Report for Home Assignment: Particle Swarm Optimization Simulation

Group members:

1. Efrem Fitwi (320352)
2. Dahlak Kiros (320353)
3. Lidia Ghebreamlak (316689)
4. Mikal Kidane (308840)

Particle Swarm Optimization Simulation

The home assignment topic that we decided to do as a home assignment was the particle swarm optimization simulation. It has two parts, the first one was simulation of the Banana function and the Shwefel function using PSO and exporting the simulations in the form of a video. And the second part was implementation of the PSO algorithm without using any PSO libraries. We decided to write one code that can accomplish both this tasks. We implemented the PSO algorithm from scratch without using any libraries and then we incorporated the two above functions as fitness functions in the PSO algorithm implementation code we wrote. In the pso_main.py file we can see that we used a class called Particle to define the main attributes of the particles in the swarm and then we created the swarm as an array of these 50 particles. Because the upper bounds and lower bounds of the banana function and the shwefel function are different, we define the bounds for the functions separately. The basic goal of the PSO program here is to find the global minima of both the functions so we define the global best fitness value and the personal best fitness value of the particles to a high value so that these values get updated and minimized to find the optimal minimum value of the functions. To do this we have initialized the best fitness values (personal and global) to infinity. The details we used for both the functions are given in the table below.

	BANANA function	SHWEFEL Function
Target value (global minima)	0	0
Target position	[1,1]	[420.9687,420.9687]
Initial global best fitness value	Positive infinity	Positive infinity
Initial personal best fitness value	Positive infinity	Positive infinity
Initial position of particles	Random positions	Random positions
Minimum error criteria	0	0.00001
Number of particles	50	50
Inertia weight W	0.8	0.8
Cognitive coefficient c1	0.5	0.5
social coefficient c2	0.3	0.3
Function formula	$f(x,y) = (1-x)^2 + 100(y-x^2)^2$	$f(x) = 418.9829\mathrm{d} - \sum_{i=1}^{d}\left(x_i \sin(\sqrt{\lvert x_i \rvert})\right)$

In order to make the simulation videos of the output we used the animation class from the matplotlib library. We stored the position of every particle in the swarm in every iteration as history so the simulation can be made from that. To facilitate this we have written a separate plotter.py program and imported it into the pso_main.py program. In the videos we can see that the particles converge in the global minima of the functions marked by red x in the graph. We have run into some challenges like the convergence of the points in the wrong position at the end of the iterations in some occassional runs. For example, the common wrong positions the swarm seems to converge in is [-302.47495416, 421.03820265]. This seems to happen in 1 out of 3 or 4 runs. We still have not come up with the exact reason this is happening, but as we were instructed by the instructor this is a common issue in Machine Learning. But after closely studying the code we wrote we figured out there was something we looked over and did not notice, a small coding mistake on our part. It turned out that when we were updating the velocity using the update_velocity function, we did not compute the velocity for the dimensions of the particle position separately, we calculated it generaly and that affected the change in the position of the particles causing the same change in position in all the dimensions of the particle. We finally fixed this and the program seems to be working fine now.

WRONG CODE:

```
def update_velocity(particle):
    W=0.8
    c1=0.5
    c2=0.3
    r1=random.random()
    r2=random.random()
    cog_vel = c1*r1*(particle.personal_bestpos-particle.pos)
    social_vel = c2*r2*(global_bestpos-particle.pos)
    return ((W*particle.velocity) + cog_vel + social_vel)
```

FIXED CODE:

```
def update_velocity(particle):
    W=0.8
    c1=0.9
    c2=0.3
    v =np.array([0.,0.])
    for i in range(2):
        r1=random.random()
        r2=random.random()
        cog_vel = c1*r1*(particle.personal_bestpos[i]-particle.pos[i])
        social_vel = c2*r2*(global_bestpos[i]-particle.pos[i])
        v[i] = float(((W*particle.velocity[i]) + cog_vel + social_vel))
    return v
```