Deep Learning Assignment 3

Kevin de Vries (10579869)
University of Amsterdam
kevin.devries@student.uva.nl

Question 1

Question 1.1

The VAE is related to pPCA and FA in that they are all latent variable models which typically or exclusively use a Gaussian distributed prior for the latent variables. They are also related in that they require or benefit from variational methods and Bayesian learning.

Question 1.2

We can sample from $p(z_n) = \mathcal{N}(0, I_D)$ simply by sampling n times from a standard normal distribution. In the case of the joint distribution $p(x_n|z_n) = \prod_{m=1}^{M} \text{Bern}(x_n^{(m)}|f_\theta(z_n)_m)$ we can sample from this distribution using ancestral sampling, where for a general joint distribution factorized by $p(x) = \prod_{k=1}^{K} p(x_k|pa_k)$ (with pa_k being the elements which x_k is factorized to be conditional on, usually being $pa_k = \{x_i\}_{i=1}^{k-1}$) we sample sequentially from $p(x_1)$, $p(x_2|x_1)$, ..., $p(x_K|x_1, ..., x_{K-1})$. This then yields a sample from the joint distribution. For $p(x_n|z_n)$ we thus sample sequentially from $\text{Bern}(x_n^{(1)}|f_\theta(z_n)_1)$ up to $\text{Bern}(x_n^{(M)}|\{f_\theta(z_n)_m\}_{m=1}^{M}, \{x_n^{(m)}\}_{m=1}^{M-1}) = \text{Bern}(x_n^{(M)}|f_\theta(z_n)_M)$ by independence as given in the definition of the joint distribution. These individual samples together function as a sample from the joint distribution $p(x_n|z_n)$.

Question 1.3

Carl Doersch's tutorial states that any d-dimensional distribution can be generated using a set of d normally distributed variables given a sufficiently complicated mapping. This mapping can effectively be learned using a powerful function approximator like a neural network, which is used to model the likelihood in Equation 4 of the assignment.

Question 1.4 a)

We can evaluate $\log p(x_n) = \log \mathbb{E}_{p(z_n)}\left[p(x_n|z_n)\right]$ using Monte-Carlo integration. This is done by sampling L samples $z_n^{(l)}$ from $p(z_n)$ and approximating the expected value of $p(x_n|z_n)$ by computing the sample mean $\mathbb{E}_{p(z_n)}\left[p(x_n|z_n)\right] = \frac{1}{L}\sum_{l=1}^{L} p(x_n|z_n^{(l)})$. This yields the expected value $\mathbb{E}_{p(z_n)}\left[p(x_n|z_n)\right]$ needed to compute $\log p(x_n)$.

Question 1.4 b)

Since MC integration requires many samples in order to get a reliable or effective estimate, the sampling procedure for $p(x_n|z_n)$ needs a lot of samples of z_n. Additionally, for many samples from

Preprint. Work in progress.

z_n, $p(x_n|z_n)$ will be close to 0. This can be seen in Figure 2 in the assignment where the contours of the latent space of the posterior are much more localized than those of the prior, which suggests that the likelihood $p(x_n|z_n)$ drops off quickly from the center around which the posterior is localized. This makes it hard to compute an estimate for the evidence $p(x_n)$ which is needed for the predictive distribution.

Question 1.5 a)

$D_{KL}(q||p) = 0$ when both distributions are the same, which means that then $\mu_q = 0$ and $\sigma_q^2 = 1$. $D_{KL}(q||p) \to \infty$ when both distributions become increasingly different, which means that $|\mu_q| \gg 0$ or $\sigma_q^2 \gg 1$ both make the KL-divergence very large.

Question 1.5 b)

The closed form formula for $D_{KL}(q||p)$ with $q = \mathcal{N}(\mu_q, \sigma_q^2)$ and $p = \mathcal{N}(0, 1)$ is given by

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

This formula was found using the link given in the hint of the assignment and adapted for the specific Gaussians used in the assignment.

Question 1.6

We can write Equation (11) from the assignment as

$$\log p(x_n) = \mathbb{E}_{q(z|x_n)} \left[\log p(x_n|Z)\right] - D_{KL}(q(Z|x_n)||p(Z)) + D_{KL}(q(Z|x_n)||p(Z|x_n))$$

which gives the definition of the log-probability of the data. Since $D_{KL}(q(Z|x_n)||p(Z|x_n)) \geq 0$ we can rewrite this as a lower bound, which results in

$$\log p(x_n) \geq \mathbb{E}_{q(z|x_n)} \left[\log p(x_n|Z)\right] - D_{KL}(q(Z|x_n)||p(Z))$$

The right hand side of this inequality of the log-probability of the data is the same as the right hand side of Equation (11) of the assignment, which justifies why it is called the lower bound on the log-probability of the data.

Question 1.7

We have to minimize the lower bound, because $D_{KL}(q(Z|x_n)||p(Z|x_n))$ is not directly-computable, which means that we cannot easily define the loss using that quantity.

Question 1.8

If the lower bound is pushed up, then $\log p(x_n)$ is being maximized or $D_{KL}(q(Z|x_n)||p(Z|x_n))$ is being minimized. For the first possibility the optimized probability of the data becomes a better approximation of the distribution of the data $p(x_n)$. For the second possibility the KL-divergence of $q(Z|x_n)$ and $p(Z|x_n)$ gets closer to zero, which means that $q(Z|x_n)$ becomes a better approximation of $p(Z|x_n)$. This makes the intractable $p(Z|x_n)$ tractable through $q(Z|x_n)$.

Question 1.9

For the reconstruction loss, the name reconstruction is appropriate, because where $q(Z|x_n)$ in the regularization loss "encodes" the data into the latent space, $p(x_n|Z)$ in the reconstruction loss "decodes" a sample from the latent space back into the data, which effectively reconstructs the data. Thus minimizing $\mathcal{L}_n^{recon}$ in turn maximizes the accuracy of the reconstruction of the data and with that the quality of the generated data samples. For the regularization loss, the name regularization is appropriate, because the minimization of the KL-divergence tries to keep the approximate distribution $q_\phi(Z|x_n)$ as similar as possible to the distribution of the latent variables $p_\theta(Z)$ during optimization. This can help prevent the model from overfitting by keeping $q_\phi(Z|x_n)$ as similar to a standard normal distribution as possible.

Question 1.10

$$\begin{aligned}
\mathcal{L}_n^{reg}(\theta,\phi) &= D_{KL}(\mathcal{N}(z_n|\mu_\phi(x_n), \text{diag}(\Sigma_\phi(x_n)))||\mathcal{N}(z_n|0, I_D)) \\
&= \frac{\text{tr}(\text{diag}(\Sigma_\phi(x_n))) + (\mu_\phi(x_n))^T(\mu_\phi(x_n)) - \log|\text{diag}(\Sigma_\phi(x_n))| - D}{2} \\
&= \frac{1}{2}\sum_{d=1}^{D} \Sigma_\phi(x_n)_d + (\mu_\phi(x_n)_d)^2 - \log \Sigma_\phi(x_n)_d - 1
\end{aligned} \tag{1}$$

$$\begin{aligned}
\mathcal{L}_n^{recon}(\theta,\phi) &= -\mathbb{E}_{z_n\sim q_\phi(z_n|x_n)}\left[\log p_\theta(x_n|z_n)\right] \\
&= -\mathbb{E}_{z_n\sim q_\phi(z_n|x_n)}\left[\log \prod_{m=1}^{M} \text{Bern}(x_n^{(m)}|f_\theta(z_n)_m)\right] \\
&= -\sum_{m=1}^{M} \mathbb{E}_{z_n\sim q_\phi(z_n|x_n)}\left[x_n^{(m)} \log f_\theta(z_n)_m + (1-x_n^{(m)})\log(1-f_\theta(z_n)_m)\right]
\end{aligned} \tag{2}$$

Question 1.11 a)

We need $\nabla_\phi \mathcal{L}$, because the variational parameters need to be learned in order to train the encoder $q_\phi(z_n|x_n)$ to be able to encode the data into the latent variables.

Question 1.11 b)

Since we sample from $q_\phi(z_n|x_n)$ directly, the gradients will have to propagate through the sampling procedure. Since the sampling procedure is not differentiable, we will not be able to produce gradients. This prevents the encoder $q_\phi(z_n|x_n)$ from training, since $\nabla_\phi \mathcal{L}$ will be equal to zero.

Question 1.11 c)

The reparametrization trick makes use of an alternative method to sample Gaussian random variables where, instead of sampling from $z_n \sim q_\phi(z_n|x_n) = \mathcal{N}(\mu_\phi(x_n), \text{diag}(\Sigma_\phi(x_n)))$ directly, we sample from a standard normal distribution $\epsilon \sim \mathcal{N}(0, I_D)$ and transform this using $z_n = \mu_\phi(x_n) + \sqrt{\Sigma_\phi(x_n)} \odot \epsilon$ to obtain the samples from $q_\phi(z_n|x_n)$. In this case the square root in $\sqrt{\Sigma_\phi(x_n)}$ is applied to each element of $\Sigma_\phi(x_n)$. This way, the variational parameters are independent from the sampling procedure. Since the transformation is differentiable with respect to the variational parameters, gradients are able to propagate to the encoder in order to train.

Question 1.12

My implementation of the Variational Autoencoder is adapted from the paper "Auto-encoding Variational Bayes" referenced in the assignment. The encoder takes x_n as input and uses a linear module with a tanh activation for the hidden layer, which then uses two separate linear modules to produce $\mu_\phi(x_n)$ and $\log \Sigma_\phi(x_n)$. The samples for the decoder are then computer using $z_n = \mu_\phi(x_n) + \sqrt{\Sigma_\phi(x_n)} \odot \epsilon_n$, with $\epsilon_n \sim \mathcal{N}(0, I)$. The decoder takes the samples z_n and uses a linear module with a tanh activation for its hidden layer. The output is then computed using a linear module and by applying a sigmoid function to get a value between 0 and 1, since this output represents the Bernoulli means of $p(x_n|z_n)$. The output of $f_\theta(z_n)$ and the computed $\mu_\phi(x_n)$ and $\Sigma_\phi(x_n)$ are then used to compute the negative ELBO given by $\mathcal{L}$. For the expectation in $\mathcal{L}_n^{recon}$ only a single sample of z_n is sampled per input sample in the batch.

Question 1.13

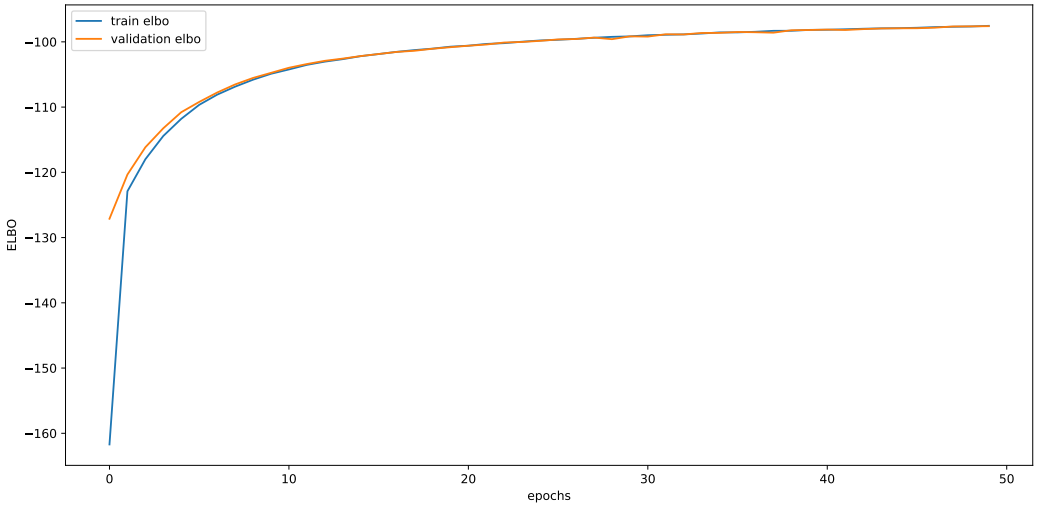
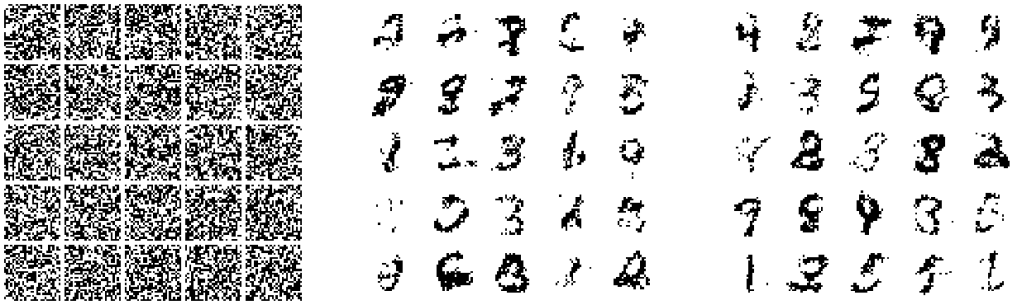


Figure 1: A plot of the average ELBO for each epoch with a 20-dimensional latent space

In Figure 1 the average ELBOs of the training and validation sets are plotted during training. The model was trained for 50 epochs using a 20-dimensional latent space using the default parameters and batch sizes given in the template code. Since the decoder would initially produce $f_\theta(z_n) \approx 0.5$, The reconstruction loss would initially be $\mathcal{L}_n^{recon} \approx M \log 2 = 784 \log 2 > 540$. Assuming that the initial regularization loss $\mathcal{L}_n^{reg}$ is negligible in comparison, the ELBOs of the training and validation sets would initially be $ELBO = -\mathcal{L} < -540$.

Question 1.14



(a) Before training (b) In the middle of training (c) After training

Figure 2: Image samples of the VAE with a 20 dimensional latent space after 0, 25 and 50 epochs respectively

In Figure 2 25 samples from the decoder before training, after 25 epochs and after training are shown. The samples are generated by sampling from standard normally distributed random variables from the prior $p(z_n)$, which are then put into the decoder that outputs the Bernoulli means of the likelihood $p(x_n|z_n)$. This output is then used to sample each pixel in the reconstructed data from a Bernoulli distribution. In Figure 2 we find that the samples are just random noise before training. After 25 epochs we find that the shapes of the digits in the dataset have appeared. After training we see that, although still not perfect, the generated digits seem to resemble the data even more.

Question 1.15

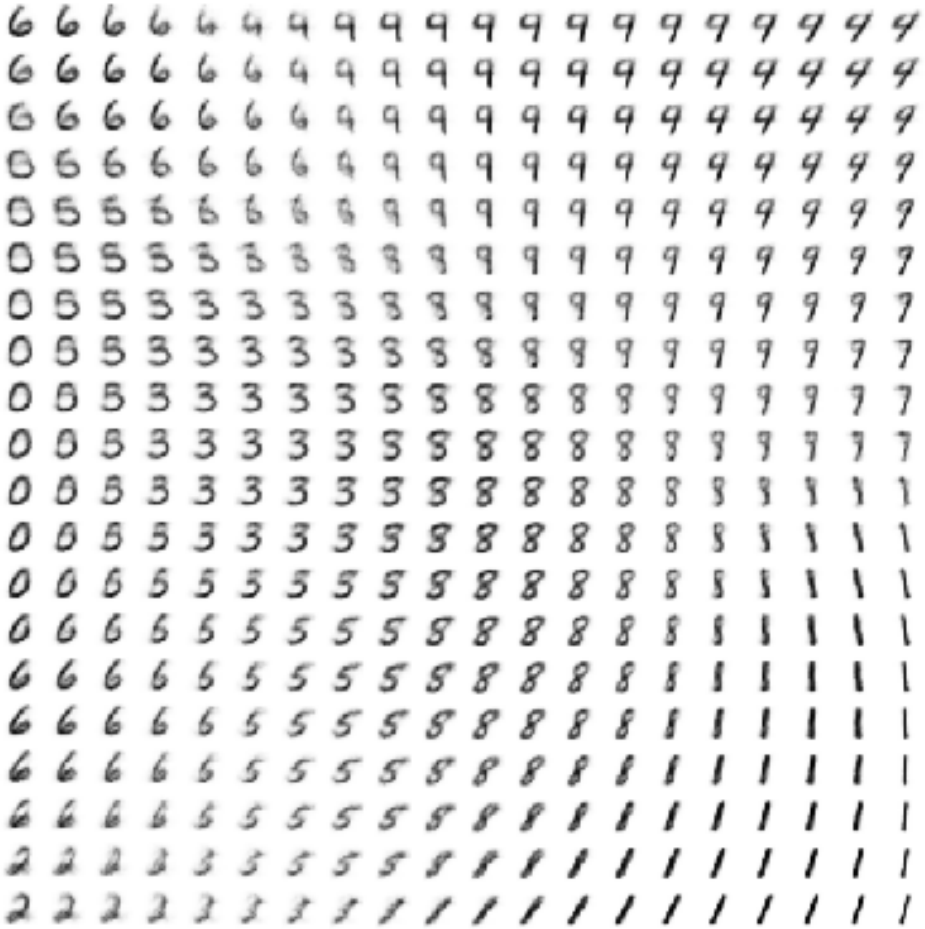


Figure 3: The data manifold for the VAE with a 2-dimensional latent space

In Figure 3 the data manifold for a two dimensional latent space is shown, where the output of the decoder is varied in z-space. This is done by using an evenly spaced grid of z-values between 0.05 and 0.95 and applying a percent point function to get values in the space where the distribution of the latent space is significantly dense. These values are used as input for the decoder. The output of the decoder is then used to plot the data manifold. The data manifold seems to interpolate between the shapes of digits which are most similar to each other. This plot shows that the VAE seems to indeed encode the data into the latent space.

Question 2

Question 2.1

The generator takes some random input noise z (from some distribution) as input and generates a fake (image) sample $x = G(z)$ as output. The discriminator takes a (image) sample x as input and gives a real value between 0 and 1 as output which indicates the probability that the (image) sample is a real sample from the data.

Question 2.2

The first term $\mathbb{E}_{p_{data}(x)}[\log D(X)]$ is the objective of the discriminator to correctly classify the real data samples as real while doing so as confidently as possible and the objective of the generator to make the discriminator less confident about classifying real samples as real and making it mistake real samples for fake samples. The second term $\mathbb{E}_{p_z(z)}[\log(1 - D(G(Z)))]$ is the objective for the generator to generate samples which make $D(G(Z))$ go to one, meaning that its generated fake samples are able to make the discriminator as confident as possible that they are real samples. For the discriminator the second term is the objective to make the discriminator as confident as possible that the fake samples generated by the generator are not real, which means that it tries to make $D(G(Z))$ go to zero.

Question 2.3

Ideally the value of $V(D, G)$ after training has converged will be $V(D, G) = -2 \log 2$, because then $D(X) = \frac{1}{2}$ and $D(G(Z)) = \frac{1}{2}$ (given that $p_{data}(x) = p_{model}(x) = \frac{1}{2}$), which means that the fake samples from the generator are good enough to make the discriminator unable to distinguish them from the real data samples.

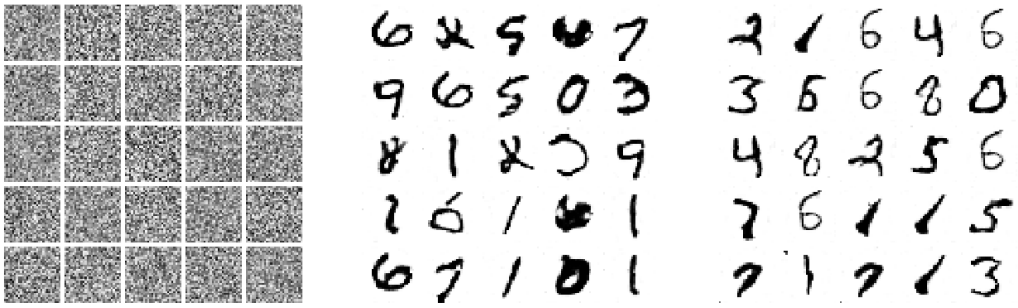
Question 2.4

The term $\log(1 - D(G(Z)))$ can be problematic for training, because the discriminator might start to recognize the fake samples early on while the fake samples are not as good yet, which leads the discriminator to converge prematurely and become a perfect discriminator. If the discriminator is perfect, then the generator gradients vanish, which makes the generator unable to train. This can be solved by not using $V(D, G)$ for the loss of the generator and instead using $\mathbb{E}_{p_z(z)}[\log D(G(Z))]$ for the loss of the generator. This term allows the generator to learn even when all its generated fake samples are recognized by the discriminator, since the gradients for the generator will only increase when $D(G(Z))$ decreases.

Question 2.5

The generator uses three hidden layers. The linear modules including the input and output layers use 128, 256, 512, 1024 and 784 neurons respectively. The input and hidden layers use a leaky ReLU activation with parameter 0.2 and the hidden layers also use batch normalization afterwards. The output layer uses a tanh function to set the pixels of the generated image between -1 and 1. The discriminator is a Convolutional neural network where the first three blocks all use a convolutional layer with 16, 32 and 64 channels respectively, 3x3 filters with stride 1 and padding 1 and leaky ReLU activation with parameter 0.2, followed by batch normalization and an average pooling with 3x3 filters with stride 2 and padding 1. The last hidden layer uses the same convolutional layer with 128 channels and batch normalization, but the average pooling uses 4x4 filters with stride 1 and no padding. The output layer is a linear layer with one unit with a sigmoid function.

Question 2.6



(a) Before training (b) In the middle of training (c) After training

Figure 4: Image samples of the GAN with a 100-dimensional latent space after 0, 93500 and 187500 batches respectively

In Figure 4 25 samples from the generator are shown from before training, after 93500 batches (≈ 100 epochs) and after training (after 187500 batches ≈ 200 epochs). The model was trained using a 100 dimensional latent space and using the Adam optimizer with default parameters. The networks were trained sequentially for every batch. We see that at the start of training the generator begins with generating random noise. In the middle of training, after 100 epochs, the samples are already looking like real digits, though with some quirks which make them somewhat unconvincing overall. After training, which is after 200 epochs, the digits seem to look more qualitatively similar to real digits, since the samples seem to be awkward representations of the digits at worst. The same digits do start to look alike, though. This hints at some loss of diversity of the image samples, meaning that the generator may have overfitted somewhat at this stage of the training. The images were also sampled during training, meaning that the images were generated using batch normalization over the training batch.

Question 2.7

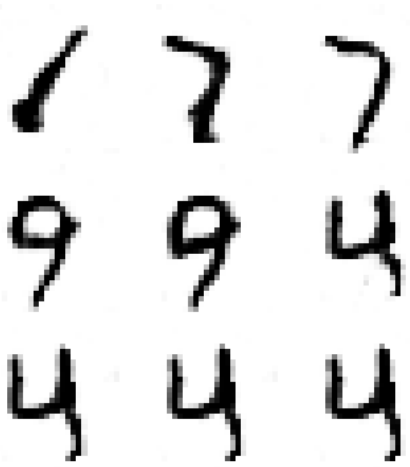


Figure 5: Images from a linear interpolation between two samples from a 100-dimensional latent space

In Figure 5 a few images generated from an interpolation between two random samples from the latent space (sampled from a standard Gaussian distribution) are shown. The interpolation is done using $\hat{z}_n^{(i)} = \alpha_i z_n^{(1)} + (1 - \alpha_i) z_n^{(2)}$ where $z^{(1)}$ and $z^{(2)}$ are the samples from the latent space, $\hat{z}^{(i)}$

is the i-th interpolation of the samples and α is a vector with evenly spaced and ordered values between and including 0 and 1. The generator is set in evaluation mode, which means that the batch normalization is turned off and hence the generated images are not correlated in contrast with the image samples in Figure 4. The order of the interpolated images goes from left to right and then from top to bottom. From the interpolated images in Figure 5 we find that the images change form between digits which look most similar to both image samples. In the case of interpolation between the digits 1 and 4 inbetween the interpolation steps the form of the digits changes from 1 to 7 to 9 and then finally to 4. This is similar to how the VAE manifold in Figure 3 changes shape between digits in a 2 dimensional latent space.

Question 3

Question 3.1

The biggest difference between the VAE and the GAN is that the VAE models a distribution to generate the images and the GAN only learns to generate images through feedback from a discriminator network and thus models the distribution implicitly. When comparing the image samples between the GAN and VAE, we find that the GAN has qualitatively better looking images, though it also uses a ConvNet for the discriminator and trains for 200 epochs, whereas the VAE only trains for 50 epochs and only uses MLPs. The VAE may get better samples if trained for longer (until it starts overfitting) or using different, more sophisticated architectures. The VAE also seems to be much more stable when training and may be more useful when there is merit in modelling the distribution of the data explicitly.