Q1:

Suppose the robot is always looking for the lower right corner, wherever its initial position is, firstly, it senses s4 cell. If s4 is free, move east, then senses new s4 cell. Repeat moving east, until its s4 is not free, then senses robot’s s6 cell. If s6 is free, move south, then senses the new s6 cell. Repeat moving south, until robot’s s6 is not free, now the robot is in the lower right corner.

Q2:

1. The inputs(x1, ..., x5) can be either 1 or 0, weight of x5 is 0.5, whether x5 is 1 or 0, it never makes difference to whether the threshold can be passed or not. So ignore x5, it need not show up in the boolean function.

2. Easily to find that at least one of x1 or x2 must be ‘on’.

2.1 If x1 is on, x2 is off, x3 and x4 must be both off, then the threshold can be passed. Even if x2 is on, it makes no difference to the result. So, x2 can be ignored in the current situation. The term is: wpsoffice.

2.2 If x1 is off, x2 is on, x3 and x4 can’t both be on, then the threshold can be passed. In current situation, x1 can be ignored because it doesn’t effect the result. The boolean term is: wpsoffice.

2.3 If x1 and x2 are both on, the threshold can be passed in any way. The boolean term is: wpsoffice.

3. The final boolean function which following the TLU implement is: wpsoffice.

Q3:

1. Suppose the length of collection of tuples of the form (x1, ..., xn, l) is L, we have M tuples (w1, ..., wn, theta), pop means tuple of (w1, ..., w), in every loop, the perceptron gives L outputs, fitness value is the number of the right output(equals to l). The fitness function returns an array which contains every fitness value.

def fitness\_function(pop):

fitness = []

for i in range(len(pop)):

fitness\_value = 0

for j in range(len(X\_train)):

x = X\_train[j]

d = D\_train[j]

output = 1 if (np.dot(x, pop[i]) >= theta[i]) else 0

if d == output:

fitness\_value += 1

fitness.append(fitness\_value)

return fitness

2. Suppose crossover rate is *pc*, swap a pop of a row with the last j bit of the pop of its next row:

def crossover(pc, pop):

crossover\_pop = pop.copy()

for i in range(len(pop)-1):

if (np.random.rand() < pc):

j = int(np.random.rand()\*(N))

crossover\_pop[i, j:] = pop[i+1, j:]

return crossover\_pop

3. Copy only the first M/2 lines data with a large fitness value:

def copy(fitness, pop):

arr = np.argsort(fitness)[::-1]

select\_pop = np.zeros((M/2, N))

for i in range(len(arr)/2):

select\_pop[i] = pop[arr[i]]

return select\_pop

4. Suppose mutation rate is pm, add a random number in [-1.0, 1.0] to the j-th bit of the pop:

def mutation(pm, pop):

mutation\_pop = pop.copy()

for i in range(len(mutation\_pop)):

if (np.random.rand() < pm):

j = int(np.random.rand()\*(N))

mutation\_pop[i, j] = pop[i, j] + np.random.uniform(-1, 1)

return mutation\_pop

5. Size of the initial generation is 5000, generated program:

random\_data = np.random.uniform(-4, 4, size=(M, N+1))

initial\_pop = random\_data[:, :9]

theta = random\_data[:, 9]

6. When a tuple of (w1, ..., wn, theta) fitness value is L(length of of collection of tuples of the form (x1, ..., xn, l)), we can stop the evolution. I assume the iteration times is *gen*, if there is no tuple’s fitness value is L, then return the tuple which has the largest fitness value. Otherwise, when finding the tuple whose fitness value is L, return this tuple immediately.

def generation(iteration\_times, pop):

result\_pop = np.zeros(N)

max\_data = 0

result\_index = 0

for i in range(iteration\_times):

fitness = fitness\_function(pop)

if max(fitness) > max\_data:

max\_data = max(fitness)

result\_index = fitness.index(max\_data)

result\_pop = pop[result\_index]

if max\_data == data\_scale:

return [result\_pop, result\_index, i]

else:

select\_pop = copy(fitness, pop)

crossover\_pop = crossover(pc, select\_pop)

mutation\_pop = mutation(pm, crossover\_pop)

length = len(pop)

pop[0:length/2] = select\_pop[:]

pop[length/2:length] = mutation\_pop[:]

return [result\_pop, result\_index, iteration\_times]

7. The output of the system is different from times to times, there is one of the outputs:

result\_pop: array([-0.87432598, -0.24358244, 3.61997334, -2.30332836, 6.13514847,

1.67799262, -6.0032493 , -2.40655813, 3.77078857])

theta: 0.9097341958873502

generation\_times: 53

Q4:

Run:

Python pacman.py -p PSAgent -l smallMap

Python pacman.py -p PSAgent -l ultMap

Q5:

1. According to north.csv, I found that only s1 or s8 equals to 1 and s2 and s3 can’t be 1, the robot will move north, so the boolean expression corresponding to the perceptron for the move north action is: wpsoffice

2. Run:

Python pacman.py -p PSAgent -l testMap  
Python pacman.py -p PSAgent -l smallMap

Python pacman.py -p PSAgent -l ultMap