3rd Symposium on Advances in Approximate Bayesian Inference, 2020 1–13

Slice Sampling Reparameterization Gradients

Anonymous Authors
Anonymous Institution

Abstract

Slice sampling is a Markov chain Monte Carlo algorithm for simulating samples from probability distributions, with the convenient property that it is rejection-free. When the slice endpoints are known, the sampling path is a deterministic function of noise variables since there are no accept-reject steps like those in Metropolis-Hastings algorithms. Here we describe how to differentiate the slice sampling path to compute *slice sampling reparameterization gradients*. Since slice sampling does not require a normalizing constant, this allows for computing reparameterization gradients of samples from potentially complicated multivariate distributions. We apply the method in synthetic examples and to fit a variational autoencoder with a conditional energy-based model approximate posterior.

1. Introduction

Probabilistic objectives that take the form of an expectation of a function with respect to a base density $p_\theta(x)$, i.e.,

$$\mathcal{L}(\theta) = \mathbb{E}_{p_\theta(x)}[\ell(x)], \tag{1}$$

are common in machine learning. Typically, this expectation cannot be computed in closed form, so optimizing this objective with respect to the distributional parameters θ requires estimating the gradient of the expectation. Such stochastic gradients arise in a number of machine learning applications, such as optimizing the evidence lower bound (ELBO) in variational inference (Blei et al., 2017), forming the policy gradient in reinforcement learning (Sutton et al., 1998), and optimizing the probability of improvement in experimental design (Wilson et al., 2018). Two popular classes of gradient estimators are score function gradients and reparameterization (or pathwise) gradients; see Mohamed et al. (2019) for a review.

Reparameterization gradients apply when samples from $p_\theta(x)$ can be generated by a deterministic transformation f_θ of samples from a base distribution $p(\epsilon)$. That is, if $\tilde{\epsilon} \sim p(\epsilon)$ then $\tilde{x} = f_\theta(\tilde{\epsilon}) \sim p_\theta(x)$. Applying this transformation, the gradient of the objective with respect to the distributional parameters θ can then be expressed as an expectation with respect to a distribution that does not depend on θ: under mild regularity conditions,

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \mathbb{E}_{p(\epsilon)}[\ell(f_\theta(\epsilon))] = \mathbb{E}_{p(\epsilon)}[\nabla_\theta \ell(f_\theta(\epsilon))]. \tag{2}$$

Monte Carlo estimates of the gradient can then be computed using samples from $p(\epsilon)$.

Many examples of reparameterization gradients exist. "One-liner" reparameterization gradients use a simple function to transform the base distribution into the desired parameterized distribution (Kingma and Welling, 2013; Rezende et al., 2014; Mohamed et al., 2019). Next, the inverse CDF can be used to transform uniform random variables into arbitrary 1D distributions, and gradients can be computed when evaluating and differentiating

© A. Authors.

the inverse CDF are numerically tractable. Additional examples are implicit reparameterization gradients (Figurnov et al., 2018), doubly reparameterized gradients (Tucker et al., 2018), reparameterization of accept-reject sampling (Naesseth et al., 2016), and generalized reparameterization gradients (Ruiz et al., 2016). We focus here on continuous distributions, but there also exists a growing literature on such methods for discrete distributions.

In this work, we present a method to compute reparameterization gradients of samples generated from slice sampling. Slice sampling (Neal, 2003) is a Markov chain Monte Carlo (MCMC) algorithm for simulating samples from distributions $p_\theta(x) = \pi_\theta(x)/Z(\theta)$ where the normalizing constant $Z(\theta)$ may be unknown. Slice sampling is often an appealing alternative to the Metropolis-Hastings algorithm, as slice sampling does not require an accept/reject step or sensitive tuning parameters. The lack of an accept/reject step means that, for a fixed pseudo-random sequence, the realized slice sampling Markov chain is differentiable with respect to the parameters θ. Therefore, reparameterized slice sampling gradients apply to potentially complicated multivariate distributions such as energy-based models (EBM, LeCun et al., 2006). The tradeoff is that the samples are correlated and gradient estimates are biased because we simulate a finite Markov chain. Nonetheless, this opens reparameterization gradients to a wider class of probability distributions. In our experiments, we demonstrate the efficacy of slice sampling reparameterization gradients and use them to fit a conditional EBM as an approximate posterior in a variational autoencoder.

2. Slice sampling reparameterization gradients

Slice sampling is an MCMC algorithm for simulating samples from a distribution with density $p(\boldsymbol{x}) = \frac{1}{Z}\pi(\boldsymbol{x})$ that does not require the normalizing constant Z (Neal, 2003). The idea is to generate samples $(\boldsymbol{x}, y)$ uniformly under the surface $\pi(\boldsymbol{x})$ and then discard the samples y. Typical slice sampling algorithms proceed with two steps. Starting from an initial point $\boldsymbol{x}_n$, a height is sampled uniformly beneath the density at $\boldsymbol{x}_n$ such that $y_{n+1} \sim \mathcal{U}(0, \pi(\boldsymbol{x}_n))$. The height y_{n+1} defines a slice through the surface of the probability density given by $S = \{\boldsymbol{x} : y_{n+1} < \pi(\boldsymbol{x})\}$. The next point $\boldsymbol{x}_{n+1}$ is then sampled uniformly from the set S. In this work we use *random-direction slice sampling* to sample from the set S (MacKay, 2002, Chapter 29.7). In this method, a random direction $\boldsymbol{d}$ is generated from a uniform distribution over directions. The direction defines a line segment through the slice with endpoints $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$. Finally, a value $\boldsymbol{x}_{n+1}$ is sampled uniformly between the endpoints. The most common procedure for finding the endpoints, as proposed by Neal (2003), is to use a reversible "stepping-out" procedure followed by "interval shrinking" to determine $\boldsymbol{x}_{n+1}$; here we do not use these procedures and instead perform a direct search for the slice boundaries.

2.1. Random-direction slice sampling with numerical slice endpoints

Here we describe the steps of random-direction slice sampling in more detail. We start from a point $\boldsymbol{x}_n \in \mathbb{R}^d$ and a continuous, unnormalized density with parameters θ, $\pi_\theta(\boldsymbol{x})$. We sample three random quantities: two uniform random numbers $u_1, u_2 \in [0, 1]$ and a uniform random unit vector $\boldsymbol{d}$. The value u_1 determines the height of the slice $u_1\pi_\theta(\boldsymbol{x}_n)$. The direction $\boldsymbol{d}$ defines a line through the slice, on which there are points where $\pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d}) =$

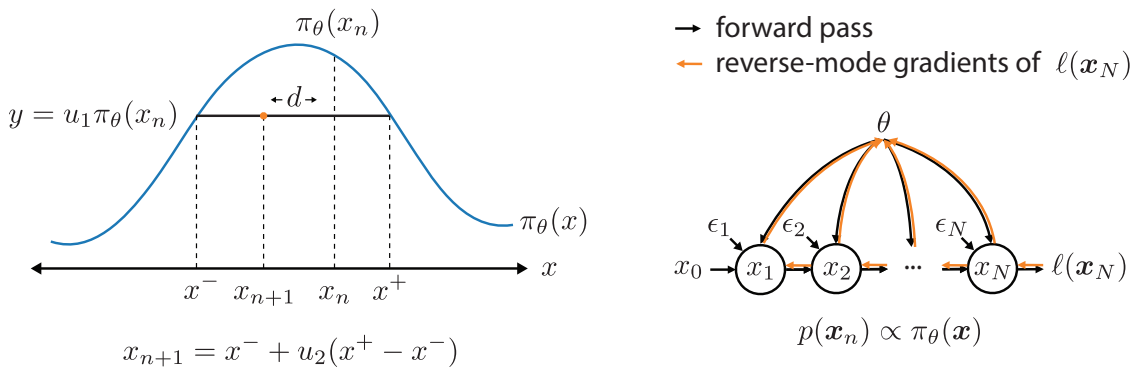

Figure 1: Slice sampling. *Left*: Visualization of one step of a 1D slice sampler. *Right*: Slice sampling computational graph with reverse mode gradients. The noise values ϵ_i are the values of u_1, u_2, and $\boldsymbol{d}$ used for generating sample $\boldsymbol{x}_i$. The reverse-mode gradients are shown for a loss computed on the final sample $\boldsymbol{x}_N$.

$u_1\pi_\theta(\boldsymbol{x}_n)$ for scalar values α. These correspond to a set

$$\mathcal{A} = \{\alpha : \pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d}) = u_1\pi_\theta(\boldsymbol{x}_n)\}\,. \tag{3}$$

Two specific members of this set, $\alpha^+ = \min_{\alpha>0}\mathcal{A}$ and $\alpha^- = \max_{\alpha<0}\mathcal{A}$, define the slice endpoints $\boldsymbol{x}^- = \boldsymbol{x}_n + \alpha^-\boldsymbol{d}$ and $\boldsymbol{x}^+ = \boldsymbol{x}_n + \alpha^+\boldsymbol{d}$. These correspond to the closest locations on the slice in the positive and negative direction where $\pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d}) = u_1\pi_\theta(\boldsymbol{x}_n)$. We identify α^+ and α^- using numerical root finding (Appendix E). Then, the next sample $\boldsymbol{x}_{n+1}$ is sampled uniformly between the endpoints $\boldsymbol{x}^-$ and $\boldsymbol{x}^+$ such that

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}^- + u_2(\boldsymbol{x}^+ - \boldsymbol{x}^-) = \boldsymbol{x}_n + u_2\alpha^+\boldsymbol{d} + (1-u_2)\alpha^-\boldsymbol{d}. \tag{4}$$

We can view α^- and α^+ as implicit functions of $\boldsymbol{x}_n$, $\boldsymbol{d}$, u_1, and θ, denoted $\alpha^+(\boldsymbol{x}_n, \boldsymbol{d}, u_1, \theta)$, and $\alpha^-(\boldsymbol{x}_n, \boldsymbol{d}, u_1, \theta)$. The location $\boldsymbol{x}_{n+1}$ is then a deterministic function of $\boldsymbol{x}_n$, θ, the random variables u_1, u_2, and $\boldsymbol{d}$:

$$\boldsymbol{x}_{n+1} = s(\boldsymbol{x}_n, \theta, u_1, u_2, \boldsymbol{d})\,. \tag{5}$$

In the next subsection, we show how to differentiate through this function to compute reparameterization gradients.

2.2. Reverse mode gradients

The slice sampling procedure generates samples $\boldsymbol{x}_{1:N}$. After generating the samples, we compute the objective function and use reverse mode automatic differentiation (AD) to compute gradients (Griewank and Walther, 2008, Section 3.2; Baydin et al., 2017). Each sample $\boldsymbol{x}_{n+1}$ depends on the previous samples $\boldsymbol{x}_{1:n}$ and the parameters θ, as shown by the computational graph (Figure 1). To compute gradients with respect to the parameters (and initial condition) we therefore require the Jacobian of $\boldsymbol{x}_{n+1}$ with respect to $\boldsymbol{x}_n$ and with

respect to θ. We denote these $\mathcal{J}_{\boldsymbol{x}_n}(\boldsymbol{x}_{n+1})$ and $\mathcal{J}_\theta(\boldsymbol{x}_{n+1})$. Using equation (4) for $\boldsymbol{x}_n$, these Jacobians can be computed indirectly via adjoints

$$\mathcal{J}_{\boldsymbol{x}_n}\left(\boldsymbol{x}_n + \boldsymbol{d}u_2\alpha^+ + \boldsymbol{d}(1-u_2)\alpha^-\right) = \boldsymbol{I} + u_2\boldsymbol{d}\nabla_{\boldsymbol{x}_n}[\alpha^+]^\mathsf{T} + (1-u_2)\boldsymbol{d}\nabla_{\boldsymbol{x}_n}[\alpha^-]^\mathsf{T} \tag{6}$$

$$\mathcal{J}_\theta\left(\boldsymbol{x}_n + \boldsymbol{d}u_2\alpha^+ + \boldsymbol{d}(1-u_2)\alpha^-\right) = u_2\boldsymbol{d}\nabla_\theta[\alpha^+]^\mathsf{T} + (1-u_2)\boldsymbol{d}\nabla_\theta[\alpha^-]^\mathsf{T}. \tag{7}$$

We use implicit differentiation to compute $\nabla_{\boldsymbol{x}_n}[\alpha^{+,-}]$. The values α are solutions to

$$f(\boldsymbol{x}_n, \boldsymbol{d}, \alpha, \theta) = \ln \pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d}) - \ln u_1 - \ln \pi_\theta(\boldsymbol{x}_n) = 0. \tag{8}$$

Applying implicit differentiation (Griewank and Walther, 2008, Chap. 15) we get:

$$\nabla_{\boldsymbol{x}_n}\alpha = -\frac{\nabla_{\boldsymbol{x}_n} f}{\partial f/\partial\alpha} = -\frac{\nabla_{\boldsymbol{x}_n}\ln\pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d}) - \nabla_{\boldsymbol{x}_n}\ln\pi_\theta(\boldsymbol{x}_n)}{\boldsymbol{d}^\mathsf{T}\nabla_{\boldsymbol{x}_n}\ln\pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d})} \tag{9}$$

$$\nabla_\theta\alpha = -\frac{\nabla_\theta f}{\partial f/\partial\alpha} = -\frac{\nabla_\theta\ln\pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d}) - \nabla_\theta\ln\pi_\theta(\boldsymbol{x}_n)}{\boldsymbol{d}^\mathsf{T}\nabla_{\boldsymbol{x}_n}\ln\pi_\theta(\boldsymbol{x}_n + \alpha\boldsymbol{d})}. \tag{10}$$

Let the loss be a sum over the samples $\boldsymbol{x}_n$ such that $L(\theta) = \frac{1}{N}\sum_{n=1}^N \ell(\boldsymbol{x}_n)$. The gradient of the loss with respect to θ is

$$\nabla_\theta L = \sum_{n=1}^N [\mathcal{J}_\theta(\boldsymbol{x}_n)]^\mathsf{T}\nabla_{\boldsymbol{x}_n} L. \tag{11}$$

The gradient $\nabla_\theta L$ depends on the gradients of the loss with respect to each sample, $\nabla_{\boldsymbol{x}_n} L$. For the final sample $\boldsymbol{x}_N$ this gradient is simply $\nabla_{\boldsymbol{x}_N} L = \frac{1}{N}\nabla_{\boldsymbol{x}_N}\ell(\boldsymbol{x}_N)$. For earlier samples $\boldsymbol{x}_n$ where $n = 1, ..., N-1$, the loss gradients need to be propagated backwards via

$$\nabla_{\boldsymbol{x}_n} L = \frac{1}{N}\nabla_{\boldsymbol{x}_n}\ell(\boldsymbol{x}_n) + [\mathcal{J}_{\boldsymbol{x}_n}(\boldsymbol{x}_{n+1})]^\mathsf{T}\nabla_{\boldsymbol{x}_{n+1}} L. \tag{12}$$

With the backpropagated gradients $\{\nabla_{\boldsymbol{x}_1} L, ...\nabla_{\boldsymbol{x}_N} L\}$ we can then compute the gradient following equation 11. Critically, the gradient $\nabla_\theta L$ can be computed without ever fully representing either of the Jacobians by using vector-Jacobian products (Appendix B). We implemented the forward sampling and reverse mode AD in JAX (Bradbury et al., 2018).

2.3. Gradient of the ELBO

Here we derive a gradient estimator of the ELBO when the approximate posterior is given by an unnormalized density $q_\phi(\boldsymbol{z}) = \frac{1}{Z(\phi)}e^{f_\phi(\boldsymbol{z})}$. The reparameterized ELBO is

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi(\boldsymbol{z})}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z})] = \mathbb{E}_{p(\epsilon)}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - \log q_\phi(\boldsymbol{z}(\phi, \epsilon))], \tag{13}$$

where $\boldsymbol{z}(\phi, \epsilon)$ is a sample generated from the slice sampler with noise ϵ and unnormalized density $f_\phi(\boldsymbol{z})$. Estimating the gradient appears to require the gradient of the normalizing constant $Z(\phi)$. However, applying the total derivative (Roeder et al., 2017) yields a Monte Carlo estimator of the gradient for a sample ϵ that does not require the normalizing constant:

$$\hat{\nabla}_{\mathrm{TD}}(\epsilon, \phi) = \nabla_{\boldsymbol{z}}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - f_\phi(\boldsymbol{z}(\phi, \epsilon))]\nabla_\phi \boldsymbol{z}(\phi, \epsilon). \tag{14}$$

See Appendix C for a full derivation. This estimator has favorable performance, and has zero variance when $p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) = q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ (Roeder et al., 2017). It may be surprising that we can compute low-variance gradients without the normalization constant. However, inspection of the pathwise gradient tells we can do just that since the path through $\boldsymbol{z}$ in the slice sampler does not depend on the normalization constant.

3. Experiments

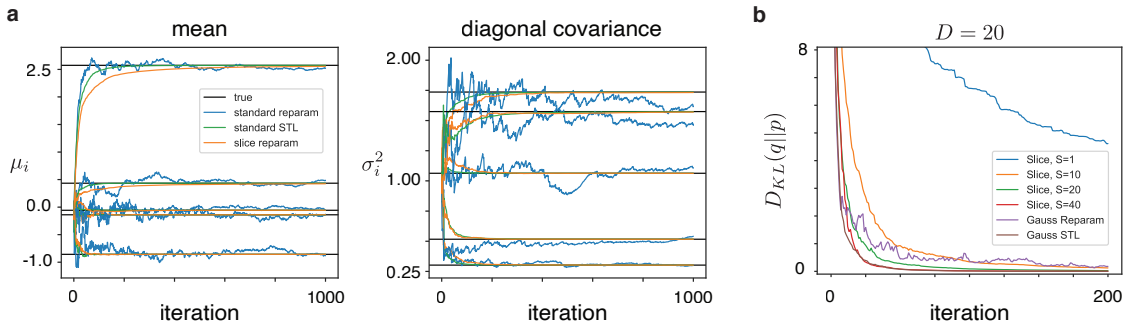

Figure 2: Matching a diagonal Gaussian. (**a**) Mean and diagonal covariance elements as a function of iteration number. (**b**) Loss as a function of iteration number for slice sampling reparameterization gradients computed using different chain lengths S.

Matching a target diagonal Gaussian In our first experiment, the objective is $\min_\phi D_{KL}(q_\phi||p)$, where q_ϕ is a diagonal multivariate Gaussian distribution with parameters ϕ and p is a known target diagonal Gaussian. First, we show that parameters optimized with slice sampling reparameterization gradients converge to the target values (Figure 2) for a low-dimensional example with $D = 5$. Next, we optimize a $D = 20$ diagonal Gaussian and compare performance with different chain lengths S. The gradient is computed by backpropagating the loss of the final sample. As S approaches D, slice sampling reparameterization gradients perform competitively with standard reparameterization gradients.

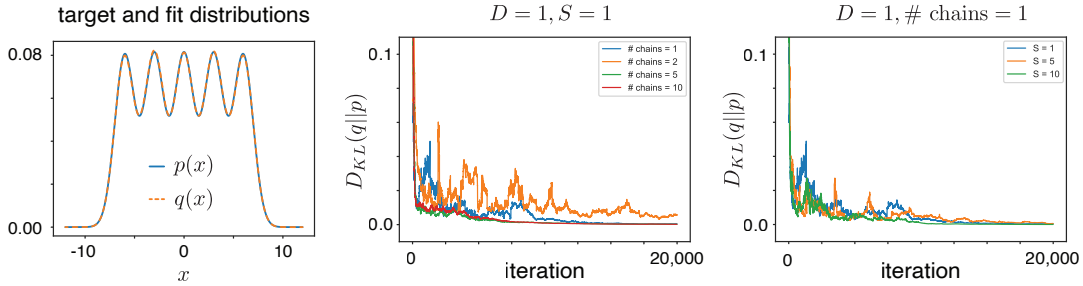

Figure 3: Fitting a mixture of Gaussians with variable numbers of samples and chains.

Matching a target mixture of Gaussians Next, we minimize $D_{KL}(q_\phi||p)$ with respect to ϕ where q and p are 1D mixtures of Gaussians (Figure 3). We estimate gradients using N independent chains each of length S samples. While performance with $S = 1$ sample from $N = 1$ MCMC chain is reasonable, increasing the number of independent chains N and the number of samples S improves performance. In conclusion, for multimodal distributions it may be advantageous to run multiple independent chains or increase the number of samples beyond the dimensionality of the space.

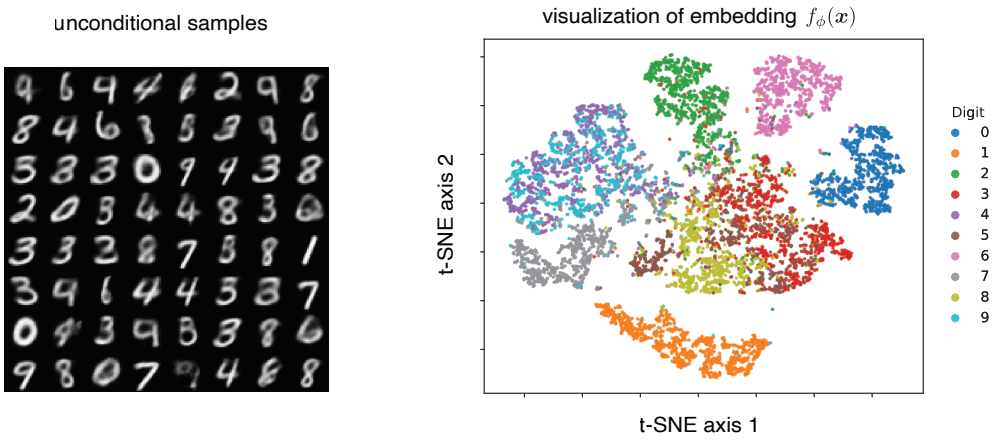

Figure 4: Unconditional samples (*left*) and embedding visualization (*right*) of a VAE fit to MNIST with an EBM approximate posterior.

VAE with conditional EBM approximate posterior We used slice sampling reparameterization gradients to fit a variational autoencoder (VAE) (Kingma and Welling, 2013; Rezende et al., 2014) with a conditional EBM as the approximate posterior:

$$q(\boldsymbol{z} \mid \boldsymbol{x}) = \frac{1}{Z(\theta, \boldsymbol{x})} \exp\{f_\phi(\boldsymbol{x})^\top g_\phi(\boldsymbol{z}) + \log p_0(\boldsymbol{z})\}. \tag{15}$$

The observations are $\boldsymbol{x}$, the latent variables are $\boldsymbol{z} \in \mathbb{R}^D$, and $\log p_0(\boldsymbol{z})$ is a prior. The functions $f_\phi(\boldsymbol{x})$ and $g_\phi(\boldsymbol{z})$ map to vectors with dimensionality D_e; we call this the embedding dimension. This differs from the standard VAE in that the encoder f does not map to the mean of the latent space, but rather a separate embedding that is nonlinearly combined by $g(\boldsymbol{z})$ to give the energy. The form of the conditional EBM has been studied previously in Khemakhem et al. (2019, 2020). We fit the VAE to MNIST data with a Bernoulli log likelihood and with $D = 20$ and $D_e = 50$. The gradients were estimated over mini batches of size 64 with one MCMC chain of $S = 100$ samples for each image. We ran the optimization for 200 epochs. Unconditional samples from the generative model look reasonable (Figure 4). We visualized the embedding of held out test images by the function f using t-SNE (Maaten and Hinton, 2008). Interestingly, we see clusters in the embedding space corresponding to digits, with some overlap for similar digits.

4. Discussion

We presented slice sampling reparameterization gradients that apply to multivariate distributions without a normalizing constant, and demonstrated the method with synthetic examples and by fitting a conditional EBM approximate posterior. In future work we will extend our experimental evaluation of the method, investigate methods to improve VAE training, and explore additional applications of the method.

References

Matthias Bauer and Andriy Mnih. Resampled priors for variational autoencoders. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 66–75. PMLR, 2019.

Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

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

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Bo Dai, Zhen Liu, Hanjun Dai, Niao He, Arthur Gretton, Le Song, and Dale Schuurmans. Exponential family estimation via adversarial dynamics embedding. In *Advances in Neural Information Processing Systems*, pages 10979–10990, 2019.

Adji Bousso Dieng, Dustin Tran, Rajesh Ranganath, John Paisley, and David Blei. Variational inference via χ upper bound minimization. In *Advances in Neural Information Processing Systems*, pages 2732–2741, 2017.

Mikhail Figurnov, Shakir Mohamed, and Andriy Mnih. Implicit reparameterization gradients. In *Advances in Neural Information Processing Systems*, pages 441–452, 2018.

Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi, and Kevin Swersky. Your classifier is secretly an energy based model and you should treat it like one. *arXiv preprint arXiv:1912.03263*, 2019.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1, page 2, 2016.

Ilyes Khemakhem, Diederik P Kingma, and Aapo Hyvärinen. Variational autoencoders and nonlinear ICA: A unifying framework. *arXiv preprint arXiv:1907.04809*, 2019.

Ilyes Khemakhem, Ricardo Pio Monti, Diederik P Kingma, and Aapo Hyvärinen. ICE-BeeM: Identifiable conditional energy-based deep models. *arXiv preprint arXiv:2002.11537*, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting Structured Data*, 1(0), 2006.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, USA, 2002. ISBN 0521642981.

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.

Christian A Naesseth, Francisco JR Ruiz, Scott W Linderman, and David M Blei. Reparameterization gradients through acceptance-rejection sampling algorithms. *arXiv preprint arXiv:1610.05683*, 2016.

Radford M Neal. Slice sampling. *Annals of statistics*, pages 705–741, 2003.

Erik Nijkamp, Mitch Hill, Tian Han, Song-Chun Zhu, and Ying Nian Wu. On the anatomy of MCMC-based maximum likelihood learning of energy-based models. *arXiv preprint arXiv:1903.12370*, 2019a.

Erik Nijkamp, Mitch Hill, Song-Chun Zhu, and Ying Nian Wu. Learning non-convergent non-persistent short-run MCMC toward energy-based model. In *Advances in Neural Information Processing Systems*, pages 5233–5243, 2019b.

Tom Rainforth, Adam R Kosiorek, Tuan Anh Le, Chris J Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. Tighter variational bounds are not necessarily better. In *ICML*, 2018.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Geoffrey Roeder, Yuhuai Wu, and David K Duvenaud. Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In *Advances in Neural Information Processing Systems*, pages 6925–6934, 2017.

Francisco JR Ruiz, Michalis K Titsias, and David M Blei. The generalized reparameterization gradient. In *Advances in neural information processing systems*, pages 460–468, 2016.

Tim Salimans, Diederik Kingma, and Max Welling. Markov chain Monte Carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226, 2015.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

George Tucker, Dieterich Lawson, Shixiang Gu, and Chris J Maddison. Doubly reparameterized gradient estimators for Monte Carlo objectives. In *International Conference on Learning Representations*, 2018.

Arash Vahdat, Evgeny Andriyash, and William G Macready. Undirected graphical models as approximate posteriors. In *International Conference on Machine Learning (ICML)*, 2020.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

James Wilson, Frank Hutter, and Marc Deisenroth. Maximizing acquisition functions for Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 9884–9895, 2018.

Appendix A. Related work

MCMC reparameterization gradients Some recent work has described computing gradients of samples generated via MCMC. Dynamics-based MCMC algorithms such as Stochastic Gradient Langevin Dynamics (Welling and Teh, 2011) that do not have accept-reject steps can be reparameterized (Salimans et al., 2015; Dai et al., 2019). More similarly to our work, Vahdat et al. (2020) propose to use unnormalized graphical models as approximate posteriors and describe reparameterized gradients from Gibbs sampling steps.

Variational autoencoders In VAEs with powerful generative models, it has been observed that the approximate posterior may collapse to the prior distribution (Bowman et al., 2015; Hoffman and Johnson, 2016). A number of papers have proposed methods to solve this problem and others related to VAE training. For example, Dieng et al. (2017) propose to add skip connections to the generative model and show that this helps avoid posterior collapse and improves the latent variable representation. Another approach is to increase the expressiveness of the prior distribution (e.g., Bauer and Mnih (2019)). We did not investigate if the VAE trained with a conditional EBM approximate posterior improved posterior collapse. However, we did show interesting structure given by the embedding function f. Similar to the encoder network of a standard VAE, which maps to the mean of a multivariate Guassian distributions, this function provides a low-dimensional representation of the input data. It will be interesting to compare the representations of these two approaches in follow up experiments.

Energy-based models Energy-based models are under increasingly active study. Grathwohl et al. (2019) relate classifiers with softmax outputs to EBMs, and propose a method for jointly training a classifier and generative model. Nijkamp et al. (2019b,a) describe using SGLD for learning deep EBMs for image data, and provide informative analysis on short run vs. long run MCMC.

Appendix B. Efficient computation

Importantly, we can compute $\nabla_\theta L$ without ever fully representing either of the two Jacobians $\mathcal{J}_{\boldsymbol{x}_n}(\boldsymbol{x}_{n+1})$ and $\mathcal{J}_\theta(\boldsymbol{x}_n)$. In the reverse mode gradient accumulation, the Jacobian $\mathcal{J}_{\boldsymbol{x}_n}(\boldsymbol{x}_{n+1})$ is always transposed and then post-multiplied by the gradient vector $\nabla_{\boldsymbol{x}_{n+1}} L$. Using the above formula for the Jacobian, this product is

$$[\mathcal{J}_{\boldsymbol{x}_n}(\boldsymbol{x}_{n+1})]^\mathsf{T} \nabla_{\boldsymbol{x}_{n+1}} L = [\boldsymbol{I} + u_2 \boldsymbol{d} \nabla_{\boldsymbol{x}_n}[\alpha^+]^\mathsf{T} + (1-u_2)\boldsymbol{d}\nabla_{\boldsymbol{x}_n}[\alpha^-]^\mathsf{T}]^\mathsf{T} \nabla_{\boldsymbol{x}_{n+1}} L \tag{16}$$

$$= \nabla_{\boldsymbol{x}_{n+1}} L + u_2 \nabla_{\boldsymbol{x}_n}[\alpha^+](\boldsymbol{d}^\mathsf{T} \nabla_{\boldsymbol{x}_{n+1}} L) + (1-u_2)\nabla_{\boldsymbol{x}_n}[\alpha^-](\boldsymbol{d}^\mathsf{T}\nabla_{\boldsymbol{x}_{n+1}} L). \tag{17}$$

Computing the products right to left only involves manipulating vectors. Next, the transpose of the Jacobian $\mathcal{J}_\theta(\boldsymbol{x}_n)$ is also always post-multiplied by $[\nabla_{\boldsymbol{x}_n} L]^\mathsf{T}$ such that

$$[\mathcal{J}_\theta(\boldsymbol{x}_n)]^\mathsf{T} \nabla_{\boldsymbol{x}_n} L = [u_2 \boldsymbol{d} \nabla_\theta[\alpha^+]^\mathsf{T} + (1-u_2)\boldsymbol{d}\nabla_\theta[\alpha^-]^\mathsf{T})]^\mathsf{T} \nabla_{\boldsymbol{x}_n} L \tag{18}$$

$$= u_2 \nabla_\theta[\alpha^+](\boldsymbol{d}^\mathsf{T}\nabla_{\boldsymbol{x}_n} L) + (1-u_2)\nabla_\theta[\alpha^-](\boldsymbol{d}^\mathsf{T}\nabla_{\boldsymbol{x}_n} L). \tag{19}$$

Appendix C. ELBO Gradient Derivation

Here we discuss optimizing the ELBO with an unnormalized energy-based model (EBM). Let the approximate posterior be given by an unnormalized density

$$q_\phi(\boldsymbol{z}) = \frac{1}{Z(\phi)} e^{f_\phi(\boldsymbol{z})}. \tag{20}$$

The reparameterized ELBO is

$$\begin{aligned}\mathcal{L}(\theta, \phi) &= \mathbb{E}_{q_\phi(\boldsymbol{z})}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z})] &(21)\\ &= \mathbb{E}_{p(\epsilon)}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - \log q_\phi(\boldsymbol{z}(\phi, \epsilon))] &(22)\end{aligned}$$

where $\boldsymbol{z}(\phi, \epsilon)$ is a sample generated from the slice sampler with noise ϵ and unnormalized density $f_\phi(\boldsymbol{z})$. The Monte Carlo estimate of the gradient given a sample ϵ is

$$\begin{aligned}\hat{\nabla}(\epsilon, \phi) &= \nabla_\phi[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - \log q_\phi(\boldsymbol{z}(\phi, \epsilon))] &(23)\\ &= \nabla_\phi[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - f_\phi(\boldsymbol{z}(\phi, \epsilon)) + \log Z(\phi)]. &(24)\end{aligned}$$

This appears to require the gradient of the normalizing constant $Z(\phi)$. We can estimate this gradient using samples generated from $q_\phi(\boldsymbol{z})$ via slice sampling. However, we take a different approach. Applying the total derivative (Roeder et al., 2017), we have

$$\begin{aligned}\hat{\nabla}_{\mathrm{TD}}(\epsilon, \phi) = \nabla_{\boldsymbol{z}}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - f_\phi(\boldsymbol{z}(\phi, \epsilon)) + \log Z(\phi)]\nabla_\phi \boldsymbol{z}(\phi, \epsilon) &(25)\\ + \nabla_\phi[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}) - \log q_\phi(\boldsymbol{z})]. &(26)\end{aligned}$$

The components $\nabla_{\boldsymbol{z}} \log Z(\phi)$ and $\nabla_\phi \log p_\theta(\boldsymbol{x}, \boldsymbol{z})$ are zero. Next, we can ignore $\nabla_\phi \log q_\phi(\boldsymbol{z})$ since it has mean zero. Removing these terms give us an estimator for the gradient that does not require the normalizing constant:

$$\hat{\nabla}_{\mathrm{TD}}(\epsilon, \phi) = \nabla_{\boldsymbol{z}}[\log p_\theta(\boldsymbol{x}, \boldsymbol{z}(\phi, \epsilon)) - f_\phi(\boldsymbol{z}(\phi, \epsilon))]\nabla_\phi \boldsymbol{z}(\phi, \epsilon). \tag{27}$$

This estimator has zero variance when $p_\theta(\boldsymbol{z} \mid \boldsymbol{x}) = q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ and favorable performance (Roeder et al., 2017). It may be surprising that we can compute low-variance gradients without the normalization constant. However, inspection of the pathwise gradient tells we can do just that since the path through $\boldsymbol{z}$ does not depend on the normalization constant.

Unfortunately, evaluation of the full ELBO *does* require the normalization constant of the approximate posterior, which is no longer available.

Appendix D. Additional experiment

We implemented the simulated inference network example from Rainforth et al. (2018), Section 4. In this example the generative model is

$$\boldsymbol{z} \sim \mathcal{N}(\mu, I), \quad \boldsymbol{x} \mid \boldsymbol{x} \sim \mathcal{N}(\boldsymbol{z}, I) \tag{28}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ are the observations and $\boldsymbol{z} \in \mathbb{R}^D$ are the latent variables. We fit the model by optimizing the ELBO with approximate posterior $q(\boldsymbol{z} \mid \boldsymbol{z}) = \mathcal{N}(A\boldsymbol{x}+b, \frac{2}{3}I)$. The parameters

are the generative parameters $\mu \in \mathbb{R}^D$ and the inference network parameters $A \in \mathbb{R}^{D\times D}$ and $b \in \mathbb{R}^D$. For a dataset $\{\boldsymbol{x}\}_{1:N}$, the optimal parameters (Rainforth et al., 2018) are

$$\mu^* = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i, \quad A^* = I/2, \quad b^* = \mu^*/2. \tag{29}$$

We simulated $N = 1000$ samples from this model and fit the generative and inference network parameters using slice sampling reparameterization gradients. The dimensionality was $D = 20$, the mini-batch size as 128, and we computed the slice sampling gradient using $S = 20$ samples after a 30 sample burn in. All parameters (true and fit) were initialized from standard multivariate Gaussian distributions. After optimizing for 1000 iterations, the fit parameters were very close to the optimal values (Figure 5).

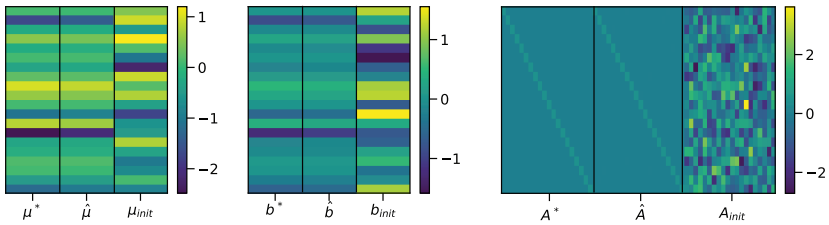

Figure 5: True ($*$), fit ($\hat{()}$), and initialized (*init*) parameters from the simulated inference network experiment (Rainforth et al., 2018).

Appendix E. Implementation details

As mentioned in the main text, we implemented the forward slice sampling with numerical root finding and reverse mode automatic differentiation in JAX Bradbury et al. (2018). Here we describe three primary components of our implementation.

Root finding The forward sampling process for the reparameterized slice sampling algorithm requires numerical root finding to identify the slice endpoints. This corresponds to finding the α^+ and α^- that satisfy

$$\pi(\boldsymbol{x}_n + \alpha \boldsymbol{d}; \theta) - u_1 \pi(\boldsymbol{x}_n\,;\,\theta) = 0. \tag{30}$$

Importantly, as mentioned above, we are interested in two specific roots: the roots $\alpha^- < 0$ and $\alpha^+ > 0$ that are closest to zero.

We used a standard bisection algorithm to identify these scalar roots. This corresponded to running two root finding operations; one bracketed below zero $\alpha^- \in [-a, \delta]$ and one bracketed above zero $\alpha^+ \in [\delta, a]$ where δ is a small positive constant. Notably, for multimodal distributions there may be multiple roots inside the brackets, and the bisection algorithm will only be guaranteed to find one of these roots (rather than the root closest to zero). Therefore we first identified the bracket value a with a stepping out procedure. We did this

by stepping out a logarithmically until the bracketing condition was satisfied. Once the condition was satisfied, we proceeded with the bisection algorithm.

We wrote the root finder using custom code in JAX such that it could be compiled using `jit` and batched using `vmap`. These two features are critical for speed. The `jit` compilation makes the root finding fast, and `vmap` makes it easy and fast to batch the root finding operation across multiple independent sampling chains.

Forward sampling The forwards sampling process takes as input random noise variables, the current parameters, and a function that computes the log probability of the distribution of interest (up to an additive constant). It proceeds with the slice sampling steps, using the root finding implementation described in the previous subsection to identify the roots. We also implemented this function to be compiled using `jit` and batched across multiple chains using `vmap`.

Reverse mode gradients We implemented the backwards pass using the efficient computations described in B. Again, we implemented this function to be compiled using `jit` and batched across multiple chains using `vmap`.