CS 188 Fall 2023 — Introduction to Artificial Intelligence — Final

Solutions last updated: Monday, December 18

- 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."

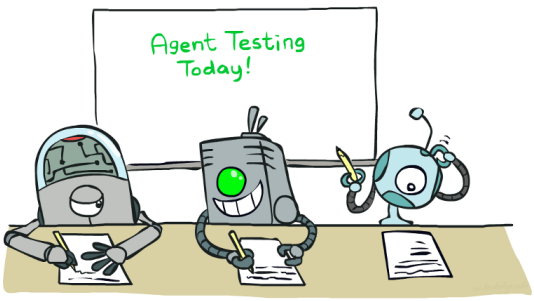
By signing below, I affirm that all work on this exam is my own work, and honestly reflects my own understanding of the course material. I have not referenced any outside materials (other than my cheat sheets), nor collaborated with any other human being on this exam. I understand that if the exam proctor 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: Squirrel Shenanigans	11
Q2.	CSPs and BNs: Bayes for Days	11
Q3.	MDPs and BNs: Coin Blackjack	10
Q4.	RL: Rest and ReLaxation	14
Q5.	Bayes' Nets: Pac Pong Performance	16
Q6.	HMMs and VPI: Markov Menagerie	13
Q7.	Games and ML: BeatBlue	12
Q8.	Potpourri	13
	Total	100

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



Q1. [11 pts] Search: Squirrel Shenanigans

A squirrel is moving around on a grid representing the UC Berkeley campus, picking up and dropping off nuts, in order to store nuts in its nest for the upcoming winter.

At the start of the problem, each square on the $M \times N$ grid contains between 0 and A nuts, inclusive. (A is some constant that is fixed for the entire problem.) There cannot be more than A nuts on any square at any time.

At the start of the problem, the squirrel is carrying 0 nuts. The squirrel can only carry between 0 and A nuts, inclusive, at any time.

The squirrel has the following actions available, and every action costs 1:

- Pick up any number of nuts from the current square (as long as the total number of nuts carried does not exceed A).
- Drop off any number of carried nuts onto the current square (as long as the total number of nuts on the square does not exceed A).
- When the squirrel is carrying C nuts, the squirrel can move 1 to $A - C + 1$ squares in any of the four cardinal directions (up, down, left, right).

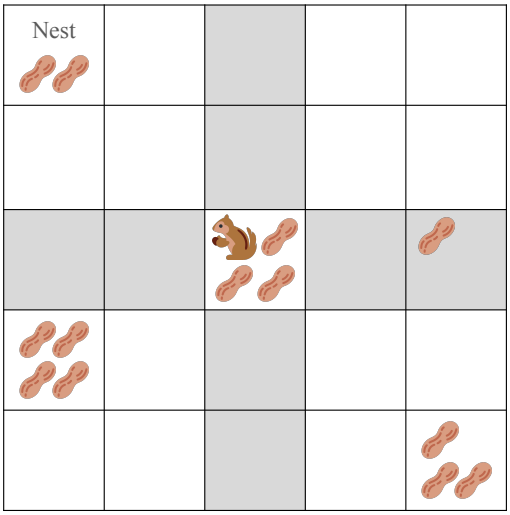
The squirrel's goal is to reach a state where the nest square contains A nuts. The nest square is always fixed to be the top-left corner of the grid.

Here is an example, though your answers should work for any arbitrary problem, not just the example shown.

In this example only, assume $A = 4$, the squirrel is currently carrying $C = 2$ nuts, and there are 3 additional nuts on the square the squirrel is on.

Possible actions from this state (each costing 1):

- Pick up 1 or 2 nuts from the current square. (The squirrel can't pick up 3 nuts, because it can't hold more than 4 nuts.)
- Drop off 1 nut on the current square. (The squirrel can't drop off 2 nuts, because the square can't hold more than 4 nuts.)
- Move to any of the shaded squares ($A - C + 1 = 2$ squares in each of the cardinal directions).



(a) [2 pts] For this subpart only, assume $N > A$ and $M > A$ (the board is very large).

What is the maximum branching factor for this search problem?

○ 1	○ $A+4$	● $5A+4$	○ $MN+A$
○ 4	○ $4A+4$	○ MN	○ AMN

$5A + 4$

The maximum branching factor occurs when the squirrel is holding 0 nuts and is on a square with A nuts.

In this scenario, the squirrel can pick up between 1 and A nuts, giving A actions. The squirrel can also move up to $A + 1$ squares in each of the four directions, giving $4(A + 1)$ actions. In total, we have $A + 4(A + 1) = 5A + 4$ actions.

The square having A nuts creates the maximum branching factor for the squirrel, because each fewer nut reduces the number of actions available by 1 (the squirrel has one fewer choice of number of nuts to pick up).

0 nuts creates the maximum branching factor for the squirrel, because each additional nut the squirrel holds reduces the net number of actions it has available by 3 (the squirrel gains 1 additional action for an extra nut it can drop, but loses 4 actions for range the squirrel loses in each of the cardinal directions).

(b) [1 pt] True or false: The squirrel's location needs to be part of the state space.

- ● True, because it's needed for the successor function.
- ○ True, because it's needed for the goal test.
- ○ False, because it's not needed for the successor function.
- ○ False, because it's not needed for the goal test.

The squirrel location is needed so that the successor function can generate new states with new squirrel locations.

Note that the squirrel location is not needed for the goal test, because the goal test only needs to check the nest square to see if it contains A nuts.

The actions available (each costing 1), repeated for your convenience:

- Pick up any number of nuts from the current square (as long as the total number of nuts carried does not exceed A).
- Drop off any number of carried nuts onto the current square (as long as the total number of nuts on the square does not exceed A).
- When the squirrel is carrying C nuts, the squirrel can move 1 to $A - C + 1$ squares in any of the four cardinal directions (up, down, left, right).

(c) [2 pts] Give a reasonably tight upper-bound on the size of the minimal state space.

Your answer should be in the form $p \cdot q^r$, where p, q, r are expressions that possibly include M, N, A, constants, and arithmetic operators (add, subtract, multiply, divide).

$p =$ MN $q =$ $A + 1$ $r =$ $MN + 1$

$MN(A+1)^{MN+1}$

- MN: Represent the squirrel location. The squirrel can be in MN possible squares.
- $(A+1)^{MN+1}$: Represent the number of nuts on each square, and the number of nuts the squirrel is carrying. For each of the MN squares, store a number between 0 and A, inclusive. For the squirrel, also store one more number between 0 and A, inclusive. In total, each number has $A + 1$ possibilties (between 0 and A), and there are $MN + 1$ such numbers to store (one for each of the MN squares, plus one for the squirrel).
- **Alternate answers:** We also accepted $p = MN(A+1)$ and $r = MN$ since you can get the same solution with these values.

(d) [3 pts] Select all of the admissible heuristics.

- ☐ A, minus the number of nuts on the nest square.
- ☐ 0 if the nest has A nuts. Otherwise, Manhattan distance to the closest non-nest square containing nuts.
- ☐ 0 if the nest has A nuts. Otherwise, Manhattan distance to the furthest non-nest square containing nuts.
- ● None of the above

Option 1: False. Consider this counterexample: The squirrel is on the nest square. The squirrel is carrying A nuts. The nest square contains 0 nuts.

The heuristic would be $A - 0 = A$. However, the actual cost to the goal is 1: the squirrel could drop off all A nuts in a single action, costing 1.

Options 2 and 3: False. Consider this counterexample: The squirrel is carrying 1 nut, and is 1 square south of the nest square. The nest square contains $A - 1$ nuts. There is one other non-nest square with nuts, but it's very far away.

The heuristic would be very large, since the closest/furthest non-nest square (which are the same, it's the only non-nest square) is very far away.

However, the actual cost to the goal is 2: The squirrel could move into the nest square, and then drop off 1 nut.

(e) [3 pts] For this subpart only, suppose we change the search problem: The squirrel is now only able to drop off or pick up a single nut in one time step.

Select all algorithms that can be used to find an optimal solution to this modified problem.

- ☐ Leave the successor function unchanged from the original problem, and run UCS.
- ■ Modify the successor function so that the squirrel can only pick up or drop off one nut at a time (costing 1 each time). Then, run BFS with the modified successor function.
- ■ Modify the costs in the successor function so that picking up or dropping off k nuts costs k, instead of 1. Then, run UCS with the modified successor function.
- ○ None of the above

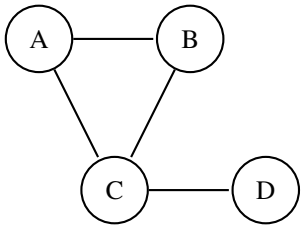
UCS on the original problem won't work, because in the original problem, picking up k nuts still costs 1.

If we modify the successor function so that the squirrel can only pick up or drop off one nut at a time, then all costs are 1, and the successor function models the new costs (i.e. picking up k nuts now costs k actions). Since all costs are 1 again, BFS on this modified successor function will return an optimal solution.

If we modify the costs in the successor function, then we can use UCS to return an optimal solution, since UCS will account for different costs.

Q2. [11 pts] CSPs and BNs: Bayes for Days

Consider the following incomplete Bayes' net, with four random variables, and four edges that do not have directions yet:



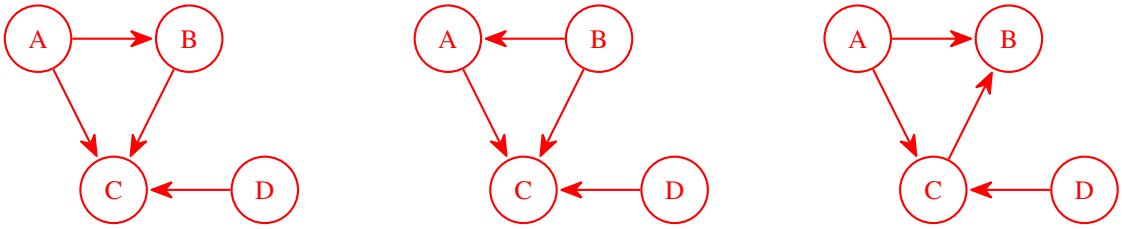
(a) [1 pt] How many undirected paths are there between A and D?

○ 1 ● 2 ○ 3 ○ 4 ○ > 4

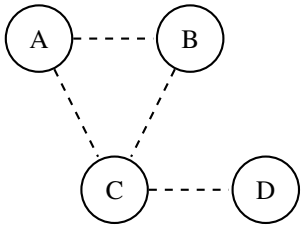
2. ABCD, and ACD.

(b) [2 pts] How many possible ways are there to assign directions to the arrows, such that A and D are independent?

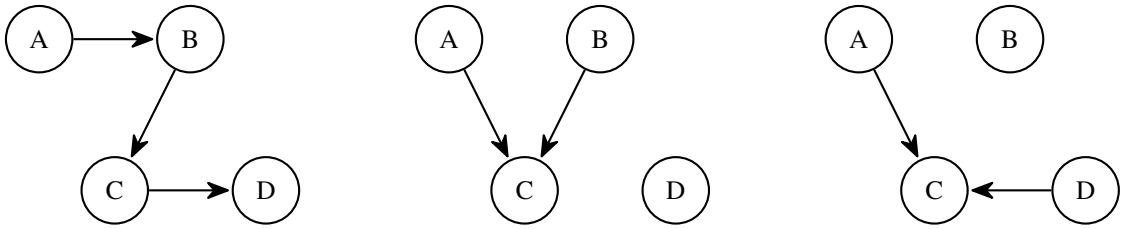
3. Considering both paths ACD and ABCD, we know ACD has to be a common effect triple. Now there are 3 configurations for ABC, excluding the configuration that results in a cycle. Both configurations where B points to C work because BCD is inactive, and if C points to B, ABC is inactive.



For the rest of the question, consider the following incomplete Bayes' net:

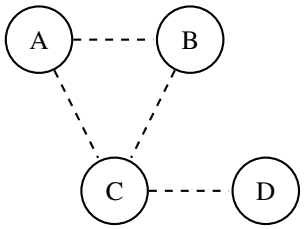


To complete this Bayes' net, each pair of variables connected by a dashed line can be filled in with an edge (in either direction), or no edge. Here are some examples of completed Bayes' nets:



Our goal is to complete this Bayes' net, such that $A \perp\!\!\!\perp B$ and $A \perp\!\!\!\perp D$.

The Bayes' net, repeated for your convenience:



We would like to use a CSP to find a way to complete this Bayes' net.

(c) [1 pt] What are the variables in this CSP?

- ● The 4 dashed edges
- ○ The 4 random variables (A, B, C, D)
- ○ The conditional independence assumptions

(d) [1 pt] How many possible values can each variable take on in this CSP?

○ 1 ○ 2 ● 3 ○ 4 ○ > 4

3. Each pair of variables can be connected with no edge, an edge in one direction, or an edge in the other direction.

For example, the dashed line between A and B could be filled in with no edge, or an edge A → B, or an edge A ← B.

For each of the following statements, select all type(s) of constraints that are needed to represent that statement in the CSP. Each statement is independent.

Note for later: something about minimal constraint, because technically you can use a four-way constraint to write a unary constraint.

(e) [2 pts] The graph must be acyclic.

☐ Unary constraint ■ Three-way constraint ○ None of the above
☐ Binary constraint ☐ Four-way constraint

The only potential cycle in the graph is between A B and C.

This introduces a constraint between AB, BC, and AC, since those three edges could potentially form a cycle.

Note that the CD edge is not relevant to this constraint; no matter how we assign the CD edge, it will not introduce a cycle.

(f) [2 pts] A and B are independent.

■ Unary constraint ☐ Three-way constraint ○ None of the above
■ Binary constraint ☐ Four-way constraint

This introduces a unary constraint, because the AB edge must be unfilled.

This also introduces a binary constraint, because AC and BC cannot be connected in a way that introduces an active path ACB.

(g) [2 pts] A and D are independent.

☐ Unary constraint
■ Binary constraint
■ Three-way constraint
☐ Four-way constraint
○ None of the above

The ACD path needs to be blocked, which introduces a binary constraint on AC and CD.

The ABCD path needs to be blocked, which introduces a three-way constraint on AB, BC, and CD.

Q3. [10 pts] MDPs and BNs: Coin Blackjack

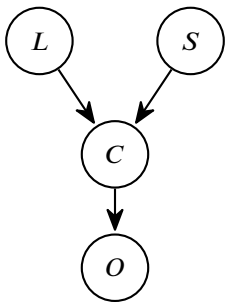
Consider the following game (similar to blackjack):

- You begin the game with a score of 0 points.
- At every time step, you can choose to flip a coin, or end the game with reward equal to your current score.
- When you choose to flip: if the coin comes up heads, then 1 point is added to your score. If the coin comes up tails, then 2 points are added to your score.
- If your score ever reaches 4 or more points, you bust and end the game with 0 reward.

There are two coins: a fair coin that comes up heads exactly half of the time, and a biased coin that comes up heads more than half of the time.

Each time you choose to flip a coin, the dealer will first randomly select one of the two coins, and then flip the chosen coin for you. The probability that the dealer selects the fair or biased coin depends on two things: the coin used on the previous flip, and your current score (before flipping).

The randomness of the coin flip can be modeled by the following Bayes' net:



- L: The coin that was last used on the previous flip (fair or biased).
- S: The current score (0, 1, 2, or 3).
- C: The selected coin (fair or biased).
- O: The outcome of the coin flip (heads or tails).

(a) [1 pt] From the Bayes' net, what is the most efficient way to learn the probability that the biased coin comes up heads? (CPT = conditional probability table)

- ○ Read the value from the CPT in the C node.
- ● Read the value from the CPT in the O node.
- ○ Perform variable elimination with C unobserved.
- ○ Perform variable elimination with C observed as evidence.
- ○ This probability is not in the Bayes' net.

The CPT corresponding to the O node is $P(O|C)$, which would contain an entry telling you the probability of flipping heads, given that the coin is biased.

We choose to model this game as an MDP. For each transition, select the corresponding transition probability.

(b) [2 pts] State: The biased coin was last used. The current score is 1.

Action: Flip.

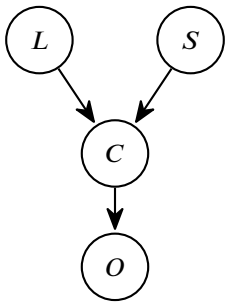
Successor state: The fair coin was last used. The current score is 3.

- ○ 0
- ● $P(C = \text{fair}|L = \text{biased}, S = 1) \cdot P(O = \text{tails}|C = \text{fair})$
- ○ $\sum_c P(C = c|L = \text{biased}, S = 1) \cdot P(O = \text{tails}|C = c)$
- ○ $\sum_l P(C = \text{fair}|L = l, S = 1) \cdot P(O = \text{tails}|C = \text{fair})$
- ○ 1

We're looking for the probability that the fair coin was chosen (since the fair coin was last used after the transition), and tails was flipped (since the score increased by 2). Both of these probabilities, given the current state (biased coin, current score is 1, fair coin is chosen), can be found in the Bayes' net.

$P(C = \text{fair} | L = \text{biased}, S = 1) \cdot P(O = \text{tails} | C = \text{fair})$

The Bayes' net, repeated for your convenience:



- L: The coin that was last used on the previous flip (fair or biased).
- S: The current score (0, 1, 2, or 3).
- C: The selected coin (fair or biased).
- O: The outcome of the coin flip (heads or tails).

(c) [2 pts] State: The fair coin was last used. The current score is 2.

Action: Flip.

Successor state: Bust.

- ○ 0
- ○ $P(C = \text{fair}|L = \text{fair}, S = 2) \cdot P(O = \text{tails}|C = \text{fair})$
- ● $\sum_c P(C = c|L = \text{fair}, S = 2) \cdot P(O = \text{tails}|C = c)$
- ○ $\sum_l P(C = \text{fair}|L = l, S = 2) \cdot P(O = \text{tails}|C = \text{fair})$
- ○ 1

There are two ways for this transition to happen, and they're mutually exclusive, so we can combine them by adding their probabilities.

Either we chose the fair coin and then flipped tails (adding 2 to the score and busting), or we chose the biased coin and flipped tails (adding 2 to the score and busting).

$P(C = \text{fair}|L = \text{fair}, S = 2) \cdot P(O = \text{tails}|C = \text{fair})+$

$P(C = \text{biased}|L = \text{fair}, S = 2) \cdot P(O = \text{tails}|C = \text{biased})$

(d) [2 pts] State: The fair coin was last used. The current score is 3.

Action: Flip.

Successor state: Bust.

- ○ 0
- ○ $P(C = \text{fair}|L = \text{fair}, S = 3) \cdot P(O = \text{tails}|C = \text{fair})$
- ○ $\sum_c P(C = c|L = \text{fair}, S = 3) \cdot P(O = \text{tails}|C = c)$
- ○ $\sum_l P(C = \text{fair}|L = l, S = 3) \cdot P(O = \text{tails}|C = \text{fair})$
- ● 1

1

No matter which coin you choose, and which way the coin flips (heads or tails), flipping on a score of 3 will lead to a bust.

Lastly, we need to deal with the first coin flip at the start of the game, when we don't know what coin was used on the previous flip. We will represent this as being uncertain about the state of the MDP: we could either be in a state where the fair coin was last used, or the biased coin was last used.

(Hint: The next subpart can be solved independently of the rest of the question.)

(e) [3 pts] Consider an MDP where you are uncertain about your current state. The probability that you are in state s is $P(s)$.

The actions available from every state are the same.

Write the Bellman expression that represents the optimal action in this situation.

(i) (ii) (iii) $\sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V(s') \right]$

(i)	○	1	○	$\sum_a$	●	$\arg\max_a$	○	$\arg\min_a$
(ii)	○	1	●	$\sum_s$	○	$\max_s$	○	$\min_s$
(iii)	○	1	●	$P(s)$	○	$T(s,a,s')$	○	$R(s,a,s')$

$\arg\max_a \sum_s P(s) \sum_{s'} T(s,a,s')[R(s,a,s') + V(s')]$

For each action, we need to consider the expected discounted reward of taking that action.

Since we don't know which state we're in, we need to check every state: for each state, compute the expected discounted reward of starting in that state, and taking the action. This is where the $\sum_s P(s)$ term is introduced.

We use an argmax over a in the first term, because we want to find and return the action that results in the highest expected discounted reward.

Q4. [14 pts] RL: Rest and ReLaxation

Consider the grid world MDP below, with unknown transition and reward functions.

A	B	C
D	E	F
G	H	I

The agent observes the following samples in this grid world:

s	a	s'	$R(s,a,s')$
E	East	F	-1
E	East	H	-1
E	South	H	-1
E	South	H	-1
E	South	D	-1

Reminder: In grid world, each non-exit action succeeds with some probability. If an action (e.g. North) fails, the agent moves in one of the cardinally adjacent directions (e.g. East or West) with equal probability, but will not move in the opposite direction (e.g. South).

Let p denote the probability that an action succeeds.

In this question, we will consider 3 strategies for estimating the transition function in this MDP.

Strategy 1: The agent does not know the rules of grid world, and runs model-based learning to directly estimate the transition function.

(a) [1 pt] From the samples provided, what is $\hat{T}$(E, South, H)?

- ○ 0
- ○ 1/5
- ○ 2/5
- ○ 3/5
- ○ 4/5
- ○ 1/3
- ● 2/3
- ○ 1/2
- ○ 1
- ○ Not enough information

There are 3 samples that move South from E, and 2 of them result in successor state H.

(b) [1 pt] From the samples provided, what is $\hat{T}$(E, West, D)?

- ○ 0
- ○ 1/5
- ○ 2/5
- ○ 3/5
- ○ 4/5
- ○ 1/3
- ○ 2/3
- ○ 1/2
- ○ 1
- ● Not enough information

We have no samples that move West from E, so there is not enough information to empirically estimate this transition probability.

Strategy 2: The agent knows the rules of grid world, and runs model-based learning to estimate p. Then, the agent uses the estimated $\hat{p}$ to estimate the transition function.

(c) [1 pt] From the samples provided, what is $\hat{p}$, the estimated probability of an action succeeding?

- ○ 0
- ○ 1/5
- ○ 2/5
- ● 3/5
- ○ 4/5
- ○ 1/3
- ○ 2/3
- ○ 1/2
- ○ 1
- ○ Not enough information

There are 3 samples of successful actions: 1 sample of E East F, and 2 samples of E South H.

The other 2 samples are unsuccessful actions: E East H, and E South D.

(d) [1 pt] Based on $\hat{p}$, what is $\hat{T}$(E, West, D)?

○ 0	● 3/5	○ 2/3	○ Not enough information
○ 1/5	○ 4/5	○ 1/2	
○ 2/5	○ 1/3	○ 1	

Even though we have never seen a sample that moves West from E, we can still use the estimated parameters in the grid world to provide an estimate. Our estimate is that actions succeed with probability 3/5, so moving West from E will succeed (and land in D) with probability 3/5.

(e) [3 pts] Select all true statements about comparing Strategy 1 and Strategy 2.

- ☐ Strategy 1 will usually require fewer samples to estimate the transition function to the same accuracy threshold.
- ☐ There are fewer unknown parameters to learn in Strategy 1.
- ■ Strategy 1 is more prone to overfitting on samples.
- ○ None of the above

Option 1: False. Estimating the transition function directly would require collecting samples for every state-action pair. By contrast, estimating the grid world parameters can be done even if you don't see every state-action pair.

Option 2: False. The transition function has probabilities for every (s, a, s') transition. By contrast, there is only one grid world parameter to estimate, the probability of an action succeeding.

Option 3: True. The transition function for some (s, a, s') transition is only estimated using the samples that start in s and take action a. If the samples for that particular state/action pair are biased, then the transition function for that value will also be biased.

The grid world and samples, repeated for your convenience:

A	B	C
D	E	F
G	H	I

s	a	s'	$R(s,a,s')$
E	East	F	−1
E	East	H	−1
E	South	H	−1
E	South	H	−1
E	South	D	−1

Strategy 3: The agent knows the rules of grid world, and uses an exponential moving average to estimate p. Then, the agent uses the estimated $\hat{p}$ to estimate the transition function.

(f) [2 pts] Consider this update equation: $\hat{p} \leftarrow (1-\alpha)\hat{p} + (\alpha)x$

Given a sample (s, a, s'), what value of x should be used in the corresponding update?

- ○ $R(s,a,s')$
- ● 1.0 if the action succeeded, and 0.0 otherwise
- ○ 1.0 if the action failed, and 0.0 otherwise
- ○ $V(s)$
- ○ $V(s')$

We're trying to estimate p, the probability of an action succeeding.

Consider a sample where the action succeeds. The estimated probability of success from that one sample is 1.0. Similarly, the estimated probability of success from a sample where the action fails is 0.0.

Note that the reward and value are not needed here, because we are not trying to estimate the action of states; instead, we are trying to estimate the probability of success.

(g) [3 pts] Select all true statements about comparing Strategy 2 and Strategy 3.

- ■ Strategy 2 gives a more accurate estimate, because it is the maximum likelihood estimate.
- □ Strategy 3 gives a more accurate estimate, because it gives more weight to more recent samples.
- ■ Strategy 3 can be run with samples streaming in one at a time.
- ○ None of the above

Option 1: True. The maximum likelihood estimate comes from the count estimate.

Option 2: False. In TD learning, we want to give weight to more recent samples because they use more accurate values in their calculation. However, when we're just estimating a probability from independent samples (whose values don't depend on the estimated values of other states), then there's no reason to give more weight to recent samples.

Option 3: True. To compute a count estimate, we need to be able to count up all the samples. The exponential moving average can be computed with each sample streaming in one at a time.

The rest of the question is independent from the previous subparts.

Suppose the agent runs Q-learning in this grid world, with learning rate $0 < \alpha < 1$, and discount factor $\gamma = 1$.

(h) [1 pt] After iterating through the samples once, how many learned Q-values will be nonzero?

○ 0 ○ 1 ● 2 ○ 3 ○ 4 ○ > 4

Q(E, East) and Q(E, South) will be nonzero.

(i) [1 pt] After iterating through the samples repeatedly until convergence, how many learned Q-values will be nonzero?

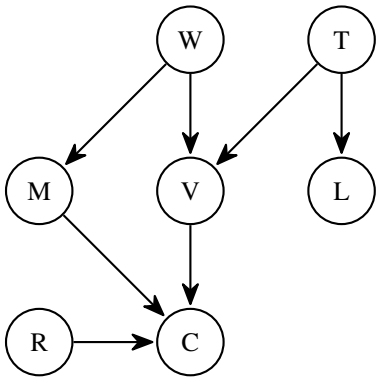
○ 0 ○ 1 ● 2 ○ 3 ○ 4 ○ > 4

Still the same two nonzero values. In order to update a Q-state, we have to see a sample with that Q-state, and we only ever see two Q-states.

Q5. [16 pts] Bayes' Nets: Pac Pong Performance

Pacman is going out to play a Pong tournament, and he wants to model all the uncertainty in the game in order to ensure victory.

Consider the Bayes' net below. All random variables are binary.



Pacman is trying to compute $P(\mathrm{C}|+\mathrm{w},-\mathrm{t})$.

First, Pacman uses the standard variable elimination algorithm (as seen in lecture) to eliminate L and M.

(a) [2 pts] Select the factor(s) needed to eliminate L and M.

- ☐ $P(+\mathrm{w})$
- ☐ $P(-\mathrm{t})$
- ■ $P(\mathrm{M}|+\mathrm{w})$
- ☐ $P(\mathrm{C}|+\mathrm{w})$
- ☐ $P(\mathrm{C}|\mathrm{R},+\mathrm{w},\mathrm{V})$
- ■ $P(\mathrm{L}|-\mathrm{t})$
- ■ $P(\mathrm{C}|\mathrm{R},\mathrm{M},\mathrm{V})$
- ☐ $P(\mathrm{R})$

These are all the factors that mention L or M.

(b) [2 pts] Select the factor(s) that remain after eliminating L and M.

- ■ $f(\mathrm{R},+\mathrm{w},\mathrm{V},\mathrm{C})$
- ☐ $f(\mathrm{R},+\mathrm{w},-\mathrm{t},\mathrm{V},\mathrm{C})$
- ■ $f(-\mathrm{t})$
- ☐ $f(+\mathrm{w},\mathrm{V},\mathrm{C})$
- ■ $f(\mathrm{V},+\mathrm{w},-\mathrm{t})$
- ■ $f(+\mathrm{w})$

Initial list of factors:

$P(+w)$

$P(-t)$

$P(M|+w)$

$P(V|+w,+t)$

$P(L|-t)$

$P(R)$

$P(C|R,M,V)$

When we join on L, we combine all factors involving L. There's only one factor involving L:

$P(L|-t)$

When we sum out L, this factor sums to 1, so we can safely drop it the list of factors.

$\sum_l P(l|-t) = 1$

When we join on M, we combine all factors involving M:

$P(M|+w) \cdot P(C|R,M,V) = f(M,+w,C,R,V)$

When we sum out M, we get:

$\sum_m f(m, +w, C, R, V) = f(+w, C, R, V)$

The remaining list of factors is:

$P(+w)$

$P(-t)$

$P(V \mid +w, -t)$

$P(R)$

$f(+w, C, R, V)$

Note: An earlier version of the solutions did not mark these three answer choices: $f(-\text{t})$, $f(\text{V}, +\text{w}, -\text{t})$, $f(+\text{w})$

The question was not clear on whether we wanted you to select the factors generated from eliminating L and M, or the entire list of factors after eliminating L and M. If you use the former interpretation, then those three answer choices would not be marked. If you use the latter interpretation, then those three answer choices would be marked.

During grading, we gave everybody points for these three choices, regardless of whether you selected them or did not select them. (The other three answer choices are unaffected by the alternate interpretation.)

(c) [2 pts] Which statement best explains why variable elimination (VE) is better than inference by enumeration (IBE)?

- ○ VE can use larger factors, so we can read off more values and use less computation.
- ● VE can use smaller factors, so we can use less computation.
- ○ While VE is usually better, sometimes IBE creates smaller factors.
- ○ VE is always strictly better, because it always uses strictly smaller factors, using less computation.

Note that the factor created by IBE is the full joint distribution, since we're multiplying every CPT in the Bayes' net together. The full joint distribution over all the variables is the largest possible factor we could generate. The factors generated by VE must be smaller or equal to the factor generated by IBE. (There will never be a factor greater than the joint distribution factor generated by IBE.)

In the worst case, VE will generate the entire joint distribution, just as in VE.

(d) [2 pts] Pac-Man claims that he could have skipped eliminating L and just removed it from the graph for his query of $P(\text{C} \mid +\text{w}, -\text{t})$. This is:

- ○ Correct, because L is not directly connected with C.
- ○ Correct, because there is no downstream path from C to L.
- ● Correct, because L is conditionally independent of C, given w and t.
- ○ Incorrect, because every variable can have relevant information to every query.
- ○ Incorrect, because there exists a path from L to C.

The first option is not correct. Hidden variables that are not directly connected to the query variable can still provide relevant information through other nodes to the query variable. (For example, W affects C).

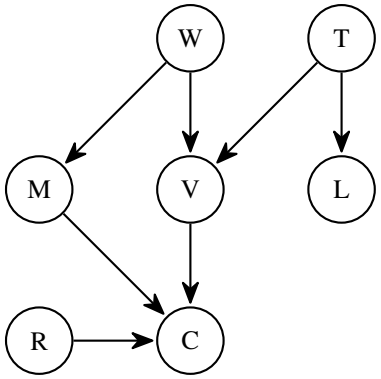
The third option is correct, because variables that don't have information relevant to our query variable can be directly removed.

The fourth option is incorrect, and the explanation stems from the third option's explanation.

The fifth option is incorrect, and the explanation stems from the third option's explanation.

Reminder: Pacman is trying to compute $P(\mathrm{C}|+\mathrm{w},-\mathrm{t})$.

The Bayes' net, repeated for your convenience:



Pac-Man is tired of variable elimination, so he takes a short break. He decides to approximate the query using various sampling methods.

In each of the next subparts, select all sampling algorithm(s) that could have generated the two **consecutive** samples shown.

Note: If Pacman rejects a sample while generating it, the rest of the sample will say "rejected."

(e) [1 pt]

Sample 1:	+r	+w	+t	−v	+m	+l	+c
Sample 2:	−r	−w	−t	+v	−m	+l	+c

- [x] Prior Sampling
- [ ] Rejection Sampling
- [ ] Likelihood Weighting
- [ ] Gibbs Sampling
- ○ None of the above

(f) [1 pt]

Sample 1:	−r	+w	−t	−v	+m	+l	−c
Sample 2:	−r	+w	−t	+v	+m	+l	−c

- [x] Prior Sampling
- [x] Rejection Sampling
- [x] Likelihood Weighting
- [x] Gibbs Sampling
- ○ None of the above

(g) [1 pt]

Sample 1:	−r	+w	+t	rejected	rejected	rejected	rejected
Sample 2:	−r	+w	−t	rejected	rejected	rejected	rejected

- [ ] Prior Sampling
- [ ] Rejection Sampling
- [ ] Likelihood Weighting
- [ ] Gibbs Sampling
- ● None of the above

(h) [1 pt]

Sample 1:	+r	+w	−t	+v	−m	+l	−c
Sample 2:	−r	+w	−t	−v	−m	+l	−c

- [x] Prior Sampling
- [x] Rejection Sampling
- [x] Likelihood Weighting
- [ ] Gibbs Sampling
- ○ None of the above

For the first table, only Prior Sampling would have samples collected that do not agree with the evidence.

For the second table, all algorithms could use these samples. Rejections sampling and likelihood weighting could create these samples, since $+w$ and $-t$ are consistent with the evidence. Gibbs sampling could consecutively create these samples, since they only differ in one variable, v.

For the third table, none of the algorithms work. Note that rejection sampling should continue in the second entry.

For the fourth table, Gibbs sampling does not work because both r and v change values. Rejection sampling and likelihood weighting could create these samples, since $+w$ and $-t$ are consistent with the evidence.

(i) [4 pts] Pacman finishes his variable elimination, in the order: L, M, R, V.

Pacman discovers that the evidence he used was wrong! The evidence was +t, not −t.

Select all steps that Pacman would need to redo. Note that the step of collecting samples does **not** involve computing the probability from those samples.

- ☐ Joining and eliminating M.
- ☐ Joining and eliminating R.
- ■ Joining and eliminating V.
- ☐ Collecting samples for prior sampling.
- ■ Collecting samples for rejection sampling.
- ■ Collecting samples for likelihood weighting.
- ■ Collecting samples for Gibbs sampling.
- ○ No steps need to be redone.

From earlier in the question, note that joining and eliminating L and M never involved any factor or CPT that included T, so that work will stay the same, even if the evidence T is different.

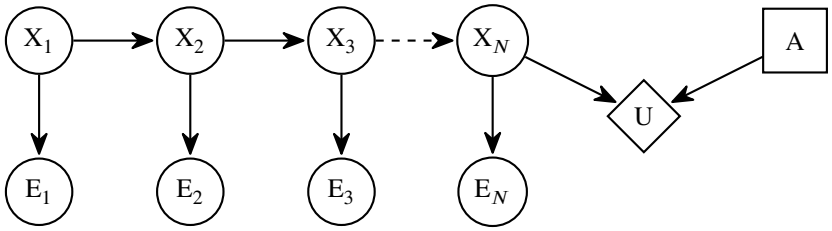
However, when we join and eliminate on V, we will need the CPT $P(V|+w,+t)$. This table will be different depending on the value of the evidence variable T. If we change the evidence, we will need to redo this step and use the corrected CPT.

The samples for prior sampling assume nothing about the evidence, so the samples don't need to be re-collected when the evidence changes.

Rejection sampling, likelihood weighting, and Gibbs sampling all only produce samples consistent with the evidence. If we change the evidence, none of the samples are consistent with the evidence anymore, so we have to re-collect samples.

Q6. [13 pts] HMMs and VPI: Markov Menagerie

Consider the following hidden Markov model (HMM). All random variables are binary.



We would like to compute VPI(E_N) = MEU(E_N) − MEU(∅).

(a) [1 pt] Which of these distributions is needed to compute MEU(∅)?

● $P(X_N)$ ○ $P(X_N|X_{N-1})$ ○ $P(X_N|E_1,\ldots,E_N)$ ○ $P(X_N|E_N)$

The utility U depends on X_N, so we need a distribution over X_N to compute the expected utility of each action.
MEU(∅) means that we are given no evidence, so the distribution over X_N should also be given no evidence.

(b) [2 pts] Which of these computations can be used to derive the distribution in part **(a)**?

○ Run the forward algorithm with no modifications.
○ Run the forward algorithm, skipping all time elapse updates.
● Run the forward algorithm, skipping all observation updates.
○ Read the conditional probability table under X_N.

We need $P(X_N)$. This is not in the Bayes' net; the CPT under X_N is $P(X_N|X_{N-1})$.
To obtain $P(X_N)$, we need to run the forward algorithm, skipping all observation updates, because we have no evidence.

(c) [3 pts] Write an expression that can be used to compute MEU(E_N).

$$\text{MEU}(E_N) = \sum_{e_N} \textbf{(i)} \left[\max_a \sum_{x_N} \textbf{(ii)} \ \textbf{(iii)} \right]$$

(i) ● $P(E_N)$ ○ $P(E_N|E_{N-1})$ ○ $P(E_N|X_N)$ ○ $P(E_N|X_1,\ldots,X_N)$

(ii) ○ $P(X_N|X_{N-1})$ ○ $P(X_N|X_1,\ldots,X_{N-1})$ ● $P(X_N|E_N)$ ○ $P(X_N|E_1,\ldots,E_N)$

(iii) ○ 1 ○ $U(x_N)$ ○ $U(a)$ ● $U(x_N,a)$

$$\sum_{e_N} P(e_N) \left[\max_a \sum_{x_N} P(x_N|e_N) U(x_N,a) \right]$$

We need $P(X_N|E_N)$ to compute the expected utility of each action.
Then, we also need $P(E_N)$ so that we can weight each expected utility by the probability of that specific evidence occurring.
Finally, we need the utility, which depends on both the value of X_N and the action selected.

(d) [2 pts] Consider the distribution in part **(a)**, which you computed in part **(b)**.
From the distribution in **(a)**, which of the following additional computations results in the distribution in blank **(ii)**?

- [x] Apply an additional observation update.
- [ ] Apply an additional time elapse update.
- [ ] Apply an additional time elapse and observation update.
- [ ] Re-run the forward algorithm from the beginning, with no modifications.

From the previous subpart, we have $P(X_N)$, but we need $P(X_N|E_N)$. This requires one extra evidence update in the forward algorithm: we need to weight every value in the table $P(X_N)$ by $P(E_N|X_N)$, and then normalize.

In equations: We have $P(X_N)$ (from the earlier subparts) and $P(E_N|X_N)$ (from the Bayes' net). We want $P(X_N|E_N)$, so we can apply Bayes' rule:

$$P(X_N|E_N) = \frac{P(X_N)P(E_N|X_N)}{P(E_N)}$$

In other words, in the numerator, we're multiplying every value in the $P(X_N)$ table by $P(E_N|X_N)$, and then we normalize (since the denominator is a constant).

(e) [2 pts] Which of the following computations can be used to derive the distribution in blank **(i)**?

- [ ] $P(x_N)P(E_N|x_N)$
- [x] $\sum_{x_N} P(x_N)P(E_N|x_N)$
- [ ] $\sum_{e_N} P(x_N)P(e_N|x_N)$
- [ ] $\sum_{x_N} P(E_N)P(x_N|E_N)$

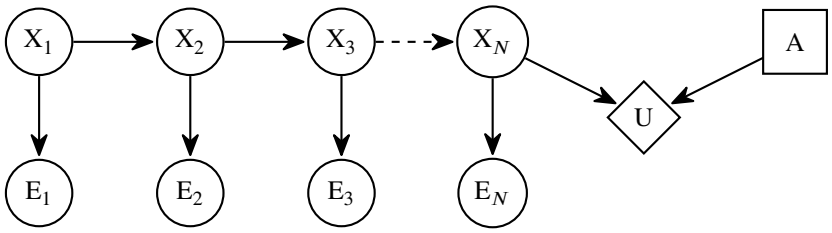
We're given $P(X_N)$ (from earlier subparts), and $P(E_N|X_N)$ (from the Bayes' net). Using the chain rule, we can get:

$$P(X_N)P(E_N|X_N) = P(X_N, E_N)$$

Then, you can sum out X_N to get:

$$\sum_{x_N} P(x_N)P(E_N|x_N) = P(E_N)$$

The diagram, repeated for your convenience:



For the rest of the question, suppose we would like to compute $\text{VPI}(E_1, \ldots, E_N) = \text{MEU}(E_1, \ldots, E_N) - \text{MEU}(\emptyset)$.

(f) [2 pts] To compute $\text{MEU}(E_1, \ldots, E_N)$, we'll need one or more distribution(s) over X_N.

Which of these computations will generate the necessary distribution(s) over X_N?

- ○ Run the forward algorithm once, with no modifications.
- ● Run the forward algorithm 2^N times, with no modifications.
- ○ Run the forward algorithm once, skipping all observation updates.
- ○ Run the forward algorithm 2^N times, skipping all observation updates.

We don't know what the evidence is, so we have to run the forward algorithm once for every possible setting of the evidence $P(X_N|e_1, \ldots, e_N)$.

(g) [1 pt] To compute $\text{MEU}(E_1, \ldots, E_N)$, which other distribution do we need?

○ $P(E_1)$ ○ $P(E_N)$ ● $P(E_1, \ldots, E_N)$ ○ $P(E_1, \ldots, E_N|X_N)$

We need to weight each $\text{MEU}(e_1, \ldots, e_N)$ by the corresponding probability of evidence, $P(E_1, \ldots, E_N)$.

Q7. [12 pts] Games and ML: BeatBlue

Pacman wants to design an agent that can play chess and beat his older (more successful) brother, DeepBlue.

First, Pacman would like to design a machine learning algorithm that takes in a board state and outputs a real number between 0.00 (for states where Pacman is losing) and 1.00 (for states where Pacman is winning).

(a) [2 pts] Pacman starts by asking experts to manually assign values to board states. Select all true statements.

- [x] We can use the manually-assigned values as our training dataset.
- [x] We can use the manually-assigned values as our testing dataset.
- [ ] Since we have values assigned by experts, the machine learning algorithm provides no additional benefit.
- ○ None of the above

First two options are true, because we use hand-labeled data for both training and testing.

Third option is false, because the machine learning algorithm might be able to help us assign values to states that we've never seen before, and we know from class that the state space of a game like chess is so large that it would be impossible to enumerate and manually assign values to every possible board state.

(b) [1 pt] Pacman suggests using a perceptron for this problem. Is it reasonable to use a perceptron for this problem?

- ○ Yes, because this is a classification problem.
- ○ Yes, because the training data is linearly separable.
- ● No, because this is a regression problem, and perceptrons output discrete classes, not continuous numbers.
- ○ No, because perceptrons should never be used when the data is not linearly separable.

The last option is false because perceptrons can still be used when data is not linearly separable; they just wouldn't have perfect training accuracy.

Pacman decides to represent the chess board state as a 64-dimensional vector. Pacman designs a fully-connected, feed-forward neural network with the following architecture:

$$h = \text{ReLU}(x \cdot W_1 + b_1)$$
$$\hat{y} = \text{ReLU}(h \cdot W_2 + b_2)$$

This network takes in a 1×64 vector, x, and outputs a scalar real number, $\hat{y}$.

Pacman sets the hidden layer size to be 128. In other words, h has dimensions 1×128.

(c) [1 pt] What are the dimensions of W_1?

○ 1×1	○ 128×1	○ 8192×1	○ 128×128
○ 64×1	○ 192×1	○ 64×64	● 64×128

From the question: x has dimension 1×64, and h has dimension 1×128.

In order to make the dimensions line up, W_1 must have dimension 64×128, so that we have $(1, 64) \times (64, 128) \rightarrow (1, 128)$.

Intuitively, the 64×128 weight array is converting our 64-dimensional input vector into a 128-dimensional hidden layer vector. In other words, if you think of the hidden layer output as the output of 128 different perceptrons, then the weight array contains 128 different weight vectors, where each weight vector is 64-dimensional.

(d) [1 pt] What are the dimensions of W_2?

- ○ 1×1
- ● 128×1
- ○ 8192×1
- ○ 128×128
- ○ 64×1
- ○ 192×1
- ○ 64×64
- ○ 64×128

From the question: h has dimension 1×128, and $\hat{y}$ is a scalar, so it has dimension 1×1

In order to make the dimensions line up, W_2 must have dimension 128×1, so that we have $(1, 128) \times (128, 1) \rightarrow (1, 1)$.

Intuitively, this layer is a single neuron taking the 128-dimensional hidden layer vector and outputting a single scalar as output. This neuron requires a 128-dimensional weight vector, so that we can take a dot product between the weight vector and hidden layer vector.

(e) [1 pt] Pacman considers changing the second layer of the neural network. Which of these proposed changes is best for this problem?

Reminders: $\text{ReLU}(x) = \max(x, 0)$ $\sigma(x) = \frac{1}{1+e^{-x}}$ $\text{sgn}(x) = \begin{cases} 1 & x > 0 \\ 0 & x = 0 \\ -1 & x < 0 \end{cases}$

- ○ $\hat{y} = \text{ReLU}(h \cdot W_2 + b_2)$
- ○ $\hat{y} = \text{sgn}(h \cdot W_2 + b_2)$
- ● $\hat{y} = \sigma(h \cdot W_2 + b_2)$
- ○ $\hat{y} = h \cdot W_2 + b_2$

The output of the machine learning algorithm needs to be a real number between 0 and 1.

The sigmoid function is the only function in the answer choices that always outputs a real number between 0 and 1.

ReLU could output positive numbers greater than 1. The sgn function could output -1. Without any non-linearity at the last layer, the outputs could be outside of the range of 0 to 1.

Pacman now has a neural network $\hat{y} = f(x)$ that takes in board states (x) and outputs values ($\hat{y}$), and would like to use the network to select actions in the chess game.

(f) [1 pt] Let $s' = g(s, a)$ represent the successor function that takes in a state s and action a, and outputs a successor state s'.

Which of the following expressions represents a reflex agent's optimal action from state s, based on the values outputted by the neural network?

- ○ $\arg\max_s f(s)$
- ○ $\max_a g(s, a)$
- ● $\arg\max_a f(g(s, a))$
- ○ $\sum_a f(g(s, a))$

For every action a, generate a successor state $g(s, a)$, then use f to evaluate this successor state. Use argmax to pick the action that led to the successor state with the highest value.

Finally, Pacman decides to run depth-limited minimax search and use the neural network as an evaluation function.

(g) [1 pt] Is it reasonable to use the neural network as an evaluation function?

- ● Yes, because the neural network maps states to real-numbered values.
- ○ Yes, because the network is guaranteed to correctly identify terminal states where the game is over.
- ○ No, because it would be more efficient and accurate to expand the entire minimax tree to the terminal states.
- ○ No, because evaluation functions should map states to actions.

The other Yes option is false because depending on the weights in the neural network, it might not correctly identify terminal states. There is also no requirement that evaluation functions need to correctly identify terminal states.

(h) [1 pt] Is it possible to use alpha-beta pruning in this problem?

- ● Yes, pruning works even if the neural network output was unbounded.
- ○ Yes, but only because the neural network only outputs numbers between 0 and 1.
- ○ No, because pruning requires you to know the values at all leaf nodes in advance.
- ○ No, because the values outputted by the neural network are not guaranteed to be correct.

In this question, we're running standard minimax (with a specific evaluation function learned from a neural network), so standard alpha-beta pruning works as well.

(i) [1 pt] Is it better to spend more time on training the neural network, or expanding more layers of the game tree?

- ○ Training the neural network
- ○ Expanding the game tree
- ● Not enough information

This depends on how accurate the neural network is, and in practice, it would probably require empirical studies to determine where the time is better spent.

(j) [2 pts] Recall that in Monte Carlo tree search, we allocate more rollouts to game states that are more promising (i.e. better utility for Pacman).

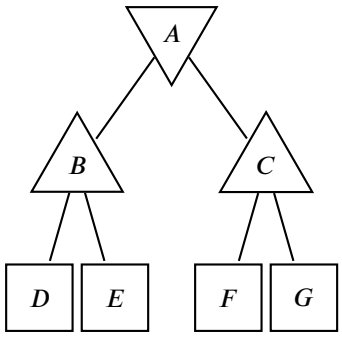
Inspired by this idea, Pacman considers running gradient descent for a longer time when using the neural network to evaluate game states that are more promising. Would this idea work?

- ○ Yes, because this causes the neural network to focus its training on promising game states.
- ○ Yes, because training for a longer time leads to better evaluations.
- ○ No, because running gradient descent for too long always leads to overfitting.
- ● No, because gradient descent runs during training, not evaluation.

The third option is false because running gradient descent for too long doesn't necessarily lead to overfitting.

Q8. [13 pts] Potpourri

(a) [4 pts] Select all true statements about the game tree shown.



Suppose that the values in the terminal nodes have the property that $D < E < F < G$, and the agents do not know this.

Assume that alpha-beta pruning visits nodes from left to right.

- [x] Two actions are needed to transition from state A to state E.
- [x] The value at the root node will always be E.
- [x] When running alpha-beta pruning, node G can always be pruned.
- [ ] When running alpha-beta pruning, node F can sometimes be pruned.
- ○ None of the above

Option 1: True. Edges represent actions. There are two edges on the path between A and E.

Option 2: True. We have:

$B = \max(D, E) = E$

$C = \max(F, G) = G$

$A = \min(B, C) = \min(E, G) = E$

Option 3: True. By the time we see F, we know that $C \geq F > E$. The minimizing agent will always prefer going to B and getting a rewards of E, and will never go to C and get a reward greater than E. Therefore, we don't have to check the value of G to know that the minimizing agent is going to B.

Option 4: False. F and G could be very negative numbers, which would cause the minimizing agent to prefer C over B. We have to check F before we know for sure that the minimizing agent will never prefer C.

(b) [4 pts] Consider a standard HMM (from lecture) with states $X_1, \ldots, X_N$ and evidence $E_1, \ldots, E_N$.

Suppose we're running particle filtering with a large number of particles. At time step t, we have a particle at state $X_t = x_t$, with weight 0.7. Select all true statements about this particle.

- [ ] We've just finished a time elapse update, and have not performed the observation update for time t yet.
- [ ] 0.7 is the probability of the particle visiting all the states it's visited so far: $P(x_1, \ldots, x_t)$.
- [x] If this particle is drawn during resampling, the resulting new particle will still be at state x_t.
- [ ] 0.3 is the probability that this particle disappears during resampling.
- ○ None of the above

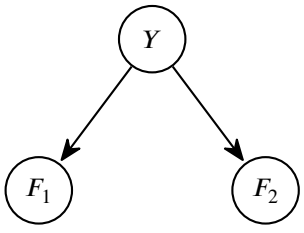
Option 1: False. The particles take on weights after the observation update, not after the time elapse update.

Option 2: False. 0.7 is the probability of the evidence, given the particle's current state. $P(e_t|x_t)$ is not equal to $P(x_1, \ldots, x_t)$.

Option 3: True. The resampling process does not change the particles' states.

Option 4: False. The weight of a particle is not the probability that it is resampled, because the weights of the particles are not normalized (don't sum to 1), and because we are drawing many particles from the weighted particle distribution, with replacement (so there are many chances for the particle to be resampled).

Consider the following Naive Bayes' model and training data. Y, F_1, and F_2 are binary random variables.



F_1	F_2	Y
$+f_1$	$+f_2$	$+y$
$+f_1$	$-f_2$	$+y$
$-f_1$	$+f_2$	$-y$

The CPT under Y:

$$P(+y) = 2/3$$
$$P(-y) = 1/3$$

The CPT under F_1:

$$P(+f_1|+y) = 1$$
$$P(-f_1|+y) = 0$$
$$P(+f_1|-y) = 0$$
$$P(-f_1|-y) = 1$$

The CPT under F_2:

$$P(+f_2|+y) = 0.5$$
$$P(-f_2|+y) = 0.5$$
$$P(+f_2|-y) = 1$$
$$P(-f_2|-y) = 0$$

(c) [2 pts] What is $P(+y \mid +f_1, +f_2)$?

- ○ 0
- ○ 1/3
- ○ 2/3
- ○ 1/2
- ● 1
- ● Not enough information

$$\begin{aligned} P(+y, +f_1, +f_2) &= P(+y) \cdot P(+f_1|+y) \cdot P(+f_2|+y) \\ &= (2/3) \cdot 1 \cdot (1/2) \\ &= 1/3 \\ P(-y, +f_1, +f_2) &= P(-y) \cdot P(+f_1|-y) \cdot P(+f_2|-y) \\ &= (1/3) \cdot 0 \cdot 1 \\ &= 0 \end{aligned}$$

After normalizing, we get $P(+y \mid +f_1, +f_2) = 1$.

Note: During grading, we accepted "Not enough information" as an alternate solution. The intended answer was 1, using the calculations shown above. However, some students pointed out that while 1 is the estimate for $P(+y| + f_1, f_2)$ given by the training data, it may not be the true probability given the full space of possible data points.

Update: the answer choice "None of the above" was accepted an an alternate answer choice, due to the reasoning that the samples given cannot represent the true probabilities of the distributions involved, and so you cannot compute the true probability asked for.

(d) [3 pts] Select all equations that are true about the model, regardless of the training data used.

- ☐ $P(F_1) = P(F_1|F_2)$
- ☐ $P(Y) = P(Y|F_1)$
- ■ $P(F_1|Y) = P(F_1|Y, F_2)$
- ○ None of the above

By the definition of Naive Bayes' or d-separation, F_1 and F_2 are conditionally independent, given Y.

By definition of conditional independence, $P(F_1|Y) = P(F_1|Y, F_2)$. In words, if you already know the label Y, then any additional features like F_2 tell you nothing more about F_1.