Exercise Sheet 8: Generative Adversarial Networks

Due on 21.06.2024, 10:00

Nick Stracke (nick.stracke@lmu.de)

General Information:

All programming exercises must be completed in Python. The proposed solution for the exercises will be compiled in Python 3 (3.8 or above). You may use standard Python libraries to complete your exercises. For the deep learning exercises, we will be using PyTorch[1].

Put your code and your report (a single PDF file) inside a single ZIP file. Both PDF file and ZIP file should contain your surname and your matriculation number (e.g.Surname-MatriculationNumber.zip). *You may export the PDF file from a Jupyter notebook, but please additionally also submit an executable Python file (.py) to ease grading.*

If you have any problems or questions about the exercise, you are welcome to use the dedicated forum on Moodle (most likely others share the same question). For technical issues about the course (for example if, for some reason, you cannot upload the solution to Moodle) you can write an email to the person responsible for the exercise (indicated at the top of the exercise sheet).

Grading:

It is possible to obtain points for a task even if the solution is incorrect. However, the submitted solution must show that the necessary effort was invested to complete the task.

[1] `https://pytorch.org/`

Task 1: Optimal Discriminator and Generator (6P)

In your lecture, the minmax game objective for adversarial learning was defined as follows:

$$V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log\left(D\left(x\right)\right)\right] + \mathbb{E}_{z\sim p(z)}\left[\log\left(1 - D\left(G\left(z\right)\right)\right)\right] \qquad (1)$$

The goal of the first task is to derive the optimal discriminator and generator from scratch.

1. Use equation (1) as a starting point to derive the optimal discriminator D^* in terms of data probability $p_{data}\left(x\right)$ and generator probability $p_G\left(x\right)$. Assume generator G is fixed.

2. Use the obtained D^* to find the optimal point minimizing V. What value does D^* have at this point and what would this value imply?

Please include all necessary steps to get to the final results in your report.

Note:
If you're stuck, take a look at the seminal paper Generative Adversarial Nets by Goodfellow et al [2].

[2] `https://arxiv.org/pdf/1406.2661`

Task 2: Training a GAN (14P)

In this task, you should implement a Generative Adversarial Network (GAN) from scratch and train it using the Fashion-MNIST dataset[3]. The generator should receive a 100-dimensional normal distributed random code which is then projected onto a 32×32 image using convolutions. Use the DCGAN [4] architecture shown in figure 1. The discriminator on the other hand should receive the image and output a label indicating whether the provided image is real or fake/generated. Both networks should additionally receive the class label to achieve class-conditional synthesis. Use equation (1) as the training signal and be sure you take care of the vanishing gradient problem.

Your report should include:

- A figure containing the evolution of training losses for discriminator and generator
- A figure containing the classification accuracy for discriminator on real and fake samples per epoch
- A few randomly synthesized images from your final generator model
- Use a single latent code and show the evolution of its corresponding image during training
- A t-SNE[5] embedding of images from train split, test split, and your generator for at least 5 classes of your choice. Take around 500 images each. For the t-SNE embedding, you should use features from a pretrained classification network on real Fashion-MNIST data. This means you need to train an extra classification CNN on Fashion-MNIST data. Afterward, you can fix the parameters of the network and take the output from an intermediate layer (suggestion: layer before classification head) to get features for the corresponding input image.
- The classification accuracy on generated data using the classifier from the previous step
- Your python code

[3] `https://pytorch.org/vision/stable/generated/torchvision.datasets.FashionMNIST.html#torchvision.datasets.FashionMNIST`

[4] `https://arxiv.org/pdf/1511.06434`

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`

Hints:

- Use Adam optimizer with a learning rate of 0.0001 and $\beta_1 = 0.5$.
- Fashion-MNIST has a standard resolution of 28×28 so make sure to resize it to 32×32.
- You may have to adapt the DCGAN architecture slightly to generate 32×32 images instead of 64×64.
- Using spectral normalization[6] on the weights of the Discriminator can help with mode collapse and make training more stable.
- You should already see decent results after a couple of epochs.

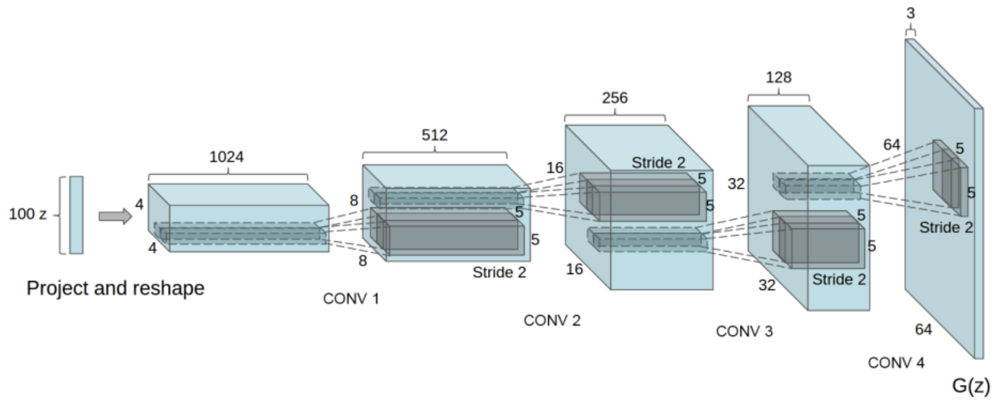

Figure 1: The DCGAN generator architecture. The latent code is linearly projected and reshaped to a $4x4$ grid. Afterward, transposed convolutions are applied for upsampling. Note that after every transposed convolution, there is a BatchNorm layer followed by a LeakyRelu activation function. Especially the existence of the BatchNorm layer is crucial. A similar but reversed architecture can be used for the discriminator.

Important Note: Submit exactly one ZIP file via Moodle before the deadline. The ZIP file should contain your executable code and your report in PDF format. Make

[6]`https://pytorch.org/docs/stable/generated/torch.nn.utils.parametrizations.spectral_norm.html#torch.nn.utils.parametrizations.spectral_norm`

sure that it runs on different operating systems and use relative paths. Non-trivial sections of your code should be explained with short comments, and variables should have self-explanatory names. The PDF file should contain your written code, all figures, explanations and answers to questions. Make sure that plots have informative axis labels, legends, and captions. Missing plots will result in point reduction.