Machine Learning Applications

Particle Swarm Optimization

Code:-

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# Define the Rastrigin function

def rastrigin(x):

n = len(x)

return 10\*n + sum([xi\*\*2 - 10\*np.cos(2\*np.pi\*xi) for xi in x])

# Define the PSO algorithm

def pso(cost\_func, dim=2, num\_particles=30, max\_iter=100, w=0.5, c1=1, c2=2):

# Initialize particles and velocities

particles = np.random.uniform(-5.12, 5.12, (num\_particles, dim))

velocities = np.zeros((num\_particles, dim))

# Initialize the best positions and fitness values

best\_positions = np.copy(particles)

best\_fitness = np.array([cost\_func(p) for p in particles])

swarm\_best\_position = best\_positions[np.argmin(best\_fitness)]

swarm\_best\_fitness = np.min(best\_fitness)

# Iterate through the specified number of iterations, updating the velocity and position of each particle at each iteration

for i in range(max\_iter):

# Update velocities

r1 = np.random.uniform(0, 1, (num\_particles, dim))

r2 = np.random.uniform(0, 1, (num\_particles, dim))

velocities = w \* velocities + c1 \* r1 \* (best\_positions - particles) + c2 \* r2 \* (swarm\_best\_position - particles)

# Update positions

particles += velocities

# Evaluate fitness of each particle

fitness\_values = np.array([cost\_func(p) for p in particles])

# Update best positions and fitness values

improved\_indices = np.where(fitness\_values < best\_fitness)

best\_positions[improved\_indices] = particles[improved\_indices]

best\_fitness[improved\_indices] = fitness\_values[improved\_indices]

if np.min(fitness\_values) < swarm\_best\_fitness:

swarm\_best\_position = particles[np.argmin(fitness\_values)]

swarm\_best\_fitness = np.min(fitness\_values)

# Return the best solution found by the PSO algorithm

return swarm\_best\_position, swarm\_best\_fitness

# Define the dimensions of the problem

dim = 2

# Run the PSO algorithm on the Rastrigin function

solution, fitness = pso(rastrigin, dim=dim)

# Print the solution and fitness value

print('Solution:', solution)

print('Fitness:', fitness)

# Create a meshgrid for visualization

x = np.linspace(-5.12, 5.12, 100)

y = np.linspace(-5.12, 5.12, 100)

X, Y = np.meshgrid(x, y)

Z = rastrigin([X, Y])

# Create a 3D plot of the Rastrigin function

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.plot\_surface(X, Y, Z, cmap='viridis')

ax.set\_xlabel('x')

ax.set\_ylabel('y')

ax.set\_zlabel('z')

# Plot the solution found by the PSO algorithm

ax.scatter(solution[0], solution[1], fitness, color='red')

plt.show()

Genetic Algorithm

Code:-

import random

# Define the fitness function

def fitness\_function(chromosome):

return sum(chromosome)

# Define the genetic algorithm

def genetic\_algorithm(population\_size, chromosome\_length, fitness\_function, mutation\_probability=0.1):

# Initialize the population with random chromosomes

population = [[random.randint(0, 1) for j in range(chromosome\_length)] for i in range(population\_size)]

# Loop until a satisfactory solution is found

while True:

# Evaluate the fitness of each chromosome in the population

fitness\_values = [fitness\_function(chromosome) for chromosome in population]

# Select the fittest chromosomes to be the parents of the next generation

parents = [population[i] for i in range(population\_size) if fitness\_values[i] == max(fitness\_values)]

# Generate the next generation by applying crossover and mutation to the parents

next\_generation = []

while len(next\_generation) < population\_size:

parent1 = random.choice(parents)

parent2 = random.choice(parents)

crossover\_point = random.randint(1, chromosome\_length - 1)

child = parent1[:crossover\_point] + parent2[crossover\_point:]

for i in range(chromosome\_length):

if random.random() < mutation\_probability:

child[i] = 1 - child[i]

next\_generation.append(child)

# Update the population with the next generation

population = next\_generation

# Check if a satisfactory solution has been found

if max(fitness\_values) == chromosome\_length:

return parents[0]

# Example usage

chromosome\_length = 10

population\_size = 50

solution = genetic\_algorithm(population\_size, chromosome\_length, fitness\_function)

print("Solution found:", solution)