Homework 6 - Kyla Yujiri (key243)

NEAT with Ms. Pacman and Space Invaders

For this homework, I wanted to explore the NEAT algorithm. I had no idea where to start with my own algorithm and the NEAT algorithm seemed a little easier to understand than SB-3 given the time limits at the end of the semester.

I had problems initially because I did not understand that I needed to download my own roms to be able to use these games. In the lab, the MountainCar environment was included with gym so I thought that was the same for Ms. Pacman and Space Invaders. However, it was not. I found roms for both games at atarimania.com. I used the original Atari 2600 versions for both of them (Space Invaders, Ms. Pacman). I originally wanted to do Breakout, but for some reason, the rom from the website did not work with gym.

```
kylayujiri@Kylas-MBP hw6 % ale-import-roms roms/
[SUPPORTED]                ms_pacman             roms/mspacman.bin


[NOT SUPPORTED]                                  roms/breakout.bin

Imported 1 / 2 ROMs
```

I based my files on the example code provided by neat-python. Specifically, I used their single pole balancing files: evolve-feedforward.py, config-feedforward, and test-feedforward.py.

Ms. Pacman

From the gym documentation's specs on Ms. Pacman, I learned that the action space is a single integer [0,18) and the observation space is (210, 160, 3). I used cv2 to reduce the observation space to 42 x 32 and made it grayscale. This made the input size for the feedforward network 1,344.

I originally had `render='human'` instead of `render='rgb_array'` so I could see all the learning happening visually. It was cool to see, but very slow so I changed it to the latter so that I could get results faster. Here is an image of the renderings:

For the config file, I left most of the values alone except for `fitness_threshold`, `pop_size`, `num_inputs`, `num_hidden`, `num_outputs`, `activation_default`, and `max_stagnation`.

I originally had `fitness_threshold` set to 200, but the algorithm finished very fast. I realized that 200 was not a very high score for Ms. Pacman.

```
kylayujiri@Kylas-MBP mspacman % python3 mspacman_neat.py

 ****** Running generation 0 ******

A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
Population's average fitness: 78.60000 stdev: 41.37681
Best fitness: 240.00000 - size: (3, 2017) - species 18 - id 18

Best individual in generation 0 meets fitness threshold - complexity: (3, 2017)
Key: 18
Fitness: 240.0
Nodes:
        0 DefaultNodeGene(key=0, bias=-1.202876822380358, response=1.0, activation=sigmoid, aggregation=sum)
        35 DefaultNodeGene(key=35, bias=1.6094863512960236, response=1.0, activation=sigmoid, aggregation=sum)
        36 DefaultNodeGene(key=36, bias=-2.2473260220552422, response=1.0, activation=sigmoid, aggregation=sum)
Connections:
        DefaultConnectionGene(key=(-1344, 35), weight=0.18039620342535387, enabled=True)
        DefaultConnectionGene(key=(-1343, 0), weight=1.3015636215850566, enabled=True)
        DefaultConnectionGene(key=(-1343, 35), weight=-1.2548109642195533, enabled=True)
        DefaultConnectionGene(key=(-1342, 0), weight=2.18533180313882, enabled=True)
        DefaultConnectionGene(key=(-1342, 36), weight=0.574808277162482, enabled=True)
```

I played Ms. Pacman myself and found out that 2700 is the score when I eat all the dots. I set the `fitness_threshold` to 2700, but I realized this was a mistake when the fitness was nowhere near 2700 after a good while. At this point, the `pop_size` was 50 and the `max_stagnation` was 30 so each generation took a long time and less effective species did not die off quickly.

```
kylayujiri@Kylas-MBP mspacman-recurrent % python3 mspacman_neat.py

 ****** Running generation 0 ******

A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]

A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
A.L.E: Arcade Learning Environment (version 0.7.5+db37282)
[Powered by Stella]
Population's average fitness: 94.40000 stdev: 77.64432
Best fitness: 450.00000 - size: (3, 1346) - species 26 - id 26
Average adjusted fitness: 0.088
Mean genetic distance 3.372, standard deviation 0.527
Population of 101 members in 50 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    0     2     70.0    0.026     0
     2    0     2     60.0    0.000     0
     3    0     2     70.0    0.026     0
     4    0     2     60.0    0.000     0
     5    0     2     70.0    0.026     0
     6    0     2     70.0    0.026     0
     7    0     2     60.0    0.000     0
     8    0     2     60.0    0.000     0
     9    0     2    350.0    0.744     0
    10    0     2     60.0    0.000     0
    11    0     2     70.0    0.026     0
    12    0     2    210.0    0.385     0
    13    0     2     60.0    0.000     0
    14    0     2     60.0    0.000     0
    15    0     2     60.0    0.000     0
    16    0     2     60.0    0.000     0
    17    0     2    210.0    0.385     0
    18    0     2     60.0    0.000     0
    19    0     2     60.0    0.000     0
    20    0     2     60.0    0.000     0
    21    0     2     70.0    0.026     0
    22    0     2    210.0    0.385     0
    23    0     2     60.0    0.000     0
    24    0     2     90.0    0.077     0
    25    0     2     70.0    0.026     0
    26    0     3    450.0    1.000     0
    27    0     2     60.0    0.000     0
    28    0     2     60.0    0.000     0
    29    0     2     60.0    0.000     0
    30    0     2     60.0    0.000     0
    31    0     2     70.0    0.026     0
    32    0     2     60.0    0.000     0
    33    0     2     60.0    0.000     0
```

The algorithm seemed to be reaching fairly high levels of fitness, but it would just not end. I did not know how to stop the program, but still save the winner.

```
 ****** Running generation 137 ******

Population's average fitness: 181.80000 stdev: 258.64795
Best fitness: 1640.00000 - size: (6, 1200) - species 52 - id 4361
Average adjusted fitness: 0.078
Mean genetic distance 2.212, standard deviation 0.966
Population of 50 members in 2 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
    52   34    27   1640.0    0.073     2
    53   33    23    880.0    0.083    18
Total extinctions: 0
Generation time: 13.884 sec (13.368 average)
```

Because I didn't know how to save it, I just terminated the program. I changed the `fitness_threshold` to a more reasonable number (1000). I also changed the `pop_size` to 30 and the `max_stagnation` to the documentation's recommended default of 15. The algorithm was able to reach the `fitness_threshold` in 20 generations.

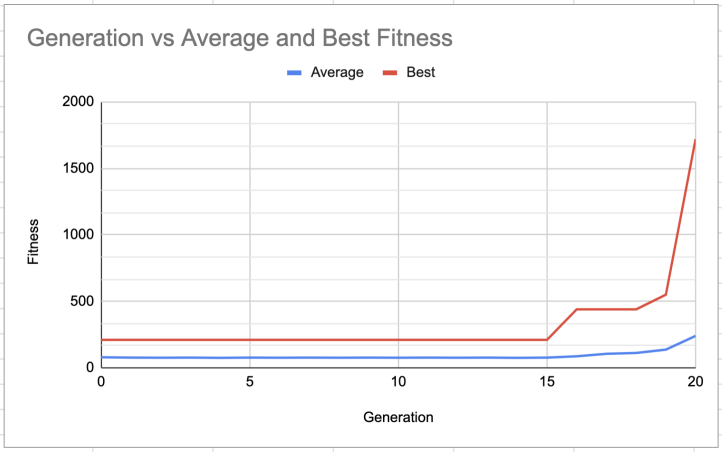
```
 ****** Running generation 20 ******

Population's average fitness: 240.50000 stdev: 377.53775
Best fitness: 1720.00000 - size: (3, 1319) - species 21 - id 130

Best individual in generation 20 meets fitness threshold - complexity: (3, 1319)
Key: 130
Fitness: 1720.0
Nodes:
        0 DefaultNodeGene(key=0, bias=0.08288632779720026, response=1.0, activation=sigmoid, aggregation=sum)
        13 DefaultNodeGene(key=13, bias=1.8593847900628724, response=1.0, activation=sigmoid, aggregation=sum)
        14 DefaultNodeGene(key=14, bias=-0.9204526506933759, response=1.0, activation=sigmoid, aggregation=sum)
Connections:
        DefaultConnectionGene(key=(-1344, 14), weight=-0.373289320316816, enabled=True)
        DefaultConnectionGene(key=(-1343, 13), weight=1.441349100246275, enabled=True)
        DefaultConnectionGene(key=(-1343, 14), weight=2.843659270249499, enabled=True)
        DefaultConnectionGene(key=(-1342, 13), weight=-1.5820797378187264, enabled=True)
        DefaultConnectionGene(key=(-1342, 14), weight=0.31503778816268696, enabled=True)
        DefaultConnectionGene(key=(-1341, 14), weight=0.6469997402139731, enabled=True)
        DefaultConnectionGene(key=(-1340, 13), weight=-1.605250164417146, enabled=True)
        DefaultConnectionGene(key=(-1338, 13), weight=0.6136163311742577, enabled=True)
        DefaultConnectionGene(key=(-1338, 14), weight=1.2421699957656658, enabled=True)
        DefaultConnectionGene(key=(-1337, 14), weight=-0.0548809665585035, enabled=True)
        DefaultConnectionGene(key=(-1336, 14), weight=1.444733369747615, enabled=True)
        DefaultConnectionGene(key=(-1335, 13), weight=2.0149263204634904, enabled=True)
```

When I ran the winning genome with `testneat.py`, though, I was a little disappointed in what I saw. I'm not sure if my expectation was unrealistic, but Ms. Pacman did not move in an efficient way. She moved a little, collected some dots, and then stopped until a ghost killed her. This happened for all 3 lives, but the dots collected resulted in the score that met my threshold.

After this, I looked back at the chart from the lecture of different Atari games and their performance versus humans. I noticed that Ms. Pacman was near the bottom which could explain this behavior.

To plot the performance of the agent, I had to manually take the fitness from the terminal and graph each one for each generation. NEAT provided numbers for the populations average fitness for that generation as well as the best fitness for that generation, so I plotted both.

Generation	Average	Best
0	79	210
1	76.25	210
2	75.85366	210
3	76.25	210
4	75.09756	210
5	76.25	210
6	75.85366	210
7	76.25	210
8	76.09756	210
9	76.25	210
10	76.09756	210
11	76.25	210
12	76.09756	210
13	76.25	210
14	75.09756	210
15	76.25	210
16	86.5	440
17	105	440
18	111.5	440
19	136.5	550
20	240	1720



Space Invaders

I wanted to see if the mediocre outcome from Ms. Pacman was because of the limitations of my implementation of the NEAT algorithm or because the game is Ms. Pacman. I chose Space Invaders because it is higher up in the chart from the lecture than Ms. Pacman.

I originally ran it with the same config settings as Ms. Pacman, except I changed the `fitness_threshold`. I started with the `fitness_threshold` as 100 and 200, but the learning was very quick. I then changed it to 700.

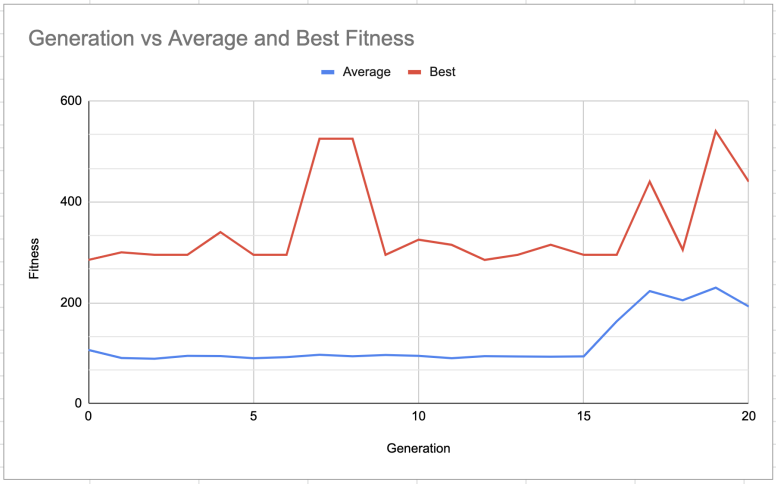
After I changed it to 700, the algorithm took 75 generations to reach that threshold.

```
 ****** Running generation 75 ******

Population's average fitness: 283.33333 stdev: 196.77539
Best fitness: 750.00000 - size: (5, 11) - species 32 - id 1420

Best individual in generation 75 meets fitness threshold - complexity: (5, 11)
Key: 1420
Fitness: 750.0
Nodes:
        0 DefaultNodeGene(key=0, bias=0.6033729636159819, response=1.0, activation=sigmoid, aggregation=sum)
        255 DefaultNodeGene(key=255, bias=1.5276150716671242, response=1.0, activation=sigmoid, aggregation=sum)
        266 DefaultNodeGene(key=266, bias=0.260080263293228, response=1.0, activation=sigmoid, aggregation=sum)
        320 DefaultNodeGene(key=320, bias=0.18670588445247138, response=1.0, activation=sigmoid, aggregation=sum)
        335 DefaultNodeGene(key=335, bias=-1.0522400584471436, response=1.0, activation=sigmoid, aggregation=sum)
Connections:
        DefaultConnectionGene(key=(-999, 0), weight=0.4542108187952849, enabled=True)
        DefaultConnectionGene(key=(-870, 0), weight=-0.563397839211028, enabled=False)
        DefaultConnectionGene(key=(-870, 266), weight=1.1662780321089128, enabled=False)
        DefaultConnectionGene(key=(-870, 335), weight=1.0, enabled=True)
```

I didn't want to manually graph all 75 generations, so I made two graphs: one with the first 21 generations and one with the last 21 generations.

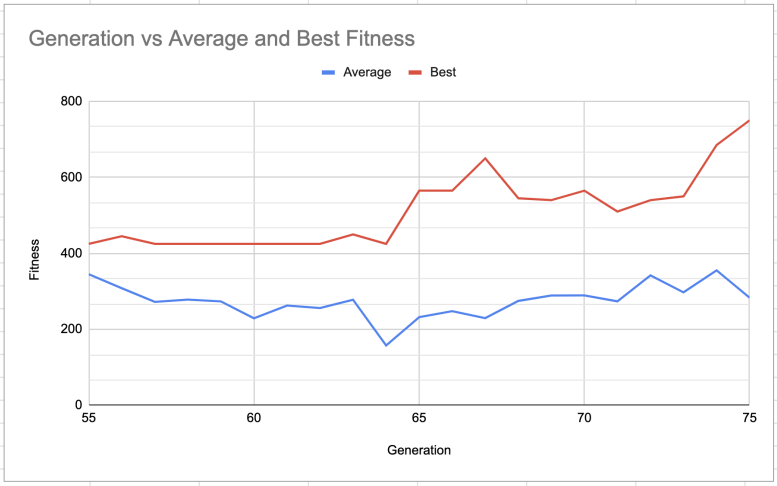
First 21 Generations vs Average and Best Fitness:

Generation	Average	Best
0	106.66667	285
1	90.66667	300
2	89.16667	295
3	95	295
4	94.58333	340
5	90.16667	295
6	92.33333	295
7	97	525
8	94	525
9	96.66667	295
10	94.91667	325
11	90.33333	315
12	94.41667	285
13	93.75	295
14	93.33333	315
15	93.91667	295
16	163.4375	295
17	223.27586	440
18	205	305
19	230	540
20	192.66667	440



Last 21 Generations vs Average and Best Fitness:

Generation	Average	Best
55	345.16667	425
56	308.33333	445
57	272.5	425
58	278.5	425
59	273.70968	425
60	229.16667	425
61	262.66667	425
62	256.33333	425
63	278	450
64	157.66667	425
65	232.33333	565
66	248	565
67	229.66667	650
68	275	545
69	289.16667	540
70	289.33333	565
71	273.83333	510
72	342	540
73	297.5	550
74	355.16667	685
75	283.33333	750



Looking at the rendering of the winner, it seemed like the agent preferred staying in one spot while moving and it did not try to avoid any of the bullets from the aliens.

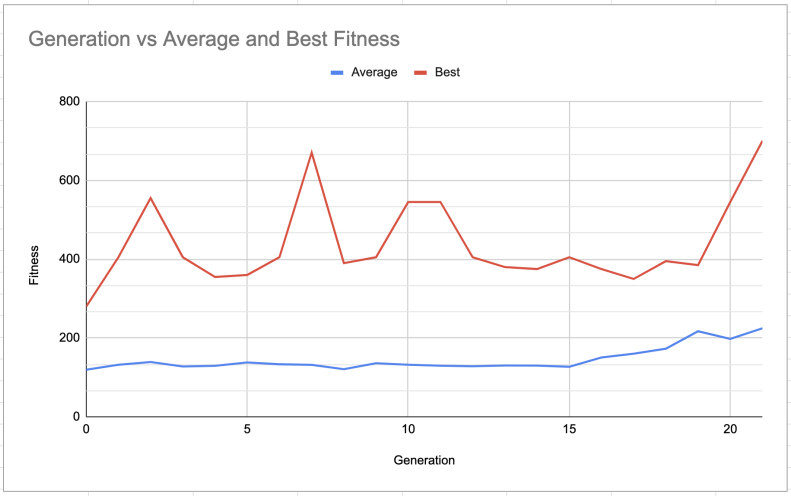
I wanted to see if changing the number of hidden layers would do anything so I kept all settings the same except for changing the number of hidden layers from 2 to 6.

This time, the algorithm only took 21 generations to get 750 points (fitness) which was a significant improvement over the previous 75 generations.

```
 ****** Running generation 21 ******

Population's average fitness: 225.00000 stdev: 132.28757
Best fitness: 700.00000 - size: (7, 3307) - species 10 - id 106

Best individual in generation 21 meets fitness threshold - complexity: (7, 3307)
Key: 106
Fitness: 700.0
Nodes:
        0 DefaultNodeGene(key=0, bias=0.6631951879237283, response=1.0, activation=sigmoid, aggregation=sum)
        55 DefaultNodeGene(key=55, bias=0.7552848159490l7, response=1.0, activation=sigmoid, aggregation=sum)
        56 DefaultNodeGene(key=56, bias=-0.4701829497721034, response=1.0, activation=sigmoid, aggregation=sum)
        57 DefaultNodeGene(key=57, bias=-1.107049796014014, response=1.0, activation=sigmoid, aggregation=sum)
        58 DefaultNodeGene(key=58, bias=0.37076505386045555, response=1.0, activation=sigmoid, aggregation=sum)
        60 DefaultNodeGene(key=60, bias=1.0062139446533034, response=1.0, activation=sigmoid, aggregation=sum)
        186 DefaultNodeGene(key=186, bias=0.33248702163577837, response=1.0, activation=sigmoid, aggregation=sum)
Connections:
        DefaultConnectionGene(key=(-1344, 57), weight=-0.09765356943633968, enabled=True)
        DefaultConnectionGene(key=(-1344, 58), weight=-0.6367993310828107, enabled=True)
        DefaultConnectionGene(key=(-1344, 60), weight=0.7908670316428189, enabled=True)
        DefaultConnectionGene(key=(-1343, 57), weight=1.0093389515442757, enabled=True)
        DefaultConnectionGene(key=(-1343, 60), weight=-0.19769664483807348, enabled=True)
```

Here is the graph of generation vs average and best fitness:

Generation	Average	Best
0	119.66667	280
1	132.5	405
2	139.16667	555
3	128.08333	405
4	129.75	355
5	138.25	360
6	133.6667	405
7	132.08333	670
8	121.16667	390
9	136.25	405
10	132.33333	545
11	130	545
12	128.66667	405
13	130.41667	380
14	130.16667	375
15	127.33333	405
16	150.92105	375
17	160.42857	350
18	173.06452	395
19	217.41935	385
20	197.90323	545
21	225	700



I also tried changing the network to a recurrent one from a feedforward one, but did not find significant changes in performance. This is not shown, but all I did to try it out was change `feed_forward` to `False` and replace `net = neat.nn.FeedForwardNetwork.create(genome, config)` with `net = neat.nn.RecurrentNetwork.create(genome, config)`