CS 188 Summer 2024 Introduction to Artificial Intelligence — Final

Solutions last updated: August 11th, 2024

- You have 170 minutes.
- The exam is closed book, no calculator, and closed notes, other than three double-sided cheat sheets that you may reference.
- Anything you write outside the answer boxes or you ~~cross out~~ will not be graded. If you write multiple answers, your answer is ambiguous, or the bubble/checkbox is not entirely filled in, we will grade the worst interpretation.

For questions with **circular bubbles**, you may select only one choice.

○ Unselected option (completely unfilled)

● Only one selected option (completely filled)

◍ Don't do this (it will be graded as incorrect)

For questions with **square checkboxes**, you may select one or more choices.

■ You can select

■ multiple squares (completely filled)

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."

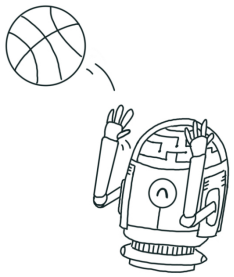
By signing below, I affirm that all work on this exam is my own work, and honestly reflects my own understanding of the course material. I have not referenced any outside materials (other than my cheat sheets), nor collaborated with any other human being on this exam. I understand that if the exam proctor 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.	Potpourri	20
Q2.	Search & Games: Lights, Camera, Action!	14
Q3.	HMMs: Inside Out	13
Q4.	BN, Naive Bayes, Perceptron: Bayesketball	14
Q5.	MDPs: Easy MDPeasy	11
Q6.	RL: Weight a Minute!	16
Q7.	ML: N&Ns	12
	Total	100

You miss 100% of the shots you don't take!



Drawing by Samantha Huang

Q1. [20 pts] Potpourri

Note: This question is relatively long, so be sure to skip ahead and come back to it if you get stuck!

(a) [2 pts] Select all true statements.

- [x] Depth-first graph search is guaranteed to be complete in a finite-space search problem.
- [x] If the cost of expanding any node is 1, breadth-first search is guaranteed to be both optimal and complete.
- [x] If $h_1(x)$ and $h_2(x)$ are two consistent heuristics, then $h_3(x) = \max\{h_1(x), h_2(x)\}$ is also consistent.
- [ ] In Monte Carlo Tree Search, performing more rollouts will always yield better decisions.
- ○ None of the above.

The first option is correct because, even though depth-first tree search is not complete, adding a visited set for graph search tracks states that have been visited before. This guarantees that we do not visit the same state twice, so we will eventually visit all possible states and can't get stuck in an infinite loop.

The second option is correct because we know that uniform cost search is both optimal and complete. Given that the cost of expanding all nodes is 1, performing uniform cost search is equivalent to performing breadth-first search, making it both optimal and complete.

The third option is correct because if both h_1 and h_2 are consistent, $h(A) - h(B) \leq cost(A, B)$. Note that both Δh_1 and Δh_2 pass the condition, so the only danger is if some Δh_3 evaluates to $h_1(A) - h_2(B)$ (or vice-versa), which requires $h_1(A) \geq h_2(A)$ and $h_2(B) \geq h_1(B)$. We can compare against Δh_1 and get that $h_1(A) - h_2(B) \leq h_1(A) - h_2(A)$ because $h_2(B) \geq h_1(B)$, meaning that we are subtracting a potentially bigger but not smaller number.

The fourth option is incorrect. While the number of rollouts in MCTS approaches infinity, the optimal decision is found, is may find worse outcomes as the number of rollouts get larger due to the stochastic nature of MCTS.

(b) [1 pt] We have a six-sided die, labeled one through six. If we end up observing the seven die-rolls of $6, 3, 4, 2, 5, 6, 1$ from this die, what is the maximum likelihood estimate of p, the probability of rolling a 6?

$\frac{2}{7}$

We have 7 observations, and 2 of those outcomes end up as 6.

(c) [2 pts] Is the perceptron algorithm a form of supervised learning, or unsupervised learning? Explain your answer choice in three sentences or fewer.

● Supervised Learning ○ Unsupervised Learning ○ Neither

It's a form of supervised learning because we're training our algorithm using labeled data. In supervised learning, training is done using data that is already labelled for us, rather than data that isn't labelled.

(d) [1 pt] Select the true statement about HMM's.

- ○ The Viterbi algorithm, for each state at time t, keeps track of the total probability of all paths to it.
- ● The Viterbi algorithm chooses the state sequence that maximizes the likelihood of the observation sequence.

The second option is the goal of the Viterbi algorithm. The first answer choice applies to the forward algorithm, not the Viterbi algorithm.

(e) [3 pts] Suppose we have a Bayes net with binary variables A, B, C, D. The conditional probability tables (CPT's) for this Bayes net are $P(A|B)$, $P(B)$, $P(C|A, B)$, and $P(D|C)$. Using Likelihood weighting to estimate the query $\mathbf{P}(-\mathbf{b}| + \mathbf{a}, +\mathbf{d})$, we obtain the two samples below.

Fill in the weight for each sample as a product of probabilities. For instance, an example (but incorrect) answer would look like $P(+a| - c)P(-a| + b, -d)$.

Sample 1: $(+a, +b, +c, +d)$: $P(+a| + b)P(+d| + c)$

Sample 2: $(+a, -b, -c, +d)$: $P(+a| - b)P(+d| - c)$

By the algorithm, we multiply each particle by the Bayes CPT probability of each observation we had to assign. One way to see why this works is that this corresponds to, in rejection filtering, the proportion of the pre- observation assignment stubs that would naturally get the correct assignment and not get filtered.

Wesley is assigning robots R_1, R_2, R_3, R_4, and R_5 to translate texts from English into the languages *German*, *Chinese*, *Japanese*, *Arabic*, and *Hindi*. Each robot will only be able to translate English into exactly one of the five languages, and so Wesley decides to model this as a CSP. Additionally, Wesley has listed the following restrictions he needs to follow:

1. R_3 will only translate *Japanese*.
2. R_1 and R_2 will translate the same language.
3. R_4 will translate a language that is different from the rest of the robots.
4. R_2 will only translate either *German* or *Hindi*.

(f) [2 pts] How many edges are in the constraint graph for this CSP, assuming R_1, R_2, R_3, R_4, and R_5 are the variables?

5

There's one edge between R_1 and R_2 due to constraint 2, and there's also an edge from R_4 to each of R_1, R_2, R_3, and R_5 due to constraint 4.

Suppose we are in the middle of running the AC-3 arc consistency algorithm on this CSP, and the domains for each of the robots currently are as follows, where only R_3 has been assigned a value:

1. $R_1 = \{German, Hindi\}$
2. $R_2 = \{German, Chinese, Japanese, Arabic, Hindi\}$
3. $R_3 = \{Japanese\}$
4. $R_4 = \{German, Chinese, Japanese, Arabic, Hindi\}$
5. $R_5 = \{German, Chinese, Japanese, Arabic, Hindi\}$

(g) [3 pts] After processing each of the arcs below, write down which arcs we need to add back to our queue as a result of it, separated by commas. Assume that these arcs are processed in **isolation** (not one after the other, so the domains stay the same with each arc). You may also ignore the arcs that are **already** in the queue (so you don't have to worry about duplicate arcs in the queue). If no new arcs get added to the queue after the arc is processed, write "None".

For instance, if you believe the arcs $R_1 \rightarrow R_2$, $R_3 \rightarrow R_4$, and $R_5 \rightarrow R_1$ get added to the queue as a result of processing the specific arc, write down "$R_1 \rightarrow R_2, R_3 \rightarrow R_4, R_5 \rightarrow R_1$".

$R_4 \rightarrow R_3$

$R_1 \rightarrow R_4, R_2 \rightarrow R_4, R_5 \rightarrow R_4$

Since R_4 being equal to Japanese doesn't allow R_3 to be any value, we must prune Japanese from R_4's domain and add in the three arcs back in the queue as shown. Note that $R_3 \rightarrow R_4$ does not get added since R_3 has been assigned a value already.

$R_5 \rightarrow R_3$

None

Since every possible value for R_5 has a corresponding value in R_3 that doesn't violate any constraints, no new arcs need to get added to the queue.

(h) [3 pts] Select the following true statements. For all answers, you may assume that "tokens" means roughly the same thing as "words.")

☐ RNNs solve the "memory bottleneck" problem in LSTMs, and so RNNs are better at preserving information about early tokens in a sequence.

☐ Masked language models predict each token based on the previous ones, while autoregressive (causal) language models predict only some fraction of the tokens based on the tokens before and after them.

■ When generating a sequence of tokens, models composed of Transformers use the attention mechanism to capture relationships between the token being generated and earlier tokens in a sequence.

☐ Language models today are first pretrained on small amounts of curated, labeled data, then fine-tuned on large amounts of uncurated, unlabeled data.

☐ After fine-tuning, an efficient and common way to encourage specific behaviors is to use RL with KL control, regardless of whether the specific behavior exists in the pretraining or fine-tuning data.

■ DDQN prevents the Q estimator learning algorithm from propagating a spike ($>> Q^*$) in one Q_{spike} value into the Q values that depend on that Q_{spike}, **but** it does not help with overfitting. Hint: recall that Double Deep Q-Network (DDQN) keeps 2 Q estimators so that in computing $V(s')$ to use in weight updates, one estimator gives $Q(s', a')$ while the other one decides a' such that it maximizes $Q_{\text{other}}(s', a')$.

○ None of the above.

Note that these subparts (h to j) were taught only in the summer 2024 semester.

LSTMs were introduced to improve on RNNs at preserving information about early tokens in a sequence. LSTMs are better than RNNs about preserving information on tokens early in a sequence, but they still have a memory bottleneck; attention was introduced to address the memory bottleneck in LSTMs.

Autoregressive LMs predict each token based on the previous ones. Masked LMs predict some fraction of masked tokens based on the tokens around them.

Language models today are first pretrained on large amounts of uncurated, unlabeled data, then fine-tuned on small amounts of curated, labeled data (large and curated flipped vs. answer option).

If the specific behavior doesn't exist in any of the data that created the model, RL cannot teach the new behavior because it only gives out rewards for the behavior when it happens which is then used to update weights to make high rewards more likely.

DDQN helps to actually better fit the training data by preventing unusual (spike) samples from breaking our model. The Q_{spike} is $>> Q^*$, so it doesn't fit the training data, and we stop it from breaking other Q values to not fit the training data. Helping with overfitting fixes the issue of fitting the training data well but not generalizing to test data.

(i) [2 pts] Bob is a TA for a computer science class. Bob's professor accidentally deletes most of the files in each student's final project (oops!). The professor asks Bob to train a neural network using data from past semesters that takes in each student's past grades, personal information (name, age, etc.), and the remaining code for their final project, and then generate a final project grade with an explanation. The professor says she'll use the model outputs to help her decide on final project grades, then release the trained model online so others can predict grades based on students' past work.

Which of the following problems should Bob be concerned about?

■ The model might learn patterns that are unfair to some students, such as predicting higher grades for older students.

☐ When assigning grades, the professor will most likely trust her own flawed judgment over the model's impartial decision, an example of automation bias.

■ Once the model is released online, other users might be able to extract students' personal information by querying the model.

○ None of the above.

This model is likely to result in allocation bias, such as assigning higher grades to some students in unfair ways based on factors like age or demographic background if those happen to correlate with certain grades. (The UK controversially implemented a similar algorithm to predict A-Level grades during Covid.) This could include predicting higher grades for older students if that's a pattern in the data, even though it's unfair to younger students who did well.

Automation bias is when people are likely to trust a model's flawed prediction over their own (better-informed) judgment, which could certainly happen if the professor trusts a low-quality model prediction over her own knowledge of a student.

Training data can indeed be extracted from publicly released models. This could violate students' rights to have their educational records remain private.

(j) [1 pt] Suppose we have 20 data points, in which 10 of them are considered positive (indicated by a $+$) and the other 10 are considered negative (indicated by a $-$). We want to use a decision tree to split up our data, and we have four features to choose from to split our data. Which of the following splits results in the greatest information gain? Assume that $\big|$ is the symbol for separating groups formed by splitting on the feature.

○ $+-+-+-+-+-\,\big|\,-+-+-+-+-+$

○ $++\,\big|\,-+-+-+-+\,\big|\,--++\,\big|\,++--$

● $++++++++++-\,\big|\,----------+$

○ $+-\,\big|\,+-\,\big|\,--\,\big|\,+++++++++------$

The third answer choice is correct because we almost completely separate our positive data from our negative data, meaning there's a very high information gain.

Q2. [14 pts] Search & Games: Lights, Camera, Action!

Vivien is currently on an $M \times N$ grid, and she wants to take pictures of k sites located on the grid, with each site occupying exactly one square on the grid. At each timestep, she can either move up, down, left, or right exactly one square (assuming this move does not push her out of the grid), or she can stay put and take a picture of her current square. The sites never move, and the k sites are labeled as $s_1, s_2, \ldots, s_k$.

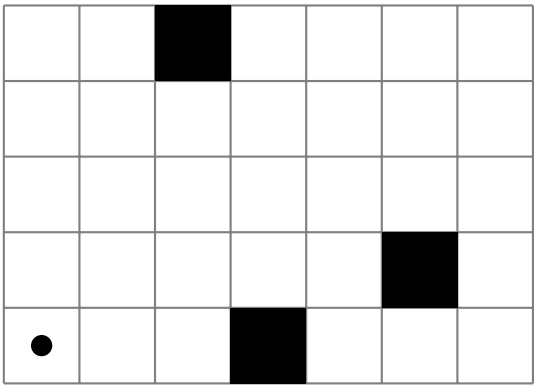


Figure 1: In the **example** 5×7 grid above, assuming that $(1, 1)$ is located at the bottom left square, Vivien is located at $(1, 1)$. In this example, $k = 3$, and the location of the three sites, indicated by filled squares, are $(4, 1)$, $(3, 5)$, and $(6, 2)$.

Vivien's goal is to take a picture of each of the k sites on the grid. She can only take a picture of a site when she is on that site's respective square. Note that Vivien can take a picture of whatever square she's currently on, even if it's a blank square or a square of an already photographed site. This action will essentially do nothing.

Vivien decides to model this as a search problem.

(a) [2 pts] Do the sites' locations **need** to be considered for the state space size calculation? Explain why or why not.

○ Yes ● No

Since the sites do not move between actions, they do not need to be considered for the calculation of the state space.

(b) [2 pts] Select all true statements.

☐ Depth-first tree search is guaranteed to be complete for this search problem.

■ When calculating the branching factor from a state, we take into account actions that result in our agent staying in the same state after taking that action.

■ At least a boolean variable is needed for each site when encoding a state representation for the search problem to check whether that site has been photographed yet.

☐ The optimal solution to this search problem is always unique, meaning there is only ever at most one possible optimal solution.

○ None of the above.

The first option is false because depth-first tree search could potentially get trapped in an infinite cycle, causing the search problem to never finish.

The second option is true because actions that don't do anything are still considered actions.

The third option is true because we need someway to know whether or not we have visited a site yet.

The fourth option is false because there could be multiple optimal paths that Vivien can take. For instance, if the grid is 2x2, Vivien was at the bottom left, and there was one site at the top right, then Vivien could either go up and right or right and up to get to the site.

From this subpart onwards, we examine two scenarios for the search problem, labelling them as scenario 1 and scenario 2:

Scenario 1: Vivien can take pictures of the k sites in any order. The goal is just to have every site be photographed at least once.

Scenario 2: Vivien still wants to take a picture of each of the k sites, but she cannot take a picture of site s_j unless sites $s_1, s_2, \ldots, s_{j-1}$ have all been photographed, for all $1 < j \leq k$.

(c) [2 pts] Vivien implements goal tests $G(s)$ and $G'(s')$ for scenarios 1 and 2, respectively. What is the tightest upper bound (in big O notation) it takes to run $G(s)$ and $G'(s')$, in terms of the variables above (as well as constants, if necessary)? Assume we are using a minimal state space representation, and the functions are optimal (written to be as fast as possible).

$G(s) =$ $O(k)$ $G'(s') =$ $O(1)$

For $G(s)$, the goal state requires checking each of the k sites to make sure that they have been photographed, which ultimately takes $O(k)$ time. However, for $G'(s')$, it only takes $O(1)$ time because only the last site, site s_k, needs to be looked at, since if site s_k has been photographed, it is guaranteed by the problem itself that sites $s_1, s_2, \ldots, s_{k-1}$ have ALSO been photographed.

(d) [3 pts] For each of the following heuristics, determine if it is admissible for scenario 1, scenario 2, both, or neither. Assume the heuristic is zero at a goal state.

One plus the maximum Manhattan distance between Vivien and any of the non-photographed sites.

■ Admissible for scenario 1. ■ Admissible for scenario 2. ○ Inadmissible for both.

We are simplifying the problem to require only one space to be photographed, and solving it exactly. Adding one improves over just the Manhattan distance because we need to spend an action to take the picture.

This is very similar to a Project 1 question where Pacman needs to eat all the foods.

One plus the Manhattan distance between Vivien and s_j, where j is the minimum number such that s_j has not been photographed yet.

■ Admissible for scenario 1. ■ Admissible for scenario 2. ○ Inadmissible for both.

Same as the last one, but we are constrained to photographing some particular spot that's not done yet, instead of the furthest remaining one.

(e) [2 pts] Now, Mustafa joins the search problem with Vivien and starts off at a random square. He can move around and take pictures of sites in the same way Vivien can.

Out of the following game scenarios, select the zero-sum games.

☐ The game ends when both Mustafa and Vivien photograph all k sites, and at the end, each person receives a score equal to the number of moves they made in the game.

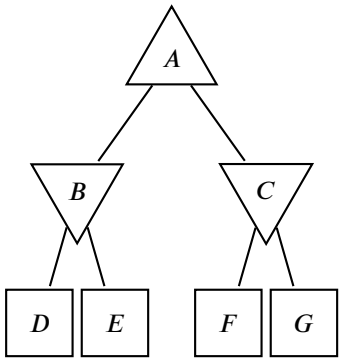
☐ The game ends when all k sites have been photographed, but only ONE of Mustafa or Vivien needs to take a picture for each site. The score is the same as the first option.

■ The game ends when all k sites are photographed, but only ONE of Mustafa or Vivien can photograph each state. However, between Mustafa and Vivien, the person that photographed the MOST of the k sites wins.

◯ None of the above.

Only the third option is zero-sum because in that game, Mustafa's gain is equal to Vivien's loss, and vice-versa. The same cannot be said for the first two options.

(f) [3 pts] Select all true statements about the mini-max game tree shown. It is **independent** from the subproblems before.



Suppose that the values in the terminal nodes have the property that either D or E is the **minimum** value out of the four terminal nodes, and either F or G is the **maximum** value out of the four terminal nodes.

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

☐ Assuming no pruning, any of the four terminal nodes can be propagated up to the root node.

☐ If F is the max terminal node value, then G is guaranteed to always be pruned.

■ This game tree represents a zero-sum game.

■ In general (not just in this game tree), alpha-beta pruning can sometimes result in different values being propagated up compared to standard mini-max tree calculations.

◯ None of the above.

The first option is false because if D is the minimum value, it will be propagated up once, but not propagated up to the root node. Only either node F or G can be the root node.

The second option is false, as F being the maximum terminal node value doesn't tell us anything about the value of G.

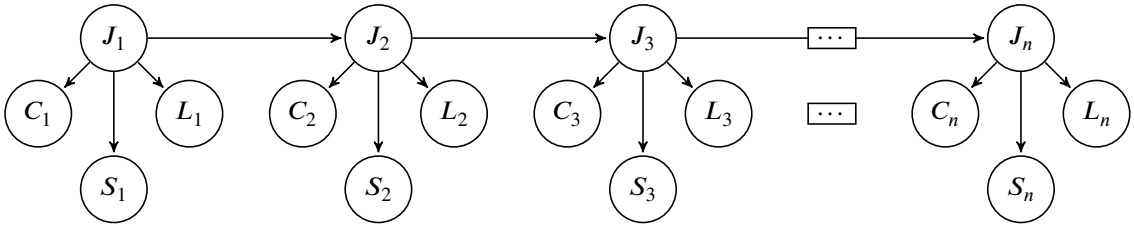
The third option is true, as mini-max game trees always represent zero-sum games.

The fourth option is true, since pruning could potentially not allow us to correctly propagate up the minimum or maximum value. However, alpha-beta pruning will always be sure to allow the root node to choose the same action as in a mini-max game tree.

Q3. [13 pts] HMMs: Inside Out

Riley has 3 different *observed* behaviors for a given time t: smiling (S_t), crying (C_t), and laughing (L_t). At each time step, each of these behaviors will either be observed ($X_t = 1$) or not observed ($X_t = 0$), and this value is dependent on whether or not her primary emotion, Joy, is active at time t ($J_t = 1$). Zero, one, or two behaviors can be observed at a given time. We cannot directly observe the state of Joy.

This setup can be modeled by the following Hidden Markov Model. You may assume that $P(C_i|J_i)$ is the same for all $1 \leq i \leq n$, and you can assume the same for $P(S_i|J_i)$ and $P(L_i|J_i)$.



(a) [2 pts] What is the minimum number of conditional probability tables (CPTs) needed to model this HMM?

- ○ 4
- ● 5
- ○ $4n$
- ○ $5n$

We need prior ($P(J)$) and transition ($P(J_{i+1}|J_i)$) distributions for J, as well as $P(C|J)$, $P(S|J)$, $P(L|J)$.

(b) [3 pts] Assume we observe that $J_2 = 1$. Which variable(s) in the HMM are guaranteed to be conditionally independent of J_1 given the observed variable?

☐ J_1	☐ S_1	☐ C_1	☐ L_1
■ S_2	■ C_2	■ L_2	■ J_3
■ S_3	■ C_3	■ L_3	○ None of the above

The chosen nodes are effectively getting their influence "blocked" by J_2 from J_1.

More formally, we can see that all paths from J_1 to any of the selected variables *must* pass through J_2, guaranteeing that all paths will include a causal chain with J_2 observed. Because all the triples through J_2 are an inactive triple (happens to be the same Unobserved → Observed → Unobserved triple), we know that J_1 is d-separated with all variables that are descendants of J_2.

We want to model our belief distribution $B(J_i)$ for whether Joy is active ($J_i = 1$) or inactive ($J_i = 0$) at any given time step i, given all observations of C, S, and L up to and including time i.

(c) [3 pts] Rewrite the update rule that would be used to derive $B(J_{i+1})$ by filling in the blanks of the expression below.

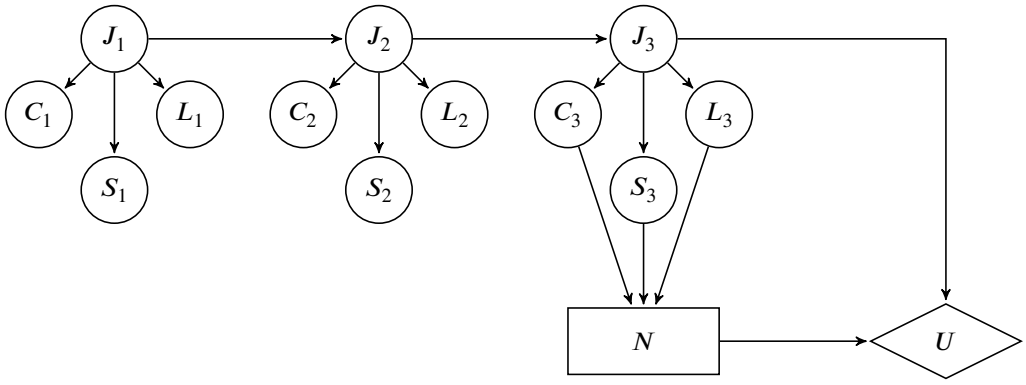
$$B(J_{i+1}) = \textbf{(i)} \cdot \textbf{(ii)} \cdot \sum_{j_i} \Big(\textbf{(iii)} \cdot B(j_i) \Big)$$

(i)	○ 1	○ $P(J_{i+1}\|J_i)$	● $P(C_{i+1}\|J_{i+1})$	○ $\sum_c \sum_\ell \sum_s P(J_i\|c_i, \ell_i, s_i)$
(ii)	○ $P(J_i)$	○ $P(J_{i+1}\|J_i)$	● $P(L_{i+1}\|J_{i+1})P(S_{i+1}\|J_{i+1})$	○ $\sum_s \sum_\ell P(J_{i+1}\|s_{i+1}, \ell_{i+1})$
(iii)	○ $P(j_i)$	● $P(J_{i+1}\|j_i)$	○ $P(j_i\|c_i, s_i, \ell_i)$	○ $P(J_{i+1}\|c_{i+1}, s_{i+1}, \ell_{i+1})$

This is the same as the normal belief distribution update rule, just incorporating the fact that there are 3 evidence variables. Notice that each evidence variable (C_i, S_i, L_i) is independent of all other evidence variables given parent J_i, so the joint probability can be rewritten as $P(C, S, L|J) = P(C|J) \cdot P(S|J) \cdot P(L|J)$.

Riley's parents want to tell her some important news, and want to know who should tell her the news out of the two. After observing her behaviors at the third timestep, they will decide whether the mom gives the news ($N = 1$) or the dad gives the news ($N = 0$). Their utility from Riley's reaction to the news depends on whether Joy is active at time step 3 ($J_3 = 1$).

We can model this new setup as the following decision network:



(d) [1 pt] VPI(C_1, S_1, L_1) _____ VPI(C_2, S_2, L_2)

○ = ○ > ○ ≥ ○ < ● ≤ ○ Not enough information

It's possible that none of these six variables give any information at all, in which case VPI(C_1, S_1, L_1) = VPI(C_2, S_2, L_2) = 0, so equality is always possible.

We know that the conditional probability tables $P(C_1|J_1)$ and $P(C_2|J_2)$, etc. are identical because this is an HMM. So, the observations at step 2 give us the same volume of information about J_2 as 1 about J_1. Finally, J_2 is closer to the U (and in fact would make J_1 conditionally independent of U/ U's parents), so the step 2 is guaranteed to be at least the same VPI or better.

(e) [1 pt] VPI(C_1, S_1, L_1) _____ VPI($C_1, S_1, L_1, C_2, S_2, L_2$)

○ = ○ > ○ ≥ ○ < ● ≤ ○ Not enough information

The right has the same, and some extra variables, so it can't be lower than the left. We can't guarantee that it is the same because the step 2 observations are not conditionally independent of U/ U's parents when conditioned on the step 1 observations.

(f) [1 pt] VPI(C_1, S_1, L_1, J_2) _____ VPI(J_2)

● = ○ > ○ ≥ ○ < ○ ≤ ○ Not enough information

C_1, S_1, L_1 are conditionally independent of U/ U's parents when conditioned on J_2. We can see this with d-Separation, because we will hit the Unobserved → Observed ($= J_2$) → Unobserved inactive triple on all possible paths.

Riley's mom will tell Riley the news ($N = 1$) if they believe their utility will be non-negative ($EU(N = 1) \geq 0$).

J_3	$U(N = 1)$
0	−8
1	10

(g) [2 pts] Let's say Riley's parents are only able to observe Riley's behaviors at time $t = 1$, and they observe the values $C_1 = 0, S_1 = 1, L_1 = 0$. Their choice N at $t = 3$ depends solely on these observations. Should Riley's mom tell her the news given the assumptions about the probability distributions?

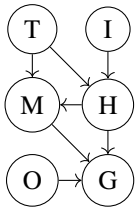
All distributions in the CPT's are uniform.

○ Yes ($N = 1$) ○ No ($N = 0$) ● Not enough information.

It could be the case that $J_3 = 1$ is very unlikely, in which case Riley's mom would not want to tell.

Q4. [14 pts] BN, Naive Bayes, Perceptron: Bayesketball

Wilson is preparing for his basketball game. He wants to predict his team's game outcome (G). He believes the game outcome is influenced by several binary random variables: Team Morale (M), Player Health (H), Training Quality (T), Opponent Strength (O), and Recent Injuries (I). Wilson builds a Bayes Network to represent the dependencies between these variables as follows:



(a) [3 pts] Select the independence statements that are **guaranteed** to be true given this Bayes Net Structure.

- [x] O ⫫ I
- [x] T ⫫ G | M, H
- [ ] I ⫫ G | M
- ○ None of the above.

O ⫫ I: These are both roots of the graph, and don't have any shared observed descendants.

T ⫫ G | M, H: Both of the parents of G are included, and T is an ancestor of G, so they are independent by the base requirement of independences in a Bayes Net. We can also see that any path will encounter the Unobserved → Observed (=M or H) → Unobserved inactive triple.

I ⫫ G | M: We don't have all of the parents. Specifically, the path I → H → G is active because the only triple in the path is active.

Wilson now computes $P(G \mid M = +m, I = -i)$ using variable elimination.

(b) [2 pts] What factors are needed to eliminate H first?

- [ ] $P(T)$
- [ ] $P(I = -i)$
- [x] $P(M = +m \mid T, H)$
- [x] $P(H \mid T, I = -i)$
- [ ] $P(O)$
- [x] $P(G \mid M = +m, H, O)$

All factors that have H in it. The number of factors is always 1+ how many arrows are coming out of the node – because each node is P(node|parents).

The basketball game is about to end and Wilson is passed the ball, forcing him to think about taking the game winning shot. He uses a Naive Bayes classifier to help him predict the outcome of the shot. He chooses three features during classification: Launch Angle (X_1), Velocity (X_2), and Distance (X_3). Below is the training dataset:

Launch Angle (X_1)	**Velocity (X_2)**	**Distance (X_3)**	**Outcome (Y)**
+	+	+	success
+	+	+	success
+	−	+	success
+	+	−	success
+	+	+	fail
+	−	−	fail

(c) [3 pts] Under Naive Bayes assumptions, solve the following probabilities:

$P(Y = \text{success}) =$ $\frac{2}{3}$ $P(Y = \text{fail}) =$ $\frac{1}{3}$

$P(X_1 = + \mid Y = \text{success}) =$ 1 $P(X_1 = + \mid Y = \text{fail}) =$ 1

All of these are just the count of samples matching all variable assignments divided by the number of samples matching the conditioned variable assignments. This is the MLE assuming each feature is conditionally independent of each other given the label.

Wilson decides to use a perceptron instead to label whether a shot will be a success (+) or fail (−). He uses 2 features: the Spin and Angle Offset (Angle), which can both be positive or negative integers.

Consider the following labeled training data, where each row signifies a shot:

Spin	Angle	Success (+) or Fail (−)
−5	1	+
3	−2	−
−2	−1	−
4	2	+

(d) [2 pts] Our perceptron weights have been initialized to $w_{\text{spin}} = 1$ and $w_{\text{angle}} = -1$. After processing the first row with the perceptron algorithm, what will be the updated values for these weights?

$w_{\text{spin}} =$ −4 $\qquad$ $w_{\text{angle}} =$ 0

We misclassified when the true label is positive, so ADD the data to the current weights. [1, -1] + [-5, 1] = [-4, 0]

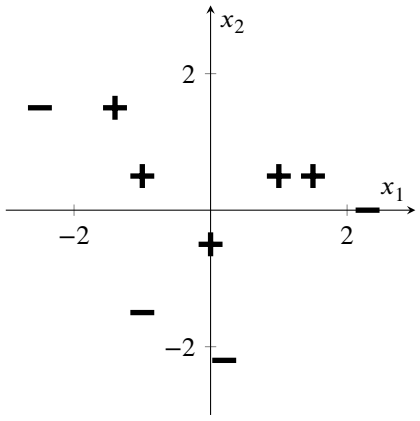
(e) [1 pt] Is the data linearly separable? *Hint:* You do not need to run the perceptron algorithm to figure this out.

● Yes $\qquad$ ○ No

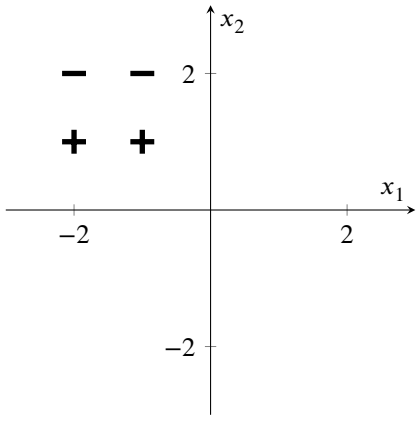
Looking for easy lines that can separate the data, we can see that all fails have a low angle and all successes have a high angle, so we can separate on just the one angle feature.

Specifically, Angle=0 separates the data, which corresponds to $w_{\text{spin}} = 0$ and $w_{\text{angle}} = 1$ or any positive number.

(f) [3 pts] For each of the basketball shot datasets represented by the graphs below, select the feature maps for which the perceptron algorithm can perfectly classify the data. Each data point is in the form $(x_1, x_2) = (\text{spin}, \text{height})$, and has some label Y, which is either a + (successful shot) or - (missed shot).



☐ x_1, x_2
■ x_1, x_2, Y
☐ $x_1, x_2, 1$
☐ x_1, x_2, x_1^2
■ $x_1, x_2, x_1^2, 1$



☐ x_1, x_2
■ x_1, x_2, Y
■ $x_1, x_2, 1$
☐ x_1, x_2, x_1^2

■ $x_1, x_2, x_1^2, 1$

Having the feature Y just tells us the correct label, so any feature set that has it automatically works, but also is not at all interesting.

First graph: is not linearly separable, disqualifying the linear-only options (1 and 3). We can see that drawing a parabola (in x_1) can separate the points, but it needs to pass below the origin, so we need a parabolic feature set with a bias, which is the last option.

Second graph: is linearly separable, but the $+$ and the $-$ are on the same line as the origin, so we need a bias. A parabola in x_1 can't separate the points (option 4) but in x_2, can (option 5). The last option (5) doesn't strictly need the bias but extra features can always be ignored.

Q5. [11 pts] MDPs: Easy MDPeasy

Lauren can be on any of floors F_1, F_2, or F_3 in her house, where F_1 is the lowest floor and F_3 is the highest floor.

Lauren can take the action **up**, which moves her from floor F_i to floor F_{i+1}. If she's originally at floor F_3 and goes up, she just stays at floor F_3. Similarly, she can also take the action **down**, which moves her from floor F_i to floor F_{i-1}. If she's originally at floor F_1 and goes down, she just stays at floor F_1.

Both of these actions work correctly with 100% probability.

Also, at any floor, Lauren can choose to take the **call** action, in which she makes a phone call on that floor. With this action, one of three events can occur:

- Her call gets accepted, which lands her in the terminal state "accepted".
- Her call gets rejected, which lands her in the terminal state "rejected".
- Her phone glitches and she isn't able to make the phone call, which causes her to stay in her current state.

Lauren decides to model this an a Markov Decision Process.

The reward is 0 for every action, except for actions that go from a floor state to one of the terminal states: going to "accepted" yields a reward of 1, and going to "rejected" yields a reward of -1. Assume we use a discount factor of $\gamma = 1$.

For the rest of the problem, the below table will represent the transition probabilities for a **call** action in our MDP, with each cell representing $T(s, call, s')$ for a given s' in the column headers and s in the row headers.

	s' = accepted	s' = s	s' = rejected
$s = F_3$	0.3	0.4	0.3
$s = F_2$	0.2	0.5	0.3
$s = F_1$	0.1	0.7	0.2

(a) [2 pts] Lauren wants to go to a floor and choose the call action repeatedly on that floor until she reaches a terminal state. Which floor would yield the highest expected reward from doing this?

○ F_1 ○ F_2 ● F_3

The highest expected reward: 0

We set up the following equations:

$$\mathbb{E}[F_3] = 0.3 + 0.4\mathbb{E}[F_3] - 0.3$$
$$\mathbb{E}[F_2] = 0.2 + 0.5\mathbb{E}[F_2] - 0.3$$
$$\mathbb{E}[F_1] = 0.1 + 0.7\mathbb{E}[F_1] - 0.2$$

Solving each equation and finding the max expected value yields $\mathbb{E}[F_3] = 0$ as the maximum expected reward.

Since the transition probabilities for the "call" action don't change at all, the optimal policy will still be to go to

(b) [3 pts] Select all true statements about this MDP.

■ Value iteration will converge to the optimal values after calculating $V_1(s)$ for all states s (assuming we start at $V_0(s)$).

☐ Changing the discount factor γ will affect the optimal values at each state for this MDP.

☐ There exists a unique optimal policy for this MDP.

■ If we introduced a negative reward when moving between floors, this could potentially change the optimal values for some states.

◯ None of the above.

The first statement is true. Performing one iteration of value iteration after $V_0(s)$ will result int he same values, meaning we have converged to the optimal values for that state.

The second statement is false because of the fact that our discount rate is being multiplied by $V_{k-1}(s')$ in the value iteration algorithm, and since $V_0(s) = 0$, the value of γ does not matter.

The third statement is false, as the optimal values are 0 for all states, which can be the result of going either up or down from every state.

The fourth option is true because this could mess with the idea that we can move freely between floors as many times as want. If we can't do this, then we should have to explore the tradeoffs between moving between floors and calling from a floor you're already on, which could mess with the optimal values for some states.

(c) [4 pts] Suppose we change the reward of moving into the "accepted" state from a non-terminal state to 10, we change the reward of moving into the "rejected" state from a non-terminal state to -10, and all other actions now yield a reward of -1 (including "up" and "down" actions between floors, and "call" actions that result in staying in the same floor).

Perform two iterations of value iteration for the three floor states, and fill the missing values in the below table. Note that the two terminal states are left out here because they always have a value of 0, and also that $\gamma = 1$.

For convenience, here is the same "call" transition table from the last page. Note that "up" and "down" actions still work correctly all the time.

	s' = accepted	s' = s	s' = rejected
$s = F_3$	0.3	0.4	0.3
$s = F_2$	0.2	0.5	0.3
$s = F_1$	0.1	0.7	0.2

States	F_1	F_2	F_3
$V_0(s)$	0	0	0
$V_1(s)$	-1	-1	-0.4

For $V_1(F_1)$, we see that using the value iteration equation, going up or down will yield a value of -1, as well as making a call on that floor. In fact, this is the case for all floors, so the whole row will be equal to -1.

For $V_2(F_1)$, we first see that going up or down will yield a value of -2 now (since we use the values in $V_1(s)$ now). However, making a call at F_1 will yield a value of $0.1 \cdot 10 - 0.2 \cdot 10 + 0.7 \cdot (0 + (-1))) = -1.7$, and since this is the max value out of all three possible actions, this is the value for $V_2(F_1)$. Similar calculations are made for $V_2(F_2)$ and $V_2(F_3)$.

This subpart is independent of the previous subparts.

(d) [2 pts] True or false: Computing the optimal values of states, given the output of policy iteration, is easier than computing the optimal values without policy iteration.

● True ○ False

True or false: Extracting a policy given optimal values is easier than extracting a policy given optimal Q-values.

○ True ● False

First: we can run policy evaluation and don't have to consider non-optimal actions. This is better than having to compute values while not knowing the optimal actions; all algorithms for calculating values are slower than just policy evaluation.

Second: we can take the $\arg\max_a Q(s, a)$ for each state s, which is $O(|S||A|)$ amount of work. If we only have V values, there's no direct way of getting what actions made that happen, we have to use one of the more complex algorithms to either get Q values from this or do some kind of policy iteration.

Q6. [16 pts] RL: Weight a Minute!

Consider a state space with states $X_1, X_2, X_3, \ldots, X_n$. In any state, one can take the action to move to any **other** state. For instance, from state X_1, you can move to any of states $X_2, X_3, \ldots, X_n$, but there is no action that allows you to stay in state X_1.

(a) [2 pts] Consider running Q-learning on this problem. How many Q-values do we need in order to represent this problem? Your answer should be an expression in terms of n.

$n(n-1)$

Total Q-values $= |S| \times |A|$.

(b) [2 pts] Compared to Q-learning, select all of the following reinforcement learning (RL) algorithms that have a **greater** space requirement for this problem. *Hint:* the space requirement for Q-learning is $|S| \times |A|$.

- [x] Model-based Learning
- [ ] Direct Evaluation
- [ ] Temporal Difference (TD) Learning
- ○ None of the above.

Model-based learning requires a space complexity of $|S|^2|A|$ due to needing all the transitions from one state to any other state. Direct evaluation and TD-learning, on the other hand, only have a space requirement of $|S|$, as only $V(s)$ needs to be stored for all states s.

Now, consider an arbitrary MDP, with unknown transition/reward models. We observe an agent acting according to policy π.

In standard TD learning, we would process each sample one at a time:

$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[r_i + \gamma V^\pi(s_i')\right]$$

Danial wants to instead process three samples (all starting at the same state) at a time in one update, rather than in three separate updates. Furthermore, he wants to process these three samples at once using a **weighted average**.

The sample yielding the highest reward out of the three receives weight w_1, the sample with the second highest reward receives weight w_2, and the sample with the lowest reward receives weight w_3, where ties are broken randomly. You can assume $w_1 > w_2 > w_3 > 0$, and $w_1 + w_2 + w_3 = 1$.

(c) [4 pts] Select Danial's modified update equation.

Sample (s_1, a_1, s_1', r_1) yields the highest reward.

Sample (s_2, a_2, s_2', r_2) yields the second highest reward.

Sample (s_3, a_3, s_3', r_3) yields the lowest reward.

$$V^\pi(s) \leftarrow (1-\alpha) \cdot \textbf{(i)} \cdot \textbf{(ii)} \; + \alpha \cdot \textbf{(iii)}\,(\textbf{(iv)} \cdot [\textbf{(v)} + \textbf{(vi)}])$$

(i)	○ 0	● 1	○ 3	○ $(1-\alpha)^2$
(ii)	○ $\sum_{i=1}^3 V^\pi(s)$	● $V^\pi(s)$	○ $w_1w_2w_3V^\pi(s)$	○ $\prod_{i=1}^3 V^\pi(s)$
(iii)	○ $\max_{1\le i\le 3}$	○ $\min_{1\le i\le 3}$	● $\sum_{i=1}^3$	○ $\prod_{i=1}^3$
(iv)	○ αw_i	● w_i	○ $(1-w_i)$	○ $\frac{1}{w_i}$
(v)	● r_i	○ γr_i	○ αr_i	○ $3r_i$
(vi)	○ $w_iV^\pi(s_i')$	○ $V^\pi(s_i')$	○ $\gamma V^\pi(s_i)$	● $\gamma V^\pi(s_i')$

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \sum_{i=1}^{3} w_i \left[r_i + \gamma V^{\pi}(s_i') \right]$$

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

- [x] More recent sets of three samples will influence the value of that state more than older sets.
- [ ] We will learn the optimal policy from this new updated equation.
- [x] If $w_1 > w_2 \geq w_3 \geq 0$ instead of $w_1 > w_2 > w_3 > 0$, we may not converge to the true value of $V^{\pi}(s)$.
- [x] Slowly shrinking the learning rate α overtime will help $V^{\pi}(s)$ stabilize its value.
- ○ None of the above.

The first option is true because due to the learning rate α, the older sets of three samples will have exponentially less weight than the newer sets.

The second option is incorrect because TD-learning is not used to find the optimal policy, and neither will this new update equation.

The third option is true because if, say, $w_1 = 1$ and $w_2 = w_3 = 0$, then this update equation essentially becomes the standard update equation for TD-learning, which we know converges already.

The fourth option is correct because decreasing the learning rate puts less weight on new samples, allowing our $V(s)$ value to converge quickly to our true value.

(e) [3 pts] Danial claims that this new update equation is guaranteed to converge to the true state values, just like the normal TD learning update equation. Curtis, however, claims that this is not necessarily true. Who is correct, and why? Explain your answer in three sentences or fewer.

○ Danial is correct ● Curtis is correct

Curtis is correct because assigning the highest weight to the largest reward value of the three samples isn't guaranteed to help you converge. Consider the case where your two possible rewards from a state are 1 and 999, and the reward of 1 occurs $\frac{2}{3}$ of the time. If our weights were, say $w_1 = 0.95, w_2 = 0.3, w_3 = 0.2$, then even though we would usually sets of three samples with rewards $1, 1, 999$, our weights would unevenly weight the largest sample by a lot more, causing us to never converge to the true value for that state.

(f) [2 pts] In model-based learning, the approximations for our transition functions $\hat{T}(s, a, s')$ are calculated using the _____.

- ○ learning rate
- ○ weighted average
- ○ maximum a posteriori
- ● maximum likelihood estimate
- ○ None of the above.

This is known as the maximum likelihood estimate, because we are finding the probability ($\hat{T}(s, a, s')$) that maximizes the chance that we observe our evidence.

Q7. [12 pts] ML: N&Ns

You're building a neural network to determine whether candies are M&Ms or Skittles, where M&Ms are represented by a 0, and Skittles are represented by a 1.

Each data point is represented as a vector x consisting of the features [candy diameter, is blue, is brown].

The neural net has the following structure:

$$h_1 = \sigma(xW_1 + b_1)$$
$$\hat{y} = \text{ReLU}(h_1 W_2 + b_2)$$

Finally, this value $\hat{y}$ is put into a loss function $\mathcal{L}(\hat{y}) = \|\hat{y} - y^*\|_2$, meaning after you subtract the vector y^* from the vector $\hat{y}$, you take the square root of the sum of squares of that resulting vector. y^* is the vector holding the true labels for your input.

(a) [3 pts] You decide to train the model with batch size 4.

Given the already filled-in dimensions, fill in the missing dimensions:

x : [4] × [3]

W_1 : [3] × [128]

h_1 : [4] × [128]

W_2 : [128] × [1]

$\hat{y}$: [4] × [1]

A single x input needs to contain all of the data, so it needs to have 4 x vectors, each 3 features long.

By definition of matrix multiplication, if $M \times N$ is valid and M is $a \times b$ and N is $c \times d$, we must have $b = c$ (because matrix multiplication is just an array of results when looking at the dot product of each possibly pair of row of M and column of N, which in turn is just a similarity check). The result of the matrix multiplication is therefore $a \times d$.

Tracking the dimensions starting from x yields the rest of the numbers. σ and ReLU are an element-wise functions, so they doesn't affect the dimensions.

This was also a part of doing Project 5.

(b) [3 pts] When training this model, the training loss remains high, and so does the test loss. Which of the following could help with this problem?

- [x] Change the column dimension of W_1 from 128 to 256 (and modify other vector/matrix dimensions to line up with this).
- [x] Increase the size of the training set.
- [ ] Increase the size of the test set.
- [ ] Increase the batch size to 10.
- [x] Add more features to x.
- ○ None of the above.

Since both training and test loss are high, we are not fitting the data. This means that either our model is not complex enough to capture the relationship (options 1 and 5), or we don't have enough data for the algorithm to work well (option 2).

Increasing the test set means we have less data in the training set (option 3). Changing the batch size (option 4) is usually a hardware-targeted optimization within a reasonable batch size selection; changing batch size from 4 to 10 will not affect the behavior of the optimization algorithm learning the weights.

(c) [4 pts] For each of the partial derivatives below, express these partial derivatives as a chain of other derivatives in the way that backpropagation would do. Do **not** compute the actual derivatives. In your partial derivatives, you may only use the variables $\mathcal{L}, b_1, b_2, h_1$, and $\hat{y}$.

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_2}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial h_1} \frac{\partial h_1}{\partial b_1}$$

This uses the chain rule, $\frac{\partial f(g(t))}{\partial t} = \frac{\partial f(g)}{\partial g} \times \frac{\partial g(t)}{\partial t}$. We apply this to the neural net formulas at the beginning of the problem.

(d) [2 pts] Bubby proposes that changing the final activation function from ReLU to sigmoid (which means $\hat{y} = \sigma(h_3)$ is the new output step) will be more suitable for our task of classifying the M&Ms. Is he correct?

- ● Yes, because the sigmoid function squashes our range to be between 0 and 1, allowing our output to be closer to the label value (which is 0 or 1).
- ○ Yes, because $\text{ReLU}(x) = \max(0, x)$ does not have a defined derivative at $x = 0$.
- ○ No, because the sigmoid function takes much longer for a computer to run than the ReLU function.
- ○ No, because the sigmoid function is undefined for certain inputs.

For binary classification tasks, the sigmoid function can represent probabilities of our input belonging to a certain class, deeming it much more appropriate for our M&M classification task.

Congratulations on completing the final! Feel free to use this space to doodle or to let us know about anything!