Deep Learning - Assignment 3

Andrei Pauliuc
11596619
andupauliuc@gmail.com

1 Variational Auto Encoders

1.1 Latent Variable Models

Question 1.1

With probabilistic PCA, the data is reduced to a lower dimensional latent space by estimating the principal axes $\mathbf{W}$ and noise term σ^2. Similarly, VAE aims to find a latent representation of the data (in a much lower dimensional space) via the encoder-decoder architecture, and by constraining the latent variables to be drawn from a standard-normal distribution. Thus, the common objective of these methods is to find a good lower dimensionality latent space and model the distribution $p(X|Z)$. However, there are huge differences between the assumptions on how to move from latent space to data, which result in differences of implementation and limitations. For example, pPCA uses linear transformations, while VAE uses non-linear due to the non-linear activations in the neural networks.

1.2 The Decoder

Question 1.2

The following sampling procedure can be used for the model presented:

- sample z_n
- pass z_n through f_θ and compute $f_\theta(z_n)$
- sample $x_n^{(m)} = Bern(x_n^{(m)}|f_\theta(z_n)_m)$ for $m = 1 \dots M$
- compute $p(x_n|z_n) = \prod_{m=1}^{M} p(x_n^{(m)}|f_\theta(z_n)_m)$
- repeat

Question 1.3

The main reason behind using a latent variable coming from a standard-normal distribution without any trainable parameters comes from the fact that any d-dimensional distribution can be generated from a set of d normal distributions mapped through a complex function. In this case, z_n is drawn from a normal distribution and passed through a neural network (having non-linear activations). We believe that the corresponding f_θ is complicated enough to transform z to such that it models the true data distribution.

Question 1.4

a) Sampling with Monte Carlo methods can be done by using the following formula:

$$\log p(\boldsymbol{x}_n) \approx \frac{1}{M} \sum_{m=1}^{M} \log p(\boldsymbol{x}_n^{(m)}|f_\theta(\boldsymbol{z}_n)_m), \quad \text{with } \boldsymbol{z}_n \sim p(Z)$$

This requires M to be large enough such that the estimate is accurate.

b) Monte Carlo estimation uses random sampling procedure for $\boldsymbol{z}_n$, from which it estimates $p(\boldsymbol{x}_n|\boldsymbol{z}_n)$. The problem with this procedure is that the number of samples M might be extremely large in order to get a good estimate. Comparing the whole latent space $p(z)$ with $p(z|x)$ as presented in the figure provided in assignment, we notice that only few samples $z \sim p(z)$ are actually from the area of interest. Consequently, many of the samples would lead to a posterior value close to 0, leading to an erroneous estimation of $p(x)$. Furthermore, there is a problem when the dimensionality of z increases. In this case, the dimension of the latent space increases as well, and many more samples have to be drawn in order to be successful with the Monte Carlo estimation.

1.3 The Encoder

Question 1.5

a) KL-divergence is 0 when the distributions match, hence $(\mu_q, \sigma_q^2) = (0, 1)$. However, its value becomes very big when the two distributions differ by far (high distance between means); for example, values $(\mu_q, \sigma_q^2) = (100, 1)$ results in $D_{\text{KL}}(q||p) = 5,000$.

b)

$$\begin{aligned} D_{\text{KL}}(q||p) &= -\log \sigma_q + \frac{\sigma_q^2 + \mu_q^2}{2} - \frac{1}{2} \\ &= \frac{1}{2}\left(-\log \sigma_q^2 + \sigma_q^2 + \mu_q^2 - 1\right) \end{aligned}$$

Question 1.6

By definition, KL-divergence is definitely non-negative, being equal to 0 only when the two distributions match. Since the second term on the right-hand-side of the equation is the KL-divergence, the log-probability of the data is lower bounded by this right-hand-side expression.

Question 1.7

If we were to optimise the log-probability directly, it would contain an intractable integral. This comes from the term $D_{\text{KL}}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$, and the problem here is that $P(Z|X)$ is not analytically computable. However, we can overcome this problem by using $Q(Z|X)$ and the KL-divergence mentioned before which will pull $Q(Z|X)$ towards $P(Z|X)$. Assuming that $Q(Z|X)$ is powerful enough, at some point it will match $P(Z|X)$, leading to 0 KL-divergence and a direct optimisation of the log-probability.

Question 1.8

When the lower bound is pushed up, either $\log p(\boldsymbol{x}_n)$ increases or $D_{\text{KL}}(q(Z|\boldsymbol{x}_n)||p(Z|\boldsymbol{x}_n))$ decreases. In the first case, an increase in the log-probability of the data means that the model fits the data well and learns to create outputs similar to the given input. In the second case, when KL-divergence decreases, it implies that $q(Z|\boldsymbol{x}_n)$ gets closer to $p(Z|\boldsymbol{x}_n)$. This is one of the objectives of the VAE, since we want $q(Z|\boldsymbol{x}_n)$ to get close to $\mathcal{N}(0, 1)$, approximating $p(Z|\boldsymbol{x}_n)$ well.

1.4 Specifying the encoder

Question 1.9

The reconstruction loss term is defined as the difference between output and the input image, given the value of a latent variable z. Since we aim to move from latent space to the image space and have a faithful reproduction of the original image, it is very suitable to name this term as *reconstruction* loss.

The second term is the regularisation loss which comes from a penalty based on how different $q_\phi(Z|\boldsymbol{x}_n)$ is from the actual $p_\theta(Z)$. The KL-divergence is suitable for this since we would like to have two matching distributions. Naming this loss as *regularisation* is suitable because it regularises the distribution $q_\phi(Z|\boldsymbol{x}_n)$ to match $p_\theta(Z)$.

Question 1.10

$$
\begin{aligned}
\mathcal{L}_n^{\text{recon}} &= -\mathbb{E}_{q_\phi(z|\boldsymbol{x}_n)}[\log p_\theta(\boldsymbol{x}_n|Z)] \\
&= -\mathbb{E}_{q_\phi(z|\boldsymbol{x}_n)}[\log \text{Bern}(\boldsymbol{x}_n^{(m)}|f_\theta(\boldsymbol{z}_n)_m))] \\
&= -\sum_{m=1}^{M} \boldsymbol{x}_n^{(m)} \log\left(f_\theta(\boldsymbol{z}_n)_m\right) + (1-\boldsymbol{x}_n^{(m)}) \log\left(1-f_\theta(\boldsymbol{z}_n)_m\right)
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L}_n^{\text{reg}} &= D_{\text{KL}}(q_\phi(Z|\boldsymbol{x}_n)\,||\,p_\theta(Z)) \\
&= D_{\text{KL}}(\mathcal{N}(\boldsymbol{z}_n|\mu_\phi(\boldsymbol{x}_n), \text{diag}(\Sigma_\phi(\boldsymbol{x}_n)))\,||\,\mathcal{N}(0,\mathbb{I})) \\
&= \frac{1}{2}\left[\text{Tr}(\mathbf{I}^{-1}\cdot\text{diag}(\Sigma_\phi(\boldsymbol{x}_n))) + \mu_\phi(\boldsymbol{x}_n)^T\cdot\mathbf{I}^{-1}\cdot\mu_\phi(\boldsymbol{x}_n) - D + \log\frac{|\mathbf{I}|}{|\text{diag}(\Sigma_\phi(\boldsymbol{x}_n))|}\right] \\
&= \frac{1}{2}\left[\text{Tr}(\text{diag}(\Sigma_\phi(\boldsymbol{x}_n))) + \mu_\phi(\boldsymbol{x}_n)^T\cdot\mu_\phi(\boldsymbol{x}_n) - D + \log\frac{1}{|\text{diag}(\Sigma_\phi(\boldsymbol{x}_n))|}\right] \\
&= \frac{1}{2}\left[\sum_d \Sigma(\boldsymbol{x}_n)_d + \sum_d \mu(\boldsymbol{x}_n)_d^2 - D - \log\sum_d \Sigma(\boldsymbol{x}_n)_d\right]
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{L} &= \sum_{n=1}^{N} \mathcal{L}_n^{\text{recon}} + \mathcal{L}_n^{\text{reg}} \\
&= \sum_{n=1}^{N} -\sum_{m=1}^{M} \boldsymbol{x}_n^{(m)} \log\left(f_\theta(\boldsymbol{z}_n)_m\right) + (1-\boldsymbol{x}_n^{(m)}) \log\left(1-f_\theta(\boldsymbol{z}_n)_m\right) \\
&+ \frac{1}{2}\left[\sum_d \Sigma(\boldsymbol{x}_n)_d + \sum_d \mu(\boldsymbol{x}_n)_d^2 - D - \log\sum_d \Sigma(\boldsymbol{x}_n)_d\right]
\end{aligned}
$$

1.5 The Reparametrization Trick

Question 1.11

a) We need $\nabla_\phi \mathcal{L}$ to backpropagate the loss gradient through the encoder network and update the weights with the corresponding error. Not being able to compute $\nabla_\phi \mathcal{L}$ would mean that the encoder does not learn anything and there is no correspondence created between input images and latent space.

b) Directly sampling z from $Q(z|X)$ using the encoder's outputs $(\mu(X), \Sigma(X))$ creates some decoupling in the backpropagation flow because the sampling procedure is, by nature, non-differentiable. The SGD methods cannot pass the gradient information to the layers in encoder due to the presence of random node (z) in the computational graph.

c) The reparametrization trick moves the random sampling to another input variable ϵ which does not come between the encoder and decoder. Using this variable to draw samples, we can combine it with the encoder's outputs and compute values of z:

$$z = \mu + \sigma * \epsilon = \mu(X) + \Sigma^{1/2}(X) * \epsilon$$

Since z would then be made up of addition and multiplication, these operations allow the gradient to flow to the other layers. This trick results in identical behaviour of the feedforward step, while also fully enabling the backpropagation algorithm.

1.6 Putting things together: Building a VAE

Question 1.12

The encoder network contains a linear layer with tanh activation, followed by other two linear layers corresponding to mean and logarithm of variance (more stable than standard deviation). The decoder contains a linear layer with tanh activation, followed by the output layer with sigmoid activation. The VAE model is trained for a given number of epochs on the training set in the following way. The model passes each batch through the encoder, and using the reparametrization trick computes values for $\boldsymbol{z}$, which are then passed to the decoder. The ELBO value for a batch is a summation of the reconstruction loss using binary cross entropy between input and output, and of the regularisation loss involving $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}^2$. Averaging over the batch size, gradients are computed and the weight updates is performed. Sampling from the VAE is done by passing through the decoder a random sample from $\mathcal{N}(0, 1)$.

Question 1.13

Using a latent space of size 20 and training the model for 20 epochs, the training and validation ELBO values are presented in figure 1.

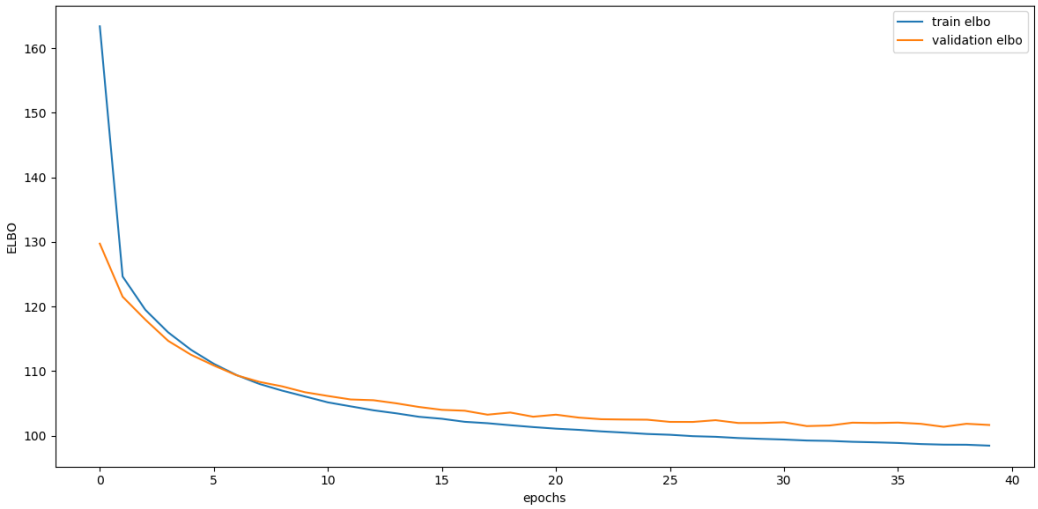

Figure 1: VAE ELBO, latent space size 20

Question 1.14

Taking random samples before training, halfway through training and after training, we plot these images in figure 2. We can also show the Bernoulli means corresponding to these images, done in figure 3

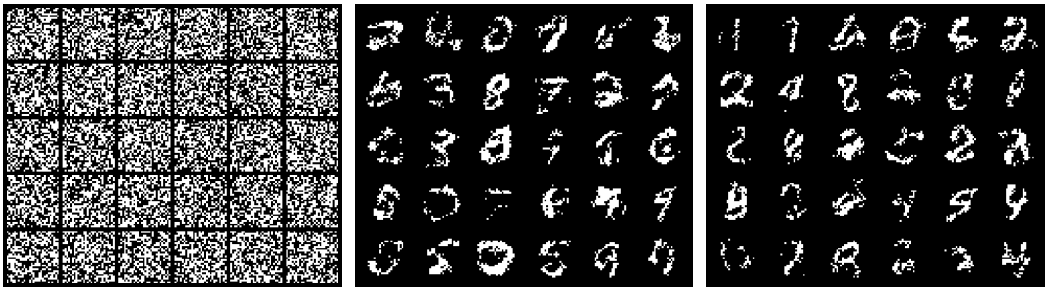

Figure 2: Sampled images: before training (left), halfway through training (centre), after training (right)

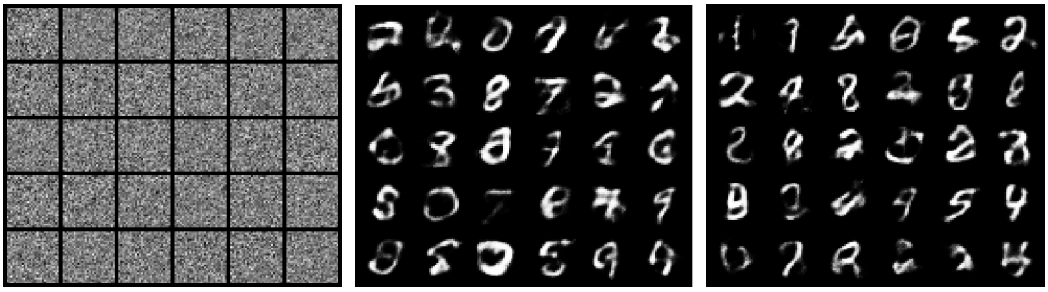

Figure 3: Bernoulli means: before training (left), halfway through training (centre), after training (right)

There is definitely an improvement between the samples taken, the last samples being more close to the MNIST dataset we have trained the VAE on.

Question 1.15

Using the percent point function, we compute the manifold values of a VAE model trained with a 2-dimensional latent space. The outputs can be plotted in a 20×20 grid, resulting in figure 4.

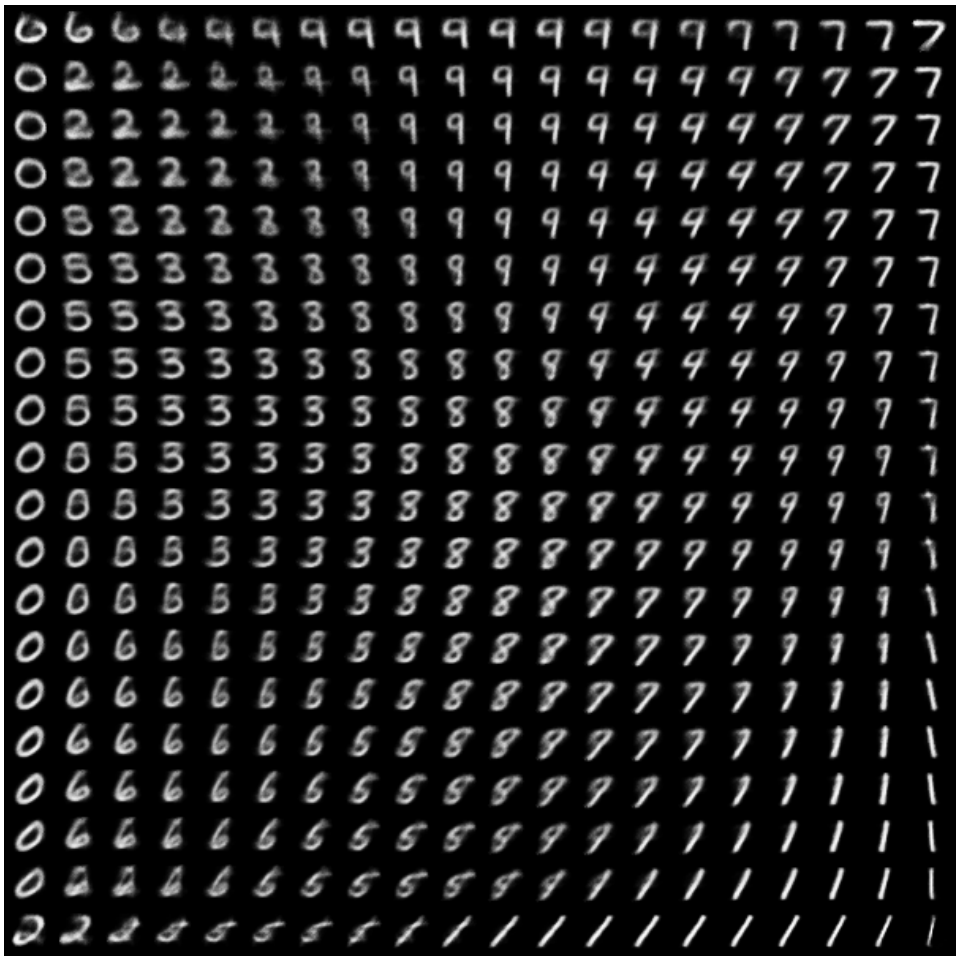

Figure 4: Manifold of a VAE with 2-dimensional latent space

2 Generative Adversarial Networks

Question 2.1

Consider G to be the generator function and D to be the discriminator function. The input of the generator function is a noise vector $x \sim \mathcal{N}(0, 1)$ of size (z_{dim}). $G(x)$ represents the output image of the generator (in our case, size 784 or 28×28), having pixel values in range $[-1, 1]$.

The input of the function D is $G(x)$, while its output, $D(G(x))$ is a probability value in $[0, 1]$, meaning how sure the discriminator is that the generated image is real. In the same time, the discriminator accepts as input a real datapoint y of same shape as $G(x)$. The real image is used to make the discriminator correctly identify fake from real images.

2.1 Training objective: A Minimax Game

Question 2.2

In the GAN training objective, the first term $\mathbb{E}_{p_{\text{data}}(\boldsymbol{x})}[\log D(X)]$, represents the expectation of the logarithm of discriminator's output with respect to X drawn from the data distribution. More specifically, this is the expected log-probability that the X came from the data. In this objective function, the aim is to maximise this with respect to D, such that the discriminator outputs values of 1 for real data.

The second term is $\mathbb{E}_{p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(Z)))]$ and it is the expected log-probability that $G(Z)$ does not come from the real data. Here, both G and D come into play, each with different purposes. On the one hand, the discriminator wants to maximise the term, thus pushing $D(G(Z))$ to 0. On the other hand, the generator wants to minimise it by pushing $D(G(Z))$ towards 1.

Question 2.3

From the GAN paper, we see that global minimum is achieved when $p_{\text{g}} = p_{\text{data}}$, leading to discriminator loss of $D_G * (x) = 1/2$. Evaluation the equation

$$C(G) = \mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{g}}(\boldsymbol{x})} \right] + \mathbb{E}_{\boldsymbol{x} \sim p_{\text{g}}} \left[\log \frac{p_{\text{g}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{g}}(\boldsymbol{x})} \right]$$

at $D_G * (x) = 1/2$, we find that $C(G) = -\log 4$. This is indeed the global minimum, since $C(G)$ can be rewritten as

$$C(G) = -\log 4 + 2 \cdot JSD(p_{\text{data}} || p_{\text{g}})$$

The Jensen-Shannon divergence is non-negative, and 0 only when the 2 distributions are equal. Thus, after training has converged, $V(D, G) = -\log 4 = -2 \log 2$.

Question 2.4

The problem with the term $\log(1 - D(G(Z)))$ during early training comes from the fact that the gradient might be not sufficient enough to let G learn to model the data well. Since G starts with random values, the discriminator D can easily differentiate between random data and training data, which in turn makes the term $D(G(Z))$ to go to 0, saturating the gradient quite early in the process. The solution to the problem posed by minimisation of $\log(1 - D(G(Z)))$ is to actually maximise $\log D(G(Z))$, which leads to the same solution, but provides stronger and more stable training gradients.

2.2 Building a GAN

Question 2.5

The generator and discriminator were constructed using the given architecture, being composed of linear layers, leaky ReLU, batch norm layers and tanh output for the generator, and sigmoid output for the discriminator. At training, a random batch of training images is drawn, together with random noise. The noise is first passed to the generator, resulting in a generated image. This is passed to the discriminator, and together with a vector of true targets of 1 the loss is computed and

backpropagated through the generator. For the discriminator, the loss is computed as the difference between $D(real\ input)$ and true labels, summed with the difference between $D(G(x))$ and fake labels (0). We thus want the discriminator to learn to differentiate between real image and generated images. The training procedure for discriminator is repeated k times for 1 generator step, k being given as hyperparameter.

Question 2.6

We have trained the GAN for 200 epochs for a latent space dimension of 100, with 5 training discriminator steps for 1 generator step. This ensures that the generator can learn something meaningful and produces correct images. At 3 equally distributed time steps during training, we have sampled random images using the generator and got the results shown in figure 5. The left image shows only noise, since the generator has not learnt anything about the dataset; while the centre image shows a lot of improvement, there are many repeated numbers and their representation is not really good. Finally, the last image shows a lot of improvement, and we can see that the generator learnt correctly to create MNIST numbers.

Figure 5: GAN sampled images: start of training (left), halfway through training (centre), after training (right)

Question 2.7

Once the generator is fully trained, we can interpolate between two digits in the latent space. This visualisation shows how the generator moves between two points in latent space, sometimes going through a space occupied by the representation of another number. Figure 6 shows two cases of interpolation, one between numbers 1 and 4, and one between 6 and 7. We can clearly see how in the first row, the number 1 switches its shape to something resembling a 9, and finally becoming a 4. This is also interesting for the second case, where 6 goes into 0 and then to 7.

Figure 6: GAN: interpolations in latent space

3 Conclusion

Question 3.1

As we have seen throughout this assignment, both VAE and GAN are good models for generative tasks. There are aspects where these two families of models are alike, and there are aspects where they are very different. Although they take a very different approach, we have seen that both models

succeed in learning to recreate samples of the MNIST dataset which are quite faithful to the original style, with the GAN model recreating much better images than VAE.

On one hand, VAE is a probabilistic graphical model with the objective to model latent variable, and its advantage comes from the probabilistic formulation where one needs to maximise the lower bound of the log-likelihood. VAE is also easier to train and use, while its implementation is straight forward.

On the other hand, GAN is a model explicitly made for generative tasks and, as we have seen, it is better when it comes to generating visual features. It offers flexibility when it comes to the architecture of the networks, but it poses some problems regarding the variety in the output samples.

In conclusion, both models are good as generative models, although GAN presents better end results than VAE. The biggest advantage of the VAE is the probabilistic framework which makes the model easy to understand.