Resampled Priors for Variational Autoencoders

Matthias Bauer*
MPI for Intelligent Systems, Tübingen, Germany
University of Cambridge, Cambridge, UK

Andriy Mnih
DeepMind, London, UK

A VAE [7, 10] models the distribution of observations $\mathbf{x}$ using a stochastic latent vector $\mathbf{z}$, by specifying its prior $p(\mathbf{z})$ along with a likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ that connects it with the observation. VAEs are typically trained by maximizing the *evidence lower bound* (ELBO) on the marginal log-likelihood:

$$\text{ELBO}(\theta, \phi) = \mathbb{E}_{q(\mathbf{x})}\left[\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \text{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))\right] \tag{1}$$

where $q(\mathbf{x})$ is the empirical training data distribution and $q_\phi(\mathbf{z}|\mathbf{x})$ is the *variational posterior*. We can rewrite the KL in terms of the *marginal* or *aggregate posterior* $q(\mathbf{z}) = \mathbb{E}_{q(\mathbf{x})}[q(\mathbf{z}|\mathbf{x})]$ [4]:

$$\mathbb{E}_{q(\mathbf{x})} \text{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})) = \text{KL}(q(\mathbf{z})||p(\mathbf{z})) + I(\mathbf{x}; \mathbf{z}), \tag{2}$$

where the second term is the mutual information between the latent vector and the training example. Thus, maximizing the ELBO corresponds to minimizing the KL between the aggregate posterior and the prior. Despite this, there usually is a mismatch between the two distributions in trained models [4, 11], sometimes described as "holes in the aggregate posterior", referring to the regions of the latent space that have high density under the prior but very low density under the aggregate posterior. These regions are almost never encountered during training and decoded samples from these regions typically do not lie on the data manifold [11]. Our goal is to make the prior $p(\mathbf{z})$ more expressive so that it can approximate $q(\mathbf{z})$ better than the original prior.

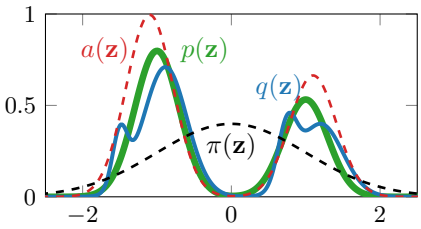


Figure 1: LARS approximates a target density $q(\mathbf{z})$ (——) by a resampled density $p(\mathbf{z})$ (——) obtained by multiplying a proposal $\pi(\mathbf{z})$ (- - -) with a learned acceptance function $a(\mathbf{z}) \in [0, 1]$ (- - -) and normalizing.

We achieve this by introducing Learned Accept/Reject Sampling (LARS) and using it to transform the original prior into a more expressive one. The proposed LARS prior is the result of applying a rejection sampler [15] with a learned acceptance function to the original prior, which now serves as the proposal distribution. The parameters of the acceptance function can be learned either jointly with all other model parameters by maximizing the ELBO, or separately by maximizing the likelihood of samples from the aggregate posterior of a trained model. Our approach is orthogonal to most existing approaches for making priors more expressive and can be easily combined with them by using them as proposals for the sampler. LARS is limited neither to variational inference nor to continuous variables and can be thought of as a generic method for transforming a tractable base distribution into a more expressive one by modulating its density while keeping learning and exact sampling efficient.

Learned Accept/Reject Sampling: LARS

Our goal is to approximate a complex target distribution $q(\mathbf{z})$ that we can sample from but whose density is either unknown or too expensive to evaluate. We start with an easy-to-sample-from base distribution $\pi(\mathbf{z})$ with a tractable density (with the same support as q), which is insufficiently expressive to approximate $q(\mathbf{z})$ well. We then transform $\pi(\mathbf{z})$ into a more expressive distribution $p_\infty(\mathbf{z})$ by multiplying it by an *acceptance probability*

$$p_\infty(\mathbf{z}) = \frac{\pi(\mathbf{z}) a_\lambda(\mathbf{z})}{Z} \tag{3}$$

*Work done during an internship at DeepMind.

Third workshop on Bayesian Deep Learning (NeurIPS 2018), Montréal, Canada.

function $a_\lambda(\mathbf{z}) \in [0, 1]$ and renormalizing to obtain the *resampled density* (Eq. (3)), where $Z = \int \pi(\mathbf{z}) a_\lambda(\mathbf{z}) \mathrm{d}\mathbf{z}$ is the normalization constant and λ are the learnable parameters of $a_\lambda(\mathbf{z})$, see Fig. 1. p_∞ is also the distribution we obtain by using rejection sampling [15] with $\pi(\mathbf{z})$ as the proposal and $a(\mathbf{z})$ as the acceptance probability. Thus, we can sample from it as follows: draw a candidate sample $\mathbf{z}$ from the proposal π and accept it with probability $a(\mathbf{z})$; if accepted, return $\mathbf{z}$ as the sample; otherwise repeat the process. The normalizing constant Z, can be interpreted as the average probability of accepting a candidate sample, which means that the average number of resampling steps until we accept a sample is given by $1/Z$. Our estimation of Z is based on Monte Carlo sampling: $Z \approx \frac{1}{S} \sum_s^S a(\mathbf{z}_s); \mathbf{z}_s \sim \pi$. During training, we smooth the per-batch MC estimators computed with 2^{10} samples by using exponential averaging, whereas for post-training evaluation we estimate Z to high accuracy using 10^{10} samples, see Appendix A.4 for details.

We also introduce a simple *truncated sampling scheme* that guarantees a minimum sampling efficiency: instead of allowing an arbitrary number of candidate samples to be rejected before accepting one, we cap this number and always accept the T^{th} sample if the preceding $T-1$ samples have been rejected. The corresponding density (Eq. (4) where $\alpha_T = (1-Z)^{T-1}$, see Appendix A.2 for the derivation) is a mixture of $p_\infty(\mathbf{z})$ and the proposal $\pi(\mathbf{z})$. While the truncated sampling density will in general be less expressive due to smoothing with π, we are guaranteed to accept at least one sample for each T candidates generated when sampling from it and thus can trade off approximation accuracy against sampling efficiency by varying the maximum number of resampling steps T.

$$p_T(\mathbf{z}) = (1-\alpha_T)\frac{a(\mathbf{z})\pi(\mathbf{z})}{Z} + \alpha_T \pi(\mathbf{z}) \quad (4)$$

Illustrative examples in 2D. We use LARS to fit a mixture of Gaussians (MoG) target distribution q in 2D, see Fig. 2 *(left)*. We train an acceptance function to reweight a standard Normal proposal distribution π to q by maximizing the supervised (maximum likelihood) objective in Eq. (5). A small MLP (2 layers of 10 units) for $a(\mathbf{z})$ is sufficient to separate out the individual modes almost perfectly (Fig. 2 (*left*)). However, an even simpler network (1 layer of 10 units) already learns to cut away the unnecessary tails of the proposal and leads to a better fit than the proposal (Fig. 2 (*middle*)). While samples from the higher capacity model are closer to the target (lower KL to the target), the sampling efficiency is lower (smaller Z). To obtain good fit without sacrificing sampling efficiency, we combine LARS with a RealNVP [3] proposal which is trained jointly with the acceptance function, see Fig. 2 (right). This approach isolates all modes but with double the efficiency of LARS with the standard normal proposal. Trained alone, RealNVP fails to isolate all the modes [5] ($\mathrm{KL}(q \,\|\, p_{\text{RealNVP}} = 0.68)$, see Fig. D.4).

$$\mathbb{E}_{q(\mathbf{z})}[\log p_\infty(\mathbf{z})] = \mathbb{E}_{q(\mathbf{z})}\left[\log \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}\right] \quad (5)$$

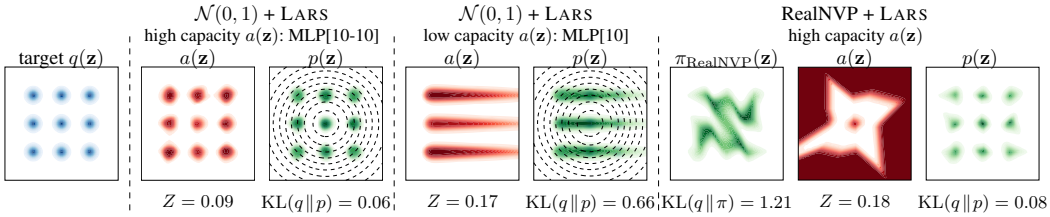


Figure 2: Learned rejection samplers $a(\mathbf{z})$ that approximate a fixed target $q(\mathbf{z})$ by reweighting a $\mathcal{N}(0,1)$ (- - -) or a RealNVP proposal to obtain an approximate density $p(\mathbf{z})$. As a reference, $\mathrm{KL}\big(q \,\big\|\, \pi_{\mathcal{N}(0,1)}\big) = 1.8$.

VAEs with Resampled Priors

We apply LARS to the prior of a VAE; that is, we use the original VAE prior as a proposal distribution $\pi(\mathbf{z})$ and define a *resampled prior* $p(\mathbf{z})$. The acceptance function is trained jointly with the other parts of the model by optimizing the ELBO objective (Eq. (1)). Thus, we replace the standard Normal distribution in the prior term, $\log p(\mathbf{z})$, by Eq. (3) for indefinite and by Eq. (4) for truncated resampling. Training a VAE with a resampled prior only requires evaluating the resampled prior density on samples from the variational posterior and generating a modest number of samples from the *proposal* to update the moving average estimate of Z as described above. Crucially, we never need to perform accept/reject sampling during training.

We evaluate LARS on a suite of standard datasets (static and dynamically binarized MNIST [9], Omniglot [8], and FashionMNIST [16]) using several architectures: (i) a VAE with single latent layer

and an MLP encoder/decoder (referred to as VAE); (ii) a VAE with two latent layers and an MLP encoder/decoder (HVAE); (iii) a VAE with two latent layers and a convolutional encoder/decoder (ConvHVAE). Inference networks for the hierarchical models were structured as in [13]. The acceptance function uses an MLP network (2 layers with 100 units each) and truncated resampling ($T = 100$, see Eq. (4)). Full details of all architectures can be found in Appendix C.2.

First, we compare the resampled prior to a standard Normal prior on several architectures, see Table 1. In all cases considered, ranging from a simple VAE to a hierarchical VAE with convolutional encoders/decoders, the resampled prior outperforms the standard prior. In most cases, the improvement in the NLL exceeds 1 nat and is notably larger for the simpler architectures.

	VAE ($L = 1$)		**HVAE** ($L = 2$)		**ConvHVAE** ($L = 2$)	
DATASET	standard	LARS	standard	LARS	standard	LARS
staticMNIST	89.11	**86.53**	85.91	**84.42**	82.63	**81.70**
dynamicMNIST	84.97	**83.03**	82.66	**81.62**	81.21	**80.30**
Omniglot	104.47	**102.63**	101.38	**100.40**	97.76	**97.08**
FashionMNIST	228.68	**227.45**	227.40	**226.68**	226.39	**225.92**

Table 1: Test negative log likelihood (NLL; *lower is better*) for different models with standard Normal prior ("standard") and our proposed resampled prior ("LARS"). L is the number of latent layers.

MODEL	dynMNIST	Omniglot
VAE ($L = 2$) + VGP [14]	81.32	
IWAE ($L = 2$) [1]		103.38
LVAE ($L = 5$) [12]	81.74	102.11
HVAE† ($L = 2$) + Vamp [13]	**81.24**	101.18
HVAE ($L = 2$) + LARS	81.62	**100.40**
VAE + IAF [6]	**79.10**	
ConvHVAE† + Vamp [13]	79.75	97.56
ConvHVAE + LARS	80.30	97.08

Figure 3 & Table 2: *left:* Ranked sample means *from the proposal* with jointly trained resampled prior. We drew 10^4 samples and show those with highest and lowest $a(\mathbf{z})$. *right:* Comparison of NLLs (†We use the same structure but a simpler architecture than [13]).

In Table 2 we compare LARS to other approaches that make the prior or variational posterior more expressive. LARS is competitive when applied to similar architectures, and we expect the results to improve further by using PixelCNN-style decoders, which have obtained state-of-the-art results on dynamic MNIST (78.45 nats for PixelVAE with VampPrior [13]) and Omniglot (89.83 with VLAE [2]). The improvements of LARS over the standard Normal prior are comparable to those obtained by the VampPrior [13], though they tend to be slightly smaller. LARS priors and VampPriors have somewhat complementary strengths. Sampling from VampPriors is more efficient as it amounts to (non-iterative) sampling from a mixture model. On the other hand, the VampPriors that achieve the above results use 500 or 1000 pseudo-inputs and thus have hundreds of thousands of parameters, whereas the number of parameters in the LARS priors is more than an order of magnitude lower. As LARS priors operate on a low-dimensional latent space they are also considerably more efficient to train than VampPriors, which are tied to the input space through the pseudo-inputs.

Our learned acceptance function assigns probability values $a(\mathbf{z})$ to all points in the latent space. This allows us to effectively *rank* draws from the proposal and to compare particularly high- to particularly low-scoring ones, see Figure 3 and Appendix D.5. Samples with $a(\mathbf{z}) \approx 1$ are visually pleasing and show almost no artefacts, whereas samples with $a(\mathbf{z}) \leq 10^{-10}$ look poor. When accepting/rejecting samples according to the resampling scheme, these low-quality samples would most likely be rejected. Thus, our approach is able to (i) automatically score (rank) samples, and (ii) reject low-quality samples.

Further results in Appendix D. We performed ablation studies to show that more expressive acceptance functions as well as later truncation lead to better model performance at the expense of additional computation and lower acceptance rate. Moreover, LARS can be similarly applied to already trained models with fixed encoders/decoders, or it can be combined with more expressive (non-factorial) proposals such as RealNVPs. It can even be applied to the discrete and high-dimensional outputs of a VAE with Bernoulli likelihood directly.

Acknowledgements

We thank Ulrich Paquet, Michael Figurnov, Guillaume Desjardins, Mélanie Rey, Jörg Bornschein, and Hyunjik Kim for their helpful comments and suggestions.

References

[1] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance weighted autoencoders". In: *Proceedings of the 4th International Conference on Learning Representations*. 2016.

[2] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Variational Lossy Autoencoder". In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. "Density estimation using Real NVP". In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[4] Matthew D. Hoffman and Matthew J. Johnson. "ELBO Surgery". In: *NIPS 2016 Workshop on Advances in Approximate Bayesian Inference* (2016).

[5] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. "Neural Autoregressive Flows". In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[6] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. "Improved Variational Inference with Inverse Autoregressive Flow". In: *Advances in Neural Information Processing Systems 29*. 2016.

[7] Diederik Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *Proceedings of the 2nd International Conference on Learning Representations*. 2014.

[8] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 6266 (2015).

[9] Hugo Larochelle and Iain Murray. "The Neural Autoregressive Distribution Estimator". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011.

[10] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning*. 2014.

[11] Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed. "Distribution Matching in Variational Inference". In: *arXiv* (2018).

[12] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. "Ladder Variational Autoencoders". In: *Advances in Neural Information Processing Systems 29*. 2016.

[13] Jakub Tomczak and Max Welling. "VAE with a VampPrior". In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. 2018.

[14] Dustin Tran, Rajesh Ranganath, and David M. Blei. "The Variational Gaussian Process". In: *Proceedings of the 4th International Conference on Learning Representations*. 2016.

[15] John von Neumann. "Various techniques used in connection with random digits". In: *Monte Carlo Method*. 1951.

[16] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 28, 2017.

Supplementary Material for "Resampled Priors for Variational Autoencoders"

A Derivations

A.1 Resampling *ad infinitum*

Here, we derive the density $p(\mathbf{z})$ for our rejection sampler that samples from a proposal distribution $\pi(\mathbf{z})$ and accepts a sample with learned acceptance probability given by $a(\mathbf{z})$.

By definition, the probability to sample a particular value $\mathbf{z}$ from the proposal is given by $\pi(\mathbf{z})$, such that the probability to sample and accept a particular z is given by the product $\pi(\mathbf{z})a(\mathbf{z})$. The probability to sample and reject the same $\mathbf{z}$ is given by the complementary probability $\pi(\mathbf{z})(1-a(\mathbf{z}))$.

Thus, the probability to accept *any* sample is just given by $\int \pi(\mathbf{z})a(\mathbf{z})\mathrm{d}\mathbf{z}$, whereas the probability to reject *any* sample is given by the complement $1 - \int \pi(\mathbf{z})a(\mathbf{z})\mathrm{d}\mathbf{z}$.

With this setup we can now derive the density $p(\mathbf{z})$ for sampling and accepting a particular sample $\mathbf{z}$. First, we consider the case, where we keep resampling until we finally accept a sample, if necessary indefinitely; that is, we derive the density $p_\infty(\mathbf{z})$ (Eq. (3) in the main paper). It is given by the sum over an infinite series of events: We could sample $\mathbf{z}$ in the first draw and accept it; or we could reject *any* sample in the first draw and subsequently sample and accept $\mathbf{z}$; and so on. As all draws are independent, we can compute the summation as a geometric series explicitly:

$$p_\infty(z) = \sum_{t=1}^{\infty} \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } t | \text{rejected previous } t-1 \text{ samples}\right) \Pr\left(\text{reject } t-1 \text{ samples}\right) \tag{A.1}$$

$$= \sum_{t=1}^{\infty} a(\mathbf{z})\pi(\mathbf{z}) \left(1 - \int \pi(\mathbf{z})a(\mathbf{z})\right)^{t-1} \mathrm{d}\mathbf{z} \tag{A.2}$$

$$= \pi(\mathbf{z})a(\mathbf{z}) \sum_{t=0}^{\infty} (1-Z)^t \qquad Z = \int \pi(\mathbf{z})a(\mathbf{z})\,\mathrm{d}\mathbf{z} \tag{A.3}$$

$$= \pi(\mathbf{z})a(\mathbf{z}) \frac{1}{1-(1-Z)} \tag{A.4}$$

$$= \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}. \tag{A.5}$$

Note that going from the second to third line we have redefined the index of summation to obtain the standard formula for the summation of a geometric series:

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x} \qquad \text{for } |x| < 1 \tag{A.6}$$

Thus, the log probability is given by:

$$\log p(z) = \log \pi(\mathbf{z}) + \log a(\mathbf{z}) - \log Z \tag{A.7}$$

As $0 \leq a(\mathbf{z}) \leq 1$ we find that $0 \leq Z \leq 1$.

A.2 Truncated Resampling

Next, we consider a different resampling scheme that we refer to as *truncated resampling* and derive its density. By truncation after the T^{th} step we mean that if we reject the first $T-1$ samples we accept the next (T^{th}) sample with probability 1 regardless of the value of $a(z)$.

Again, we can derive the density in closed form , this time by utilizing the formula for the truncated geometric series,

$$\sum_{n=0}^{N} x^n = \frac{1-x^{N+1}}{1-x} \qquad \text{for } |x| < 1 \tag{A.8}$$

$$p_T(\mathbf{z}) = \sum_{t=1}^{T} \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } t | \text{rejected previous } t-1 \text{ samples}\right) \Pr\left(\text{reject } t-1 \text{ samples}\right) \tag{A.9}$$

$$= \sum_{t=1}^{T-1} \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } t | \text{rejected previous } t-1 \text{ samples}\right) \Pr\left(\text{reject } t-1 \text{ samples}\right) \tag{A.10}$$

$$+ \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } T | \text{rejected previous } T-1 \text{ samples}\right) \Pr\left(\text{reject } T-1 \text{ samples}\right) \tag{A.11}$$

$$= a(\mathbf{z})\pi(\mathbf{z}) \sum_{t=0}^{T-2} (1-Z)^t + \pi(\mathbf{z})(1-Z)^{T-1} \tag{A.12}$$

$$= a(\mathbf{z})\pi(\mathbf{z}) \frac{1-(1-Z)^{T-1}}{Z} + \pi(\mathbf{z})(1-Z)^{T-1}. \tag{A.13}$$

Again, note that we have shifted the index of summation going from the second to third line. Moreover, note that $a(z)$ does not occur in the second term in the third line as we accept the T^{th} sample with probability 1 regardless of the actual value of $a(z)$.

The obtained probability density is properly normalized, that is, $\int p_T(\mathbf{z})\,\mathrm{d}\mathbf{z} = 1$, because $\int \pi(\mathbf{z})a(\mathbf{z})\,\mathrm{d}\mathbf{z} = Z$ and $\int \pi(\mathbf{z})\,\mathrm{d}\mathbf{z} = 1$.

Thus, the log probability is given by:

$$\log p_T(\mathbf{z}) = \log \pi(\mathbf{z}) + \log\left[a(\mathbf{z})\frac{1-(1-Z)^{T-1}}{Z} + (1-Z)^{T-1}\right] \tag{A.14}$$

As discussed in the main paper, p_T is a mixture of the untruncated density p_∞ and the proposal density π. We can consider the two limiting cases of $T = 1$ (accept every sample) and $T \to \infty$ (resample indefinitely, see Appendix A.1) and recover the expected results:

$$\log p_{T=1}(\mathbf{z}) = \log \pi(\mathbf{z}) \tag{A.15}$$

$$\lim_{T\to\infty} \log p_T(\mathbf{z}) = \log \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}. \tag{A.16}$$

That is, if we accept every sample ($T = 1$), we obtain the proposal, whereas if we sample *ad infinitum*, we converge to the result from above, Eq. (A.7).

A.3 Illustration of standard/classical Rejection Sampling

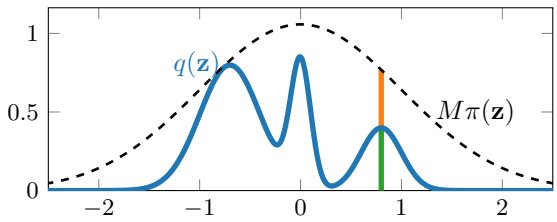


Figure A.1: In rejection sampling we can draw samples from a complicated target distribution (——) by sampling from a simpler proposal (- - -) and accepting samples with probability $a(\mathbf{z}) = \frac{q(\mathbf{z})}{M\pi(\mathbf{z})}$. The green and orange line are proportional to the relative accept and rejection probability, respectively.

A.4 Estimation of the normalization constant Z

Here we give details about the estimation of the normalization constant Z, which is needed both for updating the parameters of the resampled prior $p_T(\mathbf{z})$ as well as for evaluating trained models.

Model evaluation. For evaluation of the trained model, we estimate Z with a large number of Monte Carlo (MC) samples from the proposal:

$$Z = \frac{1}{S}\sum_{s}^{S} a(\mathbf{z}_s) \qquad \text{with } \mathbf{z}_s \sim \pi(\mathbf{z}). \tag{A.17}$$

In practice, we use $S = 10^{10}$ samples, and the evaluation of Z only takes on the order of several minutes on a GPU. The fast evaluation time is due to the relatively low dimensionality of the latent space, such that both drawing the samples as well as evaluating their acceptance function values $a(\mathbf{z})$ is fast and can be parallelized by using very large batch sizes of 10^5. For symmetric proposals, we also use antithetic sampling, that is, if we draw $\mathbf{z}$, then we also include $-\mathbf{z}$, which also corresponds to a valid/exact sample from the proposal. Antithetic sampling can reduce variance (see e.g. Section 9.3 of [14]).

In Fig. A.2 we show how the Monte Carlo estimates of the normalization Z typically evolves with the number of MC samples. We plot the estimate of Z as a function of the number of samples used to estimate it, S. That is

$$Z_S = \frac{1}{S}\sum_{i}^{S} a(\mathbf{z}_i) \qquad \text{with } \mathbf{z}_s \sim \pi(\mathbf{z}) \tag{A.18}$$

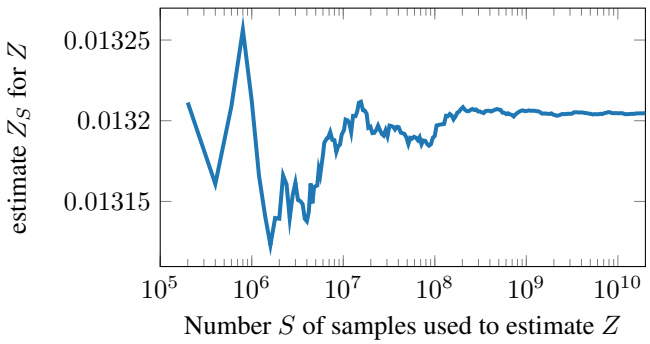


Figure A.2: Estimation of Z by MC sampling from the proposal. We plot the Z estimate as a function of the number of samples S used to estimate it, that is $Z_S = \frac{1}{S}\sum_{s}^{S} a(\mathbf{z}_s)$, with $\mathbf{z}_s \sim \pi(\mathbf{z})$. For few samples the estimate is quite noisy but converges to value of roughly 0.01321 after 10^9 samples.

Model training. During training we would like to draw as few samples from the proposal as possible in order not to slow down training. In principle, we could use the same Monte Carlo estimate as introduced above (Eq. (A.17)). However, when using too few samples, the estimate for Z can be very noisy, see Fig. A.2, and values for $\log Z$ are heavily biased. In practice, we introduce two means to get around these and related issues: (i) smoothing the Z estimate by using exponentially moving averages in the forward pass, and (ii) including the sample from $q(\mathbf{z}|\mathbf{x})$, on which we evaluate the resampled prior to compute the KL in the objective function, in the estimator. We now describe these two means in more detail.

During training we evaluate $\mathrm{KL}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))$ in a sampling fashion, that is, we evaluate $\log q(\mathbf{z}_r|\mathbf{x}_r) - \log p(\mathbf{z}_r)$ for each data point $\mathbf{x}_r$ in the minibatch and average the results. Here, $\mathbf{z}_r \sim q(\mathbf{z}|\mathbf{x}_r)$ is a sample from the approximate variational posterior that corresponds to data point $\mathbf{x}_r$. Thus, we need to evaluate the resampled prior $p(\mathbf{z}_r)$ on all samples $\mathbf{z}_r$ in that batch. The mean KL contribution of the prior terms is then given by:

$$\frac{1}{R}\sum_{r=1}^{R} \log p_T(\mathbf{z}_r) = \frac{1}{R}\sum_{r}^{R} (\log \pi(\mathbf{z}_r) + \log a(\mathbf{z}_r)) - \log Z \tag{A.19}$$

where, so far, Z is shared between all data points in the minibatch. To reduce the variability of Z as well as the bias of $\log Z$, we smoothen Z using exponentially moving averages. That is, given the

previous moving average from iteration i, $\langle Z \rangle_i$, as well as the current estimate of Z computed from S MC samples, Z_i, we obtain the new moving average as:

$$\langle Z \rangle_{i+1} = (1 - \epsilon) \langle Z \rangle_i + \epsilon Z_i \tag{A.20}$$

where ϵ controls how much Z is smoothed. In practice, we use $\epsilon = 0.1$ or $\epsilon = 0.01$. However, we can only use the moving average in the forward pass, as it is prohibitively expensive (and unnecessary) to backpropagate through all the terms in the sum. Therefore, we only want to backpropagate through the last term, which needs to be rescaled to account for ϵ. Before we show how we do this in practice, we introduce our second means to reduce variance: including the sample from $q(\mathbf{z}|\mathbf{x}_r)$ in the estimator.

For this, we introduce a datapoint dependent estimate Z_r, and replace the $\log Z$ term in Eq. (A.19) by:

$$\log Z \to \frac{1}{R} \sum_r^R \log Z_r \tag{A.21}$$

$$Z_r = \frac{1}{S+1} \left[\sum_s^S a(\mathbf{z}_s) + \frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r|\mathbf{x}_r)} a(\mathbf{z}_r) \right] \qquad \mathbf{z}_s \sim \pi(\mathbf{z}); \quad \mathbf{z}_r \sim q(\mathbf{z}|\mathbf{x}_r) \tag{A.22}$$

$$= \frac{1}{S+1} \left[S Z_S + \frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r|\mathbf{x}_r)} a(\mathbf{z}_r) \right] \tag{A.23}$$

where the sample $\mathbf{z}_r$ from the approximate posterior is reweighted by the ratio of prior and approximate posterior, similar to [1]. In practice, we do not want to backpropagate through this fraction, as this would alter the gradients for the encoder and potential parameters in the proposal. We are now in a position to write down the algorithm that computes the different Z_r values to use in the objective, see Algorithm 1.

Algorithm 1: Estimation of $\log Z$ (Eq. (A.21)) in the objective during training

input : Minibatch of samples $\{\mathbf{z}_r\}_r^R$ from $\{q(\mathbf{z}|\mathbf{x}_r)\}_r^R$; S samples $\{\mathbf{z}_s\}_s^S$ from π; previous moving average $\langle Z \rangle_i$

output : Updated moving average $\langle Z \rangle_{i+1}$; $\log Z$ estimate for current minibatch $\{\mathbf{x}_r\}_r^R$

1 $Z_S \leftarrow \frac{1}{S} \sum_s^S a(\mathbf{z}_s)$

2 **for** $r \leftarrow 1$ **to** R **do**

3 $\quad Z_{r,\text{curr}} \leftarrow \frac{1}{S+1} \left[S Z_S + \texttt{stop\_grad}\left(\frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r)} \right) a(\mathbf{z}_r) \right]$

4 $\quad Z_{r,\text{smooth}} \leftarrow (1 - \epsilon) \langle Z \rangle_i + \epsilon Z_{r,\text{curr}}$

5 $\quad Z_r \leftarrow Z_{r,\text{curr}} + \texttt{stop\_grad}(Z_{r,\text{smooth}} - Z_{r,\text{curr}})$

6 **end**

7 $\langle Z \rangle_{i+1} \leftarrow \frac{1}{R} \sum_r^R \texttt{stop\_grad}(Z_{r,\text{smooth}})$

8 $\log Z \leftarrow \frac{1}{R} \sum_r^R \log Z_r$

Note that in the forward pass, line 5 evaluates to $Z_{r,\text{smooth}}$, whereas in the backward direction (when computing gradients), it evaluates to $Z_{r,\text{curr}}$. Thus, we use the smoothed version in the forward pass and the current estimate for the gradients.

While Algorithm 1 estimates $\log Z$ used in the density of the indefinite/untruncated resampling scheme, we can easily adapt it to compute the truncated density as well, as we compute the individual Z_r, which can be used instead.

B Related Work

Expressive prior distribution for VAEs: Recently, several methods of improving VAEs by making their priors more expressive have been proposed. The VampPrior [25] is parameterized as a mixture of variational posteriors obtained by applying the encoder to a fixed number of learned pseudo-observations. While sampling from the VampPrior is fast, evaluating its density can be relatively expensive as the number of pseudo-observations used tends to be in the hundreds and the encoder needs to be applied on each one. A mixture of Gaussian prior [5, 21] provides a simpler alternative to the VampPrior, but is harder to optimize and does not perform as well [25]. Autoregressive models parameterized with neural networks allow specifying very expressive priors [e.g. 7, 9, 22] that support fast density evaluation but sampling from them is slow at it is an inherently sequential process. While sampling in LARS also appears sequential, it can be easily parallelized by considering a large number of candidates in parallel. Chen et al. [4] used autoregressive flows to define priors which are both fairly fast to evaluate and sample from. As flow-based approaches are based on smooth transformations of densities, the transformations involved need to be invertible and have efficient-to-evaluate Jacobians. Both requirements limit the power of any single transformation and thus necessitate chaining a number of them together. In contrast, LARS places no restriction on the parameterization of the acceptance function and can work well even with fairly small neural networks. On the other hand, evaluating models trained with LARS requires estimating a normalizing constant once using a large number of MC samples. As we showed above however, LARS and flows can be combined to give rise to models with better objective value as well as higher sampling efficiency.

Rejection Sampling: Variational Rejection Sampling [8] uses rejection sampling to make the variational posterior closer to the exact posterior; as it relies on evaluating the target density, it is not applicable to our setting. Naesseth et al. [20] derive reparameterized gradients for some not directly reparameterizable distributions by using rejection sampling to correct for sampling from a reparameterizable proposal instead of the distribution of interest. Similarly to VRS, this method requires being able to evaluate the target density.

Density Ratio Estimation (DRE): DRE [24] estimates a ratio of densities by training a classifier to discriminate between samples from them. It has been used to estimate the KL term in the ELBO in order to train VAEs with implicit priors [19], but the approach tends to overestimate the ELBO [23], making model comparison difficult. The acceptance probability function learned by LARS can be interpreted as a classifier that estimates the density ratio between the aggregate posterior and the proposal, which is trained end-to-end as a part of the generative process. Moreover, as the resampled prior has an explicit density, we can perform reliable model comparison by estimating the normalizing constant using a large number of samples.

Products of Experts: The density induced by LARS can be seen a special instance of a product-of-experts (PoE, [11]) architecture with two experts. However, while sampling from PoE models is generally difficult as it requires MCMC algorithms, our approach yields exact independent samples because of its close relationship to classical rejection sampling.

C Experimental Details

C.1 Datasets

The MNIST dataset [17] contains 50,000 training, 10,000 validation and 10,000 test images of the digits 0 to 9. Each image has a size of 28×28 pixels. We use both a dynamically binarized version of this dataset as well as the statically binarized version introduced by Larochelle and Murray [16].

Omniglot [15] contains 1,623 handwritten characters from 50 different alphabets, with differently drawn examples per character. The images are split into 24,345 training and 8,070 test images. Following Tomczak and Welling [25] we take 1,345 training images as validation set. Each image has a size of 28×28 pixels and we applied dynamic binarization to them.

FashionMNIST [26] is a recently proposed plug-in replacement for MNIST with 60,000 train/validation and 10,000 test images split across 10 classes. Each image has size of 28×28 pixels and we applied dynamic binarization to them.

C.2 Network architectures

In general, we decided to use very simple and standard architectures. We observed that more complicated networks can overfit quite drastically, especially on staticMNIST, which is not dynamically binarized.

Notation. For all networks, we specify the input size and then consecutively the output sizes of the individual layers separated by a " $-$ ". Potentially, outputs are reshaped to convert between convolutional layers (abbreviated by "CNN") and fully connected layers (abbreviated by "MLP"). When we nest networks, e.g. by writing $p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2) = \mathrm{MLP}[[\mathrm{MLP}[d_\mathbf{z} - 300], \mathrm{MLP}[d_\mathbf{z} - 300]] - 300 - 28 \times 28]$, we mean that first the two inputs/conditioning variables, $\mathbf{z}_1$ and $\mathbf{z}_2$ in this case, are transformed by neural networks, here an $\mathrm{MLP}[d_\mathbf{z} - 300]$, and their concatenated outputs (indicated by $[\cdot, \cdot]$) are subsequently used as an input to another network, in this case with hidden layer of 300 units and reshaped output 28×28.

VAE ($L = 1$). For the single latent layer VAE, we used an MLP with two hidden layers of 300 units each for both the encoder and the decoder. The latent space was chosen to be $d_\mathbf{z} = 50$ dimensional and the nonlinearities between the hidden layers was a `tanh`; the likelihood was chosen to be Bernoulli. The encoder parameterizes the mean μ and the log standard deviation $\log \sigma$ of the diagonal Normal approximate posterior.

$$
\begin{aligned}
q(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\mathbf{z}; \mu_\mathbf{z}(\mathbf{x}), \sigma_\mathbf{z}(\mathbf{x})) \\
p(\mathbf{x}|\mathbf{z}) &= \mathrm{Bernoulli}(\mathbf{x}; \mu_\mathbf{x}(\mathbf{z}))
\end{aligned}
$$

which are given by:

$$
\begin{aligned}
\mu_\mathbf{z}(\mathbf{x}) &= \mathrm{MLP}[28 \times 28 - 300 - 300 - d_\mathbf{z}] \\
\log \sigma_\mathbf{z}(\mathbf{x}) &= \mathrm{MLP}[28 \times 28 - 300 - 300 - d_\mathbf{z}] \\
\mu_\mathbf{x}(\mathbf{z}) &= \mathrm{MLP}[d_\mathbf{z} - 300 - 300 - 28 \times 28]
\end{aligned}
$$

where the networks for $\mu_z(\mathbf{x})$ and $\log \sigma_z(\mathbf{x})$ are shared up to the last layer. In the following, we will abbreviate this as follows:

$$
\begin{aligned}
q(\mathbf{z}|\mathbf{x}) &= \mathrm{MLP}[28 \times 28 - 300 - 300 - d_\mathbf{z}] \\
p(\mathbf{x}|\mathbf{z}) &= \mathrm{MLP}[d_\mathbf{z} - 300 - 300 - 28 \times 28]
\end{aligned}
$$

HVAE ($L = 2$). For the hierarchical VAE with two latent layers, we used two different architectures, one based on MLPs and the other on convolutional layers. Both were heavily influence by the architectural choices of Tomczak and Welling [25], however, we chose to use simpler models than them with fewer layers and regular nonlinearities instead of gated units to avoid overfitting.

The MLP was structured very similarly to the single layer case:

$$
\begin{aligned}
q(\mathbf{z}_2|\mathbf{x}) &= \mathrm{MLP}[28 \times 28 - 300 - 300 - d_\mathbf{z}] \\
q(\mathbf{z}_1|\mathbf{x}, \mathbf{z}_2) &= \mathrm{MLP}[[\mathrm{MLP}[28 \times 28 - 300], \mathrm{MLP}[d_\mathbf{z} - 300]] - 300 - d_\mathbf{z}] \\
p(\mathbf{z}_1|\mathbf{z}_2) &= \mathrm{MLP}[d_\mathbf{z} - 300 - 300 - d_\mathbf{z}] \\
p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2) &= \mathrm{MLP}[[\mathrm{MLP}[d_\mathbf{z} - 300], \mathrm{MLP}[d_\mathbf{z} - 300]] - 300 - 28 \times 28]
\end{aligned}
$$

We again use `tanh` nonlinearities between hidden layers of the MLP and a Bernoulli likelihood model

ConvHVAE ($L = 2$). The overall model structure is similar to the HVAE but instead of MLPs we use CNNs with strided convolutions if necessary. To avoid imbalanced up/down-sampling we chose a kernel size of 4 which works well with strided up/down-convolutions. As nonlinearities we used `ReLU` activations after convolutional layers and `tanh` nonlinearities after MLP layers.

$$
\begin{aligned}
q(\mathbf{z}_2|\mathbf{x}) &= \mathrm{MLP}[\mathrm{CNN}[28 \times 28 \times 1 - 14 \times 14 \times 32 - 7 \times 7 \times 32 - 4 \times 4 \times 32] - d_{\mathbf{z}}] \\
q(\mathbf{z}_1|\mathbf{x}, \mathbf{z}_2) &= \mathrm{MLP}[[\mathrm{CNN}[28 \times 28 \times 1 - 14 \times 14 \times 32 - 7 \times 7 \times 32 - 4 \times 4 \times 32], \mathrm{MLP}[d_{\mathbf{z}} - 4 \times 4 \times 32]] - 300 - d_{\mathbf{z}}] \\
p(\mathbf{z}_1|\mathbf{z}_2) &= \mathrm{MLP}[d_{\mathbf{z}} - 300 - 300 - d_{\mathbf{z}}] \\
p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2) &= \mathrm{CNN}[\mathrm{MLP}[[\mathrm{MLP}[d_{\mathbf{z}} - 300], \mathrm{MLP}[d_{\mathbf{z}} - 300]] - 4 \times 4 \times 32] - 7 \times 7 \times 32 - 14 \times 14 \times 32 - 32 \times 32 \times 1]
\end{aligned}
$$

Acceptance function $a(\mathbf{z})$. For $a(\mathbf{z})$ we use a very simple MLP architecture,

$$a(\mathbf{z}) = \mathrm{MLP}[d_{\mathbf{z}} - 100 - 100 - 1],$$

again with `tanh` nonlinearities between hidden layers and a `logistic` nonlinearity on the output to ensure that the final value is in the range $[0, 1]$.

RealNVP. For the RealNVP prior/proposal we employed the reference implementation from `TensorFlow Probablity` [18]. We used a stack of 4 RealNVPs with two hidden MLP layers of 100 units each and performed reordering permutations in-between individual RealNVPs, as the RealNVP only transforms half of the variables.

Masked Autoregressive Flows. For the Masked Autoregressive Flows [22] (MAF) prior/proposal we use a stack of 5 MAFs with $\mathrm{MLP}[100-100]$ each and again use the reference implementation from `TensorFlow Probablity`. Random permutations are employed between individual MAF blocks [22].

Datasets. We perform unsupervised learning on a suite of standard datasets: static and dynamically binarized MNIST [16], Omniglot [15], and FashionMNIST [26]. For more details about the datasets, see Appendix C.1.

C.3 Training and evaluation.

All models were trained using the Adam optimizer [13] for 10^7 iterations with the learning rate $3 \cdot 10^{-4}$, which was decayed to $1 \cdot 10^{-4}$ after 10^6 iterations, and mini-batches of size 128. Weights were initialized according to [6]. Unless otherwise stated, we used warm-up (KL-annealing) for the first 10^5 iterations [2]. We quantitatively evaluate all methods using the negative test log likelihood (NLL) estimated using 1000 importance samples [3]. We estimate Z using S MC samples (see Eq. (A.17)) with $S = 10^{10}$ for evaluation after training (this only takes several minutes on a GPU) and $S = 2^{10}$ during training. We repeated experiments for different random seeds with very similar results, so report only the means. We observed overfitting on static MNIST, so on that dataset we performed early stopping using the NLL on the validation set and halved the times for learning rate decay and warm-up.

D Additional Results

D.1 Illustrative examples in 2D

Here, we show larger versions of the images in Fig. 2 and also show the learned density of a RealNVP when used in isolation as a prior, p_{RealNVP}, see Fig. D.4. We find that the RealNVP by itself is not able to isolate all the modes, as has already been observed by Huang et al. [12], who observe a similar shortcoming for Masked Autoregressive Flows (MAFs).

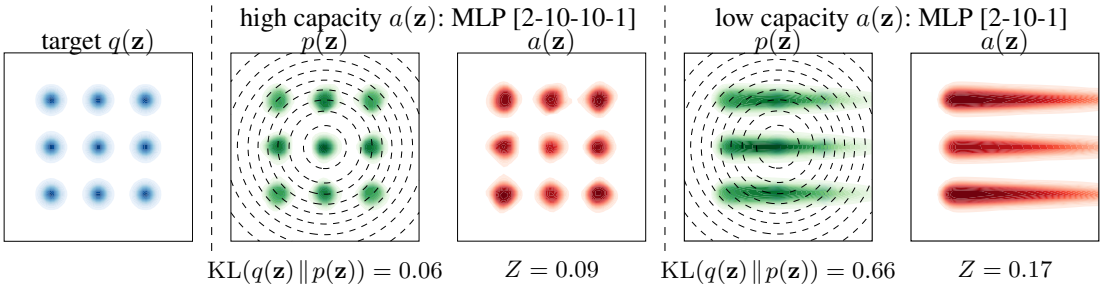


Figure D.3: Learned rejection samplers $a(\mathbf{z})$ (red) that approximate a fixed target q (blue) by reweighting a $\mathcal{N}(0, 1)$ proposal (- - -) to obtain an approximate density $p(\mathbf{z})$ (green). Darker values correspond to higher numbers.

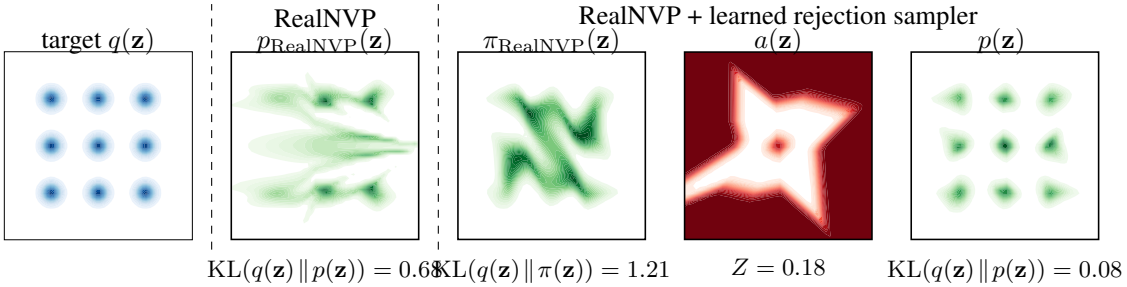


Figure D.4: Jointly trained RealNVP as proposal. The target is approximated either by a RealNVP alone (left) or a RealNVP in combination with a learned rejection sampler (right). Darker values correspond to higher numbers.

D.2 Post-hoc application of LARS to a pre-trained VAE

Applied *post-hoc* to a pretrained model (fixed encoder/decoder) the resampled prior almost reaches the performance of a jointly trained model, see Table D.1. This highlights that LARS can also be effectively employed to improve trained models. In this case, the gain in ELBO is entirely due to a reduction in the KL as the reconstruction error is determined by the fixed encoder/decoder. In contrast, joint training reduces both, and reaches a better ELBO, though the reduction in KL is smaller than for post-hoc training.

	standard $\mathcal{N}(0, 1)$				LARS *post-hoc*				LARS *joint training*			
MODEL	ELBO	KL	recons	Z	ELBO	KL	recons	Z	ELBO	KL	recons	Z
dynamicMNIST	89.0	25.8	63.2	1	87.3	24.1	63.2	0.023	86.6	24.7	61.9	0.015
Omniglot	110.8	37.2	73.6	1	108.8	35.2	73.6	0.015	108.4	35.9	72.5	0.011

Table D.1: ELBO and its components (KL and reconstruction error) for VAEs with: a standard Normal prior, a post-hoc fitted LARS prior, and a jointly trained LARS prior.

D.3 Ablation studies

Influence of network architecture. We investigated different network sizes for the acceptance function. As for the illustrative example, even a simple architecture can already notably improve over

MODEL	NLL	Z
VAE ($L = 1$, MLP)	84.97	1
VAE + LARS; $a = \text{MLP}[10 - 10]$	84.08	0.02
VAE + LARS; $a = \text{MLP}[50 - 50]$	83.29	0.016
VAE + LARS; $a = \text{MLP}[100 - 100]$	83.05	0.015
VAE + LARS; $a = \text{MLP}[50 - 50 - 50]$	83.09	0.015
VAE + LARS; $a = \text{MLP}[100 - 100 - 100]$	82.94	0.014

Table D.2: Test NLL and Z on dynamic MNIST. Different network architectures for $a(\mathbf{z})$ with $T = 100$.

MODEL	NLL	Z	KL	recons
VAE ($L = 1$, MLP)	84.97	1	25.8	63.3
VAE + LARS; $p_{T=2}$	84.60	0.19	25.4	63.2
VAE + LARS; $p_{T=5}$	84.11	0.12	25.1	63.0
VAE + LARS; $p_{T=10}$	83.71	0.08	25.0	62.6
VAE + LARS; $p_{T=50}$	83.24	0.03	24.8	62.1
VAE + LARS; $p_{T=100}$	83.05	0.014	24.7	61.9
VAE + LARS; p_∞	82.67	$2 \cdot 10^{-6}$	24.8	61.5

Table D.3: Influence of the truncation parameter on the test set of dynamic MNIST; $a = \text{MLP}[100 - 100]$.

a standard prior, with successively more expressive networks further improving performance, see Table D.2. Notably, the average acceptance probability Z is influenced much more by the truncation parameter than the network architecture. In practice, we choose $a = \text{MLP}[100 - 100]$ as more expressive networks only showed a marginal improvement. While we suspect that substantially larger networks will lead to overfitting, we did not observe this for the networks considered in Table D.2.

Influence of truncation. The truncation parameter T tunes the trade-off between sampling efficiency and approximation quality. As shown in Table D.3, the NLL improves as the number of steps before truncating increases, whereas the value for Z decreases accordingly. The gains in the NLL come both from the KL as well as the reconstruction term, though the relative contribution from the KL is larger. We stress that for truncated sampling, Z denotes the average acceptance rate *for the first* $T - 1$ *attempts* and does not take into account always accepting the T^{th} proposal. As a reference point, a model that is *trained* with indefinite (untruncated) resampling only accepts 2 out of 10^6 samples and estimating Z accurately thus requires a lot of samples.

MODEL	NLL	Z
VAE ($L = 1$) + $\mathcal{N}(0, 1)$ prior	84.97	1
VAE + RealNVP prior	81.86	1
VAE + MAF prior	81.58	1
VAE + LARS $\mathcal{N}(0, 1)$	83.04	0.015
VAE + LARS RealNVP	**81.15**	0.027

Table D.4: More expressive proposal and prior distributions. Test NLL on dynamic MNIST.

MODEL	NLL	Z
VAE ($L = 1$) + $\mathcal{N}(0, 1)$ prior	104.53	1
VAE + RealNVP prior	101.50	1
VAE + resampled $\mathcal{N}(0, 1)$	102.68	0.012
VAE + resampled RealNVP	100.56	0.018

Table D.5: Test NLL on dynamic Omniglot

Combination with non-factorial priors. So far, we have only considered VAEs with standard Normal priors. As stated before, our resampled prior is an orthogonal approach to specifying rich distributions and can be combined with other structured proposals such as non-factorial or autoregressive (flow) distributions. We investigated this using a RealNVP distribution as alternative proposal on

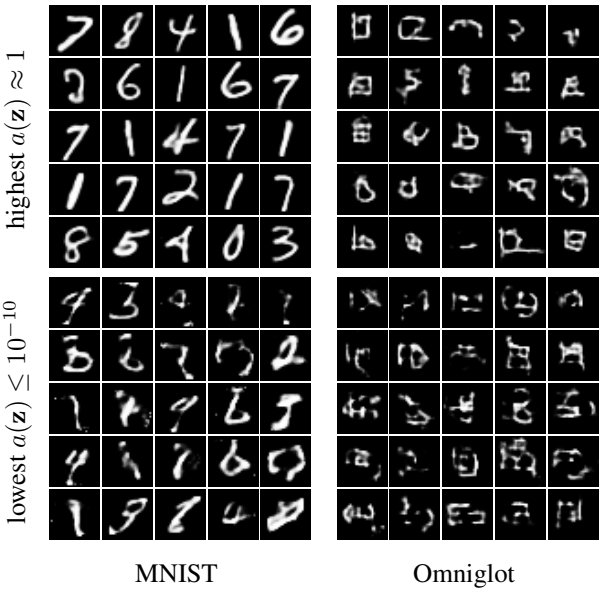


Figure D.5: Ranked sample means *from the proposal* of a VAE ($L = 1$, MLP) with jointly trained resampled prior. We drew 10^4 samples and show those with highest $a(\mathbf{z})$ (top) and lowest $a(\mathbf{z})$ (bottom).

MNIST (Table D.4) and Omniglot (Table D.5), see Appendix C.2 for details. The RealNVP prior by itself outperforms our resampling approach applied to a standard Normal prior. However, applying LARS to a RealNVP proposal distribution, we obtain a further improvement of more than 0.5 nats on both MNIST and Omniglot. A VAE trained with a more expressive Masked Autoregressive Flow (MAF) [22] as prior outperformed the RealNVP prior but was still worse than a resampled RealNVP. In practice it is hard to combine LARS with an MAF proposal due to slow sampling of the MAF.

D.4 Qualitative Results

Samples. For a qualitative analysis, we consider unconditional *samples* from a VAE with resampled prior. Our learned acceptance function assigns probability values $a(\mathbf{z})$ to all points in the latent space. This allows us to effectively *rank* draws from the proposal and to compare particularly high- to particularly low-scoring ones. We perform the same analysis twice, once for a resampled prior that is trained jointly with the VAE (Fig. D.5) and once for a resampled prior that is trained post-hoc for a fixed VAE (Fig. D.8), both times with very similar results: Samples with $a(\mathbf{z}) \approx 1$ are visually pleasing and show almost no artefacts, whereas samples with $a(\mathbf{z}) \leq 10^{-10}$ look poor. When accepting/rejecting samples according to our pre-defined resampling scheme, these low-quality samples would most likely be rejected. Thus, our approach is able to (i) automatically detect (rank) samples, and (ii) reject low-quality samples. For more samples, see Appendix D.5.

MODEL	NLL	Z
VAE ($L = 1$) + $\mathcal{N}(0, 1)$ prior	84.97	1
VAE + resampled $\mathcal{N}(0, 1)$	**83.04**	0.015
VAE with resampled output	83.63	0.014

Table D.6: Test NLL on dynamic MNIST for a VAE with *resampled outputs*, that is, the acceptance function acts in the discrete (pixel) space of the images directly.

Dimension-wise resampling. Instead of resampling the entire vector $\mathbf{z}$, a less wasteful sampling scheme would be to resample each dimension independently according to its own learned acceptance probability $a_i(z_i)$ (or in blocks of several dimension). However, such a factorized approach did not improve over a standard (factorized) Normal prior in our case. We speculate that it might be more effective when the target density has strong factorial structure, as might be the case for β-VAEs [10]; however, we leave these as future work.

D.5 More samples from jointly trained VAE models

We sampled 10^4 samples from the proposal $\pi(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, 1)$ of a jointly trained model and sorted them by their acceptance value $a(\mathbf{z})$. Fig. D.6 shows samples selected by their acceptance probability as assigned by the acceptance function $a(\mathbf{z})$. The top row shows samples with highest values (close to 1), which would almost certainly be accepted by the resampled prior; visually, these samples look very good. The middle row shows 25 random samples from the proposal, some of which look very good while others show visible artefacts. The corresponding acceptance values typically reflect this, see also Fig. D.7, in which we also show the $a(\mathbf{z})$ distribution for all 10^4 samples from the proposal sorted by their a value. The bottom row shows samples with the lowest acceptance probability (typically below $a(\mathbf{z}) \approx 10^{-10}$), which would almost surely be rejected by the resampled prior. Visually, these samples look very bad.

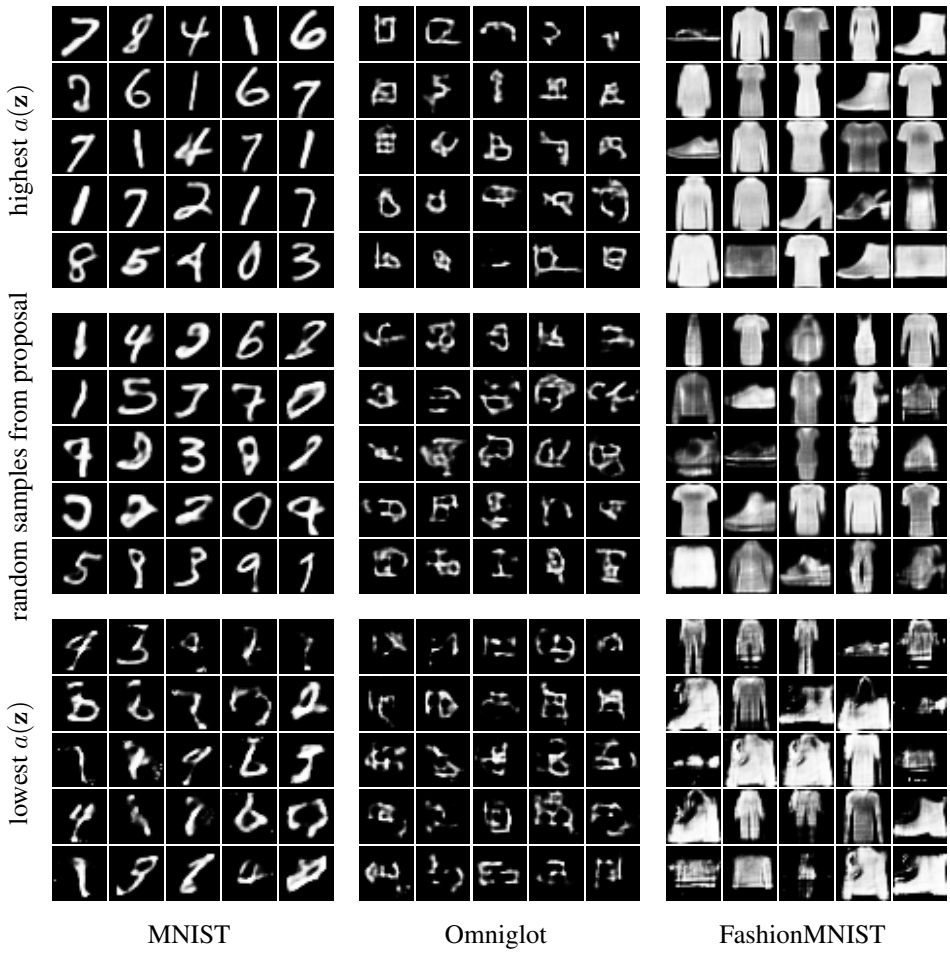


Figure D.6: Comparison of sample means from a VAE with MLP encoder/decoder and jointly trained resampled prior. Samples shown are out of 10^4 samples drawn. *top:* highest $a(\mathbf{z})$; *middle:* random samples from the proposal; *bottom:* lowest $a(\mathbf{z})$. All samples from the simple VAE model with MLP encoder/decoder.

D.6 Samples from applying LARS to a pretrained VAE model

Similarly to Appendix D.5, we now show samples from a VAE that has been trained with the usual standard Normal prior and to which we applied the resampled prior post-hoc, see Fig. D.8. We fixed the encoder and decoder and only trained the acceptance function on the usual ELBO objective.

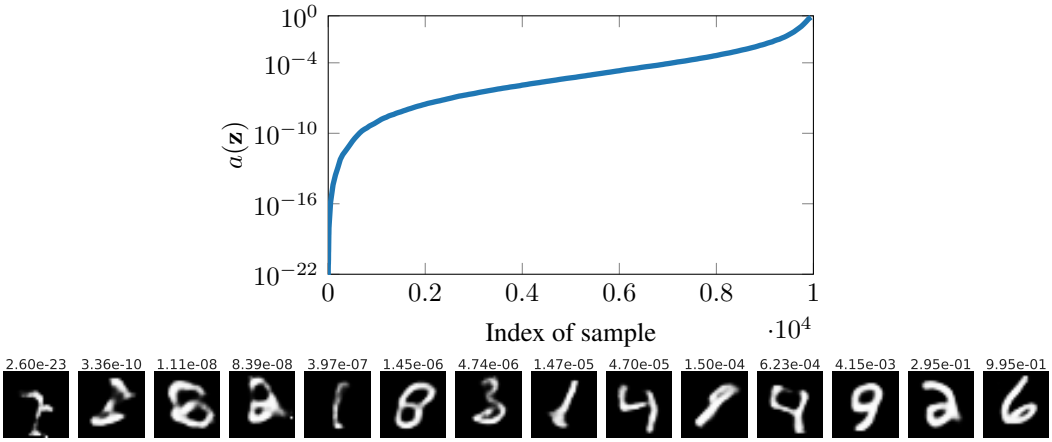


Figure D.7: Samples from the proposal of a jointly trained VAE model with resampled prior on MNIST and their corresponding acceptance function values. *top:* Distribution of the $a(\mathbf{z})$ values (index sorted by $a(\mathbf{z})$); note the log scale for $a(\mathbf{z})$. The average acceptance probability for this model is $Z \approx 0.014$. *bottom:* Representative samples and their $a(\mathbf{z})$ values. The $a(\mathbf{z})$ value correlates with sample quality.

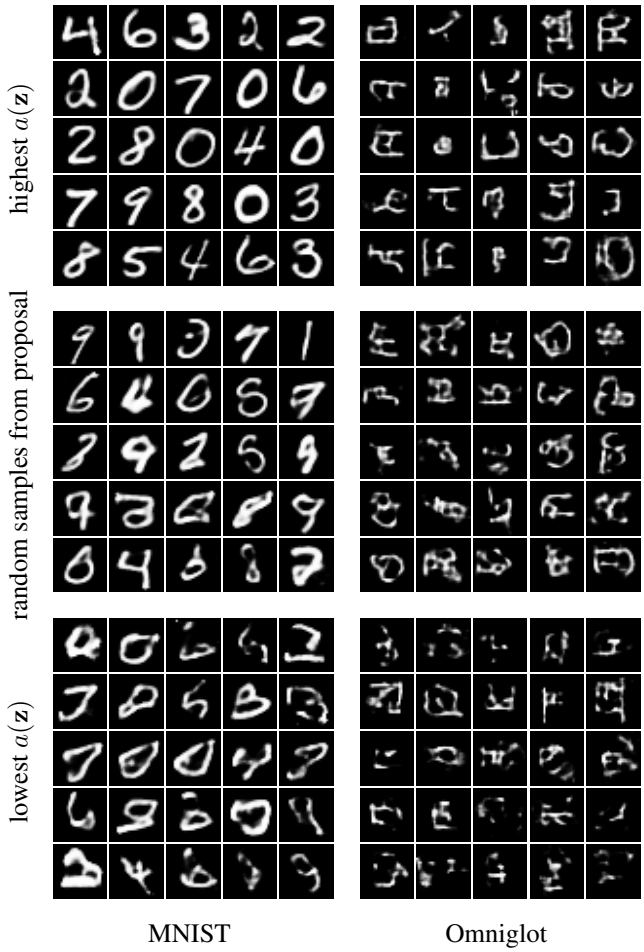


Figure D.8: Comparison of sample means from a VAE with pretrained MLP encoder/decoder to which we applied a resampled LARS prior post-hoc. Samples shown are out of 10^4 samples drawn. *top:* samples with highest $a(\mathbf{z})$; *middle:* random samples from the proposal; *bottom:* samples with lowest $a(\mathbf{z})$.

D.7 Resampling *discrete* outputs of a VAE

So far, we have applied LARS only to the continuous variables of the relatively low-dimensional latent space of a VAE. However, our approach is more general and can also be used to resample *discrete and high-dimensional* random variables. We apply LARS directly to the 784-dimensional binarized (Bernoulli) outputs of a standard VAE in the pixel space. That is, we make the marginal likelihood richer by resampling it in the same way as prior above: $\log p(\mathbf{x}) = \log \frac{\pi(\mathbf{x})a(\mathbf{x})}{Z}$, where $\pi(\mathbf{x})$ is now the original marginal likelihood, $a(\mathbf{x})$ is the acceptance function in pixel space, and Z is the normalization constant. We can lower-bound the enriched marginal likelihood via its ELBO:

$$\mathbb{E}_{q(\mathbf{z}|\mathbf{x})} \log \pi(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})) + \log \frac{a(\mathbf{x})}{Z} \tag{D.24}$$

In practice, we use the density that corresponds to truncated sampling instead. $\mathbf{x}$ corresponds to the training image under consideration, whereas Z is estimated on decoded samples from the (regular) prior $p(\mathbf{z})$:

$$Z = \int \pi(\mathbf{x})a(\mathbf{x})\mathrm{d}\mathbf{x} \approx \frac{1}{S}\sum_{s}^{S} a(\mathbf{x}_s) \tag{D.25}$$

with $\mathbf{x}_s \sim \pi(\mathbf{x}|\mathbf{z}_s)$ and $\mathbf{z}_s \sim p(\mathbf{z})$. We used the same MLP encoder/decoder as in our previous experiments, but increased the size of the $a(\mathbf{z})$ network to match the architecture of the encoder; we also employed truncated resampling with $T = 100$ as before.

Resampling the outputs of a VAE directly is slower than resampling the prior for several reasons: (i) we need to decode all prior samples $\mathbf{z}_s$, (ii) a larger $a(\mathbf{x})$ has to act on a higher dimensional space, and (iii) the estimation of Z typically requires more samples to be accurate (we used $S = 10^{11}$). These are also the reasons why we apply LARS in the lower-dimensional latent space to enrich the prior in all other experiments in this paper.

Still, LARS works fairly well in this case. In terms of NLL, the VAE with resampled outputs outperforms the original VAE by over 1 nat (Table D.6) and is only about 0.6 nats worse than the VAE with resampled prior. In Fig. D.9 we show ranked samples for a VAE with resampled marginal log likelihood, that is, we apply LARS on the *discrete* output space rather than in the continuous latent space. We find that even in this high dimensional space, our approach works surprisingly well, and $a(\mathbf{x})$ is able to reliably rank images.

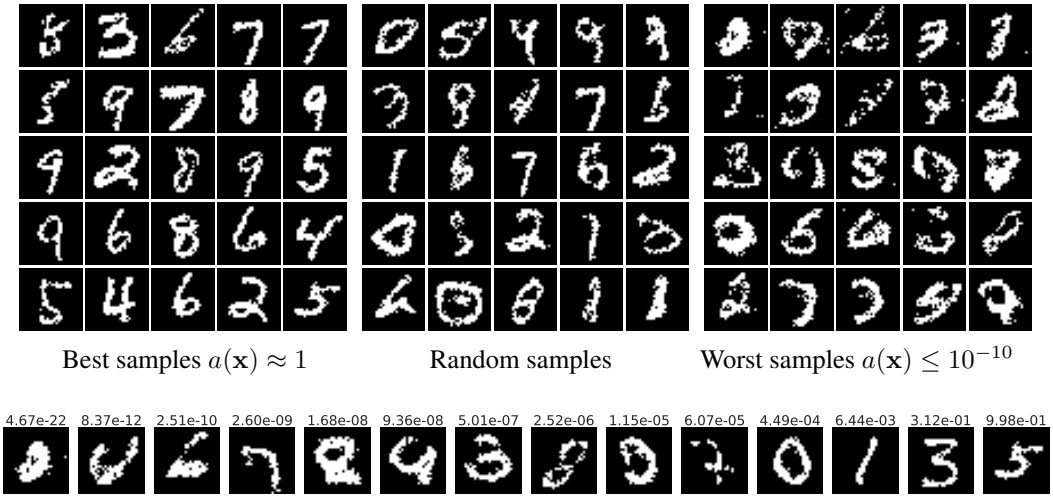


Figure D.9: Samples from a VAE with a jointly trained acceptance function on the *output*, that is, with a resampled marginal likelihood. *top:* Samples grouped by the acceptance function (best, random, and worst); *bottom:* Samples and their acceptance function values.

D.8 VAE with resampled prior and 2D latent space

For visualization purposes, we also trained a VAE with a two-dimensional latent space, $d_{\mathbf{z}} = 2$. We trained both a regular VAE with standard Normal prior as well as VAEs with resampled priors. We considered three different cases that we detail below: (i) a VAE with resampled prior with expressive/large network as a function; (ii) a VAE with resampled prior with limited/small network as a function; and (iii) a pretrained VAE to which we apply LARS post-hoc, that is, we fixed encoder/decoder and only trained the accept function a. We show the following:

- Fig. D.10: The aggregate posterior, that is, the mixture density of all encoded training data, $q(\mathbf{z}) = \frac{1}{N}\sum_n^N q(\mathbf{z}|\mathbf{x})$, as well as the standard Normal prior for a **regular VAE with standard Normal prior**. We find that the mismatch between standard Normal prior and aggregate posterior is $\mathrm{KL}(q(\mathbf{z}) \,\|\, p(\mathbf{z})) \approx 0.4$.
- Fig. D.11: The aggregate posterior, proposal, accept function and resampled density for the **VAE with jointly trained resampled prior** with high capacity/expressive accept function. We found that the LARS prior matches the aggregate posterior very well and has $\mathrm{KL}(q(\mathbf{z}) \,\|\, p(\mathbf{z})) \approx 0.15$. We note that the aggregate posterior is spread out more compared to the regular VAE, see also Fig. D.14 and note that the KL between the proposal and the aggregate posterior is larger than for the regular VAE. The accept function slices out parts of the prior very effectively and redistributes the weight towards the sides.
- Fig. D.12: Same as Fig. D.11 but with a smaller network for the acceptance function. We used the same random seed for both experiments and notice that the structure of the resulting latent space is very similar but more coarsely partitioned compared to the previous case. The final KL between aggregate posterior and resampled prior is slightly larger, $\mathrm{KL}(q(\mathbf{z}) \,\|\, p(\mathbf{z})) \approx 0.20$, which we attribute to the lower flexibility of the a function.
- Fig. D.13: We also consider the case of applying LARS post hoc to the pretrained regular VAE, that is, we only learn a on a fixed encoder/decoder. Thus, the aggregate posterior is identical to Fig. D.10. However, we find that the acceptance function is very well able to resample the standard Normal proposal to fit the aggregate posterior very well and even slightly better than in the jointly trained case. Note that the aggregate posterior is the same as in Fig. D.10 but that the LARS prior matches it much better than the standard Normal proposal.
- Fig. D.14: Scatter plots of the encoded means for all training data points for both the VAE with standard Normal prior as well as the LARS/resampled prior with large and small network for the acceptance function. Colours indicate the different MNIST classes.

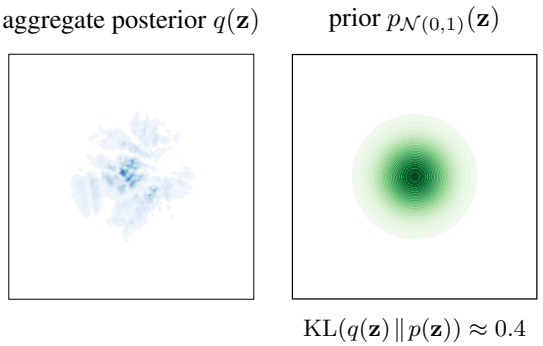


Figure D.10: Aggregate posterior and fixed standard Normal prior for a regular VAE trained with the standard Normal prior. Dynamic MNIST

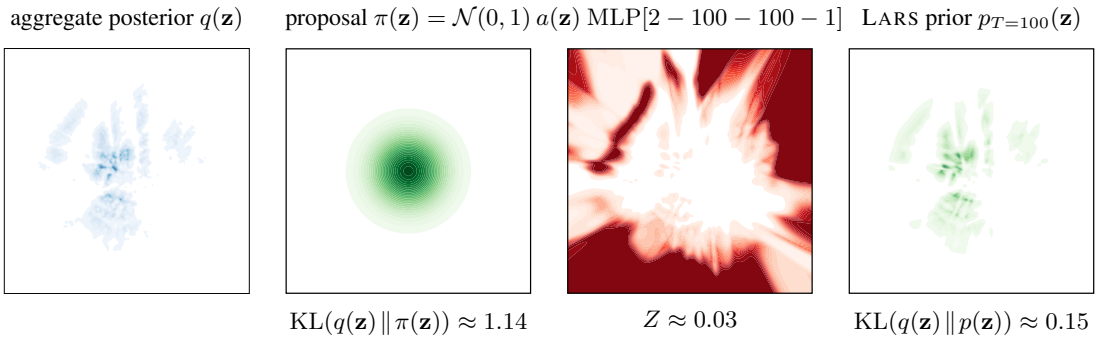


Figure D.11: Aggregate posterior and resampled prior for a VAE with LARS prior and **high capacity** network for a: $a = \mathrm{MLP}[2 - 100 - 100 - 1]$. Dynamic MNIST

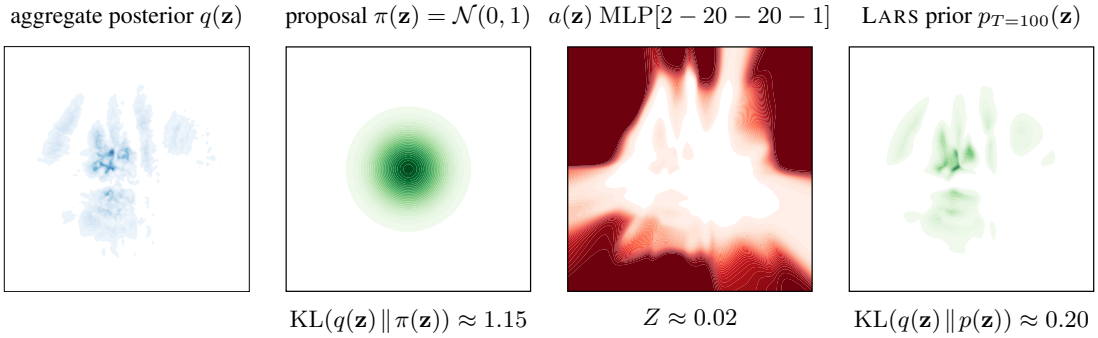


Figure D.12: Aggregate posterior and resampled prior for a VAE with LARS prior and **low capacity** network for a: $a = \mathrm{MLP}[2 - 20 - 20 - 1]$. Dynamic MNIST

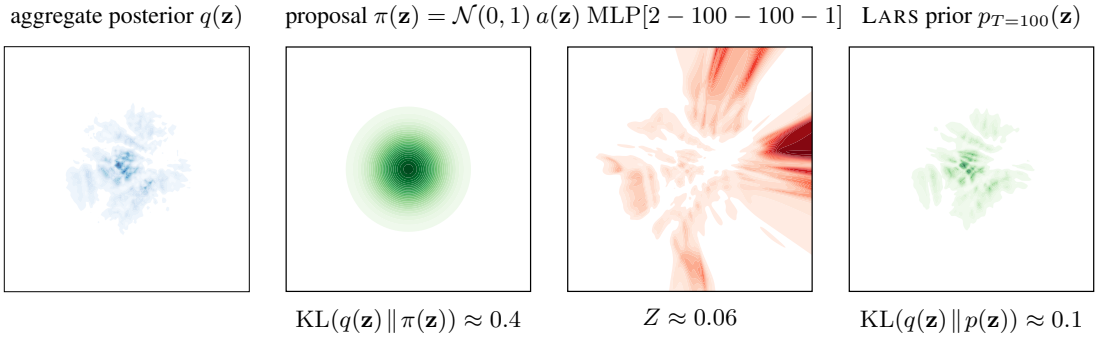
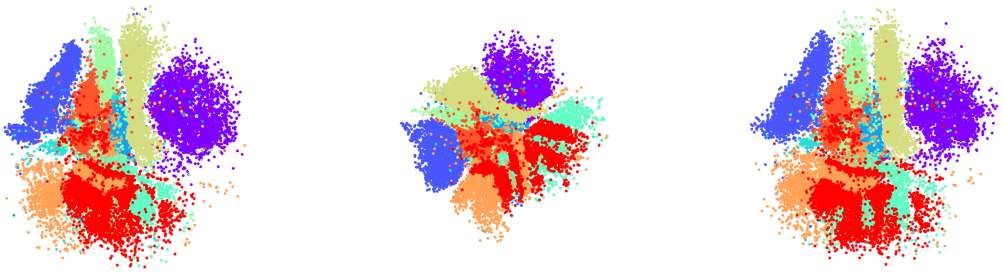


Figure D.13: Aggregate posterior and resampled prior that has been trained post-hoc on the pretrained regular VAE. Dynamic MNIST



VAE with LARS prior
$a = \mathrm{MLP}[2 - 100 - 100 - 1]$

VAE with standard prior

VAE with LARS prior
$a = \mathrm{MLP}[2 - 20 - 20 - 1]$

Figure D.14: Embedding of the MNIST training data into the latent space of a VAE. Colours indicate the different classes and all plots have the same scale.

References

[1] Aleksandar Botev, Bowen Zheng, and David Barber. "Complementary Sum Sampling for Likelihood Approximation in Large Scale Classification". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. 2017.

[2] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. "Generating Sentences from a Continuous Space". In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. 2016.

[3] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. "Importance weighted autoencoders". In: *Proceedings of the 4th International Conference on Learning Representations*. 2016.

[4] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. "Variational Lossy Autoencoder". In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[5] Nat Dilokthanakul, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan. "Deep unsupervised clustering with gaussian mixture variational autoencoders". In: *arXiv preprint arXiv:1611.02648* (2016).

[6] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 2010.

[7] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Rezende, and Daan Wierstra. "DRAW: A Recurrent Neural Network For Image Generation". In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015.

[8] Aditya Grover, Ramki Gummadi, Miguel Lazaro-Gredilla, Dale Schuurmans, and Stefano Ermon. "Variational Rejection Sampling". In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. 2018.

[9] Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. "PixelVAE: A Latent Variable Model for Natural Images". In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[10] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, and Alexander Lerchner Shakir Mohamed. "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[11] Geoffrey Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". In: *Neural Computation* (2000).

[12] Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. "Neural Autoregressive Flows". In: *Proceedings of the 35th International Conference on Machine Learning*. 2018.

[13] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd International Conference on Learning Representations*. 2015.

[14] D.P. Kroese, T. Taimre, and Z.I. Botev. *Handbook of Monte Carlo Methods*. Wiley, 2013.

[15] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 6266 (2015).

[16] Hugo Larochelle and Iain Murray. "The Neural Autoregressive Distribution Estimator". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011.

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 11 (1998).

[18] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[19] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. "Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017.

[20] Christian Naesseth, Francisco Ruiz, Scott Linderman, and David Blei. "Reparameterization Gradients through Acceptance-Rejection Sampling Algorithms". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. 2017.

[21] Eric Nalisnick, Lars Hertel, and Padhraic Smyth. "Approximate inference for deep latent gaussian mixtures". In: *NIPS Workshop on Bayesian Deep Learning*. 2016.

[22] George Papamakarios, Iain Murray, and Theo Pavlakou. "Masked Autoregressive Flow for Density Estimation". In: *Advances in Neural Information Processing Systems 30*. 2017.

[23] Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed. "Distribution Matching in Variational Inference". In: *arXiv* (2018).

[24] M. Sugiyama, T. Suzuki, and T. Kanamori. "Density-ratio matching under the Bregman divergence: a unified framework of density-ratio estimation". In: *Annals of the Institute of Statistical Mathematics* 5 (2012).

[25] Jakub Tomczak and Max Welling. "VAE with a VampPrior". In: *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*. 2018.

[26] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. 28, 2017.