Deep Learning Practical 3

Andrei Marius Sili*
Master's in Artificial Intelligence
University van Amsterdam
andrei.sili@student.uva.nl

Part1: Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

VAE, pPCA, and Factor Analysis all fall under the Continuous Latent Variable models. All 3 models assume a Standard Normal distribution of the latent variable

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

and aim to maximise the likelihood of the data as the marginal over the latent space.

$$p(\mathbf{x}) = \int_{\mathbf{Z}} p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})$$

The main difference between VAE and pPCA is in the assumptions that are made about the posterior distribution $p(\mathbf{x} \mid \mathbf{z})$. While probabilistic PCA assumes a linear dependence with isotropic Gaussian noise

$$p(\mathbf{x} \mid \mathbf{z}) \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma\mathbf{I})$$

VAE does not make any such restrictive assumptions, allowing the posterior to take the form of any distribution with parameters given by an arbitrary transformation of the latent variable.

$$p(\mathbf{x} \mid \mathbf{z}) \sim \mathcal{D}(f_\theta(\mathbf{z}))$$

Probabilistic PCA involves a tractable posterior, so it can be optimised via EM or even in closed form Tipping and Bishop [1999], while VAE requires a variational approach via the AEVB algorithm Kingma and Welling [2013].

1.2 Decoder: The Generative Part of the VAE

Question 1.2

Sampling from our model can be done via ancestral sampling:

- sample $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- compute the parameters of the Bernoulli distribution $f_\theta(\mathbf{z})$
- Random sample from the Bernoulli distribution described by the parameters $f_\theta(\mathbf{z})$.

Question 1.3

While the prior $p(\mathbf{z})$ may seem overly restrictive, the key to overcoming this is the decoder network $f_\theta(\mathbf{z})$. Any d-dimensional distribution can be approximated to arbitrary precision by taking a d-dimensional Gaussian random variable and passing it through a powerful enough function. Since

*Student ID: 11659416

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

neural networks are universal estimators given enough capacity, they are the perfect choice. One can think of the decoder as using the first few layers to transform the Standard Normal latent space to a complex one, and the remaining layers to map that space onto a data point, in this case a fully rendered MNIST digit.

Question 1.4

a) Any expectation may be approximated via Monte Carlo Integration as follows:

$$\int_{\mathbf{Z}} p(\mathbf{x} \mid f_\theta(\mathbf{z}))p(\mathbf{z})d\mathbf{z} \approx \frac{1}{M}\sum_{m=1}^{M} p(\mathbf{x} \mid f_\theta(\mathbf{z}_m)), \mathbf{z}_m \sim p(\mathbf{z})$$

b) The problem with Monte Carlo Integration stems from the approximation via random sampling technique. To accurately estimate this integral, we need to cover the region(s) of high density, i.e where $p(\mathbf{z}\,|\mathbf{x})$ is large. However, this region may be very small compared to the latent space, so most $\mathbf{z}$ samples we draw will have a small posterior $p(\mathbf{z}\,|\mathbf{x}) \approx 0$. If we just draw samples from the prior $p(\mathbf{z})$ we would need to draw a large number of samples before our estimate for the integral becomes accurate. Note that although the estimate is not explicitly dependent on the dimension as it has a complexity of $det(\mathbf{\Sigma})^{0.5}/\sqrt{M}$, the determinant of the covariance matrix grows with the number of dimensions even in the most trivial case of a diagonal matrix. This corresponds to the fact that in higher dimensions, the region with significant probability mass will become smaller and smaller relative to the entire latent space.

1.3 Encoder: $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

Question 1.5

a) The Kullback Leibler Divergence measures the dissimilarity between 2 distributions. A small KL-divergence is given by:

$$q \sim \mathcal{N}(0.1, 2) \Rightarrow D_{KL}[q||p] = 0.18$$

A large KL-divergence is given by:

$$q \sim \mathcal{N}(10, 1) \Rightarrow D_{KL}[q||p] = 22.639$$

b) The closed form formula for the Kullback-Leibler Divergence between a Gaussian and the Standard Normal distribution is given by:

$$\begin{aligned} D_{KL}[q||p] &= \int p(x)\frac{\log q(x)}{\log p(x)}dx = \int p(x)\log p(x)dx + \int p(x)\log q(x) \\ &= \log\frac{1}{\sigma_q} + \frac{\sigma_q^2 + \mu_q^2}{2} - \frac{1}{2} \\ &= \frac{1}{2}(-\log\sigma_q^2 + \sigma_q^2 + \mu_q^2 - 1) \end{aligned}$$

Question 1.6

We know that the KL-Divergence is always non-negative. This means that the log-likelihood of $\mathbf{x}_n$ is at least equal to the RHS. This justifies the name *lower bound*.

$$\begin{aligned} D_{KL}[q(\mathbf{z})||p(\mathbf{x}_n))] &\geq 0 \\ -D_{KL}[q(\mathbf{z})||p(\mathbf{x}_n))] &\leq 0 \\ \log(p(\mathbf{x}_n)) - D_{KL}[q(\mathbf{z})||p(\mathbf{x}_n))] &\leq \log(p(\mathbf{x}_n)) \\ \mathbb{E}_{q(\mathbf{z})|\mathbf{x}_n)}[p(\mathbf{x}_n|\mathbf{Z})] - D_{KL}[q_\phi(\mathbf{Z}|\mathbf{x}_n)||p_\theta(\mathbf{Z})] &\leq \log(p(\mathbf{x}_n)) \end{aligned}$$

Question 1.7

The log-probability of $\mathbf{x}$ involves an intractable integral. Thus, we can only optimise it directly if we resort to Monte Carlo Integration for approximating this integral, which is inefficient as discussed in Question 1.4. However, by optimising the lower bound, we can be sure that eventually we will be pushing the the log-likelihood up as well.

Question 1.8

When the lower bound is pushed up, we know that the LHS of equation (11) must also go up by a corresponding amount. This can happen either because the KL-Divergence decreases, or because the log-likelihood increases. So either our approximation of the log-likelihood increases, or our approximation for the posterior of the latent variable given the data becomes better, i.e. the KL-Divergence decreases.

!!! MAYBE WRITE SOMETHING MORE HERE !!!

1.4 Specifying the Encoder: $q_\phi(\mathbf{z}_n|\mathbf{x}_n)$

Question 1.9

The first term of the loss represents the negative log-likelihood of the data $\mathbf{x}_n$ given the latent variable $\mathbf{z}$. The log-likelihood represents the log-probability that our model assigns to that particular reconstruction given a latent variable. We would like that our model assigns high probability to the reconstruction of the original image, so it is natural to interpret the negative log-likelihood as a reconstruction error because the higher it is, the lower the probability that our model has assigned to ground truth reconstruction.

The second term measures the dissimilarity between our learned posterior $p(\mathbf{z}|\mathbf{x}_n)$ and our prior $p(\mathbf{z})$. We want our posterior to be reasonably close to our prior to prevent overfitting the dataset, i.e. the model memorising a latent variable for each data point in our dataset. The KL-Divergence grows larger as distributions become more disjoint, and thus it is natural to interpret the divergence as a regularisation loss. The more similar the distributions will be, the smaller the KL-Divergence will become.

Question 1.10

In the following derivation, we approximate the expectation over q_ϕ with a single sample of the latent variable as is common practice in VAEs.

$$
\begin{aligned}
\mathcal{L} &= \frac{1}{N}\sum_{n=1}^{N} \mathcal{L}_n^{recon} + \mathcal{L}_n^{reg} \\
\mathcal{L}_n^{recon} &= -\mathbb{E}_{q_\phi(\mathbf{Z}|\mathbf{x}_n)}[\log p_\theta(\mathbf{x}_n|\mathbf{Z})] \\
&= -\log p_\theta(\mathbf{x}_n|\mathbf{z}_n) \\
&= -\sum_{m=1}^{M} \log Bern\left(x_n^{(m)}|f_\theta(\mathbf{z}_n)^{(m)}\right) \\
&= -\sum_{m=1}^{M} x_n^{(m)} \log\left(f_\theta\left(\mathbf{z}_n\right)^{(m)}\right) + \left(1-x_n^{(m)}\right)\left(1-\log\left(1-f_\theta\left(\mathbf{z}_n\right)^{(m)}\right)\right) \\
\mathcal{L}_n^{reg} &= D_{KL}[q_\phi(\mathbf{Z}|\mathbf{x}_n)||p_\theta(\mathbf{Z})] \\
&= \frac{1}{2}\Big(Tr\left(\mathbf{I}^{-1} diag\left(\boldsymbol{\Sigma}_\phi\left(\mathbf{x}_n\right)\right)\right) + \left(\mathbf{0}-\boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right)\right)^T \mathbf{I}\left(\mathbf{0}-\boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right)\right) + \\
&\quad - D + \log\frac{|\mathbf{I}|}{|diag\left(\boldsymbol{\Sigma}_\phi\left(\mathbf{x}_n\right)\right)|}\Big) \\
&= \frac{1}{2}\left(Tr\left(diag\left(\boldsymbol{\Sigma}_\phi\left(\mathbf{x}_n\right)\right)\right) + \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right)^T \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right) - D - \log|diag\left(\boldsymbol{\Sigma}_\theta\left(\mathbf{x}_n\right)\right)|\right) \\
\mathcal{L} &= \frac{1}{N}\sum_{n=1}^{N} -\sum_{m=1}^{M} x_n^{(m)} \log\left(f_\theta\left(\mathbf{z}_n\right)^{(m)}\right) + \left(1-x_n^{(m)}\right)\left(1-\log\left(1-f_\theta\left(\mathbf{z}_n\right)^{(m)}\right)\right) + \\
&\quad \frac{1}{2}\left(Tr\left(diag\left(\boldsymbol{\Sigma}_\phi\left(\mathbf{x}_n\right)\right)\right) + \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right)^T \boldsymbol{\mu}_\phi\left(\mathbf{x}_n\right) - D - \log|diag\left(\boldsymbol{\Sigma}_\theta\left(\mathbf{x}_n\right)|\right)\right)
\end{aligned}
$$

1.5 Reparametrization Trick

Question 1.11

a) The variational parameters ϕ are the parameters of the encoder. Because we want to update the encoder to accurately estimate the parameters of the posterior $p(\mathbf{z}_n|\mathbf{x}_n)$, we need to gradients of the loss w.r.t the variational parameters $\nabla_\phi \mathcal{L}$ in order to do so.

b) Sampling is a stochastic operation, which is not defined in terms of a regular mathematical expression. As such, we cannot back-propagate the gradients of the loss through the sampling operation.

c) The reparameterization trick allows to back-propagate gradients though the sampling step by treating the noise of a sample as an exogenous variable that is disconnected from the computational graph. Taking the example of a Gaussian distribution, we can sample from any such distribution

$$\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

by first sampling a noise term $\boldsymbol{\epsilon}$ from the standard normal distribution, and then obtaining $\mathbf{z}$ as a function of the noise.

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$
$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\Sigma}\boldsymbol{\epsilon}$$

In the forward pass, the decoder still sees the computation of $\mathbf{z}$ as a stochastic process, but in the backward pass, the encoder sees it as a deterministic computation involving the external noise $\boldsymbol{\epsilon}$ as input. Note that I treat the general case, where the covariance matrix is not necessarily diagonal

1.6 Putting things together: Building a VAE

Question 1.12

The Variational Auto-Encoder is defined by 3 modules: an Encoder, a Decoder, and a Variational Auto-Encoder module with encapsulates the previous 2. The Encoder receives an image and outputs the variational parameters (mean and variance) of the posterior distribution of the latent variable given the image. It consists of a fully connected layer with leaky ReLU activation to extract features from the image, followed by 2 separate linear layers that output the mean and the diagonal of the log-covariance matrix respectively. We treat the outputs of the second linear layer as log-variance values because the log of the variance can take any values in $\mathbb{R}$, whereas the variance needs to be positive. This way we do not have to process the outputs any further as part of the encoder, such as taking the absolute value.

The Decoder is a simple MLP with 2 layers and leaky ReLU activations. It receives a latent variable and output a reconstruction of a digit. The first layer can be seen as mapping the latent space from the standard normal distribution to a more complex one where the different classes of digits can be separated. The final layer is can be seen as the one mapping this latent space with interactions to a fully rendered digit. Because I expect the decoder to also output the parameters for Bernoulli distributions, the output non-linearity is a Sigmoid.

The Variational Auto-Encoder module combines the 2. It receives an images, passes it through the encoder, samples a latent variable using the reparametrization trick and passes that to the decoder to get the reconstruction. The loss is the computed between the reconstruction and the original digit. Figure 1 shows the ELBO values over epochs.

Question 1.14

The samples from the VAE do improve over time, although it seems that the model has not learned a very accurate representation at the end of training. The shape of digits can be made out, but with a lot of noise around it. Random samples of images and real data are presented in Figures 2a 2b, 2c. A greedy sample at the end of training and real data are also shown for reference in Figures 2d, 2e.

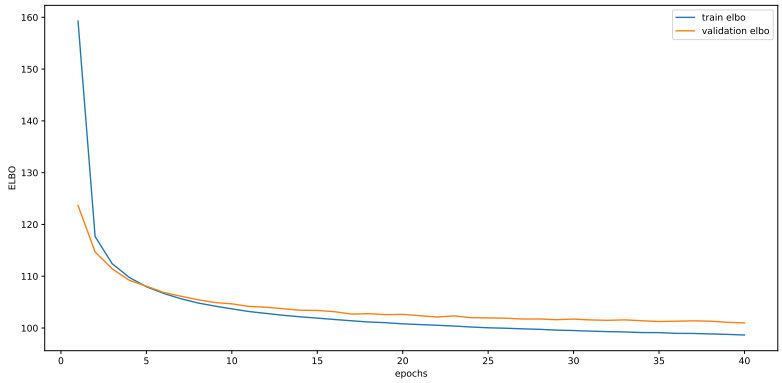
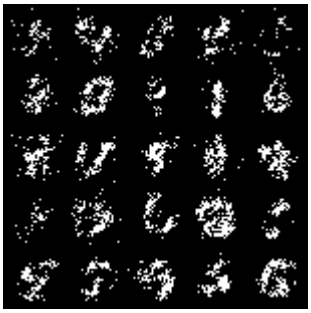
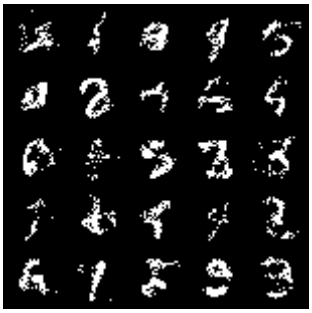


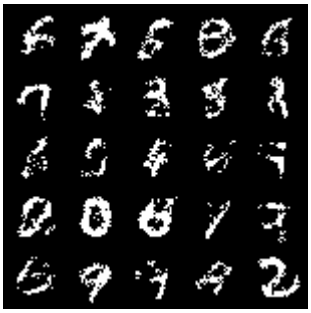
Figure 1: Question 1.13: ELBO Values for the Train and Validation set. *The ELBO values drops sharply at the beginning but do not change much over the second half of training. Importantly, the model doe snot start to overfit the training set, which is likely due to the regularisation term in the loss.*



(a) Random sample after the first epoch.



(b) Random sample after the 20th epoch



(c) Random sample after the last epoch.

(d) Greedy sample after the last epoch.

(e) Sample of real data.

Figure 2: VAE Generated Samples of MNIST Digits.

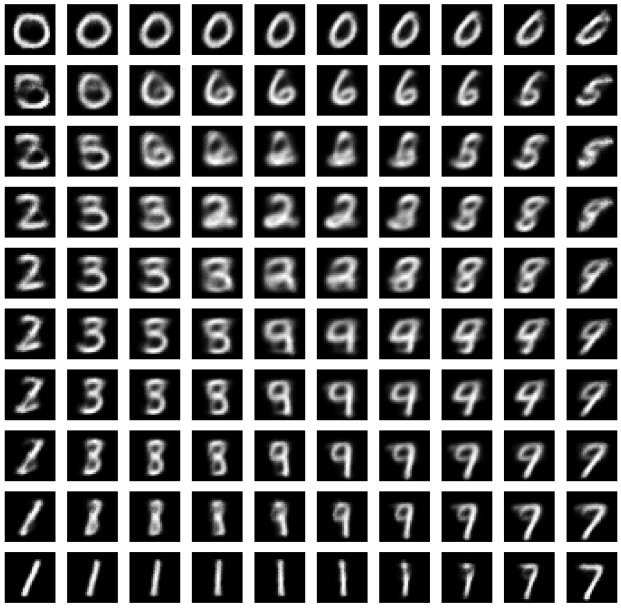


Figure 3: Learned Manifold for the Variational Auto Encoder. *Different regions in the latent space correspond to different classes of digits.*

Question 1.15

The learned manifold shows that the decoder has learned to represent different digits in the various regions of the latent space. Figure 3 shows the manifold.

Part 2: Generative Adversarial Networks

Question 2.1

Generator. The generator function takes as input a vector $\mathbf{z}$ of random noise sampled according to some distribution. In principle, it can be any distribution but the popular choices are the Uniform and Gaussian distributions. In this context:

- the vector $\mathbf{z}$ represents a sample from the latent space
- the choice for distribution of $\mathbf{z}$ represents our fixed prior over the distribution of the latent space
- the dimensionality of $\mathbf{z}$ represents the dimensionality of the latent space

The generator then outputs a sample from a distribution that follows the distribution of the data, to the extent that the generator was able to learn it. In the case of training GANs to generate hand-written digits, the output of the generator is an image of the same shape as the shape of the images from the training set.

Discriminator. The discriminator can take an input of any shape. This input can be either a real example from the training set $\mathbf{x}$, or a fake example generated by the generator function $\mathrm{G}(\mathbf{z})$. The discriminator is not aware a priori of whether this is a real or a fake example. It must learn to

distinguish between those during training. The output of the discriminator is a scalar between 0 and 1, which represents the probability of that image being real as judged by the discriminator.

2.1 Training Objective: A Minimax Game

Question 2.2

$\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log \mathrm{D}(\mathbf{x})]$**.** The first term is the log likelihood of the discriminator given the training data. It represents the log-probabilities that the discriminator classifies the training set images as real images. The task of the discriminator is to maximise this value. The generator does not play a role in this term.

$\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1-\mathrm{D}(\mathrm{G}(\mathbf{z}))]$**.** The second term represents the log-likelihood of the discriminator given the generated data. It represents the log-odds that the discriminator classifies generated data as fake. The task of the discriminator is to maximise this. The task of the generator is to minimise this.

Question 2.3

At convergence, the generator distribution and the data distribution are identical Goodfellow et al. [2014]

$$p_g = p_{data}$$

For an optimum discriminator, this means that it will be indifferent between real and fake examples:

$$\mathrm{D}^*(\mathbf{x}) = \mathrm{D}^*(\mathrm{G}(\mathbf{z})) = \frac{p_{data}}{p_{data}+p_g} = \frac{1}{2}$$

The objective at convergence can then be calculated as:

$$\begin{aligned}\mathrm{V}(\mathrm{D}^*,\mathrm{G}) &= \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log \mathrm{D}^*(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1-\mathrm{D}^*(\mathrm{G}(\mathbf{z}))] \\ &= \log\frac{1}{2} + \log\frac{1}{2} \\ &= -\log 4\end{aligned}$$

This is the only convergence point because we can rewrite the objective to make explicit the Jensen-Shannon Divergence term which is always non-negative:

$$\mathrm{V}(\mathrm{D}^*,\mathrm{G}) = -\log 4 + 2\cdot JSD(p_{data}||p_g)$$

The full derivation is available at Goodfellow et al. [2014].

Question 2.4

The generator's task is to minimise the objective function.

$$\mathrm{G}^* = \min_{\mathrm{G}} \mathrm{V}(\mathrm{D},\mathrm{G}) = \min_{\mathrm{G}}[\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log \mathrm{D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1-\mathrm{D}(\mathrm{G}(\mathbf{z}))]]$$

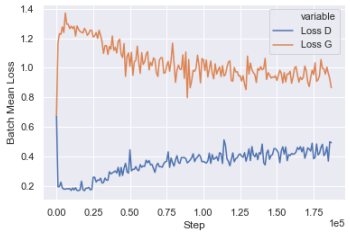
This has the same effect on G as when minimising the second term.

$$\mathrm{G}^* = \min_{\mathrm{G}} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[\log(1-\mathrm{D}(\mathrm{G}(\mathbf{z}))]$$

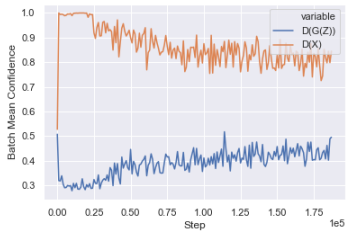
This means that the generator is trying to minimise the probability that the discriminator classifies the generated examples as fake.

At the beginning of training, the generator is typically very poor, while the discriminator can quickly learn to distinguish between real and fake images, because the training set images have a structure that is easy to distinguish from the pseudo-random examples created by the generator at first. Because of this, the discriminator will reject almost if not all the example from the generator with high confidence, which means that the second term of the objective will be a very small number.

$$\begin{aligned}\mathrm{D}(\mathrm{G}(\mathbf{z})) &\sim 1 \\ \log(1-\mathrm{D}(\mathrm{G}(\mathbf{z}))) &\sim 0\end{aligned}$$



(a) Batch Mean Losses. *Generator has a harder task, so the loss is usually higher for it. Convergence is reached when D(X) = G(G(Z)) = 0.5*



(b) Batch Discriminator Mean Confidence. The adversarial objective makes the losses of the 2 oscillate.

Figure 4: Generative Adversarial Network Training Statistics.

This means that the gradients for the generator will be extremely weak, not allowing it to learn any meaningful representations, since the small adjustments made by the generator when trying to minimise this value will not have a meaningful impact on the performance of the discriminator.

A workaround, called the non-saturating heuristic, proposes to cast the objective of the generator as maximising the probability that the discriminator classifies the generated examples as real. This keeps the same convergence guarantees as they are equivalent objectives but allows the generator to learn during early stages.

$$\mathrm{G}^* = \max_{\mathrm{G}} \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})}[\log(\mathrm{D}(\mathrm{G}(\mathbf{z}))]$$

2.2 Building a GAN

Question 2.5

In my implementation, both the generator and the discriminator follow the suggestions in the assignment. The generator is a 5-layer deep network with LeakyReLU activations and Batch Normalisation. The final output layer outputs a vector of size 784, the same size as an MNIST image, and has a TanH non-linearity to squash the values in the range of the normalised MNIST examples. The latent space is assumed to be Gaussian. The discriminator is a 3-layer deep network with LeakyReLU activations and a scalar Sigmoid output.

The adversarial loss used is the binary cross entropy loss between the outputs of the discriminator and a target vector that is either a vector of zeros or ones, according to the objective. Adam is used for optimisation. Because we need a good discriminator to provide guidance to the generator, I update the generator only once every $k = 5$ batches, while the discriminator is updated after every batch. This makes the gradients flowing to the generator be more meaningful. Losses and discriminator average confidence scores can be seen in Figures 4a 4b.

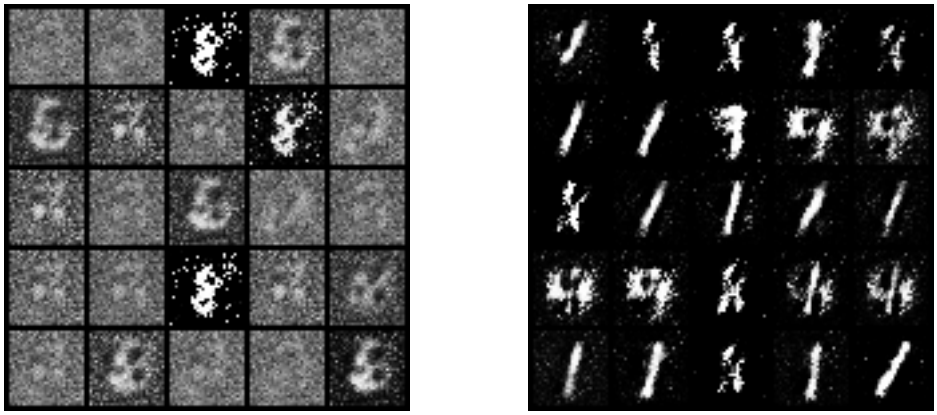
Question 2.6

As training progresses, the generator seems to learn the distribution of the data. At first, the images are just a bit more than random noise, but then evolve into some digits with noise around the edges of the image. Finally, the generator learns to create smooth images of hand-written digits with almost no noise. Figures 5

Question 2.7

To interpolate between variables in the latent space I use spherical linear interpolation White [2016]. This is motivated by the fact that linear interpolation in high-dimensional spaces will traverse areas that are highly unlikely given a Gaussian or Uniform prior on the latent space. The linear interpolation is given by

$$slerp(\mathbf{v}_1, \mathbf{v}_2, t) = \frac{\sin[(1-t)\Omega]}{\sin(\Omega)}\mathbf{v}_1 + \frac{\sin[t\Omega]}{\sin(\Omega)}\mathbf{v}_2$$



(a) Sample from batch 15,000

(b) Sample from batch 30,000

(c) Sample from batch 60,000

(d) Sample from batch 187,000

Figure 5: Generated Samples of MNIST Digits. *The samples presented according to a pseudo-exponential time scale with the amount of training done between each sample increasing.*

(a) Interpolation between 1 and 9. *Ones are over-represented in the latent space. One value in the middle seems to resemble a mix of 1 and 9.*

(b) Interpolation between 3 and 9. *Both digits seem to be well represented with values in the middle looking like a mix of a 3 and a 9.*

Figure 6: Interpolations of latent space. Because of spherical interpolation, examples in the middle are still well behaved.

where t is the interpolation parameter and Ω is the angle of the cosine between the 2 vectors

$$\Omega = \arccos(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{\|\mathbf{v}_1\| \cdot \|\mathbf{v}_2\|})$$

This interpolation scheme ensures that the statistics of the interpolations are more representative of our prior and has been shown to improve interpolation results White [2016]. Figures 6a and 6b show interpolations between 1 and 9, and between 3 and 9. It can be seen that the generator overemphasises ones, probably because ones are easy to learn and over-represented in the latent space. So likely, the generator has not converged yet to the true data distribution.

1 Part 3: Conclusion

Question 3.1

To conclude, VAEs and GANs are both generative models but they take different approaches to generate new samples. The most notable difference is that VAEs model the data manifold in the latent space, while GANs, specifically the generator part, simply provide a way to sample new examples that are similar to the data, i.e. they emulate a sampling procedure. Also, VAEs take the variational approach of maximising a lower bound on the log-probability of the data, while GANs optimise an adversarial objective cast in the form of a mini-max game.

There is no clear winner between the 2, especially because it is not easy to compare their performance, since the VAE was tasked with modelling binary data, while the GAN was modelling continuous data. Even as such, we could say that the GAN seems to provide more varied samples that resemble the training data more accurately. I should mention though that the VAE with a 2-dimensional latent space provided much prettier samples than the 20-dimensional VAE.

Overall, both are powerful generative models, as their popularity suggests. I would say if these methods are developed and refined further, great new things will come about.

References

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

Michael E. Tipping and Chris M. Bishop. Probabilistic principal component analysis. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 61(3):611–622, 1999.

T. White. Sampling Generative Networks. *ArXiv e-prints*, September 2016.