National University of Sciences and Technology

School of Electrical Engineering and Computer Science

Department of Computing

CS 405 Deep Learning

Fall 2023

# 

# Lab 12

**Generative Adversarial Networks(GAN) - PyTorch Tutorial**

**Name: Alina Nasir**

**Qalam ID: 342350**

Date: 25th Nov 2023

**Question 1: What does the latent\_dim variable used for in the cell below?**

**Answer:** The latent space is a lower-dimensional space that captures essential features or representations of the input data. In the case of VAEs or GANs, this latent space is often referred to as the "latent variable space" or simply "latent space." The idea is to learn a compact and meaningful representation of the input data in this lower-dimensional space. In the provided code snippet, latent\_dim = 100 indicates that the latent space has 100 dimensions which is a known standard.

**Question 2: Please explain the architecture of the Generator part of our GAN in the cell below. Is the Generator convolutional or fully-connected?**

**Answer:** The generator architecture is based on fully-connected layers with batch normalization and leaky ReLU activations, and it transforms a random noise vector (z) into a synthetic image. The Tanh activation is used to ensure that the generated pixel values are within a specific range. The following is the architecture of the generator:

* **Input Layer (Latent Vector):**

The generator starts with an input layer that takes a random noise vector z of size latent\_dim (e.g., 100 in this case). This vector is a random input sampled from the latent space.

* **Hidden Layers:**

The generator uses a series of fully-connected layers to transform the input noise vector into a higher-dimensional representation. Each hidden layer is implemented by the block function, which consists of a linear layer (fully-connected) followed by optional batch normalization and leaky ReLU activation.

* **Output Layer:**
* The last layer of the generator is a linear layer that transforms the high-dimensional representation into a vector of size equal to the product of the image shape (**img\_shape**). In this case, the generator seems to be designed for generating images of size 28x28 (based on **img\_shape**).
* The output of the linear layer is passed through a hyperbolic tangent (Tanh) activation function, which squashes the values to the range [-1, 1]. This is often used in GANs to generate images with pixel values in the same range.
* **Reshaping:**

Finally, the output is reshaped to match the desired image shape. In this case, the generator reshapes the output to have dimensions (batch\_size, 1, 28, 28), where batch\_size is the number of samples generated at once.

**Question 3: Please explain the architecture of the Discriminator part of our GAN in the cell below. Is the discriminator convolutional or fully-connected?**

**Answer:** The Discriminator class in the provided code is responsible for discriminating between real and fake images. the discriminator architecture is based on fully-connected layers with leaky ReLU activations, and it takes a flattened image as input to output a probability indicating the likelihood that the input is a real image. The architecture of the discriminator is given below:

1. **Input Layer (Flattened Image):**

The discriminator starts with an input layer that takes a flattened image as input. The input size is equal to the product of the image shape (img\_shape). In this case, it's designed for images of size 28x28, so the input size is 1 \* 28 \* 28.

1. **Hidden Layers:**

The discriminator uses a series of fully-connected layers with leaky ReLU activations. Each hidden layer is implemented using a linear layer followed by leaky ReLU activation. The structure of the hidden layers is as follows:

* Input: 1 \* 28 \* 28 (flattened image)
* Output: 512
* Input: 512
* Output: 256

1. **Output Layer:**

The last layer of the discriminator is a linear layer that reduces the dimensionality to 1, making a binary decision between real and fake. The output of this layer is then passed through a sigmoid activation function, which squashes the output to the range [0, 1]. This final output represents the probability that the input image is real.The structure of the output layers is as follows:

* Input: 256
* Output: 1

1. **Activation Functions:**

Leaky ReLU activation with a slope of 0.2 is used for the hidden layers to introduce a small negative slope to the activation function, allowing some information to pass even when the input is negative. This can help with training stability.The final layer uses a sigmoid activation function to squash the output into the range [0, 1], representing the probability of the input image being real.

1. **Flattening and Forward Pass:**

In the forward method, the input image is flattened using view before being passed through the discriminator model. The result is the discriminator's decision (validity) regarding whether the input image is real or fake.

**Question 4: Why is the adversarial loss binary cross-entropy used here?**

**Answer:** The Binary Cross-Entropy (BCE) loss is commonly used in a Generative Adversarial Network (GAN) when dealing with binary classification problems, since were are distinguishing between real and generated (fake) samples we have used Binary Cross-Entropy Loss.

**Question 5: Answer the following questions in the cell below:**

**(i) What are the "valid" and "fake" variables initialized for?**

**Answer:** The "valid" and "fake" variables are used as ground truth labels for the discriminator during training. In the context of a GAN, the discriminator is trained to distinguish between real and fake samples. These labels are essential for computing the adversarial loss, which is a binary classification problem.

The “valid” variable is initialized with a tensor filled with the value 1.0, representing the label for real samples. In binary classification as in our problem, 1 denotes the positive class (real), indicating that the discriminator should classify the input as a real sample whereas 0 denotes negative class fake therefore “fake” variable is initialized with a tensor filled with the value 0.0, representing the label for fake samples.

**(ii) What does "real\_imgs = imgs.type(Tensor)" do and why are we doing it?**

**Answer:** This converts the data type of the input images (**imgs**) to the data type specified by the **Tensor** variable. This conversion is necessary to ensure that the data type of the input images matches the data type expected by the model. In PyTorch, the model and the input data should have the same data type for proper computation and backpropagation. By converting imgs to the data type specified by Tensor, the code ensures consistency in data types.

**(iii) What is the tensor z used for in the line: z = Tensor(np.random.normal(0, 1, (imgs.shape[0],latent\_dim)))?**

**Answer:** The tensor z is a random noise vector sampled from a Gaussian distribution, and it serves as the input to the generator during the training of the GAN. Each row of z corresponds to a different sample in the batch, and each column represents a different dimension in the latent space.

* The normal() function generates random values from a normal (Gaussian) distribution with a mean of 0 and a standard deviation of 1. The resulting tensor has a shape of (batch size, latent dimension).
* The Tensor() converts the NumPy array to a PyTorch tensor of the specified data type (assumed to be torch.FloatTensor).

The generator takes this random noise vector as input and attempts to generate realistic images, while the discriminator then tries to distinguish between these generated images and real images.

**iv) In the code below is the Generator updated first or the discriminator? Answer:** The generator is updated before the discriminator because in the code the generator is optimized through gradient descent before the dicriminator.

**v) For discriminator you can see it is a sum of two losses "real\_loss" and "fake\_loss". What do these two losses signify?**

The “real\_loss” measures how well the discriminator classifies real images. It is the binary cross-entropy loss between the discriminator's predictions for real images and the ground truth labels for real images. The “fake\_loss” measures how well the discriminator classifies generated (fake) images. It is the binary cross-entropy loss between the discriminator's predictions for fake images, where **detach()** is used to prevent backpropagation through the generator and the ground truth labels for fake images.

**Question 6 How many images did the Generator produce?**

The generator produced 5 images for each batch, resulting in total 20 images.