Denoising Adversarial Autoencoders *

Antonia Creswell, Anil Anthony Bharath

Abstract

Unsupervised learning is of growing interest because it unlocks the potential held in vast amounts of unlabelled data to learn useful representations for inference. Autoencoders, a form of generative model, may be trained by learning to reconstruct unlabelled input data from a latent representation space. More robust representations may be produced by an autoencoder if it learns to recover clean input samples from corrupted ones. Representations may be further improved by introducing regularisation during training to shape the distribution of the encoded data in latent space. We suggest *denoising adversarial autoencoders*, which combine denoising and regularisation, shaping the distribution of latent space using adversarial training. We introduce a novel analysis that shows how denoising may be incorporated into the training and sampling of adversarial autoencoders. Experiments are performed to assess the contributions that denoising makes to the learning of representations for classification and sample synthesis. Our results suggest that autoencoders trained using a denoising criterion achieve higher classification performance, and can synthesise samples that are more consistent with the input data than those trained without a corruption process.

1 Introduction

Modelling and drawing data samples from complex, high-dimensional distributions is challenging. *Generative models* may be used to capture underlying statistical structure from real-world data. A good generative model is not only able to draw samples from the distribution of data being modelled, but should also be useful for inference.

Modelling complicated distributions may be made easier by learning the parameters of conditional probability distributions that map intermediate, *latent*, [1] variables from simpler distributions to more complex ones [3]. Often, the intermediate representations that are learned can be used for tasks such as retrieval or classification [16, 13, 18, 23].

Typically, to train a model for classification, a deep convolutional neural network may be constructed, demanding large labelled datasets to achieve high

accuracy [10]. Large labelled datasets may be expensive or difficult to obtain for some tasks. However, many state-of-the-art generative models can be trained without labelled datasets [6, 16, 9, 8]. For example, autoencoders learn a generative model, referred to as a *decoder*, by recovering inputs from corrupted [23, 8, 4] or *encoded* [9] versions of themselves.

Two broad approaches to learning state-of-the-art generative models that do not require labelled training data include: 1) introduction of a denoising criterion – where the model learns to reconstruct clean samples from corrupted ones; 2) regularisation of the latent space to match a prior [1, 9, 13] or using a latent space with a known prior distribution [6, 9, 13]; for the latter, the priors take a simple form, such as multivariate normal distributions.

The denoising variational autoencoder [8] combines both denoising and regularisation in a single generative model. However, introducing a denoising criterion makes the variational cost function – used to match the latent distribution to the prior – analytically intractable [8]. Reformulation of the cost function [8], makes it tractable, but only for certain families of prior and posterior distributions. We propose using adversarial training [6] to match the posterior distribution to the prior. Taking this approach expands the possible choices for families of prior and posterior distributions.

When a denoising criterion is introduced to an adversarial autoencoder, we have a choice to either shape the conditional distribution of latent variables given *corrupted* samples to match the prior (as done using a variational approach [8]), or to shape the *full* posterior conditional on the *original* data samples to match the prior. Shaping the posterior distribution over corrupted samples does not require additional sampling during training, but trying to shape the full conditional distribution with respect to the original data samples does. We explore both approaches, using adversarial training to avoid the difficulties posed by analytically intractable cost functions.

Additionally, a model that has been trained using the posterior conditioned on corrupted data requires an iterative process for synthesising samples, whereas using the full posterior conditioned on the original data does not. Similar challenges exist for the denoising VAE, but were not addressed by Im et al.[8]. We analyse and address these challenges for adversarial autoencoders, introducing a novel sampling approach for synthesising samples from trained models.

In summary, our contributions include: 1) Two types of denoising adversarial autoencoders, one which is more efficient to train, and one which is more efficient to draw samples from; 2) Methods to sample denoising adversarial autoencoders through Markov-Chain sampling; 3) An analysis of the quality of features learned with denoising adversarial autoencoders through their application to discriminative tasks.

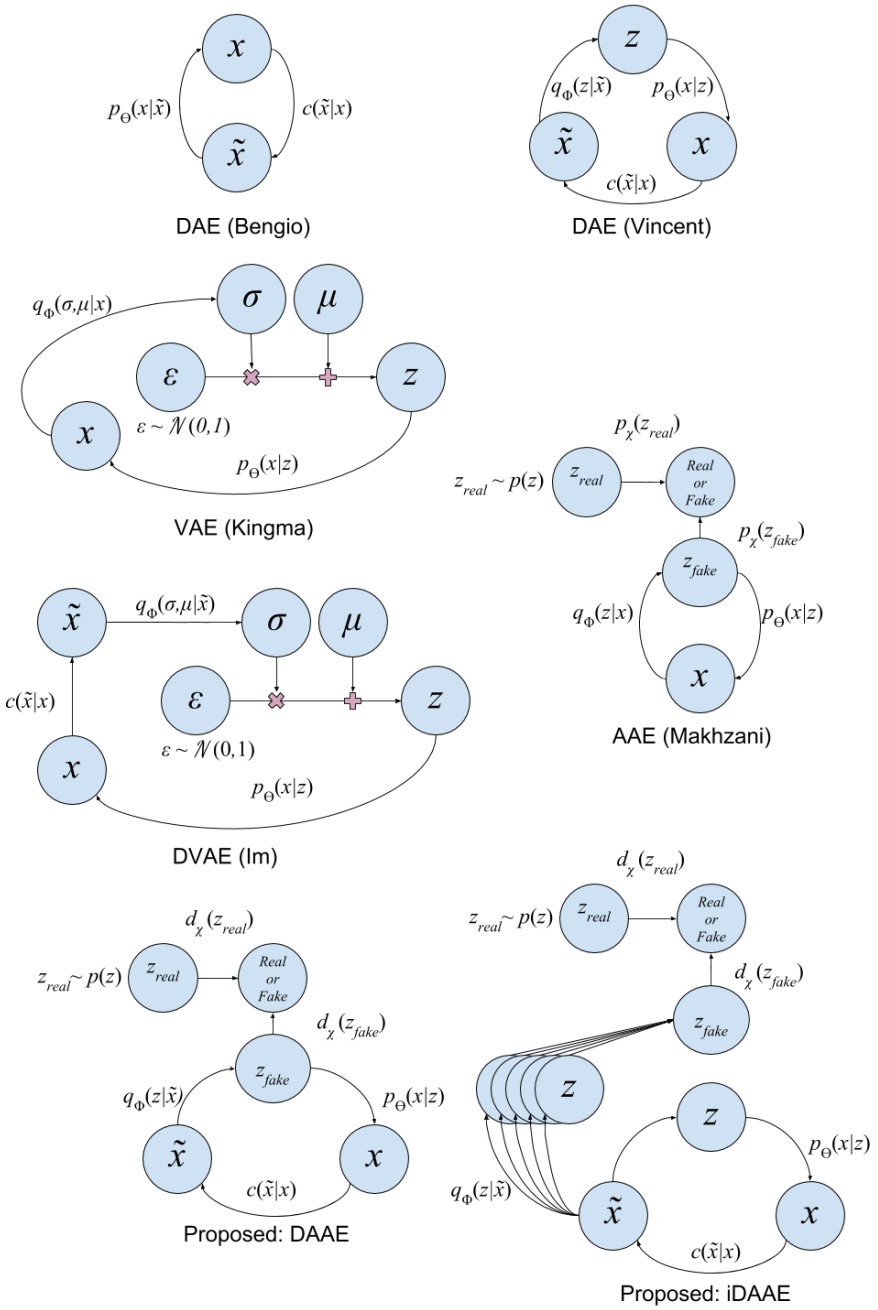


Figure 1: **Comparison of autoencoding models:** Previous works include Denoising Autoencoders (DAE) [4, 23], Variational Autoencoders (VAE) [9], Adversarial Autoencoders (AAE) [13] and Denoising Variational Autoencoders [8]. Our contributions are the DAAE and the iDAAE models. Arrows in this diagram represent mappings implemented using trained neural networks.

2 Background

2.1 Autoencoders

In a supervised learning setting, given a set of training data, $\{(y_i, x_i)\}_{i=1}^N$ we wish to learn a model, $f_\psi(y|x)$ that maximises the likelihood, $\mathrm{E}_{p(y|x)} f_\psi(y|x)$ of the true label, y given an observation, x. In the supervised setting, there are many ways to calculate and approximate the likelihood, because there is a ground truth label for every training image.

When trying to learn a generative model, $p_\theta(x)$, in the absence of a ground truth, calculating the likelihood of the model under the observed data distribution, $\mathrm{E}_{x\sim p(x)} p_\theta(x)$, is challenging. Autoencoders introduce a two step learning process, that allows the estimation, $p_\theta(x)$ of $p(x)$ via an auxiliary variable, z. The variable z may take many forms and we shall explore several of these in this section. The two step process involves first learning a *probabilistic encoder* [9], $q_\phi(z|x)$, conditioned on observed samples, and a second *probabilistic decoder* [9], $p_\theta(x|z)$, conditioned on the auxiliary variables. Using the probabilistic encoder, we may form a training dataset, $\{(z_i, x_i)\}_{i=1}^N$ where x_i is the ground truth output for $x \sim p(x|z_i)$ with the input being $z_i \sim q_\phi(z|x_i)$. The probabilistic decoder, $p_\theta(x|z)$, may then be trained on this dataset in a supervised fashion. By sampling $p_\theta(x|z)$ conditioning on suitable z's we may obtain a joint distribution, $p_\theta(x, z)$, which may be marginalised by integrating over all z to obtain, to $p_\theta(x)$.

In some situations the encoding distribution is chosen rather than learned, in other situations the encoder and decoder are learned simultaneously.

2.2 Denoising Autoencoders (DAEs)

Bengio et al. [4] treat the encoding process as a local corruption process, that does not need to be learned. The corruption process, defined as $c(\tilde{x}|x)$ where $\tilde{x}$, the corrupted x is the auxiliary variable (instead of z). The decoder, $p_\theta(x|\tilde{x})$, is therefore trained on the data pairs, $\{(\tilde{x}_i, x_i)\}_{i=1}^N$.

By using a local corruption process (e.g. additive white Gaussian noise [4]), both $\tilde{x}$ and x have the same number of dimensions and are close to each-other. This makes it very easy to learn $p_\theta(x|\tilde{x})$. Bengio et al. [4] shows how the learned model may be sampled using an iterative process, but does not explore how representations learned by the model may transfer to other applications such as classification.

Hinton et al. [7] show that when auxiliary variables of an autoencoder have lower dimension than the observed data, the encoding model learns representations that may be useful for tasks such as classification and retrieval.

Rather than treating the corruption process, $c(\tilde{x}, x)$, as an encoding process [4] – missing out on potential benefits of using a lower dimensional auxiliary variable – Vincent et al. [23, 24] learn an encoding distribution, $q_\phi(z|\tilde{x})$, conditional on corrupted samples. The decoding distribution, $p_\theta(x|z)$ learns to reconstruct images from encoded, corrupted images, see the DAEs in Figure 1.

Vincent et al. [24, 23] show that compared to regular autoencoders, denoising autoencoders learn representations that are more useful and robust for tasks such as classification. Parameters, ϕ and θ are learned simultaneously by minimisng the reconstruction error for the training set, $\{(\tilde{x}_i, x_i)\}_{i=1}^{M}$, which does not include z_i. The ground truth z_i for a given $\tilde{x}_i$ is unknown. The form of the distribution over z, to which x samples are mapped, $p_\theta(z)$ is also unknown - making it difficult to draw novel data samples from the decoder model, $p_\theta(x|z)$.

2.3 Variational Autoencoders

Variational autoencoders (VAEs) [9] specify a prior distribution, $p(z)$ to which $q_\phi(z|x)$ should map all x samples, by formulating and maximising a variational lower bound on the log-likelihood of $p_\theta(x)$.

The variational lower bound on the log-likelihood of $p_\theta(x)$ is given by [9]:

$$\log p_\theta(x) \geq \mathrm{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)] \tag{1}$$

The $p_\theta(x|z)$ term corresponds to the likelihood of a reconstructed x given the encoding, z of a data sample x. This formulation of the variational lower bound does not involve a corruption process. The term $KL[q_\phi(z|x)||p(z)]$ is the Kullback-Libeller divergence between $q_\phi(z|x)$ and $p(z)$. Samples are drawn from $q_\phi(z|x)$ via a re-parametrisation trick.

The re-parametrisation trick may be explained as follows, when $q_\phi(z|x)$ is a multivariate Gaussian, a probabilistic encoder $q_\phi(z|x)$ is made up of a (further) two step process. $q_\phi(z|x)$ is chosen to be a parametrised multivariate Gaussian, $\mathcal{N}(\mu_\phi(x), \sigma_\phi(x))$. In the first step, a model predicts the parameters, $\sigma_\phi(x)$ and $\mu_\phi(x)$. In the second step, a random variable $\epsilon \sim \mathcal{N}(0, 1)$ and $z = \mu_\phi(x) + \sigma_\phi(x) \circ \epsilon$ where $\circ$ is the Hadamard product, see the VAE in Figure 1.

If $q_\phi(z|x)$ is chosen to be a Gaussian distribution and the prior is chosen to be a Gaussian distribution, then $KL[q_\phi(z|x)||p(z)]$ may be computed analytically. KL divergence may only be computed analytically for certain (limited) choices of prior and posterior distributions.

VAE training encourages $q_\phi(z|x)$ to map observed samples to the chosen prior, $p(z)$. Therefore, novel observed data samples may be generated via the following **simple** sampling process: $z_i \sim p(z)$, $x_i \sim p_\theta(x|z_i)$ [9].

Note that despite the benefits of the denoising criterion shown by Vincent et al. [23, 24], no corruption process was introduced during the training of a VAE [9].

2.4 Denoising Variational Autoencoders

Adding the denoising criterion to a variational autoencoder is non-trivial because the variational lower bound becomes intractable.

Consider the conditional probability density function, $\tilde{q}_\phi(z|x) = \int q_\phi(z|\tilde{x})c(\tilde{x}|x)d\tilde{x}$, where $q_\phi(z|\tilde{x})$ is the probabilistic encoder conditioned on corrupted x samples,

$\tilde{x}$, and $c(\tilde{x}|x)$ is a corruption process. The variational lower bound may be formed in the following two ways [8]:

$$\log p_\theta(x) \geq \mathrm{E}_{\tilde{q}_\phi(z|x)} \log \left[\frac{p_\theta(x,z)}{q_\phi(z|\tilde{x})} \right] \geq \mathrm{E}_{\tilde{q}_\phi(z|x)} \log \left[\frac{p_\theta(x,z)}{\tilde{q}_\phi(x|z)} \right]$$

If $q_\phi(z|\tilde{x})$ is chosen to be Gaussian, then in many cases $\tilde{q}_\phi(z|x)$ will be a mixture of Gaussians. If this is the case, there is no analytical solution for $KL[\tilde{q}_\phi(z|x)||p(z)]$ and so the denoising variational lower bound becomes analytically intractable. However there may still be an analytical solution for $KL[q_\phi(z|\tilde{x})||p(z)]$. The denoising variational autoencoder therefore maximises $\mathrm{E}_{\tilde{q}(x|z)} \log[\frac{p_\theta(x,z)}{q_\phi(z|\tilde{x})}]$. We refer to the model which is trained to maximise this objective as a DVAE, see the DVAE in Figure 1. Im et al. [8] showed that the DVAE achieves lower negative variational lower bounds than the regular variational autoencoder on a test dataset.

However, note that $q_\phi(z|\tilde{x})$ is matched to the prior, $p(z)$ rather than $\tilde{q}_\phi(z|x)$. This means that generating novel samples using $p_\theta(z|x)$ is not as simple as the process of generating samples from a variational autoencoder. To generate novel samples, we should sample $z_i \sim \tilde{q}_\phi(z|x)$, $x_i \sim p_\theta(x|z_i)$, which is difficult because of the need to evaluate $\tilde{q}_\phi(z|x)$. Im et al. [8] do not address this problem.

For both DVAEs and VAEs there is a limited choice of prior and posterior distributions for which there exists an analytic solution for the KL divergence. Alternatively, adversarial training may be used to learn a model that matches samples to an arbitrarily complicated target distribution – provided that samples may be drawn from both the target and model distributions.

3 Related Work

3.1 Adversarial Training

In adversarial training [6] a model $g_\phi(w|v)$ is trained to produce samples that match a target probability distribution $t(w)$. This is achieved by iteratively training two competing models, a generative model, $g_\phi(w|v)$ and a discriminative model, $d_\chi(w)$. The discriminative model is fed with samples either from the generator (i.e. 'fake' samples) or with samples from the target distribution (i.e. 'real' samples), and trained to correctly predict whether samples are 'real' or 'fake'. The generative model is trained to generate samples that are indistinguishable from target samples in order to 'fool' [16] the discriminative model into making incorrect predictions. This may be achieved by the following mini-max objective [6]:

$$\min_g \max_d \mathrm{E}_{w\sim t(w)}[\log d_\chi(w)] + \mathrm{E}_{w\sim g_\phi(w|v)}[\log(1 - d_\chi(w))]$$

It has been shown that for an optimal discriminative model, optimising the generative model is equivalent to minimising the Jensen-Shannon divergence between the generated and target distributions [6]. In general, it is reasonable

to assume that, during training, the discriminative model quickly achieves near optimal performance [6]. This property is useful for learning distributions for which the Jensen-Shannon divergence may not be easily calculated.

The generative model is optimal when the distribution of generated samples matches the target distribution. Under these conditions, the discriminator is maximally confused and cannot distinguish 'real' samples from 'fake' ones. As a consequence of this, adversarial training may be used to capture very complicated data distributions, and has been shown to be able to synthesise images of handwritten digits and human faces that are almost indistinguishable from real data [16].

3.2 Adversarial Autoencoders

Makhzani et al. [13] introduced the adversarial autoencoder (AAE), where $q_\phi(z|x)$ is both the probabilistic encoding model in an autoencoder framework and the generative model in an adversarial framework. A new, discriminative model, $d_\chi(z)$ was introduced. The discriminative model was trained to distinguish between latent samples drawn from $p(z)$ and $q_\phi(z|x)$. The cost function used to train the discriminator, $d_\chi(z)$ is:

$$\mathcal{L}_{dis} = -\frac{1}{N}\sum_{i=0}^{N-1}[\log d_\chi(z_i)] - \frac{1}{N}\sum_{j=N}^{2N}[\log(1 - d_\chi(z_j))]$$

where $z_{i=1\ldots N-1} \sim p(z)$ and $z_{j=N\ldots 2N} \sim q_\phi(z|x)$ and N is the size of the training batch.

Adversarial training was used to match $q_\phi(z|x)$ to an arbitrarily chosen prior, $p(z)$. The cost function for matching $q_\phi(z|x)$ to prior, $p(z)$ is as follows:

$$\mathcal{L}_{prior} = \frac{1}{N}\sum_{i=0}^{N-1}[\log(1 - d_\chi(z_i))]$$

where $z_{i=0\ldots N-1} \sim q_\phi(z|x)$ and N is the size of a training batch. If both $\mathcal{L}_{prior}$ and $\mathcal{L}_{dis}$ are optimised, $q_\phi(z|x)$ will be indistinguishable from $p(z)$.

In Makhzani et al.'s [13] adversarial autoencoder, $q_\phi(z|x)$ is specified by a neural network whose input is x and whose output is z. This allows $q_\phi(z|x)$ to have arbitrary complexity, unlike the VAE where the complexity of $q_\phi(z|x)$ is usually limited to a Gaussian. In an adversarial autoencoder the posterior does not have to be analytically defined because an adversary is used to match the prior, avoiding the need to analytically compute a KL divergence.

Makhzani et al. [13] demonstrate that adversarial autoencoders are able to match $q_\phi(z|x)$ to several different priors, $p(z)$, including a mixture of 10 2D-Gaussian distributions. We explore another direction for adversarial autoencoders, by extending them to incorporate a denoising criterion.

4 Denoising Adversarial Autoencoder

We propose denoising adversarial autoencoders - denoising autoencoders that use adversarial training to match the distribution of auxiliary variables, z to a prior distribution, $p(z)$.

We formulate two versions of a denoising adversarial autoencoder which are trained to approximately maximise the denoising variational lower bound [8]. In the first version, we directly match the posterior $\tilde{q}_\phi(z|x)$ to the prior, $p(z)$ using adversarial training. We refer to this as an integrating Denoising Adversarial Autoencoder, iDAAE. In the second, we match intermediate conditional probability distribution $q_\phi(z|\tilde{x})$ to the prior, $p(z)$. We refer to this as a DAAE.

In the iDAAE, adversarial training is used to bypass analytically intractable KL divergences [8]. In the DAAE, using adversarial training broadens the choice for prior and posterior distributions beyond those for which the KL divergence may be analytically computed.

4.1 Construction

The distribution of encoded data samples is given by $\tilde{q}_\phi(z|x) = \int q_\phi(z|\tilde{x})c(\tilde{x}|x)d\tilde{x}$ [8]. The distribution of decoded data samples is given by $p_\theta(x|z)$. Both $q_\phi(z|x)$ and $p_\theta(x|z)$ may be trained to maximise the likelihood of a reconstructed sample, by minimising the reconstruction cost function:

$$\mathcal{L}_{rec} = \frac{1}{N}\sum_{i=0}^{N-1} p(x_i)\log p_\theta(x|z_i)$$

where $z_{i=0\ldots N-1} \sim q_\phi(z|x_i), x_{i=0\ldots N-1} \sim p(x)$, and $p(x)$ is distribution of the training data.

We also want to match the distribution of auxiliary variables, z to a prior, $p(z)$. When doing so, there is a choice to match either $\tilde{q}_\phi(z|x)$ or $q_\phi(z|\tilde{x})$ to $p(z)$. Each choice has its own trade-offs either during training or during sampling.

4.1.1 iDAAE: Matching $\tilde{q}_\phi(z|x)$ to a prior

In DVAEs there is often no analytical solution for the KL divergence between $\tilde{q}_\phi(z|x)$ and $p(z)$ [8], making it difficult to match $\tilde{q}_\phi(z|x)$ to $p(z)$. Rather, we propose using adversarial training to match $\tilde{q}_\phi(z|x)$ to $p(z)$, requiring samples to be drawn from $\tilde{q}_\phi(z|x)$ during training. It is challenging to draw samples directly from $\tilde{q}_\phi(z|x) = \int q_\phi(z|\tilde{x})c(\tilde{x}|x)d\tilde{x}$, but it is easy to draw samples from $q_\phi(z|\tilde{x})$ and so $\tilde{q}_\phi(z|x)$ may be approximated by $\frac{1}{M}\sum_{i=1}^{M} q_\phi(z|\tilde{x_i})$, $\tilde{x}_{i=1\ldots M} \sim c(\tilde{x}|x_0)$, $x_0 \sim p(x)$ where $x \sim p(x)$ are samples from the training data, see Figure 1. Matching is achieved by minimising the following cost function:

$$\mathcal{L}_{prior} = \frac{1}{N}\sum_{i=0}^{N-1}[\log(1 - d_\chi(\hat{z}_i))]$$

where:

$$\hat{z}_{i=0\ldots N-1} = \frac{1}{M}\sum_{j=1}^{M} z_{i,j}, \qquad z_{i=1\ldots N, j=1\ldots M} \sim q_\phi(z|\tilde{x}_{i,j}),$$

$$\tilde{x}_{i=1\ldots N, j=1\ldots M} \sim c(\tilde{x}|x_i), \qquad x_{i=0\ldots N-1} \sim p(x)$$

4.1.2 DAAE: Matching $q_\phi(z|\tilde{x})$ to a prior

Since drawing samples from $q_\phi(z|\tilde{x})$ is trivial, $q_\phi(z|\tilde{x})$ may be matched to $p(z)$ via adversarial training. This is more efficient than matching $\tilde{q}_\phi(z|x)$ since a Monte-Carlo integration step (in Section 4.1.1) is not needed, see Figure 1. In using adversarial training in place of KL divergence, the only restriction is that we must be able to draw samples from the chosen prior. Matching may be achieved by minimising the following loss function:

$$\mathcal{L}_{prior} = \frac{1}{N}\sum_{i=0}^{N-1}[\log(1 - d_\chi(z_i))]$$

where $z_{i=1\ldots N-1} \sim q_\phi(z|\tilde{x_i})$.

Though more computationally efficient to train, there are drawbacks when trying to synthesise novel samples from $p_\theta(x)$ if $q_\phi(z|\tilde{x})$ – rather than $\tilde{q}_\phi(z|x)$ – is matched to the prior.

5 Synthesising novel samples

In this section, we review several techniques used to draw samples from trained autoencoders, identify a problem with sampling DVAEs, which also applies to DAAEs, and propose a novel approach to sampling DAAEs; we draw strongly on previous work by Bengio et al. [3, 4].

5.1 Drawing Samples From Autoencoders

New samples may be generated by sampling a learned $p_\theta(x|z)$, conditioning on z drawn from a suitable distribution. In the case of variational [9] and adversarial [13] autoencoders, the choice of this distribution is simple, because during training the distribution of auxiliary variables is matched to a chosen prior distribution, $p(z)$. It is therefore easy and efficient to sample both variational and adversarial autoencoders via the following process: $z \sim p(z)$, $x \sim p_\theta(x|z)$ [9, 13].

The process for sampling denoising autoencoders is more complicated. In the case where the auxiliary variable is a corrupted image, $\tilde{x}$ [2], the sampling process is as follows: $x_0 \sim p(x)$, $\tilde{x}_0 \sim c(\tilde{x}|x_0)$, $x_1 \sim p_\theta(x|\tilde{x}_0)$ [4]. In the case where the auxiliary variable is an encoding, [23, 24] the sampling process is the same, with $p_\theta(x|\tilde{x})$ encompassing both the encoding and decoding process.

However, since a denoising autoencoder is trained to reconstruct corrupted versions of its inputs, x_1 is likely to be very similar to x_0. Bengio et al. [4] proposed a method for iteratively sampling denoising autoencoders by defining a Markov chain whose stationary distribution - under certain conditions - exists and is equivalent, under certain assumption, to the training data distribution. This approach was generalised and extended by Bengio et al. [3] to introduce a latent distribution with no prior assumptions on z.

We now consider the implication for drawing samples from denoising adversarial autoencoders introduced in Section 4.1. By using the iDAAE formulation (Section 4.1.1) – where $\tilde{q}_\phi(z|x)$ is matched to the prior over z – x samples may be drawn from $p_\theta(x|z)$ conditioning on $z \sim p(z)$. However, if we use the DAAE – matching $q_\phi(z|\tilde{x})$ to a prior – sampling becomes non-trivial.

On the surface, it may appear easy to draw samples from DAAEs (Section 4.1.2), by first sampling the prior, $p(z)$ and then sampling $p_\theta(x|z)$. However, the full posterior distribution is given by $\tilde{q}_\phi(z|x) = \int q_\phi(z|\tilde{x})c(\tilde{x}|x)d\tilde{x}$, but only $q_\phi(z|\tilde{x})$ is matched to $p(z)$ during training. The implication of this is that, when attempting to synthesize novel samples from $p_\theta(x|z)$, drawing samples from the prior, $p(z)$, is unlikely to yield samples consistent with $p(x)$. This will be come clearer in Section 5.2.

5.2 Proposed Method For Sampling DAAEs

Here, we propose a method for synthesising novel samples using trained DAAEs. In order to draw samples from $p_\theta(x|z)$, we need to be able to draw samples from $\tilde{q}_\phi(z|x)$.

To ensure that we draw novel data samples, we do not want to draw samples from the training data at any point during sample synthesis. This means that the method we used previously in Algorithm 2 of the Appendix (Section 4.1.1) to approximately draw samples from $\tilde{q}_\phi(z|x)$ may no longer be used.

Instead, similar to Bengio et al. [4], we formulate a Markov chain, which we show has the necessary properties to converge and that the chain converges to $\mathcal{P}(z) = \int \tilde{q}_\phi(z|x)p(x)dx$. Unlike Bengio's formulation, our chain is initialised with a random vector of the same dimensions as the latent space, rather than a sample drawn from the training set.

We define a Markov chain by the following sampling process:

$$\begin{aligned} z^{(0)} \sim \mathbb{R}^a, \quad x^{(t)} \sim p_\theta(x|z^{(t)}), \\ \tilde{x}^{(t)} \sim c(\tilde{x}|x^{(t)}), \quad z^{(t+1)} \sim q_\phi(z|\tilde{x}^{(t)}), \\ t \geq 0. \end{aligned} \tag{2}$$

Notice that our first sample is any real vector of dimension a, where a is the dimension of the latent space. This Markov chain has the transition operator:

$$\begin{aligned} T_{\theta,\phi}(z^{(t+1)}|z^{(t)}) = \\ \int q_\phi(z^{(t+1)}|\tilde{x}^{(t)})c(\tilde{x}^{(t)}|x^{(t)})p_\theta(x^{(t)}|z^{(t)})dxd\tilde{x} \end{aligned} \tag{3}$$

We will now show that under certain conditions this transition operator defines an ergodic Markov chain that converges to $\mathcal{P}(z) = \int \tilde{q}_\phi(z|x)p(x)dx$ in the following steps:

1. We will show that that there exists a stationary distribution $\mathcal{P}(z)$ for $z^{(0)}$ drawn from a specific choice of initial distribution (Lemma 10.1).
2. The Markov chain is homogeneous, because the transition operator is defined by a set of distributions whose parameters are fixed during sampling.
3. We will show that the Markov chain is also ergodic, (Lemma 10.2).
4. Since the chain is both homogeneous and ergodic there exists a unique stationary distribution to which the Markov chain will converge [21]. Step 1 shows that one stationary distribution is $\mathcal{P}(z)$, which we now know by 2) and 3) to be the unique stationary distribution. So the Markov chain converges to $\mathcal{P}(z)$.

In this section, only, we use a change of notation, where the training data probability distribution, previously represented as $p(x)$ is represented as $\mathcal{P}(x)$, this is to help make distinctions between "natural system" probability distributions and the learned distributions. Further, note that $p(z)$ is the prior, while the distribution required for sampling $\mathcal{P}(x|z)$ is $\mathcal{P}(z)$ such that:

$$\mathcal{P}(x) = \int \mathcal{P}(x|z)\mathcal{P}(z)dz \approx \int p_\theta(x|z)\mathcal{P}(z)dz. \tag{4}$$

$$\mathcal{P}(z) = \int \tilde{q}_\phi(z|x)\mathcal{P}(x)dx = \int\int q_\phi(z|\tilde{x})c(\tilde{x}|x)d\tilde{x}\mathcal{P}(x)dx. \tag{5}$$

Lemma 5.1. *$\mathcal{P}(z)$ is a stationary distribution for the Markov chain defined by the sampling process in 2.*

For proof see Appendix.

Lemma 5.2. *The Markov chain defined by the transition operator, $T_{\theta,\phi}(z_{t+1}|z_t)$ is ergodic, provided that the corruption process is additive Gaussian noise and that the adversarial pair, $q_\phi(z|\tilde{x})$ and $d_\chi(z)$ are optimal within the adversarial framework.*

For proof see Appendix.

Theorem 5.3. *Assuming that $p_\theta(x|z)$ is a sufficient approximation of $\mathcal{P}(x|z)$, and that the adversarial pair – $q_\phi(z|x)$ and $d_\chi(z)$ – are optimal, the transition operator $T_{\theta,\phi}(z^{(t+1)}|z^{(t)})$ defines a Markov chain whose unique stationary distribution is $\mathcal{P}(z) = \int \tilde{q}_\phi(z|x)\mathcal{P}(x)dx$.*

Proof. This follows from Lemmas 5.1 and 5.2. □

This sampling method uncovers the distribution $\mathcal{P}(z)$ on which samples drawn from $p_\theta(x|z)$ must be conditioned in order to sample $p_\theta(x)$. Assuming $p_\theta(x|z) \approx \mathcal{P}(x|z)$, this allows us to draw samples from $\mathcal{P}(x)$.

For completeness, we would like to acknowledge that there are several other methods that use Markov chains during the training of autoencoders [1, 14] to improve performance. Our approach for synthesising samples using the DAAE is focused on sampling only from trained models; the Markov chain sampling is not used to update model parameters.

6 Implementation

The analyses of Sections 4 and 5 are deliberately general: they do not rely on any specific implementation choice (e.g. particles, artificial neural networks, or parametrised forms of distribution) to capture the model distributions. In this section, we consider a specific implementation of denoising adversarial autoencoders and apply them to the task of learning models for image distributions. We define an encoding model that maps corrupted data samples to a latent space $E_\phi(\tilde{x})$, and $R_\theta(z)$ which maps samples from a latent space to an image space. These respectively draw samples according to the conditional probabilities $q_\phi(z|\tilde{x})$ and $p_\theta(x|z)$. We also define a corruption process, $C(x)$, which draws samples according to $c(\tilde{x}|x)$.

We represent an image in the form of a 3D array; all values are between $(0, 1)$, and so a pixel value may be interpreted as a probability of that pixel being on or off.

The parameters θ and ϕ of models $R_\theta(z)$ and $E_\phi(z)$ are learned under an autoencoder framework; the parameters ϕ are also updated under an adversarial framework. The models are trained using large datasets of unlabelled images.

6.1 The Autoencoder

Under the autoencoder framework, $E_\phi(x)$ is the encoder and $R_\theta(z)$ is the decoder. We used fully connected neural networks for both the encoder and decoder. Non-linear activation functions were used between all intermediate layers to encourage the networks to learn representations that capture multi-modal distributions. In the final layer of the decoder network, a sigmoid activation function is used so that the outputs of neurons represent pixels of an image. The final layer of the encoder network is left as a linear layer, so that the distribution of encoded samples is not restricted.

As described in Section 4.1, the autoencoder is trained to maximise the log-likelihood of the reconstructed image given the corrupted image. Although there are several ways in which one may evaluate this log-likelihood, we chose to measure pixel wise binary cross-entropy between the reconstructed sample, $\hat{x}$ and the original samples before corruption, x. During training we aim to learn parameters ϕ and θ that minimise the binary cross-entropy between $\hat{x}$ and x. The training process is summarised by steps 1 to 9 in Algorithm 1.

Minimising reconstruction error is not sufficient to match either $q_\phi(z|\tilde{x})$ or $\tilde{q}_\phi(z|x)$ to the prior, $p(z)$. For this, parameters ϕ must also be updated under the adversarial framework.

6.2 Adversarial Training

To perform adversarial training we define the discriminator $d_\chi(z)$, described in Section 3.1 to be a fully connected neural network, which we denote $D_\chi(z)$. The output of $D_\chi(z)$ is a "probability" because the final layer of the neural network has a sigmoid activation function, constraining the range of $D_\chi(z)$ to be between $(0, 1)$. Intermediate layers of the network have non-linear activation functions to encourage the network to capture highly non-linear relations between z and the labels, {'real', 'fake'}.

How adversarial training is applied depends on whether $\tilde{q}_\phi(z|x)$ or $q_\phi(z|\tilde{x})$ is being fit to the prior $p(z)$. z_{fake} refers to the samples drawn from the distribution that we wish to fit to $p(z)$ and z_{real}, samples drawn from the prior, $p(z)$. The discriminator, $D_\chi(z)$, is trained to predict whether z's are 'real' or 'fake'. This may be achieved by learning parameters χ that maximise the probability of the correct labels being assigned to z_{fake} and z_{real}. This training procedure is show in Algorithm 1 on Lines 14 to 16.

Drawing samples, z_{real}, involves sampling some prior distribution, $p(z)$, often a Gaussian. Now, we consider how to draw fake samples, z_{fake}. How these samples are drawn depends on whether $q_\phi(z|\tilde{x})$ (DAAE) is being fit to the prior or $\tilde{q}_\phi(z|x)$ (iDAAE) is being fit to the prior. Drawing samples, z_{fake} is easy if $q_\phi(z|\tilde{x})$ is being matched to the prior, as these are simply obtained by mapping corrupted samples though the encoder: $z_{fake} = E_\phi(\tilde{x})$.

However, if $\tilde{q}(z|x)$ is being matched to the prior, we must use Monte Carlo sampling to approximate z_{fake} samples (see Section 4.1.1). The process for calculating z_{fake} is given by Algorithm 2 in the Appendix, and detailed in Section 4.1.1.

Finally, in order to match the distribution of z_{fake} samples to the prior, $p(z)$, adversarial training is used to update parameters ϕ while holding parameters χ fixed. Parameters ϕ are updated to minimise the likelihood that $D_\chi(\cdot)$ correctly classifies z_{fake} as being 'fake'. The training procedure is laid out in lines 18 and 19 of Algorithm 1.

Algorithm 1 shows the steps taken to train an iDAAE. To train a DAAE instead, all lines in Algorithm 1 are the same except Line 11, which may be replaced by $z_{fake} = E_\phi(\tilde{x})$.

6.3 Sampling

Although the training process for matching $\tilde{q}_\phi(z|x)$ to $p(z)$ is less computationally efficient than matching $q_\phi(z|\tilde{x})$ to $p(z)$, it is very easy to draw samples when $\tilde{q}_\phi(z|x)$ is matched to the prior (iDAAE). We simply draw a random $z^{(0)}$ value from $p(z)$, and calculate $x^{(0)} = R_\theta(z^{(0)})$, where $x^{(0)}$ is a new sample. When drawing samples, parameters θ and ϕ are fixed.

Algorithm 1: iDAAE : Matching $\tilde{q}_\phi(z|x)$ to $p(z)$

$\mathbf{x} = \{x_1, x_2, x_{N-1}\} \sim p(x)$ #draw a batch of samples from the training data
for $k = 1$ *to NoEpoch* **do**
  $\tilde{\mathbf{x}} = C(\mathbf{x})$ #corrupt all samples
  $\mathbf{z} = E_\phi(\tilde{\mathbf{x}})$ #encode all corrupted samples
  $\hat{\mathbf{x}} = R_\theta(\mathbf{z})$ #reconstruct
  #minimise reconstruction cost
  $\mathcal{L}_{rec} = -(\hat{\mathbf{x}} \log \mathbf{x} + (1 - \hat{\mathbf{x}}) \log(1 - \mathbf{x})).mean()$
  $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_{rec}$
  $\theta \leftarrow \theta - \alpha \nabla_\phi \mathcal{L}_{rec}$
  #match $\tilde{q}_\phi(z|x)$ to $p(z)$ using adversarial training
  $\mathbf{z}_{fake}$ =approx_z($\mathbf{x}$) #draw samples for $\tilde{q}_\phi(z|x)$
  $\mathbf{z}_{real} \sim p(z)$ #draw samples from prior $p(z)$
  #train the discriminator
  $\mathcal{L}_{dis} = -\frac{1}{N} \sum_{i=0}^{N-1} \log D_\chi(\mathbf{z}_{real_i})$
  $-\frac{1}{N} \sum_{i=0}^{N-1} \log(1 - D_\chi(\mathbf{z}_{fake_i}))$
  $\chi \leftarrow \chi - a \nabla_\chi \mathcal{L}_{dis}$
  #train the decoder to match the prior
  $\mathcal{L}_{prior} = \frac{1}{N} \sum_{i=0}^{N-1} \log(1 - D_\chi(\mathbf{z}_{fake_i}))$
  $\phi \leftarrow \phi - \alpha \nabla_\phi$
end

If $q_\phi(z|\tilde{x})$ is matched to the prior (DAAE), an iterative sampling process is needed in order to draw new samples from $p(x)$. This sampling process is described in Section 5.2. To implement this sampling process is trivial. A random sample, $z^{(0)}$ is drawn from any distribution; the distribution does not have to be the chosen prior, $p(z)$. New samples, $z^{(t)}$ are obtained by iteratively decoding, corrupting and encoding $z^{(t)}$, such that $z^{(t+1)}$ is given by:

$$z^{(t+1)} = E_\phi(C(R_\theta(z^{(t)}))).$$

In the following section, we evaluate the performance of denoising adversarial autoencoders on three image datasets, a handwritten digit dataset (MNIST) [12], a synthetic colour image dataset of tiny images (Sprites) [17], and a complex dataset of hand-written characters [11]. The denoising and non-denoising adversarial autoencoders are compared for tasks including reconstruction, generation and classification.

7 Experiments & Results

7.1 Code Available Online

The code and results for our experiments are presented in iPython notebooks at the following link: https://github.com/ToniCreswell/DAAE_.

7.2 Datasets

We evaluate our denoising adversarial autoencoder on three image datasets of varying complexity. Here, we describe the datasets and their complexity in terms of variation within the dataset, number of training examples and size of the images.

7.2.1 Datasets: MNIST

The MNIST dataset consists of grey-scale images of handwritten digits between 0 and 9, with 50k training samples, 10k validation samples and 10k testing samples. The training, validation and testing dataset have an equal number of samples from each category. The samples are 28-by-28 pixels. The MNIST dataset is a simple dataset with few classes and many training examples, making it a good dataset for proof-of-concept. However, because the dataset is very simple, it does not necessarily reveal the effects of subtle, but potentially important, changes to algorithms for training or sampling. For this reason, we consider two datasets with greater complexity.

7.2.2 Datasets: Omniglot

In contrast to the MNIST dataset, the Omniglot dataset is a handwritten character dataset consisting of 1623 categories of character from 50 different writing systems, with only 20 examples of each character. Each example in the dataset

is 105-by-105 pixels. The dataset is split such that 19 examples from 964 categories make up the training dataset, while one example from each of those 964 categories makes up the testing dataset. The 20 characters from each of the remaining 659 categories make up the evaluation dataset. This means that experiments may be performed to reconstruct or classify samples from categories not seen during training of the autoencoders.

7.2.3 Datasets: Sprites

The sprites dataset is made up of 672 unique human-like characters. Each character has 7 attributes including hair, body, armour, trousers, arm and weapon type, as well as gender. For each character there 20 animations consisting of 6 to 13 frames each. There are between 120 and 260 examples of each character, however every example is in a different pose. Each sample is 60-by-60 pixels and samples are in colour. The training, validation and test datasets are split by character to have 500, 72 and 100 unique characters each, with no two sets having the same character.

7.3 Architecture and Training

For each dataset, we detail the architecture and training parameters of the networks used to train each of the denoising adversarial autoencoders. For each dataset, several DAAEs, iDAAEs and AAEs are trained. In order to compare models trained on the same datasets, the same network architectures, batch size, learning rate, annealing rate, size of latent code and number of epochs is used for each. For DAAEs and iDAAEs, the same level of corruption is also used. The trained AAE [13] models act as a benchmark, allowing us to compare our proposed DAAEs and iDAAEs.

7.3.1 Architecture and Training: MNIST

For the MNIST dataset, we train a total of 5 models detailed in Table 1. The encoder, decoder and discriminator networks each have two fully connected layers with 1000 neurons each. For most models the size of the encoding is 10 units, and the prior distribution that the encoding is being matched to is a 10D Gaussian. All networks are trained for 100 epochs on the training dataset, with a learning rate of 0.0002 and a batch size of 64. The standard deviation of the additive Gaussian noise used during training is 0.5 for all iDAAE and DAAE models.

An additional DAAE model is trained using the same training parameters and networks as described above but with a mixture of 10 2D Gaussians for the prior. Each 2D Gaussian with standard deviation 0.5 is equally spaced with its mean around a circle of radius 4 units. This results in a prior with 10 modes separated from each other by large regions of very low probability. This model of the prior is very unrealistic, as it assumes that MNIST digits occupy distinct regions of image probability space. In reality, we may expect two numbers that

Table 1: **Models trained on MNIST:** 5 models are trained on the MNIST dataset. *Corruption* indicates the standard deviation of Gaussian noise added during the corruption process, $c(\tilde{x}|x)$. *Prior* indicated the prior distribution imposed on the latent space. M is the number of Monte Carlo integration steps (see Algorithm 2 in Appendix) used during training - this applies only to the iDAAE.

ID	Model	Corruption	Prior	M
1	AAE	0.0	10D Gaussian	-
2	DAAE	0.5	10D Gaussian	-
3	DAAE	0.5	10-GMM	-
4	iDAAE	0.5	10D Gaussian	5
5	iDAAE	0.5	10D Gaussian	25

are similar to exist side by side in image probability space, and for there to exist a smooth transition between handwritten digits. As a consequence of this, this model is intended specifically for evaluating how well the posterior distribution over latent space may be matched to a mixture of Gaussians - something that could not be achieved easily by a VAE [9].

7.3.2 Architecture and Training: Omniglot

For the Omniglot dataset, 3 models were trained: an AAE, a DAAE and an iDAAE. Compared to the networks used for MNIST, deeper networks were needed in order for the loss functions to converge. The decoder, encoder and discriminator networks consisted of 6,3 and 2 fully connected layers respectively, each layer having 1000 neurons. The networks are trained for 1000 epochs on the training dataset, using a learning rate 1×10^{-5} and batch size of 64. The prior is a 200D Gaussian. The corruption process used to train the DAAE and iDAAE is additive Gaussian noise with standard deviation 0.5. When training the iDAAE, the number of steps used in Monte Carlo integration (see Algorithm 2 in the Appendix) was $M = 5$.

7.3.3 Architecture and Training: Sprites

For the sprites dataset, 3 models were trained: an AAE, a DAAE and an iDAAE. Both the encoder and discriminator are 2-layer fully connected neural networks with 1000 neurons in each layer. For the decoder, we found that it was necessary to use a 3-layer fully connected neural network in order to capture the complexity of the sprite dataset. However, by comparing training and test data reconstruction error during training, we found that using 1000 neurons in each layer led to over-fitting. Rather, we used a 3-layer fully connected neural network with 1000 neurons in the first layer and 500 in each of the last layers. The networks were trained for 5 epochs, using a batch size of 128 and a learning rate

of 1×10^{-4}. The size of the encoding is 200 units, and the prior is a 200D Gaussian. The corruption process used to train the DAAE and iDAAE is additive Gaussian noise with standard deviation 0.25. For the iDAAE we use $M = 5$. Using larger M values became impractical as training time increases linearly with M.

7.4 Reconstruction

The reconstruction task involves corrupting a sample from a test dataset (i.e. a sample not seen during training) and passing it through the encoder and decoder to recover the original sample. The reconstruction is evaluated by computing the mean squared error between the reconstruction and the original sample. Because DAAEs and iDAAEs reconstruct corrupted samples and an AAE reconstructs non-corrupted samples, we might expect the DAAE and the iDAAE to out-perform the AAE on reconstruction tasks. We do not expect a large difference between the DAAE and the iDAAE; however, one might anticipate that reconstructions from the iDAAE are closer to mean (average) images, and lacking in detail. This is because – assuming that the posterior is matched perfectly to a Gaussian prior – $\tilde{q}_\phi(z|x)$ for the iDAAE will be a unimodal Gaussian, while $\tilde{q}_\phi(z|x)$ for the DAAE will be a mixture of Gaussians, making the latent space more expressive.

7.4.1 Reconstruction: MNIST

Table 2 shows reconstruction error on samples from the testing dataset, for each of the models trained on the MNIST training dataset. Reconstruction error for the DAAE and iDAAE trained with a 10D Gaussian prior do not outperform the AAE. However, the reconstruction task for the AAE model was less challenging than that of the DAAE and iDAAE models because samples were not corrupted before the encoding step. The best model for reconstruction was the iDAAE with $M = 5$, increasing M to 25 led to worse reconstruction error, as reconstructions tended to appear more like mean samples.

Reconstruction error for the DAAE trained using a 2D mixture of 10 Gaussians under-performed compared to the rest of the models. All reconstructions looked like mean images, which may be expected given the nature of the prior.

7.4.2 Reconstruction: Omniglot

Table 3 presents reconstruction error on samples from the testing dataset for each model trained on the Omniglot training dataset. The reconstruction errors for both the iDAAE and the DAAE are less than the AAE. Perhaps more impressive is that the AAE reconstruction are on non-corrupted samples, while the DAAE and iDAAE are on corrupted samples. The results suggest that using the denoising criterion during training helps the network learn more robust features compared to the non-denoising variant. The corruption process at test time was the same as that used during training.

Table 2: **MNIST Reconstruction:** *Recon.* shows the mean squared error for reconstructions of corrupted test data samples accompanied by the standard error. *Corruption* is the standard deviation of the additive Gaussian noise used during training and testing.

Model & Training				Recon.
Model	Corruption	Prior	M	Mean ± s.e.
AAE	0.0	10D Gaussian	-	0.017±0.001
DAAE	0.5	10D Gaussian	-	0.023±0.001
DAAE	0.5	2D 10-GMM	-	0.043±0.001
iDAAE	0.5	10D Gaussian	5	0.022±0.001
iDAAE	0.5	10D Gaussian	25	0.026±0.001

Table 3: **Omniglot Reconstruction:** *Recon.* shows the mean squared error for reconstructions of corrupted test data samples accompanied by the standard error. *Corruption* is the standard deviation of additive Gaussian noise used during training and testing.

Model & Training				Recon.
Model	Corruption	Prior	M	Mean
AAE	0.0		-	0.047
DAAE	0.5	200D Gaussian	-	0.029
iDAAE	0.5		5	0.033

The smallest reconstruction error was achieved by the DAAE rather than the iDAAE; qualitatively, the reconstructions using the DAAE captured small details while the iDAAE lost some. This is likely to be related to the multimodal nature of $\tilde{q}_\phi(z|x)$ in the DAAE compared to the unimodal nature of $\tilde{q}_\phi(z|x)$ in an iDAAE.

7.4.3 Reconstruction: Sprites

Table 4 shows reconstruction error on samples from the sprite test dataset for models trained on the sprite training data. Similarly to the results on the Omniglot dataset, both the DAAE and the iDAAE models out-performed the AAE. Again, results are for the AAE reconstructing non-corrupted samples, while the DAAE and iDAAE models are reconstructing samples that have been corrupted. Consistent with results found on the Omniglot dataset, the DAAE out-performed the iDAAE model.

Table 4: **Sprite Reconstruction:** *Recon.* shows the mean squared error for reconstructions of corrupted test data samples accompanied by the standard error. *Corruption* is the standard deviation of additive Gaussian noise used during training and testing.

Model & Training				Recon.
Model	Corruption	Prior	M	Mean ± s.e.
AAE	0.0	200D Gaussian	-	0.019±0.006
DAAE	0.25		-	0.017±0.005
iDAAE	0.25		5	0.018±0.006

7.5 Sampling DAAEs and iDAAEs

Having considered how well DAAEs and iDAAEs can reconstruct samples, we now look to how well they synthesise samples. In reconstruction, we start from a test data sample, corrupt and encode it to get a latent sample and decode to reconstruct the original. For synthesis the initial sample, z, is drawn from latent space and passed through a decoder to synthesise a data sample, x.

Samples may be synthesised using the decoder of a trained iDAAE or AAE by passing latent samples drawn from the prior through the decoder. On the other hand, if we pass samples from the prior through the decoder of a trained DAAE, the samples are likely to be inconsistent with the training data. To synthesise more consistent samples using the DAAE, we draw an initial $z^{(0)}$ from any random distribution – we use a normal distribution for simplicity – and decode, corrupt and encode the sample several times for each synthesised sample. This process is equivalent to sampling a Markov chain where one iteration of the Markov chain includes decoding, corrupting and encoding to get a $z^{(t)}$ after t iterations. The $z^{(t)}$ may be used to synthesise a novel sample which we call, $x^{(t)}$. $x^{(0)}$ is the sample generated when $z^{(0)}$ is passed through the decoder.

To evaluate the quality of synthesised samples, we calculated the log-likelihood of real samples under the model [13]. This is achieved by fitting a Parzen window to a number of synthesised samples. Further details of how the log-likelihood is calculated for each dataset is given in the following subsections.

We expect the log-likelihood of samples drawn from a trained iDAAE to be higher (better) than for samples drawn from a trained AAE. We expect initial samples, $x^{(0)}$'s drawn from the DAAE to have a lower (worse) log-likelihood than those drawn from the AAE, however we expect Markov chain sampling to improve synthesised samples, such that $x^{(t)}$ for $t > 0$ should have larger log-likelihood than samples drawn from the AAE. It is not clear whether $x^{(t)}$ for $t > 0$ drawn using a DAAE will be better than samples drawn form an iDAAE.

7.5.1 Sampling: MNIST

To calculate the log-likelihood of samples drawn from DAAE, iDAAE and AAE models trained on the MNIST dataset, a Parzen window is fitted to 1×10^4

generated samples from a single trained model. The bandwidth for constructing the Parzen window was selected by choosing one of 10 values, evenly space along a log axis between -1 and 0, that maximises the likelihood on a validation dataset. The log-likelihood for each model was then evaluated on the test dataset.

Table 5 shows the log-likelihood on the test dataset for samples drawn from models trained on the MNIST training dataset. First, we will discuss models trained with a 10D Gaussian prior. The best log-likelihood was achieved by the iDAAE with $M = 5$. Synthesised samples generated by Markov chain sampling from the DAAE model caused the log-likelihood to decrease. This may be because the samples tended towards mean samples, dropping modes, causing the log-likelihood to decrease. Initial samples and those obtained after 5 iterations of sampling are shown in Figure 2. The samples in 2(d) are clearer than in 2(c). Although mode-dropping is not immediately apparent, note that the digit 4 is not present after 5 iterations.

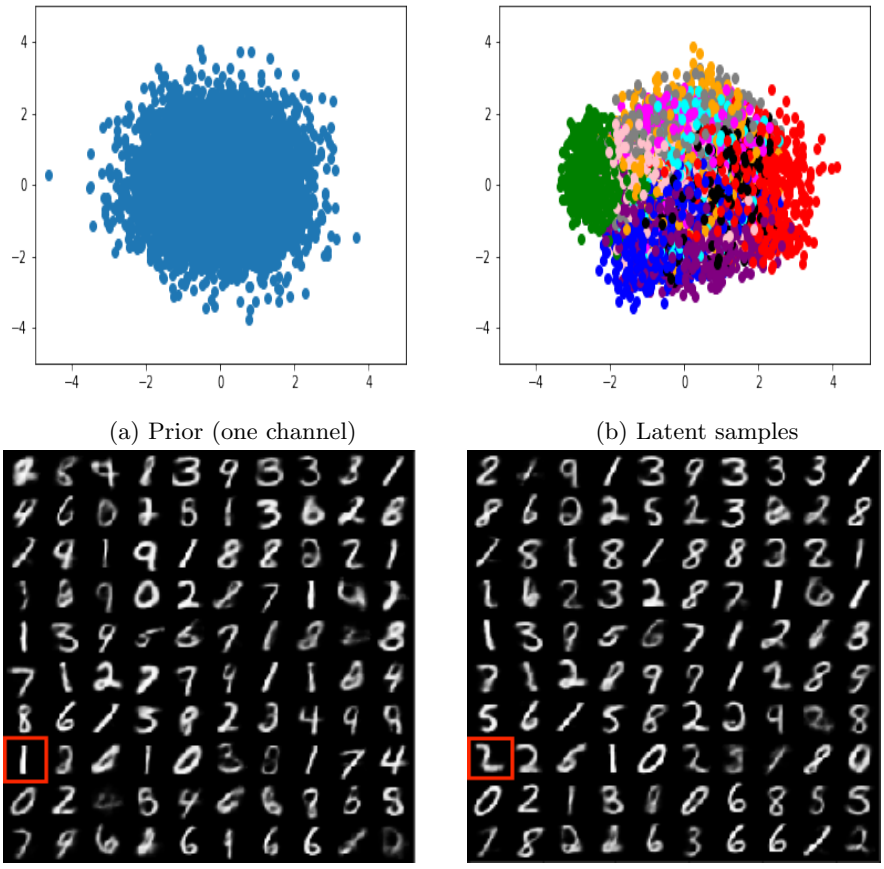
Now, we consider the DAAE model trained with a 2D, 10-GMM prior. Samples drawn from the prior are shown in Figure 3(a). The purpose of this experiment was to show the effects of Markov chain sampling, where the distribution from which initial samples of z_0 are drawn is significantly different to the prior. Samples of z_0 were drawn from a normal distribution and passed through the decoder of the DAAE model to produce initial image samples, see Figure 3(c). A further 9 steps of Markov chain sampling were applied to synthesise the samples shown in Figure 3(d). As expected, the initial image samples do not look like MNIST digits, and Markov chain sampling improves samples dramatically. Unfortunately however, many of the samples appear to correspond to samples at modes of the data distribution. In addition, several modes appear to be missing from the model distribution. This may be attributed to the nature of the prior, since we did not encounter this problem to the same extent when using a 10D Gaussian prior (see Figure 2).

Synthesising MNIST samples is fairly trivial, since there are many training examples and few classes. For this reason, it is difficult to see the benefits of using DAAE or iDAAE models compared to AAE models. Now, we focus on the two more complex datasets, Omniglot and Sprites. We find that iDAAE models and correctly sampled DAAE models may be used to synthesise samples with higher log-likelihood than samples synthesised using an AAE.

7.5.2 Sampling: Sprites

To calculate log-likelihood of samples, a Parzen window was fit to 1×10^3 synthesised samples. The bandwidth was set as previously described in Section 7.5.1; we found the optimal bandwidth to be 1.29. The log-likelihood was evaluated on the testing dataset to give the results shown in Table 6.

In alignment with expectation, the iDAAE model synthesises samples with higher (better) log-likelihood than the AAE. The initial image samples drawn from the DAAE model under-perform compared to the AAE model, however after just one iteration of sampling the synthesised samples have higher log-



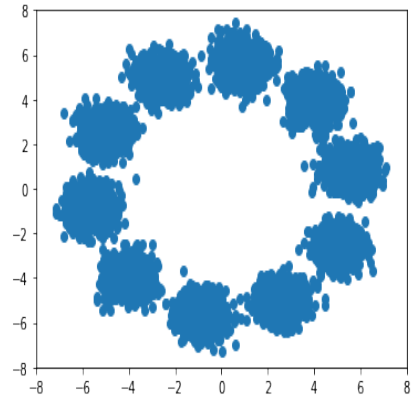
(a) Prior (one channel)

(b) Latent samples

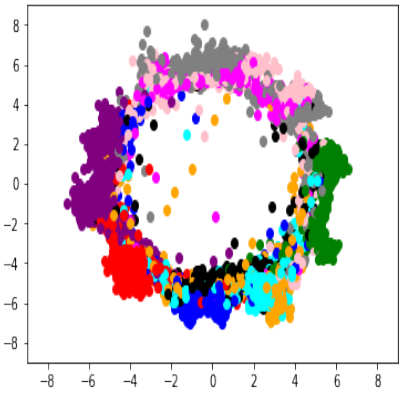
(c) $x^{(0)}$

(d) $x^{(5)}$

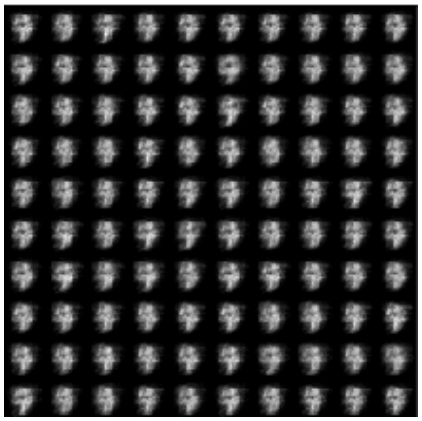
Figure 2: **MNIST Markov chain sampling: One channel of the 10D Gaussian Prior** a) One channel of the 10D Gaussian prior used to train the DAAE. b) Samples drawn from $q_\phi(z|\tilde{x})$, projected in to 2D space, and colour coded by the digit labels. c) Initial samples, $x^{(0)}$ generated by a normal distribution. d) **Corresponding** samples after 5 iterations of Markov chain sampling. Notice how the highlighted "1" changes to a "2" after 5 iterations of Markov chain sampling.



(a) 2D Gaussian Mixture prior



(b) Latent samples



(c) $x^{(0)}$

(d) $x^{(9)}$

Figure 3: **MNIST Markov chain sampling:** a) 2D Gaussian Mixture prior used to train a DAAE. b) Samples drawn from $q_\phi(z|\tilde{x})$ projected into 2D space using PCA and colour coded by the digit labels. c) Initial sample, $x^{(0)}$, generated by a normal distribution. d) *Corresponding* samples generated after 9 further iterations of Markov chain sampling.

Table 5: **MNIST log-likelihood of $p_\theta(x)$:** To calculate the log-likelihood of $p_\theta(x)$, a Parzen window was fit to 10^4 generated samples and the mean likelihood was reported for a testing data set. The bandwidth used for the Parzen window was determined using a validation set. The training, test and validation datasets had different samples.

Model & Training				Log-likelihood	
Model	Corruption	Prior	M	$x^{(0)}$	$x^{(5)}$
AAE	0.0	10D Gaussian	-	529	-
DAAE	0.5	10D Gaussian	-	529	444
DAAE	0.5	2D 10-GMM	-	9	205
iDAAE	0.5	10D Gaussian	5	532	-
iDAAE	0.5	10D Gaussian	25	508	-

Table 6: **Sprites log-likelihood of $p_\theta(x)$:** To calculate the log-likelihood of $p_\theta(x)$, a Parzen window was fit to 10^3 generated samples and the mean likelihood was reported for a testing data set. The bandwidth used for the Parzen window was determined using a validation set. The training, test and validation datasets have samples from different character classes.

Model & Training				Log-likelihood	
Model	Corruption	Prior	M	$x^{(0)}$	$x^{(1)}$
AAE	0.0	200D Gaussian	-	2085±5	-
DAAE	0.25	200D Gaussian	-	2056±5	2261±5
iDAAE	0.25	200D Gaussian	5	2122±4	-

likelihood than samples from the AAE. Results also show that synthesised samples drawn using the DAAE after one iteration of Markov sampling have higher likelihood than samples drawn using the the iDAAE model.

When more that one step of Markov chain sampling is applied, the log-likelihood decreases. As with the MNIST dataset, this may be caused by synthesised samples tending towards mean samples, thus dropping modes. This may also be related to how the training, and test data are split, each dataset has a unique set of characters, so combinations seen during training will not be present in the testing dataset, which further suggests that Markov chain sampling pushes synthesised samples towards modes in the training data. Next, we explore the Omniglot dataset, where we look at scores on both a testing and evaluation dataset, where the testing dataset has samples from the same classes as the training dataset - we will see that when evaluating log-likelihood on the testing dataset, the log-likelihood of synthesised samples increases with the number of Markov chain sampling steps.

7.5.3 Sampling: Omniglot

To calculate log-likelihood of samples, a Parzen window was fit to 1×10^3 synthesised samples, where the bandwidth was determined on the testing dataset in a similar way to that in Section 7.5.1. The log-likelihood was evaluated on both the evaluation dataset, and the testing dataset. To compute the log-likelihood on the of the testing dataset a Parzen window was fit to a new set of synthesised samples, different to those used to calculate the bandwidth. The results are shown in Tables 7 and 8.

First, we discuss the results on the evaluation dataset. The results, shown in Table 7, are consistent with what is expected of the models. The iDAAE out-performed the AAE, with a less negative (better) log-likelihood. The initial samples drawn using the DAAE had more negative (worse) log-likelihood values than samples drawn using the AAE. However, after one iteration of Markov chain sampling, the synthesised samples have less negative (better) log-likelihood values than those from the AAE. When more iterations of Markov chain sampling were applied using the DAAE, the log-likelihood of synthesised samples became more negative (worse) compared to just one iteration. This behaviour is likely to be for reasons discussed in Section 7.5.2.

Unlike the sprites dataset, where the training, test and validation datasets have samples from different classes, the Omniglot testing dataset consists of one example of every category in the training dataset. This means that if multiple iterations of Markov sampling cause synthesised samples to tend towards modes in the training data, the likelihood score on the testing dataset is likely to increase. The results shown in Table 8 confirm this expectation; the log-likelihood for the 5^{th} sample is less negative (better) than for the 1^{st} sample. These apparently conflicting results (in Tables 8 and 7) – whether sampling improves or worsens synthesised samples – highlights the challenges involved with evaluating generative models using the log-likelihood, discussed in more depth by Theis et al. [22]. For this reason, we also show qualitative results.

Figure 4 shows samples synthesised using the DAAE with Markov chain sampling. The visual quality of samples appears to increase with the number of iterations. We show fewer, larger image samples so that changes may be seen more easily. Further, to show that the effects of mode dropping are not visually detectable, we also show a larger set of initial ($x^{(0)}$) and synthesised samples after 9 iterations ($x^{(9)}$) of Markov chain sampling in Figure 5(b). The samples shown in Figure 5(b) appear to be varied; this result should be compared to those of Figure 3(d), where synthesised samples are visually very similar.

We conclude this section on sampling AAE, DAAE and iDAAE models, by making the following observations; samples synthesised using iDAAEs outperformed AAEs on all datasets, where $M = 5$. Improvement, when using a relatively small M value is convenient for training purposes, as the time needed to train networks increases linearly with M. We also observed that initial samples synthesised using the DAAE are poor and in all cases and – except the DAAE model trained on MNIST with a 10D Gaussian prior – even one iteration of Markov chain sampling improves image synthesis. We also showed that

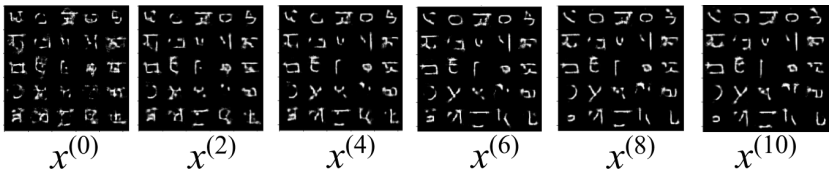


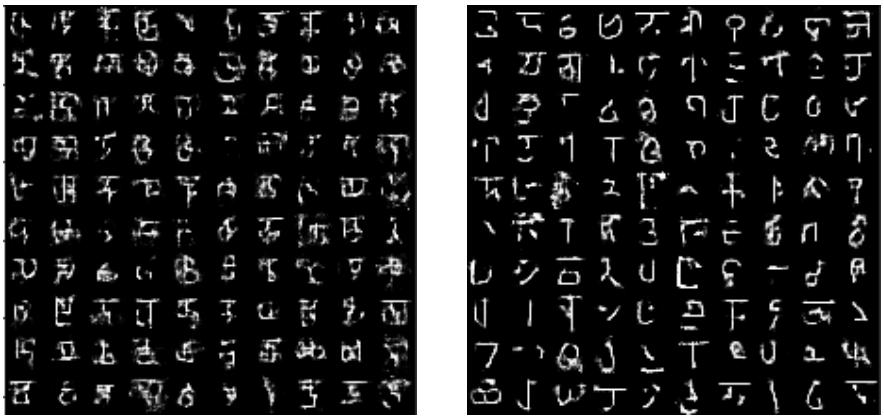
Figure 4: **Omniglot Markov chain samples:** samples synthesised from a DAAE, trained on the Omniglot dataset, using a Markov chain. The chain is initialised with $z^{(0)} \sim \mathcal{N}(0, 1)$.

Table 7: **Omniglot log-likelihood of $p_\theta(x)$: using evaluation dataset.** To calculate the log-likelihood of $p_\theta(x)$, a Parzen window was fit to 10^3 generated samples and the mean likelihood was reported for the Omniglot evaluation data set. The bandwidth used for the Parzen window was determined using a test set. The training and evaluation datasets have samples from **different** handwritten character classes. All models were trained using a 200D Gaussian prior.

Model & Training			Log-likelihood		
Model	Corruption	M	$x^{(0)}$	$x^{(1)}$	$x^{(5)}$
AAE	0.0	-	-823±15	-	-
DAAE	0.25	-	-858±15	-722±15	-762±15
iDAAE	0.25	5	-631±14	-	-

Table 8: **Omniglot log-likelihood of $p_\theta(x)$: using testing dataset.** To calculate the log-likelihood of $p_\theta(x)$, a Parzen window was fit to 10^3 generated samples and the mean likelihood was reported for the Omniglot test data set. The bandwidth used for the Parzen window was determined by fitting the window to a separate set of generated samples and taking measurements on the Omniglot test set. The training and testing datasets have samples from the **same** handwritten character classes. All models were trained using a 200D Gaussian prior.

Model & Training			Log-likelihood		
Model	Corruption	M	$x^{(0)}$	$x^{(1)}$	$x^{(5)}$
AAE	0.0	-	-572±55	-	-
DAAE	0.25	-	-677±54	-530±54	-487±57
iDAAE	0.25	5	-413±52	-	-



(a) $x^{(0)}$ (b) $x^{(9)}$

Figure 5: **Omniglot Markov Chain sampling:** (a) Initial sample, $x^{(0)}$ and (b) *Corresponding* samples, $x^{(9)}$ after 9 iterations of Markov chain sampling. The chain was initialised with $z^{(0)} \sim \mathcal{N}(0, 1)$.

if samples of the latent variable, z_0, are drawn from a distribution that is very different to the prior, Markov chain sampling improves synthesis.

Finally, evaluating generated samples is challenging: log-likelihood is not always reliable [22], and qualitative analysis is subjective. For this reason, we provided both quantitative and qualitative results to communicate the benefits of introducing Markov chain sampling when a trained DAAE, and the advantages of iDAAEs over AAEs. Examples of synthesised samples for both iDAAEs and DAAEs on each dataset are shown in Figure 6 in the Appendix.

7.6 Classification

The previous section focused on the quality of generated samples; here, we focus on the quality of the encoding. During training, the encoder may learn to map visually similar samples in image space to an encoding space in which classes of the input data can be separated. The classes may be linearly separable in latent space, or separable after applying a simple kernel operation, for example a Radial Basis Function (RBF) kernel. To evaluate the separability of the latent space, we encode both a training and testing (or evaluation) dataset. An SVM with an RBF kernel is trained on top of the encoded training samples. The encoder is evaluated by using the trained classifier on encoded test (or evaluation) samples. We report the accuracy of the classifier on the encoded test samples as the percentage of correctly classified samples. Models that learn encodings which better separate the classes in latent space will have a higher accuracy. We expect both the encoder of the DAAE and iDAAE models to outperform the AAE models on all datasets. We also compare classification scores using the learned encoders to those achieved using PCA on image samples, for

Table 9: **MNIST Classification:** SVM classifiers with RBF kernels were trained on encoded MNIST training data samples. The samples were encoded using the encoder of the trained AAE, DAAE or iDAAE models. Classification scores are given for the MNIST test dataset.

Model & Training				Accuracy
Model	Corruption	Prior	M	%
AAE	0.0	10D Gaussian	-	95.19%
DAAE	0.5	10D Gaussian	-	95.28%
DAAE	0.5	2D 10-GMM	-	73.81%
iDAAE	0.5	10D Gaussian	5	96.48%
iDAAE	0.5	10D Gaussian	25	96.24%
PCA + SVM				94.73%

the same number of dimensions as the hidden representation. This is used to see whether the learned encoders produce encodings with better class separability than PCA.

7.6.1 Classification: MNIST

The MNIST dataset consists of 10 classes, $[0, 9]$, the classification task involves correctly predicting a label in this interval. For the MNIST dataset, the SVM classifier is trained on encoded samples from the MNIST training dataset and evaluated on encoded samples from the MNIST testing dataset, results are shown in Table 9.

First, we consider the results for DAAE, iDAAE and AAE models trained with a 10D Gaussian prior. Classifiers trained on encodings extracted from the encoders of trained DAAE, iDAAE or AAE models out-performed classifiers trained on PCA of image samples. Classifiers trained on the encodings extracted from the encoders of learned DAAE and iDAAE models out-performed those trained on the encodings extracted from the encoders of the AAE model.

The differences in classification score for each model on the MNIST dataset are small; this might be because it is relatively easy to classify MNIST digits with very high accuracy [20]. We turn now to a more complicated dataset, the Omniglot dataset, to better show the benefits of using denoising when training adversarial autoencoders.

7.6.2 Classification: Omniglot

Classifying samples in the Omniglot dataset is very challenging: the training and testing datasets consists of 946 classes, with only 19 examples of each class in the training dataset. The 946 classes make up 30 writing systems, where symbols between writing systems may be visually indistinguishable. Previous work has focused on only classifying 5, 15 or 20 classes from within a single

writing system [19, 25, 5, 11], however we attempt to perform classification across all 946 classes. The Omniglot training dataset is used to train SVMs on encodings extracted from encoding models of the trained DAAE, iDAAE and AAE models. Classification scores are reported on the Omniglot evaluation dataset, (Table 10).

Results show that the DAAE and iDAAE out-perform the AAE on the classification task. The DAAE and iDAAE also out-perform a classifier trained on encodings obtained by applying PCA to the image samples, while the AAE does not, further showing the benefits of using denoising.

We perform a separate classification task using only 20 classes from the Omniglot evaluation dataset (each class has 20 examples). This second test is performed for two key reasons: a) to study how well autoencoders trained on only a subset of classes can generalise as feature extractors for classifiers of classes not seen during *autoencoder* training; b) to facilitate performance comparisons with previous work [5]. Features are learned from the Omniglot training dataset, classification is performed on 20 classes from the Omniglot evaluation dataset by splitting samples into two further subsets consisting of 19 training and 1 testing example per character. We extract features for the all examples of the 20 classes from evaluation dataset, using the encoders from AAE, DAAE and iDAAE models trained on the Omniglot training dataset. A linear SVM classifier is trained on the 19 samples from each of the 20 classes in the evaluation dataset and tested on the remaining 1 sample from each class. We perform the classification 20 times, leaving out a different sample from each class, in each experiment. The results are shown in Table 11. For comparison, we also show classification scores when PCA is used as a feature extractor instead of a learned encoder.

Results show that the the DAAE model out-performs the AAE model, while the iDAAE performs less well, suggesting that features learned by the DAAE transfer better to new tasks, than those learned by the iDAAE. The AAE, iDAAE and DAAE models also out-perform PCA.

From this section, we may conclude that introducing the denoising criterion to an AAE, is useful for learning encodings that are more separable, contributing to better classification scores.

7.7 Trade-offs in Performance

The results presented in this section suggest that both the DAAE and iDAAE out-perform AAE models on most generation and classification tasks and strongly suggest it is beneficial to incorporate denoising into the training of adversarial autoencoders. However, it is less clear which of the two new models, DAAE or iDAAE, are better for classification and sample synthesis. When evaluating which one to use, we must consider both the practicalities of training, and for generative purposes, the practicalities – primarily computational load – of each model.

For most classification tasks, the iDAAE out-performs the DAAE. However, the integrating steps required for training an iDAAE means that it takes longer

Table 10: **Omniglot Classification on all** 964 **Test Set Classes.** SVM classifiers with RBF kernels were trained on encoded Omniglot training data samples. The samples were encoded using the encoder of the trained AAE, DAAE or iDAAE models. The classification scores are reported for the Omniglot testing dataset.

Model & Training				Accuracy
Model	Corruption	Prior	M	%
AAE	0.0	200D Gaussian	-	18.36%
DAAE	0.5	200D Gaussian	-	31.74%
iDAAE	0.5	200D Gaussian	5	34.02%
PCA + SVM				31.02%
Random chance				0.11%

Table 11: **Omniglot Classification On 20 Evaluation Classes:** Classifier is trained using 20 classes from the Omniglot evaluation dataset. SVMs are trained on 19 evaluation samples from each of the 20 classes, encoded using the encoder of a AAE, DAAE or iDAAE trained on the Omniglot training dataset. Testing is performed on the remaining 1 encoded sample from each class. This is repeated for all 20 perturbations where a different samples is left out for testing. Samples in the training and evaluation datasets are from **different** handwritten character classes.

Model & Training				Accuracy
Model	Corruption	Prior	M	%
AAE	0.5	200D Gaussian	-	78.75%
DAAE	0.0	200D Gaussian	-	83.00%
iDAAE	0.5	200D Gaussian	5	78.25%
PCA + SVM				76.75%
Random chance				5%

to train than a DAAE. Once the model is trained, however, the time taken to compute encodings for classification is the same for both models. Further, results suggested that using as few as $M = 5$ integrating steps during training, leads to an improvement in classification score. This means that for some classification tasks, it may be worthwhile to train an iDAAE rather than a DAAE.

For generative tasks, neither the DAAE nor the iDAAE model consistently out-perform the other in terms of log-likelihood of synthesised samples. The choice of model may be more strongly affected by the computational effort required during training or sampling. In terms of log-likelihood on the synthesised samples, an iDAAE using even a small number of integration steps ($M = 5$) during training leads to better quality images being generated, and similarly using even one step of sampling leads to better generations.

Conflicting log-likelihood values of generated samples between testing and evaluation datasets means that these measurements are not a clear indication of how the number of sampling iterations affects the visual quality of samples synthesised using a DAAE. In some cases it may be necessary to visually inspect samples in order to assess effects of greater or smaller numbers of sampling iterations (Figure 4).

8 Conclusion

Variational autoencoders [9], VAEs maximise a variational lower bound in order to model distributions, which involves shaping a posterior distribution to match a prior. However, VAEs [9] rely on the existence of an analytical solution for the KL divergence between the prior and posterior, limiting the choices of distributions. If a denoising criterion is introduced to a VAE, there is often no analytical solution for the KL divergence [8].

Another approach to training autoencoders, which does not require the KL divergence to be calculated analytically, is to use adversarial training [6]. Adversarial training involves training a pair of competing models, a generator and a discriminator. The generator, is trained to generate 'fake' samples consistent with a target distribution, while the discriminator is trained to correctly distinguish 'real' samples from 'fake' ones. This adversarial training process can be applied in a latent space [13], where the target is a desired prior distribution over latent space.

We propose two types of denoising autoencoders, where a posterior is shaped to match a prior using adversarial training. In the first, we match the posterior conditional on corrupted data samples to the prior; we call this model a DAAE. In the second, we match the posterior, conditional on original data samples, to the prior. We call the second model an integrating DAAE, or iDAAE, because the approach involves using Monte Carlo integration during training.

Our first contribution is the extension of adversarial autoencoders (AAEs) to denoising adversarial autoencoders (DAAEs and iDAAEs). Our second contribution includes identifying and addressing challenges related to synthesising data samples using DAAE models. We propose synthesising data samples by

iteratively a DAAE according to a Markov chain transition operator, defined by the learned encoder and decoder of the DAAE model, and the corruption process used during training.

Finally, we present results on three datasets, for three tasks that compare both DAAE and iDAAE to AAE models. The datasets include: handwritten digits (MNIST [12]), handwritten characters (Omniglot [11]) and a collection of human-like sprite characters (Sprites [17]). The tasks are reconstruction, classification and sample synthesis.

9 Future Work

In addition to applying the principles of denoising autoencoders and improvements in sampling to different types of data and problems, there are many other interesting ways in which DAAE and iDAAE models themselves may be extended. These include: 1) exploring different priors, and designing priors suitable for a specific task; 2) using convolutional neural networks, rather than fully connected networks, for the encoder, decoder and discriminator; 3) using f-GANs [15] which minimise the KL divergence rather than the Jensen-Shannon divergence to more accurately maximise the variational lower bound of the log-likelihood of $p(x)$; 4) using labelled training data to learn a conditional DAAE and iDAAE in order to generate samples of specific classes; 5) applying the Markov chain sampling method, introduced here, to denoising variational autoencoders [8].

Acknowledgments

We acknowledge the Engineering and Physical Sciences Research Council for funding through a Doctoral Training studentship. We would also like to thank Kai Arulkumaran for interesting discussions and managing the cluster on which many experiments were performed.

References

[1] Philip Bachman and Doina Precup. "Variational Generative Stochastic Networks with Collaborative Shaping". In: *Proceedings of the 32nd International Conference on Machine Learning*. 2015, pp. 1964–1972.

[2] Yoshua Bengio. "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[3] Yoshua Bengio et al. "Deep generative stochastic networks trainable by backprop". In: *Journal of Machine Learning Research: Proceedings of the 31st International Conference on Machine Learning*. Vol. 32. 2014.

[4] Yoshua Bengio et al. "Generalized denoising auto-encoders as generative models". In: *Advances in Neural Information Processing Systems*. 2013, pp. 899–907.

[5] Harrison Edwards and Amos Storkey. "Towards a Neural Statistician". In: *arXiv preprint arXiv:1606.02185* (2016).

[6] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. 2014, pp. 2672–2680.

[7] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *science* 313.5786 (2006), pp. 504–507.

[8] Daniel Jiwoong Im et al. "Denoising criterion for variational auto-encoding framework". In: *arXiv preprint arXiv:1511.06406* (2015).

[9] Diederik P Kingma and Max Welling. "Auto-encoding variational Bayes". In: *Proceedings of the 2015 International Conference on Learning Representations (ICLR-2015), arXiv preprint arXiv:1312.6114*. 2014. URL: https://arxiv.org/abs/1312.6114.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[11] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. "Human-level concept learning through probabilistic program induction". In: *Science* 350.6266 (2015), pp. 1332–1338.

[12] Yann LeCun, Corinna Cortes, and Christopher JC Burges. *The MNIST database of handwritten digits*. 1998.

[13] Alireza Makhzani et al. "Adversarial autoencoders". In: *arXiv preprint arXiv:1511.05644* (2015).

[14] Anh Nguyen et al. "Plug & Play Generative Networks: Conditional Iterative Generation of Images in Latent Space". In: *arXiv preprint arXiv:1612.00005* (2016).

[15] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. "f-GAN: Training generative neural samplers using variational divergence minimization". In: *Advances in Neural Information Processing Systems*. 2016, pp. 271–279.

[16] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *International Conference on Learning Representations (ICLR) 2016, arXiv preprint arXiv:1511.06434*. 2015. URL: https://arxiv.org/pdf/1511.06434.pdf.

[17] Scott E Reed et al. "Deep visual analogy-making". In: *Advances in Neural Information Processing Systems*. 2015, pp. 1252–1260.

[18] Tim Salimans et al. "Improved techniques for training GANs". In: *arXiv preprint arXiv:1606.03498* (2016).

[19] Adam Santoro et al. "One-shot learning with memory-augmented neural networks". In: *arXiv preprint arXiv:1605.06065* (2016).

[20] Patrice Y Simard, David Steinkraus, John C Platt, et al. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis." In: *ICDAR*. Vol. 3. Citeseer. 2003, pp. 958–962.

[21] Helsinki University of Technology. *Markov Chains and Stochastic Sampling*. [Chapter 1: pg5: Theorem (Markov Chain Convergence)]. URL: `www.tcs.hut.fi/Studies/T-79.250/tekstit/lecnotes_01.pdf`.

[22] Lucas Theis, Aäron van den Oord, and Matthias Bethge. "A note on the evaluation of generative models". In: *Proceedings of the International Conference of Learning Representations, arXiv preprint arXiv:1511.01844*. 2015. URL: `https://arxiv.org/pdf/1511.01844v3.pdf`.

[23] Pascal Vincent et al. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. ACM. 2008, pp. 1096–1103.

[24] Pascal Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of Machine Learning Research* 11.Dec (2010), pp. 3371–3408.

[25] Oriol Vinyals et al. "Matching networks for one shot learning". In: *Advances in Neural Information Processing Systems*. 2016, pp. 3630–3638.

10 Appendix

10.1 Proofs

Lemma 10.1. *$\mathcal{P}(z)$ is a stationary distribution for the Markov chain defined by the sampling process in 2.*

Proof. Consider the case where $z^{(0)} \sim \mathcal{P}(z)$. $x^{(0)} \sim p_\theta(x|z^{(0)})$ is from $\mathcal{P}(x)$, by equation 4. Following the sampling process, $\tilde{x}^{(0)} \sim c(\tilde{x}|x^{(0)})$, $z^{(0)} \sim q_\phi(z|\tilde{x}^{(0)})$, $z^{(1)}$ is also from $\mathcal{P}(z)$, by equation 5. *Similar to proof in Bengio [3].* Therefore $\mathcal{P}(z)$ is a stationary distribution of the Markov chain defined by 2. □

Lemma 10.2. *The Markov chain defined by the transition operator, $T_{\theta,\phi}(z_{t+1}|z_t)$ is ergodic, provided that the corruption process is additive Gaussian noise and that adversarial pair, $q_\phi(z|\tilde{x})$ and $d_\chi(z)$ are optimal within the adversarial framework.*

Proof. Consider $X = \{x : \mathcal{P}(x) > 0\}$, $\tilde{X} = \{\tilde{x} : c(\tilde{x}|x) > 0\}$ and $Z = \{z : \mathcal{P}(z) > 0\}$. Where $Z \subseteq \{z : p(z) > 0\}$ and $X \subseteq \tilde{X}$:

1. Assuming that $p_\theta(x|z)$ is a good approximation of the underlying probability distribution, $\mathcal{P}(x|z)$, then $\forall x_j \sim \mathcal{P}(x)\ \exists\ z_i \sim \mathcal{P}(z)$ s.t. $p_\theta(x_j|z_i) > 0$.

2. Assuming that adversarial training has shapped the distribution of $q_\phi(z|\tilde{x})$ to match the prior, $p(z)$, then $\forall z_i \sim p(z)\ \exists\ \tilde{x}_j$ s.t. $q_\phi(z_i|\tilde{x}_j) > 0$. This holds because if not all points in $p(z)$ could be visited, $q_\phi(z|\tilde{x})$ would not have matched the prior.

1) suggests that every point in X may be reached from a point in Z and 2) suggests that every point in Z may be reached from a point in $\tilde{X}$. Under the assumption that $c(\tilde{x}|x)$ is an additive Gaussian corruption process then $\tilde{x}_i$ is likely to lie within a (hyper) spherical region around x_i. If the corruption process is sufficiently large such that (hyper) spheres of nearby x samples overlap, for an x_i and x_{i+m} $\exists$ a set $\{x_{i+1}, ..., x_{i+m-1}\}$ such that, $\text{sup}(c(\tilde{x}|x_i)) \cap \text{sup}(c(\tilde{x}|x_{i+1})) \neq \emptyset, \forall i = 1, ..., (m-1)$ and where sup is the support. Then, it is possible to reach any z_i from any z_j (including the case $j = i$). Therefore, the chain is both irreducible and positive recurrent.

To be ergodic the chain must also be aperiodic: between any two points x_i and x_j, there is a boundary, where x values between x_i and the boundary are mapped to z_i, and points between x_j and the boundary are mapped to z_j. By applying the corruption process to $x^{(t)} = x_i$, followed by the reconstruction process, there are always at least two possible outcomes because we assume that all (hyper) spheres induced by the corruption process overlap with at least one other (hyper) sphere: either $\tilde{x}^{(t)}$ is not pushed over the boundary and $z^{(t+1)} = z_i$ remains the same, or $\tilde{x}^{(t)}$ is pushed over the boundary and $z^{(t+1)} = z_j$ moves to a new state. The probability of either outcome is positive, and so there is always more that one route between two points, thus avoiding periodicity provided that for $x_i \neq x_j$, $z_i \neq z_j$. Consider, the case where for $x_i \neq x_j$, $z_i = z_j$, then it would

not be possible to recover both x_i and x_j using $\mathcal{P}(x|z)$ and so if $\mathcal{P}(x_i|z) > 0$ then $\mathcal{P}(x_j|z) = 0$ (and vice verse), which is a contradiction to 1).

□

Theorem 10.3. *Assuming that $p_\theta(x|z)$ is a sufficient approximation of $\mathcal{P}(x|z)$, and that the adversarial pair – $q_\phi(z|x)$ and $d_\chi(z)$ – are optimal, the transition operator $T_{\theta,\phi}(z^{(t+1)}|z^{(t)})$ defines a Markov chain whose unique stationary distribution is $\mathcal{P}(z) = \int \tilde{q}_\phi(z|x)\mathcal{P}(x)dx$.*

Proof. This follows from Lemmas 10.2 and 10.1. □

10.2 Examples Of Sythesised Samples

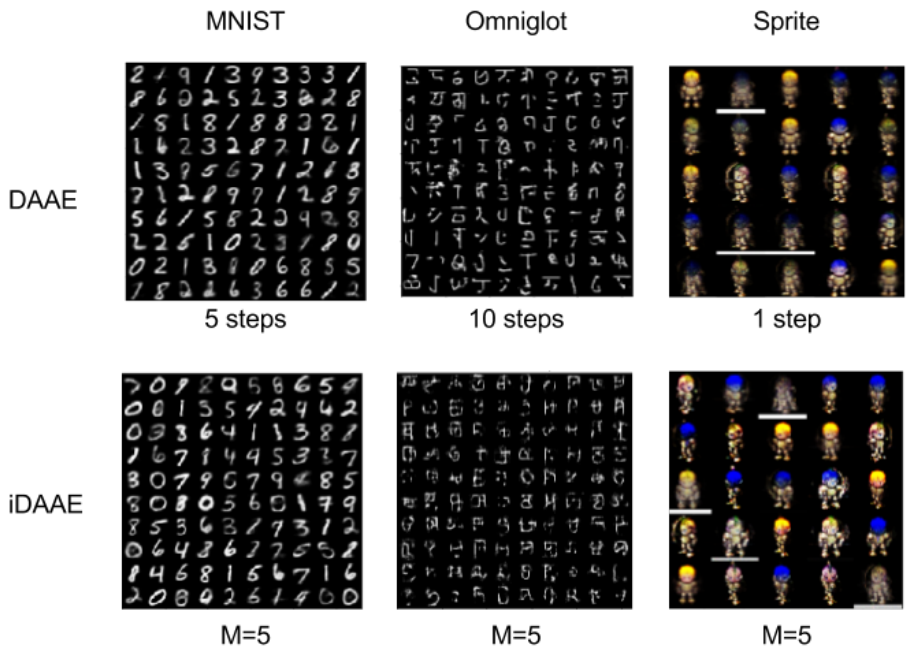


Figure 6: **Examples of synthesised samples:** Examples of randomly sytheisised data samples.

10.3 Algorithm For Monte Carlo Integration

Algorithm 2: Drawing samples from $\tilde{q}_\phi(z|x)$

function: approx_z($\{x_1, x_2, ...x_{N-1}\}$)
for $i = 0$ *to* $N - 1$ **do**
    $\hat{z}_i = [\,]$
    for $j = 1$ *to* M **do**
        $\tilde{x}_{i,j} = C(x_i)$
        $z_{i,j} = E_\phi(\tilde{x}_j)$
    end
    $\hat{z}_i.append(\frac{1}{M}\sum_{j=1}^{M} z_j)$
end
return $\hat{\mathbf{z}} = \{\hat{z}_1, \hat{z}_2 ... \hat{z}_{N-1}\}$