**ASSESMENT-8**

**PROBLEM STATEMENT-** Applying Generative Adversial Networks for image generation and unsupervised tasks.

**SOURCE CODE:**

* Open the folder in the Spyder IDE
* The Command to install

**“pip install tensorflow matplotlib”**

**File Name: program8.py**

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, LeakyReLU, Input

from tensorflow.keras.optimizers import Adam

# Load and preprocess the dataset

(X\_train, \_), (\_, \_) = mnist.load\_data()

X\_train = X\_train / 127.5 - 1.0  # Normalize the images to [-1, 1]

X\_train = np.reshape(X\_train, (X\_train.shape[0], -1))

# Generator model

generator = Sequential([

    Dense(256, input\_dim=100),

    LeakyReLU(0.2),

    Dense(512),

    LeakyReLU(0.2),

    Dense(1024),

    LeakyReLU(0.2),

    Dense(784, activation='tanh')

])

# Discriminator model

discriminator = Sequential([

    Dense(1024, input\_dim=784),

    LeakyReLU(0.2),

    Dense(512),

    LeakyReLU(0.2),

    Dense(256),

    LeakyReLU(0.2),

    Dense(1, activation='sigmoid')

])

# Compile discriminator

discriminator.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])

# Combined model

discriminator.trainable = False

gan\_input = Input(shape=(100,))

generated\_image = generator(gan\_input)

gan\_output = discriminator(generated\_image)

gan = Model(gan\_input, gan\_output)

gan.compile(loss='binary\_crossentropy', optimizer=Adam(0.0002, 0.5))

# Training parameters

epochs = 30000

batch\_size = 128

sample\_interval = 1000

# Training the GAN

for epoch in range(epochs):

    # Select a random batch of real images

    idx = np.random.randint(0, X\_train.shape[0], batch\_size)

    real\_images = X\_train[idx]

    # Generate a batch of fake images

    noise = np.random.normal(0, 1, (batch\_size, 100))

    fake\_images = generator.predict(noise)

    # Train the discriminator

    d\_loss\_real = discriminator.train\_on\_batch(real\_images, np.ones(batch\_size))

    d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, np.zeros(batch\_size))

    d\_loss = 0.5 \* np.add(d\_loss\_real, d\_loss\_fake)

    # Train the generator

    noise = np.random.normal(0, 1, (batch\_size, 100))

    g\_loss = gan.train\_on\_batch(noise, np.ones(batch\_size))

    # Print progress

    if epoch % 100 == 0:

        print(f"Epoch: {epoch}, D Loss: {d\_loss[0]}, Acc.: {100\*d\_loss[1]}, G Loss: {g\_loss}")

    # If at sample interval, save generated image samples

    if epoch % sample\_interval == 0:

        # Plot generated images

        r, c = 5, 5

        noise = np.random.normal(0, 1, (r \* c, 100))

        gen\_imgs = generator.predict(noise)

        gen\_imgs = 0.5 \* gen\_imgs + 0.5

        fig, axs = plt.subplots(r, c)

        cnt = 0

        for i in range(r):

            for j in range(c):

                axs[i, j].imshow(gen\_imgs[cnt, :].reshape(28, 28), cmap='gray')

                axs[i, j].axis('off')

                cnt += 1

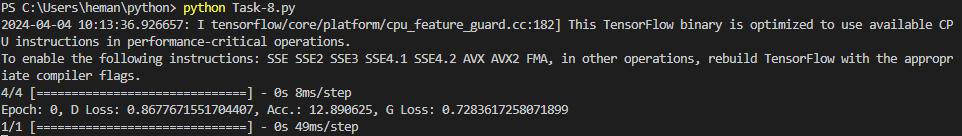
        plt.show()

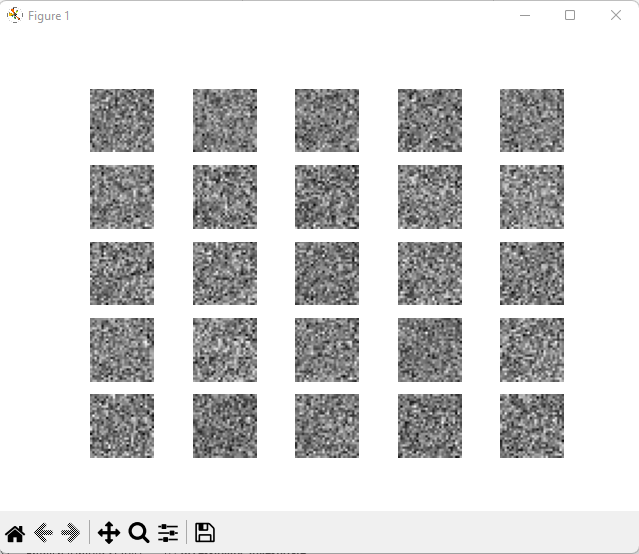
**Explanation:**

1. **Importing Libraries**: Import necessary libraries including NumPy for numerical operations, Matplotlib for plotting, and TensorFlow and Keras for building and training the neural network models.
2. **Loading and Preprocessing the Dataset**: Load the MNIST dataset using **mnist.load\_data()** function from Keras. Preprocess the dataset by normalizing the pixel values to the range [-1, 1] and reshaping the images to a flat vector format.
3. **Generator Model**: Define the generator neural network model using Keras **Sequential** API. It consists of fully connected layers with LeakyReLU activation functions, aiming to transform random noise into realistic-looking images.
4. **Discriminator Model**: Define the discriminator neural network model using Keras **Sequential** API. It also consists of fully connected layers with LeakyReLU activation functions, aiming to distinguish between real and fake images.
5. **Compiling the Discriminator**: Compile the discriminator model using binary cross-entropy loss and Adam optimizer.
6. **Combined Model (GAN)**: Combine the generator and discriminator into a GAN. In this step, the discriminator weights are frozen to prevent it from being trained during the generator training process.
7. **Compiling the GAN**: Compile the GAN model using binary cross-entropy loss and Adam optimizer.
8. **Training Parameters**: Define the number of epochs, batch size, and sample interval for training.
9. **Training the GAN**: Iterate over epochs and train the GAN model. In each epoch:
   * Randomly select a batch of real images from the dataset.
   * Generate a batch of fake images using random noise as input to the generator.
   * Train the discriminator using both real and fake images.
   * Train the generator by generating fake images and passing them through the discriminator, aiming to fool it.
   * Print the loss values periodically.
   * Visualize a grid of generated images at specified intervals.

This process continues for the defined number of epochs, gradually improving the generator's ability to generate realistic images that can fool the discriminator.

**OUTPUT:**

****

****

**Explanation of Output:**

Each line corresponds to a particular epoch during training, with the following information:

* Epoch: The epoch number, indicating which iteration of training is currently being executed.
* D Loss: The loss value of the discriminator for the current epoch.
* Acc.: The accuracy of the discriminator for the current epoch, represented as a percentage.
* G Loss: The loss value of the generator for the current epoch.