A. Gradient of the entropy with respect to density functions

Consider a probability density function $p_g(x)$. We assume p_g is the pushforward of some prior distribution $p(z)$ by a mapping $g_\theta : z \mapsto x$. Our goal is to compute the gradient of the entropy of p_g wrt the parameter θ. Following Roeder et al. (2017), we show that the entropy gradient can be rewritten as Equation (1).

Proof. By the law of the unconscious statistician* (LOTUS, Theorem 1.6.9 of Durrett (2019)), we have

$$\begin{aligned}
\nabla_\theta H(p_g(x)) &= \nabla_\theta \mathop{\mathbb{E}}_{x\sim p_g(x)}[-\log p_g(x)] \\
&\overset{*}{=} \nabla_\theta \mathop{\mathbb{E}}_{z\sim p(z)}[-\log p_g(g_\theta(z))] \\
&= -\nabla_\theta \int p(z) \log p_g(g_\theta(z)) dz \\
&= -\cancel{\int p(z) \nabla_\theta \log p_g(x)|_{x=g_\theta(z)} dz} - \int p(z) [\nabla_x \log p_g(x)|_{x=g_\theta(z)}]^\intercal \mathbf{J}_\theta g_\theta(z) dz \\
&= -\mathop{\mathbb{E}}_{z\sim p(z)} \left[[\nabla_x \log p_g(x)|_{x=g_\theta(z)}]^\intercal \mathbf{J}_\theta g_\theta(z) \right].
\end{aligned}$$

where the crossed-out term is due to the following identity

$$\begin{aligned}
\mathop{\mathbb{E}}_{z\sim p(z)} \left[\nabla_\theta \log p_g(x)\big|_{x=g_\theta(z)} \right] = \mathop{\mathbb{E}}_{x\sim p_g(x)} [\nabla_\theta \log p_g(x)] &= \int p_g(x) \nabla_\theta \log p_g(x) dx \\
&= \int \cancel{p_g(x)} \frac{1}{\cancel{p_g(x)}} \nabla_\theta p_g(x) dx = \nabla_\theta \int p_g(x) dx = \nabla_\theta 1 = 0.
\end{aligned}$$

□

B. Properties of residual DAE

Proposition 1. *Let x and u be distributed by $p(x)$ and $\mathcal{N}(0, I)$. For $\sigma \neq 0$, the minimizer of the functional $\mathbb{E}_{x,u}[||u + \sigma f(x + \sigma u)||^2]$ is almost everywhere determined by*

$$f^*(x; \sigma) = \frac{-\mathbb{E}_u[p(x - \sigma u)u]}{\sigma \mathbb{E}_u[p(x - \sigma u)]}.$$

Furthermore, if $p(x)$ and its gradient are both bounded, f^* is continuous wrt σ for all $\sigma \in \mathbb{R} \setminus 0$ and $\lim_{\sigma\to 0} f^*(x; \sigma) = \nabla_x \log p_g(x)$.

Proof. For simplicy, when the absolute value and power are both applied to a vector-valued variable, they are applied elementwise. The characterization of the optimal function f^* can be derived by following Alain & Bengio (2014). For the second part, the symmetry of the distribution of u implies

$$\begin{aligned}
f^*(x; \sigma) &= \frac{-\mathbb{E}_u[p(x - \sigma u)u]}{\sigma \mathbb{E}_u[p(x - \sigma u)]} \\
&= \frac{\mathbb{E}_u[p(x + \sigma u)u]}{\sigma \mathbb{E}_u[p(x + \sigma u)]} = f^*(x; -\sigma),
\end{aligned}$$

so we only need to show f^* is continuous for $\sigma > 0$. Since p is bounded, by the dominated convergence theorem (DOM), both $\mathbb{E}_u[p(x - \sigma u)u]$ and $\mathbb{E}_u[p(x - \sigma u)]$ are continuous for $\sigma > 0$, and so is $f^*(x, \sigma)$.

Lastly, an application of L'Hôpital's rule gives

$$\lim_{\sigma\to 0} f^*(x; \sigma) = \lim_{\sigma\to 0} \frac{\frac{d}{d\sigma} \mathbb{E}_u[p(x + \sigma u)u]}{\frac{d}{d\sigma} \sigma \mathbb{E}_u[p(x + \sigma u)]},$$

which by another application of DOM (since gradient of p is bounded) is equal to

$$\lim_{\sigma\to 0}\frac{\mathbb{E}[\nabla p(x+\sigma u)^\top uu]}{\mathbb{E}[p(x+\sigma u)]+\sigma\,\mathbb{E}[\nabla p(x+\sigma u)^\top u]}.$$

Applying DOM a final time gives

$$\lim_{\sigma\to 0} f^*(x;\sigma)=\frac{\nabla p(x)\odot\mathbb{E}[u^2]}{p(x)}=\nabla\log p(x).$$

□

Proposition 2. $\lim_{\sigma\to\infty}\frac{f^*(x;\sigma)}{\nabla_x\log\mathcal{N}(x;\mathbb{E}_p[X],\sigma^2 I)}\to 1$.

Proof. We rewrite the optimal gradient approximator as

$$f^*(x;\sigma)=\frac{1}{\sigma^2}\int\frac{\mathcal{N}(u;0,I)p(x-\sigma u)}{\int\mathcal{N}(u';0,I)p(x-\sigma u')du'}\cdot\sigma u\,du.$$

Changing the variables $\epsilon=\sigma u$ and $\epsilon'=\sigma u'$ gives

$$\frac{1}{\sigma^2}\int\frac{\mathcal{N}(\epsilon/\sigma;0,I)p(x-\epsilon)}{\int\mathcal{N}(\epsilon'/\sigma;0,I)p(x-\epsilon')d\epsilon'}\cdot\epsilon\,d\epsilon,$$

which can be written as $\frac{1}{\sigma^2}\mathbb{E}_{q(\epsilon)}[\epsilon]$ where $q(\epsilon)\propto\mathcal{N}(\epsilon/\sigma;0,I)p(x-\epsilon)$ is the change-of-variable density.

By DOM (applied to the numerator and denominator separately, since the standard Gaussian density is bounded), $\mathbb{E}_q[\epsilon]\to\int p(x-\epsilon)\epsilon\,d\epsilon$ as $\sigma\to\infty$. The latter integral is equal to $\mathbb{E}_p[X]-x$ (which can be seen by substituting $y=x-\epsilon$). □

C. Signal-to-noise ratio analysis on DAE's gradient

Fixing x and u, the gradient of the L2 loss can be written as

$$\Delta:=\nabla||u+\sigma f(x+\sigma u)||^2=\nabla\left(\sum_i(u_i+\sigma f_i(x+\sigma u))^2\right)=\sum_i\nabla(u_i+\sigma f_i(x+\sigma u))^2,$$

where i iterates over the entries of the vectors u and f, and ∇ denotes the gradient wrt the parameters of f. We further expand the gradient of the summand via chain rule, which yields

$$\begin{aligned}\nabla(u_i+\sigma f_i(x+\sigma u))^2&=2\sigma(u_i+\sigma f_i(x+\sigma u))\nabla f_i(x+\sigma u)\\&=2\sigma\left(u_i\nabla\underbrace{f_i(x+\sigma u)}_{A}+\sigma\underbrace{f_i(x+\sigma u)}_{B}\nabla\underbrace{f_i(x+\sigma u)}_{C}\right).\end{aligned}$$

Taylor theorem with the mean-value form of the remainder allows us to approximate $f_i(x+\sigma u)$ by $f_i(x)$ as σ is small:

$$f_i(x+\sigma u)=f_i(x)+\sigma\nabla_x f_i(\hat{x})^\top u \tag{8}$$

$$=f_i(x)+\sigma\nabla_x f_i(x)^\top u+\frac{\sigma^2}{2}u^\top\nabla_x^2 f_i(\tilde{x})u, \tag{9}$$

where ∇_x denotes the gradient wrt the input of f, and $\hat{x}$ and $\tilde{x}$ are points lying on the line interval connecting x and $x+\sigma u$. Plugging (9) into A and (8) into B and C gives

$$\begin{aligned}&2\sigma\left(u_i\nabla\left(f_i(x)+\sigma\nabla_x f_i(x)^\top u+\frac{\sigma^2}{2}u^\top\nabla_x^2 f_i(\tilde{x})u\right)+\sigma\left(f_i(x)+\sigma\nabla_x f_i(\hat{x})^\top u\right)\nabla\left(f_i(x)+\sigma\nabla_x f_i(\hat{x})^\top u\right)\right)\\&=2\sigma u_i\nabla f_i(x)+2\sigma^2 u_i\nabla\nabla_x f_i(x)^\top u+\sigma^3 u_i\nabla u^\top\nabla_x^2 f_i(\tilde{x})u\\&\quad+2\sigma^2 f_i(x)\nabla f_i(x)+2\sigma^3 f_i(x)\nabla\nabla_x f_i(\hat{x})^\top u+2\sigma^3\nabla_x\left(f_i(\hat{x})^\top u\right)\nabla f_i(x)+2\sigma^4\nabla_x\left(f_i(\hat{x})^\top u\right)\nabla\nabla_x f_i(\hat{x})^\top u.\end{aligned}$$

With some regularity conditions (DOM-style assumptions), marginalizing out u and taking σ to be arbitrarily small yield

$$\begin{aligned}\mathbb{E}_u[\Delta] &= \sum_i 2\sigma^2 \nabla \frac{\partial}{\partial x_i} f_i(x) + 2\sigma^2 f_i(x) \nabla f_i(x) + o(\sigma^2) \\ &= 2\sigma^2 \nabla \left(\mathbf{tr}(\nabla_x f(x)) + \frac{1}{2} ||f(x)||^2 \right) + o(\sigma^2).\end{aligned}$$

In fact, we note that the first term is the stochastic gradient of the implicit score matching objective (Theorem 1, Hyvärinen (2005)), but it vanishes at a rate $\mathcal{O}(\sigma^2)$ as $\sigma^2 \to 0$.

For the second moment, similarly,

$$\mathbb{E}_u[\Delta\Delta^\top] = 4\sigma^2 \sum_i \nabla f_i(x) \nabla f_i(x)^\top + o(\sigma^2).$$

As a result,

$$\frac{\mathbb{E}[\Delta]}{\sqrt{\mathrm{Var}(\Delta)}} = \frac{\mathbb{E}[\Delta]}{\sqrt{\mathbb{E}(\Delta\Delta^\top) - \mathbb{E}(\Delta)\mathbb{E}(\Delta)^\top}} = \frac{\mathcal{O}(\sigma^2)}{\sqrt{\mathcal{O}(\sigma^2) - \mathcal{O}(\sigma^4)}} = \mathcal{O}(\sigma).$$

D. Experiment: Error analysis

D.1. Main experiments

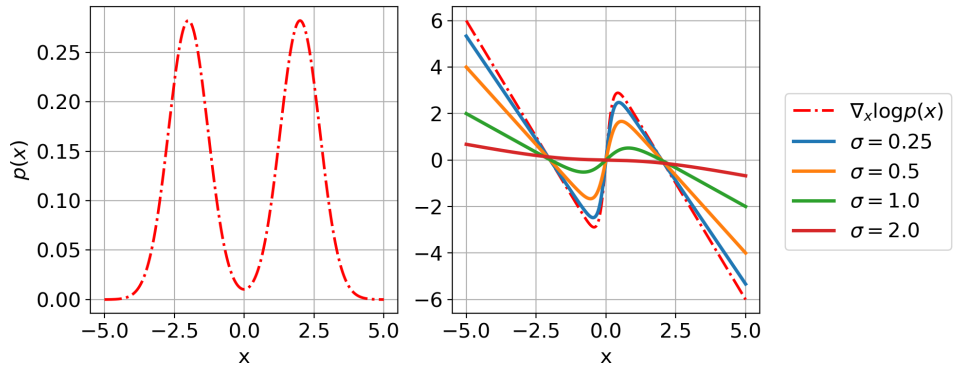

Figure S1. Left: Density function of mixture of Gaussians. Right: gradient of the log density function (dotdash line) and gradient approximations using optimal DAE with different σ values (solid lines).

Dataset and optimal gradient approximator As we have described in Section 4.2, we use the mixture of two Gaussians to analyze the approximation error (see Figure S1 (left)). Formally, we define $p(x) = 0.5\mathcal{N}(x; 2, 0.25) + 0.5\mathcal{N}(x; -2, 0.25)$. For notational convenience, we let p_1 and p_2 be the density functions of these two Gaussians, respectively. We obtain $\nabla_x \log p(x)$ by differentiating $\log p(x)$ wrt x using auto-differentiation library such as PyTorch (Paszke et al., 2017). With some elementary calculation, we can expand the formula of the optimal gradient approximator f^* as,

$$f^*(x;\sigma) = \frac{-\mathbb{E}_u[p(x-\sigma u)u]}{\sigma\, \mathbb{E}_u[p(x-\sigma u)]} = \frac{-\sum_{i=1}^2 S_i' \mu_i'}{\sigma \sum_{i=1}^2 S_i'},$$

where $S_i' = {}^1/\sqrt{2\pi(0.5^2+1^2)} \exp\left(-(\mu_i + x/\sigma)^2 / 2(0.5^2+1^2)\right)$ for $i \in 1, 2$, $\mu_1 = -2$, and $\mu_1 = 2$.

Proof. The numerator $\mathbb{E}_u[\mathbb{E}_u[p(x-\sigma u)u](x-\sigma u)u]$ can be rewritten as follows:

$$\mathop{\mathbb{E}}_u[p(x-\sigma u)u] = \int \left(0.5p_1(x-\sigma u) + 0.5p_2(x-\sigma u)\right) p(u)u\, du = \frac{0.5}{\sigma} \sum_{i=1}^2 S_i' \int \mathcal{N}(u; \mu_i', \sigma_i') u\, du = \frac{0.5}{\sigma} \sum_{i=1}^2 S_i' \mu_i',$$

where $S_i' = {}^1/\sqrt{2\pi(0.5^2+1^2)} \exp\left(-(\mu_i + x/\sigma)^2 / 2(0.5^2+1^2)\right)$ for $i \in 1, 2$, $\mu_1 = -2$, and $\mu_1 = 2$.

The second equality comes from the fact that all p_1, p_2, and $p(u)$ are normal distributions, and thus we have

$$p_i(x - \sigma u)p(u) = \frac{1}{\sigma} p_i\left(u - x/\sigma\right) p(u) = \frac{1}{\sigma} S_i' \mathcal{N}(u; \mu_i', \sigma_i').$$

Similarly, we can rewrite the denominator as $\mathbb{E}_u[p(x - \sigma u)] = \frac{0.5}{\sigma} \sum_{i=1}^{2} S_i'$. □

Experiments For AR-DAE, we indirectly parameterize it as the gradient of some scalar-function (which can be thought of as an unnormalized log-density function); *i.e.* we define a scalar function and use its gradient wrt the input vector. The same trick has also been employed in recent work by Saremi et al. (2018); Saremi & Hyvarinen (2019). We use the network architecture with the following configuration[4]: $[2 + 1, 256]$ + $[256, 256] \times 2$ + $[256, 1]$, with the `Softplus` activation function. We use the same network architecture for *resDAE* except it doesn't condition on σ. For *regDAE*, the network is set to reconstruct input.

All models are trained for 10k iterations with a minibatch size of 256. We use the Adam optimizer for both AR-DAE and the generator, with the default $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For all models, the learning rate is initially set to 0.001 and is reduced by half every 1k iterations during training.

For *regDAE* and *resDAE*, we train models individually for every σ value in Figure 3. For *regDAE*$_{annealed}$ and *resDAE*$_{annealed}$, we anneal σ from 1 to the target value.For AR-DAE, δ is set to 0.05 and we sample 10 σ's from $N(0, \delta^2)$ for each iteration. We train all models five times and present the mean and its standard error in the figures.

D.2. Symmetrizing the distribution of σ

In Section 4.1, we argue that neural networks are not suitable for extrapolation (vs. interpolation), to motivate the use of a symmetric prior over σ. To contrast the difference, we sample $\sigma \sim N(0, \delta^2)$ and compare two different types of σ-conditioning: (1) conditioning on σ, and (2) conditioning on $|\sigma|$. We use the same experiment settings in the previous section, but we use a hypernetwork (Ha et al., 2017) that takes σ (resp. $|\sigma|$) as input and outputs the parameters of AR-DAE, to force AR-DAE to be more dependent on the value of σ (resp. $|\sigma|$). The results are shown in Figure S2.

We see that the two conditioning methods result in two distinct approximation behaviors. First, when AR-DAE only observes positive values, it fails to extrapolate to the σ values close to 0. When a symmetric σ distribution is used, the approximation error of AR-DAE is more smooth. Second, we notice that the symmetric σ distribution bias f_{ar} to focus more on small σ values. Finally, the asymmetric distribution helps AR-DAE reduce the approximation error for some σ. We speculate that AR-DAE with the asymmetric σ distribution has two times higher to observe small σ-values during training, and thus improves the approximation. In general, we observe that the stability of the approximation is important for our applications, in which case AR-DAE need to adapt constantly in the face of non-stationary distributions.

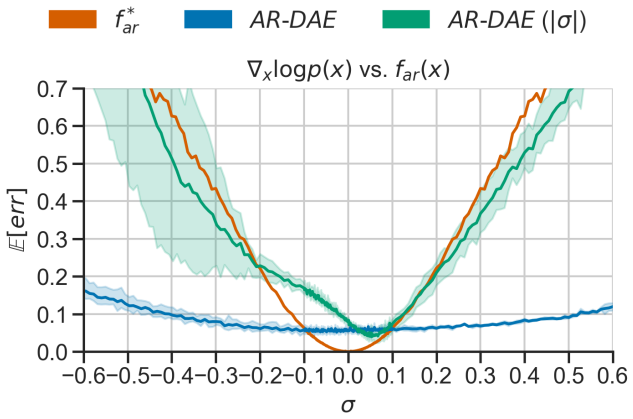

Figure S2. Comparison of two σ-conditioning methods to approximate log density gradient of 1D-MOG. AR-DAE: conditioning on σ. AR-DAE ($|\sigma|$): conditioning on $|\sigma|$. σ is sampled from $N(0, \delta)$ for all experiments.

[4] $[d_{\texttt{input}}, d_{\texttt{output}}]$ denotes a fully-connected layer whose input and output feature sizes are $d_{\texttt{input}}$ and $d_{\texttt{output}}$, respectively.

E. Experiment: Energy Fitting

Potential $U(\mathbf{z})$
1: $\frac{1}{2}\left(\frac{\|\mathbf{z}\|-2}{0.4}\right)^2 - \ln\left(e^{-\frac{1}{2}\left[\frac{z_1-2}{0.6}\right]^2} + e^{-\frac{1}{2}\left[\frac{z_1+2}{0.6}\right]^2}\right)$
2: $\frac{1}{2}\left(\frac{z_2-w_1(\mathbf{z})}{0.4}\right)^2$
3: $-\ln\left(e^{-\frac{1}{2}\left[\frac{z_2-w_1(\mathbf{z})}{0.35}\right]^2} + e^{-\frac{1}{2}\left[\frac{z_2-w_1(\mathbf{z})+w_2(\mathbf{z})}{0.35}\right]^2}\right)$
4: $-\ln\left(e^{-\frac{1}{2}\left[\frac{z_2-w_1(\mathbf{z})}{0.4}\right]^2} + e^{-\frac{1}{2}\left[\frac{z_2-w_1(\mathbf{z})+w_3(\mathbf{z})}{0.35}\right]^2}\right)$
where $w_1(\mathbf{z}) = \sin\left(\frac{2\pi z_1}{4}\right)$, $w_2(\mathbf{z}) = 3e^{-\frac{1}{2}\left[\frac{z_1-1}{0.6}\right]^2}$, $w_3(\mathbf{z}) = 3\sigma\left(\frac{z_1-1}{0.3}\right)$, $\sigma(x) = \frac{1}{1+e^{-x}}$.

Table 4. The target energy functions introduced in Rezende & Mohamed (2015).

E.1. Main experiments

Parametric densities trained by minimizing the reverse KL divergence tend to avoid "false positive", a well known problem known as the zero-forcing property (Minka et al., 2005). To deal with this issue, we minimize a modified objective:

$$D_{KL\alpha}(p_g(x)||p_{\texttt{target}}(x)) = -H(p_g(x)) - \alpha \mathop{\mathbb{E}}_{x\sim p_g(x)}\left[\log p_{\texttt{target}}(x)\right], \tag{10}$$

where α is annealed from a small value to 1.0 throughout training. This slight modification of the objective function "convexifies" the loss landscape and makes it easier for the parametric densities to search for the lower energy regions. For AR-DAE training, we use Equation (2) with a fixed prior variance $\delta = 0.1$.

For all experiments, we use a three-hidden-layer MLP for both hierarchical distribution as well as implicit distribution. More specifically, the generator network for the hierarchical distribution has the following configuration: $[d_z, 256]$ + $[256, 256] \times 2$ + $[256, 2] \times 2$. d_z indicates the dimension of the prior distribution $p(z)$ and is set to 2. The last two layers are for mean and log-variance[5] of the conditional distribution $p_g(x|z)$. For the auxiliary variational method, the same network architecture is used for $h(z|x)$ in Equation (5). When we train the hierarchical distribution with AR-DAE, we additionally clamp the log-variance to be higher than -4. Similar to the hierarchical distribution, the generator of the implicit distribution is defined as, $[d_z, 256]$ + $[256, 256] \times 2$ + $[256, 2]$. Unlike the hierarchical distribution, d_z is set to 10. `ReLU` activation function is used for all but the final output layer.

For AR-DAE, we directly parameterize the residual function f_{ar}. We use the following network architecture: $[2, 256]$ + $[256, 256] \times 2$ + $[256, 2]$. `Softplus` activation function is used.

Each model is trained for 100,000 iterations with a minibatch size of 1024. We update AR-DAE N_d times per generator update. For the main results, we set $N_d = 5$. We use the Adam optimizer for both the generator and AR-DAE, where $\beta_1 = 0.5$ and $\beta_2 = 0.999$. The learning rate for the generator is initially set to 0.001 and is reduced by 0.5 for every 5000 iterations during training. AR-DAE's learning rate is set to 0.001. To generate the figure, we draw 1M samples from each model to fill up 256 equal-width bins of the 2D histogram.

E.2. Effect of the number of updates (N_d) of the gradient approximator

In addition to the main results, we also analyze how the number of updates of AR-DAE per generator update affects the quality of the generator. We use the same implicit generator and AR-DAE described in the main paper, but vary N_d from 1 to 5. The result is illustrated in Figure S3. In principle, the more often we update AR-DAE, the more accurate (or up-to-date) the gradient approximation will be. This is corroborated by the improved quality of the trained generator.

[5] diagonal elements of the covariance matrix in log-scale

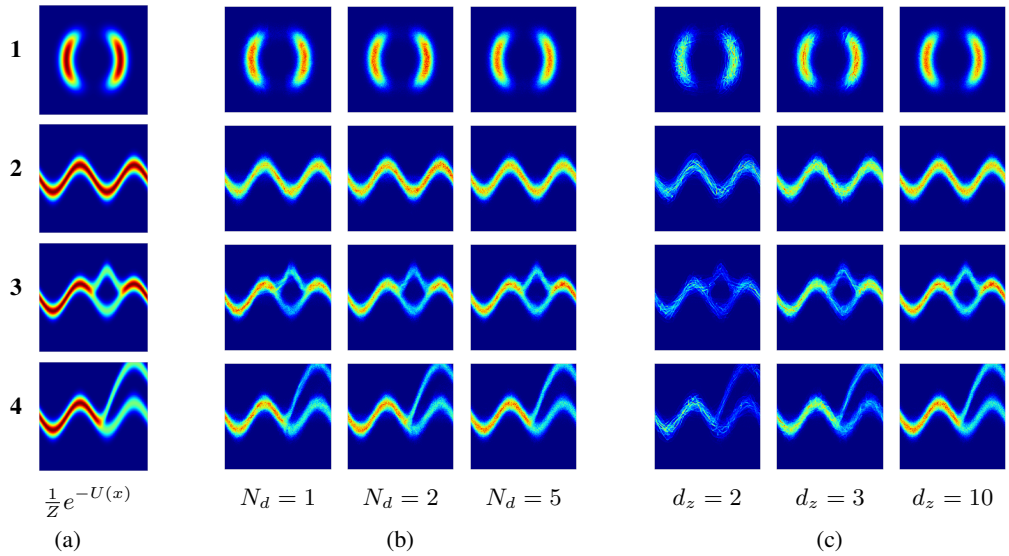

Figure S3. Fitting energy functions with implicit model using AR-DAE. (a) Target energy functions. (b) Varying number of AR-DAE updates per model update. (c) Varying the dimensionality of the noise source d_z.

E.3. Effect of the noise dimension of implicit model

In this section, we study the effect of varying the dimensionality of the noise source of the implicit distribution. We use the same experiment settings in the previous section. In Figure S3 (right panel), we see that the generator has a degenerate distribution when $d_z = 2$, and the degeneracy can be remedied by increasing d_z.

F. Experiment: variational autoencoders

F.1. VAE with the entropy gradient approximator

Let $p_\omega(x|z)$ be the conditional likelihood function parameterized by ω and $p(z)$ be the the prior distribution. We let $p(z)$ be the standard normal. As described in Section 6.2, we would like to maximize the ELBO (denoted as $\mathcal{L}_{ELBO}$) by jointly training p_ω and the amortized variational posterior $q_\phi(z|x)$. Similar to Appendix A, the posterior $q_\phi(z|x)$ can be induced by a mapping $g_\phi : \epsilon, x \mapsto z$ with a prior $q(\epsilon)$ that does not depend on the parameter ϕ. The gradient of $\mathcal{L}_{ELBO}$ wrt the parameters of the posterior can be written as,

$$\nabla_\phi \mathcal{L}_{ELBO}(q) = \mathop{\mathbb{E}}_{\substack{z\sim q_\phi(z|x) \\ x\sim p_{\text{data}}(x)}} \left[[\nabla_z \log p_\omega(x,z) - \nabla_z \log q_\phi(z|x)]^\intercal \, \mathbf{J}_\phi g_\phi(\epsilon, x) \right]. \tag{11}$$

We plug in AR-DAE to approximate the gradient of the log-density, and draw a Monte-Carlo sample of the following quantity to estimate the gradient of the ELBO

$$\hat{\nabla}_\phi \mathcal{L}_{ELBO}(q) \doteq \mathop{\mathbb{E}}_{\substack{z\sim q_\phi(z|x) \\ x\sim p_{\text{data}}(x)}} \left[[\nabla_z \log p_\omega(x,z) - f_{ar,\theta}(z;x,\sigma)|_{\sigma=0}]^\intercal \, \mathbf{J}_\phi g_\phi(\epsilon, x) \right], \tag{12}$$

F.2. AR-DAE

To approximate $\nabla_z \log q_\phi(z|x)$, we condition AR-DAE on both the input x as well as the noise scale σ. We also adaptively choose the prior variance δ^2 for different data points instead of fixing it to be a single value.

In addition, we make the following observations. (1) The posteriors q_ϕ are usually not centered, but the entropy gradient approximator only needs to model the dispersion of the distribution. (2) The variance of the approximate posterior can be very small during training, which might pose a challenge for optimization. To remedy these, we modify the input of AR-DAE to be $\tilde{z} \doteq s(z - b(x))$, where s is a scaling factor and $b(x)$ is a pseudo mean. Ideally, we would like to set $b(x)$ to be $\mathbb{E}_{q(z|x)}[z]$. Instead, we let $b(x) \doteq g(0, x)$, as 0 is the mode/mean of the noise source. The induced distribution of $\tilde{z}$ will

be denoted by $q_\phi(\tilde{z}|x)$. By the change-of-variable density formula, we have $\nabla_z \log q(z|x) = s\nabla_{\tilde{z}} \log q(\tilde{z}|x)$. This allows us to train AR-DAE with a better-conditioned distribution and the original gradient can be recovered by rescaling.

In summary, we optimize the following objective

$$\mathcal{L}_{\texttt{ar}}\left(f_{ar}\right) = \mathop{\mathbb{E}}_{\substack{x\sim p(x)\\ \tilde{z}\sim q(\tilde{z}|x)\\ u\sim N(0,I)\\ \sigma|x\sim N(0,\delta(x)^2)}} \left[\left\|u + \sigma f_{ar}(\tilde{z}+\sigma u; x, \sigma)\right\|^2\right]. \tag{13}$$

where $\delta(x) \doteq \delta_{\texttt{scale}} S_{z|x}$ and $S_{z|x}$ is sample standard deviation of z given x. We use n_z samples per data to estimate $S_{z|x}$. $\delta_{\texttt{scale}}$ is chosen as hyperparameter.

In the experiments, we either directly parameterize the residual function of AR-DAE or indirectly parameterize it as the gradient of some scalar-function. We parameterize $f_{ar}(\tilde{z}; x, \sigma)$ as a multi-layer perceptron (MLP). Latent z and input x are encoded separately and then concatenated with σ (denoted by "mlp-concat" in Table 8). The MLP encoders have $m_{\texttt{enc}}$ hidden layers. The concatenated representation is fed into a fully-connected neural network with $m_{\texttt{fc}}$ hidden layers. Instead of encoding the input x directly, we either use a hidden representation of the variational posterior q or $b(x)$. We use d_h hidden units for all MLPs. We stress that the learning signal from $\mathcal{L}_{\texttt{ar}}\left(f_{ar}\right)$ is not backpropagated to the posterior.

Algorithm 1 VAE AR-DAE

Input: Dataset $\mathcal{D}$; mini-batch size $n_{\texttt{data}}$; sample size n_z;prior variance δ^2; learning rates α_θ and $\alpha_{\phi,\omega}$
Initialize encoder and decoder $p_\omega(x|z)$ and $q_\phi(z|x)$
Initialize AR-DAE $f_{ar,\theta}(z|x)$
repeat
  Draw $n_{\texttt{data}}$ datapoints from $\mathcal{D}$
  for $k = 0 \dots N_d$ **do**
    Draw n_z latents per datapoint from $z \sim q_\phi(z|x)$
    $\delta_i \leftarrow \delta_{\texttt{scale}} S_{z|x_i}$ for $i = 1, \dots, n_{\texttt{data}}$
    Draw n_σ number of σ_is per z from $\sigma_i \sim N(0, \delta_i^2)$
    Draw $n_{\texttt{data}} n_z n_\sigma$ number of us from $u \sim N(0, I)$
    Update θ using gradient $\nabla_\theta \mathcal{L}_{f_{ar}}$ with learning rate α_θ
  end for
  $z \sim q_\phi(z|x)$
  Update ω using gradient $\nabla_\omega \mathcal{L}_{ELBO}$ with learning rate $\alpha_{\phi,\omega}$
  Update ϕ using gradient $\hat{\nabla}_\phi \mathcal{L}_{ELBO}$ with learning rate $\alpha_{\phi,\omega}$, whose entropy gradient is approximated using $f_{ar,\theta}(z|x)$.
until Until some stopping criteria

F.3. Experiments

We summarize the architecture details and hyperparameters in Table 7 and 8, respectively.

Mixture of Gaussian experiment For the MoG experiment, we use 25 Gaussians centered on an evenly spaced 5 by 5 grid on $[-4, 4] \times [-4, 4]$ with a variance of 0.1. Each model is trained for 16 epochs: approximately 4000 updates with a minibatch size of 512.

For all experiments, we use a two-hidden-layer MLP to parameterize the conditional diagonal Gaussian $p(x|z)$. For the implicit posterior q, the input x and the d_ϵ-dimensional noise are separately encoded with one fully-connected layer, and then the concatenation of their features will be fed into a two-hidden-layer MLP to generate the 2-dimensional latent z. The size of the noise source ϵ in the implicit posterior, *i.e.* d_ϵ, is set to 10.

MNIST We first describe the details of the network architectures and then continue to explain training settings. For the *MLP* experiments, we use a one-hidden-layer MLP for the diagonal Gaussian decoder $p(x|z)$. For the diagonal Gaussian posterior $q(z|x)$. aka vanilla VAE, input x is fed into a fully-connected layer and then the feature is later used to predict the mean and diagonal component of the covariance matrix of the multivariate Gaussian distribution. For the hierarchical

posterior, both $q(z_0|x)$ and $q(z|z_0, x)$ are one-hidden-layer MLPs with diagonal Gaussian similar to the vanilla VAE. For the implicit posterior, the input is first encoded and then concatenated with noise before being fed into another MLP to generate z.

For *Conv*, the decoder starts with a one-fully connected layer followed by three deconvolutional layers. The encoder has three convolutional layers and is modified depending on the types of the variational posteriors, similar to *MLP*. For *ResConv*, five convolutional or deconvolutional layers with residual connection are used for the encoder and the decoder respectively.

Following Maaløe et al. (2016); Ranganath et al. (2016), when the auxiliary variational method (HVI aux) is used to train the hierarchical posterior, the variational lower bound is defined as, we maximize the following lower bound to train the hierarchical variational posterior with auxiliary variable (HVI aux)

$$\log p(x) \geq \mathop{\mathbb{E}}_{z\sim q(z|x)} \left[\log p(x,z) - \log q(z|x)\right] \geq \mathop{\mathbb{E}}_{\substack{z_0\sim q(z_0|x)\\ z\sim q(z|z_0,x)}} \left[\log p(x,z) - \log q(z_0|x) - \log q(z|z_0,x) + \log h(z_0|z,x)\right].$$

For the dynamically binarized MNIST dataset, we adopt the experiment settings of Mescheder et al. (2017). The MNIST data consists of 50k train, 10k validation, and 10k test images. In addition to the original training images, randomly selected 5k validation images are added to the training set. Early stopping is performed based on the evaluation on the remaining 5k validation data points. The maximum number of iterations for the training is set to 4M.

For the statically binarized MNIST dataset, we use the original data split. Early stopping as well as hyperparameter search are performed based on the estimated log marginal probability on the validation set. We retrain the model with the selected hyperparameters with the same number of updates on the combination of the train+valid sets, and report the test set likelihood. We also apply polyak averaging (Polyak & Juditsky, 1992).

We evaluate $\log p(x)$ of the learned models using importance sampling (Burda et al., 2016) (with n_{eval} samples). For the baseline methods, we use the learned posteriors as proposal distributions to estimate the log probability. When a posterior is trained with AR-DAE, we first draw n_{eval} z's from the posterior given the input x, and then use the sample mean and covariance matrix to construct a multivariate Gaussian distribution. We then use this Gaussian distribution as the proposal.

G. Experiment: entropy-regularized reinforcement learning

G.1. Soft actor-critic

Notation We consider an infinite-horizon Markov decision process (MDP) defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p_{\text{env}}, \gamma)$ (Sutton et al., 1998), where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$ are the spaces of state, action and reward, respectively, $p_{\text{env}}(s_{t+1}|s_t, a_t)$ and $p_{\text{env}}(s_0)$ represent the transition probability and the initial state distribution, $r(s_t, a_t)$ is a bounded reward function, and γ is a discount factor. We write τ as a trajectory resulting from interacting with the environment under some policy $\pi(a_t|s_t)$.

The entropy-regularized reinforcement learning (Ziebart, 2010) is to learn a policy $\pi(a_t|s_t)$ that maximizes the following objective;

$$\mathcal{L}(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi, p_{\text{env}}} \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))\right)\right], \tag{14}$$

where α is an entropy regularization coefficient. We define a soft state value function V^π and s soft Q-function Q^π as follows,

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi, p_{\text{env}}} \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))\right) \middle| s_0 = s\right]$$

$$Q^\pi(s, a) = \mathop{\mathbb{E}}_{\tau\sim\pi, p_{\text{env}}} \left[r(s_t, a_t) + \sum_{t=1}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))\right) \middle| s_0 = s, a_0 = a\right].$$

By using these definitions, we can rewrite V^π and Q^π as $V^\pi(s) = \mathbb{E}_{a\sim\pi}\left[Q^\pi(s, a)\right] + \alpha H(\pi(\cdot|s))$ and $Q^\pi(s) = [r(s, a) + \mathbb{E}_{s'\sim p_{\text{env}}} \gamma V^\pi(s')]$.

Soft actor-critic One way to maximize (14) is to minimize the following KL divergence,

$$\pi_{\texttt{new}} = \arg\min_{\pi} D_{KL}\left(\pi(\cdot|s_t) \middle\| \frac{\exp\left(Q^{\pi_{\texttt{old}}}(s_t,\cdot)\right)}{Z^{\pi_{\texttt{old}}}(s_t)}\right),$$

where $Z^{\pi_{\texttt{old}}}(s_t)$ is the normalizing constant $\int \exp\left(Q^{\pi_{\texttt{old}}}(s_t,a)\right) da$. Haarnoja et al. (2018) show that for finite state space the entropy-regularized expected return will be non-decreasing if the policy is updated by the above update rule. In practice, however, we do not have access to the value functions, so Haarnoja et al. (2018) propose to update the policy by first approximating $Q^{\pi_{\texttt{old}}}$ and $V^{\pi_{\texttt{old}}}$ by some parametric functions Q_ω and V_ν, and training the policy by minimizing

$$\mathcal{L}(\pi) = \mathop{\mathbb{E}}_{s_t\sim\mathcal{D}}\left[D_{KL}\left(\pi(a_t|s_t) \middle\| \frac{\exp\left(Q_\omega(s_t,\cdot)\right)}{Z_\omega(s_t)}\right)\right],$$

where $\mathcal{D}$ is a replay buffer that stores all the past experience. The soft Q-function and soft state value function will be trained by minimizing the following objectives,

$$\mathcal{L}(V_\nu) = \mathop{\mathbb{E}}_{s_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\nu(s_t) - \mathop{\mathbb{E}}_{a_t\sim\pi}\left[Q_\omega(s_t,a_t) - \alpha\log\pi(a_t|s_t)\right]\right)^2\right]$$

$$\mathcal{L}(Q_\omega) = \mathop{\mathbb{E}}_{s_t,a_t\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\omega(s_t,a_t) - \hat{Q}(s_t,a_t)\right)^2\right],$$

where $\hat{Q}(s_t,a_t) \doteq r(s_t,a_t) + \gamma\,\mathbb{E}_{s_{t+1}\sim p_{\texttt{env}}}[V_{\bar{\nu}}(s_{t+1})]$ and $V_{\bar{\nu}}$ is a target value network. For the target value network, SAC follows Mnih et al. (2015): $V_{\bar{\nu}}$ is defined as a polyak-averaged model (Polyak & Juditsky, 1992) of V_ν. Note that V_ν is inferred from Q_ω via Monte Carlo, *i.e.* $V_\nu(s_t) \doteq Q_\omega(s_t,a_t) - \alpha\log\pi(a_t|s_t)$ where $a_t\sim\pi(a_t|s_t)$. Moreover, we follow the common practice to use the clipped double Q-functions (Hasselt, 2010; Fujimoto et al., 2018) in our implementations.

G.2. SAC-AR-DAE and its implementations

Main algorithm Our goal is to train an arbitrarily parameterized policy within the SAC framework. We apply AR-DAE to approximate the training signal for policy. Similar to the implicit posterior distributions in the VAE experiments, the policy consists of a simple tractable noise distribution $\pi(\epsilon)$ and a mapping $g_\phi : \epsilon, s \mapsto a$. The gradient of $\mathcal{L}(\pi)$ wrt the policy parameters can be written as

$$\nabla_\phi\mathcal{L}(\pi) = \mathop{\mathbb{E}}_{\substack{s_t\sim\mathcal{D}\\ \epsilon\sim\pi}}\left[\left[\nabla_a\log\pi_\phi(a|s_t)|_{a=g_\phi(\epsilon,s_t)} - \nabla_a Q_\omega(s_t,a)|_{a=g_\phi(\epsilon,s_t)}\right]^\intercal \mathbf{J}_\phi g_\phi(\epsilon,s_t)\right].$$

Let $f_{ar,\theta}$ be AR-DAE which approximates $\nabla_a\log\pi_\phi(a|s)$ trained using Equation (13). Specifically for the SAC experiment, AR-DAE is indirectly parameterized as the gradient of an unnormalized log-density function $\psi_{ar,\theta} : a, s, \sigma \mapsto \mathbb{R}$ as in,

$$f_{ar,\theta}(a;s,\sigma) \doteq \nabla_a\psi_{ar,\theta}(a;s,\sigma).$$

As a result, $\log\pi(a|s)$ can also be approximated by using $\psi_{ar,\theta}$: $\log\pi(a|s) \approx \psi_{ar,\theta}(a;s,\sigma)|_{\sigma=0} - \log Z_\theta(s)$, where $Z_\theta(s) = \int \exp\left(\psi_{ar,\theta}(a;s,\sigma)|_{\sigma=0}\right) da$.

Using AR-DAE, we can modify the objective function $\mathcal{L}(V_\nu)$ to be

$$\hat{\mathcal{L}}(V_\nu) = \mathop{\mathbb{E}}_{s_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\nu(s_t) - \mathop{\mathbb{E}}_{a_t\sim\pi}\left[Q_\omega(s_t,a_t) - \psi_{ar,\theta}(a_t;s_t,\sigma)|_{\sigma=0}\right] - \log Z_\theta(s_t)\right)^2\right].$$

The same applies to $\mathcal{L}(Q_\omega)$. We also use the polyak-averaged target value network and one-sample Monte-Carlo estimate as done in SAC. Finally, the gradient signal for the policy can be approximated using AR-DAE:

$$\hat{\nabla}_\phi\mathcal{L}(\pi) \doteq \mathop{\mathbb{E}}_{\substack{s_t\sim\mathcal{D}\\ \epsilon\sim\pi}}\left[\left[f_{ar,\theta}(g_\phi(\epsilon,s_t);s_t,\sigma)|_{\sigma=0} - \nabla_a Q_\omega(s_t,a)|_{a=g_\phi(\epsilon,s_t)}\right]^\intercal \mathbf{J}_\phi g_\phi(\epsilon,s_t)\right].$$

We summarize all the details in Algorithm 2.

Algorithm 2 SAC-AR-DAE

Input: Mini-batch size $n_{\texttt{data}}$; replay buffer $\mathcal{D}$; number of epoch T; learning rates $\alpha_\theta, \alpha_\phi, \alpha_\omega, \alpha_\nu$
Initialize value function $V_\nu(s)$, critic $Q_\omega(s,a)$, policy $\pi_\phi(a|s)$, and AR-DAE $f_{ar,\theta}(a|s)$
Initialize replay buffer $\mathcal{D} \leftarrow \varnothing$
for epoch $= 1, ..., T$ **do**
  Initialize a state from $s_0 \sim p_{\texttt{env}}(s_0)$
  for $t = 0 \dots$ **do**
    $a \sim \pi_\phi(.|s_t)$
    $(r_t, s_{t+1}) \sim p_{\texttt{env}}(\cdot|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
    for each learning step **do**
      Draw $n_{\texttt{data}}$ number of (s_t, a_t, r_t, s_{t+1})s from $\mathcal{D}$
      for $k = 0 \dots N_d$ **do**
        Draw n_a actions per state from $a \sim \pi_\phi(a|s)$
        $\delta_i \leftarrow \delta_{\texttt{scale}} S_{a|s_i}$ for $i = 1, \dots, n_{\texttt{data}}$
        Draw n_σ number of σ_is per a from $\sigma_i \sim N(0, \delta_i^2)$
        Draw $n_{\texttt{data}} n_a n_\sigma$ number of us from $u \sim N(0, I)$
        Update θ using gradient $\nabla_\theta \mathcal{L}_{f_{ar}}$ with learning rate α_θ
      end for
      Update ν using gradient $\nabla_\nu \hat{\mathcal{L}}_V$ with learning rate α_ν
      Update ω using gradient $\nabla_\omega \hat{\mathcal{L}}_Q$ with learning rate α_ω
      Update ϕ using gradient $\hat{\nabla}_\phi \mathcal{L}_\pi$ which is approximated with $f_{ar,\theta}(a|s)$
      $\bar{\nu} \leftarrow \tau\nu + (1-\tau)\bar{\nu}$
    end for
  end for
end for

Bounded action space The action space of all of our environments is an open cube $(-1,1)^{d_a}$, where d_a is the dimensionality of the action. To implement the policy, we apply the hyperbolic tangent function. That is, $a := \tanh(g_\phi(\epsilon, s_t))$, where the output of g_ϕ (denoted as $\tilde{a}$) is in $(-\infty, \infty)$. Let $\tilde{a}_i$ be the i-th element of $\tilde{a}$. By the change of variable formula, $\log \pi(a|s) = \log \pi(\tilde{a}|s) - \sum_{i=1}^{d_a} \log(1 - \tanh^2(\tilde{a}_i))$.

In our experiments, we train AR-DAE on the pre-$\tanh$ action $\tilde{a}$. This implies that AR-DAE approximate $\nabla_{\tilde{a}} \log \pi(\tilde{a}|s)$. We correct the change of volume induced by the tanh using

$$\nabla_{\tilde{a}} \log \pi(a|s) = \nabla_{\tilde{a}} \log \pi(\tilde{a}|s) + 2 \tanh(\tilde{a}).$$

To sum up, the update of the policy follows the approximated gradient

$$\hat{\nabla}_\phi \mathcal{L}(\pi) \doteq \mathop{\mathbb{E}}_{\substack{s_t \sim \mathcal{D} \\ \epsilon \sim \pi}} \left[\left[f_{ar,\theta}(g_\phi(\epsilon, s_t); s_t, \sigma)|_{\sigma=0} + 2\tanh(g_\phi(\epsilon, s_t)) - \nabla_{\tilde{a}} Q_\omega(s_t, \tanh(\tilde{a}))|_{\tilde{a}=g_\phi(\epsilon,s_t)} \right]^\intercal \mathbf{J}_\phi g_\phi(\epsilon, s_t) \right].$$

Estimating normalizing constant In order to train SAC-AR-DAE in practice, efficient computation of $\log Z_\theta(s)$ is required. We propose to estimate the normalizing constant (Geyer, 1991) using importance sampling. Let $h(a|s)$ be the proposal distribution. We compute the following (using the log-sum-exp trick to ensure numerical stability)

$$\begin{aligned}
\log Z_\theta(s) &= \log \int \exp\left(\psi_{ar,\theta}(a; s, \sigma)|_{\sigma=0}\right) da \\
&= \log \mathop{\mathbb{E}}_{a \sim h} \left[\exp\left(\psi_{ar,\theta}(a; s, \sigma)|_{\sigma=0} - \log h(a|s)\right) \right] \\
&\approx \log \frac{1}{N_Z} \sum_j^{N_Z} \left[\exp\left(\psi_{ar,\theta}(a_j; s, \sigma)|_{\sigma=0} - \log h(a_j|s) - A\right) \right] + A,
\end{aligned}$$

where a_j is the j-th action sample from h and $A := \max_{a_j} \exp\left(\psi_{ar,\theta}(a_j; s, \sigma)|_{\sigma=0} - \log h(a_j|s)\right)$. For the proposal distribution, we use $h(a|s) \doteq N(\mu(s), cI)$, where $\mu(s) \doteq \psi_{ar,\theta}(g_\phi(\epsilon, s); s, \sigma)|_{\epsilon=0,\sigma=0}$ and c is some constant. We set c to be $\log c = -1$.

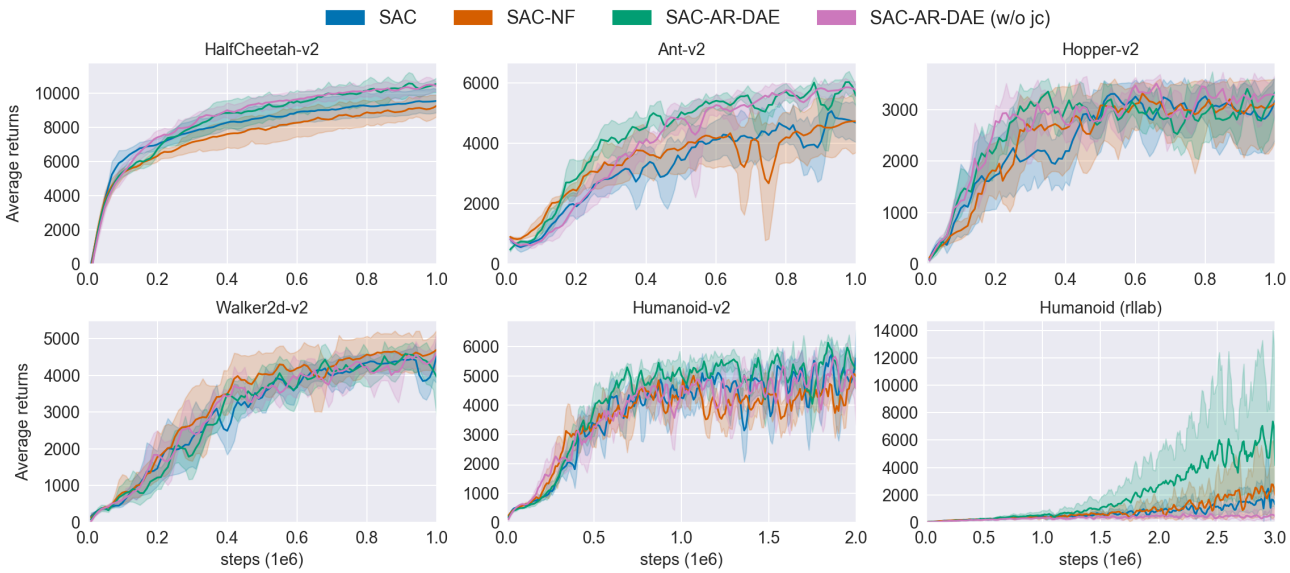

Figure S4. Additional results on SAC-AR-DAE, ablating Jacobian clamping regularization on implicit policy distributions in comparison with the rest.

Target value calibration In order to train the Q-function more efficiently, we calibrate its target values. Training the policy only requires estimating the gradient of the Q-function wrt the action, not the value of the Q-function itself. This means that while optimizing Q_ω (and V_ν), we can subtract some constant from the true target to center it. In our experiment, this calibration is applied when we use one-sample Monte-Carlo estimate and the polyak-averaged Q-network $Q_{\bar{\omega}}$. That is, $\mathcal{L}(Q_\omega)$ can be rewritten as,

$$\mathcal{L}(Q_\omega) = \mathop{\mathbb{E}}_{\substack{s_t, a_t, s_{t+1} \sim \mathcal{D} \\ a_{t+1} \sim \pi}} \left[\frac{1}{2} \left(Q_\omega(s_t, a_t) + B - r(s_t, a_t) - \gamma \left(Q_{\bar{\omega}}(s_{t+1}, a_{t+1}) - \alpha \log \pi(a_{t+1}|s_{t+1}) \right) \right)^2 \right].$$

where B is a running average of the expected value of $\gamma\alpha \log \pi(a|s)$ throughout training.

Jacobian clamping In addition, we found that the implicit policies can potentially collapse to point masses. To mitigate this, we regularize the implicit distributions by controlling the Jacobian matrix of the policy wrt the noise source as in Odena et al. (2018); Kumar et al. (2020), aka *Jacobian clamping*. The goal is to ensure all singular values of Jacobian matrix of pushforward mapping to be higher than some constant. In our experiments, we follow the implementation of Kumar et al. (2020): (1) stochastic estimation of the singular values of Jacobian matrix at every noise, and the Jacobian is estimated by finite difference approximation, and (2) use of the penalty method (Bertsekas, 2016) to enforce the constraint. The resulting regularization term is

$$\mathcal{L}_{\text{reg}}(\pi) = \mathop{\mathbb{E}}_{\substack{s_t \sim \mathcal{D} \\ \epsilon \sim \pi \\ v \sim N(0,I)}} \left[\min \left(\frac{\| g_\phi(\epsilon + \xi v, s_t) - g_\phi(\epsilon, s_t) \|_2^2}{\xi^2 \|v\|^2} - \eta, 0 \right)^2 \right],$$

where $\eta, \xi > 0$, and $n_{\texttt{perturb}}$ number of the perturbation vector v is sampled. We then update policy π with $\hat{\nabla}_\phi \mathcal{L}(\pi) + \lambda \nabla_\phi \mathcal{L}_{\text{reg}}(\pi)$ where λ is increased throughout training. We set $\lambda = 1 + i^\nu / 1000$ at i-th iteration and $\nu \in [1.1, 1.3]$.

G.3. Experiments

For the SAC-AR-DAE experiments, aside from the common practice for SAC, we follow the experiment settings from Mazoure et al. (2019) and sample from a uniform policy for a fixed number of initial interactions (denoted as *warm-up*). We also adopt the same network architecture for the Q-network, discounting factor γ, entropy regularization coefficient α, and target smoothing coefficient τ. For AR-DAE, we use the same network architecture as VAE. We also rescale the unbounded action $\tilde{a}$ by s for better conditioning. The details of hyperparameters are described in Table 9.

We run five experiments for each environment without fixing the random seed. For every 10k steps of environment interaction, the average return of the policy is evaluated with 10 independent runs. For visual clarify, the learning curves are smoothed by second-order polynomial filter with a window size of 7 (Savitzky & Golay, 1964). For each method, we evaluate the maximum average return: we take the maximum of the average return for each experiment and the average of the maximums over the five random seeds. We also report 'normalized average return', approximately area under the learning curves: we obtain the numerical mean of the 'average returns' over iterates. We run SAC and SAC-NF with the hyperparameters reported in Mazoure et al. (2019).

G.4. Additional Experiments

In addition to the main results in Figure 7 and Table 3, we also compare the effect of Jacobian clamping regularization on implicit policy distribution in SAC-AR-DAE. In each environment, the same hyperparameters are used in SAC-AR-DAEs except for the regularization. Our results are presented in Figure S4 and Table 5, 6.

The results shows that Jacobian clamping regularization improves the performance of SAC-AR-DAE in general, especially for *Humanoid-rllab*. In *Humanoid-rllab*, we observe that implicit policy degenerates to point masses without the Jacobian clamping, potentially due to the error of AR-DAE. However, the Jacobian clamping helps to avoid the degenerate distributions, and the policy facilitates AR-DAE-based entropy gradients.

	SAC	SAC-NF	SAC-AR-DAE	SAC-AR-DAE (w/o jc)
HalfCheetah-v2	9695 ± 879	9325 ± 775	**10907 ± 664**	10677 ± 374
Ant-v2	5345 ± 553	4861 ± 1091	**6190 ± 128**	6097 ± 140
Hopper-v2	3563 ± 119	3521 ± 129	3556 ± 127	**3634 ± 45**
Walker-v2	4612 ± 249	4760 ± 624	4793 ± 395	**4843 ± 521**
Humanoid-v2	5965 ± 179	5467 ± 44	**6275 ± 202**	6268 ± 77
Humanoid (rllab)	6099 ± 8071	3442 ± 3736	**10739 ± 10335**	761 ± 413

Table 5. Maximum average return. ± corresponds to one standard deviation over five random seeds.

	SAC	SAC-NF	SAC-AR-DAE	SAC-AR-DAE (w/o jc)
HalfCheetah-v2	8089 ± 567	7529 ± 596	8493 ± 602	**8636 ± 307**
Ant-v2	3280 ± 553	3440 ± 656	**4335 ± 241**	4015 ± 363
Hopper-v2	2442 ± 426	2480 ± 587	2631 ± 160	**2734 ± 194**
Walker-v2	3023 ± 271	**3317 ± 455**	3036 ± 271	3094 ± 209
Humanoid-v2	3471 ± 505	3447 ± 260	**4215 ± 170**	3808 ± 137
Humanoid (rllab)	664 ± 321	814 ± 630	**2021 ± 1710**	332 ± 136

Table 6. Normalized average return. ± corresponds to one standard deviation over five random seeds.

H. Improved techniques for training AR-DAE and implicit models

In order to improve and stabilize the training of both the generator and AR-DAE, we explore multiple heuristics.

H.1. AR-DAE

Activity function During preliminary experiments, we observe that *smooth activation functions* are crucial in parameterizing AR-DAE as well as the residual form of regular DAE. We notice that ReLU gives less reliable log probability gradient for low density regions.

Number of samples and updates In the VAE and RL experiments, it is important to keep AR-DAE up-to-date with the generator (i.e. posterior and policy). As discussed in Appendix E, we found that increasing the number of AR-DAE updates helps a lot. Additionally, we notice that increasing n_z is more helpful than increasing $n_{\texttt{data}}$ given $n_{\texttt{data}} n_z$ is fixed.

Scaling-up and zero-centering data To avoid using small learning rate for AR-DAE in the face of sharp distributions with small variance, we choose to scale up the input of AR-DAE. As discussed in Appendix F.2, we also zero-center the

latent samples (or action samples) to train AR-DAE. This allows AR-DAE to focus more on modeling the dispersion of the distribution rather than where most of the probability mass resides.

H.2. Implicit distributions

Noise source dimensionality We note that the implicit density models can potentially be degenerate and do not admit a density function. For example, in Appendix E we show that increasing the dimensionality of the noise source improves the qualities of the implicit distributions.

Jacobian clamping Besides of increasing noise source dimensionality, we can consider Jacobian clamping distributions to prevent implicit posteriors from collapsing to point masses. As pointed out in Appendix G.2, we observe that using this regularization technique can prevent degenerate distributions in practice, as it at least regularizes the mapping locally if its Jacobian is close to singular.

	$p(x\|z)$	$q(z\|x)$ Common	Gaussian	HVI	implicit
MLP `toy`	$[2, 256]$ $[256, 256] \times 2$ $[256, d_x] \times 2$	$[d_x, 256]$	-	-	$[256 + d_\epsilon, 256]$ $[256, 256]$ $[256, d_z]$
MLP `dbmnist`	$[d_z, 300]$ $[300, d_x]$	$[d_x, 300]$	$[300, d_z] \times 2$	$[300, d_z] \times 2$ (or $[300, d_{z_0}] \times 2$)	$[300 + d_\epsilon, 300]$ $[300, d_z]$
Conv `dbmnist`	$[d_z, 300]$ $[300, 512]$ $[32, 32, 5 \times 5, 2, 2,$ deconv$]$ $[32, 16, 5 \times 5, 2, 2,$ deconv$]$ $[16, 1, 5 \times 5, 2, 2,$ deconv$]$	$[1, 16, 5 \times 5, 2, 2]$ $[16, 32, 5 \times 5, 2, 2]$ $[32, 32, 5 \times 5, 2, 2]$	$[512, 800]$ $[800, d_z] \times 2$	$[512, 800]$ $[800, d_z] \times 2$ (or $[800, d_{(z_0)}] \times 2$)	$[512 + d_\epsilon, 800]$ $[800, d_z]$
ResConv `dbmnist` (or `sbmnist`)	$[d_z, 450]$ $[450, 512]$ [upscale by 2] $[32, 32, 3 \times 3, 1, 1,$ res$]$ $[32, 32, 3 \times 3, 1, 1,$ res$]$ [upscale by 2] $[32, 16, 3 \times 3, 1, 1,$ res$]$ $[16, 16, 3 \times 3, 1, 1,$ res$]$ [upscale by 2] $[16, 1, 3 \times 3, 1, 1,$ res$]$	$[1, 16, 3 \times 3, 2, 1,$ res$]$ $[16, 16, 3 \times 3, 1, 1,$ res$]$ $[16, 32, 3 \times 3, 2, 1,$ res$]$ $[32, 32, 3 \times 3, 1, 1,$ res$]$ $[32, 32, 3 \times 3, 2, 1,$ res$]$ $[512, 450,$ res$]$	$[450, d_z] \times 2$	$[450, 450]$ $[450, d_z] \times 2$ (or $[450, d_{z_0}] \times 2$)	$[450 + d_\epsilon, 450,$ res$]$ $[450, d_z,$ res$]$

Table 7. Network architectures for the VAE experiments. Fully-connected layers are characterized by [input size, output size], and convolutional layers by [input channel size, output channel size, kernel size, stride, padding]. "res" indicates skip connection, aka residual layer (He et al., 2016). Deconvolutional layer is marked as "deconv".

			MLP `toy`	*MLP* `dbmnist`	*Conv* `dbmnist`	*ResConv* `dbmnist`	*ResConv* `sbmnist`
AR-DAE	model	parameterization	gradient	gradient	gradient	residual	residual
		network	mlp-concat	mlp-concat	mlp-concat	mlp-concat	mlp-concat
		$m_{\texttt{fc}}$	3	5	5	5	5
		$m_{\texttt{enc}}$	3	5	5	5	5
		activation	softplus	softplus	softplus	softplus	softplus
		d_h	256	256	256	512	512
		s	10000	10000	10000	100	100
	learning	n_z	256	625	256	625	625
		$n_{\texttt{data}}$	512	128	128	128	128
		n_σ	1	1	1	1	1
		N_d	1	{1,2}	{1,2}	2	2
		$\delta_{\texttt{scale}}$	0.1	{0.1, 0.2, 0.3}	{0.1, 0.2, 0.3}	{0.1, 0.2, 0.3}	{0.1, 0.2, 0.3}
		optimizer	rmsprop, 0.5	rmsprop, 0.5	rmsprop, 0.9	rmsprop, 0.9	rmsprop, 0.9
		learning rate α_θ	0.0001	0.0001	0.0001	0.0001	0.0001
Encoder/decoder	model	network	mlp	mlp	conv	rescov	rescov
		d_z	2	32	32	32	32
		d_{z_0} or d_ϵ	10	100	100	100	100
	learning	$n_{\texttt{data}}$	512	128	128	128	128
		optimizer	adam, 0.5, 0.999	adam, 0.5, 0.999	adam, 0.5, 0.999	adam ,0.9, 0.999	adam 0.9, 0.999
		learning rate $\alpha_{\phi,\omega}$	0.0001	0.0001	0.0001	{0.001, 0.0001}	{0.001, 0.0001}
		β-annealing	no	no	no	{no, 50000}	{no, 50000}
		e-train with `train+val`	no	no	no	no	yes
Evaluation		polyak (decay)	-	no	no	no	0.998
		polyak (start interation)	-	no	no	no	{0, 1000, 5000, 10000}
		n_{eval}	-	40000	40000	20000	20000

Table 8. Hyperparameters for the VAE experiments. `toy` is the 25 Gaussian dataset. `dbmnist` and `sbmnist` are dynamically and statically binarized MNIST, respectively.

			HalfCheetah-v2	Ant-v2	Hopper-v2	Walker-v2	Humanoid-v2	Humanoid (rllab)
AR-DAE	model	parameterization	gradient	gradient	gradient	gradient	gradient	gradient
		network	mlp	mlp	mlp	mlp	mlp	mlp
		$m_{\texttt{fc}}$	5	5	5	5	5	5
		$m_{\texttt{enc}}$	5	5	0	1	1	1
		activation	elu	elu	elu	elu	elu	elu
		d_h	256	256	256	256	256	256
		s	10000	10000	10000	10000	10000	10000
	learning	$n_{a,\texttt{dae}}$	128	64	128	128	64	64
		$n_{\texttt{data}}$	256	256	256	256	256	256
		n_σ	1	1	1	1	4	4
		N_d	1	1	1	1	1	1
		$\delta_{\texttt{scale}}$	0.1	0.1	0.1	0.1	0.1	0.1
		optimizer	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999
		learning rate α_θ	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
policy	model	network	mlp	mlp	mlp	mlp	mlp	mlp
		$m_{\texttt{fc}}$	1	1	1	2	2	2
		$m_{\texttt{enc}}$	1	1	2	1	3	3
		activation	elu	elu	elu	elu	elu	elu
		d_h	256	256	256	256	64	64
		d_ϵ	10	10	10	10	32	100
	learning	$n_{\texttt{perturb}}$	10	10	10	10	10	10
		optimizer	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999
		ξ, η, ν	0.01, 0.1, 1.1	0.01, 0.01, 1.1	0.01, 0.01, 1.1	0.01, 0.01, 1.1	0.01, 0.1, 1.3	0.01, 0.1, 1.3
		learning rate α_ϕ	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
Q-network	model	network	mlp	mlp	mlp	mlp	mlp	mlp
		$m_{\texttt{fc}}$	2	2	2	2	2	2
		activation	relu	relu	relu	relu	relu	relu
		d_h	256	256	256	256	256	256
	learning	optimizer	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999	adam, 0.9, 0.999
		learning rate α_ω	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
general		α	0.05	0.05	0.05	0.05	0.05	0.05
		τ	0.005	0.005	0.005	0.005	0.005	0.005
		γ	0.99	0.99	0.99	0.99	0.99	0.99
		n_Z	100	10	100	100	10	10
		target calibration	no	no	yes	no	no	no
		warm-up	10000	10000	10000	10000	10000	10000

Table 9. Hyperparameters for RL experiments.