TME #8: Generative Adversarial Networks

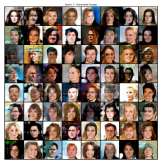
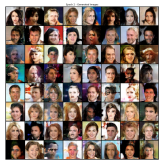
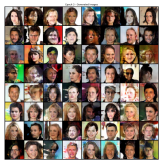
In this TME, we experiment with Generative Adversarial Networks (GANs) on two datasets: the CelebA - a set of photo portraits of celebrities - and the famous MNIST dataset. All experiments were made directly on the cloud with the given Google Colab Notebook.

The GANs models are made of 2 modules: the Generator G takes a noise z living in a latent space and outputs an image, the Discriminator D takes as input an image and outputs a scalar between 0 and 1: corresponding to the probability of the input image to be from the data distribution p_{data} rather than from the generated one p_g.

We start by proposing a training loop for the given Generator and Discriminator modules. The maximization of $log(D(x)) + log(1 - D(G(z)))$ - with regards to D - and of $log(D(G(z)))$ - with regards to G, are equivalent to the minimization of a cross-entropy loss between the output of D and the true labels 1 and 0: the real probabilities for the real (resp. generated) image to be from the data distribution.

We keep the notebook's default batch size of 128. For the optimization algorithm, we follow the recommendations of Radford et al. in the DCGAN paper, and use the Adam optimizer with parameters $\beta_1 = 0.5$ and $\beta_2 = 0.999$, and a learning rate of 0.002.

We launch the training loop for 5 epochs, and look at the output of the generator epoch after epoch:

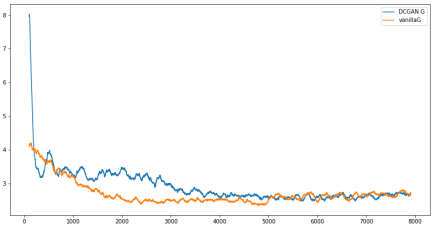
Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
				

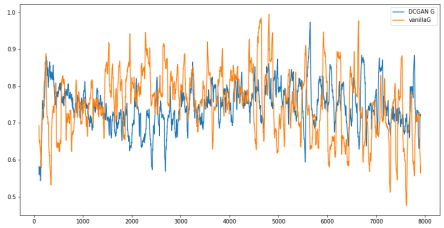
After 5 epochs, the generated images are still fuzzy. We however see a clear progress during the training, with the generated faces becoming more precise and realistic epoch after epoch.

DCGAN

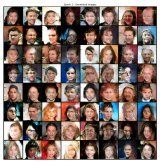
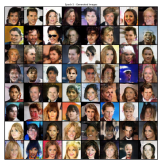
We implement the DCGAN generator for the CelebA dataset (see the `DCGANGenerator` class). That architecture adds a step of projection of z from the latent space into an intermediate 3D tensor with 1024 channels. We implement this projection step with a simple linear layer, followed by a step of "reshaping".The DCGAN Generator thus obtained has more than 12K parameters, almost 4 times more than our first vanilla generator with 3,5K parameters. We keep the same training loop and and qualitatively follow the evolution of the training.

We can compare the evolution of the loss of our DCGAN architecture, compared to our first vanilla one:





The generator now takes longer to converge, but the generated images are of better quality:

Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
				

MNIST Dataset

We adapt the Generator and Discriminator to the dimensions of the MNIST dataset. Because the MNIST dataset is a relatively simple dataset to learn, we use as a basis the first simple Generator / Discriminator architectures rather than the DCGAN. We remove one convolutional block from both modules, and adapt the respective paddings to a batch of (1 x 28 x 28) images. Our generator now has 1,066,880 parameters and our discriminator 661,248 parameters ! We decrease the batch size to 64 for more frequent updates. To train our model, we use the native `torchvision.datasets.MNIST` Dataset, with a normalization transformation of mean and standard deviation (0.5,0.5).

Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5