Deep Learning 2020 **(Due: [20-12-2020])**

Submission Assignment #5

Instructor: Jakub Tomczak *Name:* Jens van Holland, *Netid:* jhd660

This report is about the fifth assignment, namely, the VAE's and GAN's. Firstly, the questions from the first part are answered directly, the other parts are in a report format. I used code from different sources, these are stated in this file and/or in the attached Python script. At last, due to the time I had for this assignment, one weekend, I haven't done the full assignment. The missing parts: the analysis on the imagenette data set and the FID/Inception score (I tried but no reportable results were made). Nevertheless, it was fun to do. Enjoy.

1 Question answers

Question 1 *Write objective functions for VAEs and GANs. Use standard normal prior p(z) for VAE.)*

Objective functions
The objective function for the VAE, described by Kingma and Welling (2013), is the following,

$$\mathcal{L}(\theta, \phi; \mathbf{x}^{(i)}) = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)})||p_\theta(\mathbf{z})) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}^{(i)})}\big[\log(p_\theta(\mathbf{x}^{(i)}|\mathbf{z})\big]$$

where,

$$\begin{aligned}
\phi &:= \text{the variational parameters} \\
\theta &:= \text{the generative parameters} \\
q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) &:= \text{the probabilistic encoder, is assumed to be } \mathcal{N}(\mathbf{z}|\mu^{(i)}, \sigma^{2(i)}\mathbf{I}) \\
p_\theta(\mathbf{x}|\mathbf{z}) &:= \text{the probabilistic decoder which is chosen, } \mathcal{N}(\mathbf{x}|\theta(\mathbf{z}), 1) \\
p_\theta(\mathbf{z}) &:= \text{prior over the latent variables, is assumed to be } \mathcal{N}(0, \mathbf{I})
\end{aligned}$$

The D_{KL} is the Kullback-Leibler divergence which indicates how similar two distributions are, in this case $q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ and $p_\theta(\mathbf{z})$. The expectation on the right hand side is the reconstruction error.
The actual function that we want to maximize, from the lecture slides,

$$\text{ELBO} = \ln\big(\mathcal{N}(\mathbf{x}|\theta(\mathbf{z}), 1)\big) - \Big[\ln\big(\mathcal{N}(\mathbf{z}|\mu^{(i)}, \sigma^{2(i)})\big) - \ln\big(\mathcal{N}(0, 1)\big)\Big]$$

Since one wants to maximize ELBO, one can minimize -ELBO. The implementation is done with the code from the lecture slides, this post and for the the ELBO this repository.
Next, the objective function for the GAN. The next function is from the slides, also called the adversarial loss,

$$\min_G \max_D \mathbb{E}_{\mathbf{x}\sim p_{real}}\big[\log(D(\mathbf{x}))\big] - \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\big[\log(1 - D(G(\mathbf{z}))\big]$$

where,

$$\begin{aligned}
D(\cdot) &:= \text{the discriminator function} \\
G(\cdot) &:= \text{the generator function}
\end{aligned}$$

As can be seen above, one want to minimize w.r.t. the generator and maximize w.r.t. the discriminator. So, for the implementation two different optimizers are used. This tutorial is used for programming the GAN.

Model components
Firstly, the components of the VAE. The VAE uses an *encoder*, a *decoder* and a *sampling procedure*. The encoder encodes the data, say image, to the latent space. This encoding is a form of dimension reduction, one encodes the image to lower dimension. Next, the output of the encoder are the parameters μ and σ.
Using these parameters, one can sample from this latent space using the reparametrization trick. This sampled vector is called $\mathbf{z}$. Next, the decoder decodes $\mathbf{z}$ back to the original space, this is in the same space as the

original image.
During training, the model tries to optimize both the encoding and decoding. Since the model learns a distribution in the latent space, one can generate, after training, data such as images.

Next, the GAN. The GAN has three main components, namely, the *sampling*, the *generator* function, and the *discriminator* function. Basically, the model learns by generating (using the generator and sampling) or provided, say, images and tries to detect whether an image is real or not (using the discriminator). During training, the generator tries to out perform the discriminator and like wise. This makes the model learn a distribution for generating images, which can be used to generate images.

2 Problem statement

For this assignment, one has to build a VAE and a GAN and train those models on the MNIST data set. The data set contains handwritten digits (graphs), these digits are values ranging from 0 to 9. Thus, there are 10 different classes for a model to learn. The training set contains 60,000 samples, the test set contains 10,000 samples. Each graph (written digit) is represented as a vector of length 784. The values of such vector are ranging from 0 to 255, indicating the intensity of a pixel. These values are standardised before they go into the model. The main goal is to generate images, handwritten digits, and see if they are any good.

3 Methodology

The VAE model is trained on a part of the data set, namely 55,000 samples, and validated on the validation set which contains 5,000 samples.
The encoder and decoder of the VAE have the following, fairly simple, architecture:

Type	architecture
Encoder	$Linear(784,300) \rightarrow ReLU() \rightarrow Linear(300,\ 32)$
Decoder	$\text{Linear}(16,\ 300) \rightarrow ReLU() \rightarrow Linear(300,\ 784) \rightarrow Sigmoid()$

Table 1: VAE encoder and decoder architecture

From table 1, the latent space has 16 dimensions. The optimizer that is used is the Adam optimizer with a learning rate of 0.005 and the number of epochs is 50.

The generator and the discriminator in the GAN have the following, also fairly simple, architecture:

Type	architecture
Generator	$Linear(16,300) \rightarrow ReLU() \rightarrow Linear(300,\ 784)$
Discriminator	$Linear(784,\ 300) \rightarrow ReLU() \rightarrow Linear(300,\ 1) \rightarrow Sigmoid()$

Table 2: GAN generator and discriminator architecture

The GAN uses two optimizers, one for the generator and one for the discriminator. The Adam optimizer with a learning rate of 0.0002 is used for both. Also, the GAN is trained for 50 epochs.
For both the VAE and the GAN, a batchsize of 64 is used.

4 Experiments

To start with the VAE. The learning curve is shown in figure 1. Firstly, the loss curve goes down during training. Secondly, after many tries I could not figure out why the loss is negative. The results (2 - 6) show that the VAE learned how to decode. So, I assume that the training went well enough despite the somewhat weird loss curve.

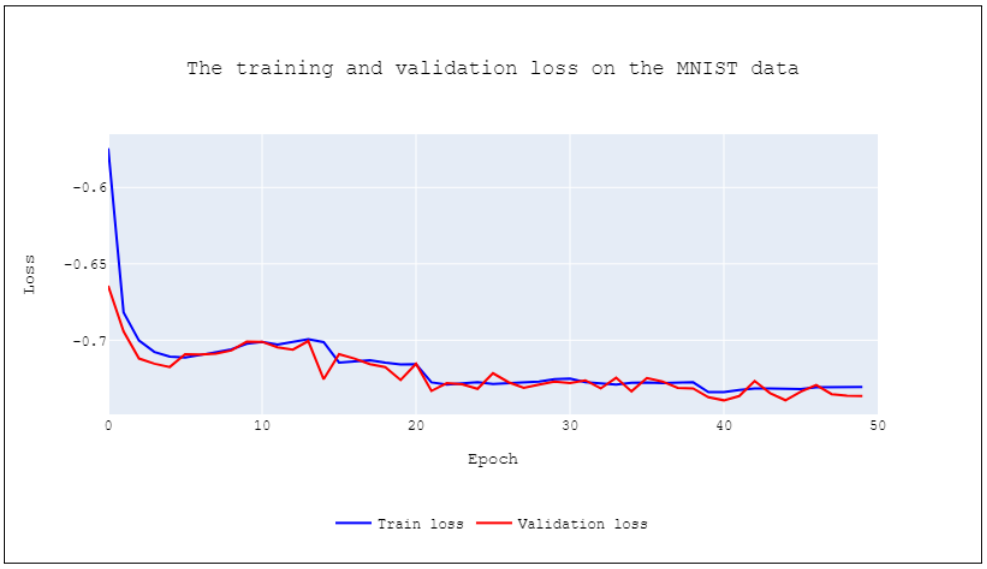

Figure 1: The validation and training loss of the VAE.

To see how good the VAE mimics the input data, the figures 2 - 6 show how good the decoder works. The first row are the original images and the second row are the decoder outputs.

Figure 2: The output after the first epoch

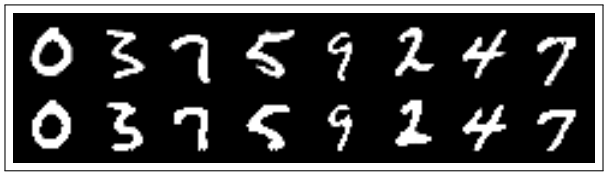

Figure 3: The output after the 10'th epoch

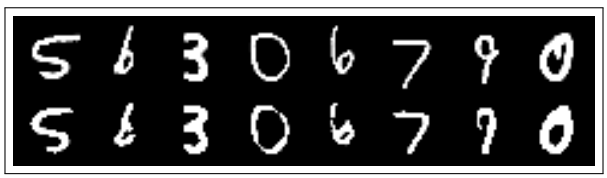

Figure 4: The output after the 20'th epoch

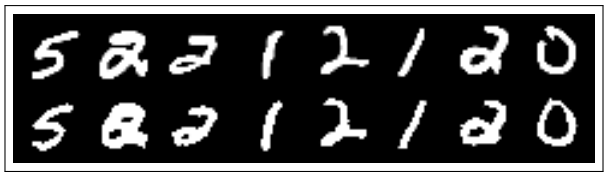

Figure 5: The output after the 30'th epoch

Figure 6: The output after the 40'th epoch

Already after the first epoch does the VAE a reasonable job of mimicking the data. Next, the latent variable can be visualised using T-SNE. T-SNE is used to visualize high dimensional data in 2d, or 3d. This is shown for the test data set in figure 7. It seems that the visualisation of the latent space contains ten clusters, corresponding to the ten digits.

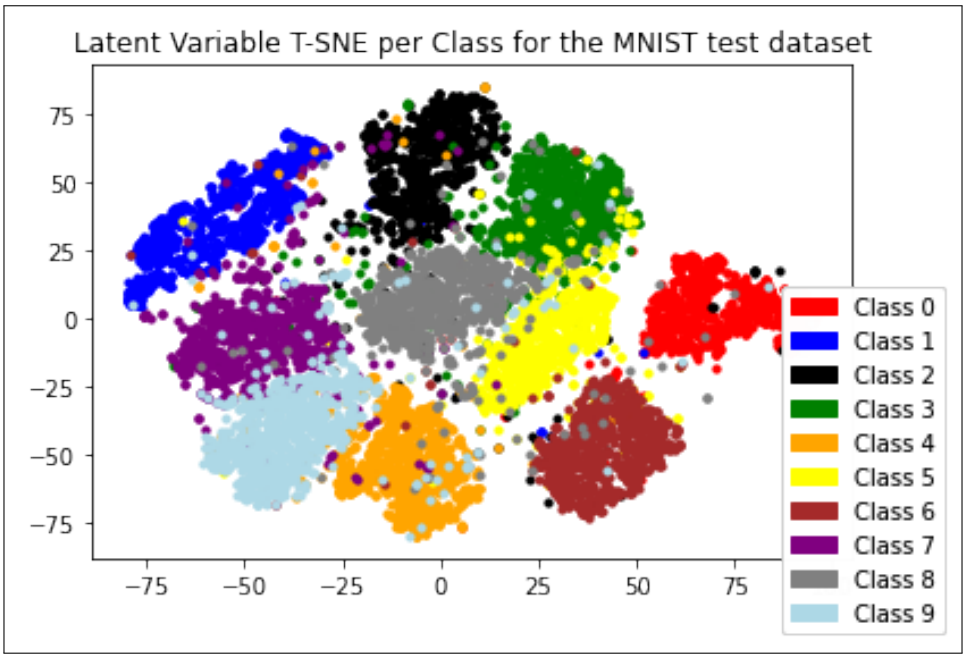

Figure 7: Visualisation of the latent variable, this is made using code from this repository.

Next, the GAN. The loss of the GAN is divided into 3 groups, namely, the discriminator loss for the fake data and real data and the generator loss. This is shown in figure 8. It seems that the learning process is somewhat 'wavy', I am not sure what this means but it looks interesting.

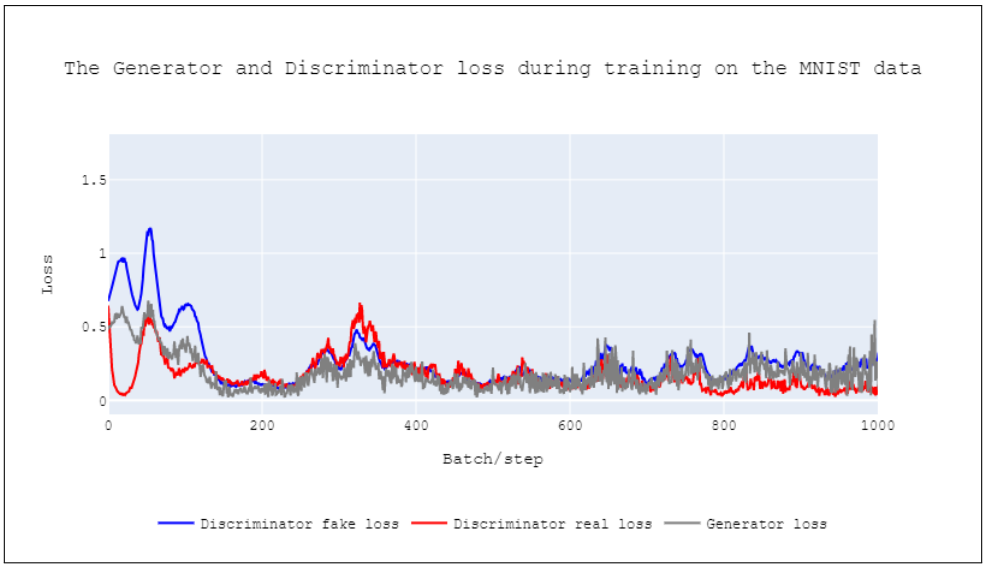

Figure 8: The loss of the generator and discriminator for the GAN.

Furthermore, the generators output after several epochs is shown in figures 9 - 13. From these figures, the GAN is particularly font of the number 1, 4, 7, 8 and 9.

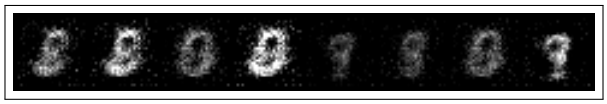

Figure 9: The output after the first epoch

Figure 10: The output after the 10'th epoch

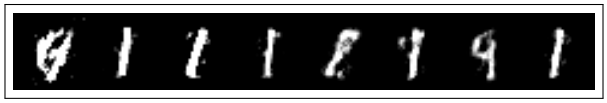

Figure 11: The output after the 20'th epoch

Figure 12: The output after the 30'th epoch

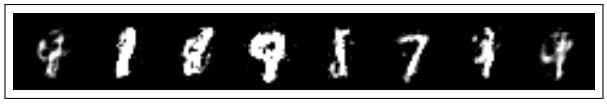

Figure 13: The output after the 40'th epoch

To end this analysis, two points from the multivariate standard normal distribution are picked and linearly interpolated. For the VAE, two random values are chosen for the mean and standard deviation. This can

be seen in the figure below ((a) - (l)), the first row is from the VAE decoder and the second row is from the generator of the GAN. It looks like the each image is slightly different than the previous one.

(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l)

5 Results and discussion

This report analysed a simple GAN and VAE. The goal of this report was to train these models on the MNIST data set and generate images. The training of the VAE went well, except for the somewhat weird loss curve. From figures 2 - 7, it seems that the decoder of the VAE does a good job. The generated images, figures above, are a bit vague in the context of digits; I can't tell if it is a 7 or a part of a 9.
Furthermore, the GAN also learned fairly well. From figures 9 - 13, it clearly shows improvement over time (epochs), but I have the feeling that the generator only knows a few digits and not the whole set.

References

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.