HW 2

Question 1

a) Let V be the possible points in the space where I can place the sensors. For a set of points A ⊆ V. Let F(A) be area covered by sensors placed at points in A. Suppose Si is the area covered by sensors placed ij points, then F(A) will be the union of the area covered by sensors. Therefore, when a new sensor is added to the space at point p, let B be the set of points A union p. F(B) = F(A union p). Therefore, extra area covered by adding the new sensor at p = F(B)-F(A). Let this area be denoted by S'(p).

Consider a space where we had already deployed 5 sensors. When we add a 6th sensor, it provides more additional information as compared to a case in which we had deployed 100 sensors initially, and then add 101th sensor. S'(6)> S'(101). This property is called diminishing returns. The problem of placing these sensors is a submodular problem.

We can utilize the property of submodularity to provide near optimal solutions which perform well in practice ,when we use the greedy algorithm. The theorem given by Nemhauser et al. [1978] is given by-

For monotonic functions, the quality of the set of elements Agreedy obtained by the greedy algorithm is within a constant factor of the optimal solution A* to Eq.(1).

F(Agreedy) ≥ (1-1/e) F(A*)
The result of the above theorem gives a performance guarantee of at least 63% of the optimal.

b) The algorithm described in the question is the cost greedy benefit algorithm, in which we select greedily a sensing location s* until the budget exceeds:

$s^* = \text{argmax}\ ( (F(A \cup \{s\}) - F(A))/c(s) )$ . Where the numerator is the marginal benefit and c(s) is marginal cost. The algorithm takes into account both the marginal cost and the marginal benefit while choosing. So if suppose we can gain a lot of information from s* and less costly (traveling cost to s* is less), then it would give a better value and be chosen. Suppose another point is more beneficial than s* but the cost is too high, then that point won't be considered by the algorithm.

The cost-benefit greedy performs poorly in some cases. For example, Let's consider two points a and b. F(p) for a is $2\epsilon$ and for b be 1. C(p) for a is $\epsilon$ and for b is one. When the algorithm maximizes the function, the algorithm picks a over b as F(p)/C(p) for a is 2 and is higher than that for b. Let C(A) <=1. Once it picks a, it cannot afford b and cannot pick b as available cost now is $1-\epsilon$. If the value of epsilon is very low, benefit from a is also very less and could reduce overall system performance compared to selecting b. Thus, the algorithm performs arbitrarily bad. Therefore, it gives no performance guarantee.

So, if the system has complex costs, the cost-benefit greedy can be misled to choose less beneficial paths or locations. As the epsilon decreases, we get lesser information from choosing point a. And we are not left with enough cost for b, leading to poorer system performance.

Question 2

Step 1

To calculate the information quality, I got the prediction image from the WorldEstimatingNetwork class and used the values of the pixels in the prediction image. I got the robot's current location and moved it to the neighbor (left,right,up or down) with maximum pixel value or estimated information. So, the robot moves such that it can access more useful information by moving to the particular neighbor.

I used the softmax output value to decide when to move to a corner and stop moving around greedily. The output of the DigitClassificationNetwork() was passed through the logSoftmax() function of torch and thus had negative values. I got the softmax value by normalizing the output of the class using the formula:

$$S(f_{y_i}) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$

Intuitively, what the softmax does is that it converts a vector of numbers into a vector of probabilities. This gives us an array of size 10 (number of digits from 0 to 9) with a probability of the map number being that particular digit respectively.I stop the greedy search when the probability of the map number being that particular digit is greater than 0.9. If none of the digits converges to 0.9 in 300 steps of the

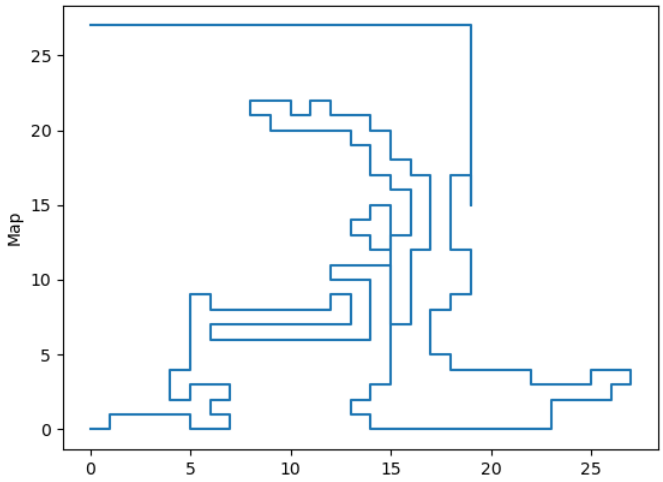
robot, I stop searching greedily and move towards the goal of the digit with the highest probability.

Example Trajectories-

1) Map number -1

Score = -118

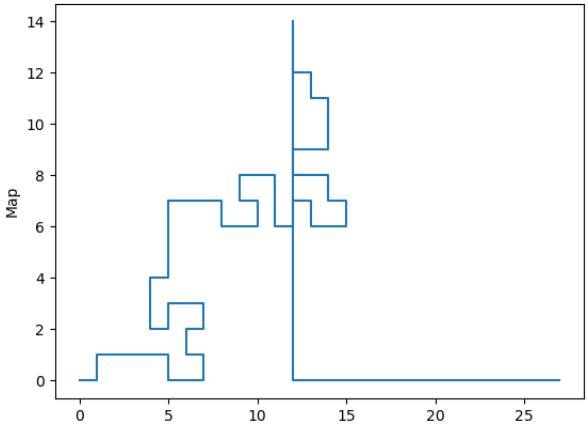
Found goal after step: 219




2) Map number- 2

Found goal at time step: 87

Score: 14




Average reward over ten trials is -194.2.

Step2

I used a sampling based algorithm. The robot picks three random points on the map and chooses the point which has the maximum pixel value. The robot then moves towards this point. Similarly, the robot samples three points randomly, and moves towards the point with maximum pixel value. The process continues until either the probability of any of the digits in the softmax output is more than or equal to 0.9 or the number of steps taken by the robot is greater than 300. Once the probability of softmax output reaches 0.9, the robot moves towards the corner associated with that digit.

Algorithm:

```
Predicted map = get from  WorldEstimatingNetwork class
if map number predicted probability >0.92 or no. of pixels visited > 300:
    goal = get goal(predicted map number)
    move towards = goal
else:
    get three unvisited random points
    Intermediate goal = get maximum pixel value (the three random points)
    move towards = Intermediate goals
```

The average reward over 10 trials is -158.7. The average time taken to find the solution is 0.94 s.

Discussion:

1) I chose a sampling based approach as it allows the robot to consider points in the grid which are not adjacent to or near the robot's current location. This speeds up the process of exploring points on the map, before reaching the 0.9 probability threshold of softmax output. I also used the pixel value information to choose between the three sampled points. Aarhus, we reduce the chances of choosing uninformative points. The design criteria used is reducing the number of steps traveled by the robot and increasing the visibility of the map.

   We can compare this algorithm to A*. Using A* would result in a huge cost as it would explore and visit a lot more points on the grid before it reaches the goal. If we use RRT for this problem, we will randomly sample points and check for feasible points. It won't use information of the pixel values for the sampled points to select the best point.

2) I would subtract the value of uncertainty in prediction from the softmax value before normalizing the exponential. This would have provided me with a more accurate prediction of the digit. That would reduce the chances of moving towards the wrong corner and thereby incurring a big penalty. I would have used a lower probability value than 0.9 to stop searching and move towards the goal. This would help the robot to reach the goal a bit faster and improve the value of reward received.
3) I would modify my algorithm to update the predicted image after each robot moves. Additionally, I would check for probability convergence after each robot moves. This would improve the accuracy of the prediction and thus improve the reward. With multiple robots, different areas of the room can be explored faster. I would check that the robots are moving towards different corners, are not exploring and visiting the same pixels. I would incentivize exploration of robots away from each other by giving an additional reward for that.