CS 486/686: Assignment 4 (100 Points)

Instructors: Jesse Hoey & Victor Zhong
Due Date: Wednesday April 2nd, at 11:59 pm (ET; Waterloo Time)

Instructions

- Submit written solutions in a file named `writeup.pdf` and code solutions in a file (named `reinforcement.py`) to the A4 Dropbox on `Learn` (`https://learn.uwaterloo.ca`). Please **do NOT zip your files**.
- No late assignment will be accepted.
- This assignment is to be done individually.
- Use the latest Python 3.12 to implement the code.
- Lead TAs:
  - Yuxuan Li (`y624li@`)
  - Shivani Upadhyay (`sjupadhy`)
  - Dake Zhang (`dake.zhang@`)

  The TA office hours will be scheduled on Piazza.

1 Expectation Maximization (50 points)

Olertawo is a university town in New Zealand in which there is a strange medical condition called Dunetts Syndrome. Dunetts Syndrome comes in two forms: mild and severe (as well as not being there at all). It is known that about half the population of Olertawo have Dunetts, and that about half of those with Dunetts have it in severe form. Dunetts Syndrome has three observable symptoms, Sloepnea, Forienndditis, and Degar spots, all of which may be present if the patient has Dunetts, but with varying frequencies. In particular, it is well known that Forienndditis is present much more often when the condition is in its mild form (it is not as common in severe cases), whereas Degar spots are present much more often when the condition is in its severe form (and not as common in the mild cases). Sloepnea is likely to be present in either form of Dunetts Syndrome (mild or severe), but is not likely to be present when Dunetts Syndrome is not present. However, about 10% of the population have a gene (called TRIMONO-HT/S) that makes it so they hardly ever show symptom Sloepnea (whether they have Dunetts Syndrome or not), but does not affect the other two symptoms. Symptoms Sloepnea, Forienndditis, and Degar spots are sometimes present (but much less often) even if the person does not have Dunetts Syndrome.

1. Construct a Bayesian network (BN) for the domain explained above. Assign priors to each CPT based on the prior information given above. You don't need to be precise - just make rough guesses as to what the CPTs may be based on the description above.

2. You are given a dataset from 2000 patients, giving the existence of the three symptoms Sloepnea, Forienndditis and Degar spots, and whether they have gene TRIMONO-HT/S or not. About 5% of the data also has a record of whether the patient actually had Dunetts Syndrome or not. Using the Expectation Maximization (EM) algorithm, learn the CPTs for your BN. Run EM until the likelihood of the complete data (the sum of all your "weights" over Dunetts Syndrome) only changes by 0.01 or less.

   Start EM from your prior model, but add a small amount δ of random white noise to each parameter. Do this by adding a different randomly generated number $r \in [0, \delta]$ to each probability in each CPT, and then normalizing. So if you have a distribution $[p, 1-p]$ you draw two random numbers δ_1 and δ_2 between $[0, \delta]$ for whatever δ you are using, and then you compute $\left[\frac{p+\delta_1}{1+\delta_1+\delta_2}, \frac{(1-p)+\delta_2}{1+\delta_1+\delta_2}\right]$. Repeat the EM learning for a range of settings of δ from $\sim$ 0.0 to 4.0, and do 20 trials for each setting with different randomizations at the start of each trial (but not at the start of each EM iteration). Use at least 20 values of δ evenly spread in $[0, 4)$. The idea is to evaluate the sensitivity of EM to the initial guess. As δ gets bigger, your initial CPTs will become more and more random, and you should get increasing numbers of trials with low accuracy. If your initial guess is not very good, you may even find that adding a small amount of noise helps.

3. To validate your learned model, you are given 100 test instances in which it is known whether the patient had Dunetts Syndrome. Make a prediction of whether Dunetts Syndrome is present for each example in the test set based on your learned model using EM and compare with the actual values of Dunetts Syndrome to find accuracy. Generate the plot described below.

The datasets are available on LEARN.

- `trainData.txt` is a file of 2000 training examples with five columns. The first three give the presence/absence of each symptom (in order Sloepnea, Forienndditis, Degar spots with 0 indicating the symptom is not present, and 1 indicating it is). The fourth column gives the presence/absence of gene TRIMONO-HT/S (with 0 indicating the gene is not present, and 1 indicating it is). The fifth column is -1 if there is no record of Dunetts Syndrome, and $0, 1, 2$ if Dunetts Syndrome is recorded as being not present, mild or severe, respectively.
- `testData.txt` is the 100 examples for testing, has an additional column giving the severity of Dunetts Syndrome (the last column: 0=none, 1=mild, 2=severe).

What to hand in:

1. **[10 pts]** A drawing of your Bayesian network showing all CPTs.
2. **[15 pts]** Your code for doing EM on this model. You may paste your code in this writeup. You don't have to write code to do EM in general, just for this specific example.
3. **[25 pts]** A graph showing the prediction accuracy on the test set, for each value of δ, giving mean accuracy and standard deviation (error bars) over the 20 trials. Plot both the accuracy before and after running EM on the same set of axes.

2 [50 points] Reinforcement Learning

In this problem, you will study the emergence of communication in a two-agent reinforcement learning problem.[1] One agent, the receiver, is in a 5×5 grid-world (with walls) and starts at the center location and can move horizontally or vertically only and cannot move through walls. A prize is hidden (from the receiver) somewhere on the grid. The prize is placed randomly but cannot be in a wall location, and is never placed at the receiver's starting position. The second agent, the sender, knows the location of the prize only, and can send the receiver a single message as one of a set of N arbitrary symbols. Thus, since the symbols have no "meaning" at the start, the agents have to jointly establish this meaning for each of the N symbols. For example, if $N = 25$, the sender can precisely specify where the prize is, but the two agents still have to assign meaning to the symbols. If $N < 2$, the sender is useless.

The sender and the receiver each get the prize (a reward of 1.0 each) upon the receiver entering the cell that contains the prize. After receiving the sender's message, the receiver acts until it finds the reward or is terminated. The receiver is terminated at each step with some fixed probability, δ. These two agents must somehow figure out a messaging mechanism (they must build an agreed-upon language) that allows the receiver to find the prize most quickly.

Implement two Q-learning agents, one for the receiver, and one for the sender. The sender agent's states are the location of the prize on the grid. Its actions are the possible symbols (messages) it can send. The receiver agent's states are tuples giving the received message and the current location (which is fully observable).

[1] This problem is based on the following paper: Ivana Kajić, Eser Aygün and Doina Precup. Learning to cooperate: Emergent communication in multi-agent navigation arXiv:2004.01097, 2020 `https://arxiv.org/abs/2004.01097`

The Q-functions for these agents can simply be implemented as tables or matrices that you fill in with a nested loop.[2] The outer loop consists of the sender taking an action (starting an "episode"), while the inner loop consists of the receiver acting until it is terminated. When terminated, the receiver's learning episode ends. The receiver has a δ chance of being terminated after each action it takes, and receives a reward of 0 if this happens. It is also terminated (and the episode ends) after moving into the prize's grid cell, but receives a reward of 1.0.[3] The prize is placed randomly at each episode, never at a wall, and never at the start location. The receiver is not penalized for moving into a wall (but the move is ineffective).

You will train the Q-learning agents over a series of N_{ep} episodes, each consisting of a sender action (message), followed by a series of receiver actions (moves until termination). Each agent updates its Q-function after each of its iterations. Initialize your Q functions as all zeros. Note that the sender always enters a random next state, so we drop the $\gamma \max_{a'} Q[s', a']$ term from its Q update. (This is equivalent to setting $\gamma = 0$ for the sender.)

Your Q-learning agents should use ϵ-greedy exploration during training, and should switch to greedy actions when testing (switch to $\epsilon = 0$ for testing). Use a learning rate α which starts at $\alpha_0 = 0.9$ and decreases linearly down to $\alpha_f = 0.01$ over the episodes until N_{ep} episodes.[4] Use a discount factor of $\gamma = 0.95$ and a termination probabilty of $\delta = 1 - \gamma$ for the receiver.

You will use the grids shown in Figure 1. The start position is always the center (marked "*") and the prize location is randomly selected at each episode, but is never at the start location and is never in a wall location (or off the grid).

A Python file, `reinforcement.py`, has been provided to get you started. You may change it however you like, or elect to start from scratch on your own. At a minimum, you will need to add code to all locations that say `# Your code here!` in order to get something that works. You will also need to add code to the `if __name__ == "__main__:"` block to run the experiments below and produce the required plots.

Submit your code in a single second file (named reinforcement.py), and then answer the following questions.

1. **[8 pts]** Consider an empty 5×5 grid. What is the discounted average reward for the optimal search strategy for the receiver to take if the sender is useless (you can implement this by giving the sender only one possible message to send) and the receiver is only terminated when it reaches the prize? There may be many such optimal search strategies. Note however that the receiver can only move in a single direction from a grid cell (it has a history of 0), so the optimal search strategy will not cross its own path. Draw an optimal search strategy on 5×5 grid stating at the middle, and write the discounted average reward using $\gamma = 0.95$. This calculation is done by hand.

[2]You can use any other Q-function representation if you prefer, e.g. a deep neural network, but a table will suffice for this example. To simplify the implementation, you can include a "dummy" representation for the wall states, i.e., the Q-table can have rows that are never updated. However, you may have a fancier representation that doesn't waste the wall space.

[3]You may feel a certain sadness for the receiver in this case, but you must understand that your receiver re-spawns instantly when terminated, and keeps its reward, and will continue to do so indefinitely!

[4]That is, decrease the learning rate by $\frac{\alpha_0 - \alpha_f}{N_{ep}}$ after each episode.

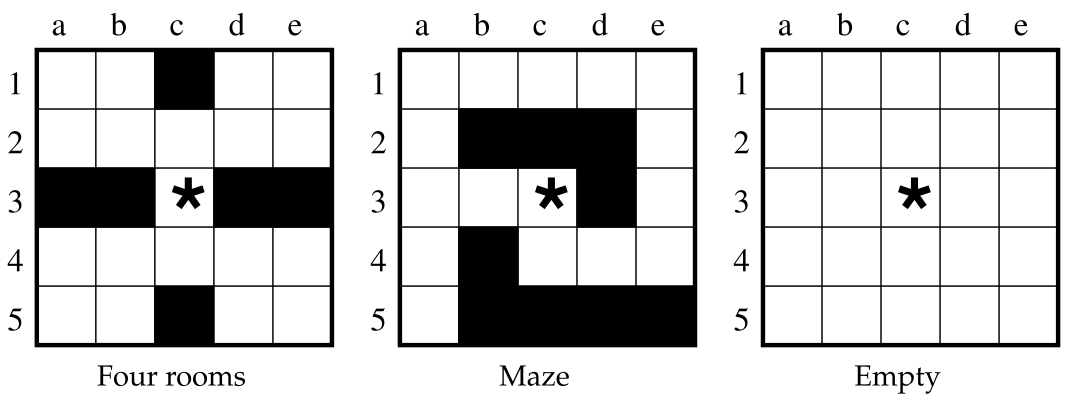

Figure 1: Domains to be used. The robot start location is shown with a "*".

2. **[15 pts]** Using the "four room" grid shown above and $N = 4$ messages, run 10 tests (each test being a set of N_{ep} episodes) for each of $N_{ep} = 10, 100, 1000, 10000, 50000, 100000$ and for each of $\epsilon = 0.01, 0.1, 0.4$. Use the integers $0, 1, 2, 3$ as the N symbols (sender messages). Each episode is a single update step for the sender, but is a randomized number of updates (depending on δ) for the receiver. After each test, run the trained agents for an additional 1000 episodes using $\epsilon = 0$, measuring the discounted reward obtained by either sender or receiver. Plot the average discounted reward received as a function of $\log(N_{ep})$ for each value of ϵ. Plot error bars as standard deviations over the 10 tests. Show an example of a policy for the $\epsilon = 0.1$ case with $N_{ep} = 100000$. Show 4 grids, one for each sender message (integer) with a $\rightarrow$ in each grid cell showing the direction that would be taken. Show a fifth grid for the sender policy.

3. **[8 pts]** Using $\epsilon = 0.1$, run the same set of tests for different N values of $N = 2, 4, 10$. Show a single graph with three lines on it with errorbars showing average discounted reward of for each N values as a function of N_{ep}.

4. **[8 pts]** Using the "maze" grid, run the same 10 tests as above, but for each of $N = 2, 3, 5$ and report the same graph of the average discounted reward as in the last question. Use only $\epsilon = 0.1$

5. **[8 pts]** Finally, using the "empty" grid, run a further 10 tests. Use only $\epsilon = 0.1$ and $N = 1$. Again, report the same graph.

6. **[3 pts]** How many iterations of Q-learning to you think you will need to run to learn the optimal search strategy for the empty domain with one message (i.e. with no sender). Use your graph from Question 5 and your result from Question 1.

What to hand in:

- A printout of your code as a separate file.
- Question 1: a grid showing the optimal search strategy and the average discounted reward.

- Question 2: A graph showing average discounted reward received as a function of N_{ep} for each value of ϵ
- Question 2: An example of four graphs showing the policy from each cell using $\rightarrow$,$\leftarrow$,$\uparrow$, and $\downarrow$.
- Question 3: A graph showing average discounted reward received as a function of N_{ep} for each value of $N = 2, 4, 10$.
- Question 4: A graph showing average discounted reward received as a function of N_{ep} for each value of $N = 2, 3, 5$.
- Question 5: A graph showing average discounted reward received as a function of N_{ep} for $N = 1$.
- Question 6: An estimate of the number of iterations of Q-learning will be needed to solve the empty room problem.