Artificial Intelligence

Fall 2017: Practice Final

Instructions: This exam is governed by the **Emory Honor Code**. This exam is closed book and closed notes. However, you may use 2 sheets of notes. You may also use an ancient calculator (not an app but the actual device). You will have 90 minutes for this exam. The point values are indicated beside each problem.

Name (print): _____Solution_____

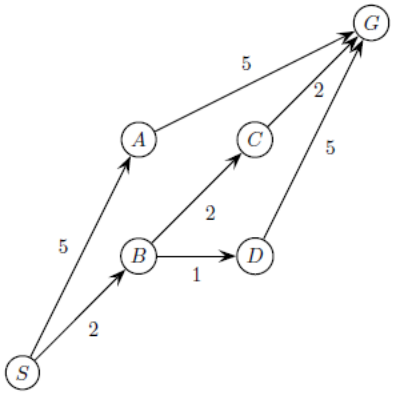
This table is for grading, please leave it blank.

Problem	***Points***	***Score***
1: Search	10	
2: Games	10	
3: Inference and HMMs	10	
4: MDP	10	
5: RL (classical)	10	
6: Neural Networks & Deep RL	20	
Total	80	

This page is intentionally left blank (use it as you like)

Problem 1 (10 pts): Search

For questions a-c, consider the search problem on the right:



(a) (2pts): Which of the heuristics shown are admissible?

h0, h1

Node	h_0	h_1	h_2
S	0	5	6
A	0	3	5
B	0	4	2
C	0	2	5
D	0	5	3
G	0	0	0

(b) (2pts): Give the path A* search will return using heuristic h_2 ?

H0: S B C G
H1: S B C G
H2: S B D G

(c) (2pts): Which path will greedy best-first search return using h_1 ?

S A G

For questions **d** and **e**, consider the following variations of the **fringe** in the **A* tree search algorithm**. In all cases, *g* is the cumulative path cost of a node *n*, *h* is a lower bound on the shortest path to a goal state (e.g., Manhattan distance). Assume all costs are positive, and all heuristics *h* are admissible.

(i) Standard A*

(ii) A*, but we apply the goal test before enqueuing nodes, rather than after dequeuing

(iii) A*, but prioritize fringe by g(n) only (ignoring h(n))

(iv) A*, but prioritize fringe by h(n) only (ignoring g(n))

(d) (2pts): Which of the above are *complete* (always finds goal if exists)?

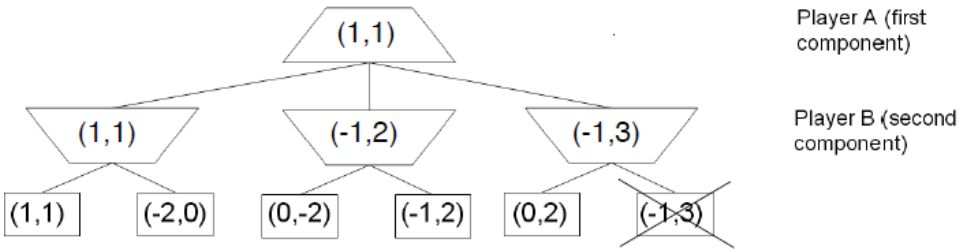
i, ii, iii. All but (iv) are complete if costs are positive, none are complete if costs can be zero (we accepted both answers). For a tree search to be incomplete, there must be some constant c such that there is an infinite path in the tree where all nodes have priority less than c. That can't happen with positive costs and g in the priority. (iv) is greedy search, which is clearly not complete for h = 0 in general infinite graphs.

(e) (2pts): Which of the above are *optimal* (always finds cheapest path to goal)?

i, iii only. (i) is optimal, as shown in class. (ii) is suboptimal, as shown by an example in lecture (e.g. when the last arc of a suboptimal is really expensive but the prefix is cheap). (iii) is UCS, a subcase of (i). (iv) is greedy, which we saw was suboptimal in class.

Problem 2 (10 pts): Games

The standard Minimax algorithm calculates worst-case values in a zero-sum two player game, i.e. a game in which for all terminal states s, the utilities for players A (MAX) and B (MIN) obey $U_A(s)+U_B(s) = 0$. In the zero-sum case, we know that $U_A(s) = -U_B(s)$ and so we can think of player B as simply minimizing UA(s). In this problem, you will consider the non zero-sum generalization, in which the sum of the two players' utilities are not necessarily zero. Because player A's utility no longer determines player B's utility exactly, the leaf utilities are written as pairs (U_A, U_B), with the first and second component indicating the utility of that leaf to A and B respectively. In this generalized setting, A seeks to maximize U_A, the first component, while B seeks to **maximize** U_B, the second component.



(a) (5pts) Propagate the terminal utility pairs up the tree using the appropriate generalization of the minimax algorithm on this game tree. Fill in the values (as pairs) at each of the internal node. Assume that each player maximizes their own utility. Hint: just as in minimax, the utility pair for a node is the utility pair of one of its children.

(b) (2pts) Briefly explain why no alpha-beta style pruning is possible in the general non-zero sum case. Hint: think first about the case where $U_A(s) = U_B(s)$ for all nodes.

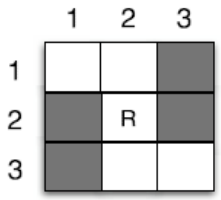
The values that the first and second player are trying to maximize are independent, so we no longer have situations where we know that one player will never let another player down the branch of the game tree. For instance, in the case where $U_A(s) = U_B(s)$, this problem reduces to searching for the max valued leaf, which could appear anywhere in the tree.

(c) (3 points) For Minimax, we know that the value *v* computed at the root (say for player A = MAX) is a worse-case value, in the sense that, if the opponent MIN doesn't act optimally, the actual outcome v_0 for MAX can only be better, never worse, than *v*. In the general non-zero sum setup, can we say that the value U_A computed at the root for player A is also a worst-case value (as in Minimax), or can A's outcome be even worse than the computed U_A if B plays suboptimally? Briefly justify your answer.

YES or NO?
Justification:

YES. A's outcome can be worse than the computed U_A. For instance, in the example game, if B chooses (-2,0) over (-1,2), then A's outcome will decrease from 1 to -1.

Problem 3 (10pts): Inference



Robot Localization Grid

Suppose you are a robot navigating a maze (see figure 1), where some of the cells are free and some are blocked. At each time step, you are occupying one of the free cells. You are equipped with sensors which give you noisy observations, (w_U, w_D, w_L, w_R) of the four cells adjacent to your current position (*up*,*down*,*left*, and *right* respectively). Each w_i is either *free* or *blocked*, and is accurate 80% of the time, independently of the other sensors or your current position. [1]

Imagine that you have experienced a motor malfunction that causes you to randomly move to one of the four adjacent cell with probability $\frac{1}{4}$.[2]

a) Suppose you start in the central cell in figure 1. One time step passes and you are now in a new, possibly different state and your sensors indicate (*free*,*blocked*,*blocked*,*blocked*). Which states have a non-zero probability of being your new position?

Solution:$(2,1), (2,2), (2,3)$

b) Give the posterior probability distribution over your new position.

Solution: The posterior probabilities of each state are proportional to the probability of ending up in the state after one random move weighted by the probability of the observation from that state:

$$\begin{aligned} P(s_1 = (2,1)|w_U, w_D, w_L, w_R) &\propto \frac{1}{4} \cdot 0.8^1 \cdot 0.2^3 = 0.0016 \\ P(s_1 = (2,2)|w_U, w_D, w_L, w_R) &\propto \frac{1}{2} \cdot 0.8^3 \cdot 0.2^1 = 0.0512 \\ P(s_1 = (2,3)|w_U, w_D, w_L, w_R) &\propto \frac{1}{4} \cdot 0.8^3 \cdot 0.2^1 = 0.0256 \end{aligned}$$

Normalizing, we have

$$\begin{aligned} P(s_1 = (2,1)|w_U, w_D, w_L, w_R) &= 0.02 \\ P(s_1 = (2,2)|w_U, w_D, w_L, w_R) &= 0.65 \\ P(s_1 = (2,3)|w_U, w_D, w_L, w_R) &= 0.33 \end{aligned}$$

This distribution matches our intuition: a move upward was quite unlikely because of the observation, which disagrees with the layout around $(2,1)$ in three out of four directions.

Problem 4 (10 pts): Markov Decision Furby

You are designing an evil Furby doll (shown on right) to be *maximally annoying*. *That is, the goal is to maximize the total annoyance reward (*R*), imparted by the toy to the owner's family. The Furby has two states: ON and OFF. If the Furby is destroyed, it stays in OFF state, and there is no future reward. Otherwise, it stays in ON state, since this torture device's battery never runs out. It starts in the ON state.

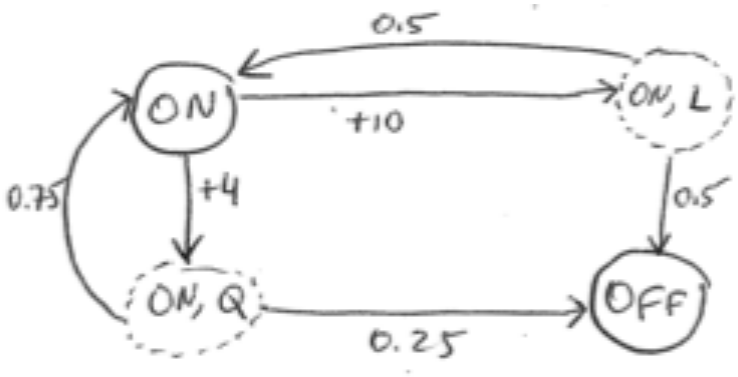
This Furby has two actions:

***L*: Loud** annoying laugh, with immediate annoyance reward +10.

***Q*:** Quiet annoying laugh, with immediate annoyance reward +4.

The probability of the annoyed family destroying the Furby after action ***L*** is 0.5, and after action ***Q*** is 0.25.

(a) (3 pts). Draw the state transition diagram for MDF, clearly marking the **states**, **q-states**, **transition probabilities**, and the **rewards** associated with each action.



(b) (2pts) Write down the recurrence equation for the expected total reward for the policy **πL**="**always-L**", that is, always do action **L**. Assume discount γ is 1 (no decrease in "reward" over time)

$$V(\pi_L) = 10 + 0.5(V(\pi_L)) + 0.5(\emptyset)$$

(c) (2pts) Estimate the total rewards for the policy **πL** = "**always-L**". Assume discount γ is 1.

$$10 + 10 \cdot 0.5 + 10 \cdot 0.5^2 \ldots = \frac{10}{1-0.5} = 20$$

(d) (3pts) What is the optimal policy for this MDF, and associated optimal value? Assume discount γ is 1. Show sufficient work to justify your answer.

$$V(\pi_Q) = 4 + 0.75(V(\pi_Q)) + 0.25(\emptyset) \quad 4 + 4 \cdot 0.75 + 4 \cdot 0.75^2 \ldots = \frac{4}{1-0.75} = \frac{4}{0.25} = 16$$

$$V(\pi_L) = 20$$

FYI: Infinite geometric series: $a+ar+ar^2+ar^3+ar^4+\ldots = a/(1-r)$ (converges to this closed form for r<1)

* Aww it's so cute, you say? Not so much. See here: https://www.youtube.com/watch?v=B_0FWuJuKpk

Problem 5 (10 pts): Reincarnating Q-Learning Furby 2.0

Same states and actions as in Problem 3 above, but with the following changes:

- Furby 2.0 has been *upgraded* with an annoyance sensor, which measures how annoyed the family is, and returns a reward score ranging from 0 (no reward) to +10 (maximum annoyance)
- Furby 2.0 now has reincarnation capability (if destroyed, the Furby re-orders itself from Amazon, and resets itself to ON state). All the (Q-Values) are saved on AWS cloud storage, so training continues…
- The transition probabilities and action rewards are no longer known.

To maximize the total annoyance, the Furby 2.0 implements the Q-Learning algorithm.

(a) 8pts: Compute the Q-Values (that you drew in 3a), **after each training episode below**, provided as sequence of tuples (State, Action, NextState, Reward). Assume α=0.5, and γ =1. Show work.

Episode 1: (ON, Q, ON, +4), (ON, L, OFF, +10)

Q-Values:

E1	Q(on, Quiet)	Q(on, Loud)	|| Q(on, Quiet)= 0+0.5(4+max(0,0)=2
0	0	0	|| Q(on,Loud) = 0+0.5(10+0) = 5
1	2	0	
2	2	5	

Episode 2: (ON, Q, ON, +4), (ON, Q, OFF, +4)

Q-Values:

E1	Q(on, Quiet)	Q(on, Loud)	|| Q(on, Quiet)= 2*0.5 + 0.5(4+max(2 , 5) = 1+0.5(4+5) = 5.5
0	2	5	|| Q(on, Quiet) = 5.5*0.5 + 0.5(4+0) = 5.5/2 +2 =2.75+2 = 4.75
1	5.5	5	
2	**4.75**	**5**	

(a) **2 pts:** What **policy** does the Furby learn after 2 episodes of training above?

ON → Loud

OFF: n/a

Problem 6 (20 pts): Neural Networks and Deep RL

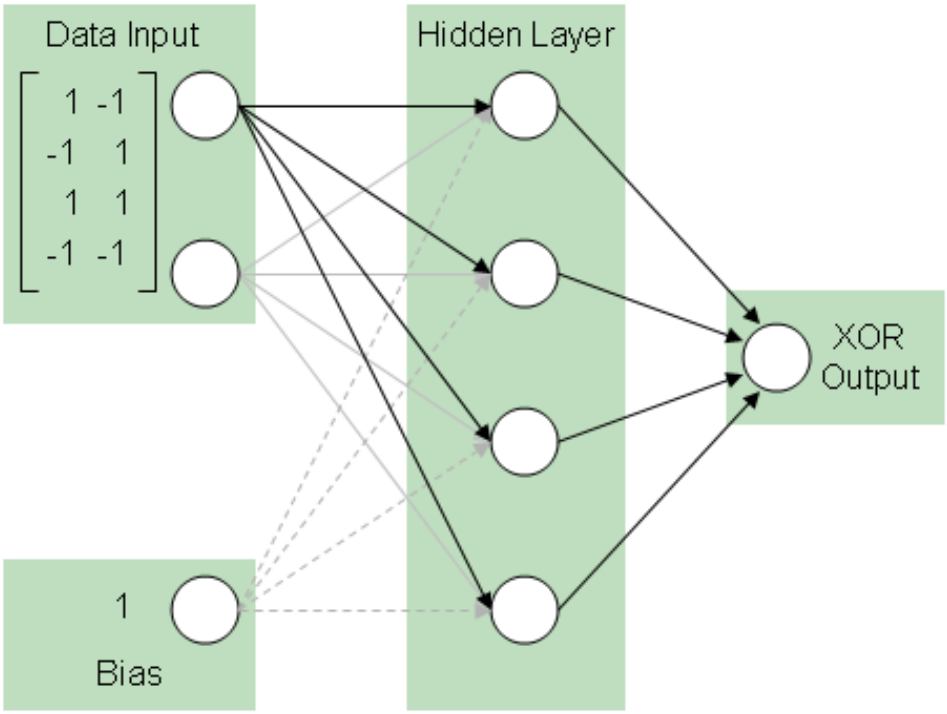
(a) (4 pt) Circle true or false for each statement about a perceptron classifier in general. Assume weight vectors are initialized to 0s.

(i) **F** (true or false) Can produce non-integer-valued weight vectors from integer-valued features
(ii) F (true or false) Estimates a probability distribution over the training data
(iii) F (true or false) Assumes features are conditionally independent given the class
(iv) F (true or false) Perfectly classifies any training set eventually

(**b) (8 pts)** Construct by hand a simple feed-forward neural network, which computes the XOR function of two inputs *a* and *b*. Make sure to specify what sort of units you are using, and weights that would give the desired behavior. The XOR input/output table is provided -→

a	b	XOR
1	1	0
0	1	1
1	0	1
0	0	0

http://www.mlopt.com/?p=160



(d) (3 pts) Can the XOR function be learned by a Perceptron (single-layer) neural network? Why or why not?

No, because XOR is not linearly separable (see many examples online). Need 2nd layer to provide non-linear boundary.

For the problems below, refer to the "mountain car" control problem.

Aim: learn how to drive the car up a big hill to the goal on right, where the car's engine is weak (requires building up enough speed first by going up the little hill on left, away from the goal. The car starts standing at bottom of hill (0,0)
Environment representation:
Observations: car's (position, velocity) (real numbers/scalars)
Actions (accelerator): a real number between (-1,1)
Rewards: -0.1 time penalty, only 1 reward of +100 iff reaches the goal.
Best score possible: about 90 (at least 100 time steps required to reach goal)

(d) (2 pts): Can exact Q-learning work here in finite time? If yes, describe how. If not, explain why not.

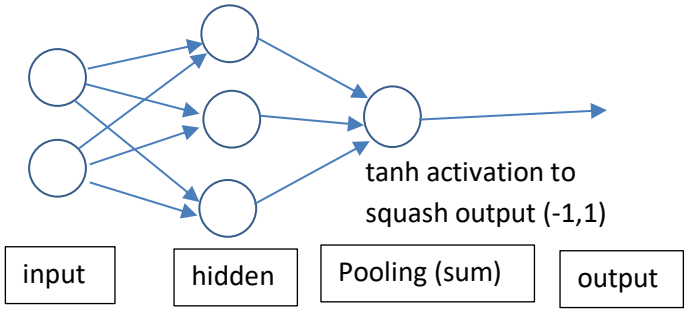
NO, for two reasons:

1. The observation space is continuous (position, velocity are real numbers), action space (accelerator) are also real numbers. There are infinite number of possible states, impossible to create a dictionary with all of them. Possible solution is to **discretize** the observation and action space, which **could** work, but not guaranteed
2. The rewards only come in the end, after many (>>100) steps. Randomized (off-policy) exploration of Q-learning will reach the goal (positive reward) with very small probability, which would require a very large (effectively infinite) amount of training episodes.

(e) (2 pts): Can approximate Q-learning work here in finite time? If yes, describe how. If not, explain why not.

No, for reason 2 above: even with discretizing the states as proposed in 1 above, the issue of extremely delayed rewards reachable in only tiny fraction of attempts makes classic (even approximate) Q-learning either require effectively infinitely many episodes or extreme luck.

(f) (4 pts): Draw a **simple** policy neural network that could be used to directly learn an effective policy for this problem. Make sure to specify exactly the input layer to match the observations, and output layer to match the action space. Note: might also learn with only 2 hidden layer neurons, but 3 or 4 seem better in my experiments.



(g) 2 pts: Let's use a **Policy Gradient** (PG) method to train your network. Write 1-2 lines of pseudocode just for the parameter update step. Must specify loss w.r.t. to total episode reward. For simplicity, assume back-propagation is implemented and available.

Loss: (p-y)^2 * \gamma^t * reward,
where *p* is the prediction, *y* is the action (e.g., -1 or 1) and \gamma is the discount for the final reward for the episode, t steps from the end of the episode counting backwards

Update: SGDUpdate (network, Loss), where SGD is a stochastic gradient descent update in tensorflow or any other NN package.

End of Exam.