CS 188 Fall 2015 — Introduction to Artificial Intelligence — Final

- You have approximately 2 hours and 50 minutes.
- The exam is closed book, closed calculator, and closed notes except your one-page crib sheet.
- Mark your answers ON THE EXAM ITSELF. If you are not sure of your answer you may wish to provide a *brief* explanation. All short answer sections can be successfully answered in a few sentences AT MOST.

First name	
Last name	
SID	
edX username	
Name of person on your left	
Name of person on your right	

For staff use only:

Q1.	Game Trees	/7
Q2.	CSP Futoshiki	/11
Q3.	Search, logic, and learning	/19
Q4.	Probability and Bayes Nets	/20
Q5.	Decision Networks and MDPs	/17
Q6.	MDPs/RL	/8
Q7.	Machine Learning	/18
	Total	/100

Q1. [7 pts] Game Trees

(a) Alpha-beta pruning true/false For each true/false question, circle the correct answer. Missing choices and wrong choices with no explanation are worth zero.

(i) [1 pt] [*true* or *false*] Minimax search with alpha-beta pruning may not find a minimax optimal strategy.
False. Alpha-beta will always find the optimal strategy for players playing optimally.

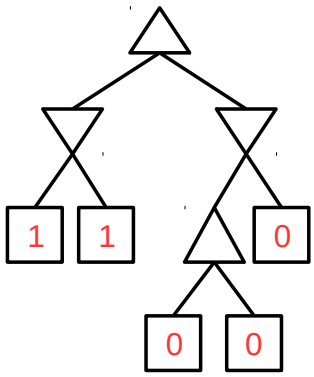
(ii) [1 pt] [*true* or *false*] Alpha-beta pruning prunes the same number of subtrees independent of the order in which successor states are expanded.
False. If a heuristic is available, we can expand nodes in an order that maximizes pruning.

(iii) [1 pt] [*true* or *false*] Minimax search with alpha-beta pruning generally requires more run-time than minimax without pruning on the same game tree.
False. Alpha-beta will require less run-time than minimax except in contrived cases, which is the whole point of pruning.

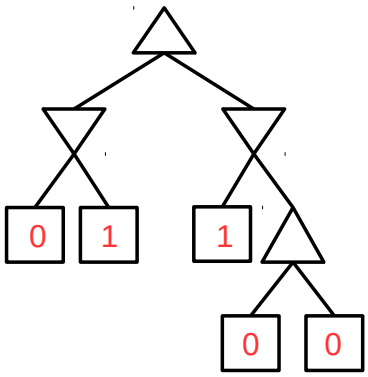
(b) [4 pts] For each of the following minimax game trees (max is at the root), fill in the leaf nodes with 0 or 1 as utility values. For the left tree (i), fill in the leaves so that as many leaves will be pruned by alpha-beta as possible. For the right tree (ii), fill in the leaves so that no leaves are pruned. Note that the two trees are slightly different.

Assume (1) left to right traversal while pruning, and (2) leaf values are potentially drawn from $[-\infty, \infty]$ (not from $[0, 1]$).

(i) Maximum Pruning



(ii) No Pruning

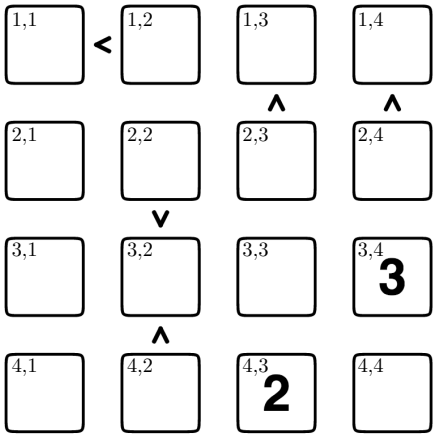


These answers are one example, and other solutions are also possible. For (i), the values chosen should cause the last leaf node to be pruned.

Q2. [11 pts] CSP Futoshiki

Futoshiki is a Japanese logic puzzle that is very simple, but can be quite challenging. You are given an n x n grid, and must place the numbers $1, \ldots n$ in the grid such that every row and column has exactly one of each. Additionally, the assignment must satisfy the inequalities placed between some adjacent squares.

To the right is an instance of this problem, for size $n = 4$. Some of the squares have known values, such that the puzzle has a unique solution. (The letters mean nothing to the puzzle, and will be used only as labels with which to refer to certain squares). Note also that inequalities apply only to the two adjacent squares, and do not directly constrain other squares in the row or column.



Let's formulate this puzzle as a CSP. We will use 4^2 variables, one for each cell, with X_{ij} as the variable for the cell in the ith row and jth column (each cell contains its i, j label in the top left corner). The only unary constraints will be those assigning the known initial values to their respective squares (e.g. $X_{34} = 3$).

(a) [3 pts] Complete the formulation of the CSP using only binary constraints (in addition to the unary constraints specificed above. In particular, describe the domains of the variables, and all binary constraints you think are necessary. You do not need to enumerate them all, just describe them using concise mathematical notation. You are not permitted to use n-ary constraints where $n \geq 3$.

Domains: $X_{ij} \in \{1, 2, 3, 4\}, \ \forall i, j$
Unary constraints: $X_{34} = 3$, $X_{43} = 2$
Inequality binary constraints: $X_{11} < X_{12}$, $X_{13} < X_{23}$, $X_{14} < X_{24}$, $X_{32} < X_{22}$, $X_{32} < X_{42}$
Row binary contraints: $X_{ij} \neq X_{ik}, \forall i, j, k, j \neq k$
Column binary contraints: $X_{ij} \neq X_{kj}, \forall i, j, k, i \neq k$

(b) [2 pts] After enforcing unary constraints, consider the binary constraints relating X_{14} and X_{24}. Enforce arc consistency on just these constraints and state the resulting domains for the two variables. $X_{14} \in \{1, 2\}$, $X_{24} \in \{2, 4\}$. Note that both threes are removed from the column constraint with X_{34}.

(c) [2 pts] Suppose we enforced unary constraints and ran arc consistency on this CSP, pruning the domains of all variables as much as possible. After this, what is the maximum possible domain size for any variable? [*Hint*: consider the least constrained variable(s); you should *not* have to run every step of arc consistency.] The maximum possible domain size is 4 (ie, no values are removed from the original domain). Consider X_{21} – we will not be able to eliminate any values from its domain through arc consistency.

(d) [2 pts] Suppose we enforced unary constraints and ran arc consistency on the initial CSP in the figure above. What is the maximum possible domain size for a variable adjacent to an inequality? The maximum domain size is 3 – you must always eliminate either 1 or 4 from a variable participating in an inequality constraint.s

(e) [2 pts] By inspection of column 2, we find it is necessary that $X_{32} = 1$, despite not having found an assignment to any of the other cells in that column. Would running arc consistency find this requirement? Explain why or why not. No, arc consistency would not find this requirement. Encorcing the $X_{32} \rightarrow X_{42}$ and the $X_{42} \rightarrow X_{43}$ arc leaves X_{42} with a domain of $\{3, 4\}$. Enforcing the $X_{32} < X_{22}$ constraints leaves $X_{32} \in \{1, 2, 3\}$ and $X_{22} \in \{2, 3, 4\}$. Enforcing that they are all different does not remove any values. After this point, every arc in this column is consistent and X_{32} is not required to be 1.

Q3. [19 pts] Search, logic, and learning

In this question we consider the problem of searching for the smallest propositional logic sentence ϕ that satisfies some condition $G(\phi)$. (E.g., "find the smallest unsatisfiable sentence containing two distinct symbols.") Recall that a propositional logic sentence is a proposition symbol, the negation of a sentence, or two sentences joined by $\wedge$, $\vee$, $\Rightarrow$, or $\Leftrightarrow$. The proposition symbols are given as part of the problem. The size of a sentence is defined as the sum of the sizes of its logical connectives, where $\neg$ has size 1 and the other connectives have size 2. (It is helpful to think of the sentence as a syntax tree with proposition symbols at the leaves; then the size is the number of edges in the tree. Figure 1(a) shows an example.)

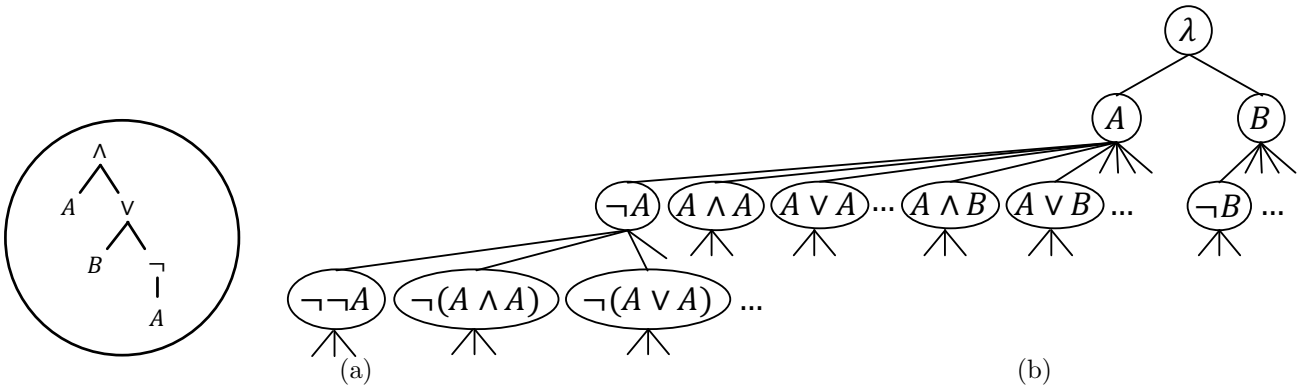


Figure 1: (a) The sentence $A \wedge (B \vee \neg A)$ drawn as a syntax tree with 5 edges. (b) Part of the "parsimonious" search space for sentences with symbols A and B.

(a) [2 pts] Hank Harvard proposes the following problem formulation to explore the search space of sentences:

- There is a dummy start state λ, which is an empty sentence that never satisfies G. The actions applicable in this state simply replace λ by one of the proposition symbols.
- For all other states, the actions take any occurrence of a proposition symbol (call it p) in the sentence and replace it with either $\neg q$ for any symbol q, or $r * s$ where r and s are any symbols and "$*$" is any binary connective.

Buzzy Berkeley agrees with Hank that this set of actions will generate all and only the syntactically valid sentences, but proposes her own formulation:

- For all other states, the actions take any occurrence of a proposition symbol (call it p) in the sentence and replace it with either $\neg p$, or $p * q$ where q is any symbol and "$*$" is any binary connective.

Part of the search space generated by Buzzy's formulation is shown in Figure 1(b). Buzzy claims that her formulation is more efficient than Hank's but still generates all and only the syntactically valid sentences. Is she right? Explain.

Yes. It is more efficient because it has a smaller branching factor and eliminates redundant copies of sentences. For any sentence with an occurrence of p in some location, there is another sentence identical except that the occurrence of p is replaced by an occurrence of q. (Essentially this is the inductive hypothesis in the proof.) Hence there is no need to replace p by other symbols.

(b) [1 pt] Define the step cost function for Buzzy's formulation.

1 for negating a symbol, 2 for introducing a binary connective.

(c) [1 pt] Assuming there are n symbols, give a $O()$-expression for the branching factor at depth d of Buzzy's search tree.
Each step adds no more than 1 symbol occurrence; so at depth d there are up to d symbols. There is one way to negate a symbol and n ways to add each of four binary connectives, so the branching factor is $O(4nd)$.

(d) [1 pt] Using your $O()$ answer for the branching factor, give a $O()$-expression for the number of nodes at depth d of Buzzy's search tree.
The off-the-shelf answer of b^d doesn't work because b is not constant. The number of nodes is the product of the branching factors, so $O((4n)^d \cdot d!)$.

(e) [2 pts] We will say that G is a *semantic* condition if $G(\phi)$ depends only on the *meaning* of ϕ, in the following sense: if two sentences ϕ and ψ are logically equivalent, then $G(\phi) = G(\psi)$. Not wishing to be outdone by Buzzy, Hank now makes the following claim: Whenever Buzzy's search space contains a solution for a semantic G then so does the reduced search space using only $\neg$, $\wedge$, and $\vee$. Is he right? Explain.
Yes. Every propositional logic sentence can be expressed in CNF, which uses only $\neg$, $\wedge$, and $\vee$.

(f) [2 pts] Suppose we are running a uniform cost graph search, which maintains the property that $g(n)$ for every node in the frontier is the optimal cost to reach n from the root. In a standard graph search, we discard a newly generated successor state without adding it to the frontier if and only if the *identical* state is already in the frontier set or explored set. Assuming we have a semantic condition G, when is it possible to discard a newly generated successor state? Check all that apply.

- [x] Sentences that are identical to a sentence already in the frontier or explored set
- [ ] Sentences that logically entail a sentence already in the frontier or explored set
- [x] Sentences that are logically equivalent to a sentence already in the frontier or explored set
- [ ] Sentences that are logically entailed by a sentence already in the frontier or explored set

Now we will apply this general search machinery for sentences to supervised machine learning: the problem of finding concise hypotheses that are consistent with a set of labeled examples and predict labels for unlabeled examples.

- Let $X_1, \ldots, X_n$ be the Boolean input variables and Y be the "output" Boolean property we are trying to predict. A hypothesis H asserts that Y is some function of the inputs, i.e., H is a sentence $Y \Leftrightarrow \phi$, where ϕ is a propositional formula containing only $X_1, \ldots, X_n$. For example, let H_0 be the sentence $Y \Leftrightarrow (X_1 \vee X_2)$; then H_0 asserts that the output is true exactly when either of the inputs X_1, X_2 is true.
- A labeled example E gives the output value for particular values of the inputs; i.e., it is a sentence $\psi \Rightarrow \chi$, where ψ is a conjunction of literals, one for each input symbol, and χ is a literal containing the output symbol. Thus if there are just two inputs, three examples might be

$$\begin{aligned} E_1 &: \quad X_1 \wedge X_2 \Rightarrow Y \\ E_2 &: \quad X_1 \wedge \neg X_2 \Rightarrow Y \\ E_3 &: \quad \neg X_1 \wedge \neg X_2 \Rightarrow \neg Y \end{aligned}$$

(g) [2 pts] Show, in general, that every labeled example sentence is *false in exactly one model* for the symbols $X_1, \ldots, X_n, Y$.
An implication is false iff the LHS is true and the RHS is false. Those conditions set the values of all the symbols because the LHS is a conjunction of n literals, one per input symbol.

(h) [4 pts] Show, by completing the following truth table and marking the relevant rows, that the hypothesis H_0 given above logically entails all three examples.

X_1	X_2	Y	E_1	E_2	E_3	H_0
F	F	F	T	T	T	T ←
F	F	T	T	T	F	F
F	T	F	T	T	T	F
F	T	T	T	T	T	T ←
T	F	F	T	F	T	F
T	F	T	T	T	T	T ←
T	T	F	F	T	T	F
T	T	T	T	T	T	T ←

(i) [2 pts] For supervised machine learning, then, we want to find an H, by searching among formulas ϕ, that entails all the examples. Suppose you have available a solver $SAT(\alpha)$ that returns *true* if α is satisfiable and *false* otherwise. Which of the following expressions correctly implements $G(\phi)$?

- ○ $SAT(H \wedge E_1 \wedge E_2 \wedge E_3)$
- ○ $SAT(H \wedge \neg(E_1 \wedge E_2 \wedge E_3))$
- ● $\neg SAT(H \wedge \neg(E_1 \wedge E_2 \wedge E_3))$
- ○ $\neg SAT(H \wedge E_1 \wedge E_2 \wedge E_3)$

(j) [2 pts] Now suppose we have an unlabeled test example U_4 described only by the ψ-part:

$$U_4 : \qquad \neg X_1 \wedge X_2$$

Now we want to predict the label for U_4 given a hypothesis H. Which of the following does this correctly?

- ○ $SAT(H \wedge U_4 \wedge Y)$
- ○ $\neg SAT(H \vee (U_4 \rightarrow Y))$
- ○ $\neg SAT(H \wedge (U_4 \rightarrow Y))$
- ● $\neg SAT(H \wedge \neg(U_4 \rightarrow Y))$

Extra Credit [2] (Extra credit, HARD) Any ideas for a *nontrivial* admissible heuristic for the supervised machine learning problem in this setting?

Q4. [20 pts] Probability and Bayes Nets

(a) [3 pts] A, B, C, and D are Boolean random variables. How many entries are in the following probability tables and what is the sum of the values in each table? Write "?" if there is not enough information given.

Table	Size	Sum
$P(a \mid B)$	2	?
$P(A \mid B, c, D)$	8	4
$P(B, C \mid \neg a)$	4	1

(b) For each of the following expressions for the joint distributions $P(A, B, C, D)$ write the all of the independence assumptions used:

(i) [2 pts] $P(A, B, C, D) = P(A \mid B, C)P(D \mid B, C)P(B)P(C \mid B)$

$D \perp\!\!\!\perp A \mid B, C$

(ii) [2 pts] $P(A, B, C, D) = P(B)P(C)P(A \mid B, C)P(D \mid A, B, C)$

$B \perp\!\!\!\perp C$

(iii) [2 pts] $P(A, B, C, D) = P(B)P(C \mid B)P(A \mid B)P(D \mid B, C)$

$(A \perp\!\!\!\perp C \mid B$ and $A \perp\!\!\!\perp D \mid B, C)$ or $(A \perp\!\!\!\perp C \mid B, D$ and $A \perp\!\!\!\perp D \mid B)$

(c) Write each of the following expressions in its simplest form, i.e., a single term (factor). Make no independence assumptions.

(i) [2 pts]

$$\sum_{a'} P(a' \mid D)P(b \mid a', D)$$

$P(b \mid D)$

(ii) [2 pts]

$$\sum_{b', c', d'} P(A)P(b' \mid A)P(c' \mid A, b')P(d' \mid A, b', c')$$

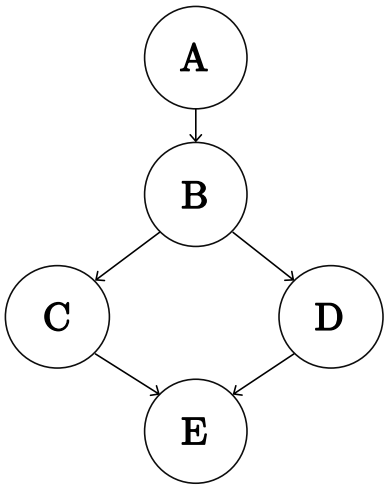
$P(A)$

(iii) [2 pts]

$$\frac{\sum_{b'} P(A \mid b', D)P(b' \mid D)P(D)}{\sum_{a', b'} P(a' \mid b', D)P(b' \mid D)P(D)}$$

$P(A \mid D)$

(d) [2 pts] Given the following Bayes net, write an expression for the probability distribution $P(A|e)$ using only the conditional probability distributions associated with this Bayes net, e.g. $P(A)$, $P(C|B)$, etc. Normalization constants, such as α or Z are not permitted.



$P(A|e) =$

$$\frac{\sum_{b',c',d'} P(A)P(A|b')P(c'|b')P(d'|b')P(e|c',d')}{\sum_{a',b',c',d'} P(a')P(a'|b')P(c'|b')P(d'|b')P(e|c',d')}$$

(e) Sampling: Our new droid BB-8 is using likelihood weighted sampling to answer various queries on the Bayes net from part (d).

(i) [1 pt] [*true* or *false*] For the query $P(E|b)$, the value sampled for variable A will have no effect on the weight of the complete sample.
False. The weight on the sample will be $P(b|a)$, which could definitely be different for different values a.

BB-8 has implemented a simpler version of likelihood weighted sampling to answer the query $P(E|b)$. BB-8's method skips sampling variable A and skips incorporating the weight associated with B and proceeds to sample values for C, D, and then E from $P(C|B)$, $P(D|B)$, and $P(E|C,D)$, respectively.

(ii) [1 pt] [*true* or *false*] BB-8's simpler sampling method will converge to the same answer for $P(E|b)$ as standard likelihood weighted sampling.
True. Without loss of generality, assume A is a binary variable. With standard likelihood weighted sampling, the weights can be different for different sampled values of A. However, A is conditionally independent from C, D, and E, given B, ...TODO

(iii) [1 pt] Is it possible to make a similar simplification to likelihood weighted sampling to make it more efficient (but still accurate in the limit) when answering the query $P(E|c)$. If so, briefly describe the simplification.
No, it is not possible. There are no independence relationships that can allow us to get around the need to sample A, B, D, and E, and compute the weight associated with C.

Q5. [17 pts] Decision Networks and MDPs

Typically, we define a Markov Decision Process (MDP) as a set of states S, a set of actions A, a reward function $R(s)$ (note that we don't use $R(s, a, s')$ here), a starting state s_0, and a transition model $T(s, a, s')$. Assume that our discount factor $\gamma = 1$. In this question, we'll explore whether or not we can use *decision networks* to solve MDPs. Recall that a decision network is a Bayes net augmented with action and utility variables to model decision problems.

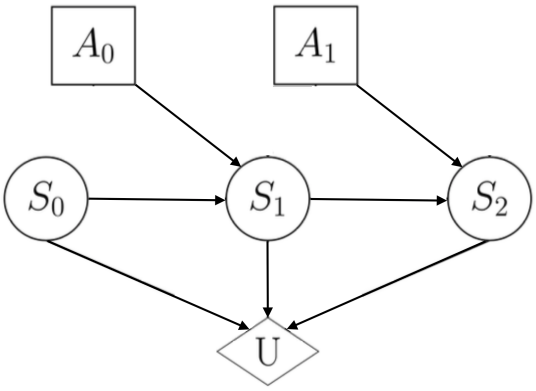
Here's the idea of how we'll try to solve an MDP using decision networks: our random variable nodes will model the state over the first $t + 1$ timesteps, i.e., $S_0, \ldots, S_t$, and our action nodes will model the action over the first t timesteps, i.e., $A_0, \ldots, A_{t-1}$. Using this formulation, we can then just use the decision algorithm to figure out the sequence of actions to take from time 0 to $t - 1$. We can repeat this to continue to find actions forever.

First, let's introduce the specific, very simple MDP that we'll work with for the rest of the question. In this MDP, we have two states *init* and *goal*, and two actions *stay* and *move*. The starting state is *init*. The reward and transition functions are as follows:

s	$R(s)$
init	0
goal	1

s	a	s'	$T(s, a, s')$
init	*stay*	*init*	1
init	*stay*	*goal*	0
init	*move*	*init*	0.2
init	*move*	*goal*	0.8
goal	*stay*	*init*	0
goal	*stay*	*goal*	1
goal	*move*	*init*	0.8
goal	*move*	*goal*	0.2

Let's use $t = 2$ in this question to keep things simple. For this MDP, we draw the decision network below. Note that we do not write out the probability table for S_2, because it is the same as S_1 shifted over.



s_0	$P(s_0)$
init	1
goal	0

s_0	a_0	s_1	$P(s_1 \mid s_0, a_0)$
init	*stay*	*init*	1
init	*stay*	*goal*	0
init	*move*	*init*	0.2
init	*move*	*goal*	0.8
goal	*stay*	*init*	0
goal	*stay*	*goal*	1
goal	*move*	*init*	0.8
goal	*move*	*goal*	0.2

s_0	s_1	s_2	$U(s_0, s_1, s_2)$
init	*init*	*init*	0
init	*init*	*goal*	1
init	*goal*	*init*	1
init	*goal*	*goal*	2
goal	*init*	*init*	1
goal	*init*	*goal*	2
goal	*goal*	*init*	2
goal	*goal*	*goal*	3

(a) [4 pts] For the given decision network, fill out the utility table above. *Hint:* Think intuitively about how we measure utility in an MDP.

(b) [5 pts] Using this decision network that is now complete, let's calculate the expected utilities of all possible action sequences $[A_0, A_1]$. As a starting point, we've calculated the expected utility of $[A_0, A_1] = [move, stay]$ as 1.6, and the expected utility of $[move, move]$ as 1.12.

(i) [2 pts] What is the expected utility of $[stay, stay]$?

$$EU(stay, stay) = \sum_{s_0, s_1, s_2} P(s_0, s_1, s_2 \mid stay, stay) U(s_0, s_1, s_2) = 0$$

Because the only possible sequence of states, if $s_0 = init$ and our actions are $stay, stay$, is $init, init, init$.

(ii) [2 pts] What is the expected utility of $[stay, move]$?

$$\begin{aligned} EU(stay, stay) &= \sum_{s_0, s_1, s_2} P(s_0, s_1, s_2 \mid stay, move) U(s_0, s_1, s_2) \\ &= \sum_{s_2} P(s_2 \mid init, move) U(init, init, s_2) \\ &= 0.2(0) + 0.8(1) = 0.8 \end{aligned}$$

Since we can be sure that both s_0 and s_1 are $init$ with probability 1.

(iii) [1 pt] What is the optimal action sequence $[A_0, A_1]$?
$[move, stay]$ because it yield the highest expected utility.

(c) [4 pts] Does this optimal action sequence give us the same optimal policy that we would obtain if we solved the MDP using value or policy iteration? Why, or why not?
No, an action sequence is not as good as a policy. With an action sequence, we determine at the beginning which actions to take for both time steps. However, with an optimal policy, we wait until we find out what value S_1 takes before choosing A_1, and that way we can pick the best action. The optimal action sequence we found was $[move, stay]$, but if the $move$ action didn't work, then the second action would optimally be $move$. The optimal policy can deal with this, but the optimal sequence of actions is still stuck with $stay$.

(d) [4 pts] Suppose we fix $A_0 = move$. Calculate $VPI(S_1)$. Clearly show and label your work for potential partial credit. *Hint:* you may need some of the expected utilities used in part (b).
As mentioned above, if S_1 takes on the value $init$, that means our $move$ action failed, and we should deal with this by using another $move$ action for A_1. So, intuitively, observing S_1 should increase our maximum expected utility.

$$\begin{aligned} VPI(S_1 \mid A_0 = move) &= \sum_{s_1} P(s_1 \mid A_0 = move) MEU(\{s_1\}) - MEU(\{\}) \\ &= 0.8(EU(A_1 = stay \mid S_1 = goal)) + 0.2(EU(A_1 = move \mid S_1 = init)) - 1.6 \\ &= 0.8(2) + 0.2(0.8) - 1.6 = 0.16 \end{aligned}$$

Q6. [8 pts] MDPs/RL

(a) [5 pts] **Multiple Choice.** Select the single best answer for each question. We are given an MDP (S, A, T, γ, R), where R is only a function of the current state s. We are also given an arbitrary policy π.

i) If $f(s) = R(s) + \sum_{s'} \gamma T(s, \pi(s), s') f(s')$, then f computes

◯ V^* ◯ Q^* ◯ π^* ⊗ V^π ◯ Q^π ◯ None of these

ii) If $g(s) = \max_a \sum_{s'} T(s, a, s')[R(s) + \gamma \max_{a'} Q^*(s', a')]$, then g computes

⊗ V^* ◯ Q^* ◯ π^* ◯ V^π ◯ Q^π ◯ None of these

iii) If $h(s, a) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma h(s', a)]$, then h computes

◯ V^* ◯ Q^* ◯ π^* ◯ V^π ◯ Q^π ⊗ None of these

iv) Which of the following iterative MDP-solving techniques typically converges in the fewest number of iterations?

◯ Asynchronous Value Iteration ◯ Batch Value Iteration ⊗ Policy Iteration

v) Which of the following reinforcement learning techniques sometimes diverges?

◯ Exact Q-Learning ⊗ Q-Learning with linear function approximations
◯ Exact TD-Learning

(b) [3 pts] Consider policy evaluation in a setting where the reward R is a function of s, a, s', instead of just s. Suppose we have n states, s_1 through s_n. Then for any s, we have the following policy evaluation equation:

$$V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')].$$

Now, suppose the policy $\pi(s)$ that we are evaluating behaves as follows. At each timestep, it picks one out of m different "local" policies $\pi_1(s), \pi_2(s), ..., \pi_m(s)$ with corresponding probabilities $p_1, p_2, ..., p_m$ of being picked. (Note that $p_1 + p_2 + ... + p_m = 1$.) For this timestep, it acts according to the chosen policy. Write down the policy evaluation equation for $V^\pi(s)$ in terms of the local policies $\pi_1(s), \pi_2(s), ..., \pi_m(s)$.

$$V^\pi(s) = \sum_{i=1}^{m} p_i \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V^\pi(s')]$$

Q7. [18 pts] Machine Learning

(a) Overfitting: For each case, select the scenario that is more likely to **overfit** to training data.

(i) [2 pts] Decision Tree

○ Decision Tree with height 10 ● Decision Tree with height 100

(ii) [2 pts] Classifying CS188 students against Non-CS188 student

● Using student id as a feature ○ Using GPA as a feature

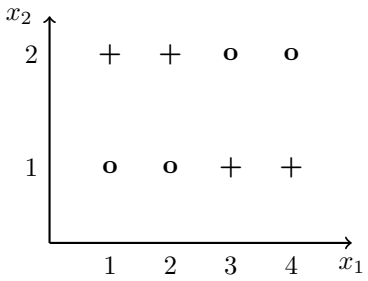
(iii) [2 pts] Statistical Learning

● Laplace smoothing with $k = 2$ ○ Laplace smoothing with $k = 5$

(iv) [2 pts] Linear Regression: Using loss function

● $\sum_i^N (y_i - w^T x_i)^2$ ○ $\sum_i^N (y_i - w^T x_i)^2 + \lambda \sum_i^d w_i^2$

(b) Suppose we have a following training data with class 1, 0 with x_1,x_2 features.



For each true/false question, circle the correct answer. Missing choices and wrong choices with no explanation are worth zero.

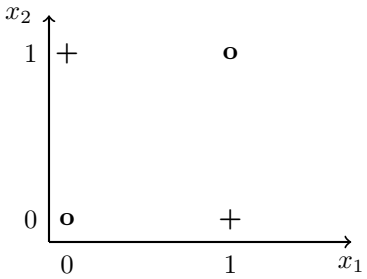
(i) [1 pt] [*true* or *false*] We can use one perceptron to perfectly classify this training data.

(ii) [1 pt] [*true* or *false*] We can use a decision tree of depth 2 to perfectly classify this training data.

(iii) [1 pt] [*true* or *false*] According to information gain, x_1 is a better feature to split than x_2 in decision tree.

(c) Multilayer Perceptron

As you can see from the diagram below, we can use x_1 XOR x_2 to perfectly classify the four data points. We aim to construct a neural net to represent this function.



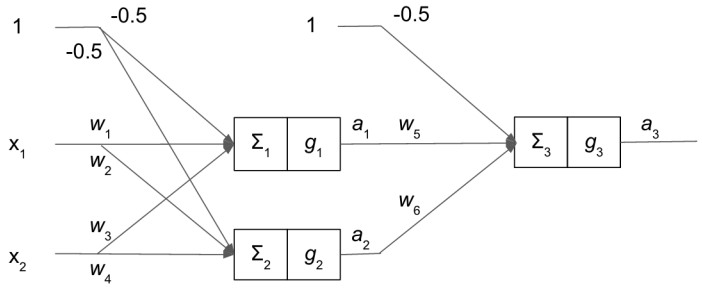
x_1	x_2	x_1 XOR x_2
0	0	o
1	0	+
0	1	+
1	1	o

(i) [1 pt] First, express x_1 XOR x_2 in conjunctive normal form (CNF) with x_1 and x_2 as symbols.

$(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

Another way to express x_1 XOR x_2 is in DNF as $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$. We will use this logical expression to represent our neural net.

We create the following neural net representation:



with Σ_1, Σ_2, and Σ_3 defined as the dot product of the input values and the weights (e.g., $\Sigma_1 = 1 \cdot -0.5 + x_1 w_1 + x_2 w_3$),

and g_1, g_2, and g_3 defined by a threshold function:

$$g(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}.$$

x_1, x_2 are Boolean (0/1) inputs. The weights for the bias inputs for g_1 and g_2 are already defined as -0.5.

(ii) [6 pts] Using this information, find values for weights $w_1, w_2, w_3, w_4, w_5, w_6$ so that:

- the output of g_1, a_1, represents $x_1 \wedge \neg x_2$,
- the output of g_2, a_2, represents $\neg x_1 \wedge x_2$, and
- the output of g_3, a_3, represents $a_1 \vee a_2 = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2) = x_1$ XOR x_2.

[*Hint*: it suffices to consider weights in the set $\{+1, -1\}$.]

$w_1 =$ 1	$w_3 =$ -1	$w_5 =$ 1
$w_2 =$ -1	$w_4 =$ 1	$w_6 =$ 1

THIS PAGE IS INTENTIONALLY LEFT BLANK