COMP5211 2020 Fall Semester Assignment #1
Suggested solution

Problem 1. (10%)

A simple solution is as follows:

$$\overline{s_2} \rightarrow North$$
$$\overline{s_8} \rightarrow West$$
$$1 \rightarrow Nil$$

Problem 2. (10%)

$$x_1x_2 + x_1\bar{x}_3\bar{x}_4 + x_2\left(\bar{x}_3 + \bar{x}_4\right)$$

or any equivalent boolean function.

Problem 3. (30%) Accept any reasonable solution. Here provide one of them.

1. What's your fitness function?
   For each instance in the training set, if the given program agrees with the label, then it gets 1 point, otherwise, it gets 0 point. The fitness value is then the sum. The maximum value is sample size in the training set, which is 100.

2. What's your crossover operator?
   Pick an $1 \leq i \leq n$, generate

   $$\left[w_1, \ldots, w_i, w'_{i+1}, \ldots, w'_n, w'_{n+1}\right]$$

   from

   $$\left[w_1, \ldots, w_{n+1}\right], \left[w'_1, \ldots, w'_{n+1}\right]$$

   where w_{n+1} and w'_{n+1} are the thresholds of the respective perceptrons.

3. What's your copy operator?
   Tournament selection.

4. What's your mutation operator, if you use any?
   A gene is randomly selected. The selected gene is assigned a random value.

5. What's the size of the initial generation, and how are programs generated?
   1000 (of course, the larger the better); the programs are generated with random values uniformly sampled from [-4,4]

6. When do you stop the evolution? Evolve it up to a fixed iteration, when it satisfies a condition on the fitness function, or a combination of the two?
   Stop when it reach MAX iterations **or** one of the program has the max fitness value.

7. What's the output of your system for the training set in the next page? This training set will be uploaded to canvas as a csv file.
   The training set was randomly generated from the following perceptron:

$$[-0.3, -1.9, -2.8, 1.4, 1.9, -1.3, 2.4, -0.7, 0.8, 0]$$

   Any other program that satisfies the training set is also acceptable.

Problem 4. (25%)

The learning result depends on your choices of initial weights, learning rate, and threshold. If we set initial weights to 0, set learning rate to 1, and add a new input that always have value 1 as threshold, the resulting weights for each direction are:

North: `[1,-2,-2,0,0,0,0,1,-1]`

East : `[0,1,1,-2,-2,0,0,0,-1]`

South: `[0,0,0,1,1,-2,-2,0,-1]`

West : `[-2,0,0,0,0,1,1,-2,-1]`

The corresponding boolean functions are:

$$(s_1 + s_8)\,\overline{s_2}\,\overline{s_3} \rightarrow north$$
$$(s_2 + s_3)\,\overline{s_4}\,\overline{s_5} \rightarrow east$$
$$(s_4 + s_5)\,\overline{s_6}\,\overline{s_7} \rightarrow south$$
$$(s_6 + s_7)\,\overline{s_1}\,\overline{s_8} \rightarrow west$$

A sample implementation of `ECAgent` is:

```
class ECAgent(Agent):

    def getAction(self, state):
        ''' Your code goes here! '''
        s = state.getPacmanSensor() + [1]
        w_north = [1, -2, -2, 0, 0, 0, 0, 1, -1]
        w_east = [0, 1, 1, -2, -2, 0, 0, 0, -1]
        w_south = [0, 0, 0, 1, 1, -2, -2, 0, -1]
        w_west = [-2, 0, 0, 0, 0, 1, 1, -2, -1]

        def dot(x, y):
            return sum(i * j for i, j in zip(x, y))
        if dot(s, w_north) >= 0:
            return Directions.NORTH
        if dot(s, w_east) >= 0:
            return Directions.EAST
        if dot(s, w_south) >= 0:
            return Directions.SOUTH
        if dot(s, w_west) >= 0:
            return Directions.WEST
        return Directions.NORTH
```

Problem 5. (25%)

1. Using the production system that we have learned in class, a sample implementation of SMAgent is:

```
class SMAgent(Agent):

    def registerInitialState(self,state):
        sense = state.getPacmanImpairedSensor()
        self.prevAction = Directions.STOP
        self.prevSense = sense

    def getAction(self, state):
        sense = state.getPacmanImpairedSensor()
        prevSense = self.prevSense
        prevAction = self.prevAction
        w = [1,sense[0],1,sense[1],1,sense[2],1,sense[3]]
        w[0] = 1 if prevSense[0] and Directions.EAST == prevAction else 0
        w[2] = 1 if prevSense[1] and Directions.SOUTH == prevAction else 0
        w[4] = 1 if prevSense[2] and Directions.WEST == prevAction else 0
        w[6] = 1 if prevSense[3] and Directions.NORTH == prevAction else 0

        if (w[1] and not w[3]):
            action = Directions.EAST
        elif (w[3] and not w[5]):
            action = Directions.SOUTH
        elif (w[5] and not w[7]):
            action = Directions.WEST
        elif (w[7] and not w[1]):
            action = Directions.NORTH
        elif w[0]:
            action = Directions.NORTH
        elif w[2]:
            action = Directions.EAST
        elif w[4]:
            action = Directions.SOUTH
        elif w[6]:
            action = Directions.WEST
        else:
            action = Directions.NORTH

        self.prevSense = sense
        self.prevAction = action
        return action
```

2. No... and yes?

No: We cannot implement the production system with perceptron of such feature vector. This is because the action functions is not linearly-separable with respect to such feature vector. For instance, the boolean function for the east action is

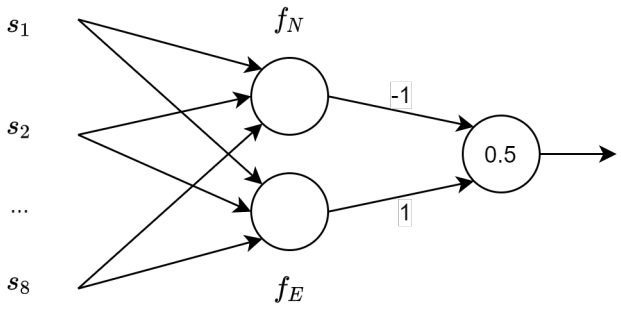
$$s_2\bar{s_4} + \overline{s_4\bar{s_6} + s_6\bar{s_8} + s_8\bar{s_2} + \tilde{s_2}E\tilde{s_4}S}$$

Setting $s_2 = s_4, s_6 = s_8, E = 0, \tilde{s_4} = S = 1$, then we have the boolean function equals to

$$\overline{s_2\bar{s_6} + s_6\bar{s_2}} = \overline{s_2 \oplus s_6} = s_2 \oplus \bar{s_6}$$

and we know that $a \oplus b$ (`a xor b`) is not linearly separable.

Yes: In problem 4, we have the priority of north, east, south, west when more than one action functions output 1. Such priority in fact encodes a multilayer perceptron implicitly. Denote f_N, f_E be the action functions trained for moving north and east respectively, then the rule for moving east is encoded in the following multilayer perceptron.



Note that the latter part of this multilayer perceptron represensts $f_E\overline{f_N}$.

In light of this, we can make up a priority of action functions to encode similar multilayer perceptrons for Problem 5. For example, we may train another 4 action functions for the 4 directions and infer a decision from these new functions if the original 4 functions output 0.