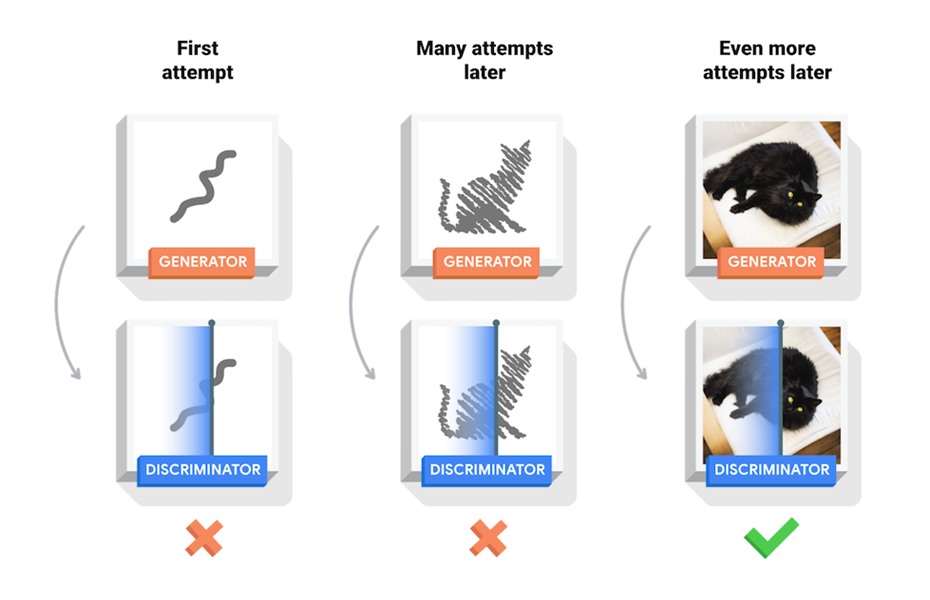
| **Ex No: 8**  **Date: 18/09/2024** | **DCGAN Implementation for MNIST Handwritten Digits** |
| --- | --- |

**Objective:**

The main objective is to implement a Deep Convolutional Generative Adversarial Network (DCGAN) to generate images of handwritten digits from the MNIST dataset. The generator network creates realistic-looking digit images from random noise, while the discriminator network tries to differentiate between real and generated (fake) images.



**Code Explanation for DCGAN:**

**def discriminator\_loss(real\_output, fake\_output):**

**# START YOUR CODE HERE**

**real\_loss = cross\_entropy(tf.ones\_like(real\_output), real\_output)**

**fake\_loss = cross\_entropy(tf.zeros\_like(fake\_output), fake\_output)**

**total\_loss = real\_loss + fake\_loss**

**# END YOUR CODE HERE**

**return total\_loss**

Explanation of the Code:

1. Function Definition:

**def discriminator\_loss(real\_output, fake\_output):**

This function takes two arguments:

- `real\_output`: the discriminator's prediction (output) when it is fed real images from the dataset.

- `fake\_output`: the discriminator's prediction when it is fed fake images generated by the generator.

2.Real Loss Calculation:

**real\_loss = cross\_entropy(tf.ones\_like(real\_output), real\_output)**

Goal: The discriminator should ideally classify real images as "real" (label 1).

- `tf.ones\_like(real\_output)` creates a tensor of ones with the same shape as `real\_output`, representing the true labels (real = 1).

- `cross\_entropy` computes the cross-entropy loss between the true label (`1`s) and the discriminator's actual prediction `real\_output`.

3. Fake Loss Calculation:

**fake\_loss = cross\_entropy(tf.zeros\_like(fake\_output), fake\_output)**

- Goal: The discriminator should classify fake images as "fake" (label 0).

- `tf.zeros\_like(fake\_output)` creates a tensor of zeros with the same shape as `fake\_output`, representing the true labels (fake = 0).

- `cross\_entropy` computes the loss between the true label (`0`s) and the discriminator's prediction `fake\_output`.

4. Total Loss Calculation:

**total\_loss = real\_loss + fake\_loss**

The total loss is simply the sum of the two losses: how well the discriminator classifies real images as real and fake images as fake.

5. Return the Loss:

**return total\_loss**

The function returns the total discriminator loss, which will be used to update the discriminator during training.

Dataset Preprocessing: The dataset used is the MNIST dataset, which consists of 60,000 grayscale images of handwritten digits. The images are normalized to a range between [0, 1] before being input to the network.

**DCGAN Architecture:**

1. Generator Architecture (make generator model):

* Initializes the generator model.
* Defines a dense layer to expand the random noise into a larger shape (7x7x256).
* Normalizes the activations to stabilize training.
* Adds a leaky ReLU activation to introduce non-linearity.
* Reshapes the output into a 7x7 feature map with 256 channels.
* Applies transposed convolution to upsample the feature map at each step.
* Adds activation after each upsampling step.
* Outputs the final image with tanh activation, producing values between -1 and 1.

2. Generator Image Creation:

* Generates random noise as input to the generator.
* Uses the generator to produce an image from the random noise.
* Displays the generated image in grayscale.

3. Discriminator Architecture (make discriminator model):

* Defines the convolutional layers to downsample the input image.
* Adds leaky ReLU activation to introduce non-linearity.
* Drops units randomly during training to prevent overfitting.
* Flattens the output to feed into a dense layer.
* Outputs a single value indicating the real or fake classification.

4. Discriminator Classification:

* Classifies the generated image as real or fake.
* Prints the discriminator's decision (positive for real, negative for fake).

5. Loss Functions and Optimizers:

* Defines the cross-entropy loss function for both the generator and discriminator.

6. Discriminator Loss (discriminator loss):

* Computes loss between real images and true labels (all ones).
* Computes loss between fake images and true labels (all zeros).
* Computes the total loss for the discriminator.

7. Generator Loss (generator loss):

* Computes the loss for the generator by comparing fake outputs to labels of ones (aiming to fool the discriminator).

8. Optimizers:

* Defines the Adam optimizer for the generator with a learning rate of 0.0001.
* Defines the Adam optimizer for the discriminator with the same learning rate.

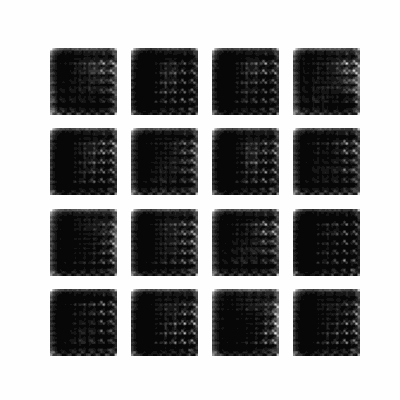
**Training Process:**

* The training involves alternating between updating the discriminator and the generator:
  + The discriminator is trained to correctly classify real images as real and fake images as fake.
  + The generator is trained to fool the discriminator by producing realistic images that the discriminator classifies as real.
* Both networks are optimized using the binary cross-entropy loss function and the Adam optimizer.

**Results**

The DCGAN successfully learns to generate realistic-looking handwritten digits from the MNIST dataset after sufficient epochs of training. Initially, the generated images appear as noise, but as the generator improves, the generated digits become sharper and more recognizable. The discriminator also learns to distinguish between real and fake images more effectively. The loss curves of both the generator and discriminator show the adversarial dynamic, where the generator tries to fool the discriminator and the discriminator improves in its classification. Over time, the generated images closely resemble real handwritten digits, demonstrating the effectiveness of the model.

**Result Analysis:**

****

**Summary:**

This notebook implements a Deep Convolutional Generative Adversarial Network (DCGAN) for generating handwritten digits from the MNIST dataset. By utilizing transposed convolutions in the generator and regular convolutions in the discriminator, the DCGAN framework learns to generate high-quality synthetic images that mimic real data. The adversarial training process ensures that both the generator and discriminator improve in their respective tasks. This notebook highlights the core principles of GANs, including the balance between the two networks and the gradual improvement in image quality, offering a practical demonstration of generative modeling.

**GitHub Link:**

<https://github.com/chandanab1/Deep_Learning>