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 probability given by $a(\mathbf{z})$.

By definition, the probability of sampling $\mathbf{z}$ from the proposal is given by $\pi(\mathbf{z})$, so the probability of sampling and then accepting a particular $\mathbf{z}$ is given by the product $\pi(\mathbf{z})a(\mathbf{z})$. The probability of sampling and then rejecting $\mathbf{z}$ is given by the complementary probability $\pi(\mathbf{z})(1 - a(\mathbf{z}))$.

Thus, the expected probability of accepting a sample is given by $\int \pi(\mathbf{z})a(\mathbf{z})\mathrm{d}\mathbf{z}$, whereas the expected probability of rejecting a 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 the sample in the first draw and subsequently sample and accept $\mathbf{z}$; and so on. As all draws are independent, we can compute the sum as a geometric series:

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

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

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

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

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

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

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

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

Average number of resampling steps until acceptance. We start by noting that the number of resampling steps performed until a sample is accepted follows the geometric distribution with success probability Z:

$$\Pr(\text{resampling time} = t) = Z(1-Z)^{t-1} \tag{A.8}$$

This is due to the fact that each proposed candidate sample is accepted with probability Z, such decisions are independent, and the process continues until a sample is accepted. As the expected value of a geometric random variable is simply the reciprocal of the success probability, the expected number of resampling steps is $\langle t \rangle_{p_\infty} = 1/Z$.

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 $a(\mathbf{z})$.

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

$$\sum_{n=0}^{N} x^n = \frac{1-x^{N+1}}{1-x} \qquad \text{for } |x| < 1 \tag{A.9}$$

$$\begin{aligned}
p_T(\mathbf{z}) &= \sum_{t=1}^{T} \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } t | \text{rejected previous } t-1 \text{ samples}\right) \Pr\left(\text{reject } t-1 \text{ samples}\right) && \text{(A.10)}\\
&= \sum_{t=1}^{T-1} \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } t | \text{rejected previous } t-1 \text{ samples}\right) \Pr\left(\text{reject } t-1 \text{ samples}\right) && \text{(A.11)}\\
&\quad + \Pr\left(\text{accept sample } \mathbf{z} \text{ at step } T | \text{rejected previous } T-1 \text{ samples}\right) \Pr\left(\text{reject } T-1 \text{ samples}\right) && \text{(A.12)}\\
&= a(\mathbf{z})\pi(\mathbf{z}) \sum_{t=0}^{T-2} (1-Z)^t + \pi(\mathbf{z})(1-Z)^{T-1} && \text{(A.13)}\\
&= a(\mathbf{z})\pi(\mathbf{z}) \frac{1-(1-Z)^{T-1}}{Z} + \pi(\mathbf{z})(1-Z)^{T-1}. && \text{(A.14)}
\end{aligned}$$

Again, note that we have shifted the index of summation going from the second to third line. Moreover, note that $a(\mathbf{z})$ 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 $a(\mathbf{z})$.

The obtained probability density is 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:

$$\log p_T(\mathbf{z}) = \log \pi(\mathbf{z}) + \log\left[a(\mathbf{z})\frac{1-(1-Z)^{T-1}}{Z} + (1-Z)^{T-1}\right] \tag{A.15}$$

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:

$$\log p_{T=1}(\mathbf{z}) = \log \pi(\mathbf{z}) \tag{A.16}$$

$$\lim_{T\to\infty} \log p_T(\mathbf{z}) = \log \frac{\pi(\mathbf{z})a(\mathbf{z})}{Z}. \tag{A.17}$$

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

Average number of resampling steps until acceptance. We can derive the average number of resampling steps for truncated resampling by using the result for indefinite resampling, $\langle t \rangle_{p_\infty} = 1/Z$, and noting that the geometric distribution is memoryless:

$$\begin{aligned}
\langle t \rangle_{p_T} &= \langle t \rangle_{p_\infty} - \langle t \rangle_{p_{T..\infty}} + \langle t \rangle_{\text{accept } T\text{th sample}} && \text{(A.18)}\\
&= \frac{1}{Z} - \left(T - 1 + \frac{1}{Z}\right)(1-Z)^{T-1} + T(1-Z)^{T-1} && \text{(A.19)}\\
&= \frac{1-(1-Z)^T}{Z} && \text{(A.20)}
\end{aligned}$$

where we used the memoryless property of the geometric series for the second term. Also note that as the probability to accept the T^{th} sample given that we rejected all previous $T-1$ samples is 1 instead of Z, the third term does not include a factor of Z.

From this result we can derive the following limits and special cases, which agree with the intuitive behaviour for truncated resampling; specifically, if we reject all samples ($a(\mathbf{z}) \approx 0$, thus $Z \to 0^+$), the average number of resampling steps achieves the maximum possible value (T), whereas if we accept every sample, it goes down to 1:

$$\langle t \rangle_{p_{T=1}} = 1 \tag{A.21}$$

$$\langle t \rangle_{p_{T=2}} = 2 - Z \leq 2 \tag{A.22}$$

$$\langle t \rangle_{p_{T\to\infty}} = \frac{1}{Z} \tag{A.23}$$

$$\lim_{Z\to 0^+} \langle t \rangle_{p_T} = T \tag{A.24}$$

$$\lim_{Z\to 1^-} \langle t \rangle_{p_T} = 1 \tag{A.25}$$

A.3 Illustration of standard/classical Rejection Sampling

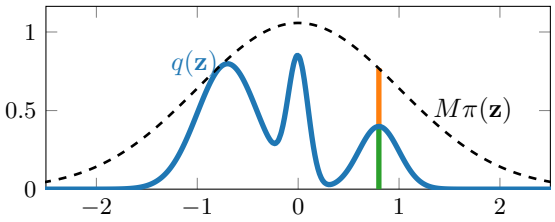


Figure A.1: In rejection sampling we can draw samples from a complicated target distribution (——) by sampling from a simpler proposal (- - -) and accepting samples with probability $a(\mathbf{z}) = \frac{q(\mathbf{z})}{M\pi(\mathbf{z})}$. The green and orange line are proportional to the relative accept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

$$Z = \frac{1}{S}\sum_{s}^{S} a(\mathbf{z}_s) \qquad \text{with } \mathbf{z}_s \sim \pi(\mathbf{z}). \tag{A.26}$$

In practice, we use $S = 10^{10}$ samples, and evaluating Z takes only several minutes on a GPU. The fast evaluation time is due to the relatively low dimensionality of the latent space, so drawing the samples and evaluating their acceptance function values $a(\mathbf{z})$ is fast and can be parallelized by using large batch sizes (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 [4]).

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

$$Z_S = \frac{1}{S}\sum_{i}^{S} a(\mathbf{z}_i) \qquad \text{with } \mathbf{z}_s \sim \pi(\mathbf{z}) \tag{A.27}$$

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.26)).

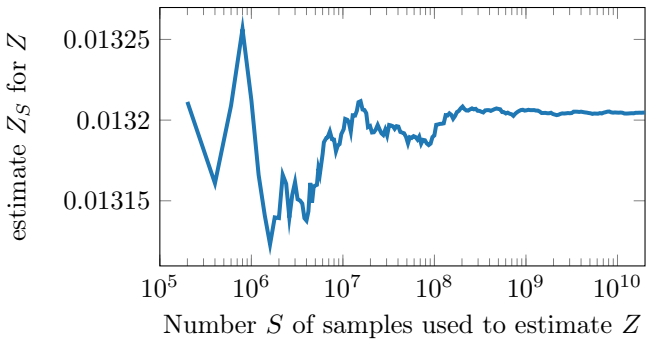


Figure A.2: Estimation of Z by MC sampling from the proposal. For less than 10^7 samples the estimate is quite noisy but it stabilizes after about 10^9 samples.

However, when using too few samples, the estimate for Z can be very noisy, see Fig. A.2, and the values for $\log Z$ are substantially biased. In practice, we use two techniques to deal with 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 techniques in more detail.

During training we evaluate $\mathrm{KL}(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}))$ in a stochastic 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 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:

$$\frac{1}{R}\sum_{r=1}^{R}\log p_T(\mathbf{z}_r) = \frac{1}{R}\sum_{r}^{R}(\log \pi(\mathbf{z}_r) + \log a(\mathbf{z}_r)) - \log Z \tag{A.28}$$

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 smooth 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:

$$\langle Z\rangle_{i+1} = (1-\epsilon)\langle Z\rangle_i + \epsilon Z_i \tag{A.29}$$

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 (Z_i), which needs to be rescaled to account for ϵ. Before we show how we do this, we introduce our second way 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.28) by:

$$\log Z \to \frac{1}{R}\sum_{r}^{R}\log Z_r \tag{A.30}$$

$$Z_r = \frac{1}{S+1}\left[\sum_{s}^{S} a(\mathbf{z}_s) + \frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r|\mathbf{x}_r)}a(\mathbf{z}_r)\right] \qquad \mathbf{z}_s \sim \pi(\mathbf{z}); \quad \mathbf{z}_r \sim q(\mathbf{z}|\mathbf{x}_r) \tag{A.31}$$

$$= \frac{1}{S+1}\left[SZ_S + \frac{\pi(\mathbf{z}_r)}{q(\mathbf{z}_r|\mathbf{x}_r)}a(\mathbf{z}_r)\right] \tag{A.32}$$

where the sample $\mathbf{z}_r$ from the variational posterior is reweighted by the ratio of prior and variational posterior, similar to [2]. Note that we do not want to backpropagate through this fraction, as this would alter the gradients for the encoder and 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.30)) 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 Experimental Details

B.1 Datasets

The MNIST dataset [7] 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 [6].

Omniglot [5] 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 [9] 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 [11] 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.

B.2 Network architectures

In general, we decided to use simple 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 $p(\mathbf{x}|\mathbf{z}_1, \mathbf{z}_2) = \text{MLP}[[\text{MLP}[d_{\mathbf{z}} - 300], \text{MLP}[d_{\mathbf{z}} - 300]] - 300 - 28 \times 28]$, 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 $\text{MLP}[d_{\mathbf{z}} - 300]$, 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 nonlinearity for the hidden layers was `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 variational posterior.

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

which are given by:

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

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:

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

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 inspired by the architectural choices of Tomczak and Welling [9], 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:

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

We again use `tanh` nonlinearities in 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.

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

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

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

again with `tanh` nonlinearities in the 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` [1]. 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 [8] (MAF) prior/proposal we use a stack of 5 MAFs with $\text{MLP}[100 - 100]$ each and again use the reference implementation from `TensorFlow Probablity`. Random permutations are employed between individual MAF blocks [8].

B.3 Training procedure for a VAE with resampled prior

LARS constitutes another method to specify rich priors, which are parameterized through the acceptance function $a_\lambda(\mathbf{z})$; we use subscript λ in this section to highlight the learnable parameters of the acceptance function. The acceptance function only modifies the prior of the model with the encoder and decoder remaining unchanged. Thus, the only part of the training objective, which changes compared to training of a normal VAE, is the evaluation of prior contribution to the KL term, $\log p(\mathbf{z})$. This log density is evaluated on samples from the encoder distribution $q_\varphi(\mathbf{z}|\mathbf{x})$. Both for truncated and untruncated sampling, evaluation of $\log p(\mathbf{z})$ entails evaluating (i) the proposal density $\pi(\mathbf{z})$ (ii) the acceptance function $a_\lambda(\mathbf{z})$ (iii) the normalizer Z and their logarithms. Evaluation of the proposal and acceptance function is straight forward and evaluation of the normalizer and its logarithm are detailed in Algorithm 1.

As discussed in the main paper, we never need to perform accept/reject sampling during training but only need to evaluate the prior density. To update the normalizer during training, we require a modest number of samples from the proposal; however, we never actually perform the accept/reject step and never need to decode these samples into image space using the decoder model.

The full training procedure for a VAE as well as the changes necessary due the resampled prior are detailed in Algorithm 2. We detail the pseudo code for resampled priors with untruncated/indefinite resampling but an extension to the truncated case is straight forward.

Algorithm 2: Pseudo code for training of a VAE with resampled prior (for untruncated resampling)

input : dataset of images $\mathcal{D}_{\text{train}} = \{\mathbf{x}_n\}_n^N$
output : trained model parameters φ, θ (encoder/decoder) and $\lambda, \log Z$ (resampled prior)

initialize parameters of encoder φ and decoder θ
initialize parameters of the resampled prior λ and the moving average $\langle Z \rangle$
for $it \leftarrow 1$ **to** N_{it} **do**
  sample a minibatch of data $\{\mathbf{x}_r\}_r^R$
  sample latents $\{\mathbf{z}_r\}_r^R$ from the encoder distribution $\{q_\varphi(\mathbf{z}|\mathbf{x}_r)\}_r^R$ (using reparameterization)
  evaluate the reconstruction term: recon $\leftarrow \frac{1}{R}\sum_r^R \log p_\theta(\mathbf{x}|\mathbf{z}_r)$
  update moving average $\langle Z \rangle$ and compute $\log Z$ using Algorithm 1
  evaluate the KL term: kl $\leftarrow \frac{1}{R}\sum_r^R [\log q_\varphi(\mathbf{z}_r|\mathbf{x}_r) - \log \pi(\mathbf{z}_r) - \log a_\lambda(\mathbf{z}_r)] + \log Z$
  evaluate the objective: recon $-$ kl
  backpropagate gradients into all parameters φ, θ and λ
end
compute final normalizer for evaluation: $Z \leftarrow \frac{1}{S_{\text{eval}}}\sum_s^{S\text{eval}} a(\mathbf{z}_s)$ with $\mathbf{z}_s \sim \pi(\mathbf{z})$

B.4 Training a VAE with resampled outputs

Here, we elaborate on the training procedure for a VAE with *resampled discrete outputs* presented in Section 5.3. In particular, we explain why we essentially perform post-hoc fitting for the acceptance function $a(\mathbf{x})$.

As stated in the main text, the ELBO for the resampled objective is given by:

$$\mathbb{E}_{q_\varphi(\mathbf{z}|\mathbf{x})} \log \pi_\theta(\mathbf{x}|\mathbf{z}) - \mathrm{KL}(q_\varphi(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})) + \log \tfrac{a_\lambda(\mathbf{x})}{Z}, \tag{B.33}$$

where $\mathbf{x}$ corresponds to the observation under consideration and we have reintroduced the subscripts φ, θ, λ to indicate the learnable parameters of the encoder, decoder, and acceptance function, respectively. The normalizer Z is estimated on decoded samples from the (regular) prior $p(\mathbf{z})$:

$$Z = \int \pi_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) a_\lambda(\mathbf{x}) \mathrm{d}\mathbf{x}\mathrm{d}\mathbf{z} = \mathbb{E}_{\pi_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})}[a_\lambda(\mathbf{x})] \approx \tfrac{1}{S}\sum_s^S a_\lambda(\mathbf{x}_s), \tag{B.34}$$

with $\mathbf{x}_s \sim \pi(\mathbf{x}|\mathbf{z}_s)$ and $\mathbf{z}_s \sim p(\mathbf{z})$. Note that because the images in the MNIST dataset are binary, both the observations $\mathbf{x}$ as well as the model samples $\mathbf{x}_s$ are discrete variables. This does not make learning the parameters λ of the acceptance function $a_\lambda(\mathbf{x})$ more difficult than in the continuous case, as it does not involve propagating gradients *through* $\mathbf{x}$.

However, the situation is different for the parameters θ of the decoder, as the distribution of the model samples $\mathbf{x}_s$ and, thus, our estimate of Z depends on them. Therefore we need to compute the gradient of the sample-based estimate Z w.r.t. the decoder parameters θ; this would be be easy to do by using the reparameterization trick if $\mathbf{x}_s$ were continuous. Unfortunately, in our case, the samples $\mathbf{x}_s$ are discrete and thus non-reparameterizible, which means that we have to use a more general gradient estimator such as REINFORCE [10]. Unfortunately, estimators of this type usually have much higher variance than reparameterization gradients; so for simplicity we choose to ignore this contribution to the gradients of the decoder parameters. Computationally, this amounts to estimating Z using

$$Z \approx \frac{1}{S}\sum_s^S a_\lambda(\texttt{stop\_grad}(\mathbf{x}_s)). \tag{B.35}$$

Thus, the encoder and decoder parameters are trained effectively by optimizing the first two terms of Eq. (B.33), whereas the parameters of the acceptance function are trained by optimizing the last term of Eq. (B.33). This is essentially equivalent to the post-hoc setting considered in Table 2 because the acceptance function has no effect on training the rest of the model.

C Additional results

C.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. C.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. [3], who observe a similar shortcoming for Masked Autoregressive Flows (MAFs).

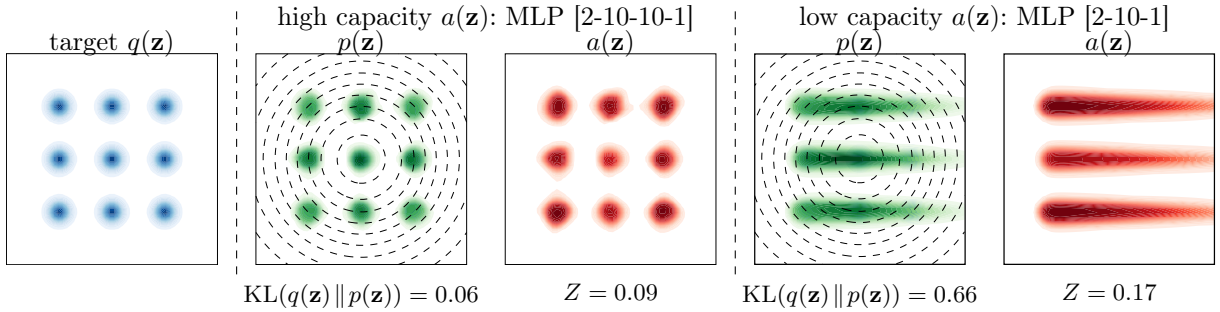


Figure C.3: Learned acceptance functions $a(\mathbf{z})$ (red) that approximate a fixed target q (blue) by reweighting a $\mathcal{N}(0, 1)$ proposal (- - -) to obtain an approximate density $p(\mathbf{z})$ (green). Darker values correspond to higher numbers.

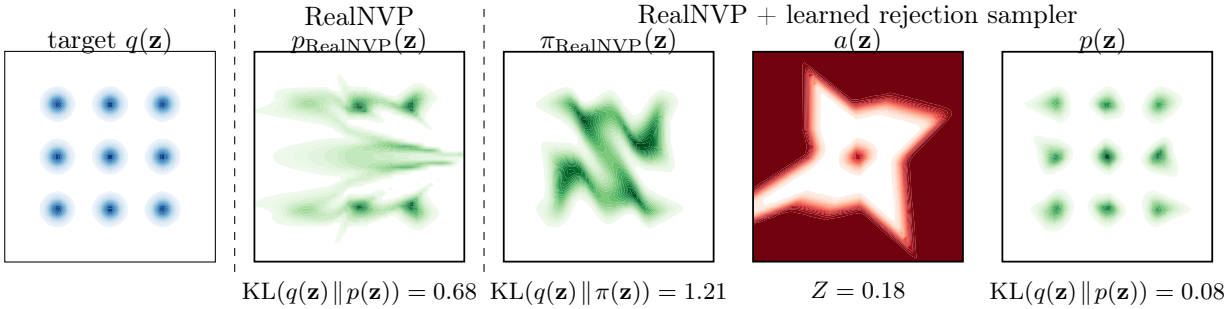


Figure C.4: Training with a RealNVP proposal. The target is approximated either by a RealNVP alone (left) or a RealNVP in combination with a learned rejection sampler (right). Darker values correspond to higher numbers.

C.2 LARS in combination with a non-factorial prior on Omniglot

In Table C.1 we show results for combining a RealNVP proposal with LARS. Similarly to MNIST (Table 7 in the main paper), we find that a RealNVP used as a prior by itself outperforms a combination of VAE with resampled prior. However, if we combine the RealNVP proposal with LARS, we obtain a further improvement.

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 C.1: Test NLL on dynamic Omniglot. Equivalent table to Table 7 in the main paper.

C.3 Samples from jointly trained VAE models

We generated 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. C.5 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 are 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. C.6, 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 are very poor.

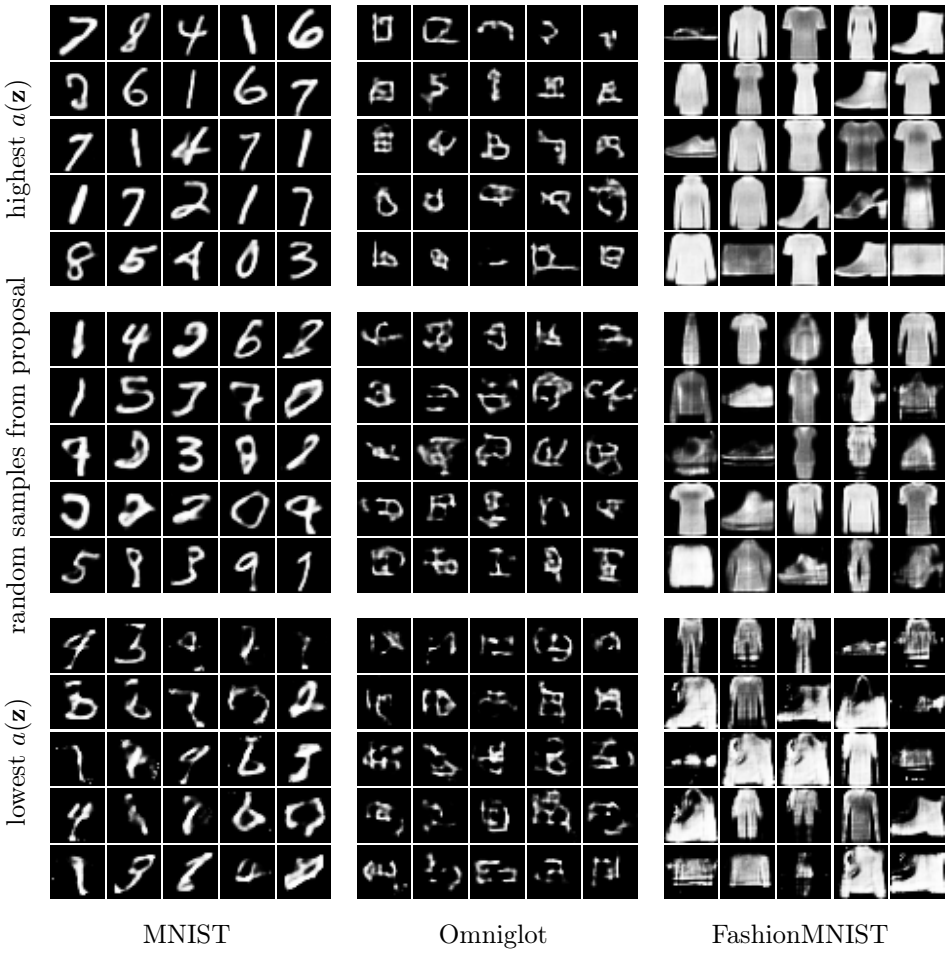


Figure C.5: Comparison of sample means from a VAE with MLP encoder/decoder and jointly trained resampled prior. Samples shown are out of 10^4 samples drawn. *top:* highest $a(\mathbf{z})$; *middle:* random samples from the proposal; *bottom:* lowest $a(\mathbf{z})$. All samples from the simple VAE model with MLP encoder/decoder.

C.4 Samples from applying LARS to a pretrained VAE model

Similarly to Appendix C.3, 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. C.7. We fixed the encoder and decoder and only trained the acceptance function on the usual ELBO objective.

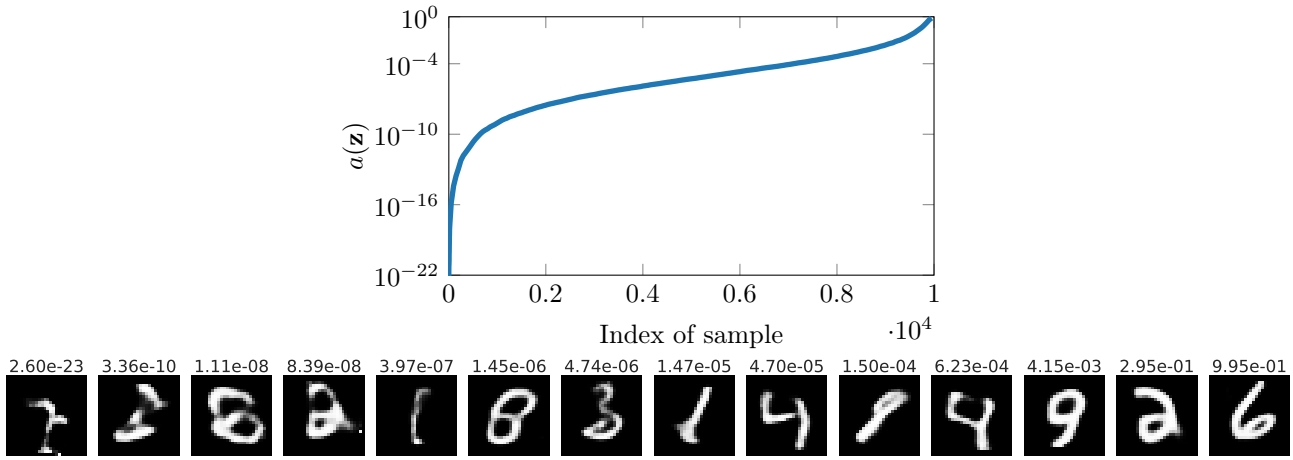


Figure C.6: Samples from the proposal of a jointly trained VAE model with resampled prior on MNIST and their corresponding acceptance function values. *top:* Distribution of the $a(\mathbf{z})$ values (index sorted by $a(\mathbf{z})$); note the log scale for $a(\mathbf{z})$. The average acceptance probability 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.

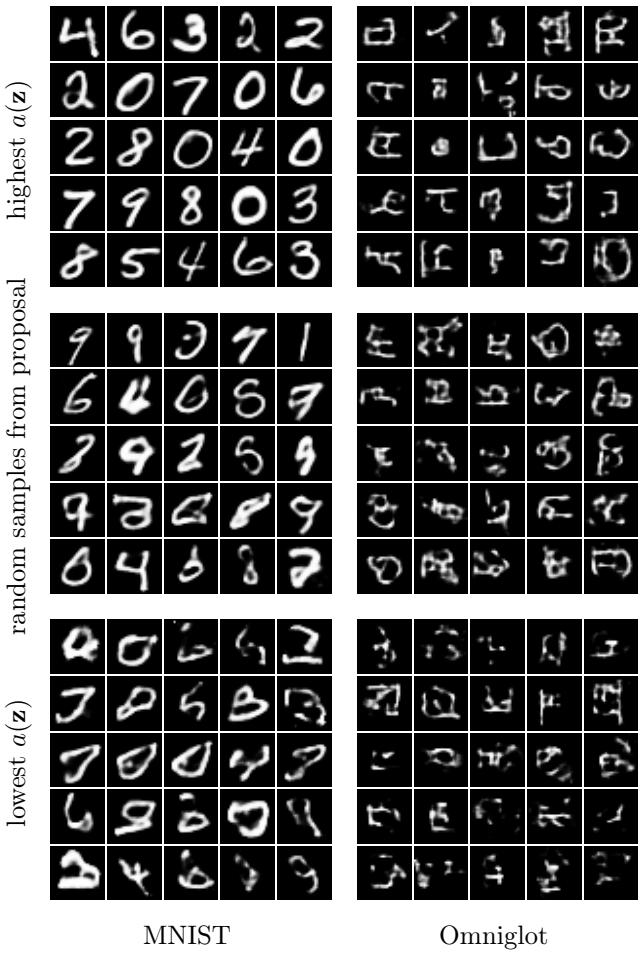
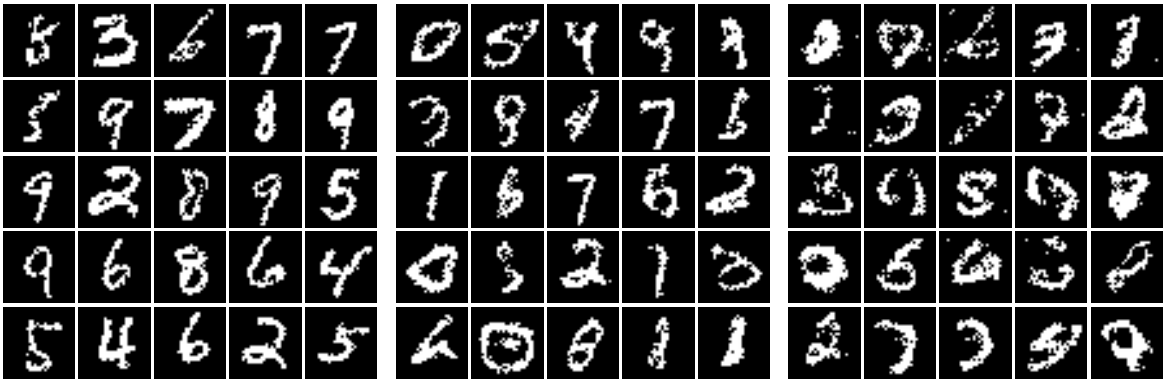


Figure C.7: Comparison of sample means from a VAE with pretrained MLP encoder/decoder to which we applied a resampled LARS prior post-hoc. Samples shown are out of 10^4 samples drawn. *top:* samples with highest $a(\mathbf{z})$; *middle:* random samples from the proposal; *bottom:* samples with lowest $a(\mathbf{z})$.

C.5 Samples from a VAE with LARS output distribution

In Fig. C.8 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 well, and $a(\mathbf{x})$ is able to reliably rank images.



Best samples $a(\mathbf{x}) \approx 1$ | Random samples | Worst samples $a(\mathbf{x}) \leq 10^{-10}$

Figure C.8: 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.

C.6 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 a resampled prior with expressive/large network as a function; (ii) a VAE with a 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. C.9: 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. C.10: 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. C.13 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. C.11: Same as Fig. C.10 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. C.12: 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. C.9. However, we find

that the acceptance function is able to modulate 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. C.9 but that the LARS prior matches it much better than the standard Normal proposal.

- Fig. C.13: 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.

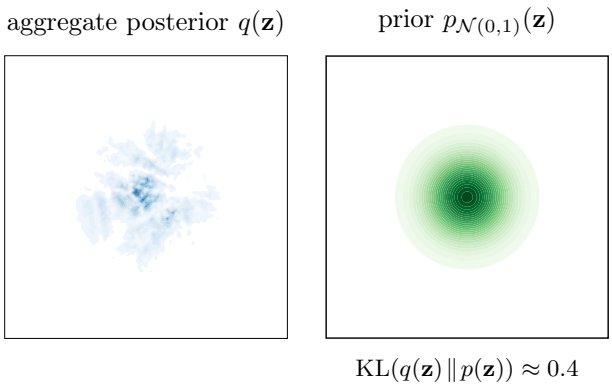


Figure C.9: Aggregate posterior and fixed standard Normal prior for a regular VAE trained with the standard Normal prior. Dynamic MNIST

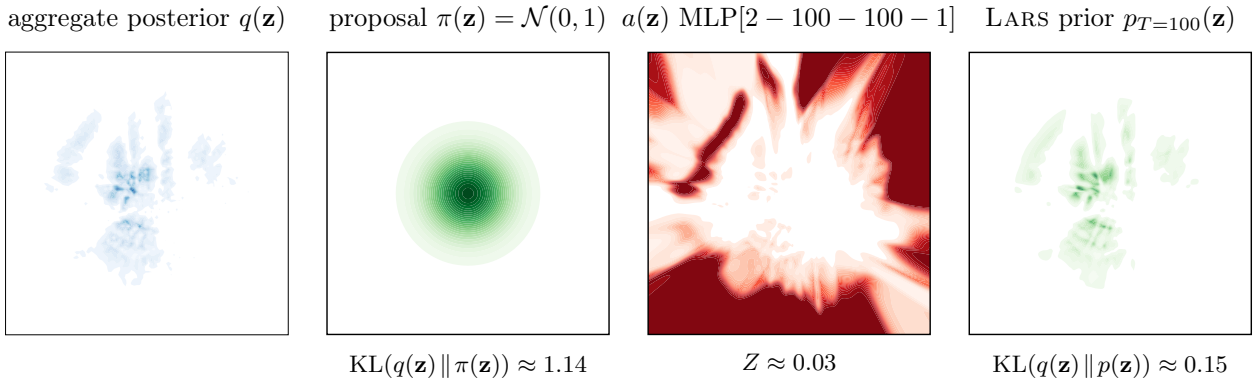


Figure C.10: Aggregate posterior and resampled prior for a VAE with LARS prior and **high capacity** network for a: $a = \mathrm{MLP}[2 - 100 - 100 - 1]$. Dynamic MNIST

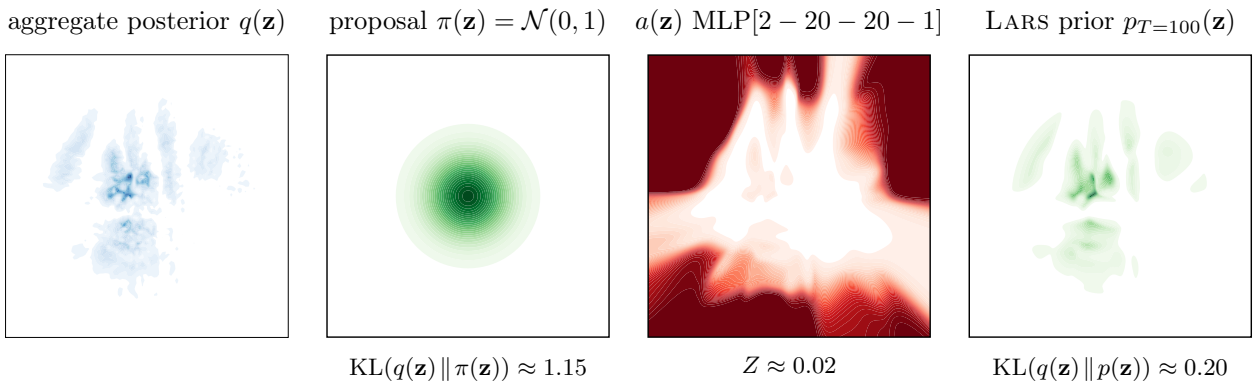


Figure C.11: Aggregate posterior and resampled prior for a VAE with Lars prior and **low capacity** network for a: $a = \mathrm{MLP}[2 - 20 - 20 - 1]$. Dynamic MNIST

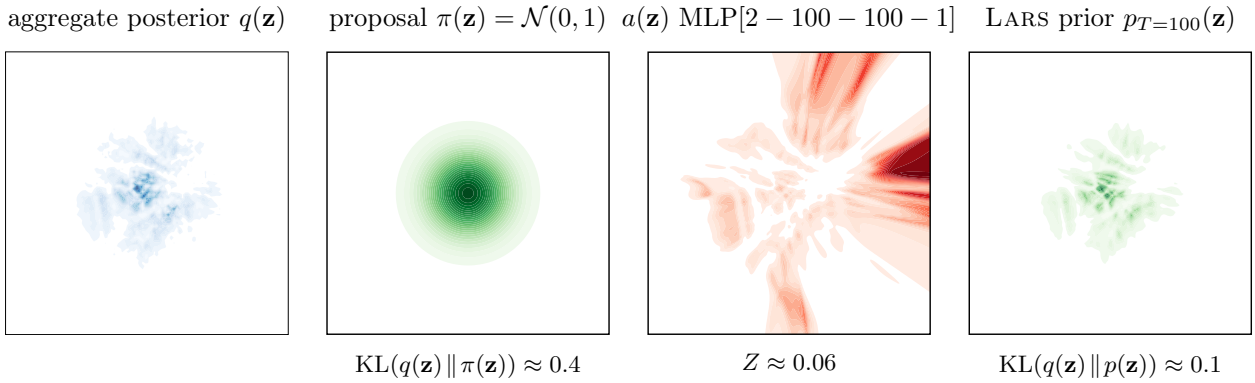
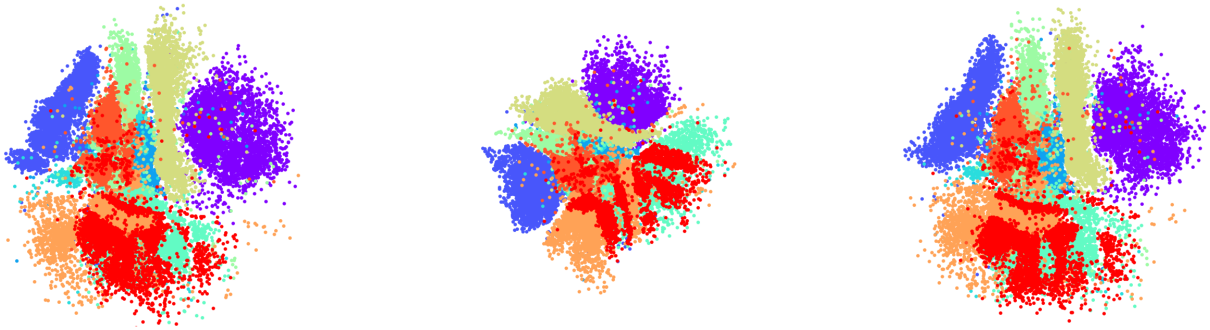


Figure C.12: Aggregate posterior and resampled prior that has been trained post-hoc on the pretrained regular VAE. Dynamic MNIST



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 C.13: 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] 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.

[2] 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.

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

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

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

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

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

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

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

[10] Ronald J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 3 (1992).

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