#### EXPERIMENT – 3.4

**Mapped Course Outcome**

CO1: Identify and describe soft computing techniques and their roles in building intelligent machines.

**AIM:**  
Write a program to implement Generative Adversarial Networks (GANs) using genetic algorithms.

**Theory**  
Generative Adversarial Networks (GANs) are a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014. GANs consist of two neural networks: a generator and a discriminator. The generator creates fake data, while the discriminator evaluates the authenticity of the data. Genetic algorithms (GAs) are optimization techniques inspired by the process of natural selection, which can be used to optimize the GANs' parameters by evolving a population of candidate solutions.

**Procedure:**

**Step 1: Setup and Installation**

1. **Install Anaconda:**
   * Follow the same installation steps as provided in EXPERIMENT – 1.1.
2. **Install Required Libraries:**
   * Open Anaconda Navigator.
   * Ensure that tensorflow, keras, numpy, matplotlib, and deap (Distributed Evolutionary Algorithms in Python) are installed.

sh

Copy code

conda install tensorflow keras numpy matplotlib

pip install deap

**Step 2: Implementing GANs using Genetic Algorithms**

1. **Import Necessary Libraries:**

python

Copy code

import numpy as np

import tensorflow as tf

from tensorflow.keras.layers import Dense, Reshape, Flatten, Dropout, BatchNormalization, LeakyReLU

from tensorflow.keras.models import Sequential

from deap import base, creator, tools, algorithms

import random

import matplotlib.pyplot as plt

1. **Define GAN Components:**

python

Copy code

def build\_generator(latent\_dim):

model = Sequential([

Dense(256, input\_dim=latent\_dim),

LeakyReLU(alpha=0.2),

BatchNormalization(momentum=0.8),

Dense(512),

LeakyReLU(alpha=0.2),

BatchNormalization(momentum=0.8),

Dense(1024),

LeakyReLU(alpha=0.2),

BatchNormalization(momentum=0.8),

Dense(28 \* 28 \* 1, activation='tanh'),

Reshape((28, 28, 1))

])

return model

def build\_discriminator(img\_shape):

model = Sequential([

Flatten(input\_shape=img\_shape),

Dense(512),

LeakyReLU(alpha=0.2),

Dropout(0.4),

Dense(256),

LeakyReLU(alpha=0.2),

Dropout(0.4),

Dense(1, activation='sigmoid')

])

return model

1. **Load and Preprocess the Dataset:**

python

Copy code

(x\_train, \_), (\_, \_) = tf.keras.datasets.mnist.load\_data()

x\_train = (x\_train.astype(np.float32) - 127.5) / 127.5 # Normalize to [-1, 1]

x\_train = np.expand\_dims(x\_train, axis=3)

1. **Build the GAN Model:**

python

Copy code

latent\_dim = 100

img\_shape = (28, 28, 1)

generator = build\_generator(latent\_dim)

discriminator = build\_discriminator(img\_shape)

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

z = tf.keras.Input(shape=(latent\_dim,))

img = generator(z)

discriminator.trainable = False

validity = discriminator(img)

combined = tf.keras.Model(z, validity)

combined.compile(loss='binary\_crossentropy', optimizer='adam')

1. **Define Genetic Algorithm Functions:**

python

Copy code

def evolve\_population(population, toolbox, ngen=10, cxpb=0.5, mutpb=0.2):

for gen in range(ngen):

offspring = algorithms.varAnd(population, toolbox, cxpb=cxpb, mutpb=mutpb)

fits = toolbox.map(toolbox.evaluate, offspring)

for fit, ind in zip(fits, offspring):

ind.fitness.values = fit

population = toolbox.select(offspring, k=len(population))

return population

def train\_gan(gan, generator, discriminator, data, epochs, batch\_size, population\_size=10, generations=10):

latent\_dim = generator.input\_shape[1]

img\_shape = discriminator.input\_shape[1:]

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))

creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

toolbox.register("attr\_float", random.uniform, -1.0, 1.0)

toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr\_float, latent\_dim)

toolbox.register("population", tools.initRepeat, list, toolbox.individual)

def evaluate(individual):

noise = np.array(individual).reshape(1, latent\_dim)

generated\_image = generator.predict(noise)

return discriminator.predict(generated\_image),

toolbox.register("evaluate", evaluate)

toolbox.register("mate", tools.cxBlend, alpha=0.5)

toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.2, indpb=0.2)

toolbox.register("select", tools.selTournament, tournsize=3)

population = toolbox.population(n=population\_size)

for epoch in range(epochs):

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

real\_images = data[idx]

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

fake\_images = generator.predict(noise)

real\_labels = np.ones((batch\_size, 1))

fake\_labels = np.zeros((batch\_size, 1))

d\_loss\_real = discriminator.train\_on\_batch(real\_images, real\_labels)

d\_loss\_fake = discriminator.train\_on\_batch(fake\_images, fake\_labels)

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

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

valid\_y = np.ones((batch\_size, 1))

g\_loss = combined.train\_on\_batch(noise, valid\_y)

if epoch % 100 == 0:

print(f"Epoch {epoch} [D loss: {d\_loss[0]} | D accuracy: {100\*d\_loss[1]}] [G loss: {g\_loss}]")

population = evolve\_population(population, toolbox, ngen=generations)

return population

1. **Train the GAN:**

python

Copy code

epochs = 10000

batch\_size = 64

population\_size = 20

generations = 5

trained\_population = train\_gan(combined, generator, discriminator, x\_train, epochs, batch\_size, population\_size, generations)

1. **Visualize the Results:**

python

Copy code

def plot\_generated\_images(generator, examples=10, dim=(1, 10), figsize=(10, 1)):

noise = np.random.normal(0, 1, (examples, generator.input\_shape[1]))

generated\_images = generator.predict(noise)

plt.figure(figsize=figsize)

for i in range(examples):

plt.subplot(dim[0], dim[1], i + 1)

plt.imshow(generated\_images[i].reshape(28, 28), interpolation='nearest', cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()

plot\_generated\_images(generator)

**Step 3: Running the Program**

1. Open Jupyter Notebook from Anaconda Navigator.
2. Create a new Python 3 notebook.
3. Copy and paste the above code sections into the notebook cells.
4. Execute each cell sequentially to build, train, and visualize the GAN using genetic algorithms.

**Video Tutorial**

[GANs with Genetic Algorithms](https://www.youtube.com/watch?v=1V_9G0w78Ag)

**Further Reading**

Rolon-Mérette, D., Ross, M., Rolon-Mérette, T., & Church, K. (2016). Introduction to Anaconda and Python: Installation and setup. Python for research in psychology, 16(5), S5-S11.

**Prospective Viva Questions**

1. Explain Generative Adversarial Networks and their primary use.
2. Describe the roles of the generator and discriminator in GANs.
3. Discuss how genetic algorithms can be used to optimize GANs.
4. Explain the concept of the fitness function in the context of genetic algorithms.
5. Provide examples of real-world applications where GANs can be effectively used.