# Deep Learning
# Assignment 3: Deep Generative Models

**Daniel Daza**
University of Amsterdam
daniel.dazacruz@student.uva.nl

## Abstract

In this work we explore three generative models used in deep learning: Variational Autoencoders, Generative Adversarial Networks, and Normalizing Flows. We discuss their theoretical properties and implementation. We run experiments to evaluate the performance of these models on image data, and we discuss their differences qualitatively and from the point of view of generative modeling.

## 1 Variational Autoencoders

### Question 1.1

1. An Autoencoder (AE) and a Variational Autoencoder (VAE) can be used to find a code underlying an observation. Ideally, this code should serve as a compressed representation of the input that contains the right amount of information about it, finding a balance that ignores irrelevant details, and keeps meaningful features. In this sense, the models are not different. However, unlike the AE, the VAE is a generative model that models the probability density of an observation, while solving a problem of statistical inference. The VAE also allows the code to have a probabilistic interpretation. For some applications, this can be part of the main function, making it different from the AE. An example of this use is the Bayesian Skipgram [1], where words are embedded as probability distributions that capture uncertainty in the representation. This allows to represent ambiguous words with distributions of high variance, and viceversa; and to compute similarities between words via the KL divergence. This would not be possible with the AE, where the code is deterministic and does not have the ability to represent uncertainty.
2. The AE is not generative, because it does not model the data-generating distribution, either explicitly or implicitly. Instead, it finds a deterministic map $\mathbf{z} = f(\mathbf{x})$ for the encoder, and $\mathbf{x} = g(\mathbf{z})$ for the decoder. These functions do not have a probabilistic interpretation, therefore they do not allow for the calculation of likelihoods or expectations.
3. Both the AE and the VAE are trained in an unsupervised way, and the training procedure is based on the reconstruction of the inputs. For this reason, a VAE can be used in place of a standard AE, provided that the additional term for the KL divergence between the approximate posterior and the prior is added to the loss function. For downstream tasks, such as classification, the mean parameterized by the encoder of the VAE can be used in place of the deterministic code provided by the AE.
4. The aspect that enables the VAE to be generative is the ability to define the probability distribution of the latent variable $p(\mathbf{z})$, and the distribution $p(\mathbf{x}|\mathbf{z})$ of the observation conditioned on the latent variable. With these components, the VAE learns the joint distribution $p(\mathbf{x}, \mathbf{z})$, making it a generative model. A standard AE is only concerned with finding a map to copy the input to the output, and in general it does not have any incentive to find a code that generalizes to unseen observations.

Assignment 3, Deep Learning, University of Amsterdam, 2019.

**Question 1.2**

According to the graphical model described by the VAE, to obtain a sample of $\mathbf{x}$ we can use ancestral sampling, that is, first we sample the root of the directed graphical model, which in this case corresponds to the latent variable $\mathbf{z}$, and then we use this sample to obtain a sample of its children, corresponding to $\mathbf{x}$. Since we are assuming the pixels to be independent conditioned on $\mathbf{z}$, we only need $\mathbf{z}^{(m)}$ (the $m$-th element of $\mathbf{z}$) to sample $\mathbf{x}^{(m)}$. The sample procedure is then the following:

$$\begin{aligned} \mathbf{z} &\sim \mathcal{N}(0, \mathbf{I}_D) \\ \mathbf{x}^{(m)} &\sim \text{Bern}(f_\theta(\mathbf{z})^{(m)}) \; \forall n = 1, \ldots, M \end{aligned}$$

**Question 1.3**

The standard normal prior of the VAE is not a restrictive assumption because samples from it are transformed by the decoding network, $f_\theta(\cdot)$, which can be given arbitrary capacity. We can specify a decoder with a large number of parameters and multiple layers so that it represents a nonlinear transformation representing a map from the simple standard prior, to a sufficiently complicated distribution.

**Question 1.4**

**(a)** We can approximate the expectation with a Monte Carlo estimate:

$$\begin{aligned} \log \int p(\mathbf{x}_n|\mathbf{z}_n)p(\mathbf{z}_n)d\mathbf{z}_n &= \log \mathbb{E}_{p(\mathbf{z}_n)}[p(\mathbf{x}_n|\mathbf{z}_n)] \\ &\approx \frac{1}{L}\sum_{i=1}^{L} p(\mathbf{x}_n|\mathbf{z}_i) \end{aligned}$$

with $\mathbf{z}_i \sim \mathcal{N}(0, \mathbf{I}_D)$.

**(b)** Intuitively, the previous procedure estimates $\log p(\mathbf{x}_n)$ by obtaining a sample of $\mathbf{z}_n$ from the prior, and using it to calculate the probability that the model assigns to $\mathbf{x}_n$. We expect to obtain a good estimate by averaging this probability over multiple sampling steps. The problem with this approach is that values of $\mathbf{z}_n$ where $p(\mathbf{x}_n|\mathbf{z}_n)$ assigns a high probability to an observation can lie in a region much smaller than the broad regions that the prior covers. This means that most of the time we will be obtaining samples of $\mathbf{z}_n$ that assign low probability to $\mathbf{x}_n$. Hence, to obtain a good estimate, we will need many samples. This effects worsens as the dimension of $\mathbf{z}$ increases, making the approach very inefficient.

**Question 1.5**

**(a)** The smallest KL divergence is obtained when the two distributions are the same. Therefore, if $\mu_q = 0$ and $\sigma_q = 1$, $D_{\text{KL}}(q\|p) = 0$. From the definition of the KL divergence, we can see that if for some $x$, $p(x) \to 0$ and $q(x) >> 0$, then $q(x)/p(x) \to \infty$. We can then make $D_{\text{KL}}(q\|p)$ large by letting $\mu \to \infty$ and $\sigma_q = \mu_q$, so that $q(x)$ assigns high probability to regions where $p(x)$ assigns low probability.

**(b)** The KL divergence between a standard prior and a Gaussian distribution with mean $\mu_q$ and variance $\sigma_q^2$ in the univariate case is given by the following expression:

$$D_{\text{KL}}(q\|p) = \frac{1}{2}(\sigma_q^2 + \mu_q^2 - \log(\sigma_q^2) - 1)$$

**Question 1.6**

Since the KL divergence is nonnegative, we can write

$$\begin{aligned} \log p(\mathbf{x}_n) &= \mathbb{E}[\log p(\mathbf{x}_n|Z)] - D_{\text{KL}}(q(Z|\mathbf{x}_n)\|p(Z)) + D_{\text{KL}}(q(Z|\mathbf{x}_n)\|p(Z|\mathbf{x}_n)) \\ &\geq \mathbb{E}[\log p(\mathbf{x}_n|Z)] - D_{\text{KL}}(q(Z|\mathbf{x}_n)\|p(Z)) \end{aligned}$$

Therefore this last expression is a lower bound for the log-probability.

### Question 1.7

If we wanted to optimize the log-probability, we would need to calculate the true posterior distribution $p(\mathbf{z}|\mathbf{x})$:

$$\begin{aligned} p(\mathbf{z}|\mathbf{x}) &= \frac{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}{\int p(\mathbf{z})p(\mathbf{x}|\mathbf{z})d\mathbf{z}} \end{aligned}$$

The integral in the denominator is not tractable, so we don't have direct access to the posterior.

### Question 1.8

When the lower bound is maximized, the log-probability increases and the KL divergence between the approximate and true posteriors decreases. This means that the model assigns a higher probability density to the observations, while improving $q(\mathbf{z}|\mathbf{x})$ as a better approximation of the true posterior $p(\mathbf{z}|\mathbf{x})$. This circumvents the problem of an intractable posterior because if the approximate and true posteriors match, the objective function being maximized gets closer to the log-probability.

### Question 1.9

If we only use one sample of $\mathbf{z}$, then we have the following loss for a data point:

$$-\log p_\theta(\mathbf{x}|Z) + D_{\mathrm{KL}}(q_\phi(Z|\mathbf{x})\|p(Z))$$

To minimize the first term, the model must assign a high density to the observation $\mathbf{z}$ given the latent variable. If the decoder can use a sample of the latent variable and produce a reconstruction close to the observation, this will mean that the observation is likely under the model. For this reason this loss is also known as the reconstruction loss.

To minimize the second term, the KL divergence between the prior and the approximate posterior must be low. This enforces a constraint in the parameters of the encoder, such that they are free to change in order to create a useful mapping from the observation to the latent variable, but not so much as to deviate from a predefined structure determined by the prior distribution. This term thus acts as a regularizer of the parameters of the model.

### Question 1.10

In the following we will omit the subscript in $\mathbf{x}_n$, as we will specify the loss for a single sample.

The first step to calculate the loss is to obtain samples of the latent variable from the encoder, to estimate the reconstruction loss. This step comprises a forward pass through the decoder with the observation $\mathbf{x}$ as an input, and a step where $L$ samples of $\mathbf{z}$ are drawn:

$$\begin{aligned} \boldsymbol{\mu}_e &= \mu_\phi(\mathbf{x}) \\ \boldsymbol{\sigma}_e &= \Sigma_\phi(\mathbf{x}) \\ \mathbf{z}_i &\sim \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_e, \mathrm{diag}(\boldsymbol{\sigma}_e)),\ i = 1, \ldots, L \end{aligned}$$

The samples are then passed through the decoder to obtain the parameters of the Bernoulli distribution:

$$\boldsymbol{\mu}_{di} = f_\theta(\mathbf{z}_i),\ i = 1, \ldots, L$$

We can now estimate the reconstruction loss as follows:

$$\begin{aligned}
\mathcal{L}_n^{\text{recon}} &= -\mathbb{E}_{q_\phi(Z|\mathbf{x})}[\log p_\theta(\mathbf{x}|Z)] \\
&\approx -\frac{1}{L}\sum_{i=1}^{L} \log p_\theta(\mathbf{x}|\mathbf{z}_i) \\
&= -\frac{1}{L}\sum_{i=1}^{L} \log \prod_{m=1}^{M} \text{Bern}(\mathbf{x}^{(m)}|\boldsymbol{\mu}_{di}^{(m)}) \\
&= -\frac{1}{L}\sum_{i=1}^{L}\sum_{m=1}^{M} \mathbf{x}^{(m)} \log \boldsymbol{\mu}_{di}^{(m)} + (1-\mathbf{x}^{(m)}) \log(1-\boldsymbol{\mu}_{di}^{(m)})
\end{aligned}$$

To calculate the regularization term, we can make use of the closed form expression of the KL divergence. Previously we specified it assuming univariate Gaussians, and in the multivariate case we obtain:

$$\begin{aligned}
\mathcal{L}^{\text{reg}} &= D_{\text{KL}}(q_\phi(Z|\mathbf{x})\|p(Z)) \\
&= \frac{1}{2}\sum_{i=1}^{D}((\boldsymbol{\sigma}_e^2)^{(i)} + (\boldsymbol{\mu}_e^2)^{(i)} - \log(\boldsymbol{\sigma}_e^2)^{(i)} - 1)
\end{aligned}$$

**Question 1.11**

The gradient of the loss with respect to the variational parameters $\phi$ is required because it contains the information required to update the parameters of the encoder. The encoder influences the loss through the reconstruction, as samples from the approximate posterior are required to obtain the reconstructions. It also affects the loss through the regularization term. Therefore, if we want to optimize the VAE objective, we need to compute $\nabla\mathcal{L}_\phi$.

The problem with obtaining this gradient is that the steps to calculate the loss involve sampling from a probability distribution parameterized by the encoder, an operation that is not differentiable with respect to the parameters of the encoder. This can be solved with the reparameterization trick, which consists of obtaining a sample from a distribution that is not parameterized by the encoder, and passing it through a deterministic function such that the result is equivalent to sampling from the approximate posterior. This function is chosen so that it is differentiable with respect to $\phi$, so we can calculate the required gradients.

**Question 1.12**

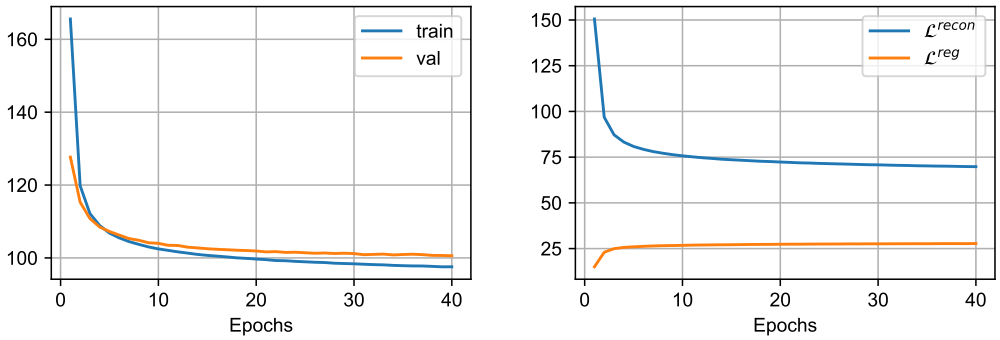
We run experiments on a binary version of the MNIST dataset [2], containing black and white images of size 28×28. We use 50,000 images for training and 10,000 for validation.

The encoder of the VAE is a two layer multilayer perceptron (MLP) with 500 units in the first layer. The second layer produces two separate outputs, corresponding to the mean $\boldsymbol{\mu}_e$ and logarithm of the variance $\log \boldsymbol{\sigma}_e^2$ of the approximate posterior. We choose to calculate $\log \boldsymbol{\sigma}_e^2$ instead of $\boldsymbol{\sigma}_e$ to improve numerical stability. We use 1 sample of the latent variable. For the decoder we use an MLP with two layers, with 500 units in the hidden layer and $28\times28 = 784$ units in the output layer.

We train the model using the Adam optimizer [3] with a learning rate of 0.001, and a batch size of 128.

**Question 1.13**

The mean of the training data with binary images is 0.1306. If we disregard the effect of the KL divergence, we can estimate the reconstruction loss for a randomly initialized model as follows:




(a) (b)

Figure 1: Training curves for the VAE. (a) ELBO for the training and validation sets. (b) Reconstruction loss $\mathcal{L}^{\text{recon}}$ and KL divergence $\mathcal{L}^{\text{reg}}$ on the training set.

$$\sum_{m=1}^{784} \mathbf{x}^{(m)} \log \boldsymbol{\mu}_d^{(m)} + (1 - \mathbf{x}^{(m)}) \log(1 - \boldsymbol{\mu}_d^{(m)}) \approx \sum_{m=1}^{784} 0.1306 \log 0.5 + (1 - 0.1306) \log 0.5$$
$$\approx 543$$

Therefore, the values of the evidence lower bound (ELBO) during training should be below this value.

We show the ELBO for the training and validation set in Figure 1a. As expected, the values are smaller than those of a randomly initialized model. The curves of the ELBO show that the training procedure effectively minimizes the loss in the training and validation sets, indicating that the likelihood function has increased under the generative model.

Previous works have observed that in a failure mode of VAEs, the decoder can learn to ignore the latent variable, especially in cases where a flexible decoder can exploit dependencies in the output [4], such that the KL divergence term of the loss reduces to 0. However, since in our case we are using a model where pixels are independent of each other, this should not be a problem. To verify this, we also show in Figure 1b separately the reconstruction loss and the KL divergence. We observe that at the beginning of training the KL divergence starts at a low value and then increases, showing that the model is driving the approximate posterior away from the prior, which helps to encode meaningful information in the latent variable that the decoder uses to reduce the reconstruction loss.

### Question 1.14

We take samples from the model before training, halfway through and after training. We obtain these by following the generative procedure, that is, first we sample from the prior $p(\mathbf{z})$, and then we pass the sample $\mathbf{z}$ through the decoder to obtain the parameters of the Bernoulli distribution in pixel space, from which we sample the images. The results are shown in Figure 2. We can see that at first the samples are completely random, but as training progresses they become more similar to the images in the training data. Some samples are clearly closer to real digits than others. The reason behind this is the fact the the latent space is continuous, and some samples of $\mathbf{z}$ from the prior might lie on regions that do not correspond to specific digits, but rather to interpolations between digits.

### Question 1.15

In order to visualize the latent space in the regions of significant density in a two-dimensional latent space, we map the square of probabilities $[0.05, 0.95]^2$ to values of $Z$ using the inverse CDF of the standard Gaussian distribution. This results in a grid of latent variables, which we pass through the decoder to obtain the means of the corresponding Bernoulli distributions. The results are shown in Figure 3. We can see that the continuous latent space allows to find a smooth interpolation between the different digits in the dataset.

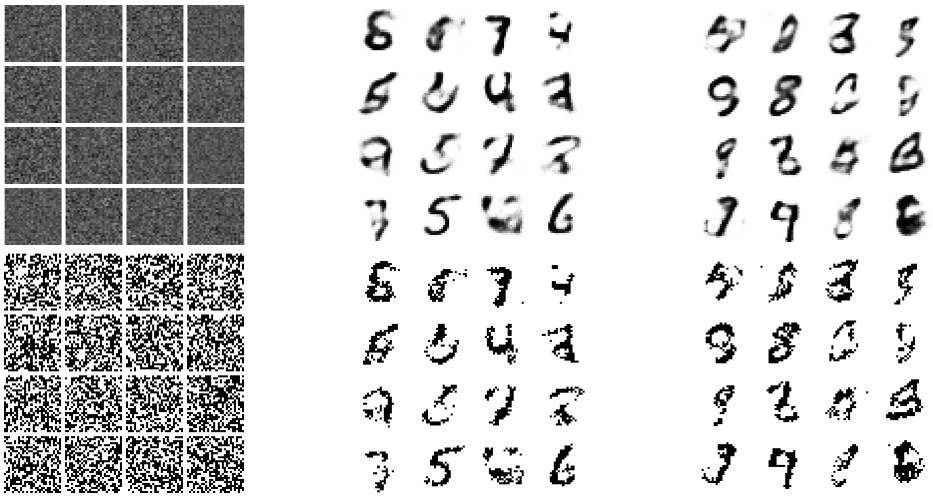


Figure 2: Means (top row) and samples (bottom row) from the VAE, from left to right: before training, halfway through and after training. As training progresses, the samples increasingly resemble samples from the training data.

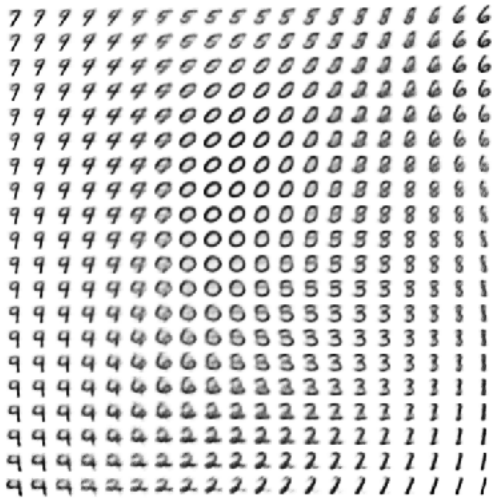


Figure 3: Data manifold learned by the VAE. Values in a grid of a 2-dimensional latent space are mapped by the decoder to the respective means of the Bernoulli distribution.

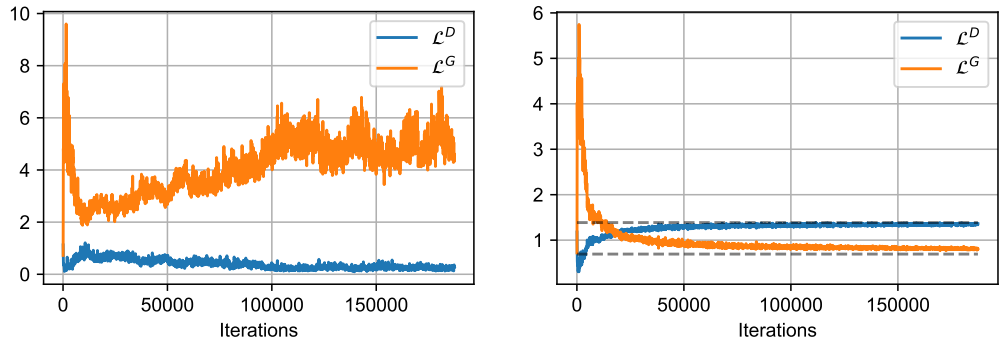
## 2 Generative Adversarial Networks

**Question 2.1**

The input of the generator is a noise sample from a prior distribution. The output is a vector in the data space. For the generator, the input is a vector in the data space, which might come from either the training data, or the generator. The output is a single number representing the probability that the input comes from the training data.

**Question 2.2**

- $\mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\log D(X)]$ is the expected log-probability of a *real* sample that the discriminator assigns to observations from the training data. By maximizing this term with respect to $D$, we make the discriminator to correctly recognize samples from the training term.
- $\mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(Z)))]$ is the expected log-probability of a *fake* sample that the discriminator assigns to observations from the generator network. Minimizing this term with




(a) (b)

Figure 4: Loss curves $\mathcal{L}_D$ for the discriminator, and $\mathcal{L}_G$ for the generator. (a) Using the default GAN loss and no dropout in the discriminator causes fake samples to be easily detected, with the loss of the discriminator decreasing as training progresses. (b) Adding dropout to the discriminator and including one-sided label smoothing improves the dynamics of training, and the final values are closer to the optima (shown with the dashed lines), corresponding to $\log 4$ for the discriminator, and $\log 2$ for the generator.

respect to $D$ allows the discriminator to detect images from the generator that are disparate from samples in the training data.

**Question 2.3**

Assuming that at convergence the generator has matched the data distribution, the discriminator has no better choice than to predict a probability of $\frac{1}{2}$ everywhere [5]. The value of $V(D, G)$ at convergence is thus $\log \frac{1}{2} + \log \frac{1}{2} = -\log 4$.

**Question 2.4**

Early on during training, the generator has not been trained enough to produce samples similar to those in the training data. In this case it is easier for the discriminator to learn to distinguish between real and fake samples, so the term $\log(1 - D(G(Z)))$ will be close to zero no matter how good $G$ is, and too small gradients, if any, will be available to update the parameters of the generator.

To solve this, we can observe the equivalence of the minimax game with the following objective:

$$\max_G \max_D \mathbb{E}_{p_{\text{data}}(\mathbf{x})}[\log D(X)] + \mathbb{E}_{p_{\mathbf{z}}(\mathbf{z})}[\log D(G(Z)] \tag{1}$$

This objective is equivalent because we now find $G$ to maximize the probability that $D$ makes a mistake by identifying a fake sample as real. The advantage from the point of view of gradient-based learning is that the term $\log D(G(Z)$ will tend to $-\infty$ the more $D$ is a good discriminator and $G$ a bad generator, thus providing better gradients for updating $G$.

**Question 2.5**

We implemented a GAN for the MNIST dataset. The latent variable has a dimension of 100, and a standard normal prior. The generator network is comprised by 5 fully-connected layers. All layers except the first use batch-normalization [6]. We use the Leaky ReLU activation with a slope of 0.2, and the $\tanh$ function for the output. We scale all images in the interval $[-1, 1]$ accordingly. The discriminator contains three fully-connected layers with Leaky ReLU activations with a slope of 0.2, and dropout of 0.3.

We use a batch size of 64 and Adam, with a learning rate of 0.0002. We train the discriminator for one step, followed by one training step of the generator. We found one-sided label smoothing beneficial

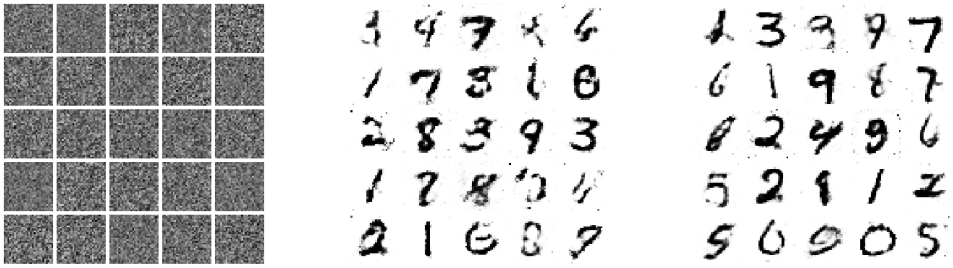


Figure 5: Samples from the GAN, from left to right: before training, halfway through and after training. At the end of training, the samples from the generative model are qualitatively closer to the images in the training data, compared to the samples from the VAE.

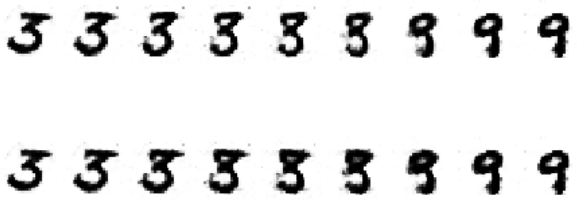


Figure 6: Samples from an interpolation of values in the latent space, using linear (top row) and spherical (bottom) interpolation. This procedure shows that a continuous, linear interpolation in the latent space produces observations that appear to change smoothly.

[7], with values between 0.7 and 1.2, otherwise the generator would collapse to a few distinct samples and the loss of the discriminator would decrease consistently, as shown in Figure 4.

**Question 2.6**

In order to evaluate the generative capabilities of the GAN, we sample images during training, as shown in Figure 5. We do this by randomly sampling the latent variable from a standard normal distribution, and passing the sample through the generator. The samples look smoother than in the VAE, because in the VAE we assumed the images to be binary, whereas the GAN models pixels in the interval $[-1, 1]$.

**Question 2.7**

By fixing the random seed at certain values, we obtain values of the latent variable that the generator maps to a specific digit. This allows us to interpolate between two samples from the generator, by interpolating between the corresponding values of the latent variable. We carry out this procedure using linear interpolation, as well as spherical interpolation across a great circle [8], which in some cases can produce qualitatively better samples [9]. The results are shown in Figure 6. We can see that the GAN enables a continuous interpolation between samples. In our experiments we did not observe a drastic difference between linear and spherical interpolation.

## 3 Generative Normalizing Flows

**Question 3.1**

Given an invertible mapping $f : \mathbb{R}^m \to \mathbb{R}^m$, the distribution $p(\mathbf{x})$, under a change of variable $\mathbf{z} = f(\mathbf{x})$, is given by

$$p(\mathbf{x}) = p(\mathbf{z}) \left| \det \frac{\partial f}{\partial \mathbf{z}} \right|$$

Where $\frac{\partial f}{\partial \mathbf{z}}$ is the $m \times m$ Jacobian matrix containing all pairwise derivatives between the elements of $f(\mathbf{z})$ and $\mathbf{z}$. If we apply this transformation $L$ times, we then obtain the following expression:

$$\log p(\mathbf{x}) = \log p(\mathbf{z}_0) + \sum_{l=1}^{L} \log \left| \det \frac{\partial f_l}{\partial \mathbf{z}_{l-1}} \right| \tag{2}$$

**Question 3.2**

In order to compute equation 2, we must be able to compute the determinant of the Jacobian. This means that the Jacobian has to be a square matrix, otherwise the determinant is not defined. This constrains the mapping $f : \mathbb{R}^m \rightarrow \mathbb{R}^n$ to the case where $m = n$, so that both $\mathbf{x} \in \mathbb{R}^m$ and $\mathbf{z}_i \in \mathbb{R}^m \; \forall i = 0, \ldots, L$.

Furthermore, from the logarithm in equation 2, the determinant can not be zero. From the inverse function theorem, this constrains $f$ to be an invertible map, so that the determinant is nonzero.

**Question 3.3**

Equation 2 involves the calculation of a Jacobian, which in general has a complexity of $O(m^3)$ per transformation. During backpropagation, the gradient of the Jacobian is required and the need for its inverse can make training unstable. This limits the flexibility with which we can choose invertible transformations to those that have a runtime complexity feasible for high dimensional problems, such as linear transformations that have linear complexity and do not require the inverse of the Jacobian [10].

**Question 3.4**

Since we are interested in maximizing the log-probability that the model assigns to the data, having discrete observations might cause the continuous model to accumulate probability mass around discrete values, rather than finding a smoother distribution that resembles the true data distribution. To solve this problem we can dequantize the data so that it takes values across a continuous interval. This can be achieved by adding random noise to a data point. One way to solve this is by adding noise from a standard distribution, as suggested by Theis et al. (2015) [11]. Alternatively, the amount of noise added can be jointly learned as part of the optimization procedure, using a flow model for the noise itself [12].

**Question 3.5**

During training, we are interested in maximizing the log-likelihood. In this case the input is the training data, which is passed through the normalizing flows (NF). This allows to compute the required log-likelihood, using equation 2. As an output we obtain the corresponding value of $\mathbf{z}$.

During inference time (assuming this corresponds to sampling from the model), the procedure is the opposite: the input is a sample from a prior, $\mathbf{z}_0 \sim p(\mathbf{z})$, that is then passed through the inverse of the flow to obtain a generated sample, which in this case is the output.

**Question 3.6**

The steps required to train a density estimation model with normalizing flows are shown in algorithm 1. The algorithm uses stochastic gradient descent (SGD) to minimize the log-likelihood function using mini-batches. At each step the flow is calculated, as well as the log-determinant of the Jacobian.

**Algorithm 1** Density estimation with NFs

**Inputs:** Training data $\mathbf{X}$
Prior $p(\mathbf{z})$
$\boldsymbol{\theta} \leftarrow$ Initialize parameters of $f_1, \ldots, f_L$
**while** not converged **do**
  $\mathbf{x} \leftarrow \{$Get minibatch from $\mathbf{X}\}$
  $\mathbf{h}_0 \leftarrow \mathbf{x}$
  $\mathcal{L} \leftarrow 0$
  **for** $l = 1, \ldots, L$ **do**
    $\mathbf{z}_l \leftarrow f_l(\mathbf{h}_{l-1})$
    $\mathcal{L} = \mathcal{L} + \log \left| \frac{\partial \mathbf{h}_l}{\partial \mathbf{z}_{l-1}} \right|$
  **end for**
  $\mathcal{L} \leftarrow \mathcal{L} + \log p(\mathbf{z}_L)$
  $\mathbf{g} \leftarrow \nabla_\theta \mathcal{L}$
  Update $\boldsymbol{\theta}$ with $\mathbf{g}$
**end while**
**return** $\boldsymbol{\theta}$

**Algorithm 2** Sampling with NFs

**Inputs:** Parameters $\boldsymbol{\theta}$ of $f_1, \ldots f_L$
Prior $p(\mathbf{z})$
$\mathbf{z}_0 \sim p(\mathbf{z})$
**for** $l = 1, \ldots, L$ **do**
  $\mathbf{z}_l \leftarrow f_l^{-1}(\mathbf{z}_{l-1})$
**end for**
**return** $\mathbf{z}_L$

The steps for sampling from the model are shown in algorithm 2. For this procedure we start from a sample from the prior distribution that is passed through the inverse flow.

### Question 3.7

We implemented a density estimation model based on non-volume preserving transformations [13]. The model consists of a series of invertible functions where $d$ elements in the input are preserved, and the rest are transformed. If we denote the inputs of the transformation as $\mathbf{x}$ and the output as $\mathbf{y}$, the transformation is given by the following expressions:

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp\left(s(\mathbf{x}_{1:d})\right) + t(\mathbf{x}_{1:d}) \end{aligned}$$

In this case, the log-determinant of the Jacobian is $\sum_j s(\mathbf{x}_{1:d})_j$.

A single application of this transformation is also known as a *coupling layer*. We implement the translation and scale functions $t(\cdot)$ and $s(\cdot)$ with fully connected neural networks with three hidden layers and ReLU activations. The first two layers are shared between $t$ and $s$. For stability during training, we pass the output of the scale network through the $\tanh$ function before computing the exponential.

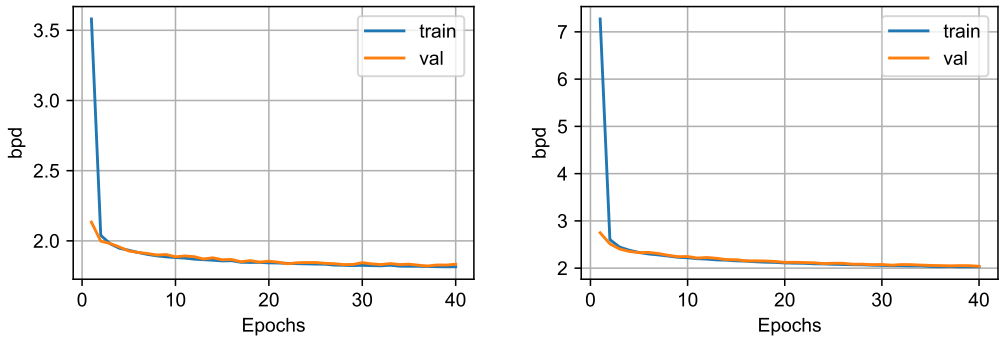
We make use of pairs of coupling layers with complementary binary masks, so that pixels preserved in one coupling layer are modified in the next one. The final model is comprised by 4 pairs of coupling layers.

The prior probability density of the last transformed value in the flow ($p(\mathbf{z}_L)$ in algorithm 1) is a standard normal. We use SGD to minimize the negative log-probability, using the Adam optimizer [3] with a learning rate of 0.001. We additionally clip the norm of the gradients to a maximum of 10.

Given the observation that a learned scale term might not offer a significant advantage compared to using a fixed scale term [14], we additionally run experiments with a constant scale of 1. In this case the flow is volume-preserving since the log-determinant of the Jacobian is zero. This setting has the positive effect of reducing the number of parameters and simplifying the update of the Jacobian, which stays the same.

### Question 3.8

The resulting training curves are shown in Figure 7, where we measure the average bit per dimension (bpd), corresponding to the negative $\log_2$-probability per pixel in an image. We obtain a bpd of 1.82




(a) (b)

Figure 7: Bit per dimension for the density model with normalizing flows. (a) With a learned scale term, the lowest values of bpd are 1.82 and 1.83 in the training and validation sets, respectively. (b) With a fixed scale of 1, the lowest values are 2.03 and 2.04, respectively.

Figure 8: Samples from the density model with normalizing flows, from left to right: before training, halfway through and after training. Top row: learned scale. Bottom row: fixed scale of 1.

and 1.83 in the training and validation sets, respectively, when using a learned scale. For a fixed scale, the bpd is 2.03 and 2.04, respectively. These values are lower than the expected bpd of 8 bits for a random model, given that the modeled images have discrete values between 0 and 255.

The difference in performance between the models with learned and fixed scale is relatively higher than that observed by Kingma et al. (2016) [14]. The training curves also show that the model with a fixed scale starts at a higher loss, which highlights the importance of the flexibility that a learned scale adds to the model for this dataset.

Samples from the models are shown in Figure 8. We observe that halfway through training, several samples are already similar to real digits. We note a slight blurriness in the samples from the model with a fixed scale, which shows that in this case a learned scale produces a model with better generative capability. We can see that samples from the model are sharp, compared to the samples and means from the VAE. On the other hand, samples from the GAN have a better quality, as most of them resemble actual digits.

## 4 Conclusion

We have explored three generative models: VAEs, GANs and Normalizing Flows, each of which with a different approach towards generative modeling. The VAE uses a variational inference framework, therefore incurring in an approximation error as expressed by the gap between the ELBO and the log-likelihood. Among the sources of this error we can identify the prescription of a distribution for

the prior and the approximate posterior, which might not be a suitable choice for generative models of high dimensional data. GANs, on the other hand, are not constrained by the choice of a prior, as samples from it are passed to the generator which transforms it into a sensible output. Unlike VAEs and NFs, GANs as described in this work are generative models with an implicit density that cannot be used to calculate the log-likelihood given an observation (strictly speaking, it is not possible to calculate it with VAEs either, but a lower bound can be obtained instead). The absence of an encoder in GANs also prevents from doing inference where a code for an observation is required, as can be done with a VAE. NFs address some of the limitations in GANs and VAEs by allowing for exact calculation of the log-likelihood, inference of the latent variable from an observation, and the possibility to turn a simple Gaussian prior into a more complex distribution through a series of transformations. Nevertheless, the flexibility of each transformation is constrained to being invertible, and having a Jacobian with a sufficiently low algorithmic complexity. The properties of each model reflect how generative modeling is an active area of research in deep learning, with multiple directions for future work.

## References


[1] Arthur Bražinskas, Serhii Havrylov, and Ivan Titov. Embedding words as distributions with a Bayesian skipgram model. *arXiv preprint arXiv:1711.11027*, 2017.

[2] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*, pages 10–21, 2016.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[7] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[8] Ken Shoemake. Animating rotation with quaternion curves. In *ACM SIGGRAPH computer graphics*, volume 19, pages 245–254. ACM, 1985.

[9] Tom White. Sampling generative networks. *arXiv preprint arXiv:1609.04468*, 2016.

[10] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1530–1538, 2015.

[11] Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[12] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275*, 2019.

[13] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[14] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016.