A. Mathematical details

A.1. Solving mean squared error to for conditional expectations

Given $\boldsymbol{x}, \boldsymbol{y} \sim \rho(\boldsymbol{x}, \boldsymbol{y})$, we want to find an estimator in some space $\mathcal{F}$ of the posterior mean function $\boldsymbol{f}_\rho : \boldsymbol{x} \mapsto \mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]$. Assuming that $\mathcal{F}$ is contained in $\mathcal{L}^2_\rho$, the class of squared-integral functions under $\rho(\boldsymbol{x})$, and that $\boldsymbol{y}$ has finite l-2 norm under $\rho(\boldsymbol{y})$, a natural cost function to learn $\boldsymbol{f}$ is the expected squared l-2 distance

$$L_E(\boldsymbol{f}) := \mathbb{E}_{\rho(\boldsymbol{y},\boldsymbol{x})}\left[\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2\right] = \mathbb{E}_{\rho(\boldsymbol{x})}\left[\mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}\left[\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2\right]\right].$$

By Jensen's inequality,

$$L_E(\boldsymbol{f}) \leq \mathbb{E}_{\rho(\boldsymbol{x})}\left[\left\|\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]\right\|_2^2\right] = L_R(\boldsymbol{f}).$$

This shows that the MSE is an upper bound on the expected l-2 distance between $\boldsymbol{f}(\boldsymbol{x})$ and the posterior mean $\mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]$. Further, the minimum of L_R is attained at an $\boldsymbol{f}$ that also minimises L_E. This can be shown through a simple decomposition

$$\begin{aligned}
L_E(\boldsymbol{f}) &= \mathbb{E}_{\rho(\boldsymbol{x})}\left[\mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}\left[\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y}\|_2^2\right]\right] \\
&= \mathbb{E}_{\rho(\boldsymbol{x})}\left[\|\boldsymbol{f}(\boldsymbol{x})\|_2^2 - \boldsymbol{f}(\boldsymbol{x}) \cdot \mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}] + \mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}\left[\|\boldsymbol{y}\|_2^2\right]\right] \\
&\overset{(1)}{=} \mathbb{E}_{\rho(\boldsymbol{x})}\left[\|\boldsymbol{f}(\boldsymbol{x})\|_2^2 - \boldsymbol{f}(\boldsymbol{x}) \cdot \mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}] + \left\|\mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]\right\|_2^2 + \mathrm{Tr}\left[\mathbb{C}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]\right]\right] \\
&= \mathbb{E}_{\rho(\boldsymbol{x})}\left[\mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}\left[\left\|\boldsymbol{f}(\boldsymbol{x}) - \mathbb{E}_{\rho(\boldsymbol{y}|\boldsymbol{x})}[\boldsymbol{y}]\right\|_2^2\right]\right] + \mathbb{E}_{\rho(\boldsymbol{x})}\left[\mathrm{Tr}\left[\mathbb{C}_{\rho(\boldsymbol{y}|\boldsymbol{x})}(\boldsymbol{y}\right]\right] \\
&= L_R(\boldsymbol{f}) + \text{term independent of } \boldsymbol{f}
\end{aligned}$$

where $\mathbb{C}_p$ is the covariance under p. Equality (1) holds because

$$\mathbb{E}_p\left[\|\boldsymbol{a}\|_2^2\right] = \mathbb{E}_p\left[\sum_i a_i^2\right] = \sum_i \mathbb{E}_p\left[a_i^2\right] = \mathbb{E}_p[a_i]^2 + \sum_i \mathbb{V}_p[a_i] = \|\mathbb{E}_p[\boldsymbol{a}]\|_2^2 + \mathrm{Tr}\left[\mathbb{C}_p[\boldsymbol{a}]\right]$$

for any $\boldsymbol{a} \in$ in $\mathcal{L}^2_p$. So $L_R(\boldsymbol{f})$ is equal to $L_E(\boldsymbol{f})$ up to a constant that depends only on ρ but not $\boldsymbol{f}$.

A.2. Boundedness of the gradient function

To learn $\boldsymbol{y}(\boldsymbol{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})]\big|_{\boldsymbol{\theta}_t}$ using regression as above, the target needs to be square-integrable under $p_{\boldsymbol{\theta}_t}(\boldsymbol{x})$, i.e. $\boldsymbol{y}(\boldsymbol{x}) \in \mathcal{L}^2_p$. Common likelihood functions are in the exponential family and has $\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} \boldsymbol{\eta}(\boldsymbol{z}) \boldsymbol{s}(\boldsymbol{x}) - \nabla_{\boldsymbol{\theta}} \Psi(\boldsymbol{z})$. Thus, it suffices to check the $\mathcal{L}^2_p$ integrability of the gradient in terms of these functions. We sketch below that this is indeed the case for common choices of model architectures.

As a simple example, consider a model

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \qquad p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z}), \boldsymbol{\Sigma}). \tag{14}$$

where $\boldsymbol{I}$ is the identity covariance matrix, $\mathrm{NN}_{\boldsymbol{w}}$ is a neural network with weights $\boldsymbol{w}$ and $\boldsymbol{\Sigma}$ is a diagonal matrix. Note that in this case, one has that

$$\Psi_{\boldsymbol{\theta}}(\boldsymbol{z}) = -\frac{1}{2}\|\boldsymbol{z}\|_2^2 - \frac{1}{2}\log|\boldsymbol{\Sigma}| + \text{constant}, \quad \boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z}) = [\boldsymbol{\Sigma}^{-1}\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z}), -\frac{1}{2}\boldsymbol{\Sigma}^{-1}], \quad \boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{\Sigma}\}, \quad \boldsymbol{s}(\boldsymbol{x}) = [\boldsymbol{x}, \boldsymbol{x}\boldsymbol{x}^T].$$

Further, assume that

1. The neural network $\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z})$ is Lipschitz and $\boldsymbol{w}$-differentiable almost everywhere, such as one that is composed of linear projections followed by Lipschitz nonlinearities (e.g., ReLU).
2. $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\left[\|\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z})\|_2^2\right] < \infty$.
3. Spectral norm of weights $\boldsymbol{W}$ in each layer of $\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z})$ is bounded above by a positive constant.

4. The diagonal elements of $\boldsymbol{\Sigma}$ are bounded below by some constant.

The first and second assumptions are mild and satisfied by NNs with ReLU activations. The third and fourth conditioons limit the ranges of the parameter values, which can be imposed by clipping or through appropriate parametrisation.

The second and fourth conditions make the gradients of $\Psi_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})$ w.r.t. $\boldsymbol{\Sigma}$ bounded; thus, we will demonstrate the integrability of the gradients w.r.t. the neural network parameter $\boldsymbol{w}$.

First term $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})]\boldsymbol{s}(\boldsymbol{x})$

Multiple applications of the Cauchy-Schwartz inequality yields

$$\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[\left\|\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{w}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})]\boldsymbol{s}(\boldsymbol{x})\right\|^2\right] \leq \sqrt{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[\|\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{w}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})]\|_2^4\|\right]}\sqrt{\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}[\|\boldsymbol{s}(\boldsymbol{x})\|_2^4]}.$$

By our assumption, $\mathrm{NN}(\boldsymbol{z})$ is Lipschitz w.r.t. $\boldsymbol{w}$ and the gradient $\nabla_{\boldsymbol{w}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})$ is bounded as, for $C_0, C_1 > 0$, $\|\nabla_{\boldsymbol{w}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})\|_2 \leq C_0 + C_1\|\boldsymbol{z}\|_2$. This can be proved by writing out $\nabla_{\boldsymbol{w}}\mathrm{NN}_{\boldsymbol{\theta}}(\boldsymbol{z})$ using chain rule, which will be a series of product involving $\boldsymbol{W}$ in each layer and derivative of Lipschitz functions, and applying the first two conditions above. Thus, we have

$$\begin{aligned}\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[\left\|\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{w}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})]\right\|_2^4\right] &\leq \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\left[\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\left[\|\nabla_{\boldsymbol{w}}\boldsymbol{\eta}_{\boldsymbol{\theta}}(\boldsymbol{z})\|_2^4\right]\right] \\ &\leq \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\left[(C_0 + C_1\|\boldsymbol{z}\|_2)^4\right] < \infty\end{aligned}$$

as the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ is a standard Gaussian.

The integrability of $\boldsymbol{s}(\boldsymbol{x})$ is equivalent to the finiteness of the corresponding moments of $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. By Lemma A.2, the marginal $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ has exponential tails, and thus the moments are finite.

Second term $\nabla_{\boldsymbol{\theta}}\Psi(\boldsymbol{z})$

$$\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z})}\left[\left\|\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}}\Psi_{\boldsymbol{\theta}}(\boldsymbol{z})]\right\|_2^2\right] \leq \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z})}\left[\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}\left[\|\nabla_{\boldsymbol{\theta}}\Psi_{\boldsymbol{\theta}}(\boldsymbol{z})\|_2^2\right]\right] = \|\boldsymbol{\Sigma}^{-1}\|_2^2 < \infty$$

where we have applied Jensen's inequality. Therefore, $\mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}}\Psi(\boldsymbol{z})]$ is a finite constant and thus in $\mathcal{L}_p^2$.

Therefore, for the generative model defined in (14), the desired target $\boldsymbol{y}(\boldsymbol{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}}\log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})]\big|_{\boldsymbol{\theta}_t}$ for regression is in $\mathcal{L}_p^2$, which can be approximated arbitrarily well by KRR (see Section 2.4)) with more sleep samples. A similar analysis can show that, for Bernoulli likelihoods whose logits are parametrised by a Lipschitz neural network, the target for the regression is also in $\mathcal{L}_p^2$, with logits bounded from above and below.

A.3. Gradient of the log marginal likelihood w.r.t. parameters

To show the result used in (7), we start from the free energy (ELBO) lower bound on the log-likelihood $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$.

$$\begin{aligned}\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log\frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} &= \int q(\boldsymbol{z})\log\left[\frac{q(\boldsymbol{z})}{q(\boldsymbol{z})}\frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right]\mathrm{d}\boldsymbol{z} = \int q(\boldsymbol{z})\log\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{q(\boldsymbol{z})}\frac{q(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right]\mathrm{d}\boldsymbol{z} \\ &= \int q(\boldsymbol{z})\log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})\mathrm{d}\boldsymbol{z} - \int q(\boldsymbol{z})\log q(\boldsymbol{z})\mathrm{d}\boldsymbol{z} + D_{\mathrm{KL}}[q(\boldsymbol{z})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})] \\ &= \mathcal{F}(q,\boldsymbol{\theta}) + D_{\mathrm{KL}}[q(\boldsymbol{z})\|p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})],\end{aligned} \tag{15}$$

where we have defined

$$\mathcal{F}(q,\boldsymbol{\theta}) = \int q(\boldsymbol{z})\log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})\mathrm{d}\boldsymbol{z} - \int q(\boldsymbol{z})\log q(\boldsymbol{z})\mathrm{d}\boldsymbol{z} = \mathbb{E}_{q(\boldsymbol{z})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})] + \mathbb{H}[q].$$

The KL term in (15) is non-negative and is zero if $q(\boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, suggesting that

$$\log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathcal{F}(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}),\boldsymbol{\theta})$$

Replacing $q(\boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ in (15) and take derivative w.r.t. θ gives (assuming all derivatives and expectations exist)

$$\begin{aligned}\Delta_{\boldsymbol{\theta}}(\boldsymbol{x}) &:= \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \\ &= \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) \mathrm{d}\boldsymbol{z} - \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} \\ &= \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) \mathrm{d}\boldsymbol{z} + \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) \mathrm{d}\boldsymbol{z} \\ &\quad - \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} - \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z}. \end{aligned} \tag{16}$$

The last term in (16) is zero since it is the expectation of the score function

$$\int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \frac{1}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = 0.$$

The first and third terms in (16) combines to give

$$\int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})} \mathrm{d}\boldsymbol{z} = \int \nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = 0.$$

We are left with only the second term in (16)

$$\Delta_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})] = \nabla_{\boldsymbol{\theta}} \mathcal{F}(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}), \boldsymbol{\theta}). \tag{17}$$

To compute the update at the t'th iteration with $\boldsymbol{\theta} = \boldsymbol{\theta}_t$, and the expectation above is taken over a fixed posterior distribution $p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})$. We evaluate the above equation at $\boldsymbol{\theta}_t$, giving (7),

$$\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x}) := \Delta_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\boldsymbol{\theta}_t} = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})]\big|_{\boldsymbol{\theta}_t} = \nabla_{\boldsymbol{\theta}} \mathcal{F}(p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}), \boldsymbol{\theta})\big|_{\boldsymbol{\theta}_t}.$$

One can also pass $\nabla_{\boldsymbol{\theta}}$ and its evaluation inside the expectation (assuming derivatives exist) to obtain (8)

$$\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})]\big|_{\boldsymbol{\theta}_t} = \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})\big|_{\boldsymbol{\theta}_t}\right]$$

which is used for direct gradient estimation.

In fact, once we know the result above, going from the right-hand side to the left is much simpler:

$$\begin{aligned}\mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})\big|_{\boldsymbol{\theta}_t}\right] &= \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}\left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big|_{\boldsymbol{\theta}_t} + \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\boldsymbol{\theta}_t}\right] \\ &= \nabla_{\boldsymbol{\theta}} \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})]\big|_{\boldsymbol{\theta}_t} + \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{x})\big|_{\boldsymbol{\theta}_t} \\ &= 0 + \Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x}).\end{aligned}$$

Additionally, a quicker and more direct way to obtain (17) uses the "score trick" as follows

$$\nabla \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = \frac{1}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \int p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x}) \mathrm{d}\boldsymbol{z} = \mathbb{E}_{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})].$$

A.4. Miscellaneous results

Theorem A.1 (Gaussian concentration inequality (Boucheron et al., 2013, Theorem 5.6))**.** *Let $X = (X_1, \ldots, X_n)$ be a vector of n independednt standard normal random variables. Let $f : \mathbb{R}^n \to \mathbb{R}$ denote an L-Lipschitz function. Then, all $t > 0$,*

$$P\left[f(X) - \mathbb{E}f(X) \geq t\right] \leq e^{-t^2/(2L^2)}.$$

Lemma A.2. *Let s^2 be the sum of the diagonal elements of $\boldsymbol{\Sigma}$. Assume $\mathbb{E}_Z[\|\mathrm{NN}_{\boldsymbol{w}}(Z)\|_2] < \infty$. For the density function $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ defined in (14), for all $t > 2s$, we have*

$$P(|\|X\| - \mathbb{E}\|X\|| \geq t) \leq 2(e^{-\frac{t^2}{8L_1^2}} + e^{-\frac{(t/2-s)^2}{2L_2^2}})$$

Proof. Note that

$$P(|\|X\| - \mathbb{E}\|X\|| \geq t) \leq P(\|X\| - \mathbb{E}\|X\| \geq t) + P(-\|X\| + \mathbb{E}\|X\| \geq t).$$

We bound the first term below (the second term can be handled similarly).

We have

$$\begin{aligned} P(\|X\|_2 - E\|X\|_2 \geq t) &= \mathbb{E}_Z\big[P(\|X\|_2 - E\|X\|_2 \geq t\big|Z)\big] \\ &\leq \mathbb{E}_Z\big[P(\|X\|_2 - \mathbb{E}_{X|Z}[\|X\|_2] \geq t/2\big|Z)\big] + P(\mathbb{E}_{X|Z}[\|X\|_2] - \mathbb{E}[\|X\|_2] \geq t/2). \end{aligned}$$

By Theorem A.1, as $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) = \mathcal{N}(\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z}), \boldsymbol{\Sigma})$,

$$P(|\|X\|_2 - \mathbb{E}_{X|Z}[\|X\|_2]| \geq t/2\big|Z) \leq e^{-\frac{t^2}{8L_1^2}},$$

where $L_1 = \|\boldsymbol{\Sigma}^{1/2}\|_{\mathrm{op}}$ is the operator norm of $\boldsymbol{\Sigma}^{1/2}$. Therefore,

$$\mathbb{E}_Z\big[P(|\|X\|_2 - \mathbb{E}_{X|Z}[\|X\|_2]| \geq t/2\big|Z)\big] \leq e^{-\frac{t^2}{8L_1^2}}.$$

Let $\boldsymbol{\mu}(\boldsymbol{z}) = \mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z})$. For the second term, as

$$\begin{aligned} &\mathbb{E}_{X|Z}[\|X\|_2] \leq \sqrt{\mathbb{E}_{X|Z}[\|X - \boldsymbol{\mu}(Z)\|_2^2]} + \|\boldsymbol{\mu}(Z)\|_2 = s + \|\boldsymbol{\mu}(Z)\|_2, \\ &\mathbb{E}_Z[\|\boldsymbol{\mu}(Z)\|_2] = \mathbb{E}_Z\big[\|\mathbb{E}_{X|Z}[X]\|_2\big] \leq \mathbb{E}_Z\big[\mathbb{E}_{X|Z}[\|X\|_2]\big] = \mathbb{E}_X[\|X\|_2], \end{aligned}$$

we have

$$P(\mathbb{E}_{X|Z}[\|X\|_2] - \mathbb{E}[\|X\|_2] \geq t/2) \leq P(\|\boldsymbol{\mu}(Z)\| - \mathbb{E}_Z[\|\boldsymbol{\mu}(Z)\|] \geq t/2 - s).$$

By the Lipschitzness of $\mathrm{NN}_{\boldsymbol{w}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(0, \mathbf{I})$, we have for all $t > 2s$

$$P(\|\boldsymbol{\mu}(Z)\| - \mathbb{E}_Z[\|\boldsymbol{\mu}(Z)\|] \geq t/2 - s) \leq e^{-\frac{(t/2-s)^2}{2L_2}},$$

where L_2 is the Lipschitz constant of $\mathrm{NN}_{\boldsymbol{w}}$ (as a function of $\boldsymbol{z}$). Combining these bounds gives

$$P(\|X\|_2 - E\|X\|_2 \geq t) \leq e^{-\frac{t^2}{8L_1^2}} + e^{-\frac{(t/2-s)^2}{2L_2^2}}$$

□

B. Method details

B.1. Alternative gradient models

To ensure that the estimate of $J_{\boldsymbol{\theta}}$ can be differentiated w.r.t. $\boldsymbol{\theta}$ to obtain an estimate of $\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x})$, the gradient model needs to depend on model parameter $\boldsymbol{\theta}$. KRR satisfies this condition in an attractive way, because its prediction depends on $\boldsymbol{\theta}$ and $\boldsymbol{\gamma}$ in two separate factors, see (12). However, though theoretically consistent, KRR estimates the gradient at a cost of N^3 in memory and time, where N is the number of sleep samples. We discuss two alternative gradient models that could potentially be much faster, but there is no theoretical guarantee that $\nabla_{\boldsymbol{\theta}} \hat{J}_{\boldsymbol{\theta},\boldsymbol{\gamma}}\big|_{\boldsymbol{\theta}_t}$ is close to $\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x})$.

B.1.1. Generic function approximator

One can train a generic function estimator, such as a neural network, to estimate $J_{\boldsymbol{\theta}}(\boldsymbol{x})$. For such parametric models, the dependence on generative model parameters $\boldsymbol{\theta}$ can be encapsulated into gradient model parameters $\boldsymbol{\gamma}$ through gradient descent.

$$\boldsymbol{\gamma}(\boldsymbol{\theta}) \leftarrow \boldsymbol{\gamma}(\boldsymbol{\theta}) - \alpha \nabla_{\boldsymbol{\gamma}} L(\boldsymbol{\theta}, \boldsymbol{\gamma}), \quad L(\boldsymbol{\theta}, \boldsymbol{\gamma}) = \sum_{n=1}^{N} |\hat{J}_{\boldsymbol{\gamma}(\boldsymbol{\theta})}(\boldsymbol{x}_n) - \log p_{\boldsymbol{\theta}}(\boldsymbol{z}_n, \boldsymbol{x}_n)|^2$$

where α is the learning rate. As such, the estimator of $J_{\boldsymbol{\theta}}$ is better denoted as $\hat{J}_{\boldsymbol{\gamma}(\boldsymbol{\theta})}$ for a neural network with fixed hyperparameters. Evaluating $\nabla_{\boldsymbol{\theta}} \hat{J}_{\boldsymbol{\gamma}(\boldsymbol{\theta})}\big|_{\boldsymbol{\theta}_t}$ can be implemented, though less straightforwardly compared to the KRR gradient model. Alternatively, we can consider small perturbations around fixed-point of the loss, and derive a relationship between $\boldsymbol{\gamma}$ and $\boldsymbol{\theta}$ at a local minimum:

$$0 = \frac{\partial L}{\partial \boldsymbol{\gamma}}(\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta}, \boldsymbol{\gamma}(\boldsymbol{\theta} + \mathrm{d}\boldsymbol{\theta})) = \frac{\partial L}{\partial \boldsymbol{\gamma}}(\boldsymbol{\theta}, \boldsymbol{\gamma}(\boldsymbol{\theta})) + \mathrm{d}\boldsymbol{\theta} \frac{\partial}{\partial \boldsymbol{\theta}} \frac{\partial L}{\partial \boldsymbol{\gamma}}(\boldsymbol{\theta}, \boldsymbol{\gamma}(\boldsymbol{\theta})) + \mathrm{d}\boldsymbol{\gamma} \frac{\partial}{\partial \boldsymbol{\gamma}} \frac{\partial L}{\partial \boldsymbol{\gamma}}(\boldsymbol{\theta}, \boldsymbol{\gamma}(\boldsymbol{\theta})).$$

The first term on the RHS is zero, and rearranging gives $\frac{\mathrm{d}\boldsymbol{\gamma}(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} = -\left(\frac{\partial^2 L}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}}\right)^{-1} \frac{\partial^2 L}{\partial \boldsymbol{\theta} \partial \boldsymbol{\gamma}}$, assuming the inverse exists. Thus,

$$\frac{\mathrm{d}\hat{J}_{\boldsymbol{\gamma}(\boldsymbol{\theta})}(\boldsymbol{x})}{\mathrm{d}\boldsymbol{\theta}} = \frac{\partial \hat{J}_{\boldsymbol{\gamma}(\boldsymbol{\theta})}(\boldsymbol{x})}{\partial \boldsymbol{\gamma}} \frac{\mathrm{d}\boldsymbol{\gamma}(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} = -\frac{\partial \hat{J}_{\boldsymbol{\gamma}(\boldsymbol{\theta})}(\boldsymbol{x})}{\partial \boldsymbol{\gamma}} \left(\frac{\partial^2 L}{\partial \boldsymbol{\gamma} \partial \boldsymbol{\gamma}}\right)^{-1} \frac{\partial^2 L}{\partial \boldsymbol{\theta} \partial \boldsymbol{\gamma}}.$$

All of the factors can be computed by automatic differentiation since the objects being differentiated are all scalars. However, for a generic neural network, the Hessian of the loss w.r.t. $\boldsymbol{\gamma}$ may not exist, and computing it can be unstable.

B.1.2. Particle estimator

The prediction of the KRR estimator may not but a valid expectation. In other words, $\hat{J}_{\boldsymbol{\theta},\boldsymbol{\gamma}}(\boldsymbol{x})$ may not correspond to the expected log joint under any valid probability distribution. To address this issue, we can approximate $\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x})$ through a set of particles $\boldsymbol{z}'$ (in the space of the latent) generated from a simulator $S_{\boldsymbol{\gamma}} : (\boldsymbol{x}, \boldsymbol{n}) \to \boldsymbol{z}'$, where $\boldsymbol{\gamma}$ is the parameter of the simulator, $\boldsymbol{x}$ is an observation, and $\boldsymbol{n}$ is a noise source distributed as $\zeta(\boldsymbol{n})$. For all $\boldsymbol{x}$ from the generative model, we want the simulator to produce particles such that $\mathbb{E}_{\zeta(\boldsymbol{n})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(S(\boldsymbol{x}, \boldsymbol{n}), \boldsymbol{x})]\big|_{\boldsymbol{\theta}_t}$ estimates of $\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x})$. This can be achieved by solving

$$\min_{\boldsymbol{\gamma}} \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z},\boldsymbol{x})} \left[\left\| \mathbb{E}_{p(\boldsymbol{n})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(S_{\boldsymbol{\gamma}}(\boldsymbol{x}, \boldsymbol{n}), \boldsymbol{x})]\big|_{\boldsymbol{\theta}_t} - \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})\big|_{\boldsymbol{\theta}_t} \right\|^2 \right],$$

which is equivalent to

$$\min_{\boldsymbol{\gamma}} \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{x})} \left[\left\| \mathbb{E}_{p(\boldsymbol{n})}[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(S_{\boldsymbol{\gamma}}(\boldsymbol{x}, \boldsymbol{n}), \boldsymbol{x})]\big|_{\boldsymbol{\theta}_t} - \mathbb{E}_{p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})} \left[\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})\big|_{\boldsymbol{\theta}_t} \right] \right\|^2 \right]$$

due to the property of mean squared error (see Appendix A.1). We know that the optimal set of particles is distributed as the posterior $p_{\boldsymbol{\theta}_t}(\boldsymbol{z}|\boldsymbol{x})$, but minimising the cost above does not necessarily drive $S_{\boldsymbol{\gamma}}$ to produce posterior samples. Nonetheless, this set of particles is adequate to approximate $\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x})$. We refer to this scheme as amortised learning by particles (AL-P). We test this on sample quality experiments and found that the KIDs and FIDs were in general worse than even the vanilla VAE. Samples from the model trained by AL-P are shown in Figure 15 to Figure 20 in section Appendix C.7.

B.1.3. Relationship between KRR gradient model and importance sampling

The KRR gradient model approximates $\Delta_{\boldsymbol{\theta}_t}(\boldsymbol{x})$ by linearly weighting $\{\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{z}_n, \boldsymbol{x}_n)\}_{n=1}^N$. This is similar to other reweighting schemes (e.g. (Dieng & Paisley, 2019)), with the most simple one being importance sampling where the proposals are from the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and the weights are normalised density ratios $p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})/p_{\boldsymbol{\theta}}(\boldsymbol{z})$. Importance sampling is an unbiased estimation method, but has huge variance and requires at least exponentially many samples as the KL divergence between the posterior and prior (Chatterjee et al., 2018).

It would then appear that KRR should perform similarly with importance sampling in estimating $\Delta(\boldsymbol{x})$, but, on closer look, they uses slightly different sources of information for estimation. KRR uses a set of samples $(\boldsymbol{z}_n, \boldsymbol{x}_n) \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})$, whereas importance sampling uses $\boldsymbol{z}_n \sim p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}^*)$. In computing the weights for a particular $\boldsymbol{x}^*$ from the dataset, KRR compares $\boldsymbol{x}^*$ with all sleep samples $\{\boldsymbol{x}_n\}_{n=1}^N$, using a similarity metric determined by the kernel function. The weights $\boldsymbol{\alpha}$ also takes into account of the similarities between all sleep samples. On the other hand, importance sampling uses $p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x}^*)$ for a given $\boldsymbol{x}^*$ and computes the weights for each sample of $\boldsymbol{z}$ independently of each other. In addition, the importance sampling weights are constrained to be non-negative and sum up to one, whereas the weights in KRR are not constrained and thus can be more flexible.

B.2. Kernel architecture

In all experiments, we used a squared-exponential kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-0.5\|\boldsymbol{\phi}_{\boldsymbol{w}}(\boldsymbol{x}) - \boldsymbol{\phi}_{\boldsymbol{w}}(\boldsymbol{x}')\|_2^2/\sigma^2)$. The feature $\boldsymbol{\phi}_{\boldsymbol{w}}$ can be the identity function, a linear projection, or a linear projection followed by batch normalisation , see Table 1

which lists the architectures used for each experiment. The linear projection and batch normalisation are primarily used on high-dimensional benchmark datasets. Nonlinear projections, such as deep neural networks, did not give significant improvement while consuming more memory. The bandwidth σ is initialised as the median of the distance between $\boldsymbol{\phi}_{\boldsymbol{w}}(\boldsymbol{x}^{(n)})$ where $\boldsymbol{x}^{(n)} \sim p_{\boldsymbol{\theta}_t}(\boldsymbol{x})$.

C. Experimental details

We list the model and training parameters used to run each experiment in Table 1. The batch size is 100 except for dynamical models and neural process where the batch size is 1.

C.1. Gradient estimation

The toy generative model has $z_1, z_2 \sim \mathcal{N}(0,1), x|z \sim \mathcal{N}(\mathrm{softplus}(\boldsymbol{w} \cdot \boldsymbol{z}) - \|\boldsymbol{w}\|_2^2, \sigma_2^2)$. The observations are 100 samples for drawn form the model with $w_1 = w_2 = 1, \sigma = 0.1$. Note that the ML solution for this synthetic problem is not unique.

For variational learning, the approximate posterior is a factorised Gaussian that minimises the ELBO. The gradient of ELBO was approximated by samples. The mean and variances are initialised as the standard Gaussian and are optimised by Adam with step size 0.01 for 300 iterations, which is sufficient for convergence. For ground truth, we estimated the gradient by importance sampling, with 5×10^4 samples proposed from the prior.

C.2. Spherical prior

The data are 16×16 Gabor images. The orientation is uniformly distributed over one period 0 to π. The generative network is taken from the first two deconvolutional layers of DCGAN so that the output size is 16×16. For VAE, we used the symmetric convolutional neural network for the encoder and a factorised Gaussian posterior. For $\mathcal{S}$-VAE, a von Mises-Fisher distribution is used as the posterior.

C.3. Hierarchical models

The penalty assigned to probability vector $\boldsymbol{m}$ in the categorical distribution is the log pdf of a Dirichlet prior $\log p(\boldsymbol{q}) = (\alpha - 1)\sum_i \log q_i + \text{const}$, where $q_i = e^{m_i} / \sum_j e^{m_j}$. We use $\alpha = 0.999$. Similarly, for the k'th component in the mixture, the Normal-InverseWishart distribution has log-likelihood that penalises $\|\mu_k\|$, $\log|\Sigma_k|$ and $\mathrm{Tr}(\Sigma_k^{-1})$. In addition, we also penalise the l-2 norm of neural network weights. These penalisation strength are set to 10^{-4}.

The relative maximum mean discrepancy (MMD) test (Bounliphone et al., 2016) is used for model comparison based on generated samples. Denote the set of real data by $\mathcal{D}$ and the set of generated samples from model A by $\mathcal{D}^{\mathrm{A}}$. The null hypothesis for this test is $\mathrm{MMD}(\mathcal{D}, \mathcal{D}^{\mathrm{SIN}}) < \mathrm{MMD}(\mathcal{D}, \mathcal{D}^{\mathrm{ALWS}})$, where MMD is the MMD distance between two sets of samples. The test returns a p-value of 0.514 based on 1500 samples from each of the three distributions, suggesting that the two models perform almost equally well on learning this data distribution. We note that SIN is trained on a full Bayesian version of the model, and the samples are reconstructions given the real dataset, giving an advantage for SIN.

C.4. Parameter identification

The linear basis (weights) are the top 36 independent components of natural images discovered by the FastICA algorithm. Each component is subtracted by their mean and normalised to have unit length. The synthesised dataset is standardised by subtracting the mean and dividing by the standard deviation. The kernel is augmented with an adaptive linear neural network feature with 300 outputs. Using 200 features produces very similar results. The regularisation strength is fixed at $\lambda = 0.001$. Adapting the filters results in slightly different filters as shown in Figure 9.

C.5. Neural process

Introduction. We briefly review the neural processes (NPs, Garnelo et al. (2018)). Suppose there is a distribution over function $f \sim \mathcal{P}(f),\ f : \mathcal{X} \to \mathcal{Y}$. We observe information a given function f through its potentially noisy values at a set of inputs $(\boldsymbol{x}, \boldsymbol{y})|f$. The task is the following: given a set of context pairs $\mathcal{D} := \{(\boldsymbol{x}_k^C, \boldsymbol{y}_k^C)\}_{k=1}^K$ drawn from an unobserved function, infer the distribution of the function value at a set of target inputs $\{\boldsymbol{x}_m^T\}_{m=1}^M$.

NPs represent the posterior of f given $\mathcal{C}$ by a random variable $\boldsymbol{z}$, which is combined with $\boldsymbol{x}_m^T$ to predict the function value.

Experiment	latent dim	data dim	M(data)	N (sleep)	L (val)	λ	# proj	batch norm?	gen lr	grad lr	nepoch
gradient estimation	2	1	100	5 000	–	0.01(f)	–	no	–	–	–
spherical prior	1	256	10 000	2 000	200	0.01	300	no	0.001	0.001	30
pinwheel	2	2	2 500	1 000	200	0.01	–	no	0.001	0.001	2 500
Independent component	36	256	100 000	2 000	200	0.001(f)	300	no	0.001	0.01	100
Matrix factorisation	100	784	5 000	2 000	150	0.001	300	yes	0.001	0.001	300
neural process	50	8	10 000	4 000	200	0.001	–	no	0.0001	0.0001	50
nonlinear oscillation	2/time step	600	1	2 000	200	0.001	200	no	0.001	0.001	5 000
Hodgkin-Huxley	3/time step	1 000	1	2 000	200	0.001	200	no	0.001	0.001	50 000
ecology	1/time step	180	1	2 000	200	0.001	–	no	0.001	0.001	50 000
B-MNIST	16	1 024	60 000	2 000	200	0.1 (f)	300	yes	0.001	0.001	50
MNIST	16	1 024	60 000	2 000	200	0.1 (f)	300	yes	0.001	0.001	50
Fashion	16	1 024	60 000	2 000	200	0.1 (f)	300	yes	0.001	0.001	50
Natural	16	1 024	100 000	2 000	200	0.1 (f)	300	yes	0.001	0.001	50
CIFAR	16	3 072	50 000	2 000	200	0.1 (f)	300	yes	0.001	0.001	50
CelebA	16	3 072	100 000	2 000	200	0.1 (f)	300	yes	0.001	0.001	50

Table 1. Data properties, and model and training parameters of ALWS for each experiment. The regularisation strength λ is sometimes fixed as indicated by (f). See Appendix B.2 for kernel architectures.

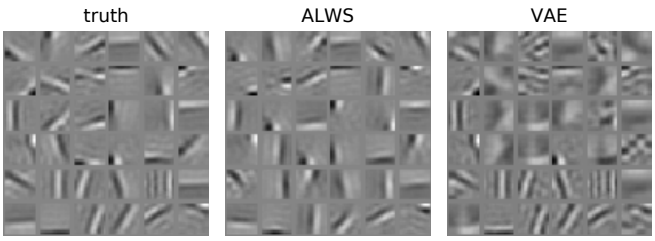

Figure 9. Same as Figure 5 but with λ adaptive.

During training, the training data comprises multiple sets of input-output pairs, and each set is always conditioned on one particular $f \sim \mathcal{P}$. The training data are split into a context set $\mathcal{C}$, used to condition the representation $\boldsymbol{z}$, and a target set $\{(\boldsymbol{x}_m^T, \boldsymbol{y}_m^T)\}_{m=1}^M$, used to evaluate the likelihood of $\boldsymbol{y}_m^T$ given $\boldsymbol{z}$ and $\boldsymbol{x}_m^T$. Formally, the generative model is specified by

$$\begin{aligned} \boldsymbol{r} &= \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\rho}_{\boldsymbol{\theta}}(\boldsymbol{x}_k^C, \boldsymbol{y}_k^C) \\ p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{r}) &= \mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{\theta}}^C(\boldsymbol{r}), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^C(\boldsymbol{r})) \\ p_{\boldsymbol{\theta}}(\{\boldsymbol{y}_m^T\}|\{\boldsymbol{x}_m^T\}, \boldsymbol{z}) &= \prod_{m=1}^{M} \mathcal{N}(\boldsymbol{y}_m^T|\boldsymbol{\mu}_{\boldsymbol{\theta}}^T(\boldsymbol{z}, \boldsymbol{x}_m^T), \boldsymbol{\Sigma}_{\boldsymbol{\theta}}^T(\boldsymbol{z}, \boldsymbol{x}_m^T)). \end{aligned}$$

In short, a latent representation of the context $\boldsymbol{z}$ is drawn from a normal distribution with parameters formed by an exchangeable function of the context set $\mathcal{C}$, and the likelihood on the target outputs are i.i.d. Gaussian conditioned on $\boldsymbol{z}$ and $\boldsymbol{x}_m^T$. The objective for learning is to maximise the likelihood of the target output conditioned on the corresponding context set from the same underlying f and the target input. Once trained, the neural process is able to produce samples from the distribution of function values (target outputs) at context inputs.

The encoding function $\boldsymbol{\rho}_{\boldsymbol{\theta}}$ plays the role of an inferential model, but we can view it as a function that parametrises the "prior" distribution on $\boldsymbol{z}$ given the context set, and the parameters in $\boldsymbol{\rho}$ can be regarded as belonging to the generative model. The gradient model trained by KRR also needs to be conditioned on each context set, but for simplicity, we train a gradient model for a single context followed by $\boldsymbol{\theta}$ update. Garnelo et al. (2018) trained the neural processes by maximising an ELBO with posteriors of the form

$$q(\boldsymbol{z}|\mathcal{C}, \mathcal{T}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{r}^{CT}), \quad \boldsymbol{r}^{CT} = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{\rho}_{\boldsymbol{\theta}}(\boldsymbol{x}_k^C, \boldsymbol{y}_k^C) + \frac{1}{M}\sum_{m=1}^{M} \boldsymbol{\rho}_{\boldsymbol{\theta}}(\boldsymbol{x}_m^T, \boldsymbol{y}_m^T),$$

which is an approximation.

Experiments. We train a neural process on a $\mathcal{P}(f)$ that have samples as shown in Figure 10 (top). They are sinusoids with random amplitudes and phase shifts and supported on $[-\pi, \pi]$. The observations are contaminated with Gaussian noise with standard deviation 0.1. Conditioning the function with a context input around $-\pi$, 0.0 and π induces large uncertainty over f; thus, we can use this to probe the representation of uncertainty.

In the NP model, the representation $\boldsymbol{r}$ and $\boldsymbol{z}$ are both 50-dimensional. And the encoding and decoding networks are fully connected with ReLU nonlinearities. During training, the number of context pairs $K = 4$, and the target set contains the context pairs and an additional four pairs, so $M = 8$ and $\mathcal{C} \subset \mathcal{T}$. The gradient model is trained for each given context set, and hence the batch size is 1. A small learning rate of 0.0001 is used for all models and parameters. The gradient model is trained to take sleep samples $\boldsymbol{y}_m^T$ evaluated for this single $\mathcal{C}$ at each $\boldsymbol{x}_m^T$. The kernel takes $\{\boldsymbol{y}_m\}_{m=1}^8$ as a single vector. We note that other kernels on sets could be used.

During test time, we evaluate the predicted function value of a dense grid of points in $[-\pi, \pi]$ given 1 to 4 context pairs. As shown in Figure 10 (lower panels), when the number of context points is small, the model trained with ALWS makes more accurate predictions, and better reflects the uncertainty of the function value when the context set is uninformative. Given four context pairs (as in training), we test the learned model on 500 functions from $\mathcal{P}(f)$ and evaluate how close samples of $\mathcal{P}(\boldsymbol{f}|\mathcal{C}, \boldsymbol{x}_m^T)$ are to the true function at $M = 100$ target locations. We use either the posterior mean or a random

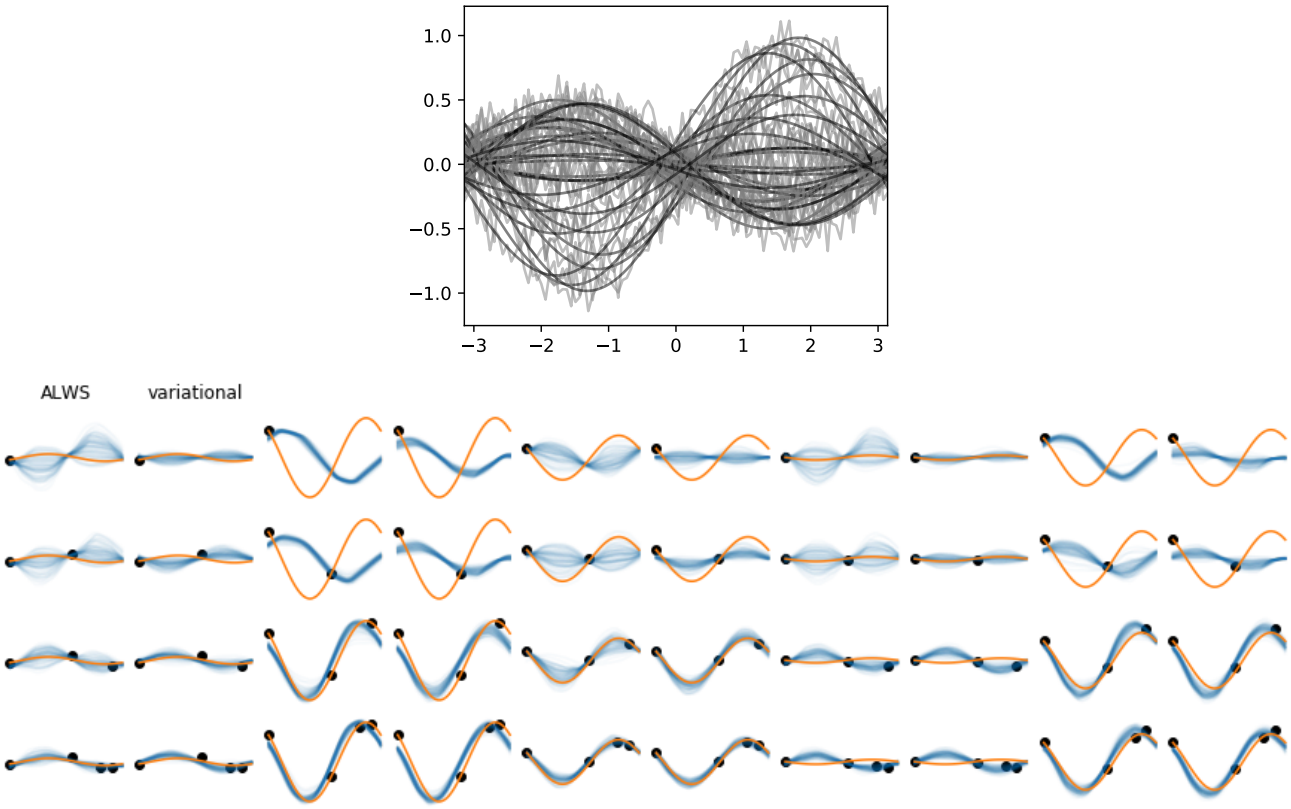

Figure 10. Neural processes. Top: samples from prior distribution of functions. Black: Latent function. Grey: noisy observations. Bottom: posterior samples (blue lines) from ALWS (odd columns) and the original variational method (even columns). Orange lines are true latent functions f. Black dots are context pairs.

posterior sample from the posterior as a point estimate, and measure the performance by mean squared error. We find that the errors are significantly smaller for ALWS-trained model based on paired tests for the posterior mean prediction (paired t-test, $t = -3.47, p = 0.00056$; mean of ALWS, -5.11; variational, -4.99. Wilcoxson test, $W = 44837.0, p = 3.7 \times 10^{-8}$, median of ALWS, -5.11, variational, -4.98) and the random sample prediction (paired t-test, $t = -2.09, p = 0.037$; mean of ALWS, -4.87; variational, -4.77. Wilcoxson test, $W = 53762.0, p = 0.0061$, median of ALWS, -4.91, variational, -4.69).

C.6. Nonlinear dynamic model

We run ALWS for generative models whose priors are defined through nonlinear transitions in time. In all of the experiments, we treat each sequence as a single multi-dimensional data point.

C.6.1. Nonlinear oscillations

We generate data from a nonlinear oscillation process according to the following equations used by Wenliang & Sahani (2019)

$$\boldsymbol{z}_t = \mathrm{Rot}(\boldsymbol{z}_{t-1}) + \boldsymbol{\epsilon}_t^{(z)}, \quad \boldsymbol{x}_t = \mathrm{Img}(z_{t,1}) + \boldsymbol{\epsilon}_t^{(x)}$$
$$\mathrm{Rot}(\boldsymbol{z}_t) = \boldsymbol{R}_\alpha \boldsymbol{z}_t \frac{r(\|\boldsymbol{z}_t\|_2)}{\|\boldsymbol{z}_t\|_2}, \quad r(a) = \mathrm{sigmoid}(4(a-0.3)), \quad [\mathrm{Img}(z)]_i = \exp(-0.5(z-\bar{z}_i)^2/0.3^2)$$

where $\boldsymbol{R}_\alpha$ is a rotation matrix by α radians, Img maps one of the latent dimensions into a 20-pixel image through Gaussian bumps with evenly spaced centers at $\bar{z}_i, i \in \{1, \ldots, 20\}$. Intuitively, the latent $\boldsymbol{z}$ is rotated by α and scaled radially so that its length remains close to 1. Samples of $\boldsymbol{x}_t$ for all $t \in \{1, \ldots, T\}$ can be plotted side by side as a $20 \times T$ image, which is shown in Figure 11 (top).

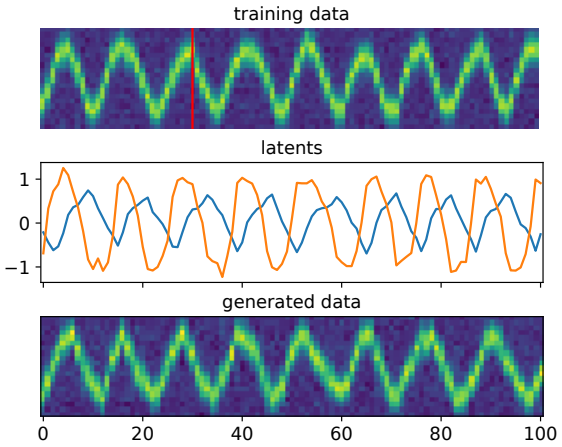

Figure 11. Nonlinear oscillations. Top: an example trajectory. Only the first 30 time steps marked by the red line is used for training. Three such 30 time step traces are used for training. Middle: latent space learned by ALWS. Bottom: Generated trajectory.

We train the following generative model:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \mathcal{N}(\boldsymbol{z}_t; \mathrm{NN}^{(z)}_{\boldsymbol{w}}(\boldsymbol{z}_{t-1}), \boldsymbol{\Sigma}_z), \qquad p_{\boldsymbol{\theta}}(\boldsymbol{x}_t|\boldsymbol{z}_t) = \mathcal{N}(\boldsymbol{x}_t; \mathrm{NN}^{(x)}_{\boldsymbol{w}}(\boldsymbol{z}_t), \boldsymbol{\Sigma}_x),$$

where the parameters are the weights and biases in the neural networks (NN), and the diagonal covariance matrices $\boldsymbol{\Sigma}_{(\cdot)}$'s. The number of units are fully connected with $2 \to 20 \to 2$ neurons for $\mathrm{NN}^{(z)}$ and $2 \to 20 \to 20$ for $\mathrm{NN}^{(x)}$. The $\tanh$ is used as the nonlinearity. We train the model on a single sequence of 30 time steps and then generate a 100-step sequence of the learnt latents and observations shown in Figure 11. The latents correctly capture the position, which directly sets the data, and the velocity, which needs to be learned from data.

C.6.2. Hodgkin-Huxley (HH) equations

The HH equations are described by

$$\begin{aligned} C_m \dot{V}(t) &= -g_l[V(t) - E_l] - \bar{g}_N m^3(t) h(t)[V(t) - E_N] - \bar{g}_K n^4(t)[V(t) - E_K] + I_{\text{in}}(t) + \epsilon(t) \\ \dot{e}(t) &= \alpha_e(V(t))[1 - e(t)] - \beta_e(V(t)) e(t), \quad e \in \{m, h, n\} \end{aligned}$$

where α_e and β_e are nonlinear functions of $V(t)$ involving a parameter V_T that sets the threshold for action potentials, see (Pospischil et al., 2008) for details.

We used forward-Euler method for simulation with a time step of $\Delta t = 0.05ms$. At each step of the simulation, we add a small Gaussian noise of standard deviation $\sigma_z = 0.1$mV to V_t as process noise. The measurements noise added to observations (but not propagated to V_{t+1}) is Gaussian with standard deviation 1.0mV. There 10 parameters for the resulting discrete-time state-space model: $\boldsymbol{\theta} = \{C_m, g_l, E_l, \bar{g}_N, E_n, \bar{g}_K, E_K, V_t, \sigma_z, \sigma_x\}$.

We train and test the model under different input current sequences I_{in}. The results are shown in Figure 12. We simulate a single trajectory from the model with some true parameters and a noisy current injection shown in Figure 12(1st row). This sequence is used as the training data Figure 12(2nd row, dotted). We then perturb these parameters, making the simulated trajectories unrealistic Figure 12(3rd row). After training, the simulated trajectories look almost identical to the training data Figure 12(2nd row, solid). To test whether the learned model can be used for prediction under a different current injection, we simulate trajectories given an unseen test current Figure 12(4th row). The responses of membrane potential under true parameters are shown in Figure 12(5th row). Samples from the trained model Figure 12(6th row, solid) under this unseen current are very similar to the trajectories given real parameters, showing generally correct phase, periodicity and amplitude. The simulated responses have less variation between trajectories, which could be due to training under a single sequence. Indeed, not all parameters converge to the true parameters Figure 12 (bottom panels).

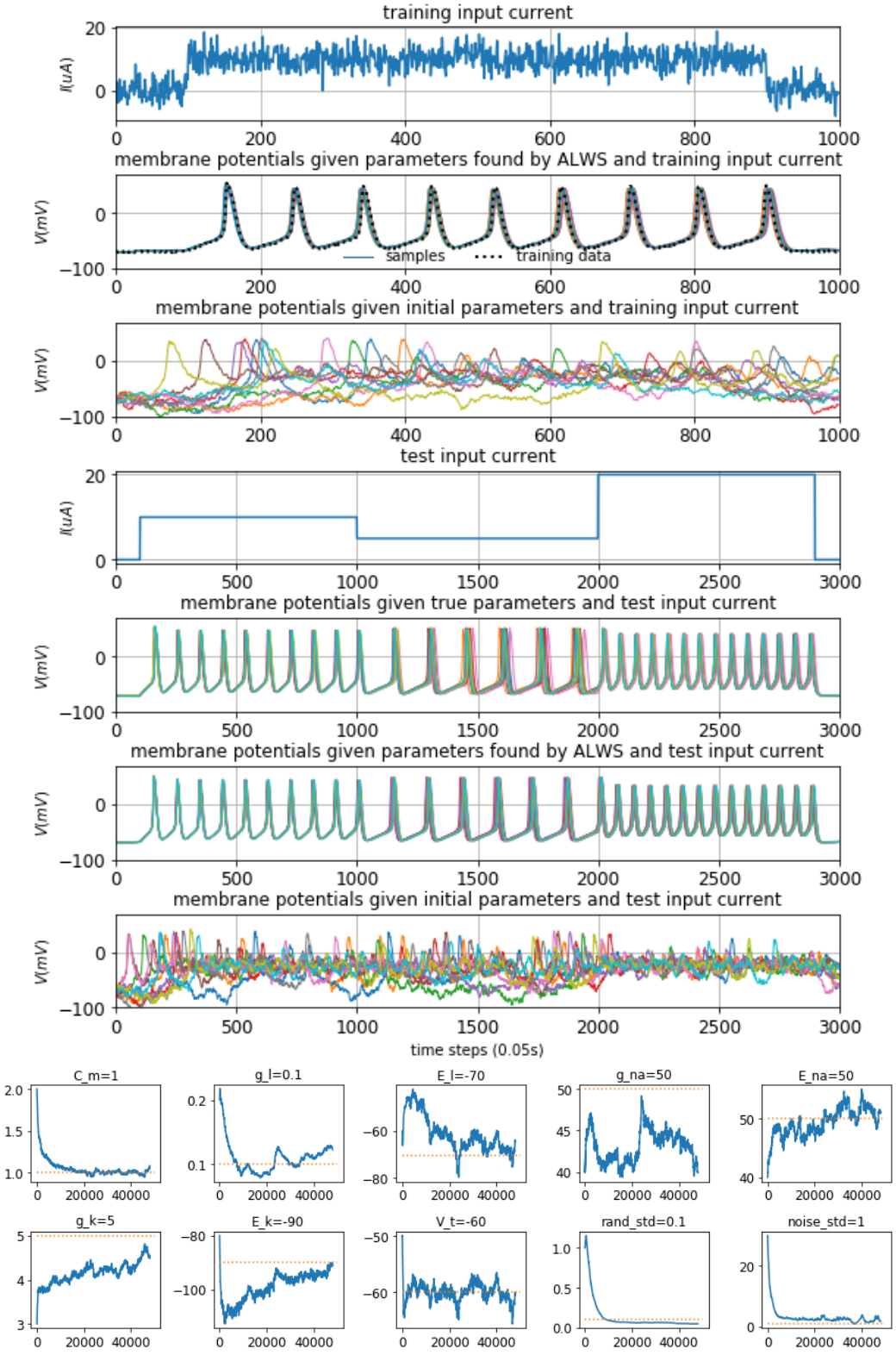

Figure 12. Hodgkin Huxley simulations. Top seven panels: 1st row, input current I_{in} during training. 2nd-3rd rows, trajectories given learnt and initial parameters under training input current. 4th row, test input current. 5th-7rd rows, trajectories given true, learnt and initial parameters under test input current. Bottom 10 panels: Blue solid: parameter value at each iteration. Yellow dashed: true parameter values.

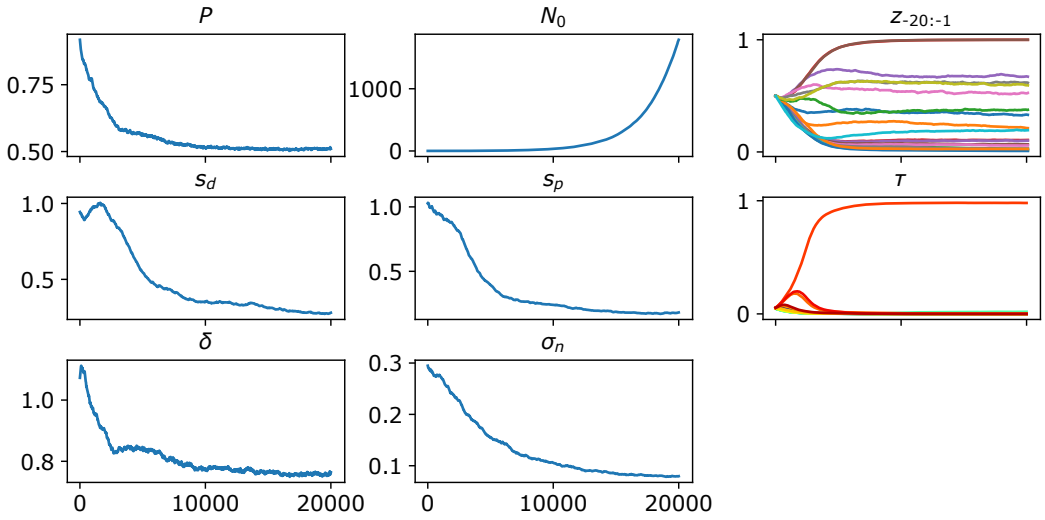

Figure 13. Evolution of parameters in the ecological model for blowfly population.

C.6.3. Ecological data

We train a model that describes the evolution of blowfly population size under food limitation (Wood, 2010). The model is given by

$$\tau \sim \text{Categorical}(\boldsymbol{m}), \tau \in \{1, \dots, 20\}, \quad e_t \sim \text{Gamma}(\frac{1}{\sigma_p^2}, \sigma_p^2), \quad \epsilon_t \sim \text{Gamma}(\frac{1}{\sigma_d^2}, \sigma_d^2),$$

$$z_t = P x_{t-\tau} \exp(-\frac{x_{t-\tau}}{N_0}) + x_t \exp(-\delta \epsilon_t), \quad p(x_t|z_t) = \text{LogNormal}(\log(z_t), \sigma_n^2)$$

Note that τ is a discrete delay drawn from a categorical distribution with logit parameters $\boldsymbol{m}$, e_t and ϵ_t are stochastic variations in births and deaths following Gamma distribution with a common mean 1.0 and standard deviations σ_p^2 and σ_d^2, respectively. The observation is noisy with log-normal noise so that $\boldsymbol{x}_t$ remains positive. Observations in the first 20 time steps depend on some past data that is not observed, so we modelled these past data $x_{-20:-1}$ as parameters, which are constrained to be between 0 and 1.0. Thus, this model has parameters $\boldsymbol{\theta} = \{\boldsymbol{m}, \sigma_d, \sigma_p, P, N_0, \delta, \sigma_n, x_{-20:-1}\}$

We fit the model on a data sequence of length 180, normalised to be between 0 and 1.0. The evolution of parameters is shown in Figure 13. As our training objective is different from that of ABC methods, we do not make direct quantitative comparison with them. But compared with the samples from three ABC methods shown in (Park et al., 2016) (Figure 2B), it is clear that samples from ALWS are visually more similar to the training data.

C.7. Sample quality on benchmark datasets

C.7.1. Data processing

All images have 32×32 pixels by their original sizes (Natural, CIFAR-10), or by zero-padding (MNIST,F-MNIST) or interpolation (CelebA). The binarised MNIST is statically binarised once before training. Each pixel is set to 1 with probability equal to the pixel value after rescaling to between 0 and 1. The natural images [9] are patches from large natural scenes. No clipping is applied. Original MNIST Fashion MNIST, CIFAR-10 and CelebA images are rescaled to between -1.0 and 1.0.

C.7.2. Model and training details

All methods use the same neural network as the DCGAN without the last convolutional layer to make the image size 32. Batch size is 100 for each update of generative and gradient model parameters. We run each algorithm on each dataset with 10 different initialisations. The neural network in the generative model has ReLU nonlinearities in intermediate layers. The

[9] github.com/hunse/vanhateren

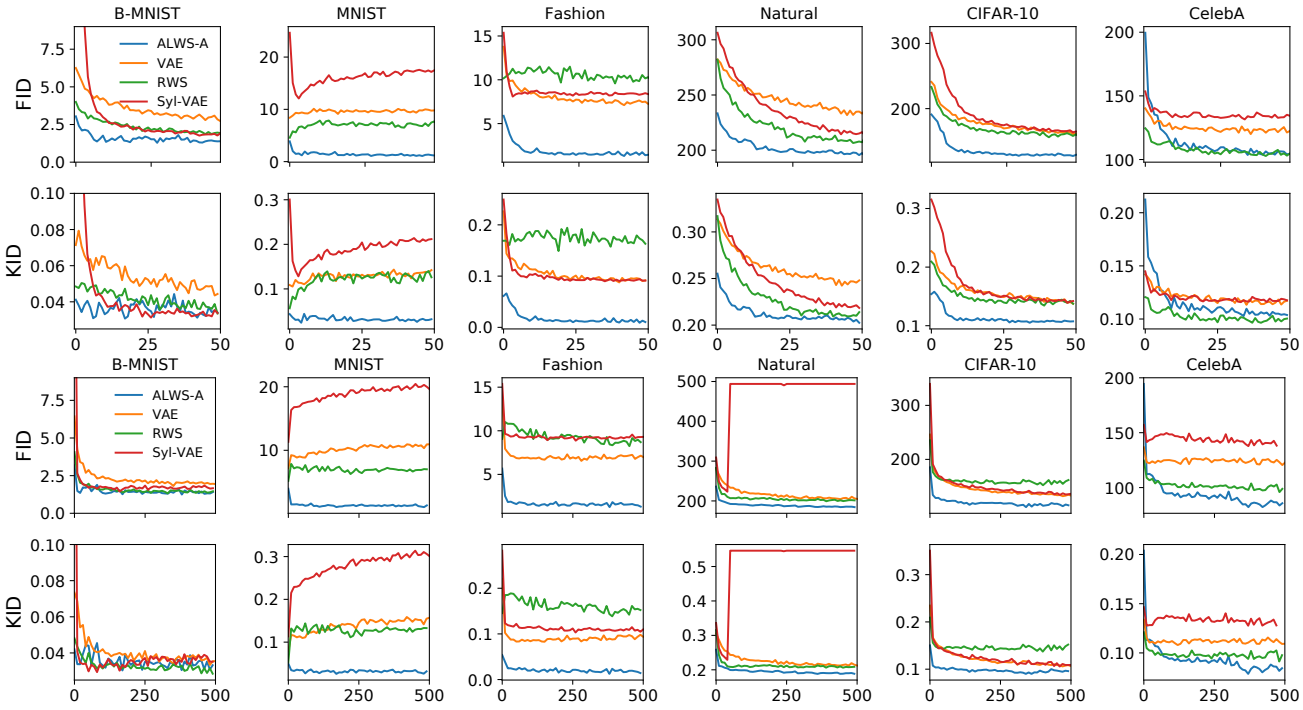

Figure 14. KID and FID scores at the end of each epoch for selected algorithms on convolutional architecture. Top two rows show distances during a run of 50 iterations at every iteration, and the bottom two rows show another run of 500 iterations at every 10th iteration.

nonlinearity for the final layer depends on the dataset: it is sigmoid for binary MNIST, linear natural images, and $\tanh$ for the other datasets.

All methods are trained for 50 epochs except for SIVI which was trained for 1 000 epochs. The optimizer is Adam with a fixed learning rate of 0.001. For ALWS, we use 2 000 sleep samples for training the gradient model. The kernel is augmented by linear projection to 300 dimensions for all datasets. A larger number of output dimension produced better results, but induces longer run time. The weights of the projection are updated after the first five epochs. The regularisation parameter λ is fixed at 0.1; this helps sample quality for CIFAR and CelebA, but does not affect or worsens sample quality for the other datasets. For ALWS-F, a fixed random projection is used throughout training. For ALWS-A, the linear weights are training at each parameter update after 5 epochs, using the two-stage training.

For VAE, the encoder network is symmetrical to the generative network and is appended with a final linear layer for posterior statistics.

For Syl-VAE. We change the gated convolutional layer in the decoder network to the same network as all the other methods. Other parts of the model remain the same. We use the orthogonal flow. A lower learning rate of 0.0005 is used for stability.

For SIVI, we find the model is unstable for learning rate of 0.001, so we change it to 0.0001. It also takes more epochs to produce good samples, so we train for 1000 epochs. We use $J = 10$ proposals from the Gaussian posterior.

For RWS, each parameter update is accompanied with both wake and sleep updates of the encoder parameters, using $K = 50$ proposals. A larger K can cause lower signal-to-noise ratio of the update for the encoder network.

For WGAN-GP, learning is unstable for a learning rate of 0.001, so we train the model using a learning rate of 0.0001 for 50 epochs, which was not sufficient for it to produce good images. We also run WGAN-GP for 500 epochs on all datasets and show the samples from Figure 15 to Figure 20. We show the results of WGAN-GP just for reference, as it is not trained using the maximum likelihood objective.

To evaluate the quality, we use standard metrics FID and KID, which are computed using features of penultimate layers of neural networks pre-trained on relevant datasets. For both MNISTs, the features are from the LeNet trained to classify MNIST digits. For Fashion, we used the LeNet network trained to classify the objects. For Natural, CIFAR-10 and CelebA,

we use inception network trained on ImageNet classification. For Natural, we duplicate the image along the channel axis to fill the three colour channels.

ALWS-A has lower FID and KID than other maximum likelihood methods in most cases, especially on original MNIST and Fashion MNIST, but does not reach the level of WGAN-GP.

The KID and FID values during training are shown in Figure 14. ALWS performs consistently better at every training epoch on all datasets except B-MNIST. On MNIST, ALWS-A converged the fastest and generates samples with stable quality. On CIFAR, ALWS-A and RWS converged faster than the others, but VAE and Syl-VAE converge very slowly. We note that these figures are plotted against epochs, not wall-clock time. The run time of ALWS is much longer than the other methods, taking around 3.5 seconds per iteration on a GeForce 1080 GPU, or 2.5 seconds on a Quadro P5000 with kernel adaptation. Nonetheless, this cost is worth the improvement over other maximum likelihood methods.

The samples from all methods are shown in Figure 15 to Figure 20. These include samples from models presented in the main text, the WGAN-GP for 500 epochs, and the AL-P algorithm introduced in Appendix B.1.2.

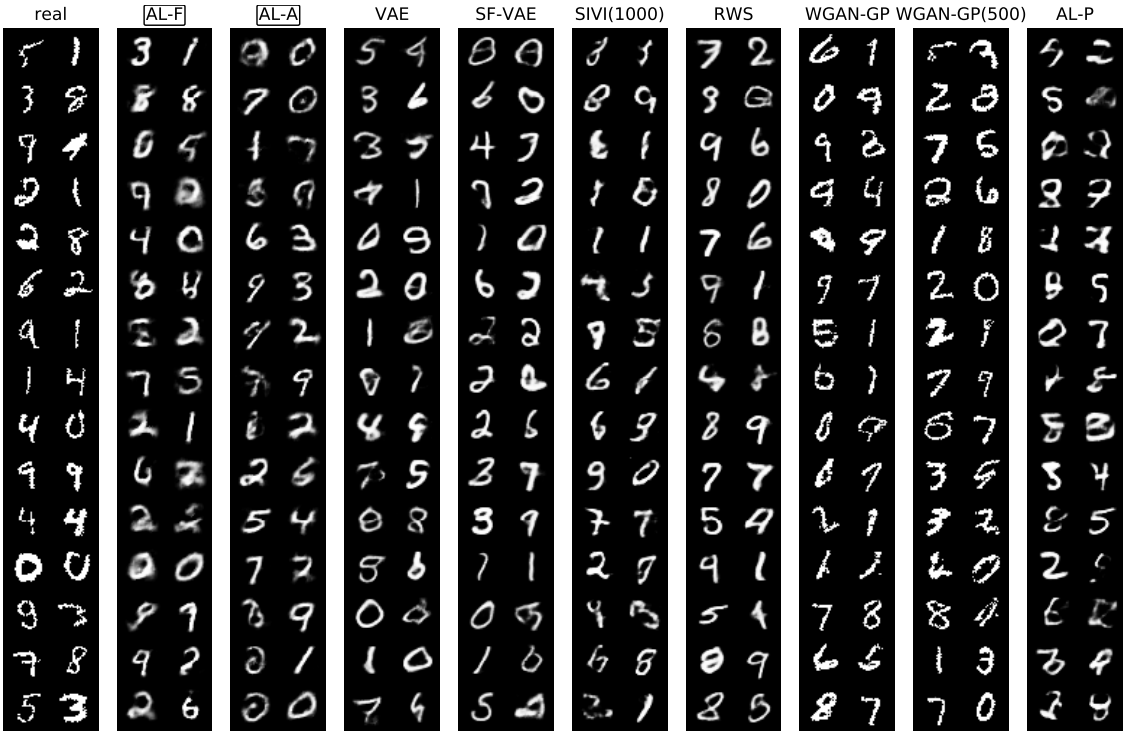

Figure 15. Samples for B-MNIST. Our main algorithms presented in the main text are highlighted in box. Each model is trained for 50 epochs, except otherwise indicated in parenthesis next to algorithm name.

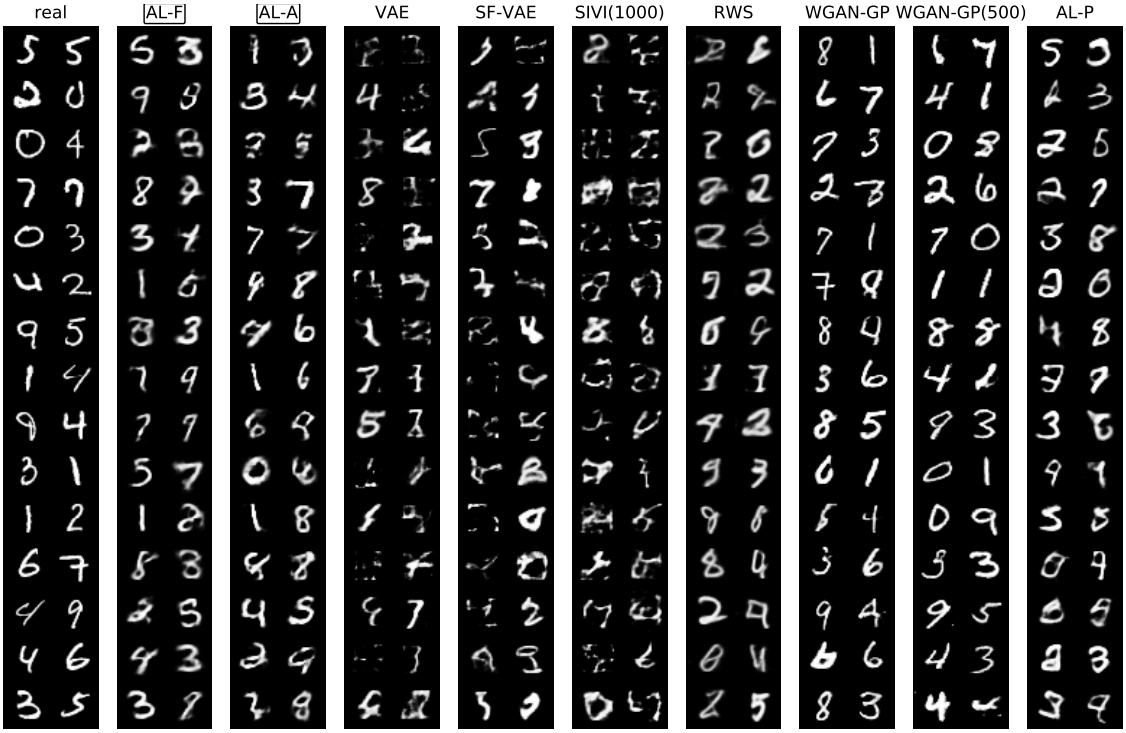

Figure 16. Samples for MNIST. Our main algorithm is highlighted in box. Each model is trained for 50 epochs, except otherwise indicated in parenthesis next to algorithm name.

Figure 17. Samples for Fashion. Our main algorithms presented in the main text are highlighted in box. Each model is trained for 50 epochs, except otherwise indicated in parenthesis next to algorithm name.

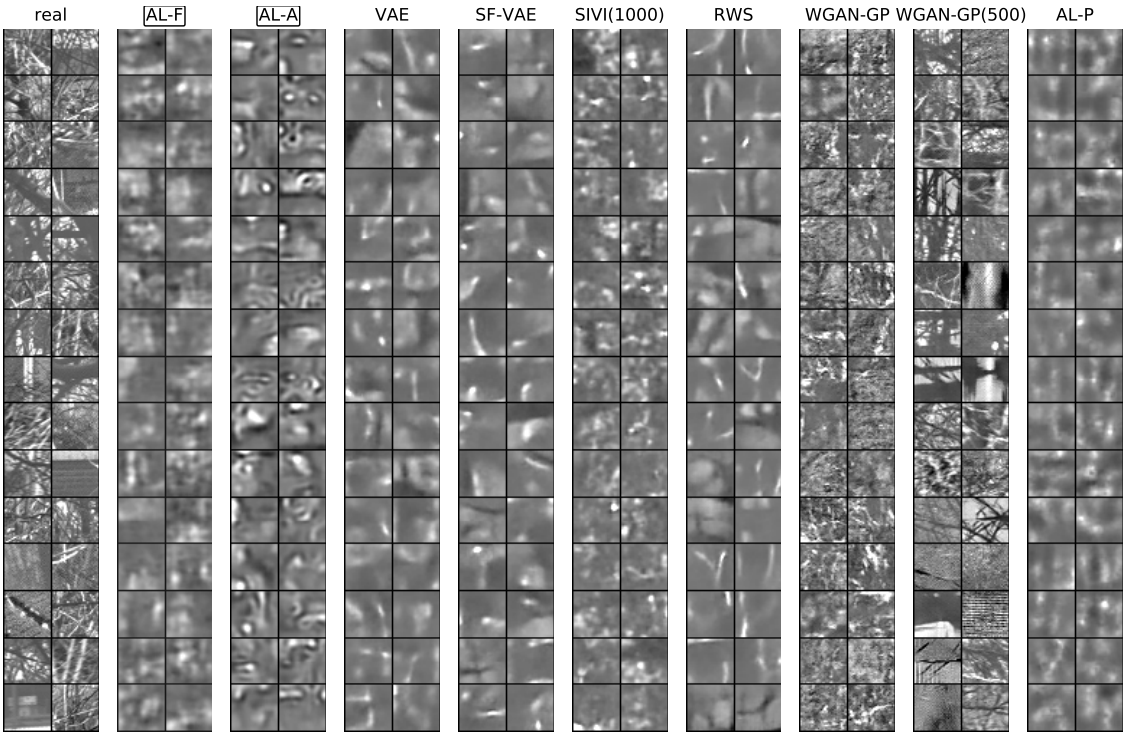

Figure 18. Samples for Natural. Our main algorithm is highlighted in box. Each model is trained for 50 epochs, except otherwise indicated in parenthesis next to algorithm name.

Figure 19. Samples for CIFAR-10. Our main algorithms presented in the main text are highlighted in box. Each model is trained for 50 epochs, except otherwise indicated in parenthesis next to algorithm name.

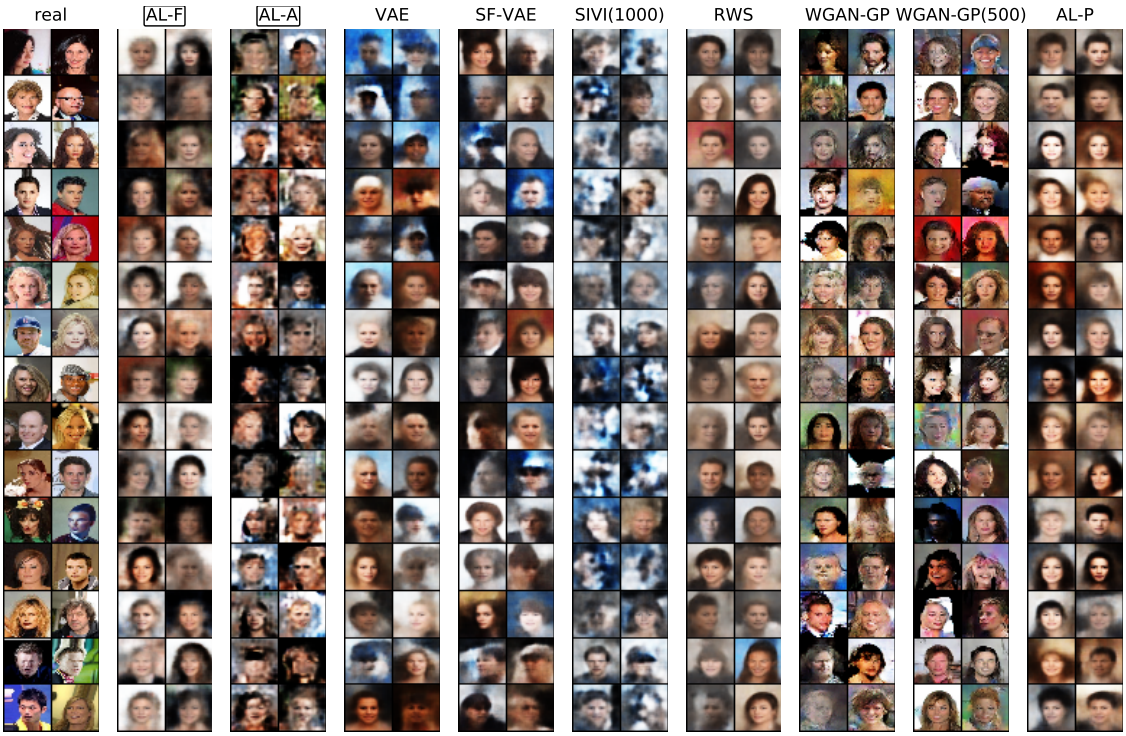

Figure 20. Samples for CelebA. Our main algorithms presented in the main text are highlighted in box. Each model is trained for 50 epochs, except otherwise indicated in parenthesis next to algorithm name.

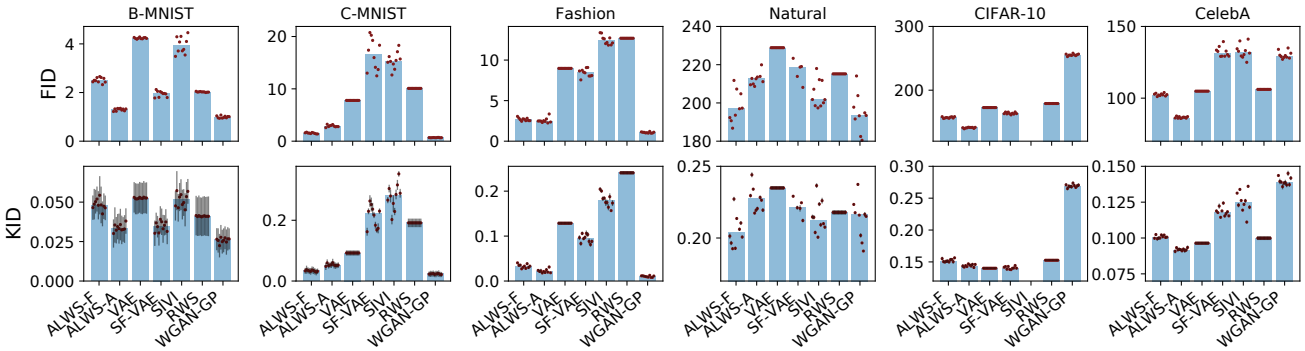

Figure 21. Same as Figure 21 but using fully connected networks. None of the SIVI runs on CIFAR-10 converge.

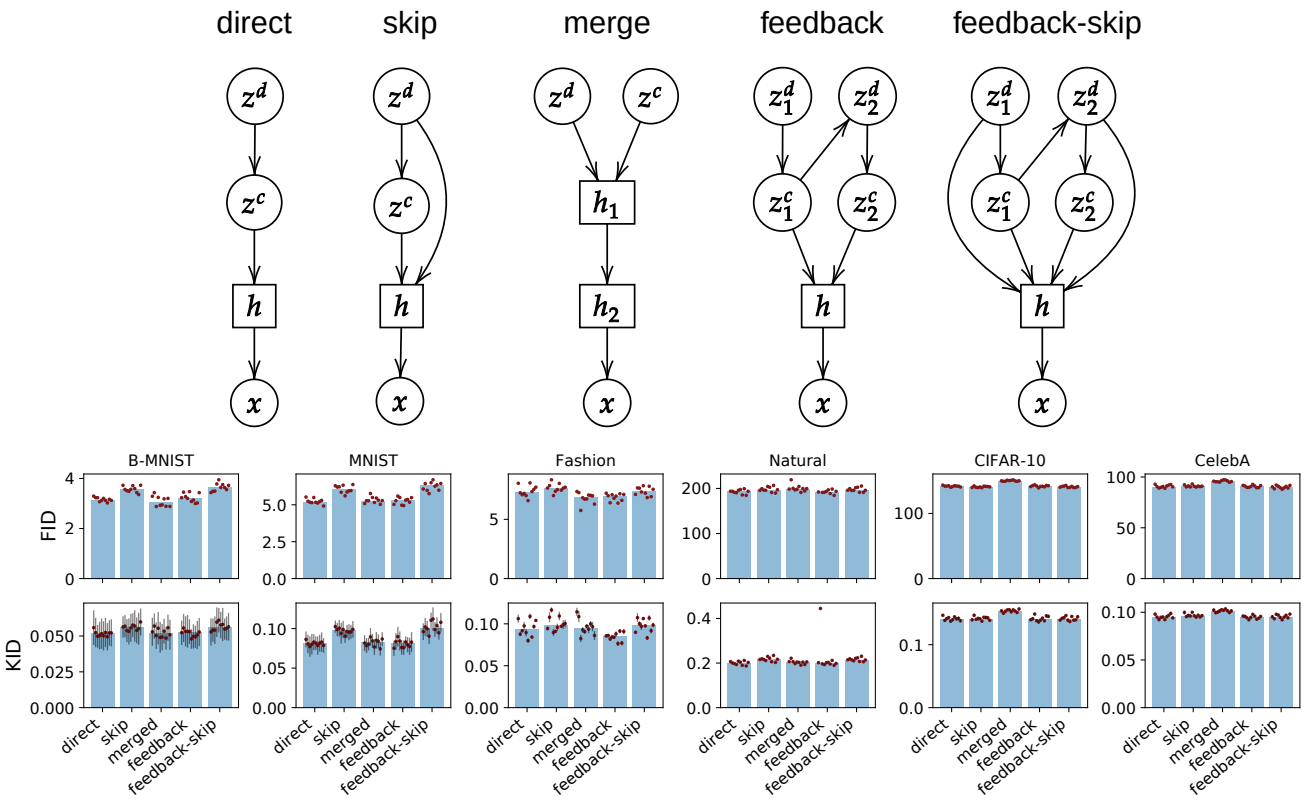

Figure 22. Top, the graphical representations of the generative models. Circles indicate random variables, with $\boldsymbol{z}^d$ as discrete Bernoulli and $\boldsymbol{z}^c$ as continuous Gaussian. Squares indicate deterministic nodes that are ReLU neurons activated by nodes with incoming arrows. The dimensionality of $\boldsymbol{z}^d$ is 10, $\boldsymbol{z}^c$ is 16, $\boldsymbol{z}_1^d$ and $\boldsymbol{z}_2^d$ are 5, and $\boldsymbol{z}_1^c$ and $\boldsymbol{z}_2^c$ are 8. The node h has 512 neurons. Bottom, FID and KID scores of generated images from architecturally complex models

C.7.3. Results on fully connected networks

We repeat the experiments for fully connected layers, with architecture $16 \to 512 \to 512 \to$ image dimension. The results are shown in Figure 21. According to FID, models trained by ALWS out-perform other ML methods on all datasets except Natural. KID agrees with FID except on CIFAR-10 where KID values are roughly the same for all ML methods.

C.7.4. RESULTS ON COMPLEX GENERATIVE NETWORKS

The goal here is to test how model architecture affects the quality of the generated samples. Discrete variables can be used to capture features such as object category, so including these in the generative model may be beneficial. In order to train models with discrete latent variables, explicit reparameterisation schemes have been developed in the past by continuous relaxation or overlapping transformation (Jang et al., 2017; Vahdat et al., 2018; Rolfe, 2017), and has shown differential performances. On the other hand, amortised learning is agnostic to the discrete or continuous nature of the latents.

We set out to explore different architectures while fixing the number of Bernoulli and Gaussian latent variables, respectively, and keep the number of parameters roughly the same. The different graphs are depicted in Figure 22 (top) and described in the legend. The **direct** model is a simple chain graph. The top Bernoulli layer connects to a Gaussian layer, where the mean is a function of the Bernoulli, and the variance is fixed at 1.0. The **skip** model is similar to the direct model, except that it adds an additional connection from the discrete latents to the hidden units in the network. The **merged** model combines the Bernoulli and Gaussian latents at the top layer, which goes through a first hidden h_1 layer of 16 units before feeding into the wide h_2 layer. The **feedback** model has an architecture inspired by (Vahdat et al., 2018). The latent $\boldsymbol{z}_1^c$ parametrises the logits for $\boldsymbol{z}_2^d$. The **feedback-skip** model is based on **feedback** and adds a skip connection to h from the top Bernoulli layer.

The results are shown in Figure 22 (bottom). Interestingly, we did not find any strong effect of model architecture on FID or KID. But the direct, merged and feedback architectures are clearly better than the other two for the two MNIST datasets according to FID.