Ex No: 9 BUILD GENERATIVE ADVERSARIAL NEURAL NETWORK

Aim:

To build a generative adversarial neural network using Keras/TensorFlow.

Procedure:

- 1. Download and load the dataset.
- 2. Perform analysis and preprocessing of the dataset.
- 3. Build a simple neural network model using Keras/TensorFlow.
- 4. Compile and fit the model.
- 5. Perform prediction with the test dataset.
- 6. Calculate performance metrics.

```
Program:
import tensorflow as tf

from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
def build_generator(noise_dim):
    model = tf.keras.Sequential()

# Dense layer to project the noise into a larger dimension
    model.add(layers.Dense(128, activation='relu', input_dim=noise_dim))

# Add more dense layers
    model.add(layers.Dense(256, activation='relu'))
    model.add(layers.Dense(512, activation='relu'))

# Final layer to output the data (usually using 'tanh' for image generation)
```

model.add(layers.Dense(28 * 28, activation='tanh'))

```
model.add(layers.Reshape((28, 28))) # Shape output as 28x28 for images like MNIST
  return model
def build discriminator():
  model = tf.keras.Sequential()
  # Flatten the input image
  model.add(layers.Flatten(input_shape=(28, 28)))
  # Add dense layers to classify real/fake
  model.add(layers.Dense(512, activation='relu'))
  model.add(layers.Dense(256, activation='relu'))
  # Final layer to output a single probability (real or fake)
  model.add(layers.Dense(1, activation='sigmoid'))
  return model
def build_gan(generator, discriminator):
  model = tf.keras.Sequential()
  model.add(generator)
  model.add(discriminator)
  return model
# Compile the discriminator
discriminator = build_discriminator()
discriminator.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Build the generator
generator = build_generator(noise_dim=100)
```

```
# Compile the GAN (discriminator is untrainable when training the generator)
discriminator.trainable = False
gan = build gan(generator, discriminator)
gan.compile(loss='binary_crossentropy', optimizer='adam')
def train_gan(generator, discriminator, gan, epochs, batch_size, noise_dim):
  (X_train, _), _ = tf.keras.datasets.mnist.load_data() # Use MNIST as example
  X_{train} = X_{train} / 127.5 - 1.0 \# Normalize images to [-1, 1]
  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, noise_dim))
     fake_images = generator.predict(noise)
     # Train the discriminator (real = 1, fake = 0)
     d_loss_real = discriminator.train_on_batch(real_images, np.ones((batch_size, 1)))
     d_loss_fake = discriminator.train_on_batch(fake_images, np.zeros((batch_size, 1)))
     # Train the generator (wants discriminator to predict all as real)
     noise = np.random.normal(0, 1, (batch_size, noise_dim))
     g_loss = gan.train_on_batch(noise, np.ones((batch_size, 1)))
     # Print progress
     if epoch \% 100 == 0:
       print(f"{epoch} [D loss: {0.5 * np.add(d_loss_real, d_loss_fake)}] [G loss: {g_loss}]")
```

Optionally save generated samples to visualize progress

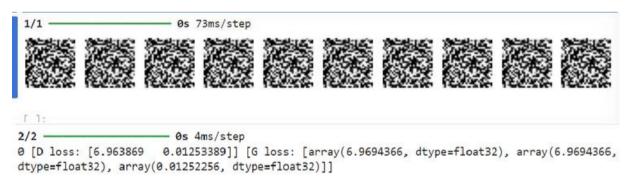
```
train_gan(generator, discriminator, gan, epochs=1000, batch_size=64, noise_dim=100)

def generate_images(generator, noise_dim, examples=10):
    noise = np.random.normal(0, 1, (examples, noise_dim))
    gen_images = generator.predict(noise)

plt.figure(figsize=(10, 10))
    for i in range(examples):
        plt.subplot(1, 10, i+1)
        plt.imshow(gen_images[i], cmap='gray')
        plt.axis('off')

plt.show()
```

Output:



Result:

Generative Adversial Neural network has been successfully built.

Call this function after training to visualize generated images

generate_images(generator, noise_dim=100)