Fall 2024
Introduction to Deep Learning 18780/6

Homework 4: Generative Adversarial Networks

18-780 - *Due: Sunday, March 9, 2025 11:59 pm*
18-786 - *Due: Friday, March 14, 2025 11:59 pm*

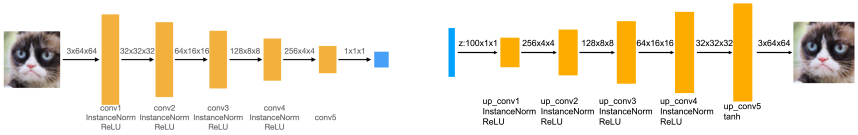
Submitting your work: For all problems on this homework we will be using **Gradescope.** To receive full credit, you need to submit both the PDF solution and a zip file containing your code solution. Please be super clear, neat, and complete when you write your solutions. **If the TAs cannot follow your work, you will not receive full credit for that problem.**

Collaboration Policy: Homework will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in figuring out answers and helping each other solve the problems. We will be assuming that, as participants in a graduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration. You must also indicate on each homework with whom you collaborated.

In this assignment, you will explore hands-on experience in coding and training Generative Adversarial Networks (GANs) Goodfellow et al. [2014]. This homework consists of two parts: the first part includes the implementation of Deep Convolutional GAN (DCGAN) Radford et al. [2016] and training it to generate grumpy cats images; the second part explores different architectures and training objectives of DCGAN; In both parts, you will gain experience implementing GANs by writing code for the generator, discriminator, and training loop, for each model. In all parts of this homework, you will gain experience implementing GANs, designing its architecture and objectives, and training loop of GANs.

Part 1: Deep Convolutional GAN (30 points)

For the first part of this assignment, we will implement a slightly modified version of Deep Convolutional GAN (DCGAN). A DCGAN is simply a GAN that uses a convolutional neural network as the discriminator, and a network composed of transposed convolutions as the generator. **In the assignment, instead of using transposed convolutions, we will be using a combination**



(a) Discriminator (b) Generator

Figure 1: DCGAN model architecture. (a) Architecture of the discriminator. (b) Architecture of the generator.

of an upsampling layer and a convolutional layer to replace transposed convolutions. To implement the DCGAN, we need to specify three things: 1) the generator, 2) the discriminator, and 3) the training procedure. We will develop each of these three components in the following subsections.

1.1 Implement the Discriminator (5 Points)

The (modified) discriminator of DCGAN consists of a series of convolutional layers, batch/instance normalization layers, and ReLU activation function, as shown in fig:discriminator,

Implement this architecture by filling in the *init* function and *forward method* of the *DCDiscriminator* class in models.py, shown below. The *conv_dim* argument does not need to be changed unless you are using larger images, as it should specify the initial image size.

1.2 Implement the Generator (5 Points)

Now, we will implement the generator of the DCGAN, which consists of a sequence of upsample+convolutional layers that progressively upsample the input noise sample to generate a fake image. The generator in this DCGAN has the architecture as shown in fig:generator.

Implement this architecture by filling in the *init* and *forward method* of the DCGenerator class in models.py. Note: Use the *up_conv* function (analogous to the conv function used for the discriminator above) in your generator implementation. We find that for the first layer (*up_conv1*) it is better to directly apply convolution layer without any upsampling to get 4x4 output. To do so, you'll need to think about what you kernel and padding size should be in this case. Feel free to use *up_conv* for the rest of the layers.

1.3 DCGAN Training Loop (20 Points)

Next, you will implement the training loop for the DCGAN. A DCGAN is simply a GAN with a specific type of generator and discriminator; thus, we train it in exactly the same way as a standard GAN. The pseudo-code for the training procedure is shown in alg:dcgan. The actual implementation is simpler than it

may seem from the pseudo-code: this will give you practice in translating math to code.

Algorithm 1: DCGAN algorithm

[1] Draw m training samples $\{x_0, x_1, ..., x_m\}$ from data distribution p_{data}. Draw m noise samples $\{z_0, z_1, ..., z_m\}$ from noise distribution p_z. Generate fake images from the noise: $G(z_i)$ for $i \in \{1, ..., m\}$. Compute the discriminator loss. Update the parameters of the discriminator. Draw m new noise samples $\{z_0, z_1, ..., z_m\}$ from noise distribution p_z. Generate fake images from the noise: $G(z_i)$ for $i \in \{1, ..., m\}$. Compute the generator loss. Update the parameters of the generator.

1.3.1 Implementation (5 Point)

Fill the *training_loop* part in vanilla_ganṗy. There are 4 numbered bullets in the code to fill in for the discriminator and 3 bullets for the generator. Each of these can be done in a single line of code, although you will not lose marks for using multiple lines.

DCGAN will perform poorly without data augmentation on a small-sized dataset because the discriminator can easily overfit to a real dataset. To rescue, we need to add some data augmentation such as random crop and random horizontal flip. You need to fill in advanced version of data augmentation in data_loaderṗy. We provide some script for you to begin with. You need to compose them into a transform object which is passed to CustomDataset.

1.3.2 Training and Visualization (15 Point)

Train DCGAN by running the completed *vanilla_ganṗy*. Provide your training hyper-parameters. Show a grid of generated cats images using basic augmentation and advanced augmentation for the beginning of training (e.g. 200 or 400 iterations) and the end of training (e.g. 3000 or 6000 iterations) here respectively. Also plot the training loss correspondingly. (In total 6 images are expected to shown here.)

Part 2: Architectures and Objective Functions (20 points)

18-780 students need not do this, but will receive +5 pts extra credit for each implemented

In this section, we will explore important works that have gone into improving the training and image quality of GANs. Some changes are architectural, some are in the loss function, and some focus on other elements. We are asking you to implement these rather famous and important techniques. Some of these may not be familiar, but the TAs will be there to help and there are plenty of online resources explaining these in detail. **You must implement these**

yourself and may not use pytorch functions that will trivialize these such as torch.nn.utils.spectral_norm

- Implement Spectral Normalization Miyato et al. [2018]
- Implement Wasserstein GAN Arjovsky et al. [2017]
- Implement Least Squares GAN Mao et al. [2017]
- (Optional) Implement upto 3 GAN training techniques and link a paper below for each. (+5 pts extra credit for each).

For each of the above, show a grid of cat images.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018. URL `https://arxiv.org/abs/1802.05957`.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2017.