UC Berkeley – Computer Science
CS188: Introduction to Artificial Intelligence
Josh Hug and Adam Janin

Final, Fall 2016

This test has **10** questions worth a total of **100** points, to be completed in 170 minutes. The exam is closed book, except that you are allowed to use three two-sided hand written cheat sheets. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Write the statement out below in the blank provided and sign. You may do this before the exam begins. Any plagiarism, no matter how minor, will result in an F.

"I have neither given nor received any assistance in the taking of this exam."

_____

_____

Signature: _____

Name: _____ **Your EdX Login:** _____

SID: _____ **Name of person to left:**_____

Exam Room: _____ **Name of person to right:** _____

Primary TA: _____

Tips:

- ◯ indicates that only one circle should be filled in.
- ▫ indicates that more than one box may be filled in.
- Be sure to fill in the ◯ and ▫ boxes completely and erase fully if you change your answer.
- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- There are a lot of problems on this exam. Work through the ones with which you are comfortable first. **Do not get overly captivated by interesting problems or complex corner cases you're not sure about.**
- Not all information provided in a problem may be useful.
- **There are some problems on this exam with a slow brute force approach and a faster, clever approach. Think before you start calculating with lots of numbers!**
- Write the last four digits of your SID on each page in case pages get shuffled during scanning.

Problem	1	2	3	4	5	6	7	8	9	10
Points	7	6	8.5	8.5	17	14	9	9	10.5	10.5

Optional. Mark along the line to show your feelings on the spectrum between :(and ☺.

Before exam: [:(_____ ☺].

After exam: [:(_____ ☺].

1. (7 pts) Naive Bayes - Josh Detector

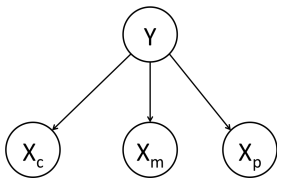
Suppose we want to find if Josh is in this office ($Y = 1$) or not ($Y = 0$) by analyzing the measurements of the following three sensors deployed in his office:

- a chair pressure sensor informs you if someone is on his chair ($X_c = 1$) or not ($X_c = 0$)
- a motion detector tells you if someone moves in the room ($X_m = 1$) or not ($X_m = 0$)
- a power meter indicates if someone consumes electricity in the room ($X_p = 1$) or not ($X_p = 0$)

Each sensor above only reveals partial information about Josh's presence. For instance, if Josh is in his office but not sitting on the chair, then the chair pressure sensor can not reliably indicate Josh's presence. Suppose we have some historical data logs of sensor measurements and Josh's presence status:

Day	1	2	3	4	5	6	7	8	9	10
X_c	0	0	0	0	0	1	0	1	0	1
X_m	0	0	0	0	0	0	0	0	1	1
X_p	1	1	0	0	1	1	1	1	0	0
Y	0	0	0	0	1	1	1	1	1	1

i. (2 pts) Let us use Naive Bayes to model the sensor measurements and Josh's presence status as shown in the Bayes Net model below. Fill in the maximum likelihood estimate (MLE) of each entry in the probability tables.



(B1)

Y	$P(Y)$
0	2/5
1	3/5

Y	X_c	$P(X_c\|Y)$
0	0	1
0	1	0
1	0	1/2
1	1	1/2

Y	X_m	$P(X_m\|Y)$
0	0	1
0	1	0
1	0	2/3
1	1	1/3

Y	X_p	$P(X_p\|Y)$
0	0	1/2
0	1	1/2
1	0	1/3
1	1	2/3

ii. (1 pt) Suppose we get a new set of sensor observations: $X_c = 0,\ X_m = 0,\ X_p = 1$. According to the probability tables' maximum likelihood estimate, is Josh more likely to be present or absent?

○ Present ● Absent ○ Equally Likely ○ Not Enough Information

Explanation: We compare:

$P(Y = 0, X_c = 0, X_m = 0, X_p = 1) = 2/5 * 1 * 1 * 1/2 = 1/5$

$P(Y = 1, X_c = 0, X_m = 0, X_p = 1) = 3/5 * 2/3 * 2/3 * 1/3 = 4/45$

The former is larger, so we predict that Josh is more likely to be absent.

Notice that this is true even though $Y = 0, X_c = 0, X_m = 0, X_p = 1$ and $Y = 1, X_c = 0, X_m = 0, X_p = 1$ appeared equally frequently in our samples.

iii. (1 pt) Suppose that instead of maximum likelihood, we now use Laplace Smoothing to estimate each entry in the following probability tables. Assume the prior strength $k = 1$.

Y	$P(Y)$
0	5/12
1	7/12

Y	X_c	$P(X_c \mid Y)$
0	0	5/6
0	1	1/6
1	0	1/2
1	1	1/2

iv. (1 pt) What happens to our CPTs as we increase k, i.e. what sort of distribution do we get in the large k limit?

Solution: In the large k limit, we get uniform distributions in each CPT.

v. (1 pt) Suppose we want to generate exactly two samples from the distribution P(Y, X_c, X_m, X_p) given by the CPTs you computed from MLE estimates in part i. Suppose we run Gibbs sampling for a long time so that the samples are indeed drawn from the correct distribution, and get the resulting sample GS1: (Y = 1, X_c = 0, X_m = 1, X_p = 0). Suppose we then generate a new sample GS2 by resampling from P(X_p | Y = 1, X_c = 0, X_m = 1). What are the different possible values for GS2, and how likely is each?

Possible Value #1 for GS2: Y = __1__, X_c = __0__, X_m = __1__, X_p = __0__, probability: ___1/3___

Possible Value #2 for GS2: Y = __1__, X_c = __0__, X_m = __1__, X_p = __1__, probability: ___2/3___

Explanation: The probabilities utilize $P(X_p \mid Y, X_c, X_m) = P(X_p \mid Y)$.

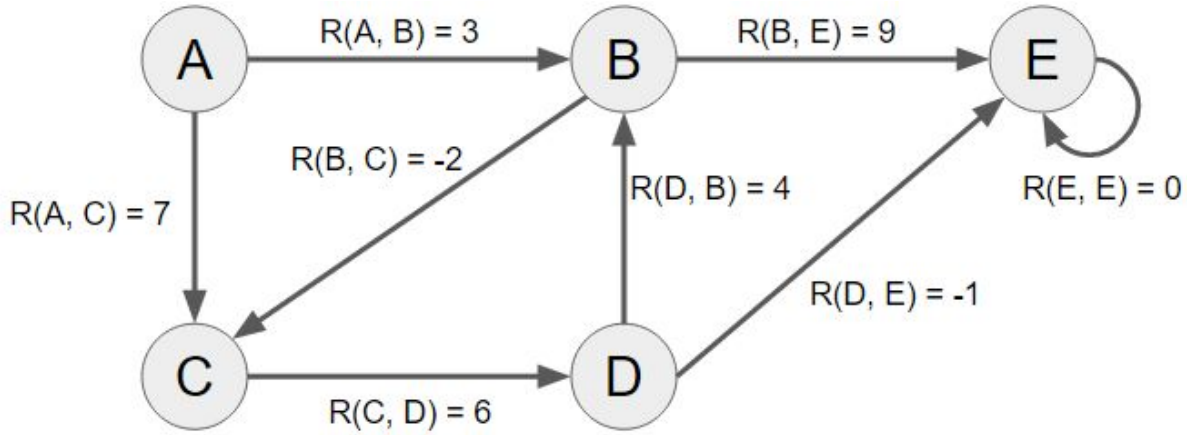
vi. (1 pt) Suppose we instead use prior sampling to generate samples PS1 and PS2. Which samples are of higher quality, the two samples generated by Gibbs Sampling or the two samples generated by Prior Sampling? We have intentionally left "higher quality" undefined. Use your best judgment. For partial credit in the event of a wrong answer, very briefly explain your answer below.

● GS1 and GS2 are higher quality ● PS1 and PS2 are higher quality ○ They are the same quality

Intended solution: When we have many samples, prior sampling and Gibbs sampling perform equally well. However, when there are only a few samples, prior sampling is better than Gibbs sampling, because the samples generated by Gibbs sampling are correlated with one another (each successive sample only differs by a single variable). For example, if there are only two samples for four variables, Gibbs sampling will only see one value for three of the variables, while prior sampling avoids this correlation bias.
Alternate interpretation: Some students didn't realize that the samples are for the same query as in the previous part (v) (where no is evidence), and considered the possibility that we have a query that *does* have evidence. In that case, Gibbs sampling is better than prior sampling, since the latter generates samples without paying attention to the evidence.

2. MDPs [6 pts]

Consider the following Markov Decision Process. Unlike most MDP models where actions have many potential outcomes of varying probability, assume that the transitions are **deterministic**, i.e. occur with T(s, a, s')=100% probability as shown in the graph below. The reward for each transition is displayed adjacent to each edge:



i. (3 pts) Compute the following quantities, assuming a discount factor of 1. Recall that $V_k(s)$ represents the expected utility of the state s under the optimal policy assuming the game ends in k more time steps, and V*(s) is the expected utility starting in state s if we play optimally forever.

$V_2(A) = R(A,C) + R(C,D) = 13$ $\quad V_5(A) = R(A,C) + R(C,D) + R(D,B) + R(B,E) + R(E,E) = 26$

$V^*(A) = \infty$ from taking the cycle $C - D - B$ repeatedly.

$V_2(B) = R(B,E) + R(E,E) = 9$ $\quad V_5(B) = R(B,C) + R(C,D) + R(D,B) + R(B,E) + R(E,E) = 17$

$V^*(B) = \infty$ from taking the cycle $B - C - D$ repeatedly.

ii. (1 pt) Consider using feature-based Q-learning to learn what the optimal policy should be. Suppose we use the feature f_1(s, a) equal to the smallest number of edges from s to E, and feature f_2(s, a) equal to the smallest number of edges from a's target to E. For example f_1(A, →B) is 2, and f_2(C, →D) is 1.

Let the current weight for these features be [3, 2]. What is the current prediction for Q(C, →D) given these weights for these two features?

Q(C, →D) = $f_1(C, \rightarrow D) * w_1 + f_2(C, \rightarrow D) * w_2 = 2 * 3 + 1 * 2 = 8$

iii. (2 pts) Suppose we update w based on a single observation of a transition from C to D, assuming $\alpha = 0.5$. What is w after updating based on this observation?

w = [6, 3.5]

Explanation:

The difference between the actual reward and the predicted reward is:

$\Delta = (R(C, \rightarrow D) + \max_{a'} Q(D, a')) - Q(C, \rightarrow D) = (6 + 5) - 8 = 3$.

We thus updates our new weights to be:

$w + \Delta * \alpha * f = [3, 2] + 3 * 0.5 * [2, 1] = [6, 3.5]$

3. Probability (8.5 pts)

Consider the following CPT, **to be used for part i only!**

A	B	C	P(A, B, C)
+a	+b	+c	0.15
+a	+b	-c	0.2
+a	-b	+c	0.3
+a	-b	-c	0.1
-a	+b	+c	0.15
-a	+b	-c	0
-a	-b	+c	0.05
-a	-b	-c	0.05

i. (2 pts) Compute the following expressions for the table above. Write each answer as a single numerical value. It is ok to leave your answer as a fraction of decimal numbers.

- P(+a | -c) = $\frac{P(+a,-c)}{P(-c)} = \frac{0.2+0.1}{0.2+0.1+0.+0.05} = 6/7$
- P(+b | -a, -c) = $\frac{P(-a,+b,-c)}{P(-a,-c)} = \frac{0}{0.+0.05} = 0$
- P(-a | +b, +c) = $\frac{P(-a,+b,+c)}{P(+b,+c)} = \frac{0.15}{0.15.+0.15} = 1/2$
- P(-b, +c | -a) = $\frac{P(-a,-b,+c)}{P(-a)} = \frac{0.05}{0.15+0+0.05+0.05} = 1/5$

ii. (1 pt) Let A, B, and C be binary random variables. Select all of the following CPTs that are guaranteed to have 4 rows **for ANY distribution**, not the table above.

M P(+a | B, C) M P(B, C | +a) M P(B | A) ☐ P(+a, C | +b)

Explanation: Since we have binary random variables, the number of rows that a distribution has is equal to 2 to (the number of unspecified variables in the distribution).

iii. (2 pts) Select all of the following that are guaranteed to equal 1 for ANY distribution, not the table above.

M P(+a) + P(-a)

M P(+a | -c)P(-c) / (P(+a, -c, +b) + P(+a, -c, -b))

☐ P(+a | +b)P(+b) + P(-a | -b)P(-b)

☐ P(-c | -a)P(-a) + P(-c | +a)P(+a)

Explanation:

The first is true by the law of total probability.

The second's numerator is equal to $P(+a,-c)$ by the product rule, and the denominator is equal to summing B to also get $P(+a,-c)$.

The third is equal to $P(+a,+b)+P(-a,-b)$, which is not simplifiable.

The fourth is equal to $P(-a,-c)+P(+a,-c)=P(-c)$.

For parts iv - vi: It's winter break, and you're staying at Sherdil's Casino. Every morning, you receive (as a gift) a random ***odd*** number of gambling chips, C, in the range 1 to k, where k is some odd integer constant, according to the distribution:

$$P(C = c) = \begin{cases} 2/(k+1), & c \text{ is odd and } c \in [1, k] \\ 0, & \text{otherwise} \end{cases}$$

Assume the number of chips you receive on a particular day is independent of the number of chips you received on any and all prior days.

iv. (1 pt) Let the number of chips you receive on days 1, 2, and 3 be C1, C2, and C3 respectively. Is C3 conditionally independent of C1 given C2? (Answer yes or no. No explanation is needed.)

⬩ Yes ⬩ No

Explanation: Depends on if we know k.

If we don't know k, then the answer is no because $C1$ gives us information about k, which $C3$ is dependent on.

If we know k, then the answer is yes.

v. (1 pt) Suppose k is 5. On average, how many chips can you expect to receive per day?

E[# chips] = 3

Explanation: $E[C] = 1/3 * 1 + 1/3 * 3 + 1/3 * 5 = 3$.

vi. (1.5 pts) Suppose you stay for five days and receive the following chip amounts: {5, 11, 1, 9, 9}. Based on your five samples, compute the maximum likelihood estimate for k.

MLE of k = 11

Explanation:

Intuitively, as long as k is at least 11, the smaller the k the larger the probability of getting any particular outcome.

Formally, the log likelihood of the outcome is:

$ll(k|C) = P(C|k) = \log \prod \frac{2}{k+1} = \sum(\log 2 - \log(k+1)) = 5\log 2 - 5\log(k+1)$ if $k \geq 11$ else 0

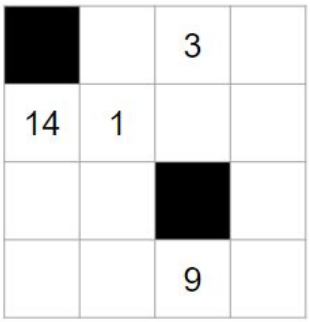
This function is strictly decreasing as k increases, so $k = 11$ is the maximum likelihood estimate.

Common Mistakes:

- The question says that k is an odd integer. Any answer that is not an odd integer is not a plausible answer; this can be a good sanity check on the answer.
- The most common incorrect answer was 13. This is presumably from using a technique like method of moments (i.e. finding k such that $E[k]$ is equal to the average of the sample), since if $k = 13$, then $E[k] = 7$. Method of moments sometimes, but *does not always*, find the maximum likelihood estimate.

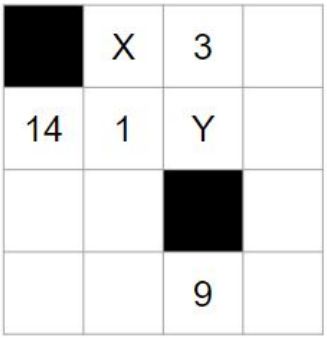
4. CSPs (8.5 pts)

In the game of Hidato, we start with a partially filled grid with N blanks, and our goal is to fill in all the blanks with integers 1 through N such that every number is unique, and every number q (except N) appears adjacent to its successor q + 1, with diagonal moves allowed. For example, if we start from the figure on the left, we can get the solution on the right. Black squares act as obstacles and never receive numbers.



We can formulate this as a CSP, where the domain of the variables is {1, …, N}, where N is the number of non-black (i.e. initially blank) squares.

i. (2.5 pts) If we start from the figure below with 4 variables assigned (to 1, 3, 9, and 14), and run AC-3 (our arc-consistency algorithm), what values remain in the domain of the variable marked with an X? Assume that all 10 blank squares (including X and Y) are unassigned.



Arc-consistency involves processing of binary arcs only, so we must start by deciding what binary arcs exist. Our original intention was that you would derive and apply the following two constraints:

- NO_EQUAL_SQUARES (NES): There are binary arcs between all squares which specify that they cannot be equal (and a separate set of constraints that require numbers to be in a sequence)
- NO_DISTANT_SUCCESSORS (NDS): There are binary arcs between non-adjacent squares that specify that they cannot be one away from each other.

In addition, you can also add a 3rd constraint that is a more powerful version of NDS:

- RADIAL_DIFFERENCE_TEST (RDT): There are binary arcs between every pair of states such that if they have values Xi and Xj respectively then |Xi - Xj| < distance between those arcs. It it redundant with NDS as far as solutions go, but it makes arc consistency more powerful.

However: it is also possible to cast the Hidato problem without any binary constraints, in which case arc-consistency won't do anything at all. For example, consider the following multi-variable constraints:

- ALL_DISTINCT: A single N-ary constraint that ensures all variables are distinct.
- ALL_PRESENT: A single N-ary constraint that says that for every X_i, one of them is i.
- NEIGHBOR_IS_ADJACENT: For each node, its larger successor is one of the neighbors. This is a Q-ary constraint, where Q-1 is the number of neighbors.

In retrospect, we should have asked you to explicitly enumerate your binary constraints, but we did not, so we gave you credit for ANY valid answer, even if you did not write out your binary constraints.

Assuming no binary constraints: The right answer is: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}

If we apply arc-consistency considering only the arcs from X to 1, 3, 9, and 14, then:

- NES eliminates: {1, 3, 9, 14}
- NDS eliminates: {8, 10}
- RDT eliminates: {7, 8, 10, 11}

Completing the entire AC-3 algorithm yields no further eliminations from X. So any answer that includes {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14} minus any of the entire sets above is eligible for full credit.

Sample correct answers:
Domain of X (assuming no binary constraints): __{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}___
Domain of X (assuming NO_EQUAL_SQUARES): ___{2, 4, 5, 6, 7, 8, 10, 11, 12, 13}____
Domain of X (assuming NES and NDS): _____{2, 4, 5, 6, 7, 11, 12, 13}_____
Domain of X (assuming NES and RDT): _____{2, 4, 5, 6, 12, 13}_____

ii. (2.5 pts) If we assign X=2, and re-run AC-3, what will Y's domain be after AC-3 completes? Assume that all 9 blank squares (including Y) other than X are unassigned.

Similar to above, the answer depends on which binary constraints you came up with and applied. In this case, the possibilities are:

- NES eliminates {1, 2, 3, 9, 14}
- NDS eliminates: {8, 10, 13}
- RDT eliminates: (nothing)

As before, any combination that involves eliminating zero or more of the entire sets above is fine.

For example:
Domain of Y (assuming no binary arcs): __{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}___
Domain of Y (assuming NES): ___{4, 5, 6, 7, 8, 10, 11, 12, 13}____
Domain of Y (assuming NES and NDS): _____{4, 5, 6, 7, 11, 12}_____
Domain of Y (assuming NDS only): _____{1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 14}_____

iii. (1 pt) If we ensure arc-consistency after every assignment in any Hidato puzzle, we'll never backtrack.

◯ True ● False
Explanation: Even if we have only binary constraints, the constraint graph is not a tree, so backtracking is possible.

iv. (2.5 pts) Repeat part i, but suppose we run an algorithm that ensures 3-consistency instead of arc-consistency (recall that arc-consistency is just 2-consistency). In this case, what is the domain of X?
Explanation: In the CSP lecture, we strongly implied that you didn't need to know how k-consistency works for k > 2. Oops.

Domain of X: ___**[Beyond scope of course]**_____

5. (17 pts) Potpourri

i. (2.5 pts) You are given a choice between the following:

- A random lottery in which you receive either \$A or \$B uniformly at random.
- An event in which you receive \$*k*, where *k* is a positive real-valued constant.

Your utility function for money is $U(d) = d^2$ where $U(d)$ is the utility corresponding to d dollars. For what values of *k* should you take the lottery? Give your answer as an interval or a union of intervals as appropriate, e.g., $[12, \infty]$ or $[5, 9] \cup [7, \infty)$. You may handle ties however you wish. Assume that A, B, and k are non-negative.

$k \in [0, \sqrt{(A^2 + B^2)/2}]$

Explanation: We value the lottery at $U(L) = (A^2 + B^2)/2$. Thus, we will take the lottery if $U(k) = k^2$ is less than or equal to this.

For parts ii and iii: Let X, Y, and Z be independent random variables with the following distributions:

X	$P(X)$		Y	$P(Y)$		Z	$P(Z)$
3	2/3		2	1/2		-2	1/4
-6	1/3		10	1/2		6	3/4

You can take either action 1 or action 2.

- Action 1 has utility $U_1(X, Y, Z) = 10 * X - 3 * Y * Z$
- Action 2 has utility $U_2(X, Y, Z) = 10 * X + 5 * Y - 5 * Y * Z$

ii. (3.5 pts) Calculate the maximum expected utility. Hint, to avoid tedious calculations, recall from your prior coursework that E[A*B] = E[A]*E[B], so long as A and B are independent variables, and that E[X+Y] = E[X] + E[Y] for any random variables X and Y.

$MEU(\{\}) = -72$

Explanation:

We first compute $E[X] = 0, E[Y] = 6, E[Z] = 4$. Since Y and Z are independent, $E[YZ] = E[Y]E[Z] = 24$. We can then calculate $E[U_1] = 10E[X] - 3E[YZ] = -72$ and $E[U_2] = 10E[X] + 5E[Y] - 5E[YZ] = -90$, so we should take the first utility value, getting $MEU(\{\}) = -72$.

iii. (2 pts) Which variable has the lowest VPI? Justify your answer either in words or numerically. Please restrict any answers to fewer than 20 words.

* X ◯ Y ◯ Z

Explanation: $VPI(X) = 0$ because the way X affects the utility is independent of the action. Since VPIs are nonnegative, this variable must have the lowest VPI (or be tied for this property, but we can verify that the VPIs of the other two are positive).

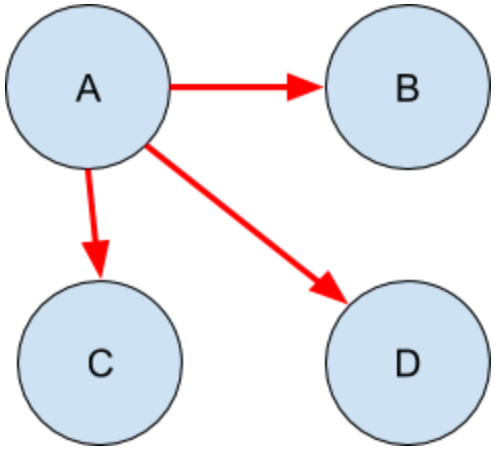
Specifically, one way of seeing this is to note that $U_1(X, Y, Z) - U_2(X, Y, Z) = 2YZ - 5Y$, which does not depend on X.

Common Mistakes:

- Saying that $VPI(X) = 0$ because $E[X] = 0$ or that $E[X] < E[Y], E[Z]$ so X affects the utility the least. Even if we changed the possible values of X to be 100 and 200, that wouldn't change the answer.
- Saying that X and the utility are independent. X definitely contributes to the utility.

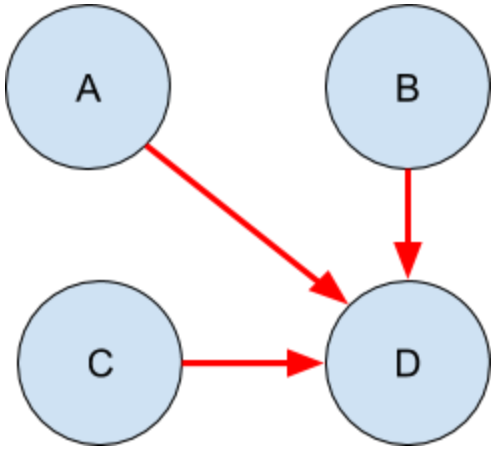
iv. (3 pts) You are given a Bayes Net with the four nodes below. The "number of rows" in a Bayes Net is the total number of rows in all CPTs (including the two-row tables for variables with no incoming edges). Assume all variables are binary.

Draw **3** edges such that the Bayes Net is valid, and the number of rows in the Bayes Net is **smaller** than the number of rows in the full joint table for all four variables.



Solution: (The above is an example solution: any solution where no variable has multiple parents would work.)
Explanation: The number of rows in the full joint table is 2^4=16. We are thus looking for a Bayes net with fewer than 16 total rows.
We note that if each edge points to a different variable, then the number of total rows is 14 (2 for the sole unconditioned variable, and 4 for each of the other CPTs).
To see that this is the necessary condition (when $n = 4$), we prove that any solution with a variable with two parents must have at least 16 rows. Specifically, a variable with three parents will have 16 rows, which is too many. If there is a variable with two parents, it will have 8 rows, but there will be another variable with one parent and thus 4 rows; combined with the two variables with no parents and thus 2 rows each, we have a total of 16 rows.

Draw **3** edges such that the Bayes Net is valid, and the number of rows in the Bayes Net is **larger** than the full joint table.



Solution: (The above is an example solution: any solution where all three edges are pointing to the same variable would work.)
Explanation: The solution for this part continues from the same reasoning as the prior: the only way to get the total number of rows to be more than 16 is if a variable has three parents.

Common Mistakes (for both parts):

- Drawing something that isn't a Bayes net. This includes having undirected edges as well as having cycles in the graph (self-loops included).
- Using more or less than three edges.

v. (2 pts) Suppose you're trying to choose a move in a two-player adversarial game by using **minimax**. Suppose we use an evaluation function *f(s)* to evaluate the quality of a state during depth limited search. From the list of possibilities for *g(s)* below, which are guaranteed to result in the **exact same action** as if we used *f(s)* instead?

You may assume that all else is equal (including any tie-breaking scheme) except the evaluation functions. You may assume that *f(s)* is always positive.

- ■ $g(s) = f(s) + 10$
- ■ $g(s) = 5f(s)$
- ■ $g(s) = f(s)^2$
- ■ $g(s) = \log f(s)$

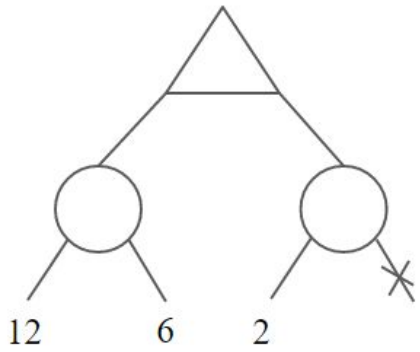
vi. (2 pts) Now suppose you're using **expectimax** instead of minimax to choose a move. From the list of possibilities for *g(s)* below, which are guaranteed to result in the **exact same action** as if we used *f(s)* instead?

Again, you may assume everything except the evaluation functions is the same, and that *f(s)* returns a positive value.

- ■ $g(s) = f(s) + 10$
- ■ $g(s) = 5f(s)$
- ☐ $g(s) = f(s)^2$
- ☐ $g(s) = \log f(s)$

Explanation (for parts v + vi):
Minimax does not care about the exact values that the evaluation function outputs: it only compares about their relative sizes. Thus, any monotonic transformation of the evaluation function (as adding constants, multiplying by a positive constant, and the log function are) would keep the same option being chosen by the algorithm.
Expectimax cares about the actual values: the transformation must be linear. Thus, taking the square or the log do not work any more.

vii. (2 pts) Suppose we are running depth 1 expectimax, and know that our evaluation function always returns a value between B and T, where B ≤ T. For what values of B and T can we perform the pruning step as shown below? Assume that all chance outcomes are equally likely.



Solution: $T \leq 16$

Explanation: The value of the left chance node is 9. Max will prune if there is no way of the remaining branch to be high enough to make the right chance node worth taking.

The best that the right chance node could be is $(2 + T)/2$, so we prune if this expression is guaranteed to be at most 9.

Notes:

- Incorporating knowledge that the evaluation function must be at least 12 or at most 2 as bounds for B and T were not marked wrong, but neither is this information relevant for pruning.
- Some students misunderstood the question and wrote answers that tried directly bounding the evaluation function, e.g. $T = 16$. However, if $T = 15$, we also know that pruning can occur.
- It was not a reasonable assumption that the output of the evaluation function must be integers, or that the output must be nonnegative/positive.

6. MDPs and RL for Solving Adversarial Games (14 pts)

Consider the problem of using an MDP to develop a strategy to maximize a player's reward in a game called **The Downward Spiral.** The game starts with both players having an "elevation" of zero. A game consists of N rounds, where each round consists of a turn by player 1, followed by a turn by player 2. After N rounds, the game is over. During each turn:

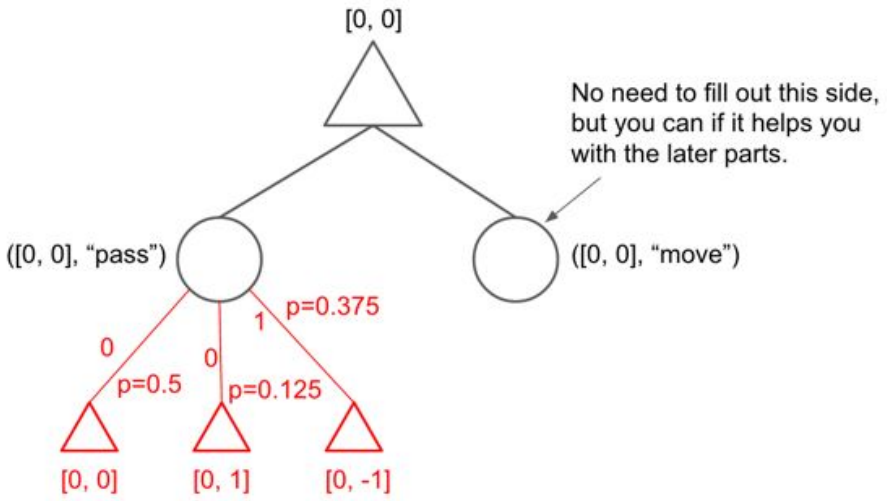
- A player may "pass" or "move".
- If a player "passes", their elevation is unchanged.
- If a player "moves", then their elevation ***increases*** by 1 with probability 0.25, and ***decreases*** by 1 with probability 0.75. Moving only changes the elevation of the active player.

Part 1: Suppose that player 1's utility is equal to the number of times that they end a round with a strictly higher elevation than player 2. One approach is to model the game's state as a pair of values representing the elevation of each player, i.e. S_0 is always equal to [0, 0]. After one round, suppose player 1 passes, and player 2 moves and goes down, then the new state is S_X = [0, -1]. To account for player 1's goal, our model has a reward function such that player 1 earns a reward of 1 point for ending the round with a higher elevation. So in the example above R(S_0, pass, S_X) = 1, since the round ended with player 1 having a higher elevation. Throughout the problem, let the discount factor be 1.

i. (3 pts) Player 1 will use an MDP model to decide what actions to take to maximize their rewards. Fill in the **MDP search tree** below for the game assuming that the chance of heads is 0.25, that the model predicts player 2 will randomly pick pass/flip with p=0.5 and the game lasts **only one round (N = 1)**. **Label transitions with their reward and probability**. Label states with the appropriate pair of values [x, y]. The starting state and Q-states are drawn for you. Only the edges and states below the left q-state ([0, 0], "pass") will be graded.

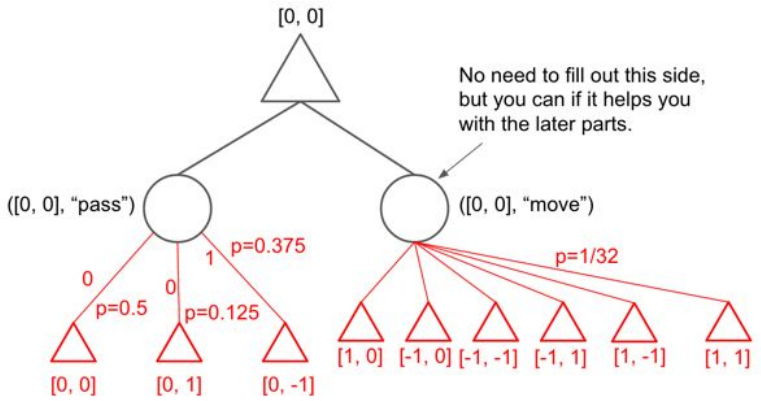
There are a few different reasonable interpretations for this problem. See the appendix to these solutions for alternate possibilities. An MDP search tree consists of decision nodes (which we denoted with upward triangles) and chance nodes (circles). The best answer is as shown below, where there are three possible outcomes:

- Outcome 1: Player 2 picks pass, no reward. Happens with chance 0.5.
- Outcome 2: Player 2 picks move, goes up. No reward. Happens with chance 0.5 * 0.25 = 0.125.
- Outcome 3: Player 2 picks move, goes down. Reward = 1. Happens with chance 0.5 * 0.75 = 0.375.



ii. (1 pt) In the **entire** MDP search tree **from part i** (including the side of the tree that you were not required to draw), what is the exact value of the smallest transition probability?

Solution: The least likely outcome is if both players chose to move and successfully increase their elevation; this happens with probability $1/4 * 1/2 * 1/4 = 1/32$. See below.



iii. (2 pts) **Let N = 2 rounds**. Let Z be the number of transitions in the entire MDP. How many total transitions are there? Give an exact answer. By transition, we mean an edge from a Q-state to a state.

Z = 90

Solution: There are 9 transitions in round 1, as partially demonstrated in part (i). In round 2, there are 9 potential states where player 1 can make an action from (each player's score can range from -1 to 1), and each has 9 possible transitions (for each combination of player 1's and player 2's possible actions), for 81 transitions in round 2.

Some students observed that the transitions in round 1 are redundant with some of the transitions in round 2, and gave an answer of 81; this is the number of unique transitions (s, a, s'), and thus also correct, although the MDP model is no longer a tree, but a graph.

If you had the wrong model for part i, there are other possible answers.

iv. (1 pt) Suppose we want to use **model-based reinforcement learning** to approximate player 2's strategy rather than assuming that they choose randomly. Assume that they have some sort of consistent, but not necessarily deterministic strategy. Assuming there are Z transitions, how many T(s, a, s') values do we need to learn? Again assume that **N = 2**. You can do this problem even if you didn't do part 1-ii. If the problem seems ambiguous, state your assumptions.

Number of T(s, a, s') values to learn = Z

Explanation: We learn one transition probability per transition, which is an edge from a Q-state to a state. Some of you felt that reusing (0, 0) was redundant, and thus we also accepted Z - sqrt(Z) or 72.

Special note: If you used one of the incorrect models, the correct answer is zero. We defined Transitions as between Q-states and regular states, and in alternate models (#1 and #2), these transitions all have probability 0.75 or 0.25, defined by the rules of the game, and thus there is nothing to learn.

Special note #2: Some of you tried to be clever and count the number of states and possible transitions between them. Others tried to note that some transition probabilities can be computed from others. Both of these approaches are incompatible with how model-based RL works, where we don't get to pick which observations we make. While it is true that sometimes you can compute some transition probabilities from others,this is counterproductive given the learning process of counting + normalizing.

v. (1 pt) Same as part iii, but how many R(s, a, s') values do we need to learn? If ambiguous, state assumptions.

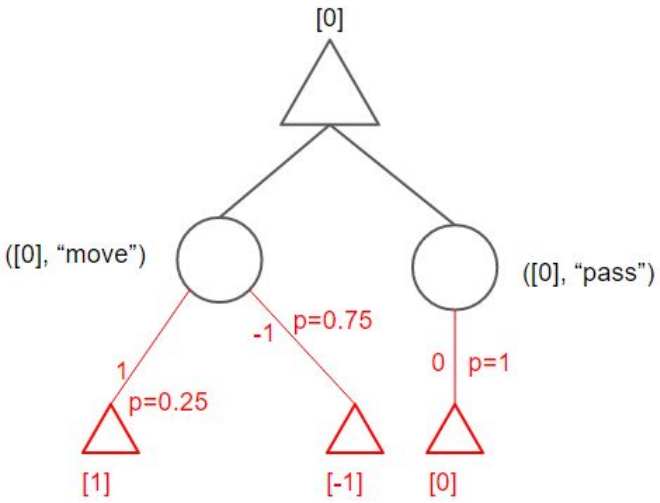
Number of R(s, a, s') values to learn = 0
Explanation: The rewards are all based on the rules of the game, and thus don't need to be learned.

Part 2) Suppose we keep the rules of The Downward Spiral the same, but change player 1's utility to **be equal to their elevation at the end of round N**, with no regard to player 2's elevation. This means adjusting our state space as well as reward function. Note: **You can do this problem without doing part 1.**

vi. (1 pt) Give a minimal representation of the state needed to achieve this new goal, and provide S_0. Hint: Player 2's actions no longer matter.

Solution 1: One representation is to store only the current elevation of player 1; the reward is given as +1 or -1 whenever we transition to a different state. The starting state would thus be S_0 = [0].
Solution 2: A better (and more minimal) representation is based off of the realization that the optimal action is independent of Player 1's elevation. Thus, our state only needs to be able to track whether Player 1 just moved up or down (in order to use the same reward function). Thus, the minimal state representation can be something like {"Up", "Down", "Same"}, with the reward for a transition $R(s, a, s')$ depending solely on s'. S_0 can be any of these states.

vii. (3 pts) Draw the entire MDP search tree for the game assuming N = 1. Make sure to label the probabilities and rewards for each transition for round 0, as well as the possible states for round 0 and round 1.

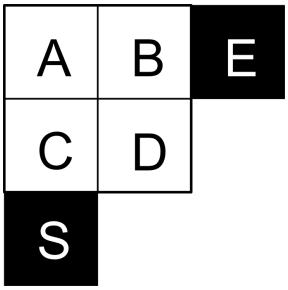


viii. (2 pt) If **N = 10,** give the expected utility $V^*(S_0)$ achieved under the optimal policy for this new MDP. Describe the optimal policy.

Solution: 0. We expect moving to move us lower, on average, and thus it is optimal to never move.

7. Where is Question Goat? (9 pts)

Question Goat is wandering around between positions A, B, C and D below. Question Goat's location at time t is X_t. At each time step, Question Goat moves clockwise [e.g. A → B, B → D, etc.] with probability 0.8, moves counterclockwise with probability 0.1, and stays at the same position with probability 0.1. To track down Question Goat's position, we deploy two sensors in the East (E) and South (S). The measurements of sensors E and S at time t are Y_E and Y_S, respectively. The distribution of sensor measurements are determined by the Manhattan distance between QG and the sensor. QG's Manhattan distance to sensor S is given by d_S and QG's Manhattan distance to sensor E is given by d_E, e.g. QG at B yields $d_S = 3$, $d_E = 1$. The measurement model of the sensors is given in the tables below. Essentially, as QG gets closer to a sensor, its chance of being on increases.



Y_E	$P(Y_E \mid d_E)$
0	$0.2\ d_E$
1	$1 - 0.2\ d_E$

Y_S	$P(Y_S \mid d_S)$
0	$0.3\ d_S$
1	$1 - 0.3\ d_S$

i. (2 pts) Let us find Question Goat's location by particle filtering with 4 particles. Suppose at $t = 33$, the particles we have are $X^{(1)}$=A, $X^{(2)}$=B, $X^{(3)}$=C, $X^{(4)}$=D. What is the probability that the particles are $X^{(1)}$=B, $X^{(2)}$=A, $X^{(3)}$=D, $X^{(4)}$=D after completion of the time elapsing step in particle filtering?

Probability: 0.0008

Explanation:

$$P(X_{34}^{(1)} = B, X_{34}^{(2)} = A, X_{34}^{(3)} = D, X_{34}^{(4)} = D \mid X_{33}^{(1)} = A, X_{33}^{(2)} = B, X_{33}^{(3)} = C, X_{33}^{(4)} = D)$$
$$= P(X_{34}^{(1)} = B \mid X_{33}^{(1)} = A) * P(X_{34}^{(2)} = A \mid X_{33}^{(2)} = A) * P(X_{34}^{(3)} = D \mid X_{33}^{(3)} = C) * P(X_{34}^{(4)} = D \mid X_{33}^{(4)} = D)$$
$$= 0.8 * 0.1 * 0.1 * 0.1 = 0.0008$$

ii. (2 pts) Assume the time elapse step yields $X^{(1)}$=B, $X^{(2)}$=A, $X^{(3)}$=D, $X^{(4)}$=D. Assume the sensor measurements at $t = 34$ are $Y_E = 0$ and $Y_S = 1$. What are the **unnormalized** particle weights given these observations?

Particle	Weights
$X^{(1)}$=B	$w^{(1)} = P(Y_E = 0 \mid X = B)P(Y_S = 1 \mid X = B) = 0.2 * 0.1 = 0.02$
$X^{(2)}$=A	$w^{(2)} = P(Y_E = 0 \mid X = A)P(Y_S = 1 \mid X = A) = 0.4 * 0.6 = 0.24$
$X^{(3)}$=D	$w^{(3)} = P(Y_E = 0 \mid X = D)P(Y_S = 1 \mid X = D) = 0.4 * 0.6 = 0.24$
$X^{(4)}$=D	$w^{(4)} = w^{(3)} = 0.24$

Notes: We know that the sensors are independent, so we just multiply the probabilities of the two sensor readings.

iii. (3 pts) Given the weights in part (b), what is the probability that the particles are resampled as $X^{(1)}$=**A**, $X^{(2)}$=**D**, $X^{(3)}$=**B**, and $X^{(4)}$=**B** at time t=34? You do not need to calculate the probability as a number. Instead, represent your solution in terms of $w^{(1)}$, $w^{(2)}$, $w^{(3)}$, $w^{(4)}$.

Solution: $\frac{(w^{(1)})^2 w^{(2)}(w^{(3)} + w^{(4)})}{(w^{(1)} + w^{(2)} + w^{(3)} + w^{(4)})^4}$

Explanation: We normalize the weights by dividing each by the sum of the weights, and then multiply the probabilities of being at A, B (twice), and D (noting that the total weight for being at D is $w^{(3)} + w^{(4)}$).

iv. (2 pts) If the sensors were malfunctioning at $t = 34$, i.e., no observations are produced, what is the probability that the original set of particles $X^{(1)}$=**A**, $X^{(2)}$=**B**, $X^{(3)}$=**C**, $X^{(4)}$=**D** are *resampled* as $X^{(1)}$=**B**, $X^{(2)}$=**A**, $X^{(3)}$=**D**, $X^{(4)}$=**D**? Probability:

After the exam, we noted that there were three possible interpretations of the problem; we gave all of them full credit.

Interpretation 1: "original set of particles" means pre-time elapse, and it is suboptimal to resample when there are no observations, so we shouldn't resample.

Solution 1: Since we aren't resampling, the answer is the same as in part (i): 0.0008.

Interpretation 2: "original set of particles" means pre-time elapse, and the particles post-time elapse are $X^{(1)}$=B, $X^{(2)}$=A, $X^{(3)}$=D, $X^{(4)}$=D as in part (ii).

Solution 2: The normalized weights for B and A are ¼, while the normalized weight for D is ½. Thus, the total probability is $1/4 * 1/4 * 1/2 * 1/2 = 1/64$.

Interpretation 3: "original set of particles" are the particles post-time elapse.

Solution 3: Our samples are uniform across the states, so each location is equally likely (1/4) for each particle. Thus, the answer is $1/4^4 = 1/256$.

8. Perceptrons and Neurons (9 pts, 1 pt each)

After a single weight update on a single incorrectly classified sample, a binary perceptron correctly classifies that sample: ○ Always ● Sometimes ○ Never

Explanation: The size of the change might not be enough; this is one reason why MIRA exists.

After a single weight update, the accuracy of a perceptron over the entire training set (check all that apply):

■ Can get better ■ Can get worse ■ Can stay the same

Explanation: Any of the outcomes can occur (since we don't know if the sample updated on is representative of the other samples, or even if the size of the change was enough (see previous part)).

In a binary perceptron, if a feature vector for a sample is exactly equal to the weight vector, the sample will be classified as part of the positive class. Assume the feature vector is not all zeros.

● Always ○ Sometimes ○ Never

Explanation: If the feature vector equals the weight vector, then the dot product is a sum of squares (nonnegative numbers); since it can't be zero (since the vectors aren't all zeroes), the dot product will always be positive.
Note: A few students have asked about how using a bias would affect the answer. The answer is that it wouldn't: if there's a bias feature, then either the bias weight = 1 (and the dot product is still positive), or the feature vector couldn't be exactly equal to the weight vector.

In a multiclass perceptron, if a feature vector for a sample is exactly equal to the weight vector for some class, the sample will be classified as part of that class. Assume the feature vector is not all zeros.

○ Always ● Sometimes ○ Never

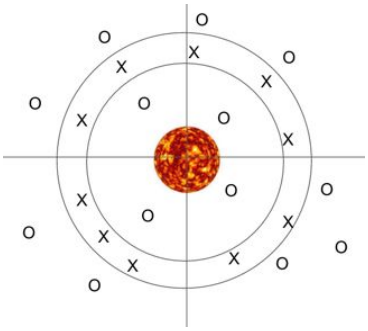
Explanation: If two classes had the same weight vector, then a sample with that same feature vector couldn't be classified as both classes. Alternatively, one class could have a weight vector that is a multiple of another class's weight vector (i.e. same direction but larger magnitude), in which case no sample would ever be classified as the class with smaller weights. And even if all weight vectors are unique, one vector may have such a large magnitude that it 'overpowers' smaller weight vectors.

The decision boundary for a binary perceptron is linear in the feature space.

● Always ○ Sometimes ○ Never

Explanation: Perceptrons only look at linear combinations of the features.

Suppose we are trying to classify the "habitable" zone around a star in a 2D universe. The data is shown below, where X denotes belonging to the habitable zone, and O is not part of the habitable zone.



A binary perceptron with the right weights would correctly classify all planets shown. Assume you can have any features you want.

● True ○ False

Explanation: There are a number of possible sets of features that could make this work. For example, we could have two indicator features that correspond to "being within first circle" and "being within second circle". Alternatively, if X was the distance from the star, we could have X and X^2.

On Project 6, you built a neural network of only a single two-class neuron with a logistic activation function that classified a sample as belonging to the positive class if its activation was greater than 0.5. The decision boundary for this neuron was linear in the feature space.

● Always ○ Sometimes ○ Never

Explanation: In general, neural networks can have non-linear decision boundaries, but that doesn't mean that this is the case for our particular neuron. The activation function used is a logistic activation function, which outputs a number larger than 0.5 iff the input is positive. The input to the activation function, however, is just the weighted sum of the inputs, and checking to see if that is positive is exactly how a perceptron would classify. Thus, our single neuron can't classify any better than a perceptron can.

A single two-class neuron as implemented in Project 6 (and described above) would be able to correctly classify the planetary data from above. Assume you can have any features you want.

● True ○ False

Explanation: As argued above, our neuron actually classifies the exact same way that a perceptron would, and a perceptron can classify this data perfectly.

The training data for Project 6 featured handwritten digits that were all about the same size. However, if you hand-wrote a single much smaller "5" and fed it to the classify method (using the same training data as in the project, and assuming the handwritten digit was appropriately converted into a 28x28 pixel format)), it would still probably be classified correctly.

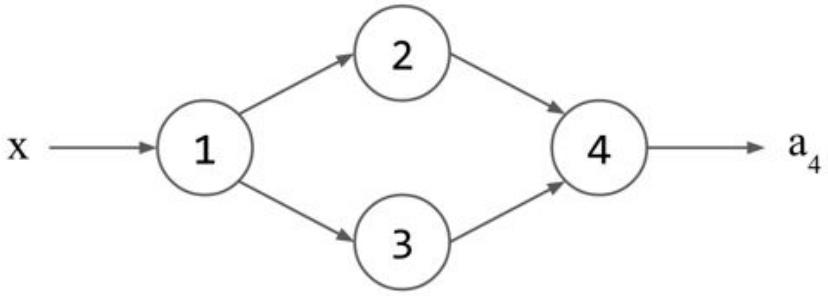
○ True ● False

Explanation: Our features were per-pixel. Thus, if the 5 was in an unexpected location (including but not limited to being smaller in size) that was never seen in the training data, then the models in the project (either the perceptron or the neuron) would not be able to classify the 5 correctly. However, some students interpreted "appropriately converted" to include the step of finding and extracting the digit, magnifying the digit to a "normal" size. Thus we took both answers.

9. Deep Learning (10.5 pts)

Consider the neural network below. Note that:

- **x**, a scalar, is the input. a_4, also a scalar, is the output.
- Each $\mathbf{a_i}$ value is the final output for neuron **i** in the network.
- Each $\mathbf{z_i}$ value is the *pre-activation* value for neuron **i** in the network (i.e. the dot product).
- $\mathbf{w_1}$ is the weight used by neuron 1.
- Let $\mathbf{w_{ij}}$ be the weight from neuron **i** to neuron **j**.
- The activation function for each neuron is the function $g(x) = e^x$ **.**
- ***L(y, a$_4$)*** is the loss function. It is $(y - a_4)^2$, where y is the training label.



i. (2.5 pts) Use the Chain Rule and the equations given to calculate $\partial L/\partial z_2$. You may use *x* and *y*, along with all of the *w*, *a*, and *z* values. Note that the picture above only shows the neural network, i.e. does not include the loss function L.

Solution: $\frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial a_2}\frac{\partial a_2}{\partial z_2} = (-2(y - a_4)) * (e^{z_4}) * (w_{24}) * (e^{z_2})$

ii. (1.5 pts) Use the Chain Rule and the equations given to calculate $\partial L/\partial z_3$. You may use *x* and *y*, along with all of the *w*, *a*, and *z* values. (*Hint*: If you're comfortable with the Chain Rule, you should be able to do this quickly by looking at your solution to Part A.)

Solution: $\frac{\partial L}{\partial z_3} = \frac{\partial L}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial a_3}\frac{\partial a_3}{\partial z_3} = (-2(y - a_4)) * (e^{z_4}) * (w_{34}) * (e^{z_3})$

iii. (2.5 pts) Use the Chain Rule and the equations given to calculate $\partial L/\partial w_1$. You may use x and y, along with all of the w, a, and z values. ***NOTE*:** You can use the symbols $\partial L/\partial z_2$ and $\partial L/\partial z_3$ in your answer, since you calculated their values above. This is *highly recommended*.

Solution:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = (\frac{\partial L}{\partial z_2}\frac{\partial z_2}{\partial a_1} + \frac{\partial L}{\partial z_3}\frac{\partial z_3}{\partial a_1})\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = (\frac{\partial L}{\partial z_2}w_{12} + \frac{\partial L}{\partial z_3}w_{13}) * (e^{z_1}) * (x))$$

iv. (1 pt) Suppose we're using stochastic gradient descent. What values will be updated based on the partial you computed in part iii, i.e. $\partial L/\partial w_1$? Check all that apply.

M w_1 ☐ w_{12} ☐ w_{13} ☐ w_{24} ☐ w_{34} ☐ x

v. (1 pt) Suppose we're using mini-batching instead of stochastic gradient descent. What values will be updated based on the computation of $\partial L/\partial w_1$? Check all that apply.

M w_1 ☐ w_{12} ☐ w_{13} ☐ w_{24} ☐ w_{34} ☐ x

Explanation for (iv) and (v): In general, calculating $\partial L/\partial w_i$ only affects w_i.

vi. (1 pt) What is the main difference between mini-batching and stochastic gradient descent? Describe in 10 words or less. **Answers with more than 10 words will receive no credit.**

Solution: SGD uses 1 sample, Mini-Batch uses random larger sub-samples**.**

Common Errors: Mistaking samples for weights, confusing SGD for Mini-batch

vii. (1 pt) We can think of backpropagation as a graph traversal over a computational graph. To ensure that all values are available at the time they are needed, only some traversals are accurate. There are some orderings of nodes/neurons that we must use for maximum efficiency. ***For this particular network***, select **all** such orderings that apply.

☐ BFS ordering, starting from a_1
☐ DFS ordering, starting from a_1
☐ UCS ordering, starting from a_1 and all edge weights are 188.
M BFS ordering, starting from a_4 and flipping all edge directions
☐ DFS ordering, starting from a_4 and flipping all edge directions
M UCS ordering, starting from a_4, flipping all edges, and all edge weights are 188.

Explanation: We want to have calculated the gradients of all of a neuron's children before we calculate the neuron itself's gradient. BFS (or UCS with uniform edge weights, which is identical to BFS) on the reverse of the graph will explore each layer from right to left, in order

10. Pacmanian Speech Recognition (10.5 pts)[1]

As Adam taught during his lecture, one common approach for converting sound files to words (speech recognition) is to first featurize the sound file by converting each 10 millisecond segment into 13 binned "spectral energies", which are essentially how strong the low frequency, medium frequency, etc. parts of the signal are for each 10 millisecond segment.

For this problem, the notation for our features is $F_t(i)$ where t ranges from 0 to the number of frames in the audio file and i ranges from 0 to 12. For example, if we are processing a 1 second clip, we'd have $F_0(i)$ through $F_{99}(i)$, for a total of 1300 features. If $F_5(0) = 2.3$, that means the very low frequency component of the sound clip starting at 50ms and ending at 60ms had an energy of 2.3 in some arbitrary unit.

Suppose throughout this problem that Pacmanian has 20 different sounds, a.k.a. "phonemes" that occur with exactly equal probability. Each word in Pacmanian is composed of a sequence of one or more phonemes. For example, the word "waka" is composed of the phonemes `/w/`, `/a/`, `/k/`, `/a/`.

Part 1: The first step of our speech processing algorithm is to build a model for how likely each phoneme is, given a sound segment. For example, our model might predict that $F_5(i)$, the sound segment between 50ms and 60ms, had a 40% percent chance of being a `/w/`, a 20% percent chance of being a `/y/`, etc.

To do this, we can build a neural network that takes in all 13 binned energies for a particular time segment, and which has 20 outputs that use a softmax activation function.

The notation for the outputs of the neural network is $N_t(i)$, where t is the frame number and i ranges from 0 to 19. $N_{15}(0)$ is for `/a/` at time 15, $N_{31}(1)$ is for `/b/` at time 31, etc.

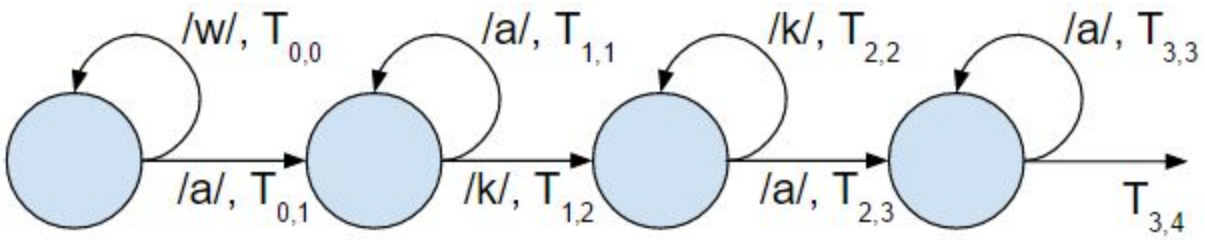
In other words we build a classifier that takes a sound clip (13 energy values) and outputs 20 values (one for each possible phoneme), where $N_t(i)$ represents how likely the sound at timestep t is actually phoneme number i.

i. (2 pts) For each of the following expressions, indicate if it is Always True, Sometimes True, or Never True for all values of t, i, and k. Mark "Sometimes" even if the chance is very small.

	Always	Sometimes	Never	Reason
$0 <= N_t(i) <= 1.0$	●	○	○	Output of softmax
$\sum_i N_t(i) = 1.0$	●	○	○	Output of softmax
$N_t(0) > N_t(1)$	○	●	○	Could be in any order
$F_t(i) = N_t(i)$	○	●	○	By coincidence

[1] I know there were some complaints about wordy problems, but I felt this problem was too hard without reexplaining at least some of Adam's lecture. With a strong enough 188 foundation, you can do this problem without having deeply understood Adam's lecture.

Part 2: One way to try to detect words is to build an HMM for each possible word. The state transition diagram for the HMM is shown below, where state i is the ith phoneme of the word, and $T_{i,j}$ is the probability of moving from state i to state j. For example, for the word waka, we might assume that there is a 20% chance that that a speaker saying the `/a/` sound keeps making that sound 10 ms later, and an 80% chance that they move on to the `/k/` sound. In that case, $T_{1,1}$ would be 0.2 and $T_{1,2}$ would be 0.8. $T_{3,4}$ represents the speaker finishes the word. If you'd like, you can think of the terminal state is the speaker being silent.



ii. (2.5 pts) Fill in the table, similar to what you did in part i. Mark "Sometimes" even if the chance is very small.

	Always	Sometimes	Never	Reason
$0 <= T_{i,k} <= 1.0$	●	○	○	Probability
$T_{0,1} + T_{0,0} = 1.0$	●	○	○	Probability
$T_{0,1} + T_{1,2} + T_{2,3} = 1.0$	○	●	○	By coincidence only
$T_{1,1} > T_{2,2}$	○	●	○	By coincidence only
$T_{1,1} > T_{3,3}$	○	○	●	These are the same value. See NOTE below.

Part 3: Suppose we have a sound file containing a single word. Suppose this sound file is S samples long, has 13 bins of spectral energy, and that Pacmanian has 20 phonemes. **For iii through vi, suppose that we want to use our HMM from above to determine whether the word is "waka".**

iii. (2 pts) How many **hidden variables** must our HMM have? Recall that an HMM is just a Bayes Net with a special structure.

Solution: S, as we have a hidden variable Q_t for each time step, where $P(Q_t = q_t)$ represents the probability that the phoneme spoken during time t is qt. Each variable is a vector of 20 probabilities (just like the weather HMM example from lecture, where the variables had values "cloudy", "rainy", etc.)

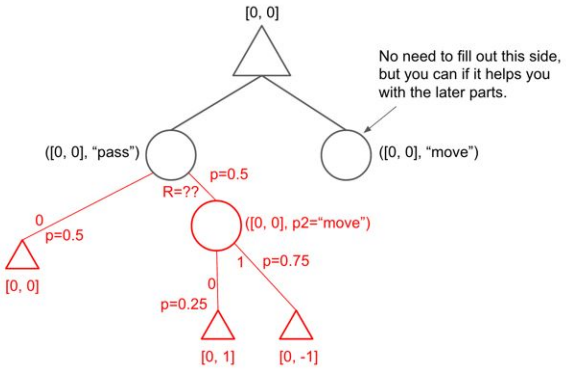
iv. (2 pts) If q_5 is our 5th hidden variable, and q_6 is our 6th hidden variable, how many rows does $P(q_6 \mid q_5)$ have? If the problem seems ambiguous, make sure to state your assumptions.
Solution: 400: there are 20 possible values for q_5, and 20 possible values for q_6.
Alternate solution: 441, where we also allow silence, is also accepted.
Alternate solution 2: If you assume that the HMM *only* needs to support the word "waka", then the only rows that the transition matrix need to support are $a \rightarrow a, a \rightarrow k, k \rightarrow a, k \rightarrow k, w \rightarrow a, w \rightarrow w$, and maybe also $a \rightarrow EXIT$. Other solutions also exist depending on assumptions of what's in the CPT-- see rubric on gradescope. See also NOTE below.

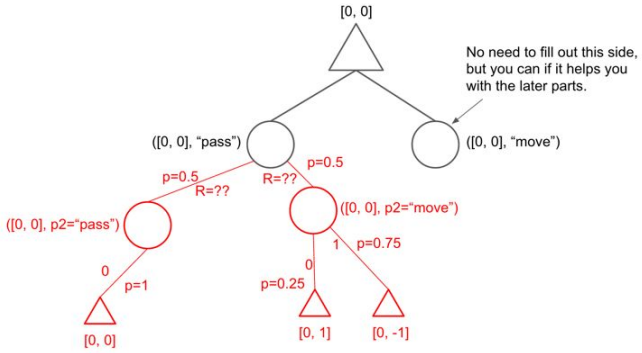
v. (2 pt) For each hidden variable, how many scalar evidence values are there?
Solution: 13: one for each spectral bin.

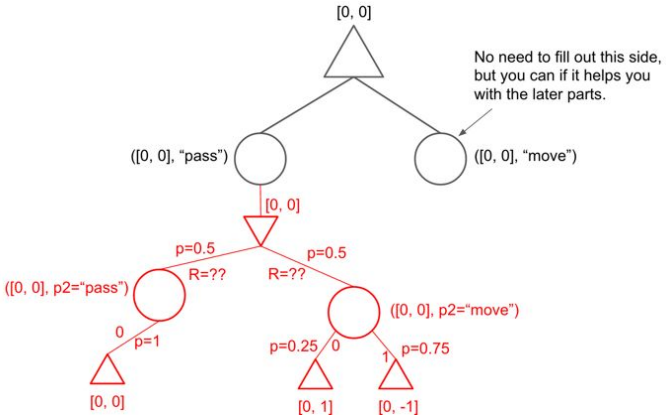
NOTE: Several students have asked about the two /a/'s in waka. They are the same /a/. The issue is the same as the weather HMM example, where all you have is a probability of it being cloudy tomorrow given it was cloudy today. It's not dependent on the history. In fact, researchers have tried making each phoneme word-position dependent, but it very quickly becomes intractable. Imagine if each and every /a/ were it's own entry -- the CPTs would be huge.

Appendix: Alternate approaches to Problem #6

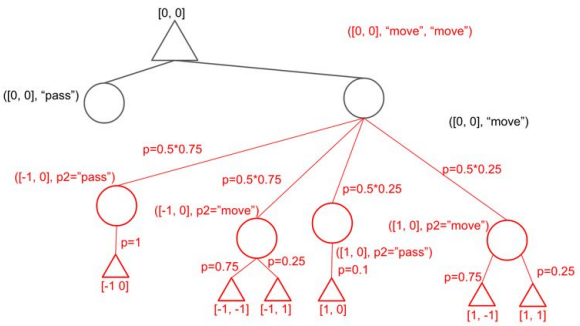
6.i One common incorrect answer was to either have transitions directly between Q-states (top two diagrams), or to have a special "opponent" node with probabilistic behavior. None of these is an MDP search tree. The top left model is fundamentally broken, since it has no Q state for p2="pass". The top right model is strange because we have to allow transitions directly between Q-states, but could theoretically work. The bottom model is an abuse of the symbol for a minimizing agent, but is ultimately equivalent to the top right model.

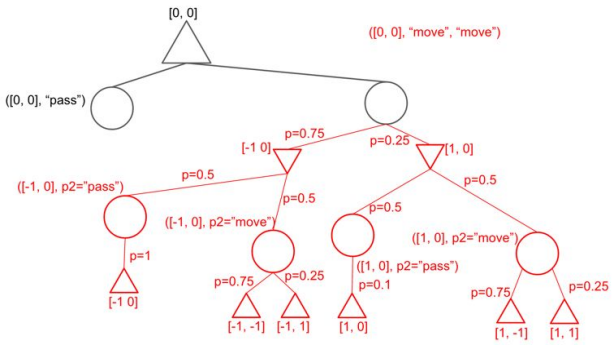






6.ii If you had one of the incorrect solutions for part i, then you didn't actually draw an MDP search tree. In this case, the question is a bit open to interpretation: the smallest transition probability was either 1/4, 1/8, or 1/32, depending on if you counted the entire aggregate transition probability from the Q-states given to the roots, as well as how you built your model.





6.iv. If you used model 1 or 2, the correct answer is zero, not Z.