algorithm: Isoperimetry suffices. In *Advances in Neural Information Processing Systems*, pp. 8094–8106, 2019.

Villani, C. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

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.

Wenzel, F., Roth, K., Veeling, B. S., Świątkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. How good is the bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*, 2020.

Zhuo, J., Liu, C., Shi, J., Zhu, J., Chen, N., and Zhang, B. Message passing Stein variational gradient descent. In *International Conference on Machine Learning*, pp. 6018–6027. PMLR, 2018.

A. Proofs

A.1. The Stein Discrepancy

In Equation 3 we defined the Stein discrepancy given f as

$$\begin{aligned}\mathrm{SD}(q \parallel p; f) =& E_{x\sim q}[f(x)^T(\nabla \log p(x) - \nabla \log q(x))] \\ =& E_{x\sim q}[f(x)^T \nabla \log p(x) + \operatorname{div} f(x)].\end{aligned} \tag{6}$$

We will now provide the assumptions under which the second equality holds, and show how it follows from integration by parts.

Proposition A.1. *Let q be a probability density on $\mathcal{X}$ and $f : \mathcal{X} \to \mathbb{R}^d$, $f \in L^2(q)$ a differentiable function. Assume that*

$$\int_{\partial\mathcal{X}} f(x)q(x)\,\mathrm{d}x = 0, \tag{7}$$

where $\partial\mathcal{X}$ denotes the boundary of $\mathcal{X}$. If $\mathcal{X}$ is instead all of $\mathbb{R}^d$, then the condition must hold in the limit $r \to \infty$ for integral over the ball B_r of radius r centered at the origin. Then

$$E_{x\sim q}[\operatorname{div} f(x)] = -E_{x\sim q}[f(x)^T \nabla \log q(x)]$$

Proof. The proposition follows directly from integration by parts. If $n(x)$ is the outward pointing unit vector on the boundary of $\mathcal{X}$, then

$$\begin{aligned}E[f(x)^T \nabla \log q(x)] &= \int_{\mathcal{X}} f(x)^T \nabla q(x)\,\mathrm{d}x \\ &= \int_{\partial\mathcal{X}} f(x)^T n(x) q(x)\,\mathrm{d}x - \int_{\mathcal{X}} \operatorname{div} f(x) q(x)\,\mathrm{d}x \\ &= -E[\operatorname{div} f(x)].\end{aligned}$$

Our desired equality follows. □

Proposition A.1 is used in many places throughout statistics and deep learning. It is the basis of the policy gradient algorithm in reinforcement learning, and the score matching algorithm for density estimation (Hyvärinen, 2005).

B. Experiment details

For all experiments except those in Appendix C.2 we choose f_θ to be a neural network with two linear hidden layers of size 32 and an output layer of size d, where d is the dimensionality of the sample space.

B.1. Comparing against ULA and SVGD on Synthetic Targets

ULA and NVGD are discretizations (stochastic and deterministic, respectively) of the same gradient flow. It is known that in the limit of infinitesimally small step-size, ULA perfectly approximates the particle flow (1) that minimizes the KL divergence. Any hope of outperforming ULA must therefore lie in a better approximation of the flow *relative to the step-size*, that is, we need to show that ULA with a step-size $\eta > 0$ is either more biased than NVGD (if η is too large) or needs more steps to converge than NVGD (if η is too small).

We compare the methods on a synthetic sampling task where it is possible to track convergence exactly, and find that NVGD outperforms ULA. In particular, we compare with two versions of ULA: the standard single-chain ULA, and a parallel version pULA where n chains are initialized and developed simultaneously. The parallel version is more similar to NVGD, which also transports a number of particles in parallel. Results can be seen in Figures 1 and 2.

In many applications, ULA is augmented with an accept/reject step that adjusts for its asymptotic bias. We compare to the *unadjusted* version because the accept/reject step is not used in large-data applications (recent work by Garriga-Alonso & Fortuin (2021) has shown it possible to do so using custom integrators, but this is beyond the scope of this work). Instead,

the stochastic gradient variant of ULA (SGLD) is preferred, as for instance in Wenzel et al. (2020). Where an accept/reject step is feasible, NVGD can be similarly augmented, though we do not study this here.

Usually, ULA/SGLD is applied sequentially, that is, running only a single chain. To reduce correlation between samples, most are discarded (e.g., only every 100th sample is retained). To compare against this version of ULA, we develop a single chain for $5000 \cdot 100 = 5 \cdot 10^5$ steps, corresponding to the same number of likelihood evaluations as 5000 steps for 100 parallel chains. However, even after discarding all but every 100th sample, the resulting samples are still correlated. This results in a high MMD error, as visible in Figure 1.

B.2. Bayesian Logistic Regression

We train a Bayesian logistic regression model on the binary Covertype dataset (Dua & Graff, 2017). This dataset consists of $581,012$ observations of 55 features and a binary label. We split off a proportion of 0.2 for use as test set. We follow Liu & Wang (2016) in using a hierarchical prior:

$$\begin{aligned}
\alpha &\sim \text{Gamma}(a_0, b_0), \\
\beta &\sim \mathcal{N}(0, \alpha^{-1}), \\
y &\sim \text{Bernoulli}\left(\frac{\exp(x^\top \beta)}{1 + \exp(x^\top \beta)}\right),
\end{aligned}$$

choosing the rate and inverse scale parameters as $a_0 = 1$ and $b_0 = 0.01$. To test our method in the stochastic gradient setting, we estimate the likelihood using minibatches of size 128 and train for one epoch (4085 steps).

For all methods, we sample 100 particles and choose the particle step-size via tuning on a validation set of size 0.1 subsampled from the training set. We see in Figure 4 that all three methods perform similarly. The test accuracy is averaged over ten runs.

C. Further experiments

C.1. Minimizing the KL in one step

This experiment is a sanity check. We confirm that, on a synthetic task, our method approximates the true gradient of the KL, that is $f_\theta(x) \approx f^*(x) = \nabla \log p(x) - \nabla \log q(x)$. In setting we have access to both p and q (which is not the case in real applications) and can thus compute f^* directly. We confirm that our method successfully learns to approximate f^*.

Results are shown in Figure 5. For comparison, we also plot the kernelized Stein discrepancy, rescaled such that the kernelized (SVGD) update has L_2-norm equal to f^* (rescaling is necessary because the scale of the update, which corresponds to the particle step size, is a priori arbitrary).

We choose the target posterior p to be Gaussian in $\mathbb{R}^{50}$ with mean zero and a diagonal covariance matrix with entries spaced logarithmically from 10^{-4} to 1. This results in a severely ill-conditioned Gaussian.

We sample a batch of 1000 particles from a 'proposal' distribution q, here chosen to be a standard Gaussian, and train the network f_θ to maximize the regularized Stein discrepancy (4) for 1000 iterations. As visible in Figure 5, the network quickly learns to match the Wasserstein gradient.

C.2. Bayesian Neural Network Inference with NVGD

This section details ongoing experiments to scale NVGD to Bayesian neural network (BNN) inference. We train a Bayesian neural network on the MNIST dataset and again compare against SVGD and stochastic Langevin dynamics (SGLD). The Bayesian classifier network consists of two convolutional and one fully-connected layer comprising 4594 parameters in total. Results are shown in Figure 6.

For MNIST we use a batch size of 128. We tune the BNN learning rate to maximize accuracy on a validation set comprising 10% of the training set, and apply the Adam optimizer to the SVGD and NVGD updates (this cannot easily be done for the SGLD gradients, so we use vanilla SGLD. For comparison, we also include results for vanilla SVGD and NVGD).

While all three methods eventually reach similar accuracy, NVGD (our method) converges more slowly. We observe similar

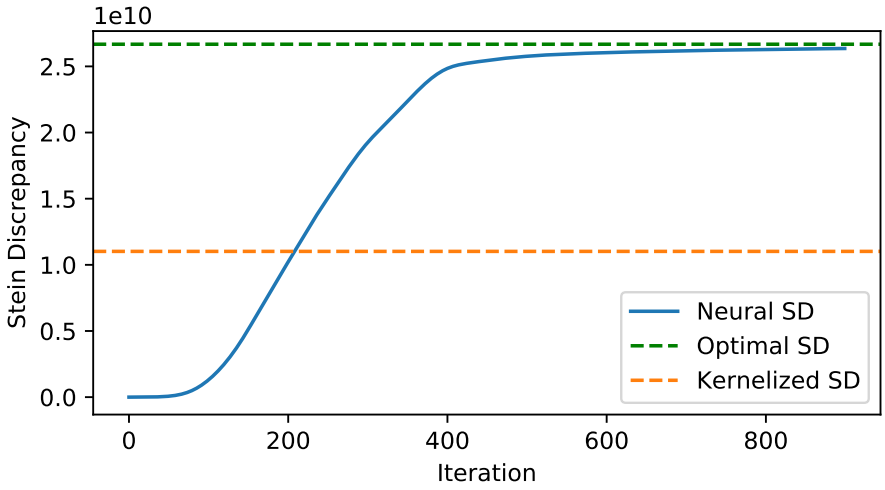

Figure 5. The Stein discrepancy over the course of gradient descent on θ. Larger is better, since the Stein discrepancy is proportional to the amount that one particle iteration reduces the KL divergence (Equation 3). Our proposed NVGD converges closely to the optimal Stein discrepancy.

shortcomings with respect to other relevant metrics such as out-of-distribution error. Our working hypothesis is that the Stein network f_θ underfits. While disappointing, it is arguably unsurprising that a simple MLP struggles to fit a challenging distribution on the CNN parameter space. We are currently performing tests of architectures more adapted to the geometry of the target space.

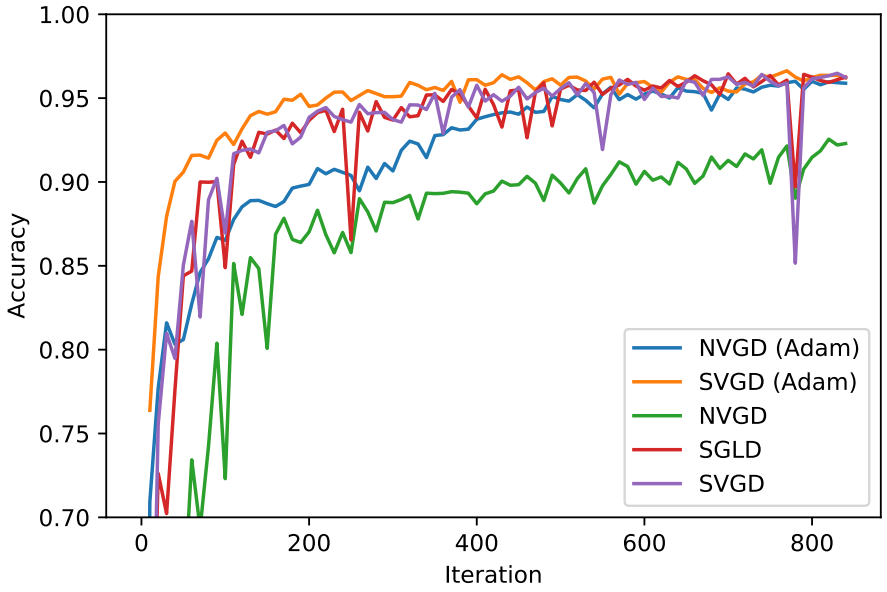

Figure 6. Bayesian neural network inference on MNIST. The vanilla version of our method (NVGD) performs poorly; when combined with the Adam optimizer, it performs better, but still converges more slowly than either SVGD and SGLD. One epoch corresponds to 420 iterations.