Assignment 3

1 Practice: GAN Architecture

As part of this assignment you will implement a GAN using PyTorch that matches the following architecture:

Generator

- Input: Noise vector of shape $(\text{BATCH\_SIZE}, 100)$
- Fully connected layer to $7 \times 7 \times 128$, then reshape
- ConvTranspose2D: $128 \rightarrow 64$, kernel size 4, stride 2, padding 1 $\Rightarrow$ output size 14×14
  - Followed by BatchNorm2D and ReLU
- ConvTranspose2D: $64 \rightarrow 1$, kernel size 4, stride 2, padding 1 $\Rightarrow$ output size 28×28
  - Followed by Tanh activation

Discriminator

- Input: Image of shape $(1, 28, 28)$
- Conv2D: $1 \rightarrow 64$, kernel size 4, stride 2, padding 1 $\Rightarrow$ output size 14×14
  - Followed by LeakyReLU(0.2)
- Conv2D: $64 \rightarrow 128$, kernel size 4, stride 2, padding 1 $\Rightarrow$ output size 7×7
  - Followed by BatchNorm2D and LeakyReLU(0.2)
- Flatten and apply Linear layer to get a single output (real/fake probability)

2 Practice: Model Deployment

Use the MNIST dataset to train a GAN model to generate hand-written digits and add it to the API implemented as part of Module 6 Class Activity. Commit your code to GitHub.

3 Theory: Building Blocks of GAN

Answer the following questions.

Question 1

Given an input feature map of size 8×8, a transposed convolution with kernel size 4×4, stride 2, padding 1, and output padding 1, what will the output size be? Show your calculations.

Question 2

How does the output size of a transposed convolution change if you increase the stride from 2 to 3, keeping everything else fixed?

Question 3

Write the formula to compute output size for a 2D transposed convolution given input size I, kernel size K, stride S, padding P, and output padding OP.

Question 4

If a Conv2DTranspose layer is used to upsample a feature map from 16×16 to 32×32, what kernel size and stride could be used (assuming no padding)? Give one possible configuration.

Question 5

Given a mini-batch of 4 values: [6, 8, 10, 6], compute the normalized output using BatchNorm without γ and β (i.e., just zero-mean, unit-variance).

Question 6

What is the key mathematical difference between ReLU and LeakyReLU? Give their formulas

Question 7

Why might LeakyReLU be preferred over ReLU in deep networks?

1. $I = 8$, $k = 4$, $s = 2$, $p = 1$, $op = 1$

$$O = (I-1) \cdot s - 2p + k + op$$

$$O = (8-1) \cdot 2 - 2 + 4 + 1 = 17$$

output $= 17 \times 17$

2. $O' = (8-1) \cdot 3 - 2 + 4 + 1 = 24$

stride $\uparrow$ output $\uparrow$

3. General form: $O = (I-1) \cdot s - 2p + k + op$

$$\longrightarrow O = \frac{I - k + 2p}{s} + 1$$

4. $p = 0$, $op = 0$

$$32 = (16-1) \cdot s - 0 + k + 0$$

$$= 15s + k$$

if $s = 2 \longrightarrow k = 2$

5. $x = [6, 8, 10, 6]$

$\mu = \frac{6+8+10+6}{4} = 7.5$

$\sigma^2 = \frac{(6-7.5)^2 + (8-7.5)^2 + (10-7.5)^2 + (6-7.5)^2}{4}$

$= 2.25$

$\sigma = 1.5$

$z_i = \frac{x_i - \mu}{\sigma}$

$z = [-1, 0.33, 1.67, -1]$.

6. ReLU: $f(x) = \max(0, x)$

Leaky ReLU: $f(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases}$

α a small constant

7. ReLU blocks $\forall$ negative values, leaky allows small negative slopes.

ReLU can cause "dying neurons," where large negative inputs make gradients zero, and the neuron never activates again.
Leaky ReLU solves this by keeping a small nonzero slope (α), allowing gradients to flow even for negative inputs