AR-DAE: Towards Unbiased Neural Entropy Gradient Estimation

Jae Hyun Lim 1 2  Aaron Courville 1 2 3 4  Christopher Pal 1 5 4  Chin-Wei Huang 1 2

Abstract
Entropy is ubiquitous in machine learning, but it is in general intractable to compute the entropy of the distribution of an arbitrary continuous random variable. In this paper, we propose the amortized residual denoising autoencoder (AR-DAE) to approximate the gradient of the log density function, which can be used to estimate the gradient of entropy. Amortization allows us to significantly reduce the error of the gradient approximator by approaching asymptotic optimality of a regular DAE, in which case the estimation is in theory unbiased. We conduct theoretical and experimental analyses on the approximation error of the proposed method, as well as extensive studies on heuristics to ensure its robustness. Finally, using the proposed gradient approximator to estimate the gradient of entropy, we demonstrate state-of-the-art performance on density estimation with variational autoencoders and continuous control with soft actor-critic.

1. Introduction
Entropy is an information theoretic measurement of uncertainty that has found many applications in machine learning. For example, it can be used to incentivize exploration in reinforcement learning (RL) (Haarnoja et al., 2017; 2018); prevent mode-collapse of generative adversarial networks (GANs) (Balaji et al., 2019; Dieng et al., 2019); and calibrate the uncertainty of the variational distribution in approximate Bayesian inference. However, in general intractable to compute the entropy of an arbitrary random variable.

In most applications, one actually does not care about the quantity of entropy itself, but rather how to manipulate and control this quantity as part of the optimization objective. In light of this, we propose to approximately estimate the gradient of entropy so as to maximize or minimize the entropy of a data sampler. More concretely, we approximate the gradient of the log probability density function of the data sampler. This is sufficient since the gradient of its entropy can be shown to be the expected value of the path derivative (Roeder et al., 2017). We can then plug in a gradient approximator to enable stochastic backpropagation.

We propose to use the denoising autoencoder (DAE, Vincent et al. (2008)) to approximate the gradient of the log density function, which is also known as denoising score matching (Vincent, 2011). It has been shown that the optimal reconstruction function of the DAE converges to the gradient of the log density as the noise level $\sigma$ approaches zero (Alain & Bengio, 2014). In fact, such an approach has been successfully applied to recover the gradient field of the density function of high-dimensional data such as natural images (Song & Ermon, 2019), which convincingly shows DAEs can accurately approximate the gradient. However, in the case of entropy maximization (or minimization), the non-stationarity of the sampler’s distribution poses a problem for optimization. On the one hand, the log density gradient is recovered only asymptotically as $\sigma \to 0$. On the other hand, the training signal vanishes while a smaller noise perturbation is applied, which makes it hard to reduce the approximation error due to suboptimal optimization. The fact that the sampler’s distribution is changing makes it even harder to select a noise level that is sufficiently small. Our work aims at resolving this no-win situation.

In this work, we propose the amortized residual denoising autoencoder (AR-DAE), which is a conditional DAE of a residual form that takes in $\sigma$ as input. We condition the DAE on $\sigma = 0$ at inference time to approximate the log density gradient while sampling non-zero $\sigma$ at training, which allows us to train with $\sigma$ sampled from a distribution that covers a wide range of values. If AR-DAE is optimal, we expect to continuously generalize to $\sigma = 0$ to recover the log density gradient, which can be used as an unbiased estimate of the entropy gradient. We perform ablation studies on the approximation error using a DAE, and show that our method provides significantly more accurate approximation than the baselines. Finally, we apply our method to improve distribution-free inference for variational auto-
The simplest form of an implicit density model contains a generator function \( g : z \mapsto x \). The likelihood of a particular realization of \( x \) is implied by the pushforward of \( p(z) \) through the mapping \( g \).

Unlike an explicit density model, an implicit density model does not require a carefully designed parameterization for the density to be explicitly defined, allowing it to approximate arbitrary data generation process more easily. This comes at a price, though, since the density function of the implicit model cannot be easily computed, which makes it hard to approximate its entropy using Monte Carlo methods.

### 2. Denoising entropy gradient estimator

Let \( z \) and \( g \) be defined as above, and let \( \theta \) be the parameters of the mapping \( g \) (denoted \( g_\theta \)). Most of the time, we are interested in maximizing (or minimizing) the entropy of the implicit distribution of \( x = g_\theta(z) \). For example, when the mapping \( g \) is a bijection, the density of \( x = g_\theta(z) \) can be decomposed using the change-of-variable density formula, so controlling the entropy of \( x \) amounts to controlling the logarithm determinant of the Jacobian of \( g_\theta \) (Rezende & Mohamed, 2015), as illustrated in Figure 1-(a). This allows us to estimate both the entropy and its gradient. However, for an iterative optimization algorithm such as (stochastic) gradient descent, which is commonly employed in machine learning, it is sufficient to compute the gradient of the entropy rather than the entropy itself.

Following Roeder et al. (2017), we can rewrite the entropy of \( x \) by changing the variable and neglecting the score function which is 0 in expectation to get

\[
\nabla_\theta H(p_\theta(x)) = -E_z \left[ \nabla_x \log p_\theta(x) | x = g_\theta(z) \right]^T J_\theta g_\theta(z),
\]

where \( J_\theta g_\theta(z) \) is the Jacobian matrix of the random sample \( x = g_\theta(z) \) w.r.t. to the sampler’s parameters \( \theta \). See Appendix A for the detailed derivation. We emphasize that this formulation is more general as it does not require \( g \) to be bijective.

Equation (1) tells us that we can obtain an unbiased estimate of the entropy by drawing a sample of the integrand, which is the path derivative of \( z \). The integrand requires evaluating the sample \( x = g_\theta(z) \) under the gradient of its log density \( \nabla_x \log p_\theta(x) \). As \( \log p_\theta(x) \) is usually intractable or simply not available, we directly approximate its gradient using a black box function. As long as we can provide a good enough approximation to the gradient of the log density and treat it as the incoming unit in the backward differentiation (see Figure 1-(b)), the resulting estimation of the entropy gradient is approximately unbiased.

In this work, we propose to approximate the gradient of the log density using a denoising autoencoder (DAE, Vincent et al. (2008)). A DAE is trained by minimizing the reconstruction loss \( d \) of an autoencoder \( r \) with a randomly perturbed input

\[
\mathcal{L}_{\text{DAE}}(r) = \mathbb{E}[d(x, r(x + \epsilon))],
\]

where the expectation is taken over the random perturbation \( \epsilon \) and data \( x \). Alain & Bengio (2014) showed that if \( d \) is the L2 loss and \( \epsilon \) is a centered isotropic Gaussian random variable with variance \( \sigma^2 \), then under some mild regularity condition on \( \log p_\theta \) the optimal reconstruction function satisfies

\[
r^*(x) = x + \sigma^2 \nabla_x \log p_\theta(x) + o(\sigma^2),
\]

as \( \sigma^2 \to 0 \). That is, for sufficiently small \( \sigma \), we can approximate the gradient of the log density using the black box function \( f_r(x) := \frac{r(x) - x}{\sigma^2} \) assuming \( r \approx r^* \).

### 3. Error analysis of \( \nabla_x \log p_\theta(x) \approx f_r(x) \)

Naively using \( f_r(x) \) to estimate the gradient of the entropy is problematic. First of all, the division form of \( f_r \) can lead to numerical instability and magnify the error of approximation. This is because when the noise perturbation \( \sigma \) is small, \( r(x) \) will be very close to \( x \) and thus both the numerator and the denominator of \( f_r \) are close to zero.
Second, using the triangle inequality, we can decompose the error of the approximation \( \nabla_x \log p_{\theta}(x) \approx f_r(x) \) into
\[
||\nabla_x \log p_{\theta}(x) - f_r(x)|| \leq ||\nabla_x \log p_{\theta}(x) - f_{r*}(x)|| + ||f_{r*}(x) - f_r(x)||.
\]
The first error is incurred by using the optimal DAE to approximate \( \nabla_x \log p_{\theta}(x) \), which vanishes when \( \sigma \to 0 \). We refer to it as the asymptotic error. The second term is the difference between the optimal DAE and the “current” reconstruction function. Since we use a parametric family of functions (denoted by \( \mathcal{F} \)) to approximate \( f_{r*} \), it can be further bounded by
\[
||f_{r*}(x) - f_r(x)|| \leq \left[ \frac{\|f_{r*}(x) - f_{\text{param}}(x)\|}{\text{param error}} + \frac{\|f_{\text{param}}(x) - f_r(x)\|}{\text{optim error}} \right].
\]
where \( r^* := \arg \min_{r \in \mathcal{R}} \mathcal{L}_{\text{DAE}}(r) \) is the optimal reconstruction function within the family \( \mathcal{F} \). The first term measures how closely the family of functions \( \mathcal{F} \) approximates the optimal DAE, and is referred to as the parameterization error. The second term reflects the suboptimality in optimizing \( r \). It can be significant especially when the distribution of \( x \) is non-stationary, in which case \( r \) needs to be constantly adapted. We refer to this last error term as the optimization error. As we use a neural network to parameterize \( r \), the parameterization error can be reduced by increasing the capacity of the network. The optimization error is subject to the variance of the noise \( \sigma^2 \) (relative to the distribution of \( x \)), as it affects the magnitude of the gradient signal \( \mathbb{E}[\|r(x + \epsilon) - x\|^2] \). This will make it hard to design a fixed training procedure for \( r \) as different values of \( \sigma \) requires different optimization specifications to tackle the optimization error.

4. Achieving asymptotic optimality

In this section, we propose the amortized residual DAE (AR-DAE), an improved method to approximate \( \nabla_x \log p_{\theta}(x) \) that is designed to resolve the numerical instability issue and reduce the error of approximation.

4.1. Amortized residual DAE

AR-DAE (denoted \( f_{ar} \)) is a DAE of residual form conditioned on the magnitude of the injected noise, minimizing the following optimization objective.
\[
\mathcal{L}_{x^c}(f_{ar}) = \mathbb{E}_{x \sim p(x), u \sim \mathcal{N}(0,1)} [||u + \sigma f_{ar}(x + \sigma u; \sigma)||^2].
\]  
(2)
This objective involves three modifications to the regular training and parameterization of a DAE: residual connection, loss rescaling, and scale conditioning for amortization.

Residual form First, we consider a residual form of DAE (up to a scaling factor): let \( r(x) = \sigma^2 f_{ar}(x) + x \), then \( \nabla_x \log p_{\theta}(x) \) is approximately equal to
\[
\frac{r(x) - x}{\sigma^2} = \frac{\sigma^2 f_{ar}(x) + x - x}{\sigma^2} = f_{ar}.
\]
That is, this reparameterization allows \( f_{ar} \) to directly approximate the gradient, avoiding the division that can cause numerical instability. The residual form also has an obvious benefit of a higher capacity, as it allows the network to represent an identity mapping more easily, which is especially important when the reconstruction function is close to an identity map for small values of \( \sigma \) (He et al., 2016).

Loss rescaling To prevent the gradient signal from vanishing to 0 too fast when \( \sigma \) is arbitrarily small, we rescale the loss \( \mathcal{L}_{\text{DAE}} \) by a factor of \( 1/\sigma \), and since we can decouple the noise level from the isotropic Gaussian noise into \( \epsilon = \sigma u \) for standard Gaussian \( u \), the rescaled loss can be written as \( \mathbb{E}[||\sigma f_{ar}(x + \sigma u) + u||^2] \).

We summarize the properties of the optimal DAE of the rescaled residual form in the following propositions:

**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 \( \sigma \) for all \( \sigma \in \mathbb{R}\setminus\{0\} \) and \( \lim_{\sigma \to 0} f^*(x;\sigma) = \nabla_x \log p_{\theta}(x) \).

The above proposition studies the asymptotic behaviour of the optimal \( f^*_{ar} \) as \( \sigma \to 0 \). Below, we show that under the same condition, \( f^*_{ar} \) approaches the gradient of the log density function of a Gaussian distribution centered at the expected value of \( x \sim p(x) \) as \( \sigma \) is arbitrarily large.

**Proposition 2.** \( \lim_{\sigma \to 0} \frac{f^*(x;\sigma)}{\nabla_x \log p_{\theta}(x;\sigma^2 I)} \to 1. \)

Scale conditioning Intuitively, with larger \( \sigma \) values, the perturbed data \( x + \sigma u \) will more likely be “off-manifold”, which makes it easy for the reconstruction function to point back to where most of the probability mass of the distribution of \( x \) resides. Indeed, as Proposition 2 predicts, with larger \( \sigma \) the optimal \( f^*_{ar} \) tends to point to the expected value \( \mathbb{E}_p[X] \), which is shown in Figure 2-left. With smaller values of \( \sigma \), training \( f_{ar} \) becomes harder, as one has to predict the vector \( -u \) from \( x + \sigma u \) (i.e. treating \( x \) as noise and trying to recover \( u \)). Formally, the training signal (\( \Delta \)) has a decaying
rate of $O(\sigma^2)$ for small $\sigma$ values, because

$$\mathbb{E}_u[\Delta] := \mathbb{E}_u[\nabla||u + \sigma f(x + \sigma u)||^2]$$

$$= 2\sigma^2 \nabla\left(\text{tr}(\nabla_x f(x)) + \frac{1}{2}||f(x)||^2\right) + o(\sigma^2),$$

where the first term is proportional to the stochastic gradient of the implicit score matching (Hyvärinen, 2005). That is, with smaller $\sigma$ values, minimizing the rescaled loss is equivalent to score matching, up to a diminishing scaling factor. Moreover, the variance of the gradient signal $\text{Var}(\Delta)$ also has a quadratic rate $O(\sigma^2)$, giving rise to a decreasing signal-to-noise ratio (SNR) $\mathbb{E}[\Delta]/\sqrt{\text{Var}(\Delta)} = O(\sigma)$, which is an obstacle for stochastic optimization (Shalev-Shwartz et al., 2017). See Appendix C for the SNR analysis.

In order to leverage the asymptotic optimality of the gradient approximation as $\sigma \to 0$ (Figure 2-right), we propose to train multiple (essentially infinitely many) models with different $\sigma$’s at the same time, hoping to leverage the benefit of training a large-$\sigma$ model while training a model with a smaller $\sigma$.

More concretely, we condition $f_{ar}$ on the scaling factor $\sigma$, so that $f_{ar}$ can "generalize" to the limiting behaviour of $f^*_{ar}$ as $\sigma \to 0$ to reduce the asymptotic error. Note that we cannot simply take $\sigma$ to be zero, since setting $\sigma = 0$ would result in either learning an identity function for a regular DAE or learning an arbitrary function for the rescaled residual DAE (as the square loss would be independent of the gradient approximator).

The scale-conditional gradient approximator $f_{ar}(x; \sigma)$ will be used to approximate $\nabla_x \log p_q(x)$ by setting $\sigma = 0$ during inference, while $\sigma$ is never zero at training. This can be done by considering a distribution of $\sigma$, which places zero probability to the event $\{\sigma = 0\}$; e.g. a uniform density between $[0, \delta]$ for some $\delta > 0$. The issue of having a non-negative support for the distribution of $\sigma$ is that we need to rely on $f_{ar}$ to extrapolate to 0, but neural networks usually perform poorly at extrapolation. This can be resolved by having a symmetric distribution such as centered Gaussian with variance $\delta^2$ or uniform density between $[-\delta, \delta]$; owing to the symmetry of the noise distribution $N(u; 0, I)$, we can mirror the scale across zero without changing the loss:

$$\mathbb{E}_u[||u + \sigma f(x + \sigma u)||^2] = \mathbb{E}[(||(-u) + \sigma f(x + \sigma(-u))||^2)]$$

$$= \mathbb{E}[||u + (-\sigma)f(x + (-\sigma)u)||^2].$$

Furthermore, Proposition 1 implies a good approximation to $f^*_{ar}(x, \sigma')$ would be close to $f^*_{ar}(x, \sigma)$ if $\sigma'$ is sufficiently close to $\sigma$. We suspect this might help to reduce the optimization error of AR-DAE, since the continuity of both $f_{ar}$ and $f^*_{ar}$ implies that $f_{ar}(x, \sigma')$ only needs to refine $f_{ar}(x, \sigma')$ slightly if the latter already approximates the curvature of $f^*_{ar}(x, \sigma')$ well enough. Then by varying different $\sigma$ values, the conditional DAE is essentially interpolating between the gradient field of the log density function of interest and that of a Gaussian with the same expected value.

4.2. Approximation error

To study the approximation error with different variants of the proposed method, we consider a 1-dimensional mixture of Gaussians (MoG) with two equally weighted Gaussians centered at 2 and -2, and with a standard deviation of 0.5, as this simple distribution has a non-linear gradient function and an analytical form of the optimal gradient approximator $f^*$. See Appendix D.1 for the formula and an illustration of approximation with $f^*$ with different $\sigma$ values.

We let $p$ be the density function of the MoG just described. For a given gradient approximator $f$, we estimate the expected error $\mathbb{E}_p[||\nabla_x \log p(x) - f||]$ using 1000 i.i.d. samples of $x \sim p$. The results are presented in Figure 3. The curve of the expected error of the optimal $f^*_{ar}$ shows the asymptotic error indeed shrinks to 0 as $\sigma \to 0$, and it serves as a theoretical lower bound on the overall approximation error.

Our ablation includes two steps of increments. First, modifying the regular DAE (regDAE) to be of the residual form (with loss rescaling, resDAE) largely reduces the param-
5. Related Works

Denoising autoencoders were originally introduced to learn useful representations for deep networks by Vincent et al. (2008; 2010). It was later on noticed by Vincent (2011) that the loss function of the residual form of DAE is equal to the expected quadratic error $\|f - \nabla_x \log p(x)\|^2$, where $p_\sigma(x') = \int p(x') N(x'; x, \sigma^2 I) dx$ is the marginal distribution of the perturbed data, to which the author refers as denoising score matching. Minimizing expected quadratic error of this form is in general known as score matching (Hyvärinen, 2005), where $\nabla_x \log p$ is referred to as the score 1 of the density $p$. And it is clear now when we convolve the data distribution with a smaller amount of noise, the residual function $f$ tends to approximate $\nabla_x \log p(x)$ better. This is formalized by Alain & Bengio (2014) as the limiting case of the optimal DAE. Saremi et al. (2018); Saremi & Hyvarinen (2019) propose to use the residual and gradient parameterizations to train a deep energy model with denoising score matching.

As a reformulation of score matching, instead of explicitly minimizing the expected square error of the score, the original work of Hyvärinen (2005) proposes the Implicit score matching and minimizes

$$E_p \left[ \frac{1}{2} \|f(x)\|^2 + \text{tr}(\nabla_x f(x)) \right]. \quad (3)$$

Song et al. (2019) proposed a stochastic algorithm called the sliced score matching to estimate the trace of the Jacobian, which reduces the computational cost from $O(d_x^2)$ to $O(d_x)$ (where $d_x$ is the dimensionality of $x$). It was later noted by the same author that the computational cost of the sliced score matching is still much higher than that of the denoising score matching (Song & Ermon, 2019).

Most similar to our work are Song & Ermon (2019) and Bigdeli et al. (2020). Song & Ermon (2019) propose to learn the score function of a data distribution, and propose to sample from the corresponding distribution of the learned score function using Langevin dynamics. They also propose a conditional DAE trained with a sequence of $\sigma$’s in decreasing order, and anneal the potential energy for the Langevin dynamics accordingly to tackle the mixing problem of the Markov chain. Bigdeli et al. (2020) propose to match the score function of the data distribution and that of an implicit sampler. As the resulting algorithm amounts to minimizing the reverse KL divergence, their proposal can be seen as a combination of Song & Ermon (2019) and our work.

Implicit density models are commonly seen in the context of likelihood-free inference (Mescheder et al., 2017; Tran et al., 2017; Li et al., 2017; Huszár, 2017). Statistics of an implicit distribution are usually intractable, but there has been an increasing interest in approximately estimating the gradient of the statistics, such as the entropy (Li & Turner, 2018; Shi et al., 2018) and the mutual information (Wen et al., 2020).

6. More Analyses and Experiments

6.1. Energy function fitting

In Section 4.2, we have analyzed the error of approximating the gradient of the log-density function in the context of a fixed distribution. In reality, we usually optimize the distribution iteratively, and once the distribution is updated,
the gradient approximator also needs to be updated accordingly to constantly provide accurate gradient signal. In this section, we use the proposed entropy gradient estimator to train an implicit sampler to match the density of some unnormalized energy functions.

Concretely, we would like to approximately sample from a target density which can be written as $p_{\text{target}}(x) \propto \exp(-U(x))$, where $U(x)$ is an energy function. We train a neural sampler $g$ by minimizing the reverse Kullback-Leibler (KL) divergence

$$D_{KL}(p_g(x)||p_{\text{target}}(x)) = \mathbb{E}_{x \sim p_g(x)}[-\log p_{\text{target}}(x)]. \quad (4)$$

where $p_g$ is the density induced by $g$. We use the target energy functions $U$ proposed in Rezende & Mohamed (2015) (see Table 4 for the formulas). The corresponding density functions are illustrated at the first column of Figure 4.

We consider two sampling procedures for $g$. The first one has a hierarchical structure: let $z$ be distributed by $\mathcal{N}(0, I)$, and $x$ be sampled from the conditional $p_g(x|z) := p(x|\mu_g(z), \sigma^2_g(z))$ where $\mu_g$ and $\log \sigma_g$ are parameterized by neural networks. The resulting marginal density has the form $p_g(x) = \int p_g(x|z)p(z)dz$, which is computationally intractable due to the marginalization over $z$. We compare against the variational method proposed by Agakov & Barber (2004), which lower-bounds the entropy by

$$H(p_g(x)) \geq \mathbb{E}_{x,z \sim p_g(x|z)p(z)} \left[ \log \frac{p_g(x|z)p(z)}{h(z|x)} \right]. \quad (5)$$

Plugging (5) into (4) gives us an upper bound on the KL divergence. We train $p_g$ and $h$ jointly by minimizing this upper bound as a baseline.

The second sampling procedure has an implicit density: we first sample $z \sim \mathcal{N}(0, I)$ and pass it through the generator $x = g(z)$. We estimate the gradient of the negentropy of both the hierarchical and implicit models by following the approximate gradient of the log density $f_{\text{ar}} \approx \log p_g$. The experimental details can be found in Appendix E.

As shown in Figure 4, the density learned by the auxiliary method sometimes fails to fully capture the target density. As in this experiment, we anneal the weighting of the cross-entropy term from 0.01 to 1, which is supposed to bias the sampler to be rich in noise during the early stage of training, the well-known mode seeking behavior of reverse KL-minimization should be largely mitigated. This suggests the imperfection of the density trained with the auxiliary method is a result of the looseness of the variational lower bound on entropy, which leads to an inaccurate estimate of the gradient. On the other hand, the same hierarchical model and the implicit model trained with AR-DAE both exhibit much higher fidelity. This suggests our method can provide accurate gradient signal even when the sampler’s distribution $p_g$ is being constantly updated.

### 6.2. Variational autoencoder

In the previous section, we have demonstrated that AR-DAE can robustly approximate the gradient of the log density function that is constantly changing and getting closer to some target distribution. In this section, we move on to a more challenging application: likelihood-free inference for variational autoencoders (VAE, Kingma & Welling (2014); Rezende et al. (2014a)). Let $p(z)$ be the standard normal. We assume the data is generated by $x \sim p(x|z)$ which is parameterized by a deep neural network. To estimate the parameters, we maximize the marginal likelihood $\int p(x|z)p(z)dz$ of the data $x$, sampled from some data dis-

---

**Figure 5.** Density estimation with VAE on $5 \times 5$ grid MoG. 1st column: data sampled from the MoG. 2nd column: VAE+DAE in residual form trained with a large $\sigma$ value. 3rd column: VAE+DAE in residual form trained with a small $\sigma$ value. 4th column: VAE+AR-DAE. Last column: averaged log variance of $z$ throughout training.
Table 1. Dynamically binarized MNIST. \(^1\)Results taken from Mescheder et al. (2017).

| Model                          | \(\log p(x)\) |
|-------------------------------|---------------|
| (Models with a trained prior) |               |
| VLAE (Chen et al., 2016)      | -79.03        |
| PixelHVAE + VampPrior (Tomczak & Welling, 2018) | -79.78 |
| (Models without a trained prior) |               |
| VAE IAF (Kingma et al., 2016) | -79.88        |
| VAE NAF (Huang et al., 2018)  | -79.86        |
| Diagonal Gaussian             | -81.43        |
| IVI AR-DAE (ours)             | -83.62        |
| HVI aux \(^\dagger\)          | -81.82        |
| HVI auxiliary\(^\dagger\)     | -80.80        |
| HVI AR-DAE (ours)             | -83.42        |
| IVI AR-DAE (ours)             | -81.26        |

Table 2. Statically binarized MNIST.

| Model                     | \(\log p(x)\) |
|---------------------------|---------------|
| Gaussian\(^\dagger\)      | -84.40        |
| HVI aux \(^\dagger\)      | -84.63        |
| HVI AR-DAE (ours)         | -83.42        |
| IVI AR-DAE (ours)         | -83.62        |

Figure 6. Generated samples (the mean value of the decoder) of the IVI AR-DAE trained on statically binarized MNIST.

We now apply AR-DAE to approximate entropy gradient in the context of reinforcement learning (RL). We use the soft actor-critic (SAC, Haarnoja et al. (2018)), a state-of-the-art off-policy algorithm for continuous control that is designed to encourage exploration by regularizing the entropy of the policy. We train the policy \(\pi(a|s)\) to minimize the following objective:

\[
\mathcal{L}(\pi) = \mathbb{E}_{s \sim \mathcal{D}} \left[ D_{KL} \left( \pi(a|s) \left\| \frac{\exp (Q(s,a))}{Z(s)} \right. \right) \right],
\]

where \(\mathcal{D}\) is a replay buffer of the past experience of the agent, \(Q\) is a “soft” state-action value function that approximates the entropy-regularized expected return of the policy, and

Gaussians. See Appendix F.3 for the experimental details.
Figure 7. Continuous control in reinforcement learning. SAC: soft actor-critic with diagonal Gaussian. SAC-NF: soft actor-critic with normalizing flows. SAC-AR-DAE: soft actor-critic with implicit distribution trained with AR-DAE. The shaded area indicates the standard error with 5 runs.

\[
Z(s) = \int_a \exp(Q(s,a)) da
\]
is the normalizing constant of the Gibbs distribution. A complete description of the SAC algorithm can be found in Appendix G.1. We compare with the original SAC that uses a diagonal Gaussian distribution as policy and a normalizing flow-based policy proposed by Mazoure et al. (2019). We parameterize an implicit policy and use AR-DAE to approximate \( \nabla_a \log \pi(a|s) \) to estimate

\[
\nabla \varphi L(\pi) = \mathbb{E}_{s \sim D} \left[ \nabla_a \log \pi(a|s) - \nabla_a Q(s,a) \right]^\top J_{\varphi} g(\epsilon, s), \tag{7}
\]

where \( \pi(a|s) \) is implicitly induced by \( a = g_{\epsilon}(\epsilon, s) \) with \( \epsilon \sim \mathcal{N}(0, I) \). We parameterize \( f_{ar} \) as the gradient of a scalar function \( F_{ar} \), so that \( F_{ar} \) can be interpreted as the unnormalized log-density of the policy which will be used to update the soft Q-network. We run our experiments on fix continuous control environments from the OpenAI gym benchmark suite (Brockman et al., 2016) and Rllab (Duan et al., 2016). The experimental details can be found in Appendix G.2. The results are presented in Table 3 and Figure 7. We see that SAC-AR-DAE using an implicit policy improves the performance over SAC-NF. This also shows the approximate gradient signal of AR-DAE is stable and accurate even for reinforcement learning. The extended results for a full comparison of the methods are provided in Table 5 and 6.

6.4. Maximum entropy modeling

As a last application, we apply AR-DAE to solve the constrained optimization problem of the maximum entropy principle. Let \( m \in \mathbb{R}^{10} \) be a random vector and \( B \in \mathbb{R}^{10 \times 10} \) be a random matrix with \( m_i \) and \( B_{ij} \) drawn i.i.d. from \( \mathcal{N}(0, 1) \). It is a standard result that among the class of real-valued random vectors \( x \in \mathbb{R}^{10} \) satisfying the constraints \( \mathbb{E}[x] = m \) and \( \text{Var}(x) = B^\top B \), \( x \sim \mathcal{N}(m, B^\top B) \) has the maximal entropy. Similar to Loaiza-Ganem et al. (2017), we solve this constrained optimization problem but with an implicit distribution. We use the penalty method and increasingly penalize the model to satisfy the constraints. Concretely, let \( \hat{m} \) and \( \hat{C} \) be the sample mean and sample covariance matrix, respectively, estimated with a batch size of 128. We minimize the modified objective

\[
- H(p_{\theta}(x)) + \lambda \sum_{i \in \{1,2\}} c_i^2
\]

where \( c_1 = ||\hat{m} - m||_2 \) and \( c_2 = ||\hat{C} - B^\top B||_F \), with...
increasing weighting $\lambda$ on the penalty. We estimate the entropy gradient using AR-DAE, and compare against the inverse autoregressive flows (IAF, Kingma et al. (2016)). At the end of training, we estimate the earth mover’s distance (EMD) from $\mathcal{N}(m, B^\top B)$.

We repeat the experiment 256 times and report the histogram of EMD in Figure 8. We see that most of the time the implicit model trained with AR-DAE has a smaller EMD, indicating the extra flexibility of arbitrary parameterization allows it to satisfy the geometry of the constraints more easily. We leave some more interesting applications suggested in Loaiza-Ganem et al. (2017) for future work.

7. Conclusion

We propose AR-DAE to estimate the entropy gradient of an arbitrarily parameterized data generator. We identify the difficulties in approximating the log density gradient with a DAE, and demonstrate the proposed method significantly reduces the approximation error. In theory, AR-DAE approximates the zero-noise limit of the optimal DAE, which is an unbiased estimator of the entropy gradient. We apply our method to a suite of tasks and empirically validate that AR-DAE provides accurate and reliable gradient signal to maximize entropy.

Acknowledgments

We would like to thank Guillaume Alain for an insightful discussion on denoising autoencoders. Special thanks to people who have provided their feedback and advice during discussion, including Bogdan Mazoure and Thang Doan for sharing the code on the RL experiment; to Joseph Paul Cohen for helping optimize the allocation of computational resources. We thank CIFAR, NSERC and PROMPT for their support of this work.

References

Agakov, F. V. and Barber, D. An auxiliary variational method. In ICONIP, 2004.

Alain, G. and Bengio, Y. What regularized auto-encoders learn from the data-generating distribution. J. Mach. Learn. Res., 2014.

Balaji, Y., Hassani, H., Chellappa, R., and Feizi, S. Entropic gans meet vaes: A statistical approach to compute sample likelihoods in gans. In ICML, 2019.

Bertsekas, D. Nonlinear programming. 3rd edn. massachusetts: Athena scientific, 2016.

Bigdeli, S. A., Lin, G., Portenier, T., Dunbar, L. A., and Zwicker, M. Learning generative models using denoising density estimators. arXiv preprint arXiv:2001.02728, 2020.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

Burda, Y., Grosse, R. B., and Salakhutdinov, R. Importance weighted autoencoders. In ICLR, 2016.

Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. Variational lossy autoencoder. In ICLR, 2016.

Dieng, A. B., Ruiz, F. J. R., Blei, D. M., and Titsias, M. K. Prescribed generative adversarial networks. arXiv preprint arXiv:1910.04302, 2019.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In ICML, 2016.

Durrett, R. Probability: theory and examples, volume 49. Cambridge university press, 2019.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In ICML, 2018.

Geyer, C. J. Reweighting monte carlo mixtures. Technical report, University of Minnesota, 1991.

Ha, D., Dai, A. M., and Le, Q. V. Hypernetworks. In ICLR, 2017.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. Reinforcement learning with deep energy-based policies. In ICML, 2017.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In ICML, 2018.

Hasselt, H. V. Double q-learning. In NIPS, 2010.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In CVPR, 2016.

Huang, C., Krueger, D., Lacoste, A., and Courville, A. C. Neural autoregressive flows. In ICML, 2018.

Huszár, F. Variational inference using implicit distributions. arXiv preprint arXiv:1702.08235, 2017.

Hyvärinen, A. Estimation of non-normalized statistical models by score matching. Journal of Machine Learning Research, 6(Apr):695–709, 2005.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. ICLR, 2014.
Kingma, D. P., Salimans, T., Józsefowicz, R., Chen, X., Sutskever, I., and Welling, M. Improving variational autoencoders with inverse autoregressive flow. In NIPS, 2016.

Kumar, A., Poole, B., and Murphy, K. Regularized autoencoders via relaxed injective probability flow. arXiv preprint arXiv:2002.08927, 2020.

Larochelle, H. and Murray, I. The neural autoregressive distribution estimator. In AISTATS, 2011.

Li, Y. and Turner, R. E. Gradient estimators for implicit models. In ICLR, 2018.

Li, Y., Turner, R. E., and Liu, Q. Approximate inference with amortised mcmc. arXiv preprint arXiv:1702.08343, 2017.

Loaiza-Ganem, G., Gao, Y., and Cunningham, J. P. Maximum entropy flow networks. arXiv preprint arXiv:1701.03504, 2017.

Maaløe, L., Sønderby, C. K., Sønderby, S. K., and Winther, O. Auxiliary deep generative models. arXiv preprint arXiv:1602.05473, 2016.

Mazoure, B., Doan, T., Durand, A., Hjelm, R. D., and Pineau, J. Leveraging exploration in off-policy algorithms via normalizing flows. arXiv preprint arXiv:1905.06893, 2019.

Mescheder, L. M., Nowozin, S., and Geiger, A. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In ICML, 2017.

Minka, T. et al. Divergence measures and message passing. Technical report, Technical report, Microsoft Research, 2005.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

Mohamed, S. and Lakshminarayanan, B. Learning in implicit generative models. arXiv preprint arXiv:1610.03483, 2016.

Odena, A., Buckman, J., Olsson, C., Brown, T. B., Olah, C., Raffel, C., and Goodfellow, I. Is generator conditioning causally related to gan performance? arXiv preprint arXiv:1802.08768, 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Polyak, B. T. and Juditsky, A. B. Acceleration of stochastic approximation by averaging. SIAM Journal on Control and Optimization, 30(4):838–855, 1992.

Ranganath, R., Tran, D., and Blei, D. Hierarchical variational models. In ICML, 2016.

Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. In ICML, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In ICML, 2014a.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. arXiv preprint arXiv:1401.4082, 2014b.

Roeder, G., Wu, Y., and Duvenaud, D. K. Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In NIPS, 2017.

Saremi, S. and Hyvarinen, A. Neural empirical bayes. Journal of Machine Learning Research, 20:1–23, 2019.

Saremi, S., Mehrjou, A., Schölkopf, B., and Hyvärinen, A. Deep energy estimator networks. arXiv preprint arXiv:1805.08306, 2018.

Savitzky, A. and Golay, M. J. Smoothing and differentiation of data by simplified least squares procedures. Analytical chemistry, 36(8):1627–1639, 1964.

Shalev-Shwartz, S., Shamir, O., and Shammah, S. Failures of gradient-based deep learning. In ICML, 2017.

Shi, J., Sun, S., and Zhu, J. A spectral approach to gradient estimation for implicit distributions. In ICML, 2018.

Song, Y. and Ermon, S. Generative modeling by estimating gradients of the data distribution. In NeurIPS, 2019.

Song, Y., Garg, S., Shi, J., and Ermon, S. Sliced score matching: A scalable approach to density and score estimation. In UAI, 2019.

Sutton, R. S., Barto, A. G., et al. Introduction to reinforcement learning, volume 2. MIT press Cambridge, 1998.

Tomczak, J. and Welling, M. Vae with a vampprior. In AISTATS, 2018.

Tran, D., Ranganath, R., and Blei, D. Hierarchical implicit models and likelihood-free variational inference. In NIPS, 2017.
Vincent, P. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

Wen, L., Zhou, Y., He, L., Zhou, M., and Xu, Z. Mutual information gradient estimation for representation learning. In *ICLR*, 2020.

Ziebart, B. D. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.
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 \( \theta \). 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

\[
\nabla_\theta H(p_g(x)) = \nabla_\theta \mathbb{E}_{x \sim p_g(x)} \left[ - \log p_g(x) \right] \\
\overset{\Delta}{=} \nabla_\theta \mathbb{E}_{z \sim p(z)} \left[ - \log p_g(g_\theta(z)) \right] \\
= - \nabla_\theta \int p(z) \log p_g(g_\theta(z)) dz \\
= - \int p(z) \nabla_\theta \log p_g(x) |_{x=g_\theta(z)} dz - \int p(z) [\nabla_x \log p_g(x) |_{x=g_\theta(z)}] ^T J_\theta g_\theta(z) dz \\
= - \mathbb{E}_{z \sim p(z)} \left[ [\nabla_x \log p_g(x) |_{x=g_\theta(z)}] ^T J_\theta g_\theta(z) \right].
\]

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

\[
\mathbb{E}_{z \sim p(z)} \left[ \nabla_\theta \log p_g(x) |_{x=g_\theta(z)} \right] = \mathbb{E}_{x \sim p_g(x)} \left[ \nabla_\theta \log p_g(x) \right] = \int p_g(x) \nabla_\theta \log p_g(x) dx \\
= \int p_g(x) \frac{1}{p_g(x)} \nabla_\theta p_g(x) dx = \nabla_\theta \int p_g(x) dx = \nabla_\theta 1 = 0.
\]

\[\square\]

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 \( \sigma \) for all \( \sigma \in \mathbb{R} \setminus 0 \) and \( \lim_{\sigma \to 0} f^*(x; \sigma) = \nabla_x \log p_g(x) \).

**Proof.** For simplicity, 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

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

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

\[
\lim_{\sigma \to 0} f^*(x; \sigma) = \lim_{\sigma \to 0} \frac{\frac{d}{d\sigma} \mathbb{E}_u [p(x + \sigma u) u]}{\frac{d}{d\sigma} \mathbb{E}_u [p(x + \sigma u)]}.
\]
which by another application of DOM (since gradient of \( p \) is bounded) is equal to
\[
\lim_{\sigma \to 0} \frac{\mathbb{E}[\nabla p(x + \sigma u)^	op uu]}{\mathbb{E}[p(x + \sigma u)] + \sigma \mathbb{E}[\nabla p(x + \sigma u)^	op u]}.
\]
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.** \( \lim_{\sigma \to \infty} \mathbb{E}_x \log \mathcal{N}(x; \mathbb{E}_p(x), \sigma^2 I) \to 1. \)

**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 \( \frac{1}{\sigma^2} \mathbb{E}_q(\epsilon) \) 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), \( \mathbb{E}_q(\epsilon) \to \int p(x - \epsilon) \, d\epsilon \) 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 \( \nabla \) denotes the gradient wrt the parameters of \( f \). We further expand the gradient of the summand via chain rule, which yields
\[
\nabla (u_i + \sigma f_i(x + \sigma u))^2 = 2\sigma (u_i + \sigma f_i(x + \sigma u)) \nabla f_i(x + \sigma u)
\]
\[
= 2\sigma \left( u_i \nabla f_i(x + \sigma u) + \sigma f_i(x + \sigma u) \nabla f_i(x + \sigma u) \right).
\]

Taylor theorem with the mean-value form of the remainder allows us to approximate \( f_i(x + \sigma u) \) by \( f_i(x) \) as \( \sigma \) 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 \( \nabla_x \) denotes the gradient wrt the parameters 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
\[
2\sigma \left( u_i \nabla f_i(x) + \sigma \nabla_x f_i(x)^\top u + \frac{\sigma^2}{2} u^\top \nabla_x^2 f_i(\tilde{x})u \right) = 2\sigma u_i \nabla f_i(x) + 2\sigma^2 u_i \nabla_x f_i(x)^\top u + \sigma^3 u_i \nabla u^\top \nabla_x^2 f_i(\tilde{x})u + 2\sigma^2 \nabla_x f_i(x)^\top u \nabla f_i(x) + 2\sigma^3 \nabla_x f_i(x)^\top u \nabla f_i(x) + 2\sigma^4 \nabla_x f_i(x)^\top u \nabla f_i(x).
\]
With some regularity conditions (DOM-style assumptions), marginalizing out $u$ and taking $\sigma$ to be arbitrarily small yield

$$E_u[\Delta] = \sum_i 2\sigma^2 \nabla \frac{\partial}{\partial x_i} f_i(x) + 2\sigma^2 f_i(x) \nabla f_i(x) + o(\sigma^2)$$

$$= 2\sigma^2 \nabla \left( \text{tr}(\nabla_x f(x)) + \frac{1}{2} ||f(x)||^2 \right) + o(\sigma^2).$$

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 $O(\sigma^2)$ as $\sigma^2 \to 0$.

For the second moment, similarly,

$$E_u[\Delta \Delta^\top] = 4\sigma^2 \sum_i \nabla f_i(x) \nabla f_i(x)^\top + o(\sigma^2).$$

As a result,

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

**D. Main experiments**

**D.1. Error analysis**

**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,

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

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 $\mu_1 = 2$.

**Proof.** The numerator $E_u[E_u[p(x - \sigma u)u]/(x - \sigma u)u]$ can be rewritten as follows:

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

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 $\mu_1 = 2$. 

*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 $\sigma$ values (solid lines).*
The second equality comes from the fact that all $p_1$, $p_2$, and $p(u)$ are normal distributions, and thus we have:

$$p_i(x - \sigma u)p(u) = \frac{1}{\sigma}p_i(u - x/\sigma)p(u) = \frac{1}{\sigma}S'_i \mathcal{N}(u; \mu'_i, \sigma'_i).$$

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

**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: $[2 + 1, 256] + [256, 256] \times 2 + [256, 1]$, with the Softplus activation function. We use the same network architecture for resDAE except it doesn’t condition on $\sigma$. 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 $\sigma$ value in Figure 3. For regDAE annealed and resDAE annealed, we anneal $\sigma$ from 1 to the target value. For AR-DAE, $\delta$ is set to 0.05 and we sample 10 $\sigma$’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 $\sigma$**

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 $\sigma$. To contrast the difference, we sample $\sigma \sim N(0, \delta^2)$ and compare two different types of $\sigma$-conditioning: (1) conditioning on $\sigma$, 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 $\sigma$ (resp. $|\sigma|$) as input and outputs the parameters of AR-DAE, to force AR-DAE to be more dependent on the value of $\sigma$ (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 $\sigma$ values close to 0. When a symmetric $\sigma$ distribution is used, the approximation error of AR-DAE is more smooth. Second, we notice that the symmetric $\sigma$ distribution bias $f_{ar}$ to focus more on small $\sigma$ values. Finally, the asymmetric distribution helps AR-DAE reduce the approximation error for some $\sigma$. We speculate that AR-DAE with the asymmetric $\sigma$ distribution has two times higher to observe small $\sigma$-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 $\sigma$-conditioning methods to approximate log density gradient of 1D-MOG. AR-DAE: conditioning on $\sigma$. AR-DAE (|$\sigma|$): conditioning on |$\sigma$|. $\sigma$ is sampled from $N(0, \delta)$ for all experiments.](attachment:Figure_S2.png)

[^4]: $[d_{\text{input}}, d_{\text{output}}]$ denotes a fully-connected layer whose input and output feature sizes are $d_{\text{input}}$ and $d_{\text{output}}$, respectively.
E. Experiment: Energy Fitting

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:

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

where \(\alpha\) 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\(^3\) 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 \(f_{\alpha,r}\). 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.

\(^3\)diagonal elements of the covariance matrix in log-scale
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

\[
\nabla_\phi \mathcal{L}_{ELBO}(q) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[ (\nabla_z \log p_\omega(x, z) - \nabla_z \log q_\phi(z|x))\right]^T J_{\phi} g_\phi(\epsilon, x).
\]

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

\[
\nabla_\phi \mathcal{L}_{ELBO}(q) \triangleq \mathbb{E}_{z \sim q_\phi(z|x)} \left[ (\nabla_z \log p_\omega(x, z) - f_{ar,\theta}(z; x, \sigma)|_{\sigma=0})\right]^T J_{\phi} g_\phi(\epsilon, x),
\]

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 \(\omega\) 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 \(p_\omega\) 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 \(\phi\). The gradient of \(\mathcal{L}_{ELBO}\) wrt the parameters of the posterior can be written as,

\[
\nabla_\phi \mathcal{L}_{ELBO}(q) = \mathbb{E}_{z \sim q_\phi(z|x)} \left[ (\nabla_z \log p_\omega(x, z) - \nabla_z \log q_\phi(z|x))\right]^T J_{\phi} g_\phi(\epsilon, x).
\]

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

\[
\nabla_\phi \mathcal{L}_{ELBO}(q) \triangleq \mathbb{E}_{z \sim q_\phi(z|x)} \left[ (\nabla_z \log p_\omega(x, z) - f_{ar,\theta}(z; x, \sigma)|_{\sigma=0})\right]^T J_{\phi} g_\phi(\epsilon, x),
\]

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 \(\sigma\). We also adaptively choose the prior variance \(\delta^2\) for different data points instead of fixing it to be a single value.

In addition, we make the following observations, (1) The posteriors \(q_\phi\) 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 \(\tilde{z} \xleftarrow{} s(z - b(x))\), where \(s\) is a scaling factor and \(b(x)\) is a pseudo mean. Ideally, we would like to set \(b(x)\) to be \(\mathbb{E}_q(z|x)\). Instead, we let \(b(x) = g(0, x)\), 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

\[
L_{ar}(f_{ar}) = \mathbb{E}_{x \sim p(x)} \left[ \left\| u + \sigma f_{ar}(\tilde{z} + \sigma u; x, \sigma) \right\|^2 \right].
\]  

(13)

where \( \delta(x) = \delta_{scale} S_{z|x} \) and \( S_{z|x} \) is sample standard deviation of \( z \) given \( x \). We use \( n_z \) samples per data to estimate \( S_{z|x} \). \( \delta_{scale} \) 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 \( \sigma \) (denoted by “mlp-concat” in Table 8). The MLP encoders have \( m_{enc} \) hidden layers. The concatenated representation is fed into a fully-connected neural network with \( m_{dz} \) 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 \( L_{ar}(f_{ar}) \) is not backpropagated to the posterior.

Algorithm 1 VAE AR-DAE

**Input:** Dataset \( D \); mini-batch size \( n_{data} \); sample size \( n_z \); prior variance \( \delta^2 \); learning rates \( \alpha_{\theta} \) 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

\( k = 0 \ldots N_d \) do

\( n_z \) per datapoint from \( \delta \)

\( \delta_i \leftarrow \delta_{scale} S_{z|x} \), for \( i = 1, \ldots, n_{data} \)

\( \sigma_i \), number of \( \sigma \) per \( z \)

\( \sigma_i \sim N(0, \delta_i^2) \)

\( n_{\sigma} \), number of \( \sigma \) per \( z \)

\( \sigma \sim N(0, I) \)

Update \( \theta \) using gradient \( \nabla_{\theta} L_{ar} \) with learning rate \( \alpha_{\theta} \)

end for

\( z \sim q_\phi(z|x) \)

Update \( \omega \) using gradient \( \nabla_{\omega} L_{ELBO} \) with learning rate \( \alpha_{\phi, \omega} \)

Update \( \phi \) using gradient \( \nabla_{\phi} L_{ELBO} \) with learning rate \( \alpha_{\phi, \omega} \), whose entropy gradient is approximated using \( f_{ar, \theta}(z|x) \).

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 \( d_z \)-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 \( \epsilon \) in the implicit posterior, \( i.e. d_\epsilon \), 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
We write $\tau$ and evaluate $\alpha$ where the entropy-regularized reinforcement learning (Ziebart, 2010) is to learn a policy $\pi$. By using these definitions, we can rewrite $r$ the transition probability and the initial state distribution, $S$ et al., 1998), where we consider an infinite-horizon Markov decision process (MDP) defined as a tuple $G$.1. Soft actor-critic

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)

$$\log p(x) \geq \mathbb{E}_{z \sim q(z|x)} \left[ \log p(x, z) - \log q(z|x) \right] \geq \mathbb{E}_{z \sim q(z_0|x_0)} \left[ \log p(x, z) - \log q(z_0|x) - \log q(z_0, x) + \log h(z_0|z, x) \right].$$

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 data. 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 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 $(S, A, R, p_{\text{tr}}, \gamma)$ (Sutton et al., 1998), where $S$, $A$, $R$ are the spaces of state, action and reward, respectively, $p_{\text{tr}}(s_{t+1}|s_t, a_t)$ and $p_{\text{tr}}(s_0)$ represent the transition probability and the initial state distribution, $r(s_t, a_t)$ is a bounded reward function, and $\gamma$ is a discount factor. We write $\tau$ 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;

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

where $\alpha$ is an entropy regularization coefficient. We define a soft state value function $V^\pi$ and s soft Q-function $Q^\pi$ as follows,

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

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

By using these definitions, we can rewrite $V^\pi$ and $Q^\pi$ as $V^\pi(s) = \mathbb{E}_{a \sim \pi} [Q^\pi(s, a)] + \alpha H(\pi(\cdot|s))$ and $Q^\pi(s) = [r(s, a) + \mathbb{E}_{a' \sim p_{\text{tr}}} \gamma V^\pi(s')]$. 
Soft actor-critic  One way to maximize (14) is to minimize the following KL divergence,

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

where $Z(\pi_{\text{old}}(s_t))$ is the normalizing constant $\int \exp (Q_{\pi_{\text{old}}}(s_t, a)) \, da$. 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 $Q_{\pi_{\text{old}}}$ and $V_{\pi_{\text{old}}}$ by some parametric functions $Q_\omega$ and $V_\nu$, and training the policy by minimizing

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

where $D$ 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,

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

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

where $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim \text{data}} [V_\nu(s_{t+1})]$ and $V_\nu$ is a target value network. For the target value network, SAC follows Mnih et al. (2015): $V_\nu$ is defined as a polyak-averaged model (Polyak & Juditsky, 1992) of $V_\nu$. Note that $V_\nu$ is inferred from $Q_\omega$ via Monte Carlo, i.e. $V_\nu(s_t) \approx Q_\omega(s_t, a_t) - \alpha \log \pi(a_t|s_t)$ 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

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

Let $f_{ar,\theta}$ be AR-DAE which approximates $\nabla_\epsilon \log \pi_\phi(a|s)$ 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,

$$f_{ar,\theta}(a; s, \sigma) \approx \nabla_\epsilon \psi_{ar,\theta}(a; s, \sigma).$$

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 $Z_\theta(s) = \int \exp(\psi_{ar,\theta}(a; s, \sigma)|_{\sigma=0}) \, da$.

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

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

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:

$$\nabla_\phi \mathcal{L}(\pi) \approx \mathbb{E}_{s_t \sim D} \left[ f_{ar,\theta}(g_\phi(\epsilon, s_t); s, \sigma)|_{\sigma=0} - \nabla_\epsilon Q_\omega(s_t, a) | a = g_\phi(\epsilon, s_t) \right] \mathbf{J}_\phi g_\phi(\epsilon, s_t).$$

We summarize all the details in Algorithm 2.
Algorithm 2 SAC-AR-DAE

**Input:** Mini-batch size $n_{\text{data}}$; replay buffer $D$; number of epoch $T$; learning rates $\alpha_\theta, \alpha_\phi, \alpha_\omega, \alpha_\nu$

Initialize value function $V_c(s)$, critic $Q_\omega(s,a)$, policy $\pi_\phi(a|s)$, and AR-DAE $f_{ar,\theta}(a|s)$

Initialize replay buffer $D \leftarrow \emptyset$

for epoch $= 1, \ldots, T$

Initialize a state from $s_0 \sim p_{\text{env}}(s_0)$

for $t = 0 \ldots$ do

$a \sim \pi_\phi(\cdot|s_t)$

$\langle r_t, s_{t+1} \rangle \sim p_{\text{env}}(\cdot|s_t, a_t)$

$D \leftarrow D \cup \{(s_t, a_t, r_t, s_{t+1})\}$

for each learning step do

Draw $n_{\text{data}}$ number of $(s_t, a_t, r_t, s_{t+1})$s from $D$

for $k = 0 \ldots N_d$ do

Draw $n_a$ actions per state from $a \sim \pi_\phi(a|s)$

$\delta_i \leftarrow \delta_{\text{scale}} S_{a|s_t}$ for $i = 1, \ldots, n_{\text{data}}$

Draw $n_\sigma$ number of $\sigma_i$s per $a$ from $\sigma_i \sim N(0, \delta_i^2)$

Draw $n_{\text{data}} n_a n_\sigma$ number of $u$s from $u \sim N(0, I)$

Update $\theta$ using gradient $\nabla_\theta \mathcal{L}_{f_{ar}}$ with learning rate $\alpha_\theta$

end for

Update $\nu$ using gradient $\nabla_\nu \mathcal{L}_{\hat{V}}$ with learning rate $\alpha_\nu$

Update $\omega$ using gradient $\nabla_\omega \mathcal{L}_{Q}$ with learning rate $\alpha_\omega$

Update $\phi$ using gradient $\nabla_\phi \mathcal{L}_{\pi}$ which is approximated with $f_{ar,\theta}(a|s)$

$\nabla \leftarrow \nabla + (1 - \tau) \nabla$

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 $g_\phi$ (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

$$\hat{\nabla}_\phi \mathcal{L}(\pi) \doteq \frac{1}{s_t \sim \pi} \mathbb{E}_{\epsilon \sim \pi} \left[ [f_{ar,\theta}(g_\phi(\epsilon, s_t); s_t, \sigma)|_{\sigma=0} + 2 \tanh(g_\phi(\epsilon, s_t)) - \nabla_{\tilde{a}} Q_\omega(s_t, \tanh(\tilde{a}))]|_{\tilde{a}=g_\phi(\epsilon, s_t)} \right]^T J_\phi g_\phi(\epsilon, s_t).$$

**Estimating normalizing constant** In order to train SAC-AR-DAE in practice, efficient computation of $\log Z_\phi(s)$ 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)

$$\log Z_\phi(s) = \log \int \exp (\psi_{ar,\theta}(a; s, \sigma)|_{\sigma=0}) \, da$$

$$= \log \mathbb{E}_{a \sim h} \left[ \exp (\psi_{ar,\theta}(a; s, \sigma)|_{\sigma=0} - \log h(a|s)) \right]$$

$$\approx \log \frac{1}{N_Z} \sum_{j} \left[ \exp (\psi_{ar,\theta}(a_j; s, \sigma)|_{\sigma=0} - \log h(a_j|s) - A) \right] + A,$$

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


**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 with respect to the action, not the value of the Q-function itself. This means that while optimizing $Q_\omega$ (and $V_\nu$), 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 $\bar{Q}_\omega$. That is, $L(Q_\omega)$ can be rewritten as,

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

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 with respect to 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

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

where $\eta, \xi > 0$, and $n_{\text{perturb}}$ number of the perturbation vector $v$ is sampled. We then update policy $\pi$ with $\nabla_\pi L(\pi) + \lambda \nabla_\phi L_{\text{reg}}(\pi)$ where $\lambda$ 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 $\gamma$, entropy regularization coefficient $\alpha$, and target smoothing coefficient $\tau$. 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 clarity, 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.

| Environment       | 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 ± 807| 3442 ± 3736| 10739 ± 10335| 761 ± 413   |

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

| Environment       | 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 $n_{data}$ given $n_{data}n_z$ 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.
| Model | p(x|z) | q(z|x) | HVI | Implicit |
|-------|-------|--------|-----|----------|
| **MLP** | [2, 256] | [d_z, 256] | [300, d_z] x 2 | [300, d_z] x 2 |
| toy | [256, 256] x 2 | [d_z, 256] | [300, d_z] x 2 | [256, 256] |
| dbmnist | [d_z, 300] | [d_z, 300] | [300, d_z] x 2 | [300, d_z] x 2 |
| Conv | [32, 32, 5 x 5, 2, 2, deconv] | [1, 16, 3 x 3, 2, 2, 1, res] | [512, 800] x 2 | [512, 800] x 2 |
| dbmnist | [32, 32, 3 x 3, 1, 1, res] | [32, 32, 3 x 3, 2, 1, res] | [450, d_z] x 2 | [450, d_z] x 2 |
| ResConv | [32, 32, 3 x 3, 1, 1, res] | [32, 32, 3 x 3, 1, 1, res] | [512, 450, res] | [512, 450, 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”.

| Model | MLP | Conv | ResConv |
|-------|-----|------|---------|
| toy | mtc | mtc | mtc |
| dbmnist | mtc | mtc | mtc |
| Conv | n_z | 3 | 1 |
| dbmnist | 512 | 128 | 128 |
| ResConv | 10000 | 256 | 512 |

AR-DAE: Towards Unbiased Neural Entropy Gradient Estimation

Table 8. Hyperparameters for the VAE experiments. toy is the 25 Gaussian dataset. dbmnist and sbmnist are dynamically and statically binarized MNIST, respectively.
### Table 9. Hyperparameters for RL experiments.

| Parameter           | HalfCheetah-v2 | Ant-v2 | Hopper-v2 | Walker-v2 | Humanoid-v2 | Humanoid (rllab) |
|---------------------|---------------|--------|-----------|-----------|-------------|-----------------|
| AR-DAE m_{fc}       | 5             | 5      | 5         | 5         | 5           | 5               |
| AR-DAE m_{enc}      | 5             | 5      | 0         | 1         | 1           | 1               |
| AR-DAE activation   | elu            | elu    | elu       | elu       | elu         | elu             |
| AR-DAE d_{hl}       | 256           | 256    | 256       | 256       | 256         | 256             |
| AR-DAE s            | 10000         | 10000  | 10000     | 10000     | 10000       | 10000           |
| learning n_{data}   | 256           | 256    | 256       | 256       | 256         | 256             |
| learning N_{d}      | 1             | 1      | 1         | 1         | 1           | 1               |
| policy learning n_{perturb optimizer} | 10 | 10 | 10 | 10 | 10 | 10 |
| policy learning ξ, η, ν | 0.01, 0.1, 1.1 | 0.01, 0.1, 1.1 | 0.01, 0.1, 1.1 | 0.01, 0.1, 1.1 | 0.01, 0.1, 1.1 | 0.01, 0.1, 1.1 |
| policy learning learning rate α | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 |
| Q-network m_{fc}    | 2             | 2      | 2         | 2         | 2           | 2               |
| Q-network m_{enc}   | 2             | 2      | 2         | 2         | 2           | 2               |
| Q-network activation | relu          | relu    | relu       | relu       | relu         | relu             |
| Q-network d_{hl}    | 256           | 256    | 256       | 256       | 256         | 256             |
| Q-network learning optimizer learning rate α | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 |
| Q-network learning learning rate α | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 |
| general α           | 0.05          | 0.05   | 0.05      | 0.05      | 0.05        | 0.05            |
| general γ           | 0.05          | 0.05   | 0.05      | 0.05      | 0.05        | 0.05            |
| general n_{Z}       | 100           | 100    | 100       | 100       | 100         | 100             |
| general target calibration | yes | yes | yes | yes | yes | yes |
| general warm-up     | 5000          | 10000  | 10000     | 10000     | 10000       | 10000           |