CS 188 Spring 2023 Introduction to Artificial Intelligence Final

Solutions last updated: Monday, May 15

- You have 170 minutes.
- The exam is closed book, no calculator, and closed notes, other than four double-sided cheat sheets that you may reference.
- For multiple choice questions,
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**

First name	
Last name	
SID	
Name of person to the right	
Name of person to the left	
Discussion TAs (or None)	

Honor code: "As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others."

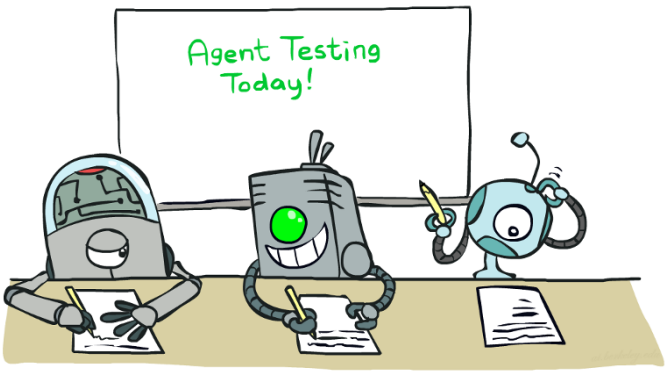
By signing below, I affirm that all work on this exam is my own work, and honestly reflects my own understanding of the course material. I have not referenced any outside materials (other than my cheat sheets), nor collaborated with any other human being on this exam. I understand that if the exam proctor catches me cheating on the exam, that I may face the penalty of an automatic "F" grade in this class and a referral to the Center for Student Conduct.

Signature: _____

Point Distribution

Q1.	Search and Slime	12
Q2.	Searching for a Path with a CSP	11
Q3.	A Multiplayer MDP	9
Q4.	Slightly Different MDPs	13
Q5.	Bayes' Nets	8
Q6.	Dynamic Bayes' Nets: Book Club	10
Q7.	Inequalities of VPI	7
Q8.	Inverted Naive Bayes	13
Q9.	Higher-Dimensional Perceptrons	8
Q10.	Q-Networks	9
	Total	100

It's testing time for our CS188 robots!
Circle your favorite robot below. (ungraded, just for fun)



Q1. [12 pts] Search and Slime

Slowmo the snail is trying to find a path to a goal in an $N \times M$ grid maze. The available actions are to move one square up, down, left, right, and each action costs 1. Moves into walls are impossible. The goal test is a black-box function (the snail does not know how the function works) that returns true if its state argument is a goal state, and false otherwise.

(a) [1 pt] Assume that there is a reachable goal state. Which algorithms are **complete** for this problem? Select all that apply.

- [x] Breadth-first tree search
- [ ] Depth-first tree search
- [x] Uniform-cost tree search
- [x] A* tree search with $h = 0$
- [x] A* graph search with $h = 0$
- ◯ None of the above

Depth-first can fail because there are cyclic paths which lead to infinitely deep branches in the tree. The other algorithms are all basically breadth-first search because costs are 1, and because the branching factor is finite they are bound to find the goal.

(b) [1 pt] Assume that there is a reachable goal state. Which algorithms are **optimal** for this problem? Select all that apply.

- [x] Breadth-first tree search
- [ ] Depth-first tree search
- [x] Uniform-cost tree search
- [x] A* tree search with $h = 0$
- [x] A* graph search with $h = 0$
- ◯ None of the above

As before, depth-first search may find a suboptimal solution or none at all, while the others are essentially breadth-first search, which is optimal for constant step costs.

(c) [2 pts] Assume that there *may or may not be* a reachable goal state. Which algorithms are **complete** for this problem? Select all that apply.

- [ ] Breadth-first tree search
- [ ] Depth-first tree search
- [ ] Uniform-cost tree search
- [ ] A* tree search with $h = 0$
- [x] A* graph search with $h = 0$
- ◯ None of the above

If there is no goal, all the tree search algorithms will follow cyclic paths indefinitely, so they will fail to report that no solution exists.

However, graph search will not expand any state more than once. This problem has a limited number of states, so once graph search finishes expanding all the states, it will report that no solution exists and terminate.

For the rest of the question, consider a modified version of the search problem: Slowmo the snail hates entering squares that are already covered in his own slime trail. This means that an action is impossible if it would enter a square that Slowmo has previously entered. There *may or may not be* a reachable goal state in this problem.

(d) [2 pts] What is the size of the state space (in terms of N and M) required to correctly reflect this modified problem?

$MN \cdot 2^{MN}$

The state space needs to include one bit for each location, indicating whether it has been slimed. There are MN locations, so there are 2^{MN} possible sequences of MN bits indicating which locations have been slimed. Also, we need to include a factor of MN to represent Slowmo's location.

(e) [2 pts] In this modified problem, which algorithms are **complete**? Select all that apply.

- [x] Breadth-first tree search
- [x] Depth-first tree search
- [x] Uniform-cost tree search
- [x] A* tree search with $h = 0$
- [x] A* graph search with $h = 0$
- ◯ None of the above

All the algorithms are complete because every path is of finite length (no more than MN steps). The memory of previously visited squares means that tree searches function more or less like graph searches.

(f) [2 pts] In this modified problem, which algorithms are **optimal**? Select all that apply.

■ Breadth-first tree search ■ Uniform-cost tree search ■ A* graph search with $h = 0$

☐ Depth-first tree search ■ A* tree search with $h = 0$ ○ None of the above

In this particular problem, the optimal path with slime memory is the same as the optimal path without slime memory, because there is no benefit to moving back into a square that you've already visited. As before, with constant costs, the algorithms other than depth-first all function like breadth-first, which is optimal.

(g) [2 pts] Consider an arbitrary search problem. We want to modify this search problem so that the agent is not allowed to move into a state more than once.

Which of the following changes to the original problem, each considered in isolation, correctly represents this modification? Select all that apply.

☐ Change the successor function to disallow actions that move into a previously-visited state.

■ Add a list of visited states in the state space, and also change the successor function to disallow actions that move into a previously-visited state.

☐ Change the goal test to return false if a state has been visited more than once.

■ Add a list of visited states in the state space, and also change the goal test to return false if a state has been visited more than once.

○ None of the above

The key realization in this problem is that the successor function and goal test are black-box functions that take in only the state, and then need to report the list of successor states (in the successor function), or whether or not the state is a goal (in the goal test). In our definition of a search problem, a list of previously-visited states is not built into the problem, so these functions cannot access this list unless we explicitly create it. Therefore, we need to encode the previously-visited states into the state representation so that it can be passed into the successor function and goal test.

(A) False. The successor function has no way to look at a state and determine if it has been previously visited before.

(B) True. This fixes the issue with (A), since now the state will tell you which states have been visited before.

(C) False. The goal test has no way to tell which states have been visited.

(D) True. This fixes the issue with (C).

Q2. [11 pts] Searching for a Path with a CSP

We have a graph with directed edges between nodes. We want to find a simple path (i.e. no node is visited twice) between node s and node g.

We decide to use a CSP to solve this problem. The CSP has one variable for each edge in the graph: the variable X_{vw} represents whether the edge between node v and node w is part of the path. Each variable has domain {true, false}.

In each of the next three parts, a constraint is described; select the logical expression that represents the constraint. Notation:

- $N(v)$ is the set of all neighbors of node v. (Recall: a node w is a neighbor of v iff there is an edge from v to w).
- The function `ExactlyOne()` returns an expression that is true iff exactly one of its inputs is true.
- $\bigwedge_{x_i \in S} x_i$ refers to the conjunction of all x_i in the set S.
- $\bigvee_{x_i \in S} x_i$ refers to the disjunction of all x_i in the set S.

(a) [2 pts] The starting node s has a single outgoing edge in the path and no incoming edge in the path.

- ● $\texttt{ExactlyOne}(\{X_{sv} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(s)} \neg X_{vs}$
- ○ $\texttt{ExactlyOne}(\{X_{sv} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(s)} X_{vs}$
- ○ $\texttt{ExactlyOne}(\{X_{vs} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(s)} \neg X_{sv}$
- ○ $\texttt{ExactlyOne}(\{X_{vs} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(s)} X_{sv}$

The variables X_{sv}, one per node v, represent the outgoing edges from node s to other nodes v. We want exactly one of these variables to be true, representing the single outgoing edge from the start node.

The variables X_{vs}, one per node v, represent the incoming edges from other nodes v to node s. We want all of these variables to be false, representing no edges incoming to s. In other words: $\neg X_{vs}$ says that the incoming edge (v, s) is not included in the path, and we want this to hold for all neighbors v.

(b) [2 pts] The goal node g has a single incoming edge in the path and no outgoing edge in the path.

- ○ $\texttt{ExactlyOne}(\{X_{gv} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(g)} \neg X_{vg}$
- ○ $\texttt{ExactlyOne}(\{X_{gv} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(g)} X_{vg}$
- ● $\texttt{ExactlyOne}(\{X_{vg} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(g)} \neg X_{gv}$
- ○ $\texttt{ExactlyOne}(\{X_{vg} \mid v \in N(s)\}) \wedge \bigwedge_{v \in N(g)} X_{gv}$

Clarification during exam: All options should use $N(g)$ instead of $N(s)$.

The idea is similar to the previous question. X_{vg} represent incoming edges from other nodes v to g, and we want exactly one of these variables to be true.

X_{gv} represents outgoing edges from g to other nodes v. We want all these to be false, so we want $\neg X_{gv}$ for all v.

Note: There was a slight bug in this question. At the top of the question, we wrote that $N(v)$ is the set of nodes w where an edge exists from v to w (i.e. direction of edges matters when considering neighbors), but this would make the sets $N(s)$ and $N(g)$ only give us outgoing edges, not incoming edges. However, among the four answer choices provided, we thought that the selected answers were unambiguously the best answers, so we did not clarify this during the exam.

(c) [2 pts] If node v has an incoming edge in the path, then it must have exactly one outgoing edge in the path.

- ○ $\bigwedge_{w \in N(v)} X_{wv} \iff \texttt{ExactlyOne}(\{X_{vw} \mid w \in N(v)\})$
- ○ $\bigwedge_{w \in N(v)} X_{wv} \implies \texttt{ExactlyOne}(\{X_{vw} \mid w \in N(v)\})$
- ○ $\bigvee_{w \in N(v)} X_{wv} \iff \texttt{ExactlyOne}(\{X_{vw} \mid w \in N(v)\})$
- ● $\bigvee_{w \in N(v)} X_{wv} \implies \texttt{ExactlyOne}(\{X_{vw} \mid w \in N(v)\})$

$\bigvee_{w \in N(v)} X_{wv}$ encodes whether or not v has an incoming edge. It checks all the variables X_{wv} representing edges from other nodes w to the chosen node v, and returns true if at least one of these variables is true (i.e. there is at least one incoming edge), or false if all variables are valse (i.e. there is no incoming edge).

We want an implication $\implies$ rather than a biconditional $\iff$ because of the if-then structure of the English statement. The statement does not suggest that the converse needs to be true.

(d) [2 pts] Suppose the graph has N nodes and M edges. How many different assignments to the variables exist in this CSP? Your answer should be an expression, possibly in terms of N and M.

2^M

Our CSP has one variable for each edge in the graph. Each variable is binary (is assigned to either true or false). Therefore, we have 2^M possible assignments to the variables in this CSP.

Suppose we set up our CSP on a graph with nodes s, a, b, c, d, e, g. We want to run a hill climbing algorithm to maximize the number of satisfied constraints. Sideways steps (variable changes that don't change the number of satisfied constraints) are allowed. All variables in the CSP initially set to false.

(e) [2 pts] Assuming that each of the following variables exists in the CSP, select all variables that might be set to true in the first step of running hill-climbing.

■ X_{sa} ■ X_{sb} □ X_{cs} □ X_{ab} □ X_{de} ■ X_{eg} ○ None of the above

Recall the hill-climbing algorithm from the local search and CSP lectures: Start with some initial assignment (here, all variables false). At each iteration, change one of the assignments to one of the variables, such that the total number of violated constraints is decreased or unchanged.

Initially, the constraints from part (c) are satisfied, because the implication is vacuously true (the left-hand-side is false). The constraints from part (a) and (b) are violated, because there are no outgoing edges from s, and there are no incoming edges to g.

Adding an outgoing edge from s satisfies the constraint in (a) without violating any constraints in (b) or (c). This corresponds to setting either X_{sa} or X_{sb} (the two outgoing edges from s in the choices) to true.

Adding an incoming edge to g satisfies the constraint in (b) without violating any constraints in (a) or (c). This corresponds to setting X_{eg} (the only incoming edge to g in the choices) to true.

Adding any other edge (i.e. any edge that isn't outgoing from s or incoming to g) won't affect constraints (a) or (b), but will cause a constraint in (c) to be violated, because it introduces one incoming edge without also introducing an outgoing edge. Therefore, the hill climbing algorithm won't set X_{cs}, X_{ab}, or X_{de} to true on the first iteration.

(f) [1 pt] Will our hill-climbing algorithm always find a solution to the CSP if one exists?

○ Yes, the algorithm will incrementally add edges to the path until the path connects s and g.

○ Yes, the algorithm will eventually search over all combinations of assignments to the variables until a satisfying one is found.

○ No, the search will sometimes find a solution to the CSP that is not a valid path.

● No, the search will sometimes fail to satisfy all of the constraints in the CSP.

The only way for the hill climb to change variable assignments without decreasing the number of satisfied constraints is to add an outgoing edge to a node with an incoming edge that doesn't have one yet or add an outgoing or incoming edge to s or g if they don't have them yet. Therefore, the search will proceed by extending a partial path from s (or backwards from g) until either (1) the partial path forms a complete path from s to g or (2) the partial path reaches a dead end in the graph at which no more edges can be added without repeating a node in the graph. In the second case, the search process will fail and the CSP will still have violated constraints.

Q3. [9 pts] A Multiplayer MDP

Alice and Bob are playing a game on a 2-by-2 grid, shown below. To start the game, a puck is placed on square A.

At each time step, the available actions are:

- Up, Left, Down, Right: These actions move the puck deterministically. Actions that move the puck off the grid are disallowed. These actions give both players a reward of 0.
- Exit: This action ends the game. Note that the Exit action can be taken no matter where the puck is. This action gives Alice and Bob rewards depending on the puck's final location, as shown below.

Each player tries to maximize their own reward, and does not care about what rewards the other player gets.

X	Y
W	Z

Figure 1: The grid that the puck moves on.

Puck location when "exit" is taken	Alice's reward	Bob's reward
W	−1	+3
X	0	0
Y	+1	+2
Z	−1	+2

Figure 2: Exit rewards for Alice and Bob.

(a) [2 pts] Suppose Bob is the only player taking actions in this game. Bob models this game as an MDP with a discount factor $0 < \gamma \leq 1$.

For which of the following states does Bob's optimal policy depend on the value of γ? Select all that apply.

☐ W ☐ X ■ Y ■ Z ○ None of the above.

Clarification during exam: To start the game, a puck is placed on square W (not A).

Note that in this question, Bob is the only player, so all we're concerned about is maximizing Bob's utility. Alice's utility does not matter in this question.

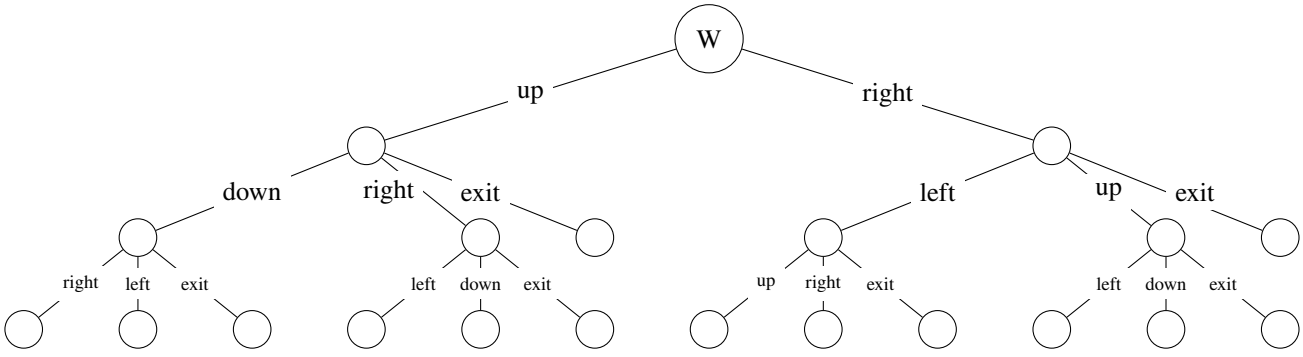
Bob gets highest reward at W, so the optimal action at W is always to exit.

The optimal policy from X is always {down, exit} to move into state W and exit there for a reward of 3. Even though there's a discount factor, we know that a reward of 3γ (one discount factor applied) is always going to be greater than the reward of 0 (if we had exited from X), because $\gamma > 0$.

At Y and Z, if the discount factor is low (i.e. future rewards are heavily discounted), the optimal action is to immediately exit and get a reward of 2. If the discount factor is high enough (i.e. future rewards are not so heavily discounted), then it's better to go to state W and get the discounted reward of 3γ for exiting from state W.

For the rest of the question, Alice and Bob alternate taking actions. Alice goes first.

Alice models this game with the following game tree, and wants to run depth-limited search with limit 3 turns. She uses an evaluation of 0 for any leaf node that is not a terminal state. (Note: Circles do not necessarily represent chance nodes.)



(b) [1 pt] This is a zero-sum game.

○ True ● False

Alice and Bob have different utilities in the tree, partially cooperative and partially competitive, as shown in the reward table.

In order for this game to be zero-sum, Alice's utility would need to be the negative of Bob's utility.

(c) [1 pt] What is Alice's value of the root node of the tree?

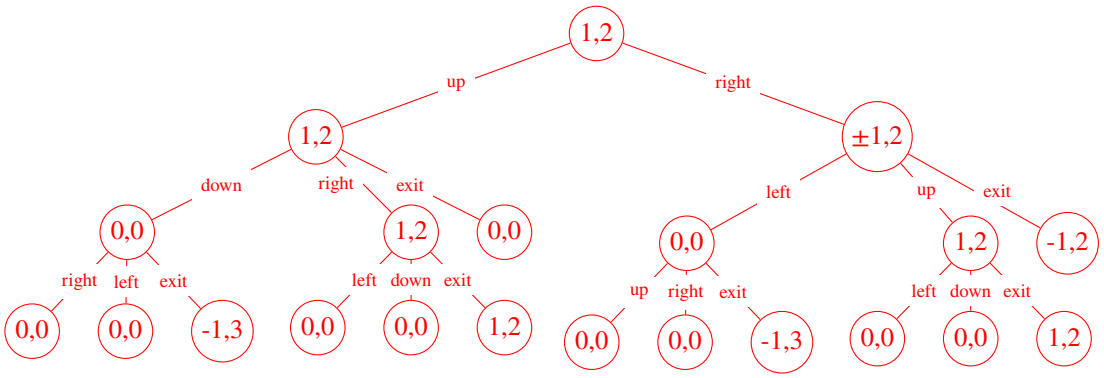
1

(d) [1 pt] What is Bob's value of the root node of the tree?

2

Clarification during exam: The left-most subtree under the game tree should say "right, up, exit" not "right, left, exit".

Filling out the tree with Alice and Bob's utility gives the following tree.



At the terminal nodes, we fill in the utilities associated with the corresponding sequence of actions. For example, going through the leaf nodes of the tree left to right:

The left-most leaf node corresponds to the actions {up, down, right}. This sequence of actions has not led to the game ending, so we cannot evaluate the value (Alice and Bob's reward) at this state. However, we note that Alice uses an evaluation of 0 for any non-terminal leaf node, so the value of this state is $(0, 0)$.

Similarly, the next leaf node corresponds to {up, down, up} (clarification corrected this from {up, down, left}, which is illegal). Again, the game has not ended, so we use the evaluation of 0 to find the value of this state is $(0, 0)$.

The next leaf node corresponds to {up, down, exit}. This ends the game with an exit from W, which gives Alice reward of -1 and Bob reward of $+3$. In the tree, we denote this as $(-1, 3)$, where the left value is Alice's reward and the right value is Bob's reward.

Going through the other leaf nodes, we can write in $(0, 0)$ for any non-terminal leaf nodes and the exit rewards for any leaf nodes where an exit action was taken.

To solve the tree, we need to evaluate which layers correspond to which player. Alice is going first, so the two branches from the root node (up and right) correspond to Alice's action (where she will maximize her own utility). Since Alice and Bob take turns, the next layer then corresponds to Bob's action (where he will maximize his own utility), and the final layer corresponds to Alice's action again.

In layers where it's Bob's turn, we pick the child with the highest right-value (second value in the tuple). In layers where it's Alice's turn, we pick the child with the highest left-value (first value in the tuple).

Note that we have a node labeled $(\pm 1, 2)$. This is because at this node, Bob is indifferent between the $(1, 2)$ and the $(-1, 2)$ nodes, since he gets a reward of 2 either way. Since we didn't specify a tiebreaker mechanism, the value here could be either $(1, 2)$ or $(-1, 2)$. However, the tiebreaker mechanism doesn't matter because at the root node, Alice has a choice between $(1, 2)$ and $(\pm 1, 2)$ and will choose $(1, 2)$. If Alice is unsure of Bob's tiebreaker strategy, she'd choose the guaranteed $(1, 2)$ over the $(\pm 1, 2)$ where she might risk getting -1.

(e) [2 pts] Assuming that Alice and Bob play optimally, what are possible sequences of actions that they might play? Select all that apply.

- [x] up, right, exit
- [ ] up, right, down
- [ ] right, left, right
- [ ] right, up, exit
- [ ] right, exit
- ○ None of the above.

Using the tree above, we see that the value $(1, 2)$ is associated with either the action sequence {up, right, exit}, or {right, up, exit}.

However, as discussed above, Alice does not know Bob's tiebreaker strategy, so the $(1, 2)$ value at the root is associated with the left branch {up, right, exit}. In other words, {right, up, exit} is suboptimal with no further knowledge of Bob's strategy, because we can't be sure if Bob will force the $(1, 2)$ or the $(-1, 2)$ outcome.

Alice instead decides to model the game as an MDP. Assumptions:

- $\gamma = 0.5$
- Alice knows Bob's policy is π.
- $D(s, a)$ represents the new state you move into when you start at state s and take action a.

(f) [2 pts] Fill in the blanks to derive a modified Bellman equation that Alice can use to compute the values of states.

Let $s' = D(s, a)$ and $s'' = D(s', \pi(s'))$.

$$V(s) = \max_a R(s, a, s') + \textbf{(i)}\ \textbf{(ii)} + \textbf{(iii)}\ \textbf{(iv)}$$

(i)	○ 1	● 0.5	○ 0.25
(ii)	○ 0	○ $R(s, \pi(s), s')$	● $R(s', \pi(s'), s'')$
(iii)	○ 1	○ 0.5	● 0.25
(iv)	○ $V(s)$	○ $V(s')$	● $V(s'')$

Recall that the Bellman equation relates the values of states $V(s)$ with the values of other states $V(s')$. In this MDP, Alice and Bob alternate choosing actions, so in order to relate Alice's value at a state to Alice's value function at some other state, we need to iterate two timesteps into the future (to reach the next time it's Alice's turn).

To consider the first time step into the future, we need to consider Alice's immediate reward $R(s, a, s')$, which is already in the answer. After the first time step, the game has transitioned from s into s'.

Then, for the second time step into the future, we need to consider the action that Bob will take, $pi(s')$. This will transition the game from state s' to another state, denoted s''. We also need to consider the reward that Alice will get from this transition (that Bob chose), which is $R(s', \pi(s'), s'')$.

Once we reach s'' two time steps later, it's Alice's turn again, so we can use the recursive definition $V(s'')$ to denote the value of Alice starting at s'' and acting optimally.

Finally, we need to make sure to apply one discount of 0.5 to the reward $R(s', \pi(s'), s'')$ one time step into the future. We need to apply two discounts to the rewards that are 2+ time steps into the future, $V(s'')$.

$V(s) = \max_a(R(s, a, s') + 0.5R(s', \pi(s'), s'') + 0.25V(s''))$

Q4. [13 pts] Slightly Different MDPs

Each subpart of this question is independent.

For all MDPs in this question, you may assume that:

- The maximum reward for any single action is some fixed number $R > 0$, and the minimum reward is 0.
- The discount factor satisfies $0 < \gamma \leq 1$.
- There are a finite number of possible states and actions.

(a) [2 pts] Which statements are always true? Select all that apply.

- ☐ $\sum_{s \in S} T(s, a, s') = 1$.
- ■ $\sum_{s' \in S} T(s, a, s') = 1$.
- ☐ $\sum_{a \in A} T(s, a, s') \leq 1$.
- ☐ For all state-action pairs (s, a), there exists some s', such that $T(s, a, s') = 1$.
- ○ None of the above

Clarification during exam: Q4 - The discount factor is $0 < \gamma < 1$.

(A): False. This adds up the probability of going from all states to a certain state, which doesn't necessarily sum to 1. As an intuitive example (disregarding the constant action a), consider a 3-state MDP with states X, Y, and Z. $P(X \rightarrow Z) + P(Y \rightarrow Z) + P(Z \rightarrow Z) \neq 1$. Maybe it's likely ($< 50\%$) to reach Z from any of the 3 states, which would make this expression add to more than 1.

(B): True. This adds up the probability of going from 1 state to all other states, which sums to 1 (because from one state, once you take an action, you have to land in some state). Using the example above, $P(X \rightarrow X) + P(X \rightarrow Y) + P(X \rightarrow Z) = 1$ because from X, once you take an action, you have to land in X, Y, or Z.

(C): False. Suppose that in the example, if you're in X, any action is guaranteed to land you in Z, no matter what action you take. Assume there are 2 actions, Left and Right. Then $T(X, \text{Left}, Z) + T(X, \text{Right}, Z) = 2$, which is not ≤ 1.

(D): False. $T(s, a, s') = 1$ would say that in state s, taking action a always lands you in s'. However, there is no guarantee that taking some action in the MDP has a guaranteed outcome. For example, consider Gridworld from lecture with the exit action removed: every action is probabilistic, and there is no action that has a guaranteed s' successor state.

(b) [2 pts] Which statements are always true? Select all that apply.

- ■ Every MDP has a unique set of optimal values for each state.
- ☐ Every MDP has a unique optimal policy.
- ☐ If we change the discount factor γ of an MDP, the original optimal policy will remain optimal.
- ■ If we scale the reward function $r(s, a)$ of an MDP by a constant multiplier $\alpha > 0$, the original optimal policy will remain optimal.
- ○ None of the above

(A): True. The optimal values are the solutions to the Bellman equations, and they exist and are finite if $0 < \gamma < 1$ and the state space is finite. In other words, from a given state, the expected discounted sum of rewards for acting optimally is a unique value.

(B): False. An MDP could have multiple optimal policies. For example, consider a state where every action results in the same successor state. Then the optimal policies could assign any action at this state.

(C): False. Consider an MDP with two states, A and B. At A we can either go to B or exit, getting a reward of 1, and at B, we can only exit, getting a reward of 10. If the discount factor is 0.9, the optimal action at A would be to go to B; whereas if the discount factor is 0.01, the optimal action at A is to exit directly.

(D): True. The optimal policy is determined by taking a argmax over values; if we scale all the values up or down by a constant, the relative ordering of values stays the same.

(c) [2 pts] Which statements are true? Select all that apply.

■ Policy iteration is guaranteed to converge to a policy whose values are the same as the values computed by value iteration in the limit.

■ Policy iteration usually converges to exact values faster than value iteration.

☐ Temporal difference (TD) learning is off-policy.

■ An agent is performing Q-learning, using a table representation of Q (with all Q-values initialized to 0). If this agent takes only optimal actions during learning, this agent will eventually learn the optimal policy.

○ None of the above

(A): True. Policy iteration is guaranteed to converge to an optimal policy. Value iteration computes the values of the optimal policy. If we compute the values of the optimal policy (from policy iteration), we'll get the same numbers as if we performed value iteration.

(B): True. Most of the time, value iteration converges towards optimal values in the limit but never reaches the exact values. However, policy iteration eventually finds the optimal policy, and then the policy evaluation step finds exact values as the solution of the linear equations.

(C): False. TD learning involves collecting samples using a particular policy $\pi(s)$ on the MDP, and thus it is on-policy, i.e., it learns values for π, the policy that is generating the samples.

(D): True.

(d) [2 pts] We modify the reward function of an MDP by adding or subtracting at most ϵ from each single-step reward. (Assume $\epsilon > 0$.)

We fix a policy π, and compute $V_\pi(s)$, the values of all states s in the original MDP under policy π.

Then, we compute $V'_\pi(s)$, the values of all states in the modified MDP under the same policy π.

What is the maximum possible difference $\left|V'_\pi(s) - V_\pi(s)\right|$ for any state s?

○ ϵ

○ $\gamma\epsilon$

● $\sum_{n=0}^{\infty} \epsilon(\gamma)^n = \epsilon/(1-\gamma)$

○ ϵR

○ None of the above

Intuitively, because the policies are the same, the future action sequences from any given state is also the same. (Formally, because an action can result in landing in multiple different states, we'd have to say something like, the distribution over futures is the same at a given state.)

Recall that the value of a state is the expected, discounted sum of rewards for acting optimally from that state for the rest of the time. So at each time step that we act, the difference in reward is at most ϵ (discounted appropriately). The sum of discounted differences at each time step is: $1 + \gamma + \gamma^2 + \ldots = \epsilon/(1-\gamma)$.

(e) [3 pts] We modify the reward function of an MDP by adding exactly C to each single-step reward. (Assume $C > 0$.) Let V and V' be the optimal value functions for the original and modified MDPs. Select all true statements.

- [x] For all states s, $V'(s) - V(s) = \sum_{n=0}^{\infty} C(\gamma)^n = C/(1-\gamma)$.
- [ ] For all states s, $V'(s) - V(s) = C$.
- [ ] For some MDPs, the difference $V'(s) - V(s)$ may vary depending on s.
- ( ) None of the above

When we add a constant, the optimal policy does not change, so we have the same distribution over histories, and adding c at each step gives a discounted sum of Cs, i.e., $C/(1-\gamma)$. This will be the same for all states.

(f) [2 pts] We notice that an MDP's state transition probability $T(s, a, s')$ does **not** depend on the action a.

Can we derive an optimal policy for this MDP without computing exact or approximate values (or Q-values) for each state?

- (●) Yes, because the optimal policy for this MDP can be derived directly from the reward function.
- ( ) Yes, because policy iteration gives an optimal policy and does not require computing values (or Q-values).
- ( ) No, because we need a set of optimal values (or Q-values) to do policy extraction.
- ( ) No, because there is no optimal policy for such an MDP.
- ( ) None of the above

We know the optimal policy is given by

$$\begin{aligned}\pi(s) &= \arg\max_a Q^*(s,a) \\ &= \arg\max_a \left[r(s,a) + \sum_{s'} T(s,a,s') V^*(s') \right].\end{aligned}$$

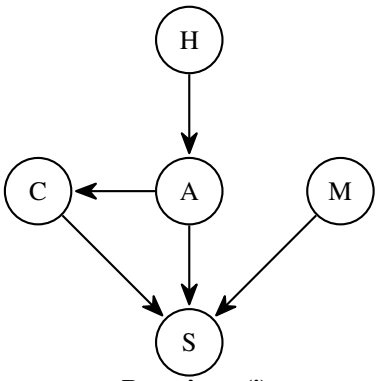
Since $T(s, a, s')$ does not depend on a, the enitre summation term is irrelevant to the argmax (it'll be a constant number, even as we vary a). Therefore, the optimal policy equation simplifies to $\pi(s) = \arg\max_a r(s, a)$, which only relies on the reward function.

Intuitively: If the transition probability doesn't depend on the action, then no matter what action we take, we end up with the same probability of landing in every state. Therefore, we should just take the action that results in the highest immediate reward.

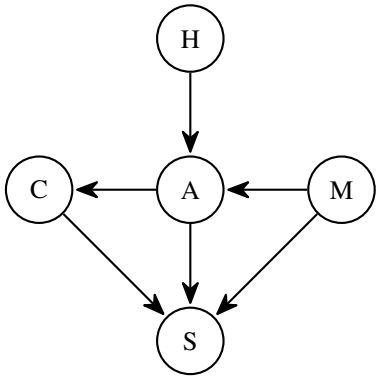
Q5. [8 pts] Bayes' Nets

The head of Pac-school is selecting the speaker (S) for the commencement ceremony, which depends on the student's major (M), academic performance (A), and confidence (C). The student's hard work (H) affects their academic performance (A), and academic performance (A) can affect confidence (C).

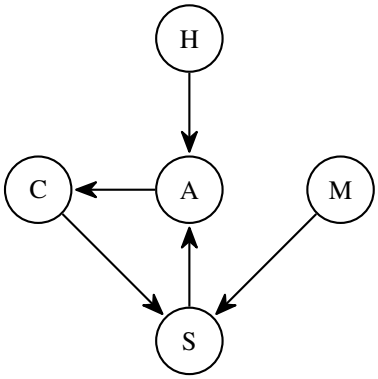
(a) [1 pt] Select all of the well-formed Bayes' nets that can represent a scenario *consistent with* the assertions given above (even if the Bayes' net is not the most efficient representation).



Bayes' net (i)



Bayes' net (ii)



Bayes' net (iii)

- [x] Bayes' net (i)
- [x] Bayes' net (ii)
- [ ] Bayes' net (iii)
- [ ] None of the above

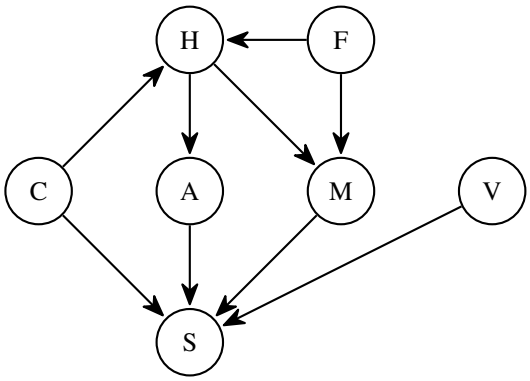
(i): True. The arrows in this diagram exactly reflect the cause-and-effect scenarios in the problem statement.

(ii): True. Adding an extra edge to the Bayes' net only makes the Bayes' net capable of representing more joint distributions; the more complex Bayes' net can still represent all the distributions that the original Bayes' net represented.

As a simple example, consider two independent coin flips with Bayes' net nodes A and B. If you drew an arrow betweeen A and B, the resulting Bayes' net can still represent two independent coin flips, by appropriately setting the CPT values in B, e.g. $P(+b|+a) = P(+b|-a) = P(-b|+a) = P(-b|-a) = 0.5$.

(iii): False. This Bayes' net has a cycle $A \to C \to S \to A$, and Bayes' nets must be acyclic.

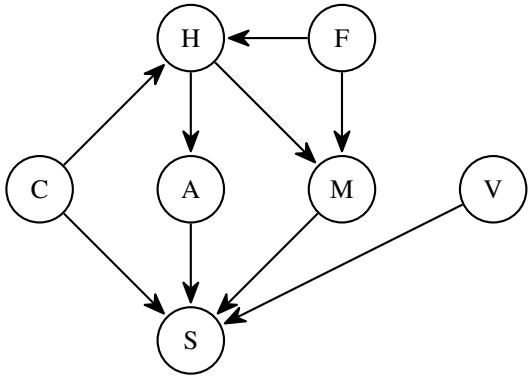
For the rest of the question, consider this Bayes net with added nodes Fearlessness (F) and Voice (V).



(b) [2 pts] Select all the conditional independence expressions that follow from the two standard axioms (1) a variable is independent of its non-descendants given its parents, and (2) a variable is independent of everything else given its Markov blanket.

- [x] $F \perp\!\!\!\perp C$
- [x] $M \perp\!\!\!\perp C \mid H, F$
- [ ] $H \perp\!\!\!\perp S \mid C, A, M$
- [x] $F \perp\!\!\!\perp A \mid H, M$
- ○ None of the above

The Bayes' net, repeated for your convenience:



Assume all the variables have a domain size of 2. Consider the query $P(S|+h,+v,+f)$ in the Bayes net from the previous part.

(c) [1 pt] Suppose the query is answered by variable elimination, using the variable ordering C, A, M. What is the size (number of entries) of the largest factor *generated* during variable elimination? (Ignore the sizes of the initial factors.)

○ 2^1 ● 2^3 ○ 2^5
○ 2^2 ○ 2^4 ○ 2^6

Initial list of factors:

$P(H|C,F), P(F), P(C), P(A|H), P(M|H,F), P(V), P(S|C,A,M,V)$

Join on C gives us $f_1(A,C,F,H,M,S,V)$. Elimating on C gives us $f_1(A,F,H,M,S,V)$. This factor has six unknowns and 2^6 rows. This is actually the largest possible factor we can generate (there were only 7 variables and we always eliminate at least one each time we generate a factor), so the answer must be 2^6 and we're done.

(d) [2 pts] If we use Gibbs sampling to answer this query, what other variables need to be referenced when sampling M?

■ H ■ F ○ None of the above
■ A ■ M
■ S ■ C

Now suppose the CPT for variable A is as follows, and assume there are no zeroes in any other CPT.

A	H	$P(A\|H)$
$+a$	$+h$	1
$-a$	$+h$	0
$+a$	$-h$	0
$-a$	$-h$	1

Hint: Recall that one requirement for Gibbs sampling to converge is that, in the limit of infinitely many samples, every state with non-zero probability is sampled infinitely often, regardless of the initial state.

(e) [1 pt] Will Gibbs sampling converge to the correct answer for the query $P(S \mid +f)$?

○ Yes, because convergence properties of Gibbs sampling depend only on the structure of Bayes net, not on the values in the CPTs.

○ Yes, but it will converge very slowly because of the 0s and 1s in the CPT.

● No, because some states with non-zero probability are unreachable from other such states.

○ No, because Gibbs sampling always fails with deterministic CPTs like this one.

○ None of the above

The deterministic CPT says that A=H; in other words, for (A,H) only states with (0,0) and (1,1) have non-zero probability. But once either of these states is reached, the other can never be reached, because sampling either A or H will always return the same value it already has.

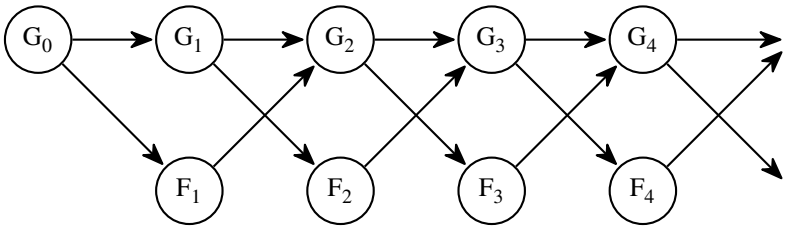
(f) [1 pt] Will Gibbs sampling converge to the correct answer for the query $P(S \mid +h, +v, +f)$?

- ◯ Yes, because convergence properties of Gibbs sampling depend only on the structure of Bayes net, not on the values in the CPTs.
- ◯ Yes, but it will converge very slowly because of the 0s and 1s in the CPT.
- ◯ No, because some states with non-zero probability are unreachable from other such states.
- ◯ No, because Gibbs sampling always fails with deterministic CPTs like this one.
- ● None of the above

In this case H is already fixed at 1, so as soon as A is sampled it will be 1, and it will stay at 1 for ever. This is fine, because (1,1) is the only state with non-zero probability. There is no need to reach (0,0). So it does converge correctly, but neither of the "Yes" reasons is correct. In fact the 0s and 1s *help* convergence here because A immediately gets the right value, which then influences S, so the evidence from H flows very quickly to the query variable.

Q6. [10 pts] Dynamic Bayes' Nets: Book Club

Each week t ($t \geq 0$), the members of a book club decide on the genre of the book G_t to read that week: romance ($+g_t$) or science fiction ($-g_t$). Each week's choice is influenced by the genre chosen in the previous week. In addition, after week t ($t \geq 0$), they collect feedback (F_{t+1}) from the members, which can be overall positive ($+f_{t+1}$) or negative ($-f_{t+1}$), and that influences the choice at week $t+2$. The situation is shown in the Bayes' net below:



(a) [1 pt] Recall that the joint distribution up to time T for the "standard" HMM model with state X_t and evidence E_t is given by $P(X_{0:T}, E_{1:T}) = P(X_0) \prod_{t=1}^{T} P(X_t \mid X_{t-1}) P(E_t \mid X_t)$.

Write an expression for the joint distribution in the model shown above.

$P(F_{1:T}, G_{0:T}) =$ $P(G_0)P(F_1 \mid G_0)P(G_1 \mid G_0) \prod_{t=2}^{T} P(F_t \mid G_{t-1})P(G_t \mid G_{t-1}F_{t-1})$

This is just the ordinary expression for the joint distribution of a Bayes net. Recall that in a Bayes net, multiplying all of the conditional probability tables together (one per node) results in the joint distribution. The tricky part to writing this expression is getting the first couple of steps right ($t=0$ and $t=1$).

(b) [2 pts] Which of the following Markov assumptions are implied by the network structure, assuming $t \geq 2$?

- ☐ $P(G_t \mid G_{0:t-1}, F_{1:t-1}) = P(G_t \mid F_{t-1})$
- ☐ $P(G_t \mid G_{0:t-1}, F_{1:t-1}) = P(G_t \mid G_{t-1})$
- ☐ $P(G_t, F_t \mid G_{0:t-1}, F_{1:t-1}) = P(G_t, F_t \mid G_{t-1})$
- ■ $P(G_t, F_t \mid G_{0:t-1}, F_{1:t-1}) = P(G_t, F_t \mid G_{t-1}, F_{t-1})$
- ☐ $P(F_t \mid G_{0:t-1}, F_{1:t-1}) = P(F_t \mid F_{t-1})$
- ■ $P(F_t \mid G_{0:t-1}, F_{1:t-1}) = P(F_t \mid G_{t-1}, F_{t-2})$
- ○ None of the above

These can all be worked out precisely using independence of non-descendants given parents. For the last one, note that the F_{t-2} is superfluous, but the equation is still correct.

The conditional probability tables for G_t and F_t are shown below, where a, b, c, d, p, q are constants between 0 and 1.

F_t	G_{t-1}	$P(F_t \mid G_{t-1})$
+	+	p
−	+	$1-p$
+	−	q
−	−	$1-q$

G_t	F_{t-1}	G_{t-1}	$P(G_t \mid F_{t-1}, G_{t-1})$
+	+	+	a
−	+	+	$1-a$
+	+	−	b
−	+	−	$1-b$
+	−	+	c
−	−	+	$1-c$
+	−	−	d
−	−	−	$1-d$

Let's consider this model as a Markov chain with state variables (F_t, G_t).

(c) [2 pts] Write out the transition probabilities below.

Your answer should be an expression, possibly in terms of a, b, c, d, p, and q.

$x_1 = P(+f_t, +g_t \mid +f_{t-1}, +g_{t-1}) =$ ap

$x_2 = P(+f_t, +g_t \mid +f_{t-1}, -g_{t-1}) =$ bq

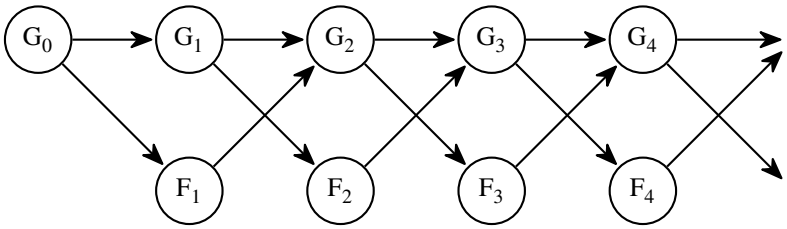
$x_3 = P(+f_t, +g_t \mid -f_{t-1}, +g_{t-1}) =$ cp

$x_4 = P(+f_t, +g_t \mid -f_{t-1}, -g_{t-1}) =$ dq

These entries follow from the fact that $P(F_t, G_t \mid F_{t-1}, G_{t-1}) = P(F_t \mid G_{t-1})P(G_t \mid F_{t-1}, G_{t-1})$. You can think of this expression as joining together two factors (CPTs) of the Bayes' net.

Intuitively, you can also derive these expressions by considering the transition dynamics of this Markov chain. For example, consider deriving x_1. At time $t-1$, we have state $+f_{t-1}, +g_{t-1}$, and we want to know how likely it is for the state at time t to be $+f_t, +g_t$. In order for this transition to happen, $+f_{t-1}$ must transition to $+f_t$. The probability that this happens is governed by $P(F_t|G_{t-1})$, and we look up the value in the table $P(+f_t| + g_{t-1})$ (since we know $+g_{t-1}$ at time $t-1$) to find p. Then, $+g_{t-1}$ must transition to $+g_t$. This transition is governed by the $P(G_t|F_{t-1}, G_{t-1})$, and substituting in the desired values $P(+g_t| + f_{t-1}, +g_{t-1})$ lets us look up the value a in the table. Since both of these transitions must happen, the probability of the overall transition is ap. The same approach can be used to derive the other three expressions.

The Bayes' net, repeated for your convenience:



(d) [2 pts] As t goes to infinity, the stationary distribution of this Markov chain is shown below, where y_1, y_2, y_3, y_4 are constants between 0 and 1.

$$y_1 = P(+f_\infty, +g_\infty)$$
$$y_2 = P(+f_\infty, -g_\infty)$$
$$y_3 = P(-f_\infty, +g_\infty)$$
$$y_4 = P(-f_\infty, -g_\infty)$$

Write an equation that must be true if this is a stationary distribution.

Your answer should be an equation, possibly in terms of x_1, x_2, x_3, x_4 (from the previous part), y_1, y_2, y_3, y_4.

$$x_1 y_1 + x_2 y_2 + x_3 y_3 + x_4 y_4 = y_1$$

Recall that in the equilibrium distribution, the probability of being in a state no longer depends on the time step. In other words, if the probabilities of being in the four states at time ∞ are y_1, y_2, y_3, y_4, then at the next time step $\infty + 1$, after the transition dynamics are applied once, the probabilities of being in the four states are still y_1, y_2, y_3, y_4.

From the previous subpart, we have the probabilities of transitioning into state $(+f, +g)$ from each of the four possible states. Therefore, we can write an equilibrium equation about the probability of being in $(+f, +g)$, which is y_1.

$x_1 y_1$ = probability of starting in $(+f, +g)$, then transitioning to $(+f, +g)$. Note that x_1 is the probability of the transition and y_1 is the probability of starting in the state.

$x_2 y_2$ = probability of starting in $(+f, -g)$ and transitioning to $(+f, +g)$.

$x_3 y_3$ = probability of starting in $(-f, +g)$ and transitioning to $(+f, +g)$.

$x_4 y_4$ = probability of starting in $(-f, -g)$ and transitioning to $(+f, +g)$.

Since these are the only four ways to transition into $(+f, +g)$, if we sum them up, we should get the probability that we're in $(+f, +g)$ on the next time step, and because this is an equilibrium distribution, this should be equal to the probability that we're in $(+f, +g)$ right now.

(e) [3 pts] Select all true statements.

■ There are values of a, b, c, d such that the feedback model defined by p, q is irrelevant to the stationary distribution of the genre G.

☐ Even if $P(F_t, G_t)$ reaches a unique stationary distribution, it is possible that the marginal probabilities $P(F_t)$ and $P(G_t)$ do not reach unique stationary distributions.

☐ For any values of a, b, c, d, p, q, there is at least one stationary distribution.

○ None of the above

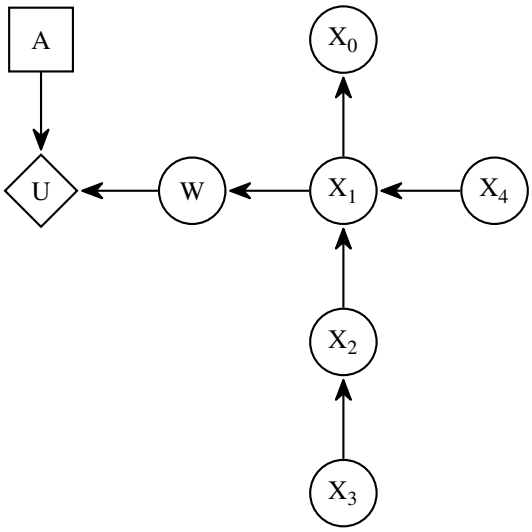
True. If $a = c = 1$ and $b = d = 0$, then G_t is simply copied from G_{t-1} regardless of F. In that case the stationary distribution is the same as $P(G_0)$, and p and q are irrelevant for computing the stationary distribution.

False. $P(F_t)$ and $P(G_t)$ can be computed directly from $P(F_t, G_t)$, so if the latter is constant then so are the former.

False. If $a = c = 0$ and $b = d = 1$, then G_t is always the opposite of G_{t-1}, i.e., a deterministic cycle, and there is no stationary distribution.

Q7. [7 pts] Inequalities of VPI

Consider the decision network shown below.



Choose the single equality or inequality symbol that results in the strongest assertion that is necessarily true in this network. (For example, if $a = b$ and $a \geq b$ are both necessarily true, choose $a = b$ because it is a strictly stronger assertion than $a \geq b$.)

(a) [1 pt] $\text{VPI}(X_1, X_2, X_3)$ _____ $\text{VPI}(X_1)$

● $=$ ○ $>$ ○ $\geq$ ○ $<$ ○ $\leq$ ○ Not enough information

Note that U is independent of X_2, X_3 given X_1: $(X_2, X_3) \perp\!\!\!\perp U | X_1$. This implies that $\text{VPI}(X_2, X_3 | X_1) = 0$ Therefore, $\text{VPI}(X_1, X_2, X_3) = \text{VPI}(X_1) + \text{VPI}(X_2, X_3 | X_1) = \text{VPI}(X_1)$

(b) [1 pt] $\text{VPI}(X_1)$ _____ $\text{VPI}(W)$

○ $=$ ○ $>$ ○ $\geq$ ○ $<$ ● $\leq$ ○ Not enough information

Note that X_2, X_3 are independent with U when conditioned on X_1, $(X_2, X_3) \perp\!\!\!\perp U | X_1$. This implies that $\text{VPI}(X_2, X_3 | X_1) = 0$ Therefore, $\text{VPI}(X_1, X_2, X_3) = \text{VPI}(X_1) + \text{VPI}(X_2, X_3 | X_1) = \text{VPI}(X_1)$

(c) [1 pt] $\text{VPI}(X_1, X_2)$ _____ $\text{VPI}(X_2) + \text{VPI}(X_1)$

○ $=$ ○ $>$ ○ $\geq$ ○ $<$ ● $\leq$ ○ Not enough information

VPI is always non-negative. Furthermore, we know from the first part that $\text{VPI}(X_2 | X_1) = 0$. Therefore, $\text{VPI}(X_1, X_2) = \text{VPI}(X_1) + \text{VPI}(X_2 | X_1) = \text{VPI}(X_1) \leq \text{VPI}(X_1) + \text{VPI}(X_2)$

(d) [1 pt] $\text{VPI}(X_3, X_4)$ _____ $\text{VPI}(X_3) + \text{VPI}(X_4)$

○ $=$ ○ $>$ ○ $\geq$ ○ $<$ ○ $\leq$ ● Not enough information

Although X_3, X_4 are independent, their impact on VPI can be (1) synergistic, i.e., joint VPI is higher than the sum, for example if neither variable individually can change the default decision, but together they can, or (2) antagonistic, e.g., if each individually *fixes* the decision, e.g., by deterministically fixing the value of X_1, such that adding the other variable cannot change it.

(e) [1 pt] $\text{VPI}(X_1, X_2 | X_3)$ _____ $\text{VPI}(X_2 | X_3)$

○ $=$ ○ $>$ ● $\geq$ ○ $<$ ○ $\leq$ ○ Not enough information

$\text{VPI}(X_1, X_2 | X_3) = \text{VPI}(X_2 | X_3) + \text{VPI}(X_1 | X_2, X_3) \geq \text{VPI}(X_2 | X_3)$

(f) [2 pts] Under which of the following circumstances would VPI(**X**) be 0 for *every* set of random variables **X**? Select all that apply.

- ☐ U is a deterministic function of A and W.
- ■ U is independent of A given W.
- ■ U is independent of W given A.
- ■ The best action choice, given W, is the same for each value of W.
- ☐ The expected utility of the best action choice, given W, is the same for each value of W.
- ○ None of the above

(A): False. U is always a deterministic function of A and W. For every configuration of A and W, we have to list the corresponding utility. The weather example from lecture can be a counterexample here: given your take/leave umbrella choice A and the weather W, your utility is fully deterministic. However, the value of knowing the weather is nonzero.

(B): True. Intuitively, this independence statement says that if you know W, then the action you take has no effect on your utility. This means that knowing W is useless; if you are indifferent to action after learning W, then you may as well never change your action after learning W. As we saw in lecture with the soup example (there are two soups but you wouldn't order either one), if your action doesn't change upon learning a variable outcome, then the VPI of that variable is 0.

(C): True. Intuitively, this statement says that once you take an action, the outcome of W has no effect on the utility. Since the outcome of W has no effect on utility, the value of knowing W is 0.

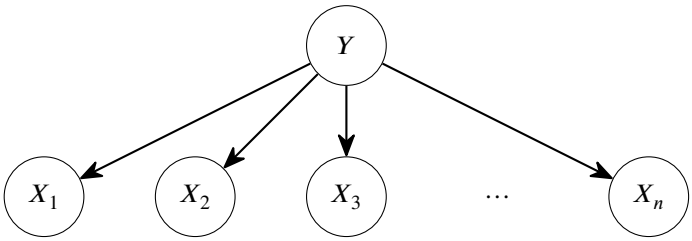
(D): If the best action choice is the same regardless of W, then knowing W is not going to change your choice of action. As mentioned above, if knowing the value of W does not change your action, then the value of knowing W is 0.

(E): Even if the expected utility of the best action choice is the same for each value of W, that does not necessarily mean that the action that leads to that best expected utility is the same. Since seeing different values of W could lead you to take different actions (i.e. the action that gets you the best utility for the current value of W), the VPI of W is nonzero.

As a concrete example: in our weather example from lecture, suppose that the utility of {sun, leave} is 100 and the utility of {rain, take} is 100. The expected utility of the best action choice given W is the same for either W = rain or W = sun. However, the value of W influences whether you choose to take or leave the umbrella, so the VPI of W is nonzero.

Q8. [13 pts] Inverted Naive Bayes

Consider a standard naive Bayes model with $n > 10$ Boolean features $X_1, \ldots, X_n$ and a Boolean class variable Y. In this question, Boolean variables have values 0 or 1.



(a) [1 pt] How many parameters do we need to learn in this model (not counting parameters that can be derived by the sum-to-1 rule)?

Example of sum-to-1 rule: Given $P(Y = 0)$, we can compute $P(Y = 1)$ since we know that these two values sum to 1. Therefore, $P(Y = 0)$ and $P(Y = 1)$ only count as one parameter we need to learn.

Your answer should be an expression, possibly in terms of n.

$2n + 1$

Under the Y node, there is a conditional probability table $P(Y)$ with two rows: $P(Y = 0)$ and $P(Y = 1)$. We must learn one of these as parameters; as soon as we learn one, the other one can be derived through the sum-to-one rule.

Under each X_i node, there is a conditional probability table $P(X_i|Y)$ with four rows. $P(X_i = 0|Y = 0)$ and $P(X_i = 1|Y = 0)$ sum to 1, so we have to learn one parameter for these two values. Also, $P(X_i = 0|Y = 1)$ and $P(X_i = 1|Y = 0)$ sum to 1, so we have another parameter to learn for these two values. In total, there are 2 parameters to learn for each X_i node.

There is 1 parameter to learn from the one Y node. There are 2 parameters to learn for each of the X_i nodes, and there are n of those, for a total of $2n$ parameters from the X_i nodes. In total, that's $2n + 1$ parameters.

(b) [2 pts] We observe one training example with features $x_1, \ldots, x_n$ and class y. Fill in the blanks to derive the likelihood of this training example.

$$L = P(x_1, \ldots, x_n, y) = P(Y = 1)^{\textbf{(i)}}\ P(Y = 0)^{\textbf{(ii)}} \prod_{i=1}^{n} P(X_i = 1|y)^{\textbf{(iii)}}\ P(X_i = 0|y)^{\textbf{(iv)}}$$

(i)	○ x	○ $1 - x$	● y	○ $1 - y$
(ii)	○ x	○ $1 - x$	○ y	● $1 - y$
(iii)	● x	○ $1 - x$	○ y	○ $1 - y$
(iv)	○ x	● $1 - x$	○ y	○ $1 - y$

Clarification during exam: The x in the answer choices should say x_i.

The probability requested here $P(x_1, \ldots, x_n, y)$ is an entry in the joint distribution. To obtain an entry in the joint distribution, we just need to multiply the corresponding entry from each of the conditional probability tables together.

First, consider the CPT under node Y, which is $P(Y)$. If $y = 0$, the CPT entry we need is $P(Y = 0)$. If $y = 1$, the CPT entry we need is $P(Y = 1)$. To write this if/else condition into our equation, we can write $P(Y = 1)^y P(Y = 0)^{1-y}$.

Note that if $y = 0$, then the first term is equal to 1 (raised to 0th power), and the second term is raised to the $1 - 0 = 1$st power, for $P(Y = 0)$, as desired. Also, if $y = 1$, then the first term is raised to the 1st power, and the second term is equal to 1 (raised to the 0th power), for $P(Y = 1)$, as desired.

Next, consider the CPTs under each X_i node. Again, if $x_i = 0$, we need the CPT entry $P(X_i = 0|y)$, and if $x_i = 1$, we need the CPT entry $P(X_i = 1|y)$. Using the same trick as we did for $P(y)$, we can write $P(X_i = 1|y)^{x_i} P(X_i = 0|y)^{1-x_i}$.

For each of the n CPTs corresponding to X_i nodes, we need to multiply one entry per CPT, which is the summation operator in the existing expression.

(c) [1 pt] Suppose that this training example was missing a feature value, so we only observed $x_2, \ldots, x_n$ and class y.

How is the likelihood with a missing feature $L' = P(x_2, \ldots, x_n, y)$, related to the original likelihood L?

- ○ L' is equal to L, but with any terms involving $P(X_1|y)$ dropped.
- ○ L' is equal to L, with an extra term related to the prior probabilities $P(X_1)$, and any terms involving $P(X_1|y)$ dropped.
- ● The correct expression for L' cannot be determined from L.

$L' = P(x_2, \ldots, x_n, y) = \sum_{x_1} P(x_1, x_2, \ldots, x_n, y)$.

In words: L' is an entry in the marginal distribution where we summed x_1 out of the joint distribution. In other words, we took the joint distribution, collected all rows with values $x_2, \ldots, x_n, y$ (there will be one row of this per value of x_1, for two rows in total), and summed up all their values.

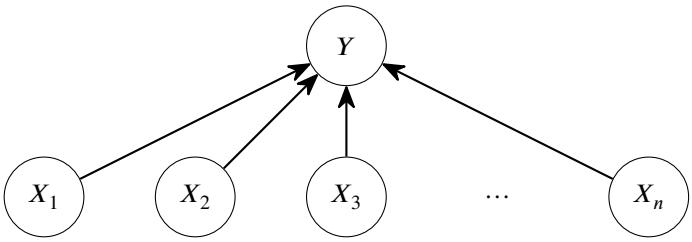
L, with a fixed value of x_1, corresponds to just one of the many rows that we summed together to get L'. With just L, we cannot deduce what the other values to sum together are, so we cannot derive L' from L.

Another way to think about this is to actually write out the summation over x_1, which is possible here since there's only two possible values of x_1:

$L' = P(X_1 = 0, x_2, \ldots, x_n, y) + P(X_1 = 1, x_2, \ldots, x_n, y)$

L is going to correspond to one of these two values (depending on if x_1 is 0 or 1). Given just L, we have no way to work out what the other value in the summation is to derive L'.

Suppose we "invert" the naive Bayes model so that the arrows point from the feature variables to the class variable:



(d) [1 pt] How many parameters do we need to learn in this model (not counting parameters that can be derived by the sum-to-1 rule)?

Your answer should be an expression, possibly in terms of n.

$n + 2^n$

Following simliar logic as part (a) here:

Under each X_i node, we have $P(X_i)$, a CPT with two rows. The two rows have to sum to 1, so each X_i node has one parameter we need to learn. There are n of these X_i nodes, so that's n parameters in total from all the X_i nodes.

$P(Y|X_1, X_2, \ldots, X_n)$ is a CPT with 2^{n+1} rows. For a fixed set of $x_1, \ldots, x_n$, the two rows $P(Y = 0|x_1, \ldots, x_n)$ and $P(Y = 1|x_1, \ldots, x_n)$ must sum to 1, so we only have to learn half of the rows in the table as parameters, and the other half can be derived from the sum-to-1 rule. In total, that's 2^n parameters (corresponding to half of the 2^{n+1} rows).

There are n parameters from the X_i nodes and 2^n parameters from the Y node, for a total of $n + 2^n$ parameters.

For the rest of this question, Boolean variables are represented as positive or negative (e.g. $+y$ and $-y$).

Suppose you have all the parameters from the original naive Bayes model. Using any relevant parameters from the original model, derive the following parameters in the inverted naive Bayes' model, so that the two models represent the same joint distribution.

Your answers should be expressions, possibly in terms of $P(+y)$, $P(-y)$, $P(+x_i|+y)$, $P(-x_i|+y)$, $P(+x_i|-y)$, and $P(-x_i|-y)$, for $1 \leq i \leq n$. Hint: You can also use summations $\sum$ and products $\prod$.

(e) [2 pts] $P(+x_i) =$ $\sum_y P(+x_i|y)P(y)$

One way to solve this is to look at the CPT values in the original Bayes' model (also listed above the question). From this list, note that the closest value to $P(x_i)$ is the CPT under the x_i node, which is $P(x_i|Y)$. Then, to get a marginal distribution over x_i, you can sum over the variable you don't care about (y).

$P(+x_i) = \sum_y P(+x_i|y)P(y)$

Alternatively, if you wrote out the summation over y explicitly, you get:

$P(+x_i) = P(+x_i| + y)P(+y) + P(+x_i| - y)P(-y)$

(f) [2 pts] $P(+y| + x_1, +x_2, \ldots, +x_n) \propto$ $P(+y) \prod_{i=1}^{n} P(+x_i| + y)$

One solution is to use Bayes' rule, since we need an expression with y on the left and x_i on the right of the conditioning bar, but our the listed expressions all have x_i on the left and y on the right.

$P(+y| + x_1, \ldots, +x_n) = \frac{P(+y)P(+x_1,\ldots,+x_n|+y)}{P(+x_1,\ldots,x_n)}$

Now, note that the denominator is a constant even if we vary y, and the question only asks for a proportional value ($\propto$, not $=$), so we can safely ignore the denominator. (Recall the normalization trick we showed in the first probability lecture for why this works.)

Looking at the numerator, $P(+x_1, \ldots, +x_n | + y)$ can be decomposed into a product of individual terms $P(+x_i | + y)$, for the same reason that this decomposition works in the standard Bayes' net. This leaves us with:

$P(+y) \prod_{i=1}^{n} P(+x_i | + y)$

Another quick solution is to note that the desired expression $P(+y | +x_1, \ldots, +x_n)$ is exactly the expression for classification in the original model. So all we have to do is write the classification for expression in the original model.

(g) [3 pts] Select all true statements.

- [x] For any set of parameters in the original naive Bayes model, there is some set of parameters in the inverted naive Bayes model that captures the same joint distribution.
- [ ] For any set of parameters in the inverted naive Bayes model, there is some set of parameters in the original naive Bayes model that captures the same joint distribution.
- [x] The inverted naive Bayes model will typically require far more training data than the original naive Bayes model to achieve the same level of test accuracy.
- ○ None of the above

For options (A) and (B), the quickest intuitive (but definitely not rigorous) answer is to note that the inverted Bayes' net has far more parameters and is probably going to have more expressive power than the normal Bayes' net. Therefore, all distributions representable in the simpler original model should still be representable in the more complex inverted model. However, not all distributions representable in the more complex inverted model can be represented in the simpler original model.

A more rigorous proof for (A) is to note that in the previous two subparts, we were able to derive every parameter in the inverted Bayes' net from the values in the original Bayes' net. Therefore, given any values in the original Bayes' net, we can derive values in the inverted Bayes' net that represent the same joint distribution.

(C): True. Intuitively, the inverted model has far more parameters to learn, so we need more training data to learn the parameters well.

More rigorously, in the original Bayes' net, we just have tables $P(Y)$ and $P(X_i|Y)$. To learn the $P(Y)$ table accurately, we just need to count the $+y$ data points $-y$ data points, and a large enough dataset should have a nontrivial number of data points from each class. To learn $P(X_i|Y)$ accurately, we just need lots of data points with each of these four configurations: $(+x, +y), (-x, +y), (+x, -y), (-x, -y)$. Again, with a large enough data set, there should be a nontrivial number of data points with each of these four possible configurations, which would allow us to take counts and estimate parameters well.

However, in the inverted Bayes' net, we have to learn the values in the table $P(Y|X_1, \ldots, X_n)$. Consider just one pair of rows: $P(+y|x_1, \ldots, x_n)$ and $P(-y|x_1, \ldots, x_n)$, for some fixed $x_1, \ldots, x_n$. To learn these two values accurately, we would need lots of data points with exactly $x_1, \ldots, x_n$ so that we can count how many have $+y$ and how many have $-y$. It's going to be pretty unusual to find many data points with exactly this set of feature values; furthermore, even if we found such data points, they would be useless for learning any other parameter in the table.

(h) [1 pt] What learning behavior should we expect to see with this inverted naive Bayes' model?

- ○ High training accuracy, high test accuracy
- ● High training accuracy, low test accuracy
- ○ Low training accuracy, high test accuracy
- ○ Low training accuracy, low test accuracy

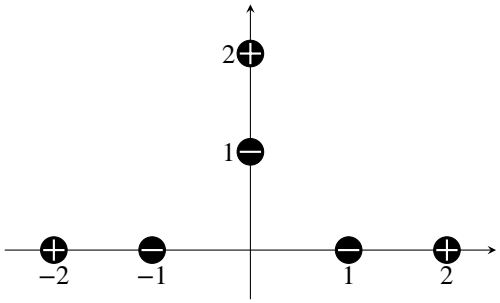
With a sizable number of features, what will probably happen is that no two inputs will have the exact same feature values, or very few inputs will have the same exact feature values. Consider the case where no two inputs have the exact same feature values. For some given training data point $x_1, \ldots, x_n$, $P(Y|x_1, \ldots, x_n)$ derived from counts will equal 1 for this data point's true class, and 0 for the other class. This will give us high training accuracy; when we try to classify that data point $x_1, \ldots, x_n$, the CPT will directly tell us its true class (which it learned from training).

However, this model will probably perform poorly on test data that it hasn't seen before. For some unseen training data point $x_1, \ldots, x_n$, $P(Y|x_1, \ldots, x_n)$ will be 0 for both values of Y, which results in lots of "ties" or undefined classifications.

In other words, this model is overfitting: the CPT under Y is essentially memorizing all the true classes of the training data points, and is not generalizing well to data points it hasn't seen before. Smoothing helps somewhat.

Q9. [8 pts] Higher-Dimensional Perceptrons

Consider a dataset with 6 points on a 2D coordinate grid Each point belongs to one of two classes. Points $[-1, 0], [1, 0], [0, 1]$ belong to the negative class. Points $[-2, 0], [2, 0], [0, 2]$ belong to the positive class.



(a) [1 pt] Suppose we run the perceptron algorithm with the initial weight vector set to $[0, 5]$.

What is the updated weight vector after processing the data point $[0, 1]$?

$[0, 4]$

We have $f = [0, 1]$ and $w = [0, 5]$. First we classify the point by computing the dot product: $f \cdot w = 5$. This classifies f in the positive class, but the true class is negative.

Our classification is wrong, so we need to adjust the weights by subtracting the feature vector: $w - f = [0, 5] - [0, 1] = [0, 4]$.

(b) [1 pt] How many iterations of the perceptron algorithm will run before the algorithm converges? Processing one data point counts as one iteration. If the algorithm never converges, write ∞.

∞

The data points are not linearly separable, so the perceptron algorithm will never terminate.

In the next few subparts, we'll consider *transforming* the data points by applying some modification to each of the data points. Then, we pass these modified data points into the perceptron algorithm.

For example, consider the transformation $[x, y] \rightarrow [x, y, x^2, 1]$. In this transformation, we add two extra dimensions: one whose value is always the square of the first coordinate, and one whose value is always the constant 1. For example, the point at $[2, 0]$ is transformed into a point at $[2, 0, 4, 1]$ in 4-dimensional space.

(c) [2 pts] Which of the following data transformations will cause the perceptron algorithm to converge, when run on the transformed data? Select all that apply.

- ☐ $[x, y] \rightarrow [y, x]$
- ☐ $[x, y] \rightarrow [x, y, 1]$
- ■ $[x, y] \rightarrow [x, y, x^2, 1]$
- ☐ $[x, y] \rightarrow [x, y, x^2 + y^2]$
- ● None of the above.

(A): False. Pictorially, this transformation reflects all the data points across the $x = y$ line. If you graph the resulting points, they're still not linearly separable, so the perceptron algorithm still never terminates.

(B): Pictorially, this transformation plots the data points along a flat plane in the 3D grid. If you graph the resulting points, they're still not linearly separable, so the perceptron algorithm still never terminates.

If you don't want to picture points in higher dimensions, another solution is to note that the resulting decision boundary here is $w_1 x + w_2 y + w_3 > 0$. In other words, you added a y-intercept term w_3 to the decision boundary line on the 2D coordinate plane. Adding a y-intercept so that the decision boundary doesn't have to cross the origin still doesn't help us linearly separate the data, though.

(C): True. Write the decision boundary equation $w_1x+w_2y+w_3x^2+w_4>0$, and note that this is some form of parabola (quadratic equation) in the 2D coordinate plane, since we have terms with y, x^2, x, plus some constant.

Intuitively, you could sketch a parabola that crosses coordinate points $[-1.5,0]$, $[0,1.5]$, and $[1.5,0]$ on the coordinate grid, which would separate the points.

If you wanted to find the exact equation of this line (which was not necessary for this question), you could start with $y=x^2$. Then note that the parabola has to point down, so it should be something like $y=-x^2$. Then note that we should shift this parabola upwards so that the negative points are "below" the parabola, to get something like $y=-x^2+1.5$. Rearranging gives the decision boundary $y+x^2-1.5>0$.

You can confirm that this equation works by plugging in all six points and noting that the negative points all have $y+x^2-1.5<0$, and all the positive points have $y+x^2-1.5>0$.

(D): False. The new feature, x^2+y^2, is equal to 1 for all the negative points and 4 for all the positive points. However, the perceptron classifies points based on the sign of the output, so we'd have somehow use the remaining features to map all the 1s to negative numbers, and all the 4s to positive numbers. We don't have a constant (like in the previous options) to help us with this, and trying to add/subtract multiples of x or y proves to not be useful either (playing around with a few possibilities should be enough to convince you that x and y can't help here).

A geometric solution (which is not necessary to solve this problem, but useful if you like thinking geometrically): note that the x^2+y^2 term looks like the equation of a circle, so adding this term introduces circles into our decision boundaries. We can draw circles and classify points inside the circle as one class, and points outside the circle as a different class. However, we lack a constant term, so our decision boundary is always going to be in the form $w_1x+w_2y+w_3(x^2+y^2)>0$. We can see that $[0,0]$ is always going to fall on the decision boundary, so the lack of constant term restricts our decision boundaries to only the circles that pass through the origin. Visually, we can see that a circle passing through the origin will not separate the points.

(d) [2 pts] Suppose we transform $[x,y]$ to $[x,y,x^2+y^2,1]$, and pass the transformed data points into the perceptron.

Write one possible weight vector that the perceptron algorithm may converge to.

[0,0,1,-2]

The new x^2+y^2 coordinate looks very useful for separating the data. Note that for the negative points, the new coordinate is always 1, and for the positive points, the new coordinate is always 4.

However, 1 and 4 are both positive, and the perceptron classifies based on the sign of the output. To fix this, we need to use the constant bias term to add any negative bias $-4<c<-1$ from every classification so that 1 maps to some negative number and 4 maps to some positive number. For example, $c=-2$ would map positive points to $1-2=-1$ and negative points to $4-2=2$.

If you used a different weight k for the x^2+y^2 feature, you would get perceptron activation of k for all the negative points and $4k$ for all the positive points. Then, your constant factor would need to be in the range $-4k<c<-k$ so that k gets mapped to a negative number, and $4k$ gets mapped to a positive number.

A nice geometric solution (which is not necessary to solve this problem): the x^2+y^2 feature, along with the constant bias, lets us draw circular decision boundaries. The restriction to circles passing through the origin, from the previous subpart, no longer applies here because we've introduced a constant bias. Now, we can draw a circle that separates the points. Any circle with center at the origin, and radius between 1 and 2, will perfectly separate the points (all negative points inside, all positive points outside). If we set the radius to be 1.5, we'd get the circle equation $x^2+y^2=1.5^2=2.25$. Rearranging terms a bit, we get the decision boundary $x^2+y^2-2.25>0$. This corresponds to the weight vector $[0,0,1,-2.25]$, which is also a correct solution.

(e) [2 pts] Construct another transformation (not equal to the ones above) that will allow the perceptron algorithm to converge.

Hint: The transformation $[x,y]\rightarrow[x,y,x^2+y^2,1]$ allows the perceptron algorithm to converge.

Fill in the blank: $[x,y]\rightarrow[x,y,\_\_\_,1]$.

x^4+y^4

One simple class of transformations that works here is any that combines the magnitudes of the two coordinates. (Pictorially, this corresponds to the fact that the negative points are closer to the origin, and the positive points are further away from the origin.) Some sample answers include: x^4+y^4, or x^6+y^6, or $|x|+|y|$, etc.

Another simple class of transformations is to add a constant factor to the x^2+y^2 feature that helped from earlier: $2(x^2+y^2)$, or $3(x^2+y^2)$, or $4(x^2+y^2)$, etc. These transformations still work because you could always adjust the third weight value from the $[x, y, x^2+y^2, 1]$ perceptron to cancel out the new coefficient, which would give you back the original decision boundary that worked on the $[x, y, x^2+y^2, 1]$ perceptron. For example, if your transformation is $x^2+y^2 \rightarrow c(x^2+y^2)$ and the original weight vector had w_3, you could adjust the weight vector to w_3/c and end up with the same decision boundary.

Another simple class of transformation is to add a constant value to the x^2+y^2 feature from earlier: x^2+y^2+1, x^2+y^2+2, etc. If your transformation is $x^2+y^2 \rightarrow x^2+y^2+c$, then you've added a constant value cw_3 to every activation value. If you adjust the constant bias weight value from w_4 to $w_4 - cw_3$, then you cancel out the new addition and end up with the same original decision boundary.

Other solutions probably exist here, but these were the simplest three that we could think of.

Exam continues on next page.

Q10. [9 pts] Q-Networks

Consider running Q-learning on the following Pacman problem: the maze is an x-by-x square, and each position can contain a food pellet or no food pellet. There are no ghosts or walls. Pacman's only actions are {up, down, left, right}.

(a) [2 pts] How many Q-values do we need to learn for this problem?

Your answer should be an expression, possibly in terms of x.

$4x^2 \cdot 2^{x^2}$

The table represents $Q(s, a)$, so we need $|S| \times |A|$ entries. $|A| = 4$. Pacman has x^2 possible locations. Every location has two potential statuses (food or empty). Hence the table size is $4x^2 \cdot 2^{x^2}$.

Recall that in Q-learning, we maintain a table of $Q(s, a)$ values, where there's one Q-value for every state-action pair.

There are 4 actions available from any given state. (Note: We got a few clarification questions during the exam about actions that would cause Pacman to leave the maze. Here, we either assumed that these actions were legal but left Pacman in the same position, or, if these actions were illegal, that 4 is a reasonable upper-bound on the number of available actions from any given state.)

The state space should include Pacman's position, and a list of Booleans indicating whether each position has a food pellet or not. There are x^2 possible locations for Pacman. There are x^2 Booleans we have to keep track of, so there are 2^{x^2} possible configurations of food pellets. In total, the problem has $x^2 \cdot 2^{x^2}$ possible states.

For each of the $x^2 \cdot 2^{x^2}$ states, there are 4 possible actions. Therefore, there are $4x^2 \cdot 2^{x^2}$ state-action pairs.

To learn every Q-value, we could run standard Q-learning, but we decide to try a different approach:

Suppose somebody tells us N exact Q-values for N different state-action pairs: the exact Q-value for the state-action pair (s_i, a_i) is q_i, for $1 \leq i \leq N$. (N is less than the total number of state-action pairs.)

We decide to use these exact Q-values to train a neural network, so that we can estimate other Q-values we don't know. To train this neural network, we need to apply gradient descent to minimize the following loss function:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} (f(\theta, s_i, a_i) - q_i)^2$$

θ represents the weights of the neural network. $f(\theta, s_i, a_i)$ represents running the neural network with weights θ on state-action pair (s_i, a_i).

(b) [1 pt] What is the gradient $\frac{\partial L}{\partial \theta}$?

Your answer should be an expression, possibly in terms of N, $\frac{\partial f}{\partial \theta}$, and q_i.

$\frac{\partial L}{\partial \theta} =$ $\frac{2}{N} \sum_{i=1}^{N} \left[\frac{\partial f}{\partial \theta} (f(\theta, s_i, a_i) - q_i) \right]$

Clarification during exam: Your expression could also use $f(\theta, s_i, a_i)$ and q_i.

Take the partial derivative using the chain rule.

The 2 comes from applying the power rule on the square of each term in the summation, and then factoring it out. The $1/N$ constant coefficient comes from the coefficient in the original expression. The summation stays because the derivative of a sum is equal to the sum of the derivatives.

Inside the summation, we're taking the derivative with respect to θ, so by the chain rule, we need to have a factor of $\frac{\partial f}{\partial \theta}$.

(c) [1 pt] After running t iterations of gradient descent, our current weights are θ_t. The learning rate is α.

What are the weights on the next iteration, θ_{t+1}?

Your answer should be an expression, possibly in terms of θ_t, α, and $\frac{\partial L}{\partial \theta}$.

$\theta_{t+1} =$ $\theta_t - \alpha \frac{\partial L}{\partial \theta}$

This is the expression for gradient descent from lecture. We take the current weights θ_t, and move them in the direction of the gradient $\frac{\partial L}{\partial \theta}$. We apply a learning rate of α, and we use subtraction because gradient descent involves moving in the opposite direction of the gradient.

(d) [2 pts] Eventually, gradient descent converges to the weights θ^*.

We use the neural network with weights θ^* to compute Q-values, and extract a policy out of these Q-values:

$$\pi(s) = \arg\max_a f(\theta^*, s, a)$$

Is π the optimal policy for this problem?

○ Yes ○ No ● Not enough information

When we train a neural network on some training data (here, some subset of all the Q-values) and then use the network to classify some unseen test data (here, the unseen Q-values), there is no guarantee that we are going to perfectly predict the test data values. In other words, test accuracy in a neural network is not guaranteed to be perfect.

It's possible (but unlikely) that we get lucky and our neural network perfectly predicts all unseen Q-values. It's also possible that our neural network makes some errors when predicting unseen Q-values. We don't have enough information to know what the test accuracy of the neural network is.

Instead of the Pacman problem, consider a different problem where the action space is continuous. In other words, there are infinitely many actions available from a given state.

(e) [3 pts] Can we still use the strategy from the previous subparts (without any modifications) to obtain a policy π?

○ Yes ● No

Briefly explain why or why not.

When we move to a continuous action space, the main part of our strategy that breaks is the policy extraction step $\pi(s) = \arg\max_a f(\theta^*, s, a)$.

When we had a finite number of actions, this argmax involved trying each value of a and picking the one with the highest $f(\theta^*, s, a)$ value. However, when we have an infinite number of actions available, this argmax becomes more difficult (or even impossible).

The neural network step should still mostly work; we can still pass in some existing training data and learn weights. Then, we can still use the neural network model to predict Q-values of state-action pairs that we've never seen before, even if there are infinitely many actions.