Assignment 1

Problem 1

In a 10x10 grid, assume the robot has 4 actions and 8 sensors, represented below:

Actions:

- Move Up, Down, Left, Right
- Assume that, the robot will move towards the closest corner in the grid

Sensors:

- Top, Bottom, Left, Right Proximity
- Top Left, Top Right, Bottom Left, Bottom Right Corner Proximity

Initial State:

- The four corners represented as follows:
  - Top Left Corner (0,0)
  - Top Right Corner (0,9)
  - Bottom Left Corner (9,0)
  - Bottom Right Corner (9,9)
- Initial position:
  - Assume that robot starts at (x,y)
  - Will direct the robot to move toward one of the four corners

Production Rule:

- **In the corner**
  - Condition: if x==(0,0) || x==(0,9) || x==(9,0) || x==(9,9)
  - Action: no further action needed
- **Move to top left corner**
  - Condition: if (x,y) closest to (0,0) is TRUE
  - Action: if x>0, x--; if y>0, y--
- **Move to top right corner**
  - Condition: if (x,y) closest to (0,9) is TRUE
  - Action: if x>0, x--; if y<9, y++
- **Move to bottom left corner**
  - Condition: if (x,y) closest to (9,0) is TRUE
  - Action: if x<9, x++; if y>0, y--
- **Move to bottom right corner**
  - Condition: if (x,y) closest to (9,9) is TRUE
  - Action: if x<9, x++; if y<9, y++

Problem 2

Assume that x are the inputs of Threshold Logic Unit (TLU), the weighted sum will be

From the weight vectors:

- x_1 and x_2 have the most positive influence
- x_3 and x_4 have negative effects
- $_5$ has a small positive contribution

Apply test cases based on and x_2:

- 1, output>1 regardless of
- 0, output>1 when x
- 1, output<1 when x
- 0, output will always <1

Hence, the boolean expression is:

Problem 3

In Folder gp, Python File: *genetic_programming.ipynb*

Fitness Function:

- The fitness function evaluates how well a perceptron matches the given training set. For a perceptron defined by a weight vector $(w_1, \dots, w_n)$ and threshold θ, the perceptron's output for an input $(x_1, \dots, x_n)$ is given by: y
- The fitness of a perceptron can be measured by the number of correct classifications it makes on the training set.
- Specifically, if a perceptron classifies a tuple $(x_1, \dots, x_n, l)$ correctly, it gains points. The fitness function can thus be the number (or percentage) of correctly classified examples.

Crossover Operator:

- The crossover operator generates offspring by combining parts of two parent perceptrons. A simple way to implement crossover is to randomly choose a crossover point in the weight vector and swap the weights between two parent perceptrons at that point.
- If two parent perceptrons have weights w^A and w^B, we can choose a random index i and generate offspring by swapping the weights: w
- For this, we may assume the crossover ratio is 90%

Copy Operator:

- The copy operator passes individuals from one generation to the next unchanged. It ensures that high-performing individuals are preserved across generations.
- For this, we may assume that copy ratio is 10%

Mutation Operator:

- The mutation operator introduces randomness into the population by randomly altering the weights of a perceptron.
- For a small probability, we introduce a random value Δw to each weight, eg:

Size of Initial Generation & Program Generation:

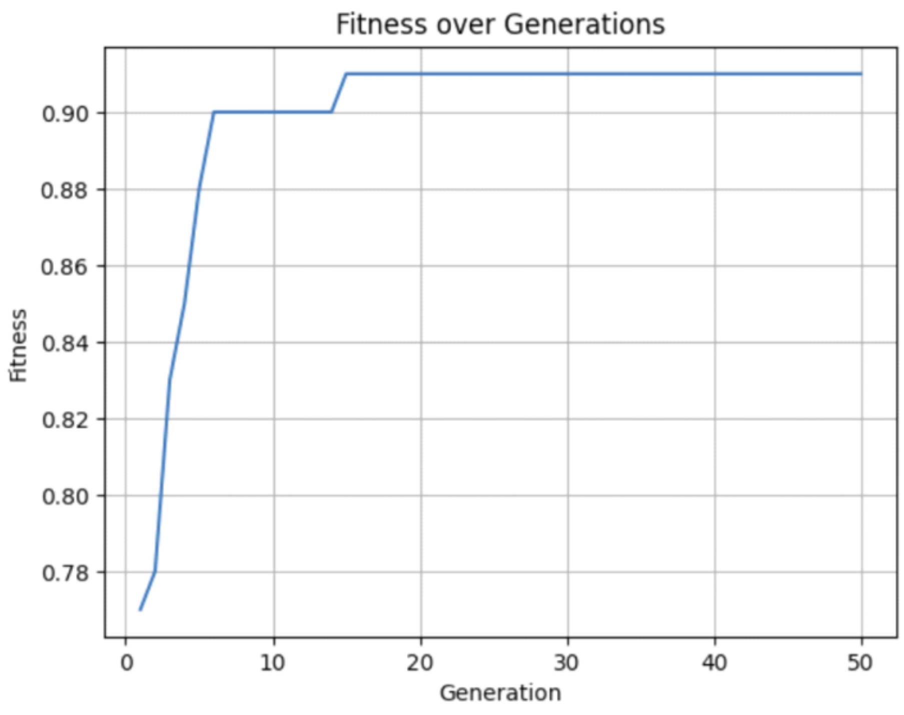
- The initial generation can be created by randomly generating the weight vectors w and threshold θ
- The size of the generation can vary depending on the problem, for this case we use 100 individuals
- The weight values and thresholds are usually drawn from a uniform distribution, for this case we use [-1,1]

Stopping Condition:

- The evolution can be stopped either after 50 generations, or 100% fitness thresold is reached

Output for the Provided Training Set:

```
Best Weights and Threshold:
Weights (w1 to w9): [0.3561407218106871, -0.8404465286543144, -1.224263787987804, 0.43940886
969204374, -0.4397450041682469, -1.0229340499818895, 0.763647488917591, -0.1582725016069777,
0.30796838620363864]
Threshold (θ): -0.796222948959038
```



Training Data (from 1-10 and 40-50):

```
Generation 1, Best Fitness: 0.6200
Generation 2, Best Fitness: 0.7400
Generation 3, Best Fitness: 0.8200
Generation 4, Best Fitness: 0.8300
Generation 5, Best Fitness: 0.8600
Generation 6, Best Fitness: 0.8800
Generation 7, Best Fitness: 0.9000
Generation 8, Best Fitness: 0.9000
Generation 9, Best Fitness: 0.9000
Generation 10, Best Fitness: 0.9000
Generation 40, Best Fitness: 0.9000
Generation 41, Best Fitness: 0.9000
Generation 42, Best Fitness: 0.9000
Generation 43, Best Fitness: 0.9000
Generation 44, Best Fitness: 0.9000
Generation 45, Best Fitness: 0.9000
Generation 46, Best Fitness: 0.9100
Generation 47, Best Fitness: 0.9100
Generation 48, Best Fitness: 0.9100
Generation 49, Best Fitness: 0.9100
Generation 50, Best Fitness: 0.9100
```

Problem 4

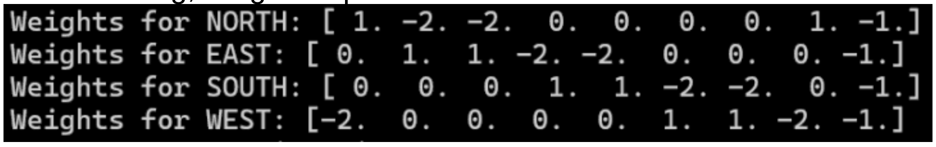
In folder pacman, files added/edited:

- TrainingData (csv File)
- ECTraining.py
- reactiveAgents.py (ECAgent Class)

The program runs on Python3, can be tested via the following way:

- open cmd, cd to pacman folder
- run: "python pacman.py --layout smallMap --pacman ECAgent"

After Training, weight outputs are:



```
Weights for NORTH: [ 1. -2. -2.  0.  0.  0.  0.  1. -1.]
Weights for EAST: [ 0.  1.  1. -2. -2.  0.  0.  0. -1.]
Weights for SOUTH: [ 0.  0.  0.  1.  1. -2. -2.  0. -1.]
Weights for WEST: [-2.  0.  0.  0.  0.  1.  1. -2. -1.]
```

Boolean Expression:
Given that training set is in the form of (s
For perceptron moving north, the boolean expression is

Problem 5

In folder pacman, files added/edited:

- reactiveAgents.py (SMAgent Class)

Note:

- Both Problem 4 and 5 stored in the same python file
- Both are implemented in reactiveAgents.py

The program runs on Python3, can be tested via the following way:

- open cmd, cd to pacman folder
- run: "python pacman.py --layout smallMap --pacman SMAgent"

Answer of sub-problem 2:
No, as it's non-linearly seperable