Deep Learning 2020 **(Due: 20 dec 2020)**

Submission Assignment #5a

Instructor: Jakub Tomczak *Name:* Mick IJzer, *Netid:* 2552611

1 Problem Statement

A Generative Model is a powerful way of learning any kind of data distribution using unsupervised learning. Two of the most commonly used and efficient generative models are Variational Autoencoders (VAE) and Generative Adversarial Networks (GAN). Both can be categorized as latent variable models. However, GAN is considered as an implicit model, while VAE is described as an prescribed model. VAE aims at maximizing the lower bound of the data log-likelihood, also known as the ELBO (see equation 1.1), and GAN aims at achieving an equilibrium between Generator and Discriminator (i.e. the adverserial loss, see equation 1.2). Equations 1.1 and 1.2 are also used as the objective functions in this report. Besides a different objective function, the architecture of GAN and VAE is also different. For GAN there is a generator that generates an image based on noise. These generated images and a set of real images are then given to the discriminator who's aim is to differentiate between real and generated images. For VAE a real image is given to an encoder, which compresses the image to a latent space. The prior of VAE prescribes the distribution within the latent space; the Kullback Leibler-divergence (i.e. the second part of equation 1.2) penalizes the model if the distribution of the latent space differs from the prescribed distribution. The reparameterization trick is then used to make the process differentiable. The decoder then attempts to reconstruct the image based on the latent variables.

In this report both GAN and VAE will be trained on two sets of images; the MNIST dataset and the Imagenette dataset. The MNIST dataset consists of 60.000 28x28 pixels greyscale images of hand-written digits. The Imagenette dataset contains around 9.500 images from 10 classes of the well known Imagenet dataset. The images are in color and vary in shape and size, but are on average 160x160 pixels. The classes in the Imagenette dataset range from people holding fishes to chainsaws, and from dogs to parachutists. The task of training a generative model on the MNIST dataset is many times easier due to the size and complexity of the images. This means that for each dataset a different architecture has to be used. The performance of the models will be evaluated by the objective functions as well as the quality of generated images. After the training process 64 images will be generated and evaluated. Furthermore, the interpolation between two points in the latent space will also be visualized and evaluated. This is done for both GAN and VAE and for both the MNIST and Imagenette datasets. The goal of this report is to compare the quality of generated images of GAN and VAE to determine which one performs best.

$$ELBO = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[ln\, p(\mathbf{x}|\mathbf{z})] - KL[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] \tag{1.1}$$

$$loss = min_G\, max_D\, \mathbb{E}_{\mathbf{x}\sim P_{real}}[log\, D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \tag{1.2}$$

2 Methodology

As explained in section 1 four different architectures will be used. The models will be implemented using PyTorch. Since the images from the MNIST dataset are relatively small it was possible to use fully connected layers for both GAN and VAE. This was not the case for the Imagenette dataset; convolutional layers were needed to reduce the number of weights. Furthermore, the size of the images within the Imagenette dataset differ, therefore as preprocessing step all images were (center-)cropped to be 160x160 pixels. To be able to adequately compare the performance of GAN and VAE the number of weights for both networks was kept similar. Therefore, the encoder (VAE) and discriminator (GAN), and the decoder (VAE) and generator (GAN) have similar structures. The architecture and total number of weights of the GAN and VAE for the MNIST dataset are described in the appendix in table 2 and for the Imagenette dataset in table 3.

The two algorithms below show the pseudocode of the training process of VAE and GAN. Note that the loss for GAN (see equation 1.2) has been split up into two separate loss functions for the generator and discriminator. Furthermore, the generator and discriminator have two separate optimizers. The Adam optimizer

is used as optimizer for all models. The ADAM optimization algorithm is an extension to the widely used stochastic gradient descent, where the per-parameter learning rate is adaptive. The per-parameter learning rates are computed based on estimates of the first and second moments of the gradients. Lastly, the prior that was chosen for VAE was the standard normal distribution. This means that the model will be penalized if the distribution of the latent space differs from the standard normal distribution. The noise for GAN will also be drawn from a standard normal distribution. Eventually, it will be possible to generate images by sampling from the standard normal distribution and passing the sample to the generator or decoder. Which will be done to evaluate the performance of both models.

Algorithm 1: Training Loop GAN

```
Result: Trained GAN
for each epoch do
    for each batch do
        Discriminator:;
        Reset gradients;
        output_T = Discriminator(input_T);
        input_F = Generator(noise);
        output_F = Discriminator(input_F);
        loss = BCELoss(output_T, output_F);
        backward(loss);
        optimzer.step();
        ;
        Generator:;
        Reset gradients;
        input = Generator(noise);
        output = Discriminator(input);
        loss = BCELoss(output);
        backward(loss);
        optimizer.step();
    end
end
```

Algorithm 2: Training Loop VAE

```
Result: Trained VAE
for each epoch do
    for each batch do
        Reset gradients;
        mu, log_std = encoder(input);
        z = reparameterize(mu, log_std);
        output = decoder(z);
        loss = MSE(input, output) + ELBO;
        backward(loss);
        optimzer.step();
    end
end
```

3 Experiments

As described in section 1, the goal of this report is to obtain information on which generative model performs best. To achieve this goal the architecture of the neural networks as well as the hyperparameters have to be tuned, while making sure that VAE and GAN are still comparable. In section 2 the final model architectures have been described. However, it must be noted that many different architectures were tried for the Imagenette dataset (e.g. number of layers and types of layers). For the final architectures several hyperparameters were tuned. The selected hyperparameters are shown in table 1. For each hyperparameter several values were tested, but the ones with the best subjective generated images and objective loss after 10 iterations were chosen.

After the hyperparameters have been selected the four separate generative models were trained on their respective datasets. The loss is calculated and stored per batch, and will be analyzed together with the generated images to evaluate the performance of each model.

Parameter	MNIST (VAE)	MNIST (GAN)	Imagenette (VAE)	Imagenette (GAN)
Learning Rate	$2e^{-4}$	$2e^{-4}$	$2e^{-4}$	$2e^{-4}$
Latent Size	20	100	100	500
Batch Size	64	64	48	48
Epochs	100	200	200	100

Table 1: Hyperparameters.

4 Results and Discussion

MNIST The loss curve, generated images, and interpolation of the VAE are shown in figures 1, 2, 3 respectively. Figures 4, 5, 6 show the same for the GAN. The objective loss of the two models is not comparable due to the different loss functions that were used. However, for both it can be said that they show that the model learns the distribution of the real MNIST dataset. The extent to which that was true can be seen in figures 2 and 5. For both models random samples from a standard normal distribution were drawn and passed through

the decoder and generator. The results show that almost all generated images are interpretable as digits. Where the images of VAE are somewhat blurry, the images of GAN are not always a complete digit (i.e. there are breaks in the lines). The last two figures 3 and 6 show that the latent space between two generated samples can be interpolated and produce recognizable digits as well. The transition between two digits is clearly visible in these figures.

Imagenette The loss curve, generated images, and interpolation of the VAE are shown in figures 7, 8, 9 respectively. Figures 10, 11, 12 show the same for the GAN. While not directly visible, the loss curve of the VAE in figure 7 was still decreasing at the moment the training was stopped. The generated images in figure 8 somewhat resemble images if looked at from far away. However, the quality is not good. Therefore, for the interpolation real images were used as starting points as opposed to random points in the latent space. The first and last column in figure 9 show the true images. The actual output of the VAE given these true images is of good quality, and the interpolation between those seems natural as well. Unfortunately, as can be seen in figures 11 and 12 the chosen architecture/hyperparameters did not provide any decent results. To solve this many different strategies were tried; label smoothing, upsampling layers instead of ConvTranspose, feature matching, reducing number of layers, different sizes for the latent space, Two Time-Scale Update Rule and more.

Conclusion With respect to the MNIST-models it can be said that both VAE and GAN produce images of similar quality. However, based on the number of epochs needed to produce these images it can be said that VAE reaches the desired performance quicker. Furthermore, VAE can be used to find features in the latent space of real images; which can then be used for other analyses. This is not the case for GAN. Lastly, as stated in sections 2 and 3 multiple architectures and hyperparameters were evaluated. During this testing the performance of VAE was a lot more stable, while the performance of GAN could drop with a slight change in architecture or hyperparameter values. The difference between the ease of use of VAE and GAN becomes much clearer for the Imagenette dataset. Where the results for VAE are not perfect, the results of GAN don't even resemble the target. Unfortunately, due to time constrains, more approaches to get good results with GAN could not be tested. In conclusion, the performance of VAE is stable, and VAE are relatively easy to work with. In contrast, GAN are a lot harder to work with. Both are able to produce decent quality images.

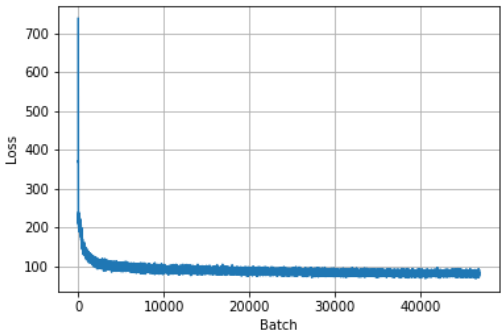

Figure 1: Loss of VAE on MNIST

Figure 2: Generated images VAE

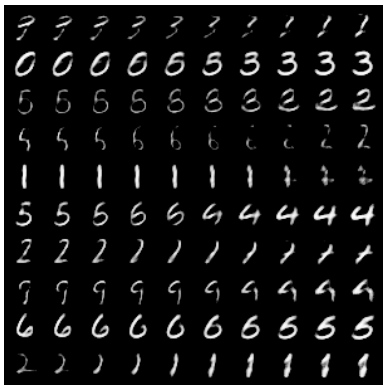

Figure 3: Interpolation VAE

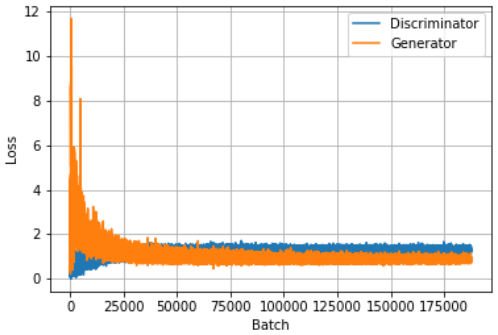

Figure 4: Loss of GAN on MNIST

Figure 5: Generated images GAN

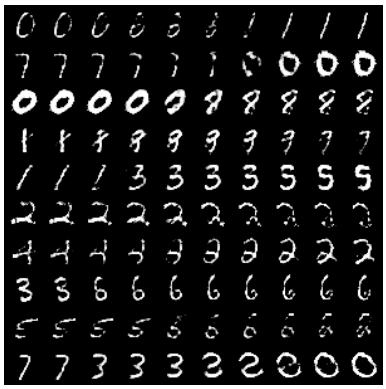

Figure 6: Interpolation GAN

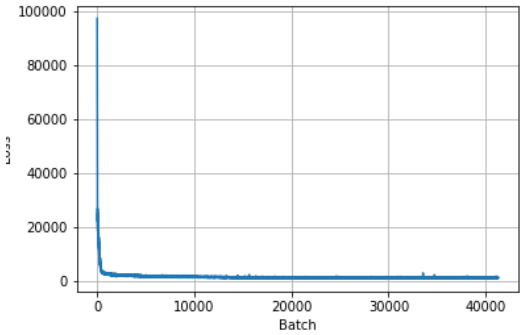

Figure 7: Loss of VAE on IMAG

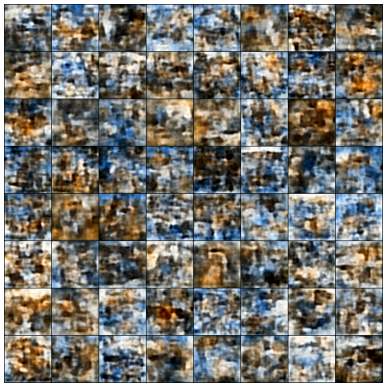

Figure 8: Generated images VAE

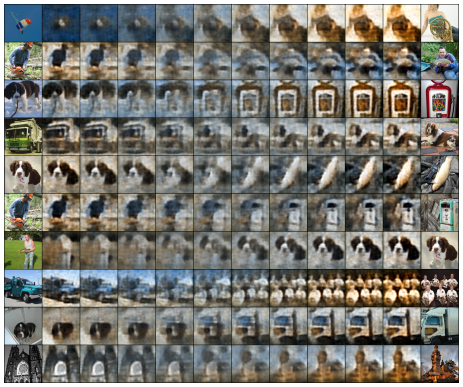

Figure 9: Interpolation VAE

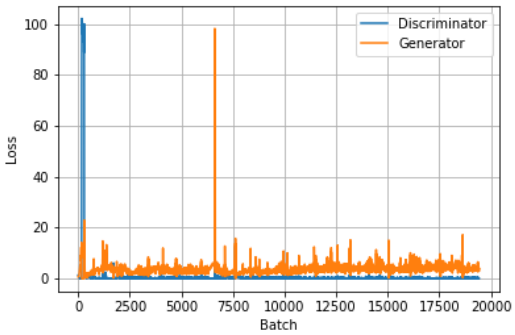

Figure 10: Loss of GAN on IMAG

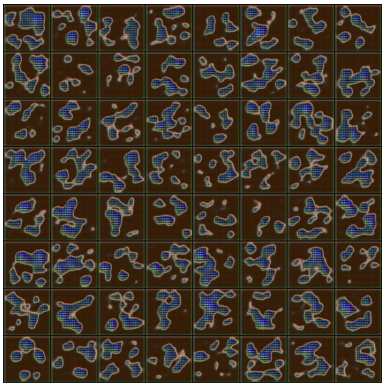

Figure 11: Generated images GAN

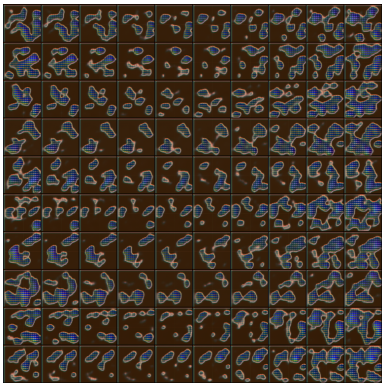

Figure 12: Interpolation GAN

Appendix

	Layer/Action	Size VAE	Size GAN
Encoder and Discriminator	**Linear**	(784, 1024)	(784, 1024)
	- *Leaky ReLU*		
	- *Dropout*	0.3	0.3
	Linear	(1024, 512)	(1024, 512)
	- *Leaky ReLU*		
	- *Dropout*	0.3	0.3
	Linear	(512, 256)	(512, 256)
	- *Leaky ReLU*		
	- *Dropout*	0.3	0.3
	Linear	(256, 40)	(256, 1)
Decoder and Generator	**Linear**	(20, 256)	(100, 256)
	- *Leaky ReLU*		
	Linear	(256, 512)	(256, 512)
	- *Leaky ReLU*		
	Linear	(512, 1024)	(512, 1024)
	- *Leaky ReLU*		
	Linear	(1024, 784)	(1024, 784)
	Total Weights:	13.204	13.206

Table 2: Architectures for MNIST dataset.

	Layer/Action	Size VAE	Size GAN
Encoder and Discriminator	**Conv**	(3, 16, 3)	(3, 16, 3)
	- *ReLU*		
	- *Maxpool*	2	2
	Conv	(16, 32, 3)	(16, 32, 3)
	- *ReLU*		
	- *Maxpool*	2	2
	Conv	(32, 64, 3)	(32, 64, 3)
	- *ReLU*		
	- *Maxpool*	2	2
	Conv	(64, 128, 3)	(64, 128, 3)
	- *ReLU*		
	- *Maxpool*	2	4
	Linear	(8.192, 400)	(2.048, 1)
Decoder and Generator	**Linear**	(200, 256*8*8)	(500, 256*8*8)
	- *Leaky ReLU*		
	ConvTransp	(256, 128)	(256, 128)
	- *Kernel/Stride*	4/2	4/2
	- *Leaky ReLU*		
	- *BatchNorm2d*		
	ConvTransp	(128, 64)	(128, 64)
	- *Kernel/Stride*	4/2	4/2
	- *Leaky ReLU*		
	- *BatchNorm2d*		
	ConvTransp	(64, 32)	(64, 32)
	- *Kernel/Stride*	4/2	4/2
	- *Leaky ReLU*		
	- *BatchNorm2d*		
	ConvTransp	(32, 16)	(32, 16)
	- *Kernel/Stride*	4/2	4/2
	- *Leaky ReLU*		
	- *BatchNorm2d*		
	ConvTransp	(16, 8)	(16, 8)
	- *Kernel/Stride*	4/2	4/2
	- *Leaky ReLU*		
	- *BatchNorm2d*		
	ConvTransp	(8, 3)	(8, 3)
	- *Kernel/Stride*	3/1	3/1
	- *Leaky ReLU*		
	- *Interpolate*	(160, 160)	(160, 160)
	Total Weights:	43.625	43.627

Table 3: Architectures for Imagenette dataset.