Neural Networks and Deep Learning - Homework #2

Chioccarello Stefano - 2011656
stefano.chioccarello.1@studenti.unipd.it

January 14, 2022

This report describes the second homework of the Neural Networks and Deep Learning course about Unsupervised Learning. The goals, methods and results achieved in this activity are described in detail in the following.

1 Introduction

The goal of this homework was to implement and test deep architectures for solving unsupervised problems, namely working with unlabelled datasets. The dataset used in this case is the *FashionMNIST* dataset, which is the same of the first homework. This allowed to analyze better the behaviour of the networks and to make comparisons with the first homework. The deep networks that were implemented for this task were: a convolutional autoencoder (CAE), a fine-tuned version of such architecture but with a supervised classification part, and a generative adversarial network (GAN) to generate samples from scratch. Moreover, latent space structures were explored by means of dimensionality reduction techniques.

2 Convolutional Autoencoder

The dataset consists of 28x28 pixels black and white images and 10 classes (*t-shirt, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, boot*), and it is divided into 60'000 images for the training set and 10'000 images for the test set. Moreover, for the parameters optimization, a validation set was created by randomly selecting 10'000 samples from the training one. In order to explore different ways of creating deep networks, the CAE architecture and the hyperparameter tuning were implemented with the *Pytorch-Lightning* and *Optuna* libraries, which allowed to make the training and the optimization very smooth and efficient.

2.1 Methods

The CAE architecture consists of two separate networks, namely the *Encoder* and the *Decoder*. The former was implemented as follows:

- a convolutional layer with *in_features* = 1 and *out_features* = n (number of channels), a 3x3 *kernel* with *padding* = 1 and *stride* = 2, so that the image is reduced from 28x28 to 15x15

- another convolutional layer with *in_features* = n and *out_features* = 2*n, a 3x3 *kernel* with *padding* = 1 and *stride* = 2, so that the image is reduced to 8x8
- one last convolutional layer with *in_features* = 2*n and *out_features* = 4*n, a 3x3 *kernel* with *padding* = 0 and *stride* = 2, so that the image is reduced to 3x3
- a flattening layer
- a fully connected layer with *in_features*= 3*3*4*n and *out_features* = 64
- finally, a last fully connected layer that maps the 64 *in_features* into the *latent space* of dimension 2.

The latter, on the other hand, has a symmetric structure, namely:

- a fully connected layer with *in_features* = 2 and *out_features* = 64
- a fully connected layer with *in_features* = 64 and *out_features* = 4*n*3*3
- an unflattening layer
- convolutional layer with *in_features* = 4*n and *out_features* = 2*n, a 3x3 *kernel* with *output_padding* = 0 and *stride* = 2, so that the image is resized to 8x8
- a convolutional layer with *in_features* = 2*n and *out_features* = n, a 3x3 *kernel* with *output_padding* = 1 and *stride* = 2, so that the image is resized 15x15
- another convolutional layer with *in_features* = n and *out_features* = 16, a 3x3 *kernel* with *output_padding* = 1 and *stride* = 2, so that the image is resized to 28x28.

Each layer is activated by a *ReLU* activation function, and the final layer of the decoder is followed by a *Sigmoid* activation function in order to normalize the output between $[0, 1]$.
The number of channels n and the *latent space* were tuned in the optimization part. An initial guess of such values was used to generate some samples and then to make the comparison with the best parameters found in the random search section. The values provided in the laboratory activity n. 5 were used, namely n=8 and *encoded_space_dim*=2. After that, during the search, the number of channels was chosen between $[3, 10]$ and the encoded space dimensionality was picked between $[2, 50]$. In addition to such parameters, the random search approach with *Optuna* was adopted by choosing also between:

- Stochastic Gradient Descent optimizer with: *learning_rate* chosen between $[1e-5, 1e-2]$; *momentum* chosen between $[0.7, 1]$; *weight_decay* chosen between $[1e-6, 1e-3]$
- Adam optimizer with same parameters.

The random search was performed over 100 models, each one trained for 50 epochs in total. The *Optuna MedianPruner* was used to perform some pruning and hence discard some of the proposed models, by using the median stopping rule. The Figures (1) and (2) show the optimization process,

while in the results section the best parameters are listed.

2.2 Results

The number of epochs and the batch size of the training set were fixed respectively at 50 and 256 for all the models. The first CAE was trained with the following parameters:

Optimizer	LR	Weight decay	N_channels	Encoded_space_dim
Adam	1e-4	1e-5	8	2

Table 1: Initial parameters

Such architecture resulted in a test loss of 0.03184. In Figure (3) are shown some images reconstructed by the network with these initial parameters.
After the tuning part, the network was instead retrained with these parameters:

Optimizer	LR	Weight decay	Momentum	N_channels	Encoded_space_dim
Adam	0.00436	2.814e-6	0.7902	9	36

Table 2: Best set of parameters

With this set of parameters the performances were significantly improved in terms of test loss, which was equal to 0.00917. Moreover, as one can observe in Figure (4) with respect to Figure (3), the reconstruction capability was enhanced, with more defined images and more evident edges and contours. This means that the CAE was able to detect the most important features of the input images, and then use them to generate new samples of a given class.

3 Fine tuning for classification

In this part of the homework it was required to resort to transfer learning techniques in order to create a classifier starting from the CAE previously implemented. As usual, the methods and results are reported below.

3.1 Methods

The structure of this deep network consisted of two separate sequences of layers. The first one was the encoder model of the previous architecture that, as previously mentioned, was employed in this classifier. The second part was instead a sequence of linear layers to perform the classification task, and it was constructed in the following way:

- A fully connected layer with *in_features=encoded_space_dim* (in this case 36) and with *out_features*=64
- Another fully connected layer with *in_features*=64 and with *out_features*=10, namely one for each of the classes of the dataset.

As for the previous architecture, also this one was implemented using a *Pytorch-lightning* module.

The performanceswere instead measured by means of the *Accuracy* module from the library *Torch-Metrics*.

3.2 Results

After the training of the network, the accuracy metric of this classifier resulted to be of 0.87, hence 87%. It is not a considerably high score, and with respect to the previous homework it is lower (91%), however it has to be considered that this classifier comes from an unsupervised architecture (the previous CAE), and only the second part is labelled, hence it can be actually considered a discrete result by taking into account that fact.

4 Network and latent space analysis

In order to explore how the network learned from the images and detect the more relevant features that would be successively used to generate new data, it is useful to have a visual representation of the latent space, namely the space that lies between the encoder and the decoder, in which the features detected in the first part of the network are saved. Two approaches were employed to perform such task: Principal Component Analysis (PCA) and t-distributed Stochastic Neighbour Embedding (t-SNE).

4.1 Methods

The two techniques mentioned before are dimensionality reduction techniques, which aim to reduce the size of variables while keeping as much information as possible. In detail we have that:

- PCA is a statistical process that maps the observations of correlated features into a set of linearly uncorrelated features by means of an orthogonal transformation. These new transformed features are called the principal components. PCA allows to emphasize strong patterns in the dataset by reducing the variances.

- t-SNE is a nonlinear dimensionality reduction technique which allows to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, hence with different initializations we can get different results and the same initialization can not be applied to different datasets.

Both techniques are composed of a first step of data normalization provided by the method *fit*, and then the method *transform* is used to effectively map the data into a lower dimension space.

4.2 Results

The results of the application of the two techniques to a random sample of the test dataset are shown in Figures (5) and (6).
The PCA does not give a completely clear representation, however there are two main clusters that can be recognized in the Figure. The smaller one on the left contains data points belonging to the

classes *['boot','sandal','sneaker']*, therefore it is clear that the encoder network was able to learn and cluster together the features of general footwear articles. The bigger one on the right, instead, is more mixed but also in this case it contains all the articles related to clothes to be worn such as shirts, t-shirts, dresses, coats. Note also that the lowest points at the margins of such cluster are the one belonging to the trousers, hence related to the lower body part. Finally, it is also interesting to observe that the class *bag* (the pink one), which is the only one that is not very much related to the others, is the most spread one across the latent space representation, meaning that the network found difficult to associate it to some already formed clusters.
On the other hand, the representation by means of the t-SNE technique is more understandable. Most of the classes are well clustered and defined, however there are some overlappings in the central and lower part of the figure. The former is given by the ambiguity of the *shirt* and *t-shirt* classes, while the latter, which is less evident, contains the *pullover* and *coat* labels, hence articles always somehow related.
To conclude, PCA gave insights about the feature extraction capability of the network while t-SNE analysis allowed to recognize its classification properties starting from unlabelled data.

5 Generative Adversarial Network

In the last part of the homework a GAN was implemented with the aim of training a network able to generate new samples from scratch.

5.1 Methods

The dataset employed in this task is once again the *FashionMNIST* so that it is easier to make comparisons between the different strategies.
The GAN architecture is structured into two different models, namely a *Discriminator* and a *Generator*. The discriminator is a classifier that has the role of classifying an image as *real* or *fake*. The generator, instead, generates new examples starting from random vectors. The basic idea behind GANs is that of training the generator to be able to fool the discriminator. The two networks work as two opponents: on one side the generator creates images that are given to the discriminator which, on the other side, classifies them as real or fake. The goodness of the model grows as the discriminator is not able anymore to classify correctly the image.
The discriminator was constructed as follows:

- A fully connected layer with *in_features=image_size* (28x28=784) and with *out_features*=256
- Another fully connected layer with *in_features*=256 and with *out_features*=256
- The last fully connected layer with *in_features*=256 and with *out_features*=1, since a binary output is needed (real or fake)

All the layers are activated by a *LeakyReLU* function with negative slope a=0.2, and the last one by a *Sigmoid*.
The generator, on the other hand, consisted of:

- A fully connected layer with *in_features*=*latent_size* (64, which is the input used to generate new samples) and with *out_features*=256
- Another fully connected layer with *in_features*=256 and with *out_features*=256
- The last fully connected layer with *in_features*=256 and with *out_features*=*image_size* (28x28=784), since the output is the generated image.

All the layers are activated by a *ReLU* function, and the last one by a *Tanh*, in order to bound the activation and hence learn more quickly.
The training for these models is a bit different from the standard networks, hence it is worth to mention it. In the discriminator, two losses are computed: in the first one the outputs are associated with label *1* to compute the loss for real images; in the other one some random vectors are generated and associated with label *0*, so that the loss is computed for fake images. Then, the two losses are combined together and the backpropagation step is done as always.
On the other hand, the generator is trained to learn to fool the discriminator by generating a fake image, but such image gets labelled as if it were a real one. The generator loss exploits the discriminator to try to make it classify the fake image as real.
The aim is to ensure that the loss of the generator reduces overtime, namely that it learns to create better images over epochs, and in the mean time the discriminator must not have an higher score than the other.

5.2 Results

The two networks were trained for 200 epochs.
The discriminator was trained with the *Binary Cross Entropy Loss* to quantify the classification efficiency, and for both the models the *Adam* optimizer with *learning_rate*=1e-4 was used to improve gradient descent. In Figure (7a) the losses of the discriminator and the generator are shown. As expected, the loss of the generator lowered over time, meaning that it was able to learn to generate image more and more similar to real images. The discriminator, instead, started from a very low loss value, since at the beginning it was easy to determine whether an image was fake while, as the generator learned how to create more realistic images, such loss went up. In Figure (7b), the scores of the discriminator are plotted. Coherently with the analysis done until now, the scores at the beginning of the training were very high for real images and almost null for fake images, but as the generator was learning it is evident how the discriminator made increasing mistakes, converging to a value of 0.5 meaning that it would guess the authenticity of the generated image with probability of $\frac{1}{2}$.
In Figure (7) it is possible to appreciate how the generator is learning over different epochs. At the beginning it is pure noise generated from random vectors, but over time the articles in the images are more and more defined and difficult to distinguish from real images.

6 Appendix

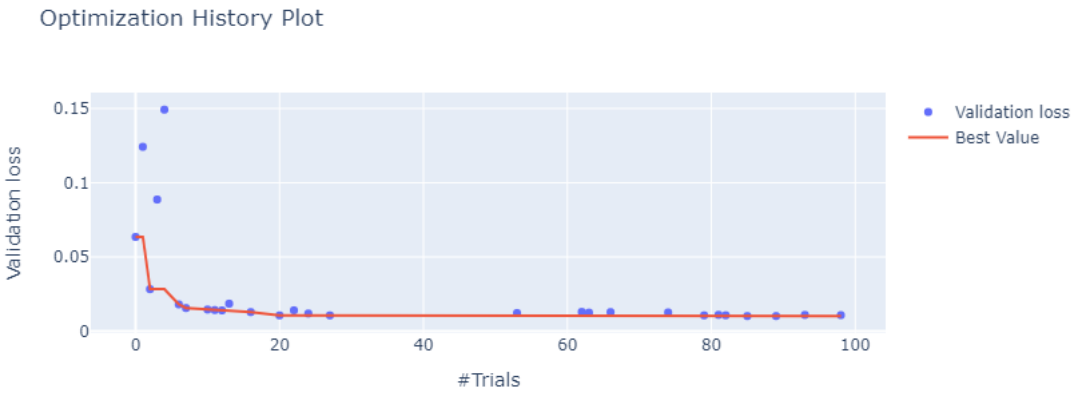


Figure 1: Optimization history plot of the random search with Optuna

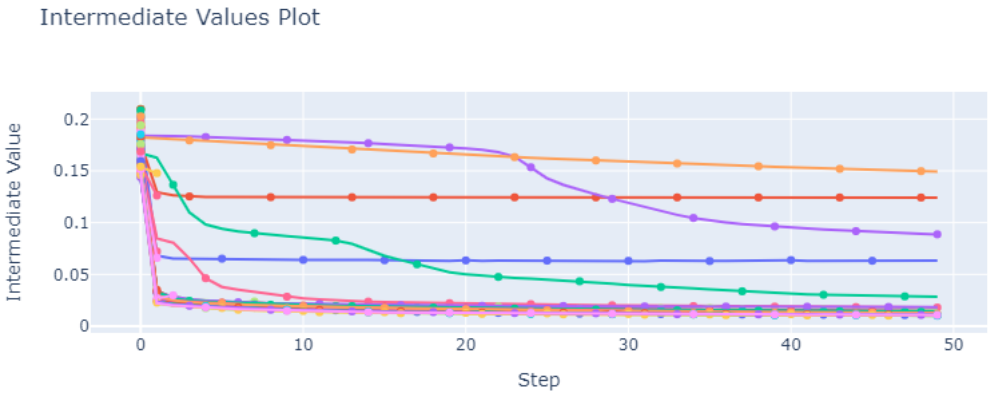


Figure 2: Intermediate values plot of the random search with Optuna

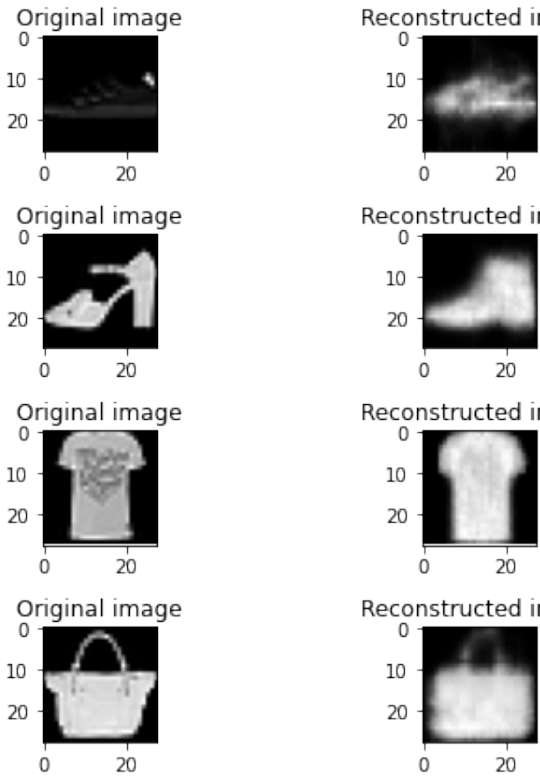


Figure 3: Samples generated by the first CAE

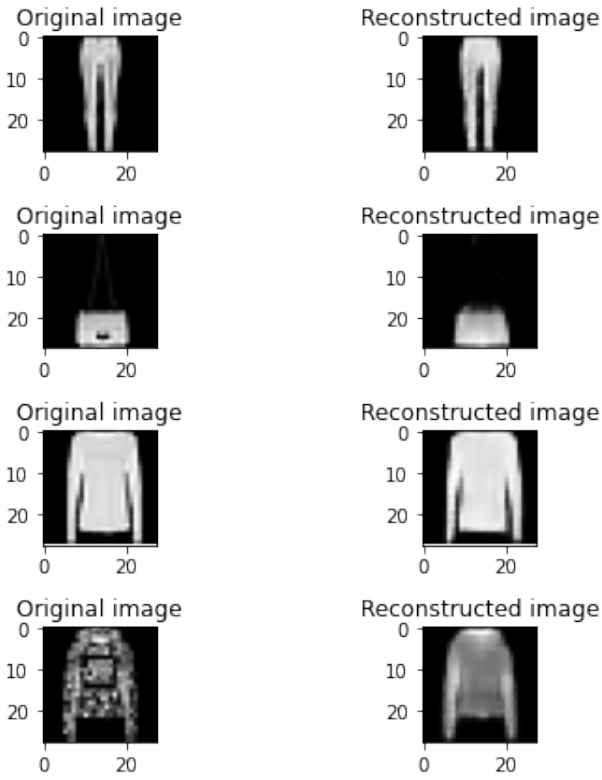


Figure 4: Samples generated by the best CAE

Figure 5: Latent space with PCA

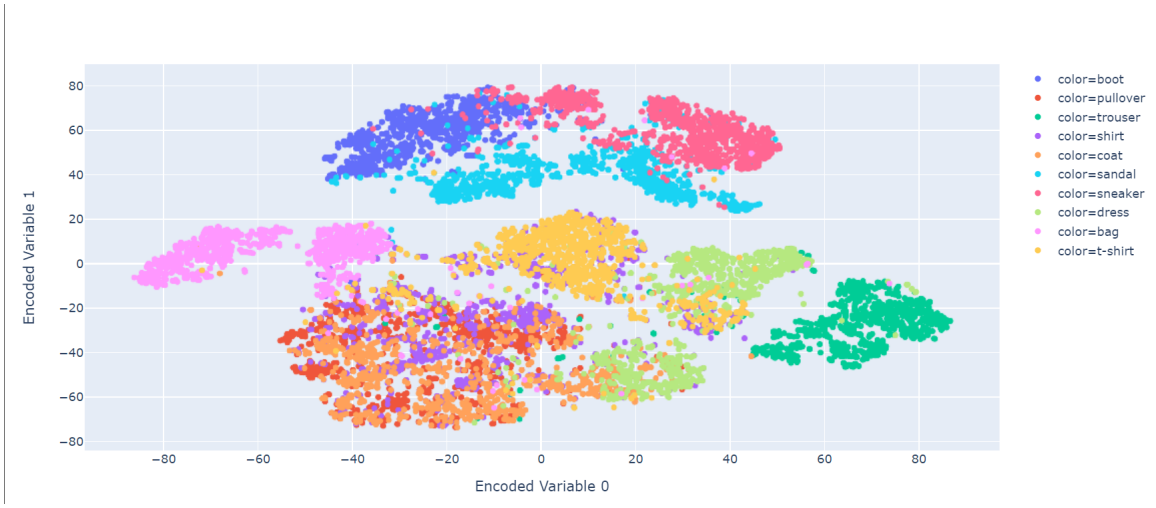
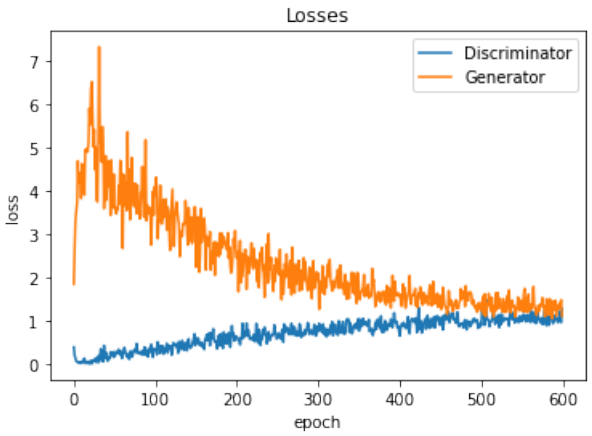
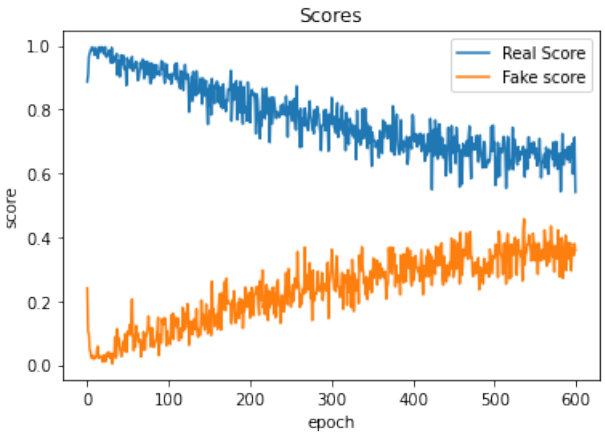


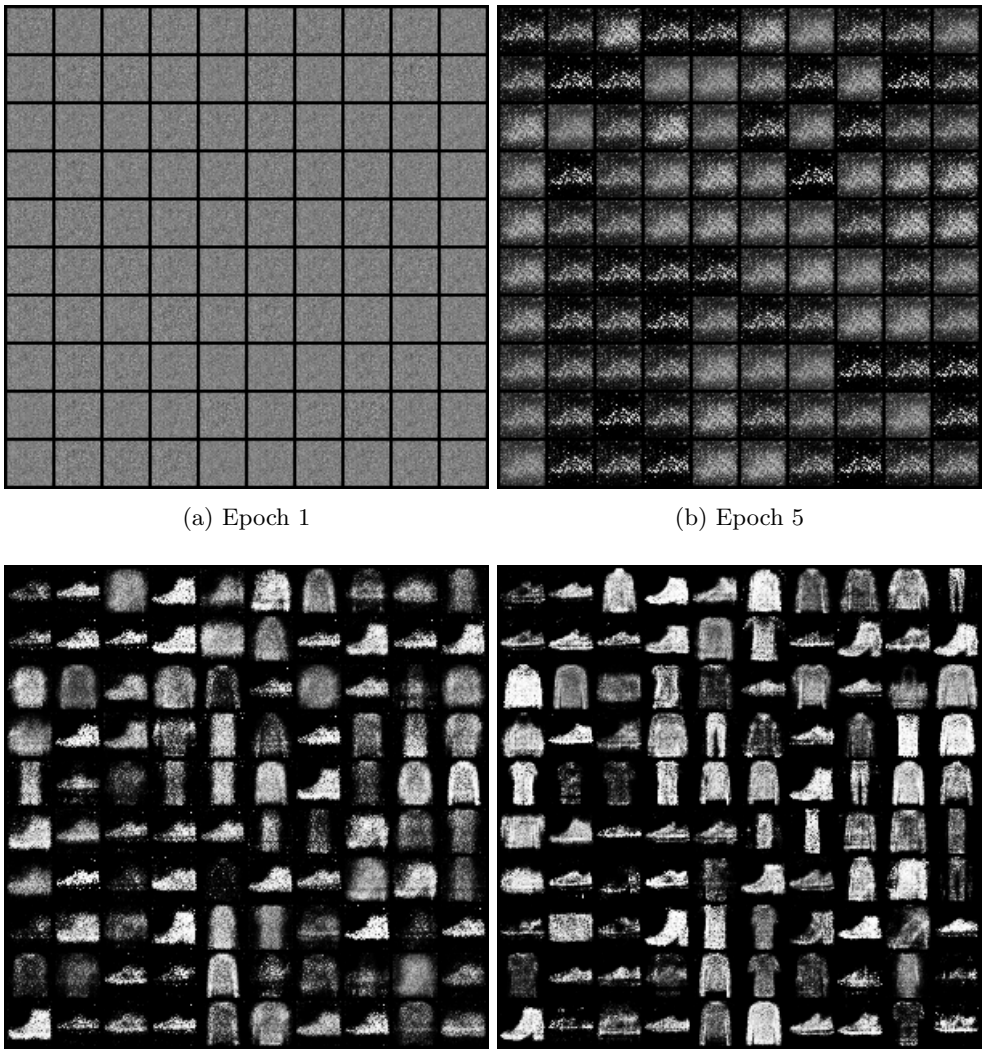
Figure 6: Latent space with t-SNE



(a) Plot of the losses in the GAN



(b) Plot of the scores of the real and fake images



(a) Epoch 1

(b) Epoch 5

(c) Epoch 50

(d) Epoch 200

Figure 7: Samples generated by the GAN at epochs 1, 5, 50, 200