Assignment 3

Luis Vitor Zerkowski - 14895730

December 19, 2023

1.2 Decoder

1.1

Given the distribution for the latent variable, we can sample a point $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_D)$. With the point $\boldsymbol{z} \in \mathbb{R}^D$, we can simply pass it through the decoder f_θ to get intensity probabilities for M pixels. Now that we have probabilities for different intensities (0 to 255) for every pixel, we can choose the intensity of each pixel. This can be done in multiple ways, the simplest being to set the intensity value to the most probable class, but we could also, for example, sample the the intensity from the categorical distribution we just computed. This process will give us a final intensity value for every pixel M, hence sample an image given the latent variable distribution and the decoder.

1.2

It is not efficient because the quality of the approximation depends on the dimensions of the latent variable $\boldsymbol{z}$ and the number of samples L we have. More specifically, the higher the dimension of our latent variable $\boldsymbol{z}$, the sparser the latent space (curse of dimensionality), which means the equation requires exponentially more data to yield a good approximation.

1.3 Encoder

1.3

To prove that the right hand side of equation 10 is a lower bound on the log-probability $\log p(\boldsymbol{x}_n)$ we just need to prove that $KL(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n|\boldsymbol{x}_n))$ is non-negative. To prove this, we do:

$$KL(p(x)||q(x)) = \sum_x p(x) \log \frac{p(x)}{q(x)} = -\sum_x p(x) \log \frac{q(x)}{p(x)} \geq$$

Now knowing that $-\log x \geq 1 - x$ for any $x > 0$ and that $p(x), q(x) \geq 0$ by definition of probabilities, we have:

$$\geq \sum_x p(x) \left(1 - \frac{q(x)}{p(x)}\right) = \sum_x (p(x) - q(x)) =$$

Again by definition of probabilities, it follows:

$$= \sum_x p(x) - \sum_x q(x) = 1 - 1 = 0$$

So we have proved that $KL(p(x)||q(x)) \geq 0$, which means in particular $KL(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n|\boldsymbol{x}_n)) \geq 0$. This last inequality together with equation 10 lead to:

$$\log p(\boldsymbol{x}_n) \geq \log p(\boldsymbol{x}_n) - KL(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n|\boldsymbol{x}_n)) = \mathbb{E}_{q(\boldsymbol{z}_n|\boldsymbol{x}_n)}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)] - KL(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n))$$

And we have just proved that the right hand size of equation 10 is indeed a lower bound to $\log p(\boldsymbol{x}_n)$.

1.4

Pushing up the lower bound means increasing the value of the right hand side of equation 10 which naturally can be caused by two things:

1. A lower value for $KL(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n|\boldsymbol{x}_n))$ and thus a better approximation of the posterior.
2. A higher value for $\log p(\boldsymbol{x}_n)$ and thus increased likelihood of the data.

1.4 Optimization Objective

1.5

The 'reconstruction' name is appropriate because this is the part of the loss that refers to the decoder, hence ability to reconstruct the data given samples from the latent variable. This part of the loss is responsible for making sure we are maximizing the log-likelihood of the data once we have the latent variable, going back to the original distribution (reconstruction once again).

The 'regularization' name, in turn, is appropriate because it accounts for our approximation of the posterior when compared to our prior, thus related to the encoder output. In this sense, it makes sure we are not simply projecting the data onto the latent space in an overfitting way, but rather taking into account the previous knowledge we have from the latent variable distribution (typically the unit Gaussian on VAE's case).

1.5 Reparametrization Trick

1.6

Sampling prevents us from computing $\nabla_\phi \mathcal{L}$ because it introduces randomness in the forward pass of our VAE, which means there's no defined rule on how to compute gradients for this part of the network during the backward pass.

The reparametrization trick solves this problem by subdividing our sampling process into a deterministic part and a non-deterministic part. So instead of sampling $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\boldsymbol{z}|\mu_\phi(\boldsymbol{x}), diag(\sigma_\phi(\boldsymbol{x})))$, we set $\boldsymbol{z} = \mu_\phi(\boldsymbol{x}) + \sigma_\phi^{\frac{1}{2}}(\boldsymbol{x}) \cdot \boldsymbol{r}$, with $r \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and $\sigma_\phi^{\frac{1}{2}}(\boldsymbol{x})$ a vector containing only the square root of the diagonal elements of the matrix $diag(\sigma_\phi(\boldsymbol{x}))$. Now we can still add randomness to the forward pass by sampling from $\boldsymbol{r}$, but we have a direct way to compute gradients for $\boldsymbol{z}$ during the backward pass.

1.6 Building a VAE

1.7

For reference, since Tensorboar's plots don't include the model's BPD on the first epoch, I evaluated it manually and got a training BPD of 4.018, around 4 as expected. Now for the final test score, I got test BPD of 0.519, around 0.52 as expected.

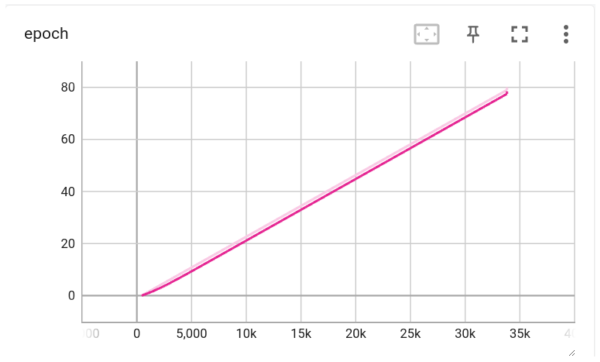
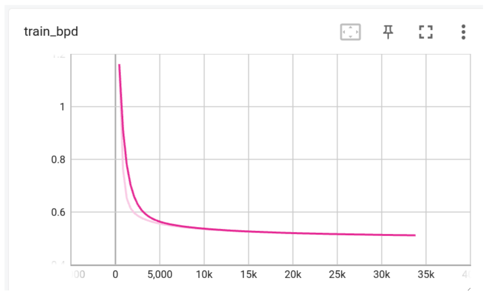
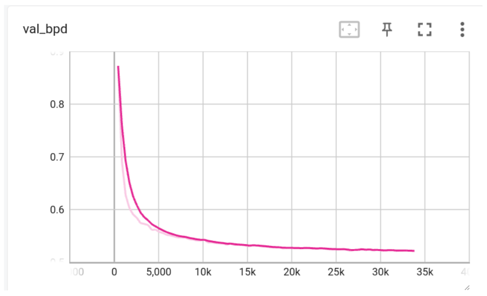


Figure 1: Plot containing Epoch X Iteration for help on mapping the iterations to epochs on the other plots.



(a) Plot of BPD X Iteration on training data.



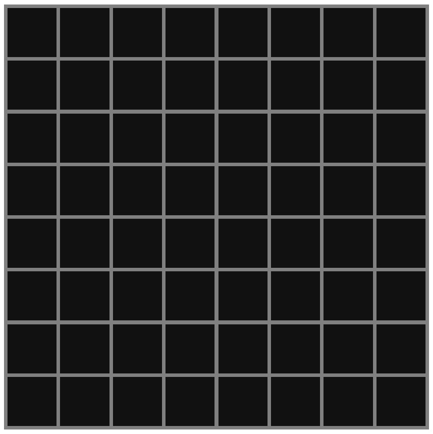
(b) Plot of BPD X Iteration on validation data.

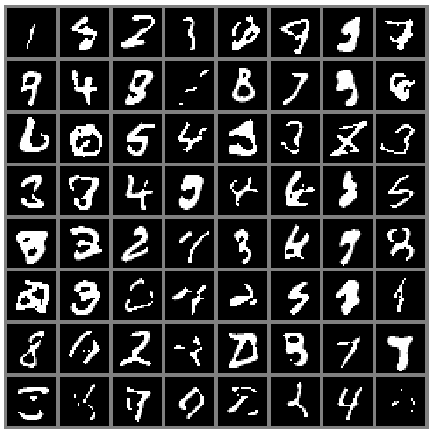
Figure 2: BPD X Iteration Plots.

1.8

From figure 3a we can see that, before training, our samples are blank, indicating we still have no capacity of generating proper images with the decoder. After 10 epochs of training, we can already see a huge improvement, as shown in figure 3b. On this stage, our samples already started looking like handwritten digits, but there still is a lot of noise.

By the end of the training process, after epoch 80, we can see that our samples are already pretty decent, generating reasonable hand written digit images from random samples pretty successfully. It's worth noting that, eve though we have good results after the full training, we can still see some noisy samples, images that clearly were supposed to be digits, but aren't quite there yet (the degenerate 8 on the seventh row and third column for example).


(a) Samples before training.


(b) Samples after 10 epochs.

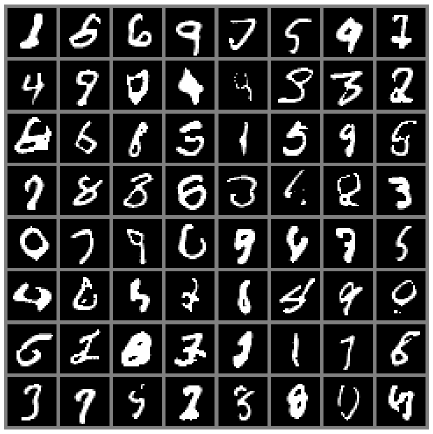

(c) Samples after 80 epochs.

Figure 3: Samples throughout the training process.

1.9

Through figure 4 it is interesting to notice that the spatial coordinate dictates a lot of patterns in the data. Digits with similar shape are close on the manifold - ones and sevens or sevens and nines for example - and digits with different shapes are distant - ones and zeros for example. There's also the orientation pattern, so it's clear that digits leaning more to the right occupy the upper part of the grid while digits leaning more to the left occupy mostly the bottom part of it.

On top of these shape and orientation patterns, it's also worth noting that digits seem to gradually transform into one another throughout the manifold. It's possible to see, for example, some samples for which the image we get is a mixture between two or more digits. This indicates that these digits with a specific shape and orientation occupy a similar region of the latent space.

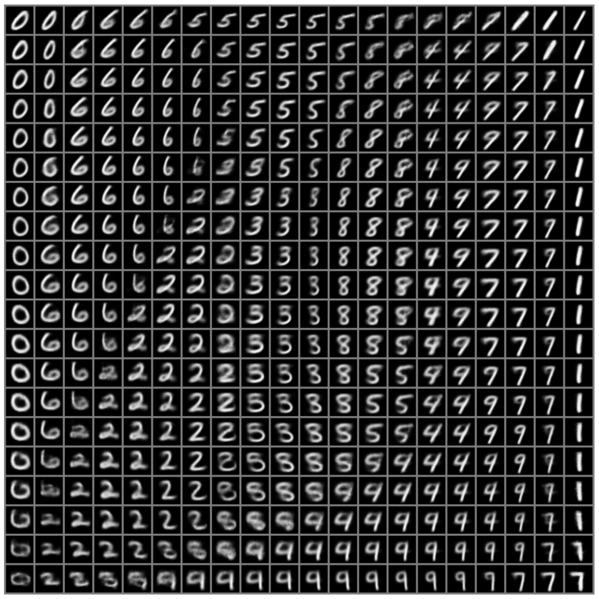


Figure 4: Data manifold for VAE trained with a 2-dimensional latent space.

2 Adversarial Autoencoder Networks

2.1

2.2

Adversarial auto-encoders can reduce the mode collapse problem because of the reconstruction error. Since they also have to take into consideration the reconstruction of the original image, if it was to always generate one class of output or even one specific instance, it would end up having a huge reconstruction error for numerous inputs. Naturally because of that, during training the model will also optimize considering the diversity of inputs, avoiding mode collapsing.

2.3

2.3.a

2.3.b

λ affects the training objective by weighting how much the model should focus on the reconstruction error. If $\lambda = 1$, the model will focus solely on reconstructing the original input, basically converging to an autoencoder and making no use of the discriminator network.

On the other hand, if $\lambda = 0$, the model will simply ignore the reconstruction error, becoming a slightly modified version of the standard GAN again, without the autoencoder. Note that it is modified because on a standard GAN you start from the latent variable ans use the generator to produce synthetic data, whereas in our case we would start from data and generate an encoded version of it.

3 Attention, Transformers, and LLMs

3.1

The main challenge is related to the self-attention mechanism. Since it computes the relationship between all token pairs, it scales quadratically with the size of the sequence. This means that long sequences can easily impose memory consumption and computational cost issues to the network.

The best way to overcome this problem is to sparsify the attention matrix. This can be done in multiple ways, but the most common is restricting the attention computation for a smaller range of neighbors (don't compute the relationship between tokens that are too far away), either by establishing a fixed range or even learn patterns related to range of context on the fly for example.